Lightweight Cryptography on Ultra-Constrained RFID Devices

Inauguraldissertation
zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften
der Universität Mannheim

vorgelegt von

Dipl.-Inf. Matthias Alexander Hamann
aus Zweibrücken

Mannheim, 2018

Dekan: Dr. Bernd Lübcke, Universität Mannheim
Referent: Prof. Dr. Matthias Krause, Universität Mannheim
Korreferent: Prof. Dr. Willi Meier, Fachhochschule Nordwestschweiz

Tag der mündlichen Prüfung: 22. Mai 2018

ABSTRACT

Devices of extremely small computational power like radio-frequency identification (RFID) tags are used in practice to a rapidly growing extent, a trend commonly referred to as *ubiquitous computing*. Despite their severely constrained resources, the security burden which these devices have to carry is often enormous, as their fields of application range from everyday access control to human-implantable chips providing sensitive medical information about a person. Unfortunately, established cryptographic primitives such as AES are way to 'heavy' (e.g., in terms of circuit size or power consumption) to be used in corresponding RFID systems, calling for new solutions and thus initiating the research area of *lightweight cryptography*.

In this thesis, we focus on the currently most restricted form of such devices and will refer to them as *ultra-constrained RFIDs*. To fill this notion with life and in order to create a profound basis for our subsequent cryptographic development, we start this work by providing a comprehensive summary of conditions that should be met by lightweight cryptographic schemes targeting ultra-constrained RFID devices. Building on these insights, we then turn towards the two main topics of this thesis: *lightweight authentication* and *lightweight stream ciphers*. To this end, we first provide a general introduction to the broad field of authentication and study existing (allegedly) lightweight approaches. Drawing on this, with the $(n, k, L)^{\langle 80 \rangle}$-protocol, we suggest our own lightweight authentication scheme and, on the basis of corresponding hardware implementations for FPGAs and ASICs, demonstrate its suitability for ultra-constrained RFIDs. Subsequently, we leave the path of searching for dedicated authentication protocols and turn towards stream cipher design, where we first revisit some prominent classical examples and, in particular, analyze their state initialization algorithms. Following this, we investigate the rather young area of *small-state stream ciphers*, which try to overcome the limit imposed by time-memory-data tradeoff (TMD-TO) attacks on the security of classical stream ciphers. Here, we present some new attacks, but also corresponding design ideas how to counter these. Paving the way for our own small-state stream cipher, we then propose and analyze the Lizard-construction, which combines the explicit use of *packet mode* with a new type of state initialization algorithm. For corresponding keystream generator-based designs of inner state length n, we prove a tight $(2n/3)$-bound on the security against TMD-TO key recovery attacks. Building on these theoretical results, we finally present Lizard, our new lightweight stream cipher for ultra-constrained RFIDs. Its hardware efficiency and security result from combining a Grain-like design with the Lizard-construction. Most notably, besides lower area requirements, the estimated power consumption of Lizard is also about 16 percent below that of Grain v1, making it particularly suitable for *passive* RFID tags, which obtain their energy exclusively through an electromagnetic field radiated by the reading device. The thesis is concluded by an extensive *Future Research Directions* chapter, introducing various new ideas and thus showing that the search for lightweight cryptographic solutions is far from being completed.

ZUSAMMENFASSUNG

Elektronische Geräte mit extrem eingeschränkter Berechnungskraft wie funkbasierte Radio-Frequency-Identification-Tags (RFID-Tags) durchsetzen zunehmend unseren Alltag. Ungeachtet ihrer geringen Ressourcen ist die Sicherheitslast, die solche Geräte tragen müssen, jedoch oftmals enorm. Denn ihr Einsatzbereich erstreckt sich bereits heute von einfachen Zugangskontrollsystemen bis hin zu implantierbaren Chips, deren Manipulation, beispielsweise im Falle eines funkgesteuerten Herzschrittmachers, sogar lebensbedrohliche Folgen haben kann. Da etablierte kryptographische Primitive wie AES jedoch zu 'schwer' (bspw. hinsichtlich Schaltungsgröße oder Energieverbrauch) für solche RFID-Systeme sind, bedarf es hier neuer Lösungen, welche im Bereich der sogenannten *leichtgewichtigen Kryptographie* zusammengefasst werden.

In dieser Dissertationsschrift konzentrieren wir uns auf die gegenwärtig eingeschränkteste Klasse von RFID-Tags, welche wir als *ultra-constrained RFIDs* bezeichnen werden. Um eine solide Basis für unsere anschließende Entwicklungstätigkeit zu legen, beginnen wir die Arbeit mit einer umfassenden Analyse der relevanten Hardwareeigenschaften und identifizieren dabei insbesondere zentrale Kenngrößen, die von entsprechenden kryptographischen Verfahren einzuhalten sind. Darauf aufbauend wenden wir uns dann unseren beiden Hauptthemen, *leichtgewichtige Authentifikation* und *leichtgewichtige Stromchiffren*, zu. Wir beginnen mit einer allgemeinen Einführung in das weite Themenfeld der Authentifikation und untersuchen zudem bereits existierende (angeblich) leichtgewichtige Verfahren. Danach stellen wir mit dem $(n, k, L)^{\langle 80 \rangle}$-Protokoll unseren eigenen Vorschlag vor und zeigen auf Basis entsprechender Hardwareimplementierungen für FPGAs und ASICs, dass dieser für ultra-constrained RFIDs geeignet ist. Anschließend wechseln wir in den Bereich der Stromchiffren und betrachten zu Anfang einige etablierte Verfahren, mit einem besonderen Fokus auf deren Zustandsinitialisierung. Diesen stellen wir dann das junge Forschungsfeld sogenannter *Small-State-Stromchiffren* gegenüber, welche zum Ziel haben, die Anfälligkeit klassischer Verfahren gegenüber Time-Memory-Data-Tradeoff-Angriffen (TMD-TO-Angriffen) zu überwinden. Hier identifizieren wir zwar neue Angriffsmöglichkeiten, können jedoch auch entsprechende Gegenmaßnahmen vorschlagen. In Vorbereitung unserer eigenen Small-State-Stromchiffre führen wir anschließend die Lizard-Konstruktion ein, welche die explizite Verwendung eines *Paketmodus* mit einer neuen Form der Zustandsinitialisierung kombiniert. Für entsprechende Keystream-Generator-basierte Designs mit innerer Zustandslänge n können wir eine scharfe $(2n/3)$-Schranke hinsichtlich der Sicherheit gegenüber TMD-TO-Key-Recovery-Angriffen zeigen. Basierend auf diesem Ergebnis stellen wir dann Lizard vor, unsere neue leichtgewichtige Stromchiffre für ultra-constrained RFIDs. Effizienz und Sicherheit resultieren hier aus der Verbindung eines Grain-artigen Designs mit der Lizard-Konstruktion. Hervorzuheben ist, neben einer geringeren Schaltungsgröße, die gegenüber Grain v1 um 16 Prozent verminderte Leistungsaufnahme, welche Lizard besonders attraktiv für *passive* RFID-Tags macht, die ihre Energie ausschließlich per elektromagnetischer Induktion über das entsprechende Lesegerät beziehen. Abgeschlossen wird diese Dissertationsschrift durch ein umfangreiches Kapitel über mögliche zukünftige Forschungsfelder. Die darin präsentierten neuen Ansätze und Ideen zeigen, dass die Suche nach leichtgewichtigen kryptographischen Lösungen noch lange nicht abgeschlossen ist.

Acknowledgements

First and foremost, I would like to thank my parents. Without their constant support, I would have never made it even close to this point. I cannot imagine two kinder, more caring, more loyal people, and my gratitude and love for them is beyond all measure.

I am also deeply indebted to my boss and PhD adviser, Prof. Dr. Matthias Krause. He always had my back in academic and personal matters and guided me safely through the sometimes arduous journey of doing a doctorate. I constantly admire the strong bonds which, throughout the years, he maintained with his habilitation adviser, Prof. Dr. Ingo Wegener, and sincerely hope that we will likewise keep in touch.

Furthermore, particular thanks go to Prof. Dr. Willi Meier, who agreed to be the second assessor of this thesis. I benefited immensely from our research collaboration and, when my father died only few weeks before the finalization of this thesis, I came to know this reputable Swiss professor also as a very kindhearted man.

Two colleagues and fellow PhD students have rendered outstanding services to this work. Christan A. Gorke not only took the strenuous job of proofreading, but also ported our new stream cipher Lizard to PHP and offers a corresponding free online service (see Appendix 8.A). Christian Müller, on the other hand, was my rescuing angel when LaTeX once more drove me nuts. Many thanks to both of you and rest assured that I am going to return the favor.

I also wholeheartedly thank the other members of our research group, Prof. Dr. Frederik Armknecht, Angela Jäschke, Vasily Mikhalev, and Alexander Moch, for the many great discussions and, not least, all the fun we had together. You know how much I like you all, but, nonetheless, I will never let you win at Atomic Bomberman [Int97]!

Deep thanks go to our two secretaries, Karin Teynor and, our newest group member, Gabi Nusser. Without them, instead of writing these acknowledgements, I would still be trying to figure out how to properly fill in a 'Reisekostenformular'.

Further thanks go to Dr. Dirk Stegemann, who was my predecessor as PhD student and teaching assistant and left a 'well-tilled field', which allowed me to have a smooth transition from student to PhD life.

I also thank Prof. Dr. Felix Freiling for having employed me as a student helper for many years, while his working group was still in Mannheim. It gave me the chance to learn a lot about IT security, meet many interesting people, and also to visit my first conferences, thus providing an early taste of academic life.

Moreover, I am very grateful to Prof. Dr. Peter Fischer and Dr. Michael Ritzert from Heidelberg University (ZITI), who provided us with the necessary technical means and additional valuable information for creating the hardware implementations of the

$(n, k, L)^{\langle 80 \rangle}$-protocol and of LIZARD.

Equally sincere thanks go to the experts from industry who, on condition of anonymity, were willing to share their inside knowledge about capabilities and limits of state-of-the-art low-cost RFID tags with us. As this information is based on real-world products of the respective companies, its significance for Chapter 2 can hardly be overestimated.

Finally, I thank the anonymous reviewers of the papers underlying this PhD thesis. Their efforts in the background are truly appreciated.

Dedication

To my best friends: Mom and Dad.

Contents

List of Figures

List of Tables

Listings

Acronyms

AES Advanced Encryption Standard. iii, v, 2, 10, 14, 15, 21, 46, 47, 55, 67, 110, 290

API application programming interface. 254

ASCII American Standard Code for Information Interchange. 42

ASIC application-specific integrated circuit. iii, v, 5, 10, 14, 19, 37, 55, 63–65, 73, 81, 82, 84–86, 209, 247, 264, 291

BDD binary decision diagram. 6, 161, 227, 243, 244, 261, 262, 274–277, 280–284, 286, 287, 292

BRAM buffer random access memory. 55

CCMP Counter Mode Cipher Block Chaining Message Authentication Code Protocol. 216

CERT computer emergency response team. 34

CKU stream cipher continuous-key-use stream cipher. 135–138, 140–142, 150, 151, 162, 273

CMOS complementary metal-oxide-semiconductor. 21, 22, 247, 248

CNF conjunctive normal form. 276

CPU central processing unit. 124, 276, 283, 284, 286

CTR mode counter mode. 109, 110

DES Data Encryption Standard. 46

DFA differential fault analysis. 246, 276, 280

DH key exchange Diffie-Hellman key exchange. 33

DoS attack denial-of-service attack. 29

ECB mode electronic codebook mode. 85, 148, 269

EEPROM electrically erasable programmable read-only memory. 16, 17, 22–24, 37, 51, 52, 119, 130, 151, 217, 250

EIRP effective isotropic radiated power. 17

EPC Electronic Product Code. 3, 10, 13, 15, 17, 22, 28, 290

FF flip-flop. 11, 15, 17, 20, 22, 69, 85, 86, 119, 148, 248

FPGA field-programmable gate array. iii, v, 5, 10, 31, 55, 63–65, 82, 85, 86, 104, 247, 252, 261, 262, 264, 265, 291

FSM finite-state machine. 69, 102, 103, 111, 112, 232

FSR feedback shift register. 17, 39, 83, 101, 109, 114, 118, 121, 123, 124, 126, 127, 129, 132, 135, 140–144, 148, 153–155, 217–219, 221, 222, 225, 226, 229, 233, 235, 237–239, 243, 248, 251, 252

GE gate equivalent. 15, 16, 21, 47, 81–85, 148, 224, 246–250, 263, 269

GSM Global System for Mobile Communications. 44, 98, 100, 104, 106, 109, 110, 160, 216, 233, 243, 262

HDL hardware description language. 12, 19

HTTPS Hypertext Transfer Protocol Secure. 6, 215, 216, 252, 292

IC integrated circuit. 16, 23, 38

ID identifier. 38, 40

IETF Internet Engineering Task Force. 34

IKE Internet Key Exchange. 33–35

IP address Internet Protocol address. 33

IV initialization vector. 5, 6, 71, 98–101, 104–107, 109–114, 117, 118, 121–124, 126, 128–131, 134–137, 139–141, 143–147, 149–151, 153–155, 158–166, 169–173, 179, 181, 182, 189, 209, 215–218, 220, 223, 224, 228–232, 234, 235, 239–246, 248–250, 252, 254, 255, 261–273, 287, 291, 292

KSG keystream generator. iii, v, 5, 60, 65, 71, 99, 102–104, 106, 107, 109, 111–114, 118, 119, 131, 132, 135, 144, 157–163, 165–171, 209, 215, 217, 220–222, 228, 230–232, 239, 242, 243, 248, 251, 271, 273, 274, 276, 277, 279, 282, 283, 286, 287, 291

LAN local area network. 32, 33, 264

LFSR linear feedback shift register. 57, 60, 66, 69, 71, 72, 75, 77, 84, 86, 90, 101–109, 111–114, 122, 125, 127–129, 131, 140, 141, 148, 153–155, 218, 219, 224–226, 235–238, 243, 244, 249, 263

LPN learning parity with noise. 3, 4, 27, 31, 32, 46–54, 57, 61, 82, 85, 86, 290

LSB least significant bit. 72, 122, 125, 126, 128, 129

LUT lookup table. 65, 85, 86

MAC message authentication code. 5, 30, 157, 164, 291

MITM man-in-the-middle. 3, 31, 32, 35, 52, 54, 56, 64, 66, 67, 70, 71, 74, 78, 79, 265, 290

NESSIE New European Schemes for Signatures, Integrity, and Encryption. 2

NFC near-field communication. 43

NFSR nonlinear feedback shift register. 38, 84, 86, 101, 107–109, 114, 122, 123, 125, 126, 128, 129, 131, 140, 141, 148, 149, 153–155, 218, 219, 224–226, 235–237, 243, 244, 249, 268, 273, 287

OBDD ordered binary decision diagram. 71, 277–287

OFB mode output feedback mode. 109, 110, 133–135, 142

OS operating system. 33

PC personal computer. 37, 39, 41, 44, 124, 282

PIN personal identification number. 104

PRNG pseudorandom number generator. 21

PROM programmable read-only memory. 16, 24

PSK pre-shared key. 34–37

RAM random-access memory. 37, 282

RFID radio-frequency identification. iii, v, viii, 3–6, 9–25, 27–32, 36, 37, 44, 45, 47, 49, 51–57, 60, 61, 63, 64, 67, 70, 73, 81–87, 97, 119, 130, 152, 209, 215, 217, 218, 247, 250, 252, 253, 261, 263, 264, 266–268, 274, 275, 287, 290–292

RNG random number generator. 21, 25, 39, 46, 47, 50, 53, 75

RSA Rivest-Shamir-Adleman cryptosystem. 47

S-box substitution-box. 12, 67, 86

SAT Boolean satisfiability problem. 39, 124, 262, 274–276, 279, 280, 282, 287

SPN substitution-permutation network. 67

SSL Secure Sockets Layer. 98, 216

TKIP Temporal Key Integrity Protocol. 98

TLS Transport Layer Security. 98, 216

TMD cost time-memory-data cost. 163, 167, 169, 170

TMD-TO time-memory-data tradeoff. iii, v, 2, 5, 6, 39, 71, 97, 104, 112, 117–121, 124, 128–130, 132, 138, 141–143, 146, 148–151, 157, 158, 161–167, 169, 171–173, 177, 178, 209, 210, 215–217, 220, 222, 228–235, 238–240, 243–245, 252, 261, 262, 264, 267, 269–275, 287, 291, 292

TRNG true random number generator. 21

UID unique identifier. 40, 41, 43, 45, 266

USB Universal Serial Bus. 41

VGA Video Graphics Array. 264

VPN virtual private network. 32–37, 42, 44, 60

WEP Wired Equivalent Privacy. 33, 98

WLAN wireless local area network. 6, 33, 35, 36, 100, 160, 215, 216, 252, 292

WPA Wi-Fi Protected Access. 33, 98

Lightweight, baby!

Ronnie Coleman (famed bodybuilder)

CHAPTER 1

Introduction

ABSTRACT

In this chapter, we first provide a brief introduction to the field of lightweight cryptography in general and explain, which kind of devices are targeted by corresponding designs. Subsequently, we then outline the contents and structure of this thesis. Finally, we also give some brief remarks and hints, which are meant to enhance the overall reading experience.

1.1 General Introduction

For about fifteen years now, *lightweight cryptography* has been among the 'hot topics' in the cryptographic community, generating a vast number of academic publications and suggested schemes. Two separate lines of research have been mainly driving this development and still do so today: *authentication* and *encryption*. We will follow both of these lines in this thesis and, in fact, conclude our work by merging them in the form of a lightweight, stream cipher-based authentication protocol.

Regarding *lightweight encryption*, several competitions have been the catalyst for designing corresponding schemes. After the AES block-cipher contest [Nat16] had ended in 2000/2001 and due to the fact that the NESSIE project [Pre03] between 2000 and 2003 had not produced a standardizable stream cipher, in 2004, the eSTREAM project [ECR08] was started in order to identify new stream ciphers for two application profiles:

> "Profile 1 contains stream ciphers more suitable for software applications with high throughput requirements. Profile 2 stream ciphers are particularly suitable for hardware applications with restricted resources such as limited storage, gate count, or power consumption." [ECR08]

The above specification of Profile 2 can be seen as the birth certificate of contemporary lightweight cryptography, as many of the metrics and limits identified during the following selection process are still in place today. After the eSTREAM contest had finished in 2008 with a portfolio of four (later reduced to three) winners in the Profile 2 category, it was now again up to block ciphers to catch up, as even highly optimized implementations of AES [DR02] still required significantly more hardware resources than, e.g., Grain v1 [HJM06], the most hardware-efficient member of the final eSTREAM portfolio [BBV12]. In 2007 and 2009, respectively, the new lightweight block ciphers PRESENT [BKL$^+$07] and KATAN/KTANTAN [DCDK09] appeared, which were actually able to compete with (and, depending on the type of implementation, even to surpass) Grain v1 in terms of low hardware requirements. After that, for many years it looked as if the race between lightweight block ciphers and lightweight stream ciphers had finally been decided in favor of block ciphers. This notion was mainly based on the fact that, for stream ciphers, so-called *time-memory-data tradeoff (TMD-TO) attacks* like those of Babbage [Bab95] and Biryukov and Shamir [BS00] seemed to imply a natural lower bound regarding the size of the inner state of these ciphers, which attributes for a significant part of their hardware costs. However, in 2015, based on the new paradigm of continuously involving the secret key in the state update, Armknecht and Mikhalev were able to 'beat' the corresponding birthday bound with their new stream cipher Sprout [AM15]. Though Sprout was broken soon after publication via non-generic attacks, it has raised significant interest in the underlying design principle and a number of related ciphers have been suggested since. In this thesis, we will present a different approach for reducing the size of the inner state below the limit formerly induced by the birthday bound in the form

of the LIZARD-construction and its first concrete instantiation, the lightweight stream cipher LIZARD.

Research in the field of *lightweight authentication* was sparked in 2001 by Hopper and Blum in the form of their "Secure Human Identification Protocols" [HB01]. As the title suggests, this scheme, which is known under the name *HB protocol* today, was actually targeting humans as the 'endpoint' in an authentication process. More precisely, the HB protocol was designed to be so simple in terms of operations, that these could even be performed by human beings. From a theoretical point of view, the approach of Hopper and Blum was appealing as well, because they were able to prove the security of their protocol against *passive adversaries* (i.e., against pure eavesdropping) based on the assumed hardness of the well-known *learning parity with noise (LPN) problem*. In 2005, Juels and Weis then transferred the scheme to the context of *radio-frequency identification devices (RFIDs)*, by adding security against (certain) *active attacks* and targeting so-called *Electronic Product Codes (EPCs)* with their new HB^+ protocol [JW05].

While, nowadays, RFID technology is commonly associated with tiny computational devices, it actually has its origin in the identification of huge objects. More specifically, it was introduced in World War II to perform *friend-or-foe recognition* through equipping airplanes with radio transmitters that were activated upon receipt of a signal sent by ground radar stations [Rob05]. Measured by production volumes, today, RFID technology rather serves civil purposes and authentication solutions are probably the most common field of application for *RFID tags*. These range from obvious representatives which many of us use every day, such as contactless smart cards for entrance control, to more hidden ones, such as RFID tags incorporated or attached to goods in order to track them or for protecting against product piracy. And the application areas of RFID-based authentication are constantly broadening. Human-implantable tags already exist, imposing new technological challenges (such as a low power and/or energy consumption) on lightweight authentication solutions, whose striving for minimality is currently still mainly motivated by the cost pressure for the production of corresponding devices.

In this spirit, with their HB^+ protocol, Juels and Weis targeted the aforementioned EPCs, which represent the most restricted type of RFID devices and are meant to replace classical barcodes. Besides the new authentication scheme, another major contribution of the respective paper [JW05] was a collection of hardware limits that such devices in the price range of \$0.05–\$0.10 actually impose on cryptographic designs. In this thesis, we will target hardware of similar capabilities, but rather refer to it as *ultra-constrained RFIDs* in order to express the now broadened application range, apart from simply replacing classical barcodes.

While one might now expect that, based on the profound basis laid by Juels and Weis in [JW05], the development of dedicated lightweight authentication schemes would have taken a similarly positive path as the field of lightweight block and stream ciphers, this is unfortunately not the case. After the HB^+ protocol was broken by active man-in-the-middle (MITM) attacks [GRS05] already in 2005, a multitude of related schemes started

to appear, which, however, seemed to increasingly submerge in the 'theoretical beauty' of the approach, eventually loosing sight of the actual hardware limits (probably also due to the lack of an eSTREAM-like competition with its concrete evaluations). As a consequence, in [AHM14], we were able to show that currently there does not seem to be a single unbroken HB-type protocol feasible for ultra-constrained RFIDs. In this thesis, employing another paradigm than the assumed hardness of the LPN problem, we will present an alternative approach for designing dedicated lightweight authentication protocols and, based on an actual hardware implementation, demonstrate its suitability for respective devices. Finally, we will also forge a bridge between lightweight authentication and state-of-the-art lightweight stream ciphers by demonstrating how our new design Lizard can be used to realize hardware-efficient, privacy-preserving authentication on ultra-constrained RFIDs.

1.2 Structure of this Thesis

In *Chapter 2*, we provide a comprehensive summary of conditions that should be met by lightweight cryptographic schemes if deployed in low-cost RFID systems. Some of these conditions have been collected from open literature, but most of them are the result of various discussions with experts from industry. Although these experts were working for different companies and were aiming for RFID-based cryptographic solutions in different areas, all of them shared more or less the same view on what *lightweight* means in the context of ultra-constrained devices and when a scheme can be considered to be relevant for real-word applications. The metrics and conditions compiled here will also serve as the basis for the design decisions and hardware evaluations made in the subsequent chapters.

In *Chapter 3*, we give an introduction to authentication in general and its lightweight forms in particular. Based on examples of failed real-world authentication solutions, we develop a number of rules that should be considered when deploying RFID-based authentication schemes in the field. Furthermore, we describe and evaluate (based on the hardware limitations as introduced in Chapter 2) the three most common approaches for lightweight authentication: (block) cipher-based protocols, LPN-based protocols, and protocols based on random selection of secret linear functions. Referring to our paper *Lightweight Authentication Protocols on Ultra-Constrained RFIDs – Myths and Facts* [AHM14], we conclude that none of the currently unbroken LPN-based protocols is suitable for ultra-constrained RFIDs. For authentication protocols based on random selection of secret linear functions, we identify a comparatively large key length and the use of involved operations as the major challenges for creating a hardware-efficient implementation.

In *Chapter 4*, we introduce the new $(n, k, L)^{\langle 80 \rangle}$-protocol, a variant of linear authentication protocols which overcomes the above problems, and analyze its security against all

currently known, relevant passive and active attacks. Moreover, we present an implementation of our protocol for field-programmable gate arrays (FPGAs) and application-specific integrated circuits (ASICs) based on the hardware description language Verilog and discuss its efficiency w.r.t. the cost metrics described in Chapter 2. The respective numbers show that the $(n, k, L)^{\langle 80 \rangle}$-protocol is a viable alternative to existing solutions and is, for example, suited for the implementation on ultra-constrained RFID tags.

In *Chapter 5*, we leave the path of searching for dedicated authentication protocols and, instead, lay the foundation for treating a fundamental question which arose to us while designing the $(n, k, L)^{\langle 80 \rangle}$-protocol: 'Why use a bitstream generator only to produce the specifications of the secret functions, but not for generating the authentication token right away?' To this end, as a first step, we revisit some prominent examples of classical stream ciphers in this chapter and, in particular, analyze their state initialization algorithms. Later, in Chapter 7, the respective insights will then serve as a basis for our new stream cipher design principle named Lizard-construction, which allows for creating provably secure small-state constructions.

In *Chapter 6*, we introduce the corresponding, rather young research area of so-called *small-state stream ciphers*, which try to overcome the limit imposed by TMD-TO attacks on the security of classical stream ciphers. In the course of our studies, existing designs and known analysis of small-state stream ciphers are revisited and new insights on distinguishers and key recovery are derived based on TMD-TO attacks. A particular result is the transfer of a generic distinguishing attack suggested in 2007 by Englund et al. [EHJ07] to this new class of lightweight ciphers. Our analysis shows that the initial hope of achieving full security against TMD-TO attacks by continuously using the secret key has failed. In particular, we provide generic distinguishers for the small-state stream ciphers Plantlet [MAM17] and Fruit [GHX16] with complexity significantly lower than that of exhaustive key search. However, by studying the assumptions underlying the applicability of these attacks, we are able to come up with a new design idea for small-state stream ciphers, which might allow to finally achieve full security against TMD-TO attacks by continuously using not only the key but also the initialization vector (IV) during keystream generation. Another contribution of this chapter is the first key recovery attack against Fruit v1. We show that there are at least 2^{64} weak keys, each of which does not provide 80-bit security as promised by the designers. As a consequence of our attack, the designers of Fruit have updated their scheme to the new version Fruit v2.

In *Chapter 7*, we propose and analyze the Lizard-construction, a new way to build stream ciphers. For corresponding keystream generator-based designs of inner state length n, we prove a tight $(2n/3)$-bound on the security against TMD-TO key recovery attacks, while the security against TMD-TO distinguishing attacks remains at the birthday-bound level $n/2$. The lower bound of the $(2n/3)$-result refers to a random oracle model which allows to derive formal security statements w.r.t. generic TMD-TO attacks. While similar frameworks have already been widely used for analyzing the security of block cipher, MAC, and hash function constructions, to the best of our knowledge this is the first

time that such a model is considered in a stream cipher context. The security analysis presented in this chapter is also of immediate practical relevance as, with Lizard, a first instantiation of our new design principle (which we hence named Lizard-construction) was introduced at FSE 2017 [HKM17b].

In *Chapter 8*, we then present Lizard, our new lightweight (small-state) stream cipher for power-constrained devices like passive RFID tags. Its hardware efficiency results from combining a Grain-like design with the Lizard-construction introduced in Chapter 7. Lizard uses 120-bit keys, 64-bit IVs, and has an inner state length of 121 bits. It is supposed to provide 80-bit security against key recovery attacks. Lizard allows to generate up to 2^{18} keystream bits per key/IV pair, which would be sufficient for many existing communication scenarios like Bluetooth, WLAN, or HTTPS.

In *Chapter 9*, we suggest three, in our opinion, particularly promising projects for future research in the field of lightweight cryptography. First, we connect our two main topics, i.e., *lightweight authentication* and *lightweight stream ciphers*, by describing how Lizard can be used to realize hardware-efficient, privacy-preserving authentication on ultra-constrained RFIDs. Second, we take our idea of continuously using the IV as part of a stream cipher's state update one step further. While in Chapter 6, this new approach 'only' serves to protect ciphers which continuously use the secret key against TMD-TO *distinguishing* attacks, we here show that, under certain conditions, it can even be used independently to thwart classical TMD-TO *inner state recovery* attacks. Third, we explain why we believe that the field of BDD-based cryptanalysis has been neglected for too long and provide several new ideas how the efficiency of such attacks could be improved in the future.

Chapter 10 concludes this thesis by summarizing our results.

1.3 Reading Guide

In the following, we give some brief remarks and hints, which are meant to enhance the reading experience for this thesis. First of all, we would like to 'warn' the reader that between the chapters, there will be some small redundancies. This redundancy, however, is intended and will be confined to the introduction of certain terms and concepts which might not be familiar to all readers. Many readers will probably be interested in selected topics only, and we want to save these readers from having to first consult several other chapters in order to understand something which, otherwise, would have become clear with just a few words of additional explanation. Nevertheless, we try to limit redundant explanations to the necessary minimum and are confident that their degree is at a level still tolerable to the more experienced reader.

Also mainly aimed at the aforementioned 'selective readers' are the brief abstracts which we give on the title page of each chapter. Note that within these chapter abstracts, we provide references only for explicit citations. This is meant to allow for quickly reading

the respective passages without any further distraction. With respect to academic rigor, we consider this approach permissible, as the corresponding references will all be given subsequently in the introductions of the respective chapters.

In addition to the abstract, each chapter's title page features a so-called *declaration of origin*, where the underlying publications of the author (along with the respective co-authors and the conferences/workshops at which the content was presented) are listed. Some publications appear in more than one of these declarations of origin, if the corresponding content was spread over multiple chapters for reasons of improved presentation as part of this thesis.

Finally, we would like to point the reader to the list of acronyms in the preface of this thesis. For reasons of better readability, at very few places of the text, an acronym that is used for the first time will not be accompanied by the respective long form, which, instead, is then given at some later point. While we tried to make sure that, where this happens, the long form is actually of no relevance for understanding the corresponding text passage, a reader of the PDF version of this thesis always has the possibility to click on an acronym, which will then take him to the respective entry of the aforementioned list.

So David triumphed over the Philistine with a sling and a stone.

1 Samuel 17:50

CHAPTER 2

Ultra-Constrained RFID Devices

ABSTRACT

While most lightweight cryptographic schemes have been well analyzed with respect to their *security*, especially for authentication protocols often only little (or even nothing) is known with respect to their *suitability* for ultra-constrained RFID devices in the production cost range of \$0.05 to \$0.10. Probably, this is mainly due to the fact that open literature rarely provides information on what conditions need to be met by a scheme in practice, hindering a sound development and analysis by the academic community.

In this chapter, we provide a comprehensive summary of conditions that should be met by lightweight cryptographic schemes if deployed in low-cost RFID systems. Some of these conditions have been collected from open literature, but most of them are the result of various discussions with experts from industry. Although these experts were working for different companies and were aiming for RFID-based cryptographic solutions in different areas, all of them shared more or less the same view on what *lightweight* means in the context of ultra-constrained devices and when a scheme can be considered to be relevant for real-word applications. The metrics and conditions compiled here will also serve as the basis for the design decisions and hardware evaluations made in the subsequent chapters.

Declaration of Origin: This chapter is partly based on the paper *Lightweight Authentication Protocols on Ultra-Constrained RFIDs – Myths and Facts* [AHM14], written together with Frederik Armknecht and Vasily Mikhalev and presented at *RFIDSec 2014*.

2.1 Introduction

Embedded systems are increasingly permeating our information society, being more and more employed also in security- and safety-critical applications. However, corresponding devices often suffer from very restricted hardware characteristics, which render existing security solutions infeasible. This has created a need for so-called *lightweight* security schemes that on the one hand, are appropriate for the targeted device characteristics but on the other hand, still provide a sufficient level of security for the intended application.

One of the major use cases for such pervasive devices are authentication solutions, e.g., access control for buildings or cars, electronic passports, or even human-implantable chips providing sensitive medical information about a person. Consequently, the search for lightweight authentication protocols became an important topic in IT security during the last years with high relevance for academia and industry, producing a huge number of different schemes and approaches (cf. Chapter 3). However, while an abundant number of works investigate the security of these schemes, often only little (or even nothing) is known with respect to their applicability for real-world scenarios. Probably, this is mainly due to the fact that there is no common agreement on what conditions need to be meet by a scheme to be considered as *lightweight*. The lack of a common understanding made the development of new schemes and the analysis of existing ones with respect to practical demands extremely difficult, if not even impossible.

Without doubt, universally applicable criteria cannot be accepted as these heavily depend on the concrete use case and deployed technology. However, things are different if focusing on a concrete scenario and having a concrete technology in mind. More precisely, in this thesis we target cryptographic protocols between RFID readers and *ultra-constrained* tags in the cost range of \$0.05 to \$0.10 like EPCs. The reasons for this specific choice are twofold: First, RFID tags which can be produced at costs of \$0.10 or cheaper have been a common motivation for existing work (see, e.g., [FDW04], [JW05], [CR08], [MME+11]). Second, if one allows for only few additional costs, standard cryptographic primitives like the Advanced Encryption Standard (AES) [DR02] become in fact feasible, thus practically removing the need for alternative solutions altogether (see, e.g., [FDW04] and Subsection 2.3.3).

Due to their prevalence in the field of lightweight cryptography, we mainly focus on ASICs in this work. They are (ex ante) tailored to a very specific need and subsequently produced in large quantities, allowing for low unit cost and making them perfectly suitable for pervasive devices like ultra-constrained RFID tags. Their counterparts are FPGAs, which are integrated circuits designed to be configured by a customer or a designer after manufacturing. Consequently, FPGAs are usually more expensive and will be discussed at only few places in this thesis. In particular, if we do not explicitly indicate a different hardware platform, we are always referring to ASICs.

In the following, we will specify and argue several conditions that need to be met by cryptographic schemes to be suitable for ultra-constrained RFID devices. These

conditions have been derived partly from open literature but most importantly from various discussions with experts from industry. Although these experts were working for different companies and were aiming for RFID-based cryptographic solutions in different areas, all of them shared more or less the same view on what *lightweight* means in the context of ultra-constrained devices and when a scheme can be considered to be relevant for real-word applications. As these conditions mostly result from long lasting experience in hardware production and have not (or only partly) been comprehensively described and summarized in open literature, we think that this information will be very helpful for assessing the suitability of existing cryptographic schemes and for providing guidance in the development of new ones.

Before we go into the corresponding details, note that Juels additionally points out that while "it is tempting to dismiss this computational poverty a temporary state of affairs, in the hope that Moore's Law will soon render inexpensive tags more computationally powerful [...] pricing pressure is a strong countervailing force" [Jue06]. And indeed, it seems that most of the limits described in, e.g., [JW05] and [Jue06], still apply today as our numerous discussions with experts from industry have revealed. Hence, the numbers presented in the following paragraphs can be expected to remain valid also in the medium term.

2.2 Excursus: Implementation Options for Block Ciphers and Stream Ciphers

Naturally, when it comes to using cryptographic schemes on resource-constrained devices, it is not only important to choose a suitable, efficient algorithm, but also, how this algorithm is eventually implemented. The three main concepts distinguished here are *parallel*, *round-based*, and *serialized* implementations.

Round-based refers to block ciphers which are built as cascade of identical (or nearly identical) rounds. In implementations of such ciphers, the hardware components for the computation of one round can usually be reused for the other rounds, which, e.g., helps to save on chip area (see Subsection 2.3.3). As a consequence, for round-based implementations, the number of clock cycles for encrypting a single block of data is usually equal to the number of rounds of the respective block cipher algorithm. If faster encryption is required, several (or even all) rounds can be computed within a single clock cycle at the cost of increased chip area and power consumption (see Subsection 2.3.4). This type of implementation is usually called *parallel*, which, however, can be misleading, as the rounds of classical block ciphers still have to be computed sequentially. Therefore, the alternative term *unrolled* has also become very common to describe such implementations.[1] Finally,

[1] Parallel/unrolled implementations can be further distinguished depending on whether the input of the encryption module is supposed to be processed within a single clock cycle or, alternatively, through several stages that are separated via flip-flops, which serve as a kind of temporary memory between

there is also the possibility to save further chip area (and power) compared to round-based implementations by splitting up the individual rounds as well, which allows to reuse round-internal components such as substitution-boxes (S-boxes). This is called a *serialized* implementation.

For stream ciphers (see Chapters 5 to 8), which, in their standard implementation, usually produce one keystream bit per clock cycle, the implementation option *parallel* commonly refers to the generation of more than one keystream bit per clock cycle, e.g., via multiple parallel feedback and output functions. In Chapter 8, this will be explained in detail at the example of our new lightweight stream cipher Lizard. The block cipher implementation option *round-based* obviously does not make much sense for common stream ciphers and the term *serialized* can have different meanings. For example, in our implementation of Lizard, we serialize parts of the state initialization, i.e., certain steps in the cipher's algorithmic description of this phase are each spread over several clock cycles (see Subsections 8.5.1 and 8.5.2). This allows us to significantly decrease the chip area of our implementation in exchange for a slightly increased (but still moderate) latency (see Subsection 2.3.2).

Finally, also note that the settings of the compiler, which transforms an algorithm's description given in a so-called *hardware description language (HDL)* (such as VHDL or Verilog) into an actual circuit (consisting of so-called *standard cells*; i.e., components such as AND/OR/XOR, which realize simple Boolean functions), can have a major impact on some of the metrics discussed below. For example, in order to be suitable for being operated at very high clock rates, a circuit needs to have a low delay (see Subsection 2.3.7), which can be achieved by reducing the logical depth of its function implementations. This, however, often comes at the cost of more expensive components in terms of area and power. Hence, in order to achieve the best possible comparison of schemes, they should always be implemented with the same standard cell libraries and compiler settings (as we do, e.g., in Subsection 8.5.1 when comparing our new lightweight stream cipher Lizard to Grain v1).

2.3 Hardware Characteristics and Limits

In this section, we provide a comprehensive summary of hardware characteristics and corresponding conditions that should be met by lightweight cryptographic schemes if deployed on ultra-constrained RFID devices. As pointed out in the chapter's introduction, part of this information has been collected from open literature, but most of it is the result of various discussions with experts from industry. Unfortunately, all of our sources requested confidentiality due to the fact that this data might allow to draw inferences

the corresponding clock cycles. The latter variant realizes a *pipeline*, which has the advantage that the *critical path* (see Subsection 2.3.7) of the respective circuit is shorter, hence allowing a higher maximum clock rate.

Table 2.1: RFID application fields, transfer rates, and range by waveband. (cf. [SCU11])

Waveband	Utilization	Bandwidth	Distance
Low Frequency (LF), 30–300 kHz	Animal Identification	< 10 kbit/s	0.1–0.5 m
Medium Frequency (MF), 0.3–3 MHz	Contactless Payment	< 50 kbit/s	0.5–0.8 m
High Frequency (HF), 3–30 MHz	Access Control	< 100 kbit/s	0.05–3 m
Ultra HF (UHF), 0.3–3 GHz	Range Counting	< 200 kbit/s	1–5 m
Super HF (SHF), 3–30 GHz	Vehicle Identification	< 200 kbit/s	ca. 10 m

about the current products of their respective companies. However, as far as possible, we tried to support and match it with publicly available information and, moreover, most parts of this section are also contained in our publication [AHM14], which has been peer-reviewed and presented at RFIDSec 2014. An interesting side effect of this matching process of public and confidential data was the realization that many of the 'older' limits for low-cost RFIDs, which have been published as early as 2005 (see, e.g., [JW05]), are still in place in today's industry.

2.3.1 Operating Frequency and Transmission Bandwidth

The waveband of RFID tags and, closely related, their maximum available transfer rate is determined by several factors. One of the most important is the targeted reading distance implied by, inter alia, a tag's purpose. For example, while it may be desirable to read a complete pallet of products with attached EPC tags over a long distance, access control should rather be confined to a close environment, e.g., someone putting his access card right on top of a corresponding reader. Table 2.1 is based on the data provided in [SCU11]. It shows that corresponding cryptographic solutions are limited to exchanging data at a rate of at most 200 kbit/s (100 kbit/s in the very common HF band) between a tag and a reader.

For authentication, the common notion is that the whole process should not take more than 150 ms (see Subsection 2.3.2). Given the above transfer rates, this implies that about 30 000 bits can be considered as the upper bound for an authentication protocol's communication complexity. Furthermore, this number is even lowered by the fact that, within those 150 ms, the respective data must be processed by the tag and that not only non-volatile memory but also volatile memory (e.g., Juels and Weis assume 32–128 bits in [JW05]) is a scarce resource, which heavily limits buffering incoming data. Also note that, depending on the number of labels that need to be read per second, the number of available bits per authentication instance may actually be much lower. For example, in [CR08] it is stated that "in accordance with C1G2, a maximum tag to reader data transmission rate of 640 kbps and a reader to tag data transmission rate of 126 kbps based on equi-probable binary ones and zeros in the transmission can be calculated" and

that "performance criteria of an RFID system demand a minimum label reading speed in excess of 200 labels per second".

For the design of lightweight ciphers, the limited transfer rates of RFID tags are highly relevant (though rarely[2] discussed), too, as they determine the maximum possible clock rates which the corresponding implementations have to be capable of (see Subsection 2.3.6 for further details).

2.3.2 Timing/Latency

Perhaps surprisingly, we were told the aforementioned upper timing bound of 150 ms for authentication by various hardware producers on the basis of rather different reasons. These ranged from human interaction in the presence of additional tag functions to regulations by the automotive industry w.r.t. timing restrictions for component interaction. From a technical point of view, UHF regulations would impose a maximum of 400 ms due to channel hopping but "user performance requirements establish a time limitation on a label operation since at least 100–300 labels must be read per second" [REC05]. For example, Feldhofer et al. designed their AES-based authentication protocol such that each tag has 18 ms time, hence "a maximum of 50 tags can be authenticated per second" [FDW04]. Consequently, the upper bound of 150 ms told to us by several industrial sources is probably already very generous and, depending on the use case, might actually be much lower by factors of 10–50. Keep in mind that this would directly translate to vastly reduced upper bounds for communication (e.g., a maximum of 600 bits instead of 30 000 bits per authentication) or available clock cycles (e.g., only 300 instead of 15 000; see Subsection 2.3.6).

For implementations of encryption algorithms, one has to distinguish between block ciphers and stream ciphers when it comes to latency. In the field of block ciphers, the term latency usually refers to the number of clock cycles required to encrypt a single plaintext block. Obviously, this number does not only depend on the algorithm itself, but also on the way it is implemented (see Section 2.2). For stream ciphers (see Chapters 5 to 8), which, in their standard implementation, usually produce one keystream bit per clock cycle, the term latency is traditionally used to describe the time (measured in clock cycles) for the so-called *state initialization*, i.e., until the first keystream bit is available. So in the case of block ciphers, latency behaves inversely proportional to the encryption speed, whereas for stream ciphers, the influence of latency on the overall time to produce some keystream segment decreases with increasing keystream lengths.

2.3.3 Area (in GE)

As explained, we focus on ASICs in this thesis because of their prevalence in the field of ultra-constrained RFID devices. Due to the fact that for ASIC implementations, area

[2] In fact, during our research, we have not encountered a single work which discusses this connection.

Table 2.2: Area requirements of selected standard cells of the UMCL18G212T3 library (in μm^2 and GE). (cf. [Pos09])

Cell		NOT	NAND	NOR	AND	OR	MUX	XOR	D FF
Area	[μm^2]	6.45	9.68	9.68	12.90	12.90	22.58	25.81	51.61
	[GE]	0.67	1.00	1.00	1.33	1.33	2.33	2.67	5.33

requirements in μm^2 strongly depend on the used standard cell library (and, thus, the fabrication technology), it has become common practice to resort to a more general metric called gate equivalents (GE) instead. In short, one GE is equivalent to the area of a two-input drive-strength-one NAND gate. This at least allows for a rough comparison of area requirements derived using different technologies. To give the reader an impression how typical standard cells contribute to the area of a circuit, in Table 2.2 we provide a corresponding overview for the UMCL18G212T3 library, which underlies, e.g., our hardware implementations of the $(n, k, L)^{\langle 80 \rangle}$-protocol (see Chapter 4) and of Lizard (see Chapter 8). In particular, note that a D flip-flop (FF), the cheapest type of flip-flop, already requires four times the area of an AND gate. This comparatively high area demand of flip-flops is one of the motivations for our study of *small-state ciphers*, starting in Chapter 6 and leading to our new lightweight stream cipher Lizard in Chapter 8.

In 2005, Juels and Weis [JW05] stated the "Security Gate Count Budget" of an EPC tag to be "200–2000 gates" and, even today, this upper bound of 2000 GE is still commonly considered to be the magic number for lightweight cryptographic implementations. From an academic perspective, this conclusion can be drawn based on the fact that many recent works (see, e.g., [WZ11], [PMK+11], [SE12], [MSGAHJ13], [BSS+15], [KJAB17]) still assume 2000 GE to be the upper bound w.r.t. tag area. Some other works assume between 200 and 4000 GE [CR08, REC05] but are sometimes not clear about whether they are actually referring to the total area of a low-cost RFID tag or just the amount of gate equivalents available for security purposes. Apart from academic publications, all experts from industry we spoke to confirmed that 2000 GE today still constitutes a plausible security gate count budged for low-cost RFIDs. For comparison, one of the currently smallest known AES implementations due to Feldhofer et al. [FWR05] requires about 3400 GE, which implies that newly suggested approaches requiring even more area should at least be obliged to justify what additional benefit they bring. This obligation to justify even the need for a single additional gate has straightforward monetary reasons as, according to [CR08], 1000 additional gates of silicon logic increase a tag's price by \$0.01, which amounts to considerable sums given production volumes of hundreds of millions in the case of low-cost RFID tags.

It should also be noted that, in addition to the number and placement of logic gates, other (security-related) components contribute to the chip area of an RFID tag as well.

For example, one way to fix constant bit values (e.g., cryptographic keys) on individual tags is to use *fuses/antifuses* and 'burn' a corresponding selection of them (usually through applying high voltages) before a tag leaves the factory (see Subsection 2.3.10). Depending on the particular technology, fuses/antifuses can require considerable area themselves (e.g., in order to ensure that no other elements of the integrated circuit (IC) are affected by burning them) and, equally important, additionally required components for setting the values (e.g., for providing the high voltages) take chip area as well. Consequently, this technique seems infeasible for ultra-constrained RFIDs when it comes to storing large amounts (i.e., thousands) of constant bits at production time. Please note that we are explicitly not referring to the use of fuses/antifuses as part of *programmable read-only memory devices (PROMs)* here. This is because in lightweight cryptography, schemes which are targeted at fixed-key scenarios (such as some of the stream ciphers in Chapter 6) assume that accessing the respective key bits takes neither time nor inflicts other costs like an increased power consumption. In other words, fixed key bits are considered to be an immediate part of the cipher's circuit. For PROMs, whose contents are accessed by providing corresponding memory addresses (or which output their contents sequentially), this is clearly not the case. In this spirit, we consider PROMs as another type of general non-volatile memory (see Subsection 2.3.9) like *electrically erasable programmable read-only memory devices (EEPROMs)* in this thesis, and not as a fixed-key storage option (see Subsection 2.3.10) in the sense of lightweight cryptographic schemes.

Providing additional security against *side-channel attacks*, which use information obtained from physical characteristics (such as power consumption) of an actual implementation, can also easily double the area requirements of a cryptographic scheme. For example, in [PMK+11] a side-channel resistant, serialized implementation of the lightweight block cipher PRESENT [BKL+07] is suggested, which, depending on the level of resistance, requires between 2282 GE and 3582 GE, as compared to an unprotected serialized implementation for only 1111 GE.[3] However, as pointed out before, we will focus on the immediate algorithmic aspects of lightweight cryptography in this thesis and leave the development of side-channel resistant versions of our schemes as future work.

2.3.4 Power

Closely related to the amount of required hardware logic is an RFID tag's power consumption, which consists of a *static* and a *dynamic* part. In short, *static power consumption* denotes the power which a circuit consumes while being in a stable state, i.e., while no switching (e.g., triggered by a clock signal or external inputs) occurs. Consequently, static power consumption is roughly proportional to the area of a circuit. *Dynamic power consumption* is the result of *switching activity*, i.e., when inputs of standard cells change

[3] Other important hardware metrics such as power consumption are affected by implementing side-channel resistance, as well. For example, w.r.t. power consumption, the relative increase for the respective implementations of PRESENT in [PMK+11] is even larger.

and new outputs need to be 'computed'. In our case, such changes are usually triggered by an external clock signal. Consequently, the higher the clock rate (see Subsection 2.3.6) of an RFID tag is, the higher its dynamic power consumption will be. A common method in hardware design to lower the switching activity independently of the clock rate is to apply so-called *clock gating*. Roughly speaking, clock gating temporarily disables parts of the circuit while they are not needed by disabling their connection to the clock signal. This technique adds some further logic, but can have very positive effects on a scheme's power consumption and is hence also used in our implementation of Lizard (see Subsection 8.5.1 for further details).

As a major part of this thesis is devoted to the search for new *small-state* stream ciphers (see Chapters 6 to 8), let us point out here that flip-flops are not only costly in terms of area (cf. Subsection 2.3.3), but also in terms of power. Their rather high *static* power consumption is related to their comparatively large size, as explained above. But especially in cryptographic contexts, also their *dynamic* power consumption is extensive. For example, when used as a component of a cryptographic feedback shift register, due to general security assumptions, in each clock cycle, such a flip-flop should have a probability of 0.5 of changing its stored value, leading to considerable switching activity and, thus, additional power consumption of a large standard cell.

The reason why power is such a 'precious resource' on low-cost RFID tags lies in the fact that these devices are commonly powered via an electromagnetic field radiated by the reader (i.e., *passively*). As the transmission power of an RFID reader is limited by factors such as regulations (e.g., for the EPC Gen 2 band, to 4 W effective isotropic radiated power (EIRP) in the U.S. and 2 W EIRP in Israel [Rep13]), the more power a tag consumes, the smaller the maximum (legally possible) reading distance becomes. In [JW05], Juels and Weis give a general upper bound of 10 μW and Saarinen and Engels emphasize that power peaks should be below 3 μW to 30 μW [SE12]. Hence, if an algorithm depends on high clock rates to perform its operations within a reasonable time span and uses, in addition, power demanding components like EEPROMs (see Subsection 2.3.9), the power budget of a lightweight RFID tag my easily be exhausted. Another design choice which may heavily influence a tag's power consumption is the technology library used to implement it. For example, in [RPLP08], running the block cipher PRESENT [BKL$^+$07] at 100 kHz is compared for the libraries 0.35μ AMIS (3.3 V), 0.25μ IHP (2.7 V), and 0.18μ UMC (1.8 V), leading to different power consumptions of 11.20 μW, 4.24 μW, and 2.52 μW, respectively. Consequently, as already pointed out in Section 2.2, especially for the comparison of power consumption it is of vital importance to employ an identical set-up w.r.t. tools, technology libraries, and compiler options.

Finally, note that, as Ingrid Verbauwhede points out in [Ver13], power consumption is also closely connected to temperature. More precisely, e.g., implanted devices such as pace makers should not lead to temperature differences larger than 1 °C, motivating her to explore the question "How much crypto in one microJoule or 10 microWatt?" [Ver13], which is in line with the limit for power consumption that we have established above.

2.3.5 Energy

Another important issue emphasized by Ingrid Verbauwhede in the aforementioned talk at Real World Crypo 2013 is that "low power is NOT low energy" [Ver13]. While a circuit may have a rather low power consumption (e.g., due to low area and few switching activity per clock cycle), it may still consume (too) much energy if it needs a lot of clock cycles to 'do the job'. In particular, a parallel or round-based implementation (see Section 2.2) of a block cipher will usually be more energy efficient (w.r.t. the metric *energy per encrypted bit*) than a serialized implementation, which has a lower power consumption but requires more clock cycles to encrypt one block of data. For stream ciphers, which, in their standard implementation, usually generate one keystream bit per clock cycle, energy consumption is important w.r.t. the length of the generated keystreams. This is due to the fact that stream ciphers usually need a *state initialization* phase before the first keystream bit can be output. In consequence, if only very short keystreams are produced (e.g., if the cipher is employed in a challenge-response authentication scheme), it may actually be preferable from an energy perspective to use a block cipher or some dedicated authentication protocol instead. In Subsection 8.5.1, we will discuss this topic in further detail as part of the hardware analysis of our new stream cipher Lizard.

Note, however, that despite the undisputed importance of the factor *energy* in general-purpose cryptographic hardware design, we will mainly focus on *power* in thesis. This is due to the fact that energy is only relevant in the context of so-called *active* RFID tags, which are equipped with a battery. The term *active* here also means that, due to their autonomous power supply, such devices can perform tasks without an RFID reader being present. For example, a group of nodes can interact among each other in order to execute some kind of consensus protocol. Due to the additional battery and their advanced capabilities, such devices are however far outside the scope of production costs between \$0.05 to \$0.10 as targeted in this thesis. On *passive* RFID tags, on the other hand, energy consumption is obviously irrelevant as long as the aforementioned limits on power consumption are satisfied. Such tags are virtually 'dead' while no reader is present.[4]

2.3.6 Clock Rate

Ceteris paribus, the higher the clock rate of a tag is, the more clock cycles are available for the cryptographic processes. But as pointed out in the previous paragraphs, factors like the power budget of a passively powered RFID tag impose an upper bound on its clock frequency. Many works (e.g., [Pos09], [FDW04], [PLHCETR09]) consider 100 kHz to be the prevalent clock rate feasible on ultra-constrained RFID tags. This value is in

[4] In fact, this is the very reason why we named our new lightweight stream cipher for power-constrained devices (see Chapter 8) Lizard. Just like a lizard depends on sun's rays to get active, the passive devices targeted by our low-power cipher need the electromagnetic field radiated by an RFID reader to perform their computations and transmit data (via modulation of the radiation reflected by the tag's antenna).

line with the information we obtained from the RFID hardware producers who demanded confidentiality.

Assuming an upper bound of 150 ms for executing a full authentication instance, a clock rate of 100 kHz immediately implies an upper bound of 15 000 clock cycles on the tag's side to authenticate successfully. In Section 3.4, we will point out that many allegedly lightweight protocols exceed this upper bound of 15 000 clock cycles even by magnitudes and, hence, would be clearly infeasible also for higher clock rates like 1 MHz.

As we have seen above, the transmission bandwidth of RFID tags is also rather limited, which has important implications for lightweight ciphers targeting such devices, because the maximum transfer rates also determine the maximum possible clock rates which the corresponding implementations have to be capable of. This holds especially for stream ciphers, which, in their standard (i.e., non-parallelized) implementation, usually produce one keystream bit per clock cycle. Given that at most 200 kbit/s have to be encrypted, we (and the experts that we contacted) do not see a need for stream cipher implementations on RFID tags that target significantly higher clock rates than 200 kHz.[5] In the eSTREAM competition [ECR08] (see Chapter 5 for further details), however, the ASIC implementations of the stream cipher candidates in the hardware category were actually also evaluated (e.g., in [GB08]) with respect to a metric called *delay* (see Subsection 2.3.7), which determines the maximum possible clock rate at which they can operate. At this maximum possible clock rate, other metrics such as power consumption were then determined. While this may have been relevant for the general-purpose hardware-efficient stream ciphers targeted in eSTREAM competition, a maximum possible clock frequency of 724.6 MHz (providing a throughput of 724.6 Mbit/s) along with a corresponding power consumption of 7772 μW (cf. [GB08]) for the eSTREAM portfolio member Grain v1 [HJM06] is by magnitudes larger than what ultra-constrained devices are capable of (also in terms of transmitting the encrypted data). Consequently, we will not treat the metrics *maximum clock frequency*, *maximum throughput*, or *delay* in detail in this thesis. For the sake of completeness, however, we will briefly discuss *delay* in the next subsection.

2.3.7 Delay (Critical Path)

As pointed out above, for creating hardware implementations of cryptographic schemes, so-called HDLs like VHDL or Verilog are used. These HDL descriptions are then transformed into logic circuits in a process called *synthesis* by means of special compilers such as Encounter RTL Compiler (see, e.g., Subsection 8.5.1 for a more detailed description at the example of our new stream cipher Lizard). To perform this transformation, compilers use so-called *standard cell libraries* auch as UMCL18G212T3 (0.18 μm, 1.8 V), where standard cells represent an abstraction for a combination of interconnected transistors

[5] Even if the tag itself should operate at a higher clock rate, it would be possible to use a so-called *clock divider* to operate the encryption circuit at a lower speed.

which realize some (simple) Boolean function such as AND, OR, XOR etc. Now when the input of such a standard cell changes (i.e., switching activitiy occurs), it takes a small amount of time until the output of this cell enters a new stable state. In particular, during this short time span, which is called the standard cell's *delay*, there is no guarantee about which value(s) are provided at the cell's output port(s). As a consequence, for a circuit where its input has to propagate through a cascade of standard cells (i.e., for a circuit which realizes a function of logical depth greater one) until it reaches the output port(s) or some flip-flop (where the corresponding value will then be available at the next clock cycle), the delays of the corresponding standard cells add up to the delay of the respective path. Clearly, even for very simple circuits, there will usually be many of such paths. The delay of the longest path (w.r.t. the delays of the standard cells contained in this path) is called the *critical path* of this circuit.

This delay of the critical path (commonly simply referred to by *the delay*), determines the maximum possible clock rate which the respective circuit can be operated at. For example, as for low-cost RFIDs, 100 kHz is the commonly assumed clock rate (see Subsection 2.3.6), this would allow a delay of up to 10^7 ps. For comparison, the delay of our implementation of Lizard is 2474 ps (see Subsection 8.5.1), which corresponds to a maximum possible clock frequency of about 404 MHz and, hence, allows for a throughput of up to 404 Mbit/s. This simple example already shows that delays as, e.g., considered for the general-purpose stream ciphers in the eSTREAM contest, play a negligible role in the context of lightweight ciphers for ultra-constrained devices and, in particular, should not serve as a design criterion. This is due to the fact that compilers can actually be instructed to optimize for smaller delays (in order to allow for higher clock rates), which, however, will lead to the use of more expensive components in terms of area and power, something, which is highly undesirable for extremely lightweight implementations. The reason why we provide these numbers anyhow for Lizard in Subsection 8.5.1 is to demonstrate that even without optimizing for this metric, our new stream cipher allows for being run at sufficiently high clock rates.[6]

2.3.8 Random Number Generator (RNG)

The hardware means of generating random numbers on a lightweight RFID tag can probably be considered the 'magic bullet' with respect to authentication protocols and are most likely the main reason why all of the hardware producers we interviewed demanded to remain unnamed. In [JW05], Juels and Weis state w.r.t. the famous HB and HB$^+$ authentication protocols (see Section 3.4) that the random *noise bit* ν (and probably also the *blinding factors* required as part of each protocol round; see, again, Section 3.4)

[6] Also note that, though targeting low-cost devices, Lizard can still be used in scenarios where throughputs larger than 404 Mbit/s are required, simply by using techniques like *pipelining* (as done for the stream cipher Espresso in [DH15]) in order to reduce the delay or by implementing a parallelized version of Lizard as described in Subsection 8.3.4.

"can be cheaply generated from physical properties like thermal noise, shot noise, diode breakdown noise, metastability, oscillation jitter, or any of a slew of other methods". While the listed physical properties can undoubtedly serve as a source for the generation of random bits, ensuring a sufficient level of entropy in these cases still constitutes a difficult task and is subject to research areas on its own. For example, [TBM08] presents a metastability-based true random number generator (TRNG) fabricated in 0.13 μm bulk complementary metal-oxide-semiconductor (CMOS) technology, which requires 0.145 mm^2 of area and consumes 1 mW of power (at a clock rate of 200 MHz). Even for lower clock rates (and, hence, lower power consumptions), the required area of 0.145 mm^2 would still render this TRNG infeasible as a component (i.e., as only one of many parts) of a low-cost RFID tag considering that "10 US cents RFID read only chips have design sizes ranging from 0.16 mm^2 to 0.25 mm^2" [REC05] and that the RNG's "circuit should not occupy more area than $100 \times 100\mu$m" [BB08].[7] TRNGs designed particularly for passive RFID tags exist, too, but we are only aware of those like in [BB08], which focus on generating 16-bit-long random numbers mainly meant for resolving collisions during communication. Hence, it is unclear to what extent such low-cost RNGs are actually suitable for generating large, continuous amounts of random bits (with sufficient entropy) in time as needed by many HB-type protocols for each authentication instance. For the sake of completeness, we would like to mention that there are also pseudorandom number generators (PRNGs) aiming at low-cost scenarios, but, e.g., LAMED [PLHCETR09] still consumes roughly 1600 GE, which is about 600 GE more than a serialized implementation of the lightweight block cipher PRESENT [RPLP08], which can be used straightforwardly to realize (one-way) authentication in the spirit of [FDW04] without the need for any random numbers at all on tag side. As none of the above TRNG/PRNG solutions seems to fit the scenario implied by HB-type protocols on ultra-constrained devices, at this point, we have resort to information provided to us by different experts from industry, who all agree that generating more than 128 true random bits per authentication on an RFID tag in the price range of \$0.05–\$0.10 seems currently implausible. For authentication schemes other than those of the HB-family, 128 true random bits would however be perfectly sufficient. This particularly applies to protocols based on the *principle of random selection* (see Section 3.5 and Chapter 4) and cipher-based approaches (see Section 3.3) like using our new stream cipher Lizard in a challenge-response authentication scheme (see Chapter 8), the latter of which would, as pointed out above, not need any random number generator at all on the tag side.

Note, however, that none of those HB-type protocols in Section 3.4 which are currently unbroken were ruled infeasible only because they require more than 128 random bits per authentication and, in addition, many protocols exceed this number even by magnitudes.

[7] In [BB08], a 0.13 μm CMOS process is used. For comparison, the AES implementation in [FWR05] is based on a 0.35 μm CMOS process and occupies 0.25 mm^2, which "compares roughly to 3400 gate equivalents" in this context.

Finally, another problem particular to HB-type protocols is that they depend on a specific probability distribution w.r.t. the noise bit ν and deriving such a fixed distribution from the aforementioned sources is also everything but a trivial task.

2.3.9 Non-Volatile Memory (NVM)

While the cost of volatile memory is usually implicitly included in the numbers for area through flip-flops/latches (respectively the components needed to build those), non-volatile memory is commonly provided through the use of EEPROMs. One drawback, however, to employing EEPROMs is their high latency. Moreover, from the first EEPROM memory unit on, corresponding charge pumps have to be included in the design in order to supply the high voltages necessary for memory programming. Hence, EEPROMs are not only a major cost driver in terms of money and area but also have a significant impact on a tags power budget when it comes to ultra-constrained RFID devices. Concretely, Ranasinghe and Cole state in [CR08] that, for low-cost RFID tags, the power required for a read operation amounts to 5–10 μW while "a write operation to its EEPROM will require about 50 μW or more", which would practically allow only read operations (in the field) given the aforementioned power limitations of, e.g., EPC UHF tags, and, hence, inhibit a tag from keeping values across a loss of power (for example, between two separate communication instances). For comparison, the implementation of our new stream cipher Lizard consumes only 2.1 μW (see Subsection 8.5.1). This is particularly important when comparing ciphers like Lizard or Grain, which need to load the key only during their state initialization phases, to ciphers that continuously access the secret key in the EEPROM also during keystream generation, like Sprout, Fruit and Plantlet (see Subsections 6.2.1 to 6.2.3 for further details).

With respect to area requirements, Nuykin et al. [NKTZ12] propose a low-cost 640-bit EEPROM for passive RFID tags fabricated in a 0.18 μm CMOS process, which requires a total area of 0.04 mm^2. They also compare their design to several other recent suggestions, which all require at least twice the area and mostly offer even less memory (e.g., 192 bits). It is therefore not surprising that, as compared to the targeted low-cost EPC-like devices, even significantly more expensive RFID tags like the HITAG 1 by NXP do not provide more than 2048 bits of EEPROM. In line with this, Juels and Weis assume "128-512 bits of read-only-storage" and "32-128 bits of volatile read-write memory" to be realistic memory resources available on low-cost RFID tags, not considering non-volatile read-write-storage at all [JW05]. Finally, our sources from industry also all agreed that 2048 bits constitute a plausible upper bound for current EEPROM sizes on ultra-constrained RFID tags in the \$0.05 to \$0.10 range.

As we have already mentioned ciphers that continuously use the secret key (which will be discussed in detail as part of Chapter 6) above, we would also like to point out that in [MAM17], Mikhalev et al. provide a further analysis of EEPROMs as a component in lightweight stream cipher design. While their evaluation does not treat the question

of power consumption, it provides valuable information on the effects that EEPROM access times have on the encryption speeds of ciphers that continuously use the secret key such as Plantlet (see Subsection 6.2.3) and Fruit (see Subsection 6.2.2). However, as the new cryptographic schemes that we suggest in this thesis need to access the secret key only two times (the stream cipher LIZARD; cf. Chapter 8) respectively four times (the $(n, k, L)^{\langle 80 \rangle}$ authentication protocol with $n = 64$ and four rounds; cf. Chapter 4) per execution, EEPROM access times are negligible in our case and will not be discussed in further detail.

2.3.10 Fixed-key Storage

As explained in the previous subsection, some modern lightweight cryptographic schemes can, despite their low area requirements, actually turn out to be unsuitable for many ultra-constrained RFIDs when used in connection with EEPROMs, because continuous access to this kind of key storage severely strains the power budget of respective devices. A more suitable option in such cases is to use *fixed keys*, which are irreversibly set at production time. This, however, comes with several drawbacks apart from the apparent one that, e.g., in case of a security breach, it is impossible to change the secret key of an RFID tag later on in the field. In particular, setting keys already at production time inevitably means that the tag manufacturer will know this key, too, which may be unacceptable for certain customers. Moreover, there is also the economical issue that such fixed-key tags cannot be produced (and sold to intermediaries) in customer-independent volumes, but have to leave the factory already in their final, specific configuration. Further drawbacks of using fixed keys depend on how these are implemented as will be explained below.

One common way of irreversibly setting the secret key already at production time is the use of key-dependent *masks*. In a nutshell, such masks are used in integrated circuit fabrication to apply the respective hardware designs to *wafers*[8] in a process called *photolithography*. This effectively means that the secret key is already part of the corresponding hardware circuit. However, while this can alleviate the need for additional components like EEPROMs (see Subsection 2.3.9) or fuses (see below) on low-cost RFID tags, it inevitably results in the potentially dangerous situation that large quantities of tags will now share the same irreversible key. Concretely, as production costs increase with each new mask (by thousands of U.S. dollars), the size of per-mask-batches must be big enough (i.e., hundreds of thousands or even millions of devices) to allow for per-tag savings (e.g., by removing the need for EEPROMs) which compensate for the additional costs of using multiple masks. At the same time, an attacker's outlook on, e.g., counterfeiting large amounts of items who are all protected by RFID tags using the same

[8] *Wafers* are plates of semiconductor material, which, at the end of the production process, contain the final microcircuits. Note that each wafer can (and usually does) consist of many independent circuits, which are then separated and packaged into individual casings.

key, may now easily justify the costs for mounting a key recovery attack against one of those tags. In Subsections 3.2.1 and 3.2.2, we will discuss this grave security hazard of *group keys* in further detail at the example of two real-world attacks by the author of this thesis against the infrastructure of the University of Mannheim.

Clearly, if the deployment scenario requires fully individual keys, key-dependent masks are not economical. Here, a fixed-key implementation alternative is the use of *fuses/antifuses*, which were already treated as part of Subsection 2.3.3 in the context of chip area. The advantage over key-dependent masks is that fuses/antifuses allow to set the secret key at a later point in the production process, at which it is actually feasible to do this on a per-tag basis. Moreover, once fixed, they neither suffer from the high power consumption nor from the latency problems characteristic of EEPROMs.[9] However, the general problems that come with factory-set fixed keys (as discussed in the first paragraph of this subsection) remain. And, like EEPROMs, fuses/antifuses are an additional component not required with key-dependent mask, which, e.g., increases chip area as explained in Subsection 2.3.3.

Finally, it should should also be noted that implementing fixed keys via masks or fuses/antifuses can actually introduce new security challenges. This is due to the fact that invasive hardware attacks (for instance, through etching and the use of an electron microscope) pose a real danger to RFID tags whose keys are set using these techniques. Especially if group keys are used (see above), an attacker will not hesitate to destroy a tag in the course of obtaining its secret key, because this same key can then still be used to manipulate all the other devices in the group.

2.4 Conclusion and Outlook

In this chapter, we presented all relevant hardware metrics that we encountered during our study of cryptographic algorithms for ultra-constrained devices. We will not provide a summarizing table of strict limits, however, for the following reason. As pointed out at several occasions above, many of the metrics are actually subject to a tradeoff (e.g., area/power vs. delay) or depend on the requirements of the actual application scenario (e.g., authenticating only one tag in at most 150 ms or a whole group of tags with only 18 ms time for each tag as done in [FDW04]). In particular for extremely lightweight algorithms, which operate at the very edge of what is technically possible, a collection strict of limits would in fact be counterproductive. Instead, we suggest to use the information presented in this chapter in the following way when designing a new cryptographic scheme for ultra-constrained devices:

(1) Ask yourself what your new scheme is exactly meant to provide. In particular, give a precise specification of its usage environment (How many tags need to be

[9] Remember that, as pointed out in Subsection 2.3.3, we are not talking about fuses/antifuses as a component of PROMs here.

authenticated simultaneously? What encryption speed is targeted? How long need the generated keystreams to be?) and hardware conditions (What capabilities is the RFID tag supposed to have?) in your design document.

(2) Identify the metrics and corresponding limits relevant for your use case by reading this chapter. Can ultra-constrained devices actually provide what your scheme needs? Which tradeoffs will be the most favorable ones?

(3) Design your scheme and create a first reference implementation with these metrics in mind.

(4) Double-check by consulting this chapter once more. Are all of the limits which are relevant for your use case satisfied? Might small tweaks of your algorithm even allow for a broadened range of application scenarios while still satisfying the limits?

Admittedly, this short 'design guide' might sound trivial or even naïve at first. However, our experience shows that, unfortunately, these basic points are often completely ignored in academic publications of new cryptographic schemes. This particularly holds for authentication protocols, as we will see in Section 3.4. For example, many of the allegedly lightweight authentication schemes discussed there simply claim to be lightweight due to the use of 'simple' operations, neglecting that they have to perform these operations an unfeasibly high number of times (implying unrealistically high clock rates or the violation of timing restrictions) or the fact that the respective hardware does not provide a powerful RNG. In the context of block ciphers and stream ciphers, on the other hand, it has been good practice for a long time now (probably also due to the respective competitions like the eSTREAM contest) to supply new design suggestions with at least some basic assessment of feasibility. Still, we think that also these design directions will benefit from the comprehensive collection of hardware metrics and limits for ultra-constrained devices presented in this chapter.

In the following Chapter 3, we will provide an introduction to the field of lightweight authentication. In particular, as indicated above, we will show (based on the limits established in the current chapter) that a broad range of allegedly lightweight authentication schemes is actually not suitable for ultra-constrained devices. In Chapter 4, we will then introduce a feasible alternative in the form of our new $(n, k, L)^{\langle 80 \rangle}$-protocol. The metrics and limits discussed above will also play an important role in Chapter 8 in the context of our new lightweight stream cipher Lizard, which explicitly targets power-constrained devices.

Da ging der Wolf fort zu einem Krämer und kaufte ein großes Stück Kreide, die aß er und machte damit seine Stimme fein. Dann kam er zurück, klopfte an die Haustür und rief: "Macht auf, ihr lieben Kinder, eure Mutter ist da und hat jedem von euch etwas mitgebracht."

Der Wolf und die sieben Geißlein (Gebrüder Grimm)

CHAPTER 3

Lightweight Authentication

ABSTRACT

Authentication solutions are probably the most common field of application for ultra-constrained RFIDs. They range from obvious representatives which many of us use every day, such as contactless smart cards for entrance control, to more hidden ones, such as RFID tags incorporated or attached to goods in order to track them or for protecting against product piracy. And the application areas of RFID-based authentication are constantly broadening. Human-implantable tags already exist, imposing new technological challenges (such as a low power and/or energy consumption) on lightweight authentication solutions, whose striving for minimality is currently still mainly motivated by the cost pressure for the production of corresponding devices.

In this chapter, we provide an introduction to authentication in general and its lightweight forms in particular. Based on examples of failed real-world authentication solutions, we develop a number of rules that should be considered when deploying RFID-based authentication schemes in the field. Furthermore, we describe and evaluate (based on the hardware limitations of ultra-constrained devices as introduced in Chapter 2) the three most common approaches for lightweight authentication: (block) cipher-based protocols, LPN-based protocols, and protocols based on random selection of secret linear functions. Building on a scheme of the latter type, namely the $(n, k, L)^{++}$-protocol, we will then suggest a new lightweight authentication protocol actually feasible for ultra-constrained RFIDs in Chapter 4.

Declaration of Origin: Subsection 3.1.1 is based on the paper *Hardware Efficient Authentication based on Random Selection* [AHK14], written together with Frederik Armknecht and Matthias Krause and presented at *Sicherheit 2014*. Sections 3.3 and 3.4 are based on the paper *Lightweight Authentication Protocols on Ultra-Constrained RFIDs – Myths and Facts* [AHM14], written together with Frederik Armknecht and Vasily Mikhalev and presented at *RFIDSec 2014*. Section 3.5 is based on the paper *The Cryptographic Power of Random Selection* [KH11], written together with Matthias Krause and presented at *SAC 2011*, and the paper [AHK14] (see above). Note that [KH11], in turn, is largely based on the diploma thesis [Ham10] of the author of this PhD thesis (see Footnote 20 on page 58 for further remarks).

3.1 Introduction

As pointed out before, embedded systems are increasingly permeating our information society, being more and more used also in security- and safety-critical applications. Devices of extremely small computational power like RFID tags are used in practice to a rapidly growing extent, a trend commonly referred to as *ubiquitous computing*. One of the major use-cases for such pervasive devices are authentication solutions, e.g., access control for buildings or cars, electronic passports or even human-implantable chips providing sensitive medical information about a person. Moreover, in times of increasing product piracy, secure (esp. unclonable) RFID tags play a major role in anti-counterfeiting solutions [STF05]. Similarly, they are also employed to guarantee the adherence to legal restrictions. In the U.S. state of Colorado, for example, the use of RFID technology is mandatory for "digital tracking of marijuana plants from seed to sale" [His14].

Before we go into the details of lightweight authentication, it is important to distinguish three different concepts that are closely connected and hence often mixed up, which can have devastating security impacts, as we will see, e.g., in Subsections 3.2.1 and 3.2.2:

Identification refers to the process of claiming an identity. This happens, for example, when you enter your username in some login form (e.g., in order to access your email account). Similarly, in our context of RFID communication, identification takes place when a tag broadcasts some unique identifier as in the case of Electronic Product Codes (EPCs), which are meant to provide an alternative to classic barcodes. Obviously, *radio-frequency identification* already contains this *identification* purpose right in its name. However, in our email example, anybody who knows *your* username (e.g., because it is your email prefix) could enter it into the respective login form. Similarly, by means of some dedicated hardware (see Subsection 3.2.2), some attacker could transmit the same identifier as the original valid RFID tag after eavesdropping on it. To prevent this, *identification* is usually accompanied by a second stage called *authentication*.

Authentication is (in our context) the act of *proving* that the previous claim made about the identity is actually true. This is usually achieved by one (or a combination) of three factors: something that the user *has* (e.g., a security token), something that the user *knows* (e.g., a password), or something that the user *is* (e.g., biometric authentication via fingerprint or iris scans). Hence, in our previous email example, the user would not only have to enter his username but also his password into the login form in order to be successfully *authenticated* by the email server. Similarly, RFID reader and tag would execute some kind of authentication protocol (e.g., a challenge-response scheme based on some common shared secret) in order to ensure that the tag claims its identity rightfully. After the process of authentication has been completed successfully, our user, however, cannot yet access his email account. This requires the additional step of *authorization*.

Authorization is the act of granting certain rights to some entity that has been successfully *identified* and *authenticated*. In the case of our valid user, this would mean that (e.g., based on some database with user rights) he is now allowed to access his mailbox. However, as he lacks the rights for the mailboxes of other users, those will be forbidden to him. (The same, e.g., applies to user folders on multi-user operating systems.) W.r.t. our RFID tag, the process of authorization could, e.g., lead to the opening of a door to some restricted area.

As pointed out above, understanding the differences between these three concepts is vital in terms of security. It is also important in order to understand what this chapter and the following Chapter 4 are meant to provide. More precisely, we will exclusively focus on the part of *authentication*. *Authorization* is naturally performed on the RFID reader's side, respectively by the *back end* (i.e., the underlying server infrastructure) which it is connected to. *Identification* is a large and interesting research area on its own, treating problems such as *privacy preservation*.

For example, already today, most people carry several RFID-capable cards (access cards, credit cards etc.) in their wallet and, for reasons of convenience, often simply hold this wallet in front of corresponding RFID readers. Without techniques for privacy preservation, this could, e.g., be used to trace to a user or to obtain other sensitive data such as membership information based on unique properties of the respective identification string. Possible countermeasures include hardware solutions such as *range reduction* (requiring, e.g., an RFID access card to be put right on top of the corresponding reader) or wallets made from special shielding materials that create a Faraday cage. Another approach is to strive for privacy preservation already on the protocol level. A very simple solution based on a symmetric secret key $k \in \{0,1\}^n$ could, e.g., look as follows: The *verifier* (i.e., the RFID reader) chooses a random challenge $c \in_R \{0,1\}^n$ and sends it to the *prover* (i.e., the RFID tag). The prover also chooses a random number $x \in_R \{0,1\}^n$ and computes the response $r := (x, F_k(c||x))$, where $F_k : \{0,1\}^{2n} \longrightarrow \{0,1\}^{2n}$ is some key-dependent pseudorandom function and $c||x$ represents the concatenation of c and x.[1] The verifier then searches his user database for some key k' such that $(x, F_{k'}(c||x)) = r$ holds. Obviously, this combined authentication-identification scheme has the severe disadvantage that it puts a heavy load on the verifier in case of large user groups due to the costs for finding the right key through repeated evaluations of $F_{k'}$ for different k'.[2]

Other, more efficient schemes (w.r.t. the costs on the verifier's side) exist, too, but they are commonly based on the use of asymmetric cryptography and additional building

[1] The reason why the prover also chooses some random number x is to thwart a *tracing attack* (e.g., to record the movements of a card holder within some area), in which the attacker would always send the same challenge c leading to identical responses due to the unique key k of the tag.

[2] Also note that this simple authentication scheme would be extremely susceptible to so-called *denial-of-service attacks (DoS attacks)*, in which the attacker would flood the verifier with authentication requests under non-existent keys in order to bring the verifier (respectively its back-end system) down (or render it at least unavailable to other, legal RFID tags) based on the resulting computational overload.

blocks such as hash functions or message authentication codes (MACs) (see, e.g., [BCI08] or [Vau07]). As such solutions, however, are currently infeasible for ultra-constrained RFID tags, we will not treat (privacy-preserving) *identification* in further detail in this thesis, but instead focus on the part of lightweight (privacy-preserving) *authentication*. In particular, when speaking of *privacy preservation*, we will thereby exclusively refer to its authentication-related aspects such as avoiding the aforementioned tracing attacks (cf. Footnote 1 on page 29).

Before we continue by providing an overview of approaches for designing concrete lightweight authentication schemes targeting ultra-constrained devices, it should be noted that, apart from the three different factors 'what you know', 'what you have', and 'what you are' discussed above, authentication has many more varieties. For example, besides *one-way authentication*, where, e.g., an RFID tag only proves its identity to the reader, there is also the concept of *mutual authentication*, where both parties prove their respective identities. Depending on the scenario, the failure to employ the latter can lead to devastating security problems as we will see in Subsection 3.2.1.

3.1.1 Authentication as targeted in this Thesis

In this thesis, which has its focus on ultra-constrained RFIDs as introduced in Chapter 2, the schemes we consider all have to operate at the very edge of what is technically possible in terms of minimality. In the spirit of this minimalism, we will concentrate on the most basic variant of authentication problem and try to solve it with as few resources as possible: an RFID tag seeking to prove its identity to the reader by means of some shared secret. The common approach here is to use challenge-response protocols (see our privacy-preserving example above), where the tag authenticates itself towards the reader by implicitly proving the knowledge of the secret. This is accomplished by answering challenges sent by the reader, where the responses depend on the shared secret.

Due to their hardware demands typical authentication protocols (e.g., those that are based on *asymmetric cryptography*) are usually not suited for ultra-constrained devices in the production cost range of \$0.05 to \$0.10. Consequently, the search for dedicated lightweight authentication protocols became an important topic in cryptography during the last years with high relevance for academia and industry, generating a significant number of different approaches and schemes.

Nonetheless, one can identify three approaches that can be seen as the most relevant principles for constructing lightweight authentication schemes today:

(1) protocols which use lightweight block ciphers as basic cryptographic operations,

(2) protocols which employ the well-researched principle of adding biased noise to a secret linear function,

(3) protocols which are based on the *principle of random selection*, being the most recent of all three paradigms.

Concerning approach (1): Block cipher-based protocols can be seen as a very straightforward approach for enabling authentication. The basic idea is that the verifier (e.g., an RFID reader) chooses a random value and sends as challenge the encryption of this value to the prover (e.g., an RFID tag). The task of the prover is to decrypt the challenge and to send back the chosen value in plaintext. (Trivially, the task of correctly encrypting an unencrypted nonce chosen at random by the verifier is equivalent here.) Obviously, the computational effort is mostly dominated by the execution of the deployed cipher. It has to be stated that very convincing proposals for lightweight block ciphers such as PRESENT [BKL+07], KATAN and KTANTAN [DCDK09], PRINCE [BCG+12], SIMON [BSS+13], and SKINNY [BJK+16] do exist, which have been analyzed in a large number of papers. However, such protocols are less flexible with respect to scalability than other approaches, e.g., as they always have to take the block size of the underlying block cipher into account.[3] Nonetheless, we will give a more detailed example for the principle feasibility of this approach w.r.t. ultra-constrained RFIDs in Section 3.3.

Clearly, as an alternative to block ciphers, also stream ciphers can be used as part of an authentication scheme. However, according to our sources in industry (see Chapter 2), corresponding solutions are currently less common on ultra-constrained RFID tags. In Section 9.2, we will present an FPGA 'prototype' of an authentication scheme feasible for such devices based on our new lightweight stream cipher LIZARD introduced in Chapter 8.

Concerning approach (2): The security of these kinds of protocols w.r.t passive attackers can be reduced to the widely accepted hardness of the LPN assumption. In a nutshell, LPN-based protocols all adapt more or less the following principle: given a challenge $a \in \mathrm{GF}(2)^n$, $n \in \mathbb{N}$, the response is computed as $f(a) \oplus e$, where $f : \mathrm{GF}(2)^n \longrightarrow \mathrm{GF}(2)$ is a secret (linear) function and e some noise bit which takes a value of 1 with a constant probability $p < 1/2$. Straightforwardly, the authentication process comprises of running the above protocol round many times and accepting finally iff the fraction of wrong answers remains below a certain threshold.

A severe drawback of these protocols is that presumably secure parameter choices often imply large amounts of transmitted data. Combined with the small available bandwidth in RFID communication, this may add up to authentication times unacceptable for many applications. A further major problem is that many variants (e.g., HB$^+$ [JW05], HB$^\#$ [GRS08], Trusted-HB [BC08]) have already been broken by active MITM attacks (see, e.g., [GRS05], [OOV08], [FS09]). In Section 3.4, we will discuss LPN-based authentication protocols in further detail and conclude that currently there does not seem to be a single unbroken suggestion feasible for ultra-constrained devices.

[3] In particular, depending on the block size and the size of the challenges and possible additional data (such as the random value $x \{0,1\}^n$ in the simple, privacy-preserving authentication scheme example given above), the hardware cost of block cipher-based authentication approaches can significantly increase beyond the cost for the core encryption algorithm, e.g., due to the need to also implement a suitable *mode of operation*.

Concerning approach (3): The *principle of random selection* implies that the secret key K consists of a small collection of L linear mappings $F_1, \ldots, F_L$. The prover computes responses to challenges a by randomly choosing one of these functions $F_l \in K$ and replying with $F_l(a')$, where a' depends on a in a way specified by the concrete protocol variant (see, e.g., Subsection 3.5.1 and Section 4.2).

The first protocols of this kind were the CKK-protocols given in [CKK08]. Further protocols based on the principle of random selection include the F_f-protocols in [BKM$^+$09] and the Linear Protocols in [KS09]. The most important and still unbroken suggestion of the latter type is the $(n, k, L)^{++}$-protocol, which is provably resistant w.r.t. to a wide family of active MITM attacks. Moreover, in analogy to HB-type protocols and the LPN problem, in [KH11] a learning problem called *RandomSelect* has been introduced and analyzed, and it is conjectured that "the complexity of RandomSelect also defines a lower bound on the security achievable by protocols using random selection of linear functions, e.g., the improved $(n, k, L)^{++}$-protocol".

In Section 3.5, we will further study the principle of random selection and also recall the $(n, k, L)^{++}$-protocol from [KS09]. This will then serve in Chapter 4 as the basis for our new $(n, k, L)^{\langle 80 \rangle}$-protocol, which, in contrast to the original $(n, k, L)^{++}$-protocol, is actually feasible for ultra-constrained RFIDs.

But before we go into details w.r.t. the above approaches for designing lightweight authentication schemes, we will now give some practical examples of security failures in the context of authentication.

3.2 Excursus: When Authentication Goes Wrong

The following two subsections contain anecdotal descriptions of flawed authentication approaches which the author of this thesis encountered during his time at the University of Mannheim. These examples shall serve to illustrate the importance of various points we have made so far in this chapter (e.g., identification vs. authentication) and Chapter 2 (e.g., random number generation). Note, however, that as the technical details of the respective attacks only partly fall into the general scope of this thesis, we will treat the corresponding background in this section at a rather high level. Our focus here is on the conceptual mistakes that enabled these attacks in the first place, as the resulting lessons learned equally apply for authentication schemes targeting ultra-constrained devices.

3.2.1 Compromising the University VPN

Virtual private networks (VPNs) are a very common method for protecting digital communication that has to pass an untrusted channel such as the internet. In particular, they are used to allow, e.g., remote workers to securely connect to the local area network (LAN) of their company. To this end, the client uses some dedicated software (or some

functionality pre-integrated in his operating system (OS)) to create a so-called *VPN tunnel*. After this tunnel, which is usually encrypted, has been established, the client can then access protected resources that would normally only be available via direct connection to the company LAN. One of the major providers of such solutions is Cisco Systems, which will also be in the focus of this subsection.

In March 2005, when the author of this thesis was a second-semester student at the University of Mannheim, VPNs, however, also served another important purpose. At that time, network communication over wireless local area networks (WLANs) was still a rather new technology in the consumer market and the means of protecting such connections were very limited. In particular, the Wired Equivalent Privacy (WEP) algorithm, which had been the standard way of encrypting WLAN communication since 1997 [Ins97], was already proven highly insecure by Fluhrer, Mantin, and Shamir in 2001 [FMS01] (due to a weakness in the key scheduling algorithm of the stream cipher RC4). In 2004, the Wi-Fi Protected Access (WPA) protocol [Ins04] was introduced as a successor to WEP, but at the beginning of 2005, a large number of devices did not yet support it. As a consequence, many German universities, including the University of Mannheim, resorted to another solution: In a first stage of the communication establishment, they allowed *unauthenticated* and *unencrypted* connections to their WLAN access points. These connections, however, were strictly limited to traffic between the respective client and a VPN appliance (usually a *Cisco VPN 3000 Series Concentrator*). In a second stage, the client then had to establish a VPN tunnel with this VPN appliance on the basis of his credentials (i.e., username and password). Only after this process had been completed successfully, the user was then allowed to access the university network (and the internet) through this encrypted tunnel.

This setup, which, as pointed out above, was used at many German universities at the beginning of 2005, got the author of this thesis wondering whether an attacker could 'make something out of it'. This suspicion was, in particular, sparked by the initial stage of establishing an *unauthenticated* and *unencrypted* WLAN connection, which, e.g., easily allows for (passively and, hence, undetectably) eavesdropping on the corresponding traffic between the clients and the VPN appliance during the negotiations of the respective VPN tunnels. Doing so, the author quickly learned that the corresponding VPN configuration obviously used to the so-called *Aggressive Mode*. In a nutshell, when using Internet Key Exchange (IKE) in this context as part of authentication setup, there are two options: *Main Mode* and *Aggressive Mode*. Both variants rely on Diffie-Hellman key exchange (DH key exchange) for establishing a common secret session key that is used in the later parts of the communication. However, in Aggressive Mode, the multi-step IKE phase of Main Mode is compressed into a smaller number of steps.[4] As a result, when Aggressive

[4]This has mainly two reasons: First, connection setup is faster due to the reduced number of authentication steps. Second, and more importantly, when used straightforwardly, Main Mode would require the use of static Internet Protocol addresses (IP addresses), which is hardly feasible in the context of, e.g., university networks.

Mode is used, parts of the authentication setup are not yet protected by the session key and, hence, leaked to an eveasdropper, the most important of it being the hash of the so-called *pre-shared key (PSK)*. In the VPN setup used by the University of Mannheim at the beginning of 2005, this PSK was employed as a *group key* (and all members of the university, including students, belonged to the same group, hence sharing the same key), which, during the IKE authentication phase, served to mutually authenticate a client and the VPN appliance. Consequently, everyone who was able to get hold of this PSK was, in principle, able to take the role of the university's VPN appliance. This will play an important part in the later course of our attack.

So what are the means to learn this important PSK? In 1999, John Pliam had already pointed out to the Internet Engineering Task Force (IETF) severe security problems w.r.t. authentication via IKE and *Xauth* (see later) with weak PSKs [Pli99]. In his respective messages, he drafts how to obtain the PSK from the IKE phase for Main Mode and Aggressive Mode based on dictionary attacks[5], where in Aggressive Mode, the attack can even be performed passively (i.e., using only eavesdropping techniques and not interacting with the valid communication parties). However, dictionary attacks can be very costly (and remember that these events take place in 2005), so the author of this thesis had to resort to another way of obtaining the PSK.

Fortunately, being a member (and, hence, valid user) of the university himself, he had access to the configuration file that each VPN client program needed to be supplied with. Naturally, in order to be used in the aforementioned authentication process, the PSK had to be contained in this file. However, for security reasons, it was there only in encrypted form. But clearly, at some point during the connection process, client software would load it. To the help of the author, older versions of the corresponding Cisco VPN client had the weakness that during connection establishment, a plaintext version of the encrypted PSK could be extracted by standard tools from the volatile memory of the respective computer system. Cisco acknowledged this problem in 2004 [Cis04], but susceptible client versions could still be obtained in March 2005 and thus allowed the author of this thesis to use this type of key recovery. (In October 2005, the situation became then even easier for attackers, when HAL 9000 of the 'security' group Evil Scientists published his *Cisco Password Revealer*, which was based on reverse engineering the code that Cisco used for encrypting the PSK before storing it in configuration files, hence allowing for direct key recovery.)

So at this point, the author of this thesis was now able to take the role of the university's VPN appliance. Through the aforementioned messages by John Pliam [Pli99] from 1999 and various recent computer emergency response team (CERT) warnings (like that of RUS-CERT of the University of Stuttgart [RUS04]) from 2004, he then became aware

[5] A *dictionary attack* targets weak passwords by searching for common terms (and variations of those) in a dictionary supplied by the attacker. In the case of the above attack against Aggressive Mode, these potential passwords are hashed and compared to the hash of the actual PSK obtained via, e.g., eavesdropping.

that in many cases, *Xauth* was used on top of IKE for additional client authentication (remember that through the IKE authentication part discussed above, the user and the VPN appliance have only been mutually authenticated using the PSK, i.e., a group key; consequently, the VPN appliance does not yet know *which* of its users is actually trying to establish a VPN tunnel). During this additional user authentication step, the respective username and password were actually sent through the newly established VPN tunnel to the VPN appliance; in particular, no kind of, e.g., challenge-response protocol was performed to additionally protect the password. As it quickly turned out, also the University of Mannheim used this problematic setup.

So just by studying corresponding security reports, the author of this thesis (then being a second-semester student) now had enough information to be convinced that it would be possible to compromise the university's VPN system and obtain the login credentials of its users. However, two technical parts of the attack were still open: First, it was necessary to set up a fake VPN appliance under the attacker's control, which had to behave like the proprietary Cisco solution at least until the login credentials had been transmitted. After this point, the connection could then be dropped as the goal was only to obtain the user passwords and not to mount a full MITM attack. Second, the valid users actually had to be lured into connecting to this fake VPN appliance.

Realizing the second part was fairly easy as the university's WLAN access points did not authenticate themselves in any way to their users. Instead, the clients automatically connected to those access point which broadcasted the name of the university WLAN and had the strongest signal. Here, it is important to know that common WLAN devices only use a signal strength of 30 mW. With special equipment (a *Senao NL-2511CD PLUS EXT2 PCMCIA card* with *Prism 2.5 chipset*), the author was however able to set up a WLAN access point with a signal strength theoretically[6] beyond 200 mW. This ensured that newly connecting clients in the periphery of the author's system would always connect to his WLAN access point instead of to the university's one. Moreover, even already connected users could be lured to this fake access point by first deauthenticating them from the regular ones with the use of tools like *void11* [van05].

The final (and more difficult) step was to create a fake VPN sever compatible with Cisco clients. Obviously, due to monetary reasons, obtaining an original Cisco VPN 3000 Series Concentrator was out of scope for the author being a student. So the only possible solution seemed to take some open-source VPN server software and modify it in a way compatible with the proprietary Cisco protocol. Fortunately, during his corresponding research, the author of this thesis then learned that Philippe Sultan had already started a similar project in 2004 [Sul04] based on the Openswan [Xel17] IPsec implementation for Linux. While some additional changes had to be made to the code provided by Philippe Sultan in order to increase compatibility and also to add some functionality

[6] In 2005, 100 mW was the legal maximum signal strength in Germany, so, naturally, the author of this thesis never exceeded this limit during his tests back then.

(such as improved credential logging), in the end this was a task that could be done by a second-semester student with some diligence and time.

Now, the complete setup was ready to be tested in a controlled environment in order not to break any laws. And, in fact, everything worked as intended. That is, in a first stage, the clients connected to the fake WLAN access point provided by the author instead of the regular one of the university. Then, they started to initiate connections to the fake VPN appliance and trusted it to be valid based on its knowledge of the PSK. Finally, they transmitted their username and password (which were automatically logged by the fake VPN server) and the connection was terminated by the fake access point.

After the author of this thesis had completed his proof-of-concept setup, he informed the university authorities and, following a transition period of several months, the whole VPN system was updated to a new, certificate-based solution, in which asymmetric cryptography was now used to prove the server's identity. A configuration of this type is still in place (and seemingly secure) today, many years after the described incident happened in March 2005.

As a final note, it should be pointed out that if revealed to the general public of the university, the consequences of this attack would have been disastrous. This is due to the fact that the captured VPN credentials were not only used for network access, but actually severed as the general login details for almost all electronic services provided by the university. In particular, they also allowed access to the mail accounts of students and staff members (including professors). Moreover, for student accounts, they allowed to subscribe and unsubscribe the respective students to any kind of exam in their course of studies and, for staff accounts, they allowed to access the electronic grade management system of their chairs, including the possibility to see student grades from the past years and enter new grades for current courses. Fortunately, the about five months of transition period, until the new VPN setup could be launched, passed without leakage (or independent discovery by some other student) of the attack described in this subsection, so, in the end, no harm to the university or its members seems to have resulted from the failed VPN authentication solution.

Now what are the lessons learned from this security incident and how do they transfer to the field of lightweight cryptography for ultra-constrained devices?

Key Management. Symmetric group keys should only be used with great care. In particular, it is of vital importance that the respective group members can actually be trusted (this was not the case for the PSK in the VPN scenario targeted above and will also play an important role in the following Subsection 3.2.2). But especially for low-cost RFID devices, there is another danger to the use of group keys, which was already mentioned in Subsection 2.3.10 and results from the trivial principle that the higher the potential gain, the more an attacker will invest into reaching his goal. The use of lightweight cryptographic solutions (with, e.g., 'only' 60- or 80-bit security), on the other hand, is often justified by the assumption that no

attacker will spend considerable resources in order to recover the secret key of, e.g., a \$0.05 to \$0.10 RFID tag that is used to protect a single box of low-cost drugs like Aspirin against counterfeiting. If, however, the producer uses the same key for all of his boxes, it might now actually be worth the effort for an attacker.[7]

Service Separation. Use low-cost RFIDs only for low-impact purposes and, in particular, protect separate resources (with different sensitivity) by separate means, even if the user is the same. This may sound trivial, but in the above VPN example, the lack of using different login credentials for different purposes was exactly what made the attack so dangerous (as it allowed to also access, e.g., mail accounts and the grade management system through the intercepted VPN user passwords).

Key Protection. The secret key is not only vulnerable in its non-volatile storage location, but also during its usage by a cryptographic algorithm. In the above VPN attack, this allowed the author of this thesis to extract the plaintext PSK from the random-access memory (RAM) of his personal computer (PC), even though it was stored in encrypted form in the configuration file on his hard drive. Similarly, especially for RFID tags, not only key recovery via targeting the EEPROM or breaking the cryptographic scheme has to be considered, but also side-channel attacks, which were already discussed as part of Subsection 2.3.3, pose a serious threat. For example, in [OP11] a side-channel key recovery attack against the popular *MIFARE DESFire MF3ICD40* contactless smart card is presented. Its predecessor, the *MIFARE Classic*, will be in the focus of the following Subsection 3.2.2.

3.2.2 Compromising the University Member ID Card

The next authentication-related security incident which we are going to analyze took place at the beginning of 2013. It is centered around the *MIFARE Classic*, an ASIC-based contactless smart card, which was introduced by the Austrian company Mikron in 1994. The name MIFARE is an acronym for *Mikron Fare Collection System*, describing the initial purpose of these low-cost cards with very limited computational power. In 1995, Mikron was acquired by Philips, whose semiconductor branch was then separated and sold in 2006, forming the new company NXP. As a product of these major companies, the MIFARE branch (including successors of the MIFARE Classic such as the MIFARE DESFire) has since turned into the world's most used type of contactless smart card, with

[7] We will see in Chapter 6 that some modern lightweight cryptographic schemes can even require the use of such group keys in order to be used efficiently. This is due to the fact that the common way of storing secret keys on ultra-constrained devices is through EEPROMs, which, if accessed continuously (as done by these schemes), severely strain the power budget of respective devices (cf. Subsection 2.3.9). A more suitable option for respective cryptographic schemes such as Plantlet (see Subsection 6.2.3) and Fruit (see Subsection 6.2.2) is to use *fixed keys* as introduced in Subsection 2.3.10. In certain cases (see, again, Subsection 2.3.10), however, this is only economical if large amounts of devices share the same fixed key, thus making it a group key and resulting in the aforementioned security hazards.

more than 10 billion ICs sold [NXP17]. According to [GRVS09], in 2009, the MIFARE Classic alone covered "more than 70 % of the contactless smartcard market".

However, with its enormous commercial success, the MIFARE Classic also made its way into applications for which it had never been designed in the beginning. Despite its initial purpose as a device for convenient *micropayment* (i.e., transactions involving very small amounts of money in the range of few USD), companies started deploying it in much more security-sensitive scenarios such as identity verification and access control. In this function, under the name *ecUM*, the MIFARE Classic was also used at the University of Mannheim in 2013, where the author of this thesis was working for the chair of theoretical computer science. Apart from access control for parking lots (20 €/month) and buildings, this MIFARE Classic-based member ID card for students and employees also served for, e.g., lending books, copying/printing, paying the semesterly student fees (> 100 €), and buying the public transport student pass (> 100 €) at special terminals as well as meals and drinks at the canteen. The count of corresponding card holders was at least 40 000, as attached institutions (such as the *Duale Hochschule Baden-Württemberg* and the *Hochschule Mannheim*) used the same card, too. These numbers already show that a compromise of the ecUM could have led to severe financial damage.

While one might already wonder why someone would want to use a smart card for micropayment from the nineties for *macropayment* (i.e., transactions involving larger amounts of money) in 2013, this gets even more disturbing when knowing that successful attacks against the MIFARE Classic started to appear as early as 2007, when Nohl and Plötz presented a partial reverse engineering of its algorithms at the *24th Chaos Communication Congress* [NP07]. Up until then, the card's vendors had tried to increase its security by keeping the inner workings secret. This foolish approach, which we will discuss in further detail at the end of this subsection, was then completely shattered in 2008, when Nohl et al. published the paper [NESP08] titled "Reverse-engineering a Cryptographic RFID Tag" about their findings and de Koning Gans et al. demonstrated how to read and modify data on MIFARE Classic cards without knowing the corresponding secret key [dKGHG08]. In quick succession, several more such papers appeared in 2008 and 2009 (e.g., [GdKGM$^+$08] and [GRVS09]), which now even described how to recover the secret keys[8] and how to clone cards (such as those of the public transport system of London and the Netherlands, respectively; see [GdKGM$^+$08]).

The security flaws, which these attacks were based on, are manifold. In particular, after having been reverse engineered, the card's stream cipher called *Crypto1* was found to be extremely weak. It uses only 48-bit symmetric keys and is based on a single 48-bit nonlinear feedback shift register (NFSR) that contains the main secret state. It is therefore not surprising that, already in 2008, an *algebraic attack* (see Subsection 8.4.4 for an explanation of this term) was published by Courtois et al. [CNO08], which can

[8]The memory of the MIFARE Classic is divided into several sectors, which can be protected by individual symmetric 48-bit keys.

recover the secret key in about 200 seconds on an off-the-shelf PC, if only about 50 bits of keystream are known. To achieve this kind of performance, the authors of [CNO08] used so-called *SAT solvers*, which will be discussed as part of Section 9.4. In [GdKGM+08], two different attacks against Crypto1 are proposed. The first one is (though not explicitly mentioned by the authors) essentially an application of Babbage's classical TMD-TO attack against stream ciphers [Bab95] (see Chapter 7 and Subsection 8.4.2 for extensive details on this type of attack). In a precomputation phase, for 2^{36} of the 2^{48} possible secret inner states, the first 64 bits of keystream are computed and stored as (*inner state*, *keystream*)-pairs in a table. Then, in the online phase, the attacker searches for a corresponding collision based on 2^{12} 64-bit keystream pieces obtained from 2^{12} real-world authentications of the targeted device. Based on the birthday paradox, w.h.p., such a collision is found and yields the respective secret inner state. The authors of [GdKGM+08] also show how, based on this state, the secret key can then be recovered. Note that, while applying a TMD-TO attack against a stream cipher with only 48 bits inner state size might seem trivial at first, the important contribution in [GdKGM+08] is that the authors also show how the required keystream can actually be obtained based on a weakness in the authentication protocol of the MIFARE Classic, which leaks 64 bits per authentication instance. Naturally, this latter result can also be combined with the aforementioned algebraic attack in [CNO08], which requires only 50 bits of known keystream (hence, eavesdropping on only a single authentication instance is sufficient) to recover the secret key in about 200 seconds on a standard PC. However, in addition to the above TMD-TO attack, the authors of [GdKGM+08] also suggest a second, more efficient attack, which makes use of a weakness in the Crypto1 stream cipher itself and can recover the secret key within only one second based on a single authentication trace. To this end, they exploit that the nonlinear output function of Crypto1 uses only feedback shift register (FSR) cells with odd indices. As pointed out at the beginning of this paragraph, many more security flaws of the MIFARE Classic have since been discovered. In particular, the RNG (see Subsection 2.3.8) used for generating the authentication nonces was also found to be surprisingly weak. In [GRVS09], this fact (along with some other observations about the employed authentication protocol) is used to mount what the authors call the "nested authentication attack". Given the secret key for one of the 16 sectors of a MIFARE Classic 1K[9], this attack is able to recover the secret keys of the other sectors in under one second. The implications of this are vast as the nested authentication attack just requires card-only access. That is, if a MIFARE Classic is used for various purposes (as in the case of the University of Mannheim's ecUM), where each of these different applications has its own card sector and a corresponding separate secret key, it is still sufficient for an attacker to eavesdrop on only one real-world authentication

[9]The MIFARE Classic contactless smart card was available in two different memory sizes: the MIFARE Classic 1K offers 1024 bytes and the MIFARE Classic 4K offers 4096 bytes. While differing in sector numbers and sector sizes, both variants use the same security mechanisms and are hence both vulnerable to the described attacks.

instance for one of the sectors. After finding the secret key of this sector with one of the above attacks, he can then recover the keys of the other sectors without any further eavesdropping and the associated danger of being caught. (For example, eavesdropping on the authentication of a smart card at a printer system located in some 'darke corner' is clearly less dangerous for an attacker than eavesdropping on an authentication instance at the main entrance's door lock, which might be protected by a different card sector with a different secret key.)

So, given that, through the above publications and the corresponding media coverage, the author of this thesis already knew in 2008 about the MIFARE Classic's severe security problems, why did the following attack against the University of Mannheim's ecUM take place not until 2013? The answer to this question is already partly contained in [GRVS09], where Garcia et al. state:

> "[C]ontactless smart cards are generally not the only security mechanism in place. For instance, public transport payment systems such as the Oyster card and OV-Chipkaart have a back-end system recording transactions and attempting to detect fraudulent activities (such as traveling on a cloned card). Systems like these will now have to deal with the fact that it turns out to be fairly easy to read and clone cards. Whether or not the current implementations of these back ends are up to the task should be the subject to further scrutiny." [GRVS09]

In the same spirit, the author of this thesis assumed that also the University of Mannheim's ecUM infrastructure would feature such a second security layer as part of its back end. In fact, a corresponding protection could have been realized fairly easily as, even after the attacks of 2008 and 2009, an important hardware element of the MIFARE Classic still remained (partly) secure: each card had its own 4-byte unique identifier (UID), which was burned in at the factory and could not be changed afterwards. Moreover, hardware devices like the *Proxmark 3* (see below) did not support full card emulation for the MIFARE Classic yet, i.e., while they could send an arbitrarily chosen UID to some reader, it was not possible to realize a completely virtual card based on the memory dump of an actually existing one. In an environment like the University of Mannheim, where the MIFARE Classic-based ecUM was only handed out to users who proved their identity by means of an official passport document, the uniqueness of each card's UID[10] would have hence allowed to trace fraudulent card modifications back to the corresponding card holder if a corresponding database had existed. Furthermore, it would have been possible to not accept cards with nonexistent UIDs at all.

[10] Given the limited number of 2^{32} possible 4-byte UIDs combined with the vast commercial success of the MIFARE Classic, it has to be expected that there actually exit cards which share the same UID. However, we will assume it as being truly unique here as an attacker would have had to buy huge amounts of cards in order to find such a collision, which is clearly not economical in the context of targeting a university ID card such as the ecUM.

At the beginning of 2013, however, this situation changed drastically with the appearance of the so-called *Magic Chinese Cards* on the Chinese black market. These special MIFARE Classic cards, whose sale had not been authorized by the rightholder NXP, now also allowed to change the UID (arbitrarily often) in the field and, hence, finally enabled the creation of perfect clones. For about $15 per card, the author of this thesis was able to import a bunch of these from China in March 2013.

Eventually allowing for total anonymity, it was now time to evaluate the strength of the ecUM's back end. In a first step, the author targeted the electronic door locks, which, on the whole campus, were used to control the access to parking lots, buildings and certain offices. Shockingly, he soon learned that the authentication scheme for these door locks was not based on any secret information at all. Instead, it was sufficient if a card (or another emulation device) sent a UID that had previously been added to the access control list of the respective lock. Note that the MIFARE Classic's UID is public in the sense that it is not protected by encryption and any reader (not only 'valid' ones) who is in close proximity to such a contactless smart card will receive this information. For example, it is already sufficient if an attacker with a malicious reader 'accidentally' touches (for less than one second) the back pocket, containing a wallet with the ecUM in it, of a victim waiting in line at the canteen. Similarly, one could also place such readers in or under seats, e.g., in the library, or simply manipulate valid readers by hiding a second, malicious one in the respective casing. So, as UID-only MIFARE Classic card emulation was already possible in 2013 (e.g., using a small $300-device called *Proxmark 3* [PRO17]), the author of this thesis did not even need the Magic Chinese Cards to open corresponding locks. Instead, he could simply *root* an off-the-shelf Android smartphone (i.e., replace its operating system by a custom one which allowed access to the underlying Linux system) and connect the Proxmark 3 to it via Universal Serial Bus (USB) after installing the corresponding drivers and software. This gave him an unobtrusive bundle fitting into a larger wallet and allowing to flexibly send arbitrarily chosen UIDs to targeted readers in order to open the corresponding doors. Naturally, the Magic Chinese Cards worked as well (and were even less obtrusive), but the fact that they were not even needed shows that the ecUM-based access control system of the university had probably already been vulnerable for years (using the UID-only emulation feature of the Proxmark 3), when the author performed his attack in 2013.

After this disturbing discovery, the next logical step was to target the payment functions of the ecUM. To this end, it was now necessary to read the memory contents of a real-world card. This, however, required to know the secret keys of the corresponding card sectors. Being a member of the university himself and, hence, having access to a valid ecUM, the author of this thesis was able to use the aforementioned attacks (more precisely, a combination of the eavesdropping-based sector key recovery attack from [GdKGM$^+$08] and the nested authentication attack from [GRVS09]) to recover all 48-bit sector keys of his own card. This allowed him to dump the full memory contents of his ecUM into a file on his PC. While this memory dump already revealed some information

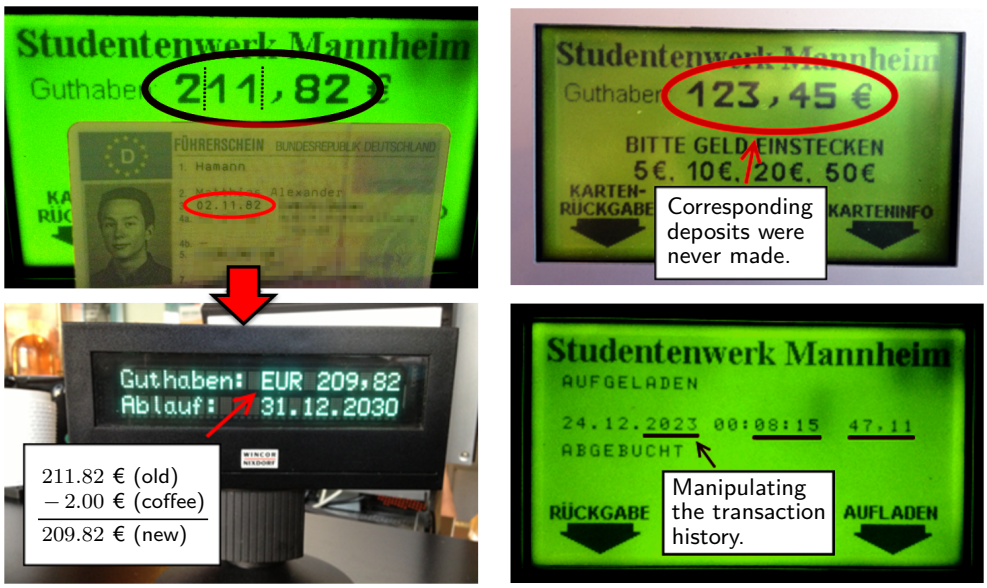

Figure 3.1: Some examples of manipulations of the ecUM that we performed during our attack. Interestingly, it was even possible to override the card's 200-EUR limit and use the corresponding credit without setting off any local or back end-based alarms.

about the card contents at first sight, such as the card number in American Standard Code for Information Interchange (ASCII) format (also printed on the card itself), most of it initially looked like a random collection of bytes. To further analyze the data stored on his card, the author then went on to perform a multitude of card transactions and, between each two of these, created a new memory dump. By comparing these dumps, he then gradually learned where and how the money on the card as well as the corresponding checksums were stored. Moreover, using dumps of other cards (kindly provided by colleagues and student helpers) as well, it was also possible to identify additional information such as the member type (e.g., students have to pay less at the canteen than employees) and the card's transaction history. Based on these findings, the author was then finally able to fully manipulate (and use!) his card at leisure (see Fig. 3.1).

Given the previous results from 2008 and 2009, the fact that, after some grunt work, the card contents could be modified arbitrarily was not really surprising. The following discoveries, however, were extremely disturbing. First, after having broken his own card, the author could use the same secret keys for accessing any other ecUM as well. Deploying such an infrastructure is grossly negligent in the context of a university as it basically assumes that any user of the system is honest (an assumption, which already lead into disaster in the case of the VPN compromise presented in Subsection 3.2.1). For a smart card like the ecUM, it takes only few seconds to read the full card contents and,

thus, steal someones identity and money.

Second, and probably even more irritating, two months of further analysis revealed that there seemed to be no back-end protection mechanism in place at all. More precisely, it was not only possible to use manipulated real-world cards for more than two months without any blocking or at least investigation from the operator's side, but even fully made-up cards (with nonexistent UIDs and ecUM card numbers) were accepted at any point in time during our investigation. When we finally informed the responsible office of our findings (providing a full history of our transactions), obviously nobody had noticed that there had some fraud been going on. This complete lack of back-end security also revealed that, in contrast to the author's initial expectations, even for attacking such a security-sensitive application as the ecUM's payment function, the new Magic Chinese Cards would have actually not been necessary. In particular, the full anonymity and the possibility to create perfect clones (including the UID) provided by the modifiable UIDs of these cards turned out to be completely irrelevant, because the back end of the university's card payment system obviously did not make use of UIDs as an additional protection layer anyhow.

During the aforementioned period of two months, in which we waited whether the ecUM's back-end system would finally detect our fraudulent activities, the author decided to additionally exemplify the impact of the card's security problems by means of modern smartphone technology. More precisely, many current smartphones feature near-field communication (NFC), e.g., to allow for contactless payment. At the beginning of 2013, one of the most common of these NFC-capable devices was the *Samsung Galaxy S3* with (as of 2015) more than 70 million units sold [Ham15]. As it turned out, the corresponding Android (the Galaxy S3's operating system) NFC programming library already supported communication with MIFARE Classic contactless smart cards. Having recovered the secret keys used by his university, it was hence an easy task for the author of this thesis to create a small Android app, which could read and modify any ecUM card. In particular, it allowed for arbitrary manipulations of the respective credit without any corresponding real-world money transactions. Moreover, using the app, a user would have also been able to easily alter his status from, e.g., employee to student, leading to reduced prices at the canteen. Finally, when Magic Chinese Cards were used as the app's writing target, it also allowed to fully clone cards or change a card's UID in order to get access to, e.g., the university's buildings and parking lots.

For obvious reasons, the above Android app, which would have enabled even non-technical users to easily commit fraud, was never published, but only used for internal demonstration purposes. Several months after we presented our findings, the ecUM was finally converted to another smart card type (the *MIFARE DESFire EV1*). It was probably only a matter of luck that seemingly no one else at the university discovered and exploited the weaknesses of the ecUM in the meantime as, given the large number of users and the security-sensitive applications of the card, this might have led to severe financial damage for the users and the operator.

Now, like in Subsection 3.2.1, what are the lessons learned from this incident and how do they transfer to the field of lightweight cryptography for ultra-constrained devices?

Kerckhoffs' Principle. Obey Kerckhoffs' principle (originally published in 1883 as part of [Ker83a] and [Ker83b]), which, adapted to contemporary cryptography, states that the security of a cryptosystem should solely depend on the secrecy of the key and, in particular, not on keeping the respective algorithms secret (see, e.g., [vTJ11]). Besides the MIFARE Classic debacle with Crypto1, many other real-world security failures are the result of disregarding this basic rule, as well. For example, shortly after the A5/1 cipher (see Subsection 5.2.2) of the Global System for Mobile Communications (GSM) standard had been reverse-engineered by Brienco et al. [BGW99], the scheme was fully broken and key recovery could even be performed on standard PCs (see, e.g., [BSW01]). The proprietary stream ciphers of the satellite phone standards GMR-1 and GMR-2 suffered a similar fate in 2012 [DHW+12]. While especially for ultra-constrained devices, whose cryptographic algorithms have to operate on the very edge of what is technically possible, it might be tempting to violate Kerckhoffs' principle and rather follow a '*security by obscurity*'-approach, history has shown that reverse engineering will be successful rather sooner than later, always eventually imposing the whole security burden on the strength of the respective cryptographic algorithms and the secrecy of their keys.

Key Management. Just like in the case of the VPN attack discussed in Subsection 3.2.1, the use of group keys was a fatal mistake for the ecUM, too. In the context of heterogeneous organizations like a university, the underlying assumption that all group members are honest (or unable to tamper with a device under their control), is disturbingly naïve. With group keys, even physical key recovery attacks that lead to the destruction of the targeted device (such as slicing and analyzing the chip layer by layer) become attractive. In consequence, especially for 'weak' targets such as ultra-constrained RFIDs (where, e.g., implementing strong protection against side-channel attacks may be uneconomic), group keys should rather be avoided, even though this might be challenging for some cryptographic schemes as pointed out by Footnote 7 on page 37.

Back-end Security. In short, a weak *front end* requires a strong *back end*. As explained above, regarding the ecUM, the lack of back-end security could have led to huge financial damage to the university, as corresponding fraud would not have been detected. While ultra-constrained RFIDs should not be used for applications like macropayment anyhow, they may still constitute a worthwhile target for an attacker due to their sheer number. Consequently, it is either necessary that, though being individually weak, these devices are strong as a group (e.g., by avoiding group keys), or that part of the security load is outsourced to the respective back end (e.g., through the introduction of plausibility checks for tag/group behavior).

Awareness. In the case of the ecUM, several conceptual mistakes where made that can be subsumed under *lack of awareness*. First, either misled by the vendor's marketing or due to carelessness (and a lack of proper research), a hardware device from the nineties originally designed for convenient micropayment was used in 2013 for much more security-sensitive applications. Second, even though the MIFARE Classic's security had been proven to be close to zero already in 2008, the ecUM remained in use until our attack in 2013. The only reason why the author of this thesis had not tried an attack before was his (wrong) believe in the existence of an appropriate back-end security mechanism based on the uniqueness of the cards' UIDs, as explained above. The office responsible for the ecUM, on the other hand, should have known immediately that such a security mechanism did not exist and that, in consequence, already from 2008 on, the card was open to any kind of fraud. The fact that, between 2008 and 2013, the ecUM was not migrated to some newer contactless smart card can hence only be explained by a lack of awareness of the MIFARE Classic's severe security issues.[11] Third and finally, also the awareness of the impacts of technological progress on the criticality of the issue was missing. When we initially reported that we were able to manipulate the ecUM at leisure with the Proxmark 3, we had to face the comment that this would not be a problem as the 'ordinary student' did not own such a $300-device. It was only until we presented the aforementioned smartphone app that the actual severity of the problem was recognized.

Clearly, these three *awareness issues* w.r.t. the ecUM straightforwardly transfer to the field of ultra-constrained RFIDs. In particular, developers should not overload such devices with inappropriate functionality in the course of time. Moreover, due to the small security margin provided by lightweight cryptographic algorithms, a raised awareness of advances in cryptanalysis is not only indispensable on the parts of manufacturers and vendors, but also for customers of corresponding products.

Identification $\neq$ Authentication. Another mistake made for the ecUM was to mix the different concepts of *identification* and *authentication* as introduced at the beginning of Section 3.1. As explained above, the UID of a MIFARE Classic contactless smart card is public information that can be easily *spoofed* (i.e., copied to or emulated) by other devices, and, hence, only suitable for identification (e.g., as a means of avoiding communication collisions when several tags are in the proximity of the same reader), but not for authentication. In the following sections and in Chapter 4, we will show how authentication for ultra-constrained RFIDs can be done properly, based on the use of symmetric cryptography.

[11] In defense of the respective administrators, one could argue that part of the blame lies also with the manufacturer/vendor of the ecUM system, as, particularly in the case of such a large infrastructure, they should have warned their customers vehemently.

Random Number Generation. Several attacks against the MIFARE Classic made use of the card's weak mechanism of generating random numbers (see, e.g., [GRVS09]). One particular problem is that the MIFARE Classic always initializes its RNG with the same value, which allows to narrow down the set of actually used random numbers by measuring or controlling the time between the (passively powered) card's start-up and the sending of its first message. This example once more confirms our notion already expressed in Subsection 2.3.8 that random number generation belongs to the most restricting aspects of lightweight cryptographic design for ultra-constrained devices. In particular, authentication protocols which, during each single authentication instance, require huge amounts of random numbers to be operated securely, are practically infeasible, even though they might be extremely lightweight with respect to other important metrics such as chip area (cf. Subsection 2.3.3). In Section 3.4, we will evaluate some of these failed approaches at the example of LPN-based authentication.

3.3 On the Principle Feasibility of Cipher-based Lightweight Authentication

Before we are going to assess dedicated authentication protocols suggested for constrained hardware, we will first point out the existence of an intuitive and, in fact, perfectly feasible approach, which makes use of existing encryption schemes: the verifier sends a random challenge to the prover, asking to encrypt it with a secret key, and finally checks whether the response is correct, ultimately leading to accepting or rejection. Typically, due to the harsh resource constraints in lightweight cryptography, only symmetric variants of encryption schemes are used as primitives for this type of protocols.

It should be noted that such cipher-based schemes are not only popular from a theoretical but also from a practical point of view, i.e., they are actually being used in industry. The reasons for this are twofold: Firstly, choosing the 'right' cipher as the encryption primitive in the above protocol description actually allows for creating feasible solutions whose security is dominated by the security of the underlying encryption function. But before we go into details about that, we would like to briefly mention a second, less technical, but in industry sometimes even more important factor: standardization. In Section 3.4, we will argue that dedicated LPN-based authentication protocols are currently not used for practical solutions because, due to their hardware costs, this is simply not possible. But even if it was, they would have a hard time competing with cipher-based schemes in real-world applications. This results from the fact that, as many engineers from industry told us, when it comes to selling cryptographic products, customers want security (or rather a sense of it) by the use of standardized components. And the best-standardized components in cryptography are, in fact, (block) ciphers (e.g., most prominently, in the form of the Data Encryption Standard (DES) and AES). Hence,

virtually everyone from industry we spoke to pointed out that, if resources allow for it, a generic cipher-based scheme using AES (or, for even more powerful devices and also depending on the deployment scenario, an asymmetric primitive like the Rivest-Shamir-Adleman cryptosystem (RSA)) will be preferred.

Unfortunately, one of the smallest currently know AES implementations by Feldhofer et al. [FWR05] still requires an area of about 3400 GE, which is well beyond the the limit of 2000 GE for ultra-constrained devices justified in Subsection 2.3.3. But while AES is generally not targeted at low-cost hardware, there exists in fact a standardized (in ISO/IEC 29192-2:2012) lightweight block cipher in the form of PRESENT [BKL+07], which we will now use to exemplify why, as claimed above, cipher-based authentication schemes are actually feasible in the context of lightweight cryptography. Section 3.4 will show that one of the main bottlenecks of LPN-based authentication protocols is their massive requirement of random numbers. In contrast, the prover (as compared to the more powerful verifier) does not need to create any random numbers at all in the case of a cipher-based authentication scheme.[12]

Similarly, also the communication complexity is much lower (some LPN-based protocols need up to hundreds of thousands of bits per authentication; see Section 3.4), as in the case of PRESENT, which has a block length of 64 bits and a key length of 80 bits, a challenge consisting of two blocks, i.e., 128 bits, should be sufficient to provide the maximum possible security (otherwise, the pseudorandomness property of the underlying block cipher would be violated). A corresponding bandwidth is available on even the least powerful devices (see Subsection 2.3.1). The remaining conditions on low-cost RFID tags as outlined in Chapter 2 are satisfied by a PRESENT-based authentication scheme as well, according to the following numbers taken from [RPLP08]. Concretely, a serialized implementation of PRESENT requires an area of about 1080 GE and 563 clock cycles to process one block, both of which are well below the previously discussed limits of 2000 GE and 15 000 clock cycles, respectively. Finally, also the limited power budget of a low-cost RFID tag (see Subsection 2.3.4) is respected, for by using the UMCL18G212T3 (0.18 μm, 1.8 V) standard cell library (see, e.g., Section 8.5 for further details), it is possible to reach as low as 2.52 μW given a clock speed of 100 kHz. In summary, the example of PRESENT has shown that it is in fact possible for an authentication scheme to satisfy the conditions of low-cost hardware as outlined in Chapter 2 and still provide the required level of security. After all, PRESENT remains unbroken so far, even without claiming provable security as several LPN-based protocols have done in the past (see Section 3.4), many of which were then shown to be insecure shortly after by considering slightly different but nonetheless plausible attack scenarios.

As a final remark w.r.t. cipher-based authentication, we would like to point out that, though we used PRESENT in the above example due to its popularity and standardization,

[12] If the tag should feature an RNG anyhow, PRESENT would even allow for mutual authentication at practically no additional costs as compared to the LPN-based protocols assessed in Section 3.4.

there are several other, similarly feasible primitives. Among these is the block cipher PRINCE [BCG+12], which is characterized as a "low-latency block cipher" by its authors and targets "applications for which a low-latency encryption and instant response time is highly desirable, such as instant authentication". Furthermore, not only block ciphers but also lightweight stream ciphers should be considered as potential building blocks for authentication protocols. Consequently, in Chapter 9, we will conclude this thesis, inter alia, with a suggestion for a privacy-preserving lightweight authentication scheme based on our new stream cipher Lizard introduced in Chapter 8.

3.4 LPN-based Authentication Protocols

The most prominent non-proprietary approach for designing dedicated lightweight authentication protocols, which do not use existing ciphers as a building block, are LPN-based schemes. This branch of research was initiated by HB [HB01] and HB$^+$ [JW05] (in 2001 and 2005, respectively), which became the prototypes for a whole family of protocols that ground their security on the hardness of the learning parity with noise (LPN) problem (or variants of it). In a nutshell, this problem, which is known to be NP-hard[13] [BMvT78], can be defined as follows.

Definition 3.1: Learning Parity with Noise (LPN) Problem

Let $n \in \mathbb{N}$ and $\eta \in (0, 0.5)$ be public parameters. Then the corresponding *learning parity with noise (LPN) problem* can be defined as the following learning problem. At the beginning of the corresponding *oracle game*, the oracle chooses uniformly and at random a secret vector $x \in_U \{0,1\}^l$. The learner then poses a number of (empty) oracle queries, each of which is answered (independently from the other queries) by the oracle as follows:

1. The oracle chooses a random *noise bit* ν according to the Bernoulli distribution (i.e., $\Pr[\nu = 1] = \eta$ and $\Pr[\nu = 0] = 1 - \eta$).
2. The oracle chooses uniformly and at random vector $a \in_U \{0,1\}^l$.
3. The oracle answers with the tuple $(a, \langle a, x\rangle \oplus \nu) \in \{0,1\}^l \times \{0,1\}$, where $\langle a, x\rangle$ denotes the scalar multiplication of the vectors a and x.

The goal of the learner is to recover the secret vector x.

As pointed out above, most LPN-based authentication schemes are variants of the

[13] Note that (even under the assumption P $\neq$ NP) the NP-hardness of a problem does not automatically imply the security of corresponding cryptographic schemes. In, particular, though being NP-hard in general, there might still be large subsets of respective problem instances which can be solved efficiently.

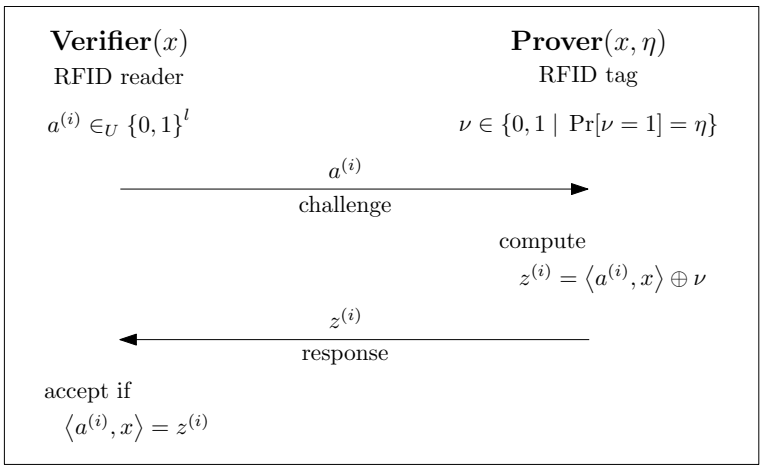

Figure 3.2: Round i of the HB protocol. (cf. [HB01])

HB protocol suggested by Hopper and Blum [HB01] or its extension HB$^+$ by Juels and Weis [JW05]. Therefore, as LPN-based schemes are not a main topic of this thesis, we will only describe these two protocols here in further detail and refer the reader to the respective original publications (e.g., summarized in [AHM14]) for the follow-up schemes.

The HB protocol [HB01] was originally developed to be used by humans and with this aim was designed to be very simple. Both the reader (verifier) and the tag (prover) share a secret $x \in \{0,1\}^l$. The protocol is composed of a number r of rounds that are conceptually all the same. At the beginning of round $1 \leq i \leq r$, the verifier chooses a random challenge $a^{(i)} \in \{0,1\}^l$ and sends it to the prover, who replies with $z^{(i)} = \langle a^{(i)}, x \rangle \oplus \nu$, where $\nu \in \{0,1\}$ represents a biased random noise bit satisfying $\Pr[\nu = 1] = \eta$ for a fixed probability $\eta \in (0, 0.5)$. Then, the reader verifies whether the received bit $z^{(i)}$ is equal to $\langle a^{(i)}, x \rangle$. If this is the case, the response is called correct and otherwise incorrect. Figure 3.2 depicts one such round of the HB protocol. The whole authentication is considered successful by the verifier if less than $\eta \cdot r$ wrong answers were given by the prover. The security of HB against *passive attacks*, in which an attacker is only able to eavesdrop on the communication between a valid reader and a valid tag, relies on the LPN problem with the public parameters l and ν (see, e.g., [LF06]).

Unfortunately, it quickly turned out that the original HB protocol is susceptible to active attacks in which a malicious reader tries to extract the tag's secret by adaptively choosing non-random challenges (see, e.g., [JW05]). In reaction, Juels and Weis suggested the augmented version HB$^+$ [JW05] in 2005, explicitly aiming for usage in the RFID context. In extension to the HB scheme, the tag and the reader now share an additional secret $y \in \{0,1\}^l$. At the beginning of round $1 \leq i \leq r$, the tag generates a random blinding factor $b^{(i)} \in \{0,1\}^l$ and sends it to the reader. Afterwards, similar to the HB protocol, the reader generates a challenge $a^{(i)} \in \{0,1\}^l$ and sends it to the tag.

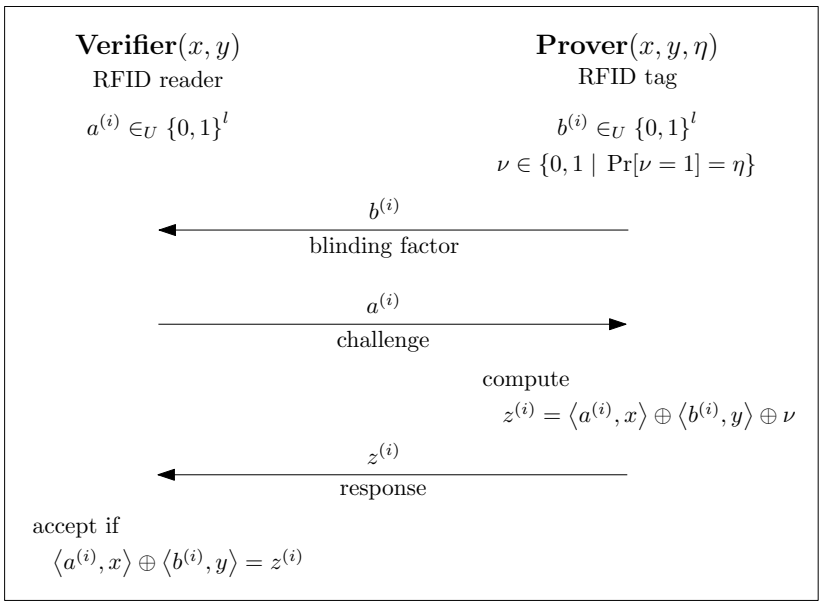

Figure 3.3: Round i of the HB$^+$ protocol. (cf. [JW05])

Then, the tag computes $z^{(i)} = \langle a^{(i)}, x \rangle \oplus \langle b^{(i)}, x \rangle \oplus \nu$, where ν is a randomly chosen noise bit satisfying $\Pr[\nu = 1] = \eta$, and responds with it to the reader for verification. Figure 3.3 depicts one such round of HB$^+$. Like for the original HB protocol, the whole authentication is successful if less than $\eta \cdot r$ wrong answers were given by the prover.

It is easy to see that, besides the parameters l and ν, also the number r of protocol rounds plays an important role for HB-type authentication shemes as each additional round increases the confidence of the verifier. If the noise probability η is chosen too close to 0.5, then a huge number r of rounds is required in order to make the protocol *reliable* (i.e., valid provers shall not be rejected). At the same time, if η is close to 0, then for obtaining the necessary level of security, extremely large key lengths for x and y are inevitable (as less noise makes it easier for an attacker to learn the shared secrets). Hence, an appropriate tradeoff needs to be found, which is specified by the choice of η.

However, besides security and reliability considerations, there are also practical aspects that impact reasonable choices for η. Usually, RNGs (see Subsection 2.3.8) are assumed to produce uniformly distributed random bits. In this case, it is much easier to implement instantiations where $\eta = 2^{-j}$, $j \in \mathbb{N}$, as j uniformly distributed bits are sufficient for the generation of one noise bit ν. For other values of η, many more uniformly random bits may be needed to realize a correspondingly biased random bit generator on top of those.

As pointed out above, there is in fact a multitude of LPN-based authentication protocol suggestions. In our paper [AHM14], we provide an overview over 18 of the most relevant of them, along with an in-depth analysis of possible parameter choices (for the noise

probability η, the required key size and the number r of rounds necessary) and the implied hardware costs. In particular, we come to the staggering conclusion there is currently not a single unbroken LPN-based authentication scheme suitable for ultra-constrained devices. Consequently, apart from the aforementioned protoypes HB and HB$^+$, we will not go into further detail w.r.t. other protocols of this type in this thesis. Instead, in the following Subsection 3.4.1, we will refer to a generalized structure of such protocols in order to explain the corresponding hardware efficiency problems. The respective insights will then help us to choose a more suitable approach in Section 3.5 and Chapter 4, where our $(n, k, L)^{\langle 80 \rangle}$-protocol for lightweight authentication will be introduced. Before that, however, we will also briefly assess some other authentication protocols in Subsection 3.4.2, which are not based on the original LPN problem but on new variants of it. Though we will see that, like for the classical HB-type schemes, these protocols are also not suitable for ultra-constrained hardware, we wanted to include them for the sake of completeness.

3.4.1 Cost Drivers of LPN-based Authentication Protocols

In Chapter 2, we have established a concrete notion of the term *lightweight* in the RFID context by providing actual hardware limits for low-cost tags. At the example of LPN-based authentication protocols, we will now demonstrate that even schemes which might look temptingly lightweight at first sight (due to their use of extremely simple operations; cf., e.g., Fig. 3.3) can eventually turn out to be very costly. In order to evaluate whether a protocol really complies to all of the respective hardware limits, it is necessary to first identify the major cost drivers of such schemes. For LPN-based authentication protocols, these are the symmetric key(s) along with challenges, blinding factors, and noise bits. In the following, we will discuss for each of the respective protocol properties how it is linked to the hardware capabilities of ultra-constrained RFID tags in the \$0.05 to \$0.10 cost range as discussed in Chapter 2.

Symmetric keys. All allegedly lightweight LPN-based authentication protocols use symmetric keys[14]. Consequently, the full shared secret must be permanently available on the (passively powered) tag, thus implying the need for some kind of key storage. Depending on the deployment scenario, multiple (e.g., batches of) RFID tags might share a single key or, in other cases, tag-individual secrets may be required. Closely related, but even more restrictive w.r.t. key storage options, is a potential need to set or change the secret key of a tag that is already in the field, as compared to irreversibly fixing the key once at production time. We refer the reader to Subsection 2.3.9 and Subsection 2.3.10 for an explanation of corresponding key storage options (e.g., EEPROMs,

[14]For the sake of simplicity, in this subsection, the term *key* will always be used to refer to the shared secret's unique representation as a binary vector in the corresponding scheme, irrespective of potential blow-up measures like, e.g., the use of Toeplitz matrices. In particular, the key size lower bounds the size of the individual key storage required on each tag.

fuses/antifuses, key-dependent masks) for ultra-constrained devices and to Subsection 3.2.1 and Subsection 3.2.2 for a discussion of the security hazards connected with group keys.

These general preconditions will now be compared to the requirements imposed by how symmetric keys are chosen and used in LPN-based authentication protocols. Unfortunately, in addition to considerations such as whether group keys can be used or not, key storage options are here further restricted by the large key size common to these schemes. In [KPC+11], key sizes for various LPN-based protocols are specified on the basis of a parameter l. For example, the key size of the HB$^+$ protocol suggested by Juels and Weis in [JW05] is given as $2l$ along with $l = 500$ described as a "typical parameter".[15] Please note that the resulting key length of 1000 bits is even at the lower end of the protocols summarized in [KPC+11] (which range from l bits for the original HB protocol [HB01], over $4.2 \cdot l$ bits for AUTH [KPC+11], up to $80 \cdot l = 40\,000$ bits for a MITM-secure protocol also suggested in [KPC+11]; see Subsection 3.4.2 for further details). However, e.g., due to area requirements, already for 1000 bits it seems highly questionable whether fuses/antifuses can still be considered a feasible option for storing the secret key on a low-cost RFID tag. Moreover, as pointed out previously, similar to (or even worse than) key-dependent masks, fuses/antifuses fail to provide substantial physical security. Ultimately, it depends on the deployment scenario whether this is an actual threat, hence requiring the use of, e.g., EEPROMs instead. Bring to mind, however, that in the context of ultra-constrained RFID devices, EEPROMs typically do not allow for storing more than 2048 bits. As a result, it must be suspected that many of the LPN-based protocols are already precluded by their key sizes from practical application on RFID tags in the \$0.05 to \$0.10 range.

Challenges, blinding factors, and noise bits. Another property characteristic of LPN-based authentication protocols is their heavy use of challenges and blinding[16] factors. As most LPN-based schemes represent variants of HB$^+$ (see Fig. 3.3), the following three phases per round can usually be identified:

(1) The prover creates a vector of random bits, the so-called *blinding factor*, which is then transmitted to the verifier.

(2) Just alike, the verifier now also creates a random bit vector (i.e., the *challenge*) and sends it to the tag.

(3) Depending on the specific protocol, the prover deterministically computes some 1-bit value based on the blinding factor in (1), the challenge in (2), as well as the

[15] In the case of HB$^+$, the parameter l corresponds to the respective sizes of the two secrets $x, y \in \{0,1\}^l$ and to the size of the challenge $a \in \{0,1\}^l$ and the size of the blinding factor $b \in \{0,1\}^l$ (see Fig. 3.3).

[16] Note that, apart from the aforementioned usage in the HB$^+$ protocol (i.e., to avoid the extraction of a tag's secret through adaptively chosen challenges by an active attacker), *blinding* techniques can actually serve various purposes in cryptography. For example, they can also be used to thwart side-channel attacks by eliminating input-characteristic function evaluation times (or respective power consumptions).

shared key. Finally, he needs to produce one more random bit, which, on contrast to the aforementioned challenge and blinding vectors, is not based on the uniform but some other, fixed distribution. Adding this so-called *noise bit* to the 1-bit value yielded by the previous operation is crucial to the security of LPN-based protocols as explained previously. The resulting bit is then sent to the verifier, who will check whether it is correct or not. As, in the latter case, either the prover was illegitimate in the first place or the noise bit was 1, multiple protocol rounds are necessary to ensure that, with high probability, bad provers will eventually be rejected while honest ones are accepted.

In the following paragraph, we will denote the number of protocol rounds per authentication run by r and, for reasons of simplicity, assume that the blinding vector in step (1) as well as the challenge vector in step (2) are both of length l (as done in the original HB$^+$ paper [JW05] and popular follow-up works like [KPC$^+$11]).

Apparently, the general protocol structure we just outlined makes heavy use of at least two hardware resources previously identified as potential bottlenecks for low-cost RFID tags: the transmission bandwidth (see Subsection 2.3.1) and the generation of random numbers (see Subsection 2.3.8). Concretely, in each round of the above archetypical scheme, the communication complexity amounts to $2l + 1$ and the prover (i.e., the ultra-constrained RFID device) needs to obtain l uniformly random bits and one differently distributed random bit from his RNG. Hence, a single authentication procedure consisting of r rounds has a communication complexity of at approximately $r \cdot 2l$ bits and requires at least $r \cdot l$ random bits on the prover's side. As in the previous paragraph about key sizes, let us exemplify the actual consequences of these complexities for LPN-based protocols using parameter choices described as "typical" in [KPC$^+$11]: $l = 500$ and $r = 250$. Moreover, as justified in Subsection 2.3.2, let us consider 150 ms to be the maximum time available for a complete authentication. As a result, at least $250 \cdot (2 \cdot 500) = 250\,000$ bits would need to be transmitted within 150 ms, corresponding to a vastly implausible transmission rate of $250\,000/0.15$ bit/s ≈ 1.66 Mbit/s (as compared to actual values between 10 kbit/s and 200 kbit/s as given in Chapter 2). Similarly far from reality is the idea that an RFID tag whose production costs are in the \$0.05–\$0.10 range could actually feature an RNG delivering as much as $250 \cdot 500 = 125\,000$ uniformly distributed random bits within just 150 ms. Apart from the apparent bottlenecks transmission bandwidth and generation of random numbers, the generalizing description of LPN-based protocol at the beginning of this paragraph contains a third aspect worth investigating. Concretely, depending on the involved operations, the first computation in step (3) can easily turn out to consume (possibly too) many clock cycles, especially in view of the fact that several operands of length 500 bits or more are involved. As this is highly protocol-specific and implementation-dependent (e.g., parallel vs. serial processing in step (3) of HB$^+$) though, we again refer to our publication [AHM14], which parts of this section are based on and where a large number of LPN-based authentication protocols are evaluated in detail.

3.4.2 Protocols based on Variants of the LPN Problem

For the sake of completeness, we conclude this section by addressing some dedicated protocols which deviate from the common 'HB-type' design approach in that they are not based on the classical LPN problem but on (new) variants of it. As a consequence, they also use rather different operations than the simple scalar multiplications which, e.g., HB and HB$^+$ are based on. Please note, however, that we will not go into the corresponding details, but instead explain on a higher level, why these authentication schemes also do not conform to the strict hardware requirements imposed by ultra-constrained devices.

AUTH, MAC$_1$, and MAC$_2$

In reaction to the continuing security problems of classical HB-type LPN-based authentication protocols, many of which were eventually broken by means of MITM attacks (see, again, [AHM14]), Kiltz et al. introduced a two-round authentication protocol called AUTH in 2011 [KPC$^+$11], which, based on a new LPN variant called *subset LPN*, provably provides active (but not MITM) security.[17] In addition, two provably MITM-secure protocols, called MAC$_1$ and MAC$_2$, are also suggested in the same paper. The authors of [KPC$^+$11] provide an assessment of the corresponding complexities (e.g., for communication, computation, and key size), which, however, partly use the Θ-notation. When compared to the constraints of low-cost RFID tags as justified in Chapter 2, the respective numbers, summarized in Table 3.1, immediately show that the provable security comes at a high price and that these protocols are, without doubt, not suited for such devices.

For AUTH, MAC$_1$ and MAC$_2$, the parameter l controls the challenge as well as the key size (like in Subsection 3.4.1) and n is similar to the number r of rounds for HB-type protocols (see Footnote 17 for further details). The tradeoff parameter c, $1 \leq c \leq n$, between key size and communication complexity is due to Gilbert et al. [GRS08] and λ is referred to as a "security parameter". Please note in particular that the term PIP in Table 3.1 subsumes the additional computational complexity of evaluating a certain pairwise independent permutation, which, according to the designers, takes $\Theta(m^2)$ time,

[17] The number of rounds here cannot be compared with classical HB-type protocols, as the authors of [KPC$^+$11] follow a completely different design approach. In a nutshell, the term *2-round* in [KPC$^+$11] actually refers to the verifier sending a (single) challenge and the prover sending a (single) response, where this whole process is performed only once (as compared to many times in classical HB-type protocols). Note, however, that also, e.g., HB (cf. Fig. 3.2) could be used in this way by having the prover send a random matrix of size $r \cdot l$ only once, instead of sending r random challenge vectors of size l over the course of r rounds. Correspondingly, the prover would answer with a single vector of size r, instead of sending r individual response bits during r rounds. But, clearly, from a hardware perspective, this would be even more challenging for an ultra-constrained RFID tag due to the additional volatile storage required for buffering the incoming matrix of size $r \cdot l$. In AUTH, however, the size of the single challenge is in fact only $2 \cdot l$. Nevertheless, the communication complexity is still huge because the single response consists of a vector of length n and a matrix of size $n \cdot l$ (or, depending on the implementation, even $n \cdot 2l$; see [KPC$^+$11]), where n equals the size of r in HB-type protocols.

Table 3.1: Complexities of the 2-round authentication protocols suggested in [KPC+11]. The designers give $l = 500$, $n = 250$, and $\lambda = 80$ as "typical parameters". c is a "tradeoff parameter" with $1 \leq c \leq n$.

Construction	Communication	Computation	Key size
AUTH	$l \cdot n \cdot 2.1/c$	$\Theta(l \cdot r)$	$l \cdot 4.2 \cdot c$
MAC_1	$l \cdot n \cdot 2.1/c$	$\Theta(l \cdot r) + \text{PIP}$	$l \cdot 12.6 \cdot c$
MAC_2	$l \cdot n \cdot 1.1/c$	$\Theta(l \cdot r) + \text{PIP}$	$l \cdot \lambda \cdot c$

where $m \approx 1200$ for MAC_1 and $m \approx 600$ for MAC_2. Clearly, the resulting numbers of additionally required clock cycles are well beyond the limits of what we justified in Subsection 2.3.6 as feasible. In addition, for AUTH as well as for MAC_1 and MAC_2, any choice of c in Table 3.1 will either result in a key size (cf. Subsection 3.4.1) or in a communication complexity (cf. Subsection 2.3.1) definitely not feasible in the context of low-cost RFID tags.

Lapin

Probably due to the efficiency problems of AUTH, MAC_1, and MAC_2, a new authentication scheme called Lapin [HKL+12] was suggested in 2012. It is based on the results in [KPC+11] and builds its security on what the designers call the *Ring LPN* problem. As compared to, e.g., AUTH, the communication complexity of Lapin (given as 1300 bits in [HKL+12]) is now actually feasible for RFID tags in the \$0.05 to \$0.10 range. However, the authors themselves state that they are "targeting lightweight tags that are equipped with (small) CPUs" as compared to "ultra constrained tokens (such as RFIDs in the price range of few cents targeting the EPC market". Moreover, the protocol was strongly criticized in [BL13], where the authors come to the conclusion that Lapin is even less efficient than AES, at the same time providing a lower level of security. A suggestion for an FPGA implementation of Lapin was made in [GLS14], which requires 36 kB of buffer random access memory (BRAM) and, hence, makes the protocol clearly infeasible when transferred to low-cost ASICs. Taking into account all these arguments, we will not discuss this scheme in further detail.

3.5 The Cryptographic Power of Random Selection

The *principle of random selection* underlies, e.g., the CKK-protocols of Cichoń, Klonowski, and Kutyłowski [CKK08] as well as the F_f-protocols in [BKM+09] and the Linear Protocols in [KS09]. It can be described as follows.

Suppose that the verifier Alice and the prover Bob run a challenge-response authentication protocol which uses a lightweight symmetric encryption operation $E : \{0,1\}^n \times \mathcal{K} \longrightarrow$

$\{0,1\}^m$ of block length n, where $\mathcal{K}$ denotes an appropriate key space.[18] Suppose further that E is weak in the sense that a passive adversary can efficiently compute the secret key $\kappa \in \mathcal{K}$ from samples of the form $(u, E_\kappa(u))$. This is obviously the case if E is linear.

Random selection denotes a method for compensating the weakness of E by using the following mode of operation. Instead of holding a single $\kappa \in \mathcal{K}$, Alice and Bob share a collection $\kappa^{(1)}, \ldots, \kappa^{(L)}$ of keys from $\mathcal{K}$ as their common secret information, where $L > 1$ is a small constant. Upon receiving a challenge $u \in \{0,1\}^n$ from Alice, Bob chooses a random index $l \in \{1, \ldots, L\}$ and outputs the response $y = E_{\kappa^{(l)}}(u)$. The verification of y with respect to u can be efficiently done by computing $E^{-1}_{\kappa^{(l)}}(y)$ for all $l = 1, \ldots, L$.

Note that the protocols introduced in [CKK08], [BKM+09], and [KS09] are based on random selection of GF(2)-linear functions. This is motivated by the fact that GF(2)-linear functions can be implemented efficiently in hardware and have desirable pseudorandomness properties with respect to a wide range of important statistical tests.

In the following Subsection 3.5.1, we will now recall the definition of the $(n,k,L)^{++}$-protocol suggested by Krause and Stegemann in 2009 [KS09]. The corresponding general security conjecture for (n,k,L)-type protocols will then be outlined in Subsection 3.5.2. Finally, in Chapter 4, we will tweak the $(n,k,L)^{++}$-protocol in order to make it suitable for ultra-constrained RFIDs.

3.5.1 The $(n,k,L)^{++}$-Protocol

The $(n,k,L)^{++}$-protocol [KS09] is a one-round challenge-response authentication protocol, whose symmetric key consists of a small number L of injective GF(2)-linear functions $F_1, \ldots, F_L : \mathrm{GF}(2)^n \longrightarrow \mathrm{GF}(2)^{n+k}$. Based on a first analysis of the underlying security assumption, the following parameter sizes were suggested by the designers: $n = 256$, $k = 64$, $L = 5$. Figure 3.4 depicts an instance of the $(n,k,L)^{++}$-protocol for a verifier Alice (RFID reader) and a prover Bob (RFID tag).

The authentication process is initiated by Alice, who chooses uniformly and at random a challenge $a \in \mathrm{GF}(2)^{\frac{n}{2}}$, $a \neq \mathbf{0}$, and sends it to the prover. Likewise, the prover chooses a random nonce $b \in \mathrm{GF}(2)^{\frac{n}{2}}$, $b \neq \mathbf{0}$, of the same length, randomly picks one of the L secret linear functions $F_1, \ldots, F_L$, and responds with $w = F_l\,(f\,(a,b))$. For the nonlinear, bijective *connection function* $f : \mathrm{GF}(2^{\frac{n}{2}})^* \times \mathrm{GF}(2^{\frac{n}{2}})^* \longrightarrow \mathrm{GF}(2^{\frac{n}{2}})^* \times \mathrm{GF}(2^{\frac{n}{2}})^*$, where $\mathrm{GF}(2^{\frac{n}{2}})^*$ denotes $\mathrm{GF}(2^{\frac{n}{2}}) \setminus \{0\}$, we identify the vector space $\mathrm{GF}(2)^{\frac{n}{2}}$ with the finite field $\mathrm{GF}(2^{\frac{n}{2}})$. f is defined by $f(a,b) = (ab, ab^3)$ for all $a, b \in \mathrm{GF}(2^{\frac{n}{2}})^*$. It is included for thwarting a certain class of MITM attacks (see Subsection 4.3.2). In order to verify the prover's response, the reader Alice first checks whether w belongs to one of the L n-dimensional subspaces $V_1, \ldots, V_L$ of $\mathrm{GF}(2)^{n+k}$ which are the images of the corresponding injective GF(2)-linear functions $F_1, \ldots, F_L$. Given that $w \in V_l$ holds for

[18] Note that in this section and in the following Chapter 4, we will use $\mathcal{K}$ for key spaces and κ for keys in order to avoid confusion, with, e.g., the parameter k of the discussed (n,k,L)-type authentication protocols.

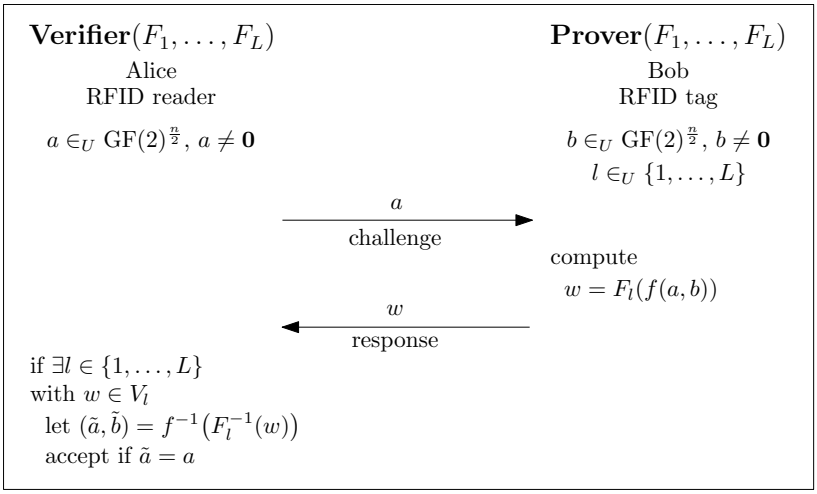

Figure 3.4: An instance of the $(n, k, L)^{++}$-protocol. (cf. [KS09])

some $l \in \{1, \ldots, L\}$, Alice subsequently computes $(\tilde{a}, \tilde{b}) = f^{-1}(F_l^{-1}(w))$. Finally, if $\tilde{a}$ equals the initial challenge a, Alice will accept the prover's valid response.

It is easy to see that the $(n, k, L)^{++}$-protocol suffers from a large key length, similar to (or even worse than) the LPN-based suggestions discussed in the previous section, as each of the L secret GF(2)-linear functions $F_1, \ldots, F_L : \{0,1\}^n \longrightarrow \{0,1\}^{n+k}$ can be expressed as a distinct $((n+k) \times n)$-matrix over GF (2). Thus, in total, $((n+k) \cdot n) \cdot L$ bits would need to be stored permanently, which is clearly infeasible for the suggested parameters $n = 256$, $k = 64$, and $L = 5$. Moreover, the nonlinear connection function f induces a large computational overhead in the form of several multiplications over $\mathrm{GF}(2^{\frac{n}{2}})^*$. Our new $(n, k, L)^{\langle 80 \rangle}$-protocol (introduced in the following Chapter 4) addresses these issues, e.g., by using an LFSR-based self-shrinking generator [MS94] to derive the specifications of the functions $F_1, \ldots, F_L$ on the basis of a now feasible 80-bit key.

Before we proceed with an overview over the general *security* properties and assumptions for (n, k, L)-type protocols in the following Subsection 3.5.2, let us briefly treat another aspect of $(n, k, L)^{++}$-based authentication, which has neither been discussed in the original publication by Krause and Stegemann [KS09], nor in follow-up works like [Ham10], [KH11], and [AHM14]: *privacy*. More precisely, privacy is not only lost if an attacker manages to break the scheme by recovering the secret key, but can already be compromised if an attacker is able to track an RFID tag that always gives the same response for the same challenge (see Section 3.1 for an example together with a simple countermeasure for block ciphers). Fortunately, the $(n, k, L)^{++}$-protocol provides privacy preservation in this scenario. This is due to the fact that the random nonce $b \in \mathrm{GF}(2)^{\frac{n}{2}}$, chosen by the prover (and not even made public), does not only protect against active extraction of the secret functions $F_1, \ldots, F_L$ through adaptively chosen

challenges, but also thwarts tracking based on repeatedly sending a constant challenge.[19] This important security property will carry over from the $(n, k, L)^{++}$-protocol to our new $(n, k, L)^{\langle 80 \rangle}$-protocol introduced in Chapter 4.

3.5.2 The Security of (n, k, L)-type Protocols

The security of (n, k, L)-type authentication protocols is based on the conjectured hardness of the learning problem *RandomSelect*(L, n, ρ), which is studied in our paper [KH11][20] and reads as follows: learn GF(2)-linear functions $f_1, \ldots, f_L : \{0,1\}^n \longrightarrow \{0,1\}^\rho$ from values $(u, f_l(u))$, where the secret indices $l \in \{1, \ldots, L\}$ and the inputs $u \in \{0,1\}^n$ are randomly chosen by an oracle.

The running time of our corresponding learning algorithm is dominated by the effort for solving a full-rank system of linear equations of $O(n^L)$ unknowns over GF(2^ρ). Note, however, that in contrast to the protocol parameters n and L, an attacker against (n, k, L)-type protocols has in fact (some) control over the size of ρ, due to the GF(2)-linearity of the secret functions $F_1, \ldots, F_L$. In Appendix B of [KH11], we sketch (based on a corresponding previous result in [KS09]) how an efficient algorithm for *RandomSelect*(L, n, ρ) can be used to attack the $(n, k, L)^+$-protocol described[21] by Krause and Stegemann in [KS09], if ρ is chosen such that $\rho \geq \lceil \log_2(L) \rceil$. The reason for this condition will become clearer during the outline of our learning algorithm given below.

Note that, to the best of our knowledge, no faster way of solving *RandomSelect*(L, n, ρ) has been found, yet, and that trivial approaches lead to a running time exponential in n. Thus, our learning algorithm, which conducts an algebraic attack in the spirit of [CKPS00], hints at how the parameters n and L should be chosen for protocols using random selection of linear functions (such as the $(n, k, L)^{++}$-protocol in [KS09] and the $(n, k, L)^{\langle 80 \rangle}$-protocol in Chapter 4) in order to achieve an appropriate level of cryptographic security. For example, when choosing $n = 128$ and $L = 8$, solving *RandomSelect*(L, n, ρ) by means of our approach implies solving a system of around

[19] Note that the principle of random selection itself already provides some protection against tracking based on constant challenges posed by malicious readers. This protection, however, is quite weak, as the set of L possible responses for some fixed challenge can usually be obtained rather quickly, which would then allow an attacker to proceed as usual.

[20] The paper [KH11] (SAC 2011) is largely based on the diploma thesis [Ham10] of the author of this PhD thesis. Therefore, we will only provide a brief summary of the corresponding results here in order to avoid any issues w.r.t. duplicate content. Giving this summary, however, is necessary, because it facilitates to understand the parameter choices and security claims for our new $(n, k, L)^{\langle 80 \rangle}$-protocol, which will be introduced in Chapter 4. Please also note that, in contrast to [KH11], we will write *RandomSelect*(L, n, ρ) instead of *RandomSelect*(L, n, a) to avoid potential confusions of the challenge a, commonly used in descriptions of (n, k, L)-type protocols, with the parameter a (now ρ) of the learning problem.

[21] We use the term *described* here, because the $(n, k, L)^+$-protocol, which corresponds to the $(n, k, L)^{++}$-protocol without the connection function f (cf. Subsection 3.5.1), was never suggested for practical application. Instead, it only served to demonstrate the necessity of such a connection function, as, without it, (n, k, L)-type protocols are vulnerable to so-called (x, y)-*equality attacks* (see Subsection 4.3.2).

$\binom{128}{8} \approx 2^{40}$ unknowns over $\mathrm{GF}(2)^\rho$ (with $\rho \geq 3$), which should be sufficiently difficult for most scenarios. In the context of lightweight cryptography, where 80 bits is a common key size, the choice $n = 64$ and $L = 8$ (resulting in about 2^{32} unknowns) is already adequate, based on the common notion that solving such a system has time complexity about $(2^{32})^3$.

Before we go into the details of our learning algorithm for $\mathit{RandomSelect}(L, n, \rho)$, we would like to point out that, besides n and L, the third protocol parameter k also plays an important role for (n, k, L)-type authentication schemes. More precisely, it must ensure that the following probabilities are sufficiently small:

1.) the probability that one of the functions F_l, $l = 1, \ldots, L$, is not injective (if they are chosen randomly from the set of all possible linear functions from $\mathrm{GF}(2)^n$ to $\mathrm{GF}(2)^{n+k}$, as in the case of our new $(n, k, L)^{\langle 80 \rangle}$-protocol introduced in Chapter 4),

2.) the probability that a random vector $w \in \mathrm{GF}(2)^{n+k}$ falls into $\bigcup_{l=1}^{L} V_l$,

3.) the probability that a random vector $w \in V_l$ falls into $V_l \cap V_k$ for some $k \neq l$,

4.) the probability that there is a pair of secret subspaces V_l, V_k, $1 \leq l \neq k \leq L$, such that $\dim(V_l \oplus V_k) < n + k$.

For an estimation of the respective probabilities (for randomly chosen linear functions) and a detailed explanation of corresponding implications, we refer the reader to [KS09] and [KH11]. We would like to briefly point out, however, that, e.g., if a secret function F_l should not be injective and, hence, imply a smaller subspace V_l of dimension, e.g., $n - 1$ instead of n, this does not mean an immediate break of the protocol's security. Instead, it will just slightly increase the load on the verifier's side, who now has to check whether a set of possible preimages under F_l actually contains the valid one (i.e., the one corresponding to his initial challenge). As the verifier is commonly assumed to be significantly more powerful than the prover, this can be considered tolerable. A similar argumentation holds for the case that a response $w \in V_l$ should fall into $V_l \cap V_k$ for some $k \neq l$. There, assuming that F_l and F_k are injective, the verifier would have to check whether either the preimage to w under F_l or the preimage to w under F_k is correct. The resulting increase of computational load on the verifier's side is negligible. Nevertheless, it is clear that if, in the case of a practical protocol instantiation, the respective probabilities should turn out to be way too large, this would in fact impair security.

A Learning Algorithm for $\mathit{RandomSelect}(L, n, \rho)$

Our approach for solving $\mathit{RandomSelect}(L, n, \rho)$ can be sketched as follows (see [KH11] for further details). We begin by representing the secret L linear basis functions $f_1, \ldots, f_L : \{0,1\}^n \longrightarrow \{0,1\}^\rho$ as an (unknown) assignment A to a collection $X = (x_i^l)_{i=1,\ldots,n, l=1,\ldots,L}$ of variables taking values from the field $K = \mathrm{GF}(2^\rho)$. In consequence, each example

$(u, f_l(u))$ induces a degree-L equation of a certain type in the X-variables, which allows for reducing the learning problem $\mathit{RandomSelect}(L, n, \rho)$ to the problem of solving a system of degree-L equations over K. While, in general, the latter problem is known to be NP-hard, we show in [KH11] an efficient way to solve this special kind of systems.

One specific problem of our approach is that, due to inherent symmetries of the degree-L equations, we can never reach a system which has full linear rank with respect to the corresponding monomials. In fact, this is the main difference between our learning algorithm and the well-known algebraic attack approaches for cryptanalyzing LFSR-based keystream generators (see, e.g., [CM03], [AK03], [Cou03], [MPC04]).

We circumvent this problem by identifying an appropriate set $T(n, L)$ of basis polynomials of degree at most L, which allow to express the degree-L equations as linear equations over $T(n, L)$. The choice of $T(n, L)$ is justified by Theorem 2 in [KH11], saying that if $|K| \geq L$, then the system of linear equations over $T(n, L)$ induced by all possible examples has full rank $|T(n, L)|$. Note that according to Theorem 1 in [KH11], this is not true if $|K| < L$, thus imposing a lower bound on an attacker's choice of ρ. Our experiments, which are presented in Section 5 of [KH11], indicate that if $|K| \geq L$, then with probability close to one, the number of examples necessary for obtaining a full rank system over $T(n, L)$ exceeds $|T(n, L)|$ only by a small constant factor. This implies that the effort for computing the unique *weak* solution $t(A) = (t_*(A))_{t_* \in T(n,L)}$ corresponding to the *strong* solution A equals the time for solving a system of about $|T(n, L)|$ linear equations (with $|T(n, L)|$ variables) over K, where $|T(n, L)| = \sum_{j=1}^{L} \binom{n}{j}(L - j + 1)$.

But in contrast to the algebraic attacks in [CM03], [AK03], [Cou03], and [MPC04], we still had to solve another nontrivial problem in [KH11], namely, to compute the *strong* solution A, which identifies the secret functions $f_1, \ldots, f_L$, from the unique weak solution $t(A)$. An efficient way to do this is described in Section 4 of the paper. The overall complexity of our approach is dominated by the effort for computing the weak solution.

As a final remark, we would like to point out that all our theoretical estimates in [KH11] were also backed-up through experimental evaluation using the computer algebra system Magma [BCP97]. We are thus convinced that they form a profound basis for the parameter choices made in the context of our new $(n, k, L)^{\langle 80 \rangle}$-protocol, which will be presented in the following Chapter 4.

3.6 Conclusion and Outlook

In this chapter, we have provided an introduction to authentication in general and its lightweight forms in particular. It started with an explanation of the different concepts *identification*, *authentication*, and *authorization*, which are all closely connected when using RFID tags in practice, but have separate aims each. In particular, our examples of real-world authentication failures (in the form of attacks against the VPN infrastructure and the ecUM contactless smart card infrastructure of the author's university) have

shown that, despite the sole appearance of the term *identification* in the acronym RFID, identification without proper authentication can easily lead to disaster. Apart from this general (and rather obvious) insight, the attacks and their underlying flaws also provided guidance w.r.t. various additional rules vital for the security of real-world authentication systems, such as the implementation of strong back-end security as well as proper random number generation and key management/protection.

In the second part of the chapter, we have described and evaluated (based on the hardware limitations of ultra-constrained devices as introduced in Chapter 2) the three most common approaches for lightweight authentication: (block) cipher-based protocols, LPN-based protocols, and protocols based on random selection of secret linear functions. Building on a protocol of the latter type, namely the $(n, k, L)^{++}$ protocol of Krause and Stegemann [KS09], we will now suggest a new lightweight authentication protocol actually feasible for ultra-constrained RFIDs in Chapter 4. In analogy to HB-type protocols and the LPN problem (cf. Section 3.4), its security will be based on the conjectured hardness of the *RandomSelect* problem as introduced in Subsection 3.5.2.

Trust, but verify.

Russian proverb

CHAPTER 4

The $(n, k, L)^{\langle 80 \rangle}$ Authentication Protocol

ABSTRACT

Lightweight authentication protocols based on random selection of secret linear functions were introduced as an alternative design paradigm besides the usage of lightweight block ciphers and the principle of adding biased noise. However, a comparatively large key length and the use of involved operations made a hardware-efficient implementation a challenging task.

In this chapter, we introduce the $(n, k, L)^{\langle 80 \rangle}$-protocol, a variant of linear authentication protocols which overcomes these problems, and analyze its security against all currently known, relevant passive and active attacks. Moreover, we present an implementation of our protocol for FPGAs and ASICs based on the hardware description language Verilog and discuss its efficiency w.r.t. the cost metrics described in Chapter 2. The respective numbers show that the $(n, k, L)^{\langle 80 \rangle}$-protocol is a viable alternative to existing solutions and is, for example, suited for the implementation on ultra-constrained RFID tags.

Declaration of Origin: This chapter is based on the paper *Hardware Efficient Authentication based on Random Selection* [AHK14], written together with Frederik Armknecht and Matthias Krause and presented at *Sicherheit 2014*.

4.1 Introduction

In previous works about the $(n, k, L)^{++}$-protocol [KS09] described in Subsection 3.5.1, two problems w.r.t. efficiency were left open for future research and prevented this type of protocol from being practically used so far: First, the large key length resulting from the need to specify the L secret linear functions. Second, certain operations deemed necessary in order to achieve MITM-security were still too demanding in hardware.

The $(n, k, L)^{\langle 80 \rangle}$-protocol introduced in this chapter aims at solving both of these problems. In particular, we are able to reduce the key length to a feasible size of 80 bits and show that the security reductions presented in [KS09] and [KH11] still apply to a large extent. Moreover, all operations used in the $(n, k, L)^{\langle 80 \rangle}$-protocol can be realized efficiently in hardware. In order to demonstrate this, we created an actual implementation for FPGAs and ASICs using the hardware description language Verilog. When compared to the hardware limits for ultra-constrained RFIDs described in Chapter 2, the corresponding results indicate that the new protocol is a viable alternative to prevalent block cipher-based constructions.

Important remark: Though being based on the paper *Hardware Efficient Authentication based on Random Selection* [AHK14], written together with Matthias Krause and Frederik Armknecht and presented at *Sicherheit 2014*, this chapter contains some important novelties. More precisely, [AHK14] was mainly focused on the fundamental approaches for making the $(n, k, L)^{++}$-protocol actually feasible. In particular, the hardware results presented there were given without an exact specification of the respective algorithm (e.g., there was no description of the feedback function used by the underlying generator) and, instead, basically served only to demonstrate the general feasibility of the applied techniques. In this chapter, we will add the missing details and, in fact, provide a concrete recommendation for a practical instantiation of the $(n, k, L)^{80}$-scheme. To this end, we now also give a reference implementation written in Verilog and corresponding test vectors for the parameter choices $n = 128$, $k = 32$, $L = 16$ and $n = 64$, $k = 32$, $L = 16$. Another important difference to the general description of the $(n, k, L)^{80}$-approach from [AHK14] is that the key size will now actually be 80 bits (as seemingly suggested by the protocol's name). In [AHK14], the key size depended on the parameter L and equaled $80 - \lceil \log_2(L) \rceil$, which, as we discovered in subsequent discussions with readers of our paper, apparently created some confusion. This has now been remedied in the concrete protocol suggestion presented in this chapter. As this specific suggestion, however, still fits into the general framework described in [AHK14], we did not want to deviate strongly from the name $(n, k, L)^{80}$. Instead, for our new instantiation, we speak of the $(n, k, L)^{\langle 80 \rangle}$-protocol, in particular, to signify that the key length is now actually 80 bits.

Structure of this chapter: In Section 4.2, we present the design rationale (including the relevant modifications w.r.t. the underlying $(n, k, L)^{++}$-protocol) and a detailed description of our new $(n, k, L)^{\langle 80 \rangle}$-protocol. Building on this, Section 4.3 then explains

how it is achieved that, despite the respective modifications, the security reductions given in [KS09] and [KH11] still apply to a large extent. In Section 4.4, we present the details of our hardware implementation for FPGAs and ASICs. Section 4.5 concludes the chapter and provides an outlook on potential future work as well as on the subsequent contents of this thesis. Test vectors and a reference implementation can be found in Appendix 4.A and Appendix 4.B, respectively.

4.2 Design Rationale and Specification

As pointed out in the above introduction and in Subsection 3.5.1, a crucial open problem of the original $(n, k, L)^{++}$-protocol was the excessively large key length. More precisely, as each of the L secret injective linear functions $F_1, \ldots, F_L : \{0,1\}^n \longrightarrow \{0,1\}^{n+k}$ can be expressed as a distinct $((n+k) \times n)$-matrix over GF (2), in total $((n+k) \cdot n) \cdot L$ bits would need to be stored permanently, which is clearly infeasible (w.r.t. ultra-constrained devices) for the parameters $n = 256$, $k = 64$, $L = 5$ suggested by the designers in [KS09].

Moreover, for such parameters sizes, even the 'simple' nonlinear connection function $f(a, b) = (ab, ab^3)$ of the $(n, k, L)^{++}$-protocol induces a large computational overhead in the form of several multiplications over $\mathrm{GF}(2^{\frac{n}{2}})^*$. Analogously, lookup tables, e.g., in order to efficiently compute b^3, would be rather expensive in terms of chip area.

In this section, we introduce the new $(n, k, L)^{\langle 80 \rangle}$-protocol to overcome the two problems mentioned above. In short, the basic ideas are summarized as follows:

- To shorten the key length, the linear functions are no longer randomly sampled and stored but are computed from a smaller seed.
- To lower the effort of the connection function, we replace it by several subfunctions which compute the same functionality but on a smaller domain.

One consequence of these modifications w.r.t. the original $(n, k, L)^{++}$-protocol is that the new protocol needs to be executed several times per authentication instance, i.e., it is composed of multiple (algorithmically identical) rounds.

4.2.1 Modifications w.r.t. the original $(n, k, L)^{++}$-Protocol

We will now explain all relevant modifications in further detail. The description of the resulting, new $(n, k, L)^{\langle 80 \rangle}$-protocol will then be given in the following Subsection 4.2.2, along with a suggestion for concrete instantiations with the parameters $n = 128$, $k = 32$, $L = 16$ and $n = 64$, $k = 32$, $L = 16$, respectively.

Shortening the Key Length. The basic idea is to take a keystream generator (KSG) G that uses a seed of length $m + M$ to (pseudorandomly) generate the $((n+k) \cdot n) \cdot L$ key bits characterizing the secret linear functions $F_1, \ldots, F_L$. In particular, we suppose

that $L = 2^M$ for a small $M \in \mathbb{N}$ (e.g., $M = 4$) and represent each index l, $1 \leq l \leq L$, as an M-bit string, where $l = 1$ corresponds to $0 \ldots 00$, $l = 2$ corresponds to $0 \ldots 01$ etc. Hence, given a secret symmetric session key $\kappa = (\kappa_0, \ldots, \kappa_{m-1})$, the entries of the matrix corresponding to F_l are certain bits from the keystream produced by G on (l, κ). Striving for a lightweight construction, it might be tempting to employ a single linear feedback shift register (LFSR) as a simple bitstream generator G. However, we show in Subsection 4.3.1 that allowing the matrices of $F_1, \ldots, F_L$ to be generated by a keystream of small linear complexity opens the door to an algebraic attack which is much more efficient than the aforementioned algorithm from [KH11].

Splitting the Connection Function. Another open problem was to reduce the cost introduced by the *connection function* $f : \mathrm{GF}(2^{\frac{n}{2}})^* \times \mathrm{GF}(2^{\frac{n}{2}})^* \longrightarrow \mathrm{GF}(2^{\frac{n}{2}})^* \times \mathrm{GF}(2^{\frac{n}{2}})^*$, which is applied to the random values $a, b \in \mathrm{GF}(2^{\frac{n}{2}})$, $a, b \neq 0$, before they are fed to one of the L secret linear functions $F_1, \ldots, F_L$.[1] Instead of using $f(a, b) = (ab, ab^3)$ as a connection function (and thus multiplying over $\mathrm{GF}(2^{\frac{n}{2}})^*$), in the new $(n, k, L)^{\langle 80 \rangle}$-protocol, we compute

$$f(a, b) = \left(\left(a_1 b_1, a_1 b_1^3 \right), \ldots, \left(a_{n/8} b_{n/8}, a_{n/8} b_{n/8}^3 \right) \right),$$

where $a_i, b_i \in \mathrm{GF}(2^4)$, $a_i, b_i \neq 0$, are obtained by splitting a and b into blocks of 4 bits, respectively. The practical security implications of this modification, which reduces the number of valid challenge-nonce pairs (a, b) from $(2^{n/2} - 1)^2$ to $(2^4 - 1)^{n/4}$, are mainly confined to the active attack discussed in Subsection 4.3.2.

Further Modification. On contrast to the (practically infeasible) $(n, k, L)^{++}$-protocol, it is necessary to run the $(n, k, L)^{\langle 80 \rangle}$-protocol multiple times in order to obtain sufficient resistance w.r.t. certain MITM attacks. The reasons for this are twofold: First, for efficiency reasons, our concrete instantiations (introduced in Subsection 4.2.2) and their respective implementations (described in Section 4.4) use challenge-nonce tuples of length at most $n = 128$ bits as compared to $n = 256$ bits suggested for $(n, k, L)^{++}$ in [KS09]. Second, one has to compensate for the aforementioned decrease of valid inputs (a, b) resulting from splitting up a and b into blocks of size 4 bits each as part of the modified connection function. In Subsection 4.3.2, we show that these modifications lead to an upper bound of $2^{-n/4}$ (e.g., 2^{16} for $n = 64$) for the success probability of a certain MITM attacker to convince an honest verifier to accept an illegitimate response. As, even with $n = 128$, this success probability is too large for practical applications, one has to run the protocol at least two times, which would, e.g., lead to an upper bound of $2^{-n/2}$ due to the fact that the rounds can be considered independent w.r.t. the details of this type

[1] Like in the description of the original $(n, k, L)^{++}$-protocol in Subsection 3.5.1, we identify the vector space $\mathrm{GF}(2)^{\frac{n}{2}}$ with the finite field $\mathrm{GF}(2^{\frac{n}{2}})$ here.

of attack. We consider a success probability of 2^{-64} sufficiently small in the context of ultra-constrained RFIDs, where it seems implausible that a respective MITM attacker will be able to interfere in a correspondingly large number of authentication instances between a valid tag and a valid reader. Moreover, as pointed out in Section 3.2, there should always be a back-end system which checks for such malicious activities and, in the above case, could simply ban the respective RFID tag after a certain number of failed authentication attempts caused by the MITM attack.

Important remark: In our paper [AHK14], where the original $(n, k, L)^{80}$-scheme was introduced, we wrote at this point:

> "As a final modification to the original $(n, k, L)^{++}$-protocol, we introduce a (publicly known) bit-wise permutation σ to the n-bit result of $f(a, b)$. Note that in terms of hardware efficiency, such a bit-wise permutation comes at practically no cost as it is realized simply through wires and does not involve any additional gates." [AHK14]

In hindsight, the author of this thesis is now convinced that the additional introduction of this bitwise permutation σ had actually no effect (i.e., neither positive nor negative) on the security of the protocol. The underlying intuition for the use of σ in [AHK14] was to compensate for the fact that the new connection function

$$f(a, b) = \left(\left(a_1 b_1, a_1 b_1^3\right), \ldots, \left(a_{n/8} b_{n/8}, a_{n/8} b_{n/8}^3\right)\right)$$

now effectively maps separate blocks of 8 bits to separate blocks of 8 bits (instead of mapping one block of n bits to one block of n bits in the original $(n, k, L)^{++}$-protocol). But while such bitwise permutations are in fact an integral part of many cryptographic schemes (such as substitution-permutation networks (SPNs) used, e.g., for the block ciphers AES [DR02] and PRESENT [BKL+07]), their use in our paper [AHK14] seems to have been rather a case of 'security by obscurity'. For in contrast to SPNs, the output of our permutation σ is not fed to a nonlinear S-box, but instead to a secret GF(2)-linear function F_l. Consequently, as σ is public, the fact that the individual bits of the 8-bit output blocks of f are now spread through σ over the whole n-bit input vector for F_l does not induce any additional difficulties for an attacker who seeks to deduce the specification of F_l. Thus, despite the fact that our statement in [AHK14] that "such a bit-wise permutation comes at practically no cost as it is realized simply through wires and does not involve any additional gates" remains true, we decided to remove σ for the $(n, k, L)^{\langle 80 \rangle}$-protocol presented in this thesis, as we believe it to be good cryptographic practice to include only components/operations for which an actual security benefit can be shown (see also our corresponding criticism w.r.t. the stream cipher Fruit in Chapter 6). Nevertheless, we would also like to point out that the security analysis for $(n, k, L)^{80}$ given in [AHK14] was independent of the use of σ and, hence, remains valid for the $(n, k, L)^{\langle 80 \rangle}$-protocol as shown in Section 4.3.

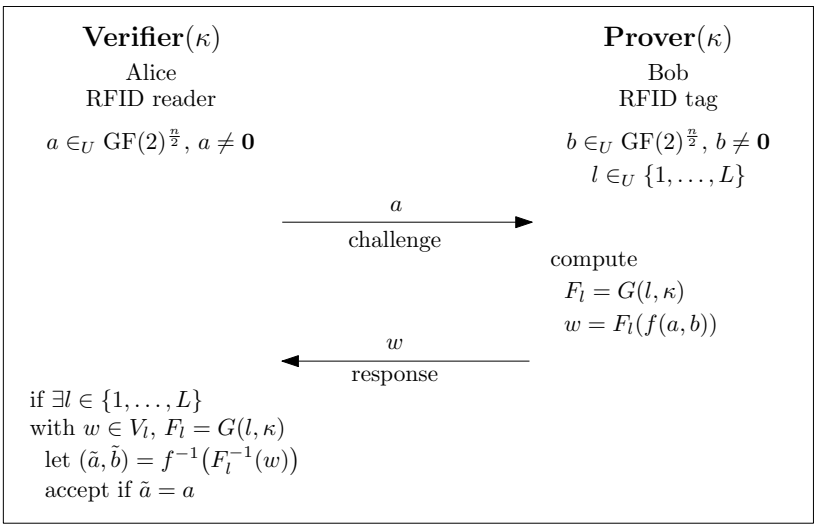

Figure 4.1: One round of the $(n, k, L)^{\langle 80 \rangle}$-protocol. The computation of $F_l = G(l, \kappa)$ through the generator G can be performed on the fly while computing $w = F_l(f(a, b))$. As for the original $(n, k, L)^{++}$-protocol, V_l denotes the vector space corresponding to the image of F_l.

4.2.2 Protocol Description

This subsection is divided into two parts. In the first part, we describe the general properties of the $(n, k, L)^{\langle 80 \rangle}$-protocol for arbitrary choices of the parameters n, k, L and an abstract generator G. Building on this, we will then suggest concrete instantiations for the parameters $n = 128$, $k = 32$, $L = 16$ and $n = 64$, $k = 32$, $L = 16$, respectively, in the second part of the subsection. These instantiations will also form the basis of our hardware implementation presented in Section 4.4 and come with corresponding sets of test vectors.

General Properties

The $(n, k, L)^{\langle 80 \rangle}$-protocol proceeds according to the original $(n, k, L)^{++}$-scheme described in Subsection 3.5.1, with the additional condition that n must be divisible by 8 (see below). As depicted in Fig. 4.1, the process is initiated by the verifier Alice, who chooses some $a \in \mathrm{GF}(2)^{\frac{n}{2}}$ uniformly and at random and sends it to the prover Bob. Bob then also randomly chooses some $b \in \mathrm{GF}(2)^{\frac{n}{2}}$ and some $l \in \{1, \ldots, L\}$, and answers with

$$w = F_l(f(a, b)) = F_l\Big(\Big(a_1 b_1, a_1 b_1^3\Big), \ldots, \Big(a_{n/8} b_{n/8}, a_{n/8} b_{n/8}^3\Big)\Big), \tag{4.1}$$

where a_i and b_i are elements of $\mathrm{GF}(2^4)$ that correspond to the respective 4-bit blocks of a and b. Remember that in order to allow for inverting $f(a, b)$ as part of the verification, only challenges a and nonces b satisfying $a_i, b_i \neq 0$ for all $i \in \{1, \ldots, \frac{n}{8}\}$ are allowed by the

protocol. The verification step of Alice is exactly the same as for the original $(n, k, L)^{++}$-protocol. Please note that while a might be known to an adversary eavesdropping on the communication between Alice and Bob, b is kept strictly secret by the prover and is only used to compute $f(a, b)$.

As outlined previously, in order to achieve a feasible key length for the $(n, k, L)^{\langle 80 \rangle}$-protocol, we suggest to deploy a generator G for specifying the secret linear functions $F_1, \ldots, F_L$ based on an 80-bit secret key $\kappa = (\kappa_0, \ldots, \kappa_{79})$ shared by Alice and Bob. For $l \in \{1, \ldots, L\}$ and κ as inputs (i.e., as a seed) of G, we denote by $G(l, \kappa)$ the corresponding $((n + k) \cdot n)$-bit output $(z_0, \ldots, z_{(n+k)\cdot n-1})$, which determines the secret function F_l and the computation of its $n + k$ output bits $(w_0, \ldots, w_{n+k-1})$, i.e., the actual authentication token, on an input $f(a, b) = (x_0, \ldots, x_{n-1})$ as follows:

$$\begin{aligned} w_0 &= z_0 \cdot x_0 \oplus \cdots \oplus z_{n-1} \cdot x_{n-1}, \\ &\cdots \\ w_{n+k-1} &= z_{(n+k-1)\cdot n} \cdot x_0 \oplus \cdots \oplus z_{(n+k-1)\cdot n+(n-1)} \cdot x_{n-1}. \end{aligned} \tag{4.2}$$

Similar to general stream cipher design (see, e.g., Chapter 5), it is obviously crucial here that the chosen generator G, which is a finite-state machine, will produce an output stream of period at least $(n + k) \cdot n$ for any seed (l, κ). In the case of our concrete instantiations of the $(n, k, L)^{\langle 80 \rangle}$-protocol, which will be presented in the second part of this subsection, we achieve this by using a maximum-length LFSR as the central component of G and explicitly forbidding the use of the all-zero key.

It should be noted that in the course of computing, e.g., w_0, only a one-bit-wide register is needed in hardware, i.e., firstly, $z_0 \cdot x_0$ is computed and stored, then $z_1 \cdot x_1$ is XORed, and so on, until $z_{n-1} \cdot x_{n-1}$ has been added and w_0 is finally ready to be transmitted to the verifier. This is an important property as volatile memory in the form of flip-flops is especially costly in terms of area and power consumption, so that its use should be restricted to an absolute minimum when designing lightweight cryptographic protocols.

While we trade an increase in the number of clock cycles for a reduction of area (and, thus, also *static* power consumption; cf. Subsection 2.3.4) in several parts of the $(n, k, L)^{\langle 80 \rangle}$-protocol (see, e.g., the above paragraph), the hardware implementation outlined in Section 4.4 also contains measures to reduce the time complexity where possible. Most notably, the blockwise evaluation of $f(a, b)$ can be performed concurrently with the initialization phase of the generator G without inducing any additional hardware cost. This allows to start computing the first token bit y_0 instantly once G is ready.

Before we proceed with the second part of this subsection containing our concrete instantiations of the $(n, k, L)^{\langle 80 \rangle}$-protocol, let us briefly explain one more design decision. Providing the secret function index l as an input to the generator G is, in fact, not stringently necessary. More precisely, we could have also obtained different specifications of the respective L secret functions on the basis of a single, larger output $\left(z_0, \ldots, z_{L\cdot(n+k)\cdot n-1}\right)$

of a generator G with κ as its only input as follows:

$$F_1 \mathrel{\widehat{=}} \left(z_0, \ldots, z_{(n+k)\cdot n-1}\right),$$
$$\ldots$$
$$F_L \mathrel{\widehat{=}} \left(z_{(L-1)\cdot(n+k)\cdot n}, \ldots, z_{L\cdot(n+k)\cdot n-1}\right).$$

This, however, would have resulted in a computational load unacceptable already for moderate parameter choices like $n = 64$, $k = 32$, and $L = 16$ (for which, in the worst case, $16 \cdot (64 + 32) \cdot 64 = 98\,304$ bits would need to be generated by G per protocol round) in the context of ultra-constrained RFIDs. Remember that for a single authentication instance with this choice of parameters, four rounds are recommended to obtain adequate (MITM) security, resulting in almost 400 000 bits to be generated by the prover's generator G.[2] Comparing this to the common limit of 150 ms w.r.t. authentication times and a clock rate of 100 kHz prevalent for ultra-constrained RFIDs as described in Chapter 2, such a solution would have clearly not been feasible. Finally, we also believe that our preferred solution of using a generator G with l as an additional input is also superior in terms of security. This judgement is based on the fact that for the generation of each function specification F_l, G then runs its initialization phase anew under a different seed (due to the change of the function index l), thus better destroying the relations between the generator's inner states underlying the different secret functions $F_1, \ldots, F_l$.

Concrete Instantiations for $n = 128$ and $n = 64$

After the generalized description of the $(n, k, L)^{\langle 80 \rangle}$-protocol in the former part of this subsection, we will now provide two suggestions for concrete instantiations with the parameters $n = 128$, $k = 32$, $L = 16$ and $n = 64$, $k = 32$, $L = 16$, respectively. These instantiations will also form the basis of our hardware implementation presented in Section 4.4. Corresponding sets of test vectors are provided in Appendix 4.A.

The respective parameter choices are motivated by our previous security considerations for (n, k, L)-type authentication protocols as discussed in Subsection 3.5.2. In particular, applying the algebraic attack approach from [KH11] against our new $(n, k, L)^{\langle 80 \rangle}$-protocol with parameters $n = 128$ and $L = 16$ (resp. $n = 64$ and $L = 16$) implies solving a system of around $\binom{128}{16} \approx 2^{66}$ (resp. $\binom{64}{16} \approx 2^{49}$) unknowns, which can be considered sufficiently difficult. For $n = 128$, choosing $L = 8$ would, in fact, already be adequate. The reason for choosing $L = 16$ anyhow, is that this allows us to use the same generator for both instantiations. We not only consider this more elegant, but it has also a very

[2] Despite the fact that, in this alternative scenario (where G takes only the key κ as its single input), the output of G would be the same for all rounds, these bits would still need to be generated anew per round, due to a lack of corresponding storage capabilities on ultra-constrained RFIDs. After all, the very reason for using a generator G is to avoid having to store an explicit specification of the secret functions $F_1, \ldots, F_L$.

practical advantage, in that it allows for applying corresponding security results to both instantiations simultaneously.

Both variants generate their respective authentication tokens (of length 160 bits for $n = 128$, $k = 32$, $L = 16$ and of length 96 bits for $n = 64$, $k = 32$, $L = 16$) as shown in Eqs. (4.1) and (4.2). In order to resist the MITM attack explained in Subsection 4.3.2, we suggest that an authentication instance for $n = 128$ should consist of two rounds, whereas an authentication instance for $n = 64$ should consist of four rounds, which, in both cases, all need to be passed by the prover in order for the verifier to finally answer with `accept`. Note that the verifier will not provide intermediate results, i.e., the prover (or an attacker) will only learn whether the authentication instance as a whole was successful or not. To this end, it is also important that always the full number of rounds per authentication instance is executed, even if the prover should have failed, e.g., already in the first round.

The last missing 'detail' in the description of our two concrete instantiations is the specification of the employed generator G, which, as pointed out above, is the same for both variants. Section 4.3 will show that while the $(n, k, L)^{\langle 80 \rangle}$-protocol can be instantiated with any secure and hardware-efficient KSG, it is actually sufficient to use a pseudorandom bit generator whose bitstream does not suffer from a small linear complexity. For this reason, we decided to use the *self-shrinking generator* by Meier and Staffelbach [MS94] on top of a simple maximum-length LFSR (i.e., an LFSR with a primitive feedback polynomial). While only few additional gates are needed to implement the logic of the self-shrinking generator when compared to implementing a mere maximum-length LFSR, the security benefit is enormous. The best currently known attacks against self-shrinking generators are a time-memory attack by Mihaljević [Mih96] and an OBDD-based attack by Zenner, Krause, and Lucks [ZKL01]. However, we do not see how to use these attacks in order to realize a nontrivial attack against the $(n, k, L)^{\langle 80 \rangle}$-protocol.[3] In particular, the fact that no algebraic attacks are known makes the self-shrinking generator seem especially suited for our context. Moreover, Meier and Staffelbach have proved in [MS94] that a self-shrinking generator based on a maximum-length LFSR of size N bits has a period of at least $2^{\lfloor N/2 \rfloor}$ and a linear complexity of at least $2^{\lfloor N/2 \rfloor - 1}$, both

[3] Also note that in a classical time-memory-data tradeoff (TMD-TO) attack (see, e.g., Chapter 7) against the $(n, k, L)^{\langle 80 \rangle}$-protocol, the n-bit result of $f(a, b)$ would have to be treated as part of the inner state. For $n = 128$ and a generator G of size 84 bits (see below), this would lead to an attack complexity of $2^{(128+84)/2} = 2^{106}$, i.e., above that of exhaustive key search. For $n = 64$, the corresponding complexity would be 2^{74} and, hence, slightly below the effort for exhaustive key search. However, here, an attacker also has to face the problem that the challenge space is only of size $2^{64/2}$ (like IVs of stream ciphers, challenges should obviously not be reused), as a result of which he will not be able to collect enough data for an attack with overall complexity below 2^{80}. Another problem of straightforwardly applying a classical TMD-TO attack against the $(n, k, L)^{\langle 80 \rangle}$-protocol would be that, per challenge, an attacker could only obtain $n + k$ bits of data (as the output tokens of the $(n, k, L)^{\langle 80 \rangle}$-protocol would correspond to the collected keystream pieces in a TMD-TO attack against stream ciphers). For our choice $k = 32$, data pieces of size $n + k$ bits would, however, not be sufficient to uniquely identify the underlying inner states comprising of the 84-bit state of G and the n-bit value of $f(a, b)$.

of which are perfectly sufficient in our context, where $N = 84$ bits. More precisely, we use the primitive feedback polynomial $x^{84} + x^{82} + x^{62} + x^{49} + x^{30} + x^{15} + 1$ and, as an initialization phase, clock the LFSR 256 times without producing output. Note that we are aware of the fact that for $N = 84$, there would also be primitive polynomials with fewer terms (such as $x^{84} + x^{71} + 1$), but we deliberately chose this one for faster diffusion of inputs. The hardware costs of the additionally required gates are negligible.

To avoid ambiguity, please find below an algorithmic description of the initialization and bitstream generation of the employed generator G, whose inner state at time $t = 0, 1, \ldots$ we denote by $(G_0^t, \ldots, G_{83}^t)$.

Key and Function Index Loading. Let $\kappa = (\kappa_0, \ldots, \kappa_{79}) \neq \mathbf{0}$ denote the 80-bit secret key and let $(l_3, \ldots, l_0)$ denote the 4-bit binary representation of the index $l \in \{1, \ldots, L\}$, $L = 16$, of the secret function F_l whose specification $(z_0, \ldots, z_{(n+k)\cdot n-1})$ shall be generated. Note that l_0 represents the least significant bit (LSB) here and that we start the binary indexing with $(0, 0, 0, 0)$; e.g., $l = 1$ corresponds to $(l_3, l_2, l_1, l_0) = (0, 0, 0, 0)$ and $l = 2$ corresponds to $(l_3, l_2, l_1, l_0) = (0, 0, 0, 1)$. The cells of G are now are initialized as follows:

$$G_i^0 := \begin{cases} l_{3-i}, & \text{for } i \in \{0, \ldots, 3\}\,, \\ \kappa_{i-4}, & \text{for } i \in \{4, \ldots, 83\}\,. \end{cases}$$

Generator Initialization. For $t = 0, \ldots, 127$, compute:

$$\begin{aligned} G_i^{t+1} &:= G_{i+2}^t, \quad \text{for } i \in \{0, \ldots, 81\}\,, \\ G_{82}^{t+1} &:= G_0^t \oplus G_2^t \oplus G_{22}^t \oplus G_{35}^t \oplus G_{54}^t \oplus G_{69}^t, \\ G_{83}^{t+1} &:= G_1^t \oplus G_3^t \oplus G_{23}^t \oplus G_{36}^t \oplus G_{55}^t \oplus G_{70}^t. \end{aligned}$$

Note that we describe a parallelized implementation of the generator's LFSR here, which requires only 128 clock cycles (instead of 256 for a classical, non-parallelized implementation under the above primitive feedback polynomial). Due to the extremely simple feedback function, this kind of parallelization is very cheap in hardware (only few additional gates) and, hence, also employed in our Verilog reference implementation presented in Appendix 4.B. The reason for using it already here as part of the algorithmic specification of our protocol instantiations is that it is especially suited for the self-shrinking generator and allows to describe the following phase rather elegantly.

Generating the Secret Function. Set $j := 0$. For $t = 128, \ldots$, do:

1) If $G_0^t = 1$, then set $z_j := G_1^t$ and $j := j + 1$.

2) If $j = (n + k) \cdot n$, then stop the computation (as we are done). Else, compute:

$$G_i^{t+1} := G_{i+2}^t, \quad \text{for } i \in \{0, \ldots, 81\}\,,$$

$$G_{82}^{t+1} := G_0^t \oplus G_2^t \oplus G_{22}^t \oplus G_{35}^t \oplus G_{54}^t \oplus G_{69}^t,$$
$$G_{83}^{t+1} := G_1^t \oplus G_3^t \oplus G_{23}^t \oplus G_{36}^t \oplus G_{55}^t \oplus G_{70}^t.$$

As pointed out previously, in practice, the generation of the secret function specification $(z_0, \ldots, z_{(n+k)\cdot n-1})$ is performed on the fly during the computation of the authentication token $(w_0, \ldots, w_{n+k-1})$ as described in Eq. (4.2).

This concludes the algorithmic specification of our concrete $(n, k, L)^{\langle 80 \rangle}$-protocol instantiations for $n = 128$, $k = 32$, $L = 16$ (with 2 rounds) and $n = 64$, $k = 32$, $L = 16$ (with 4 rounds). But before we proceed with the security analysis in the following Section 4.3, let us briefly explain why we actually suggest two variants (each with key size 80 bits), despite the fact the we consider both of them secure against the algebraic attack in [KH11] and the adapted (x, y)-*equality attack* in Subsection 4.3.2. The reason lies in the tradeoff between challenge size versus hardware costs. In Section 4.4, we will see that for $n = 128$ the chip area of an ASIC implementation is almost 50 percent higher than for $n = 64$. In the context of ultra-constrained RFIDs, it seems rather unlikely that more than $2^{64/2}$ different challenges will be required during the lifetime of a corresponding tag. However, if the challenges have to be chosen randomly (e.g., in the case of multiple verifiers that are not connected via a common back end), a larger challenge space (such as 2^{64} for $n = 128$) can, in fact, become necessary in order to reduce the risk of repetitions. To avoid potential misconceptions here, we would like to point out the following facts:

- Despite the reduced challenge space of size $2^{64/2}$ for the protocol variant $n = 64$, $k = 32$, $L = 16$ (with 4 rounds), it is not sufficient for an attacker who wants to impersonate a valid tag to be in possession of one valid authentication token for a 32-bit challenge 'accidentally' asked twice by a verifier. Instead, he has to pass all for rounds of the respective authentication instance, implying that he actually needs to know the correct answers to each of the four 32-bit challenges posed by the verifier.

- The vigilant reader may point out that our description of the general $(n, k, L)^{\langle 80 \rangle}$-protocol structure in the first part of this subsection (and, e.g., in Fig. 4.1) actually commands that the challenges are to be chosen randomly by the verifier, which would forbid to implement additional countermeasures against the risk of repeating challenges in order to maximize the lifetime of tags that use protocol instantiations with relatively small n, such as $n = 64$. However, the reason behind this kind of description is actually that we wanted to follow the description of the underlying $(n, k, L)^{++}$-protocol, where the challenges are also randomly chosen. But, in fact, studying the algebraic attack from [KH11] and the adapted (x, y)-*equality attack* in Subsection 4.3.2 shows that neither of them would benefit from non-randomly chosen challenges. Consequently, we see no reason why the verifier should not

apply suitable techniques for making sure that the posed challenges are unique, thus allowing for longer lifetimes of tags due to the eliminated risk of repeating challenges before the theoretically possible maximum $n/2$ is finally reached.

Finally, apart from the possibility to choose $n = 128$ or $n = 64$ depending on the number of challenges required by the application scenario, we also provide the two protocol instantiations as certain users might be willing to pay the cost of increased chip area for the the variant $n = 128$, $k = 32$, $L = 16$ (with 2 rounds), in order to have a larger security margin against possible future attacks (such as improved algorithms for solving the learning problem $\mathit{RandomSelect}(L, n, \rho)$ as described in Subsection 3.5.2).

4.3 Security Analysis

In this section we analyze the security of the $(n, k, L)^{\langle 80 \rangle}$-protocol, which is, as pointed out previously, in fact a variant of the $(n, k, L)^{++}$ authentication protocol from [KS09] where some modifications have been made for improving the hardware efficiency. In a nutshell, these modifications are (cf. Subsection 4.2.1):

- The linear functions F_l are not randomly chosen but, based on a secret key κ, generated from the seed (l, κ) using a bitstream generator G.
- The connection function has been broken down into several subfunctions which all realize in principle the same function, but restricted to a smaller domain.

Consequently, we investigate if and to what extent these modifications impact the security of the $(n, k, L)^{\langle 80 \rangle}$-protocol in comparison to the security of the underlying $(n, k, L)^{++}$-protocol. With respect to the latter, remember that the best attacks known so far against (n, k, L)-type protocols are the algebraic attack from [KH11] (cf. Subsection 3.5.2) and the active MITM attack named (x, y)-*equality attack* from [KS09] (cf. Subsection 4.3.2). For appropriately chosen parameters, the $(n, k, L)^{++}$-protocol is resistant against both types of attacks (see [KS09], [KH11], and Subsection 3.5.2).

4.3.1 Impact of Using a Generator G

In this subsection, we analyze the security impact if the secret linear functions $F_1, \ldots, F_L$ are not randomly chosen, but derived from a bitstream produced by a generator G. To this end, we will first demonstrate that if G is *weak* (more precisely, if the generated bitstream exhibits a small linear complexity), then, in a certain setting, the whole protocol becomes vulnerable to a variant of the algebraic attack from [KH11] that allows for computing the specifications of $F_1, \ldots, F_L$ much more efficiently compared to the general case of the respective learning algorithm. This shows the necessity for stronger generators. In fact, we argue afterwards, using a standard hybrid argument, that using G does not

imply any significant change in the protocol's security if G is a secure[4] pseudorandom bit generator.

An Algebraic Attack for Weak Generators

In the following, we present an efficient algebraic attack if the generator G produces a bitstream with a short, known linear complexity. For simplicity, we consider the case that G is realized by a maximum-length LFSR. Observe, however, that the same attack works against any G which produces a bitstream with low, known linear span.

But before we start, let us point out that the attack scenario in this subsection differs from the one described in Appendix B of [KH11], where it is shown that an efficient algorithm for the learning problem *RandomSelect*(L, n, ρ) (cf. Subsection 3.5.2) could be used to break the $(n, k, L)^+$-protocol. More precisely, in [KH11], the respective algorithm for *RandomSelect*(L, n, ρ) serves to learn the L secret subspaces $V_1, \ldots, V_L$ of $\{0,1\}^{n+k}$ corresponding to the images of the L secret functions $F_1, \ldots, F_L : \{0,1\}^n \longrightarrow \{0,1\}^{n+k}$, but not the specifications of $F_1, \ldots, F_L$ themselves. The reason for this is that the learning problem *RandomSelect*(L, n, ρ) assumes, as described in Subsection 3.5.2, that the learner knows (*input*, *output*)-tuples (but not the corresponding function indices) for the set of secret functions whose specifications he wants to deduce.

In (n, k, L)-type protocols, however, the secret nonce $b \in \{0,1\}^{n/2}$ is also part of the input to F_l, thus making it impossible to apply *RandomSelect*(L, n, ρ) directly to $F_1, \ldots, F_L$ in practice. Instead, for the attack against the $(n, k, L)^+$-protocol, it is assumed that the secret subspaces $V_1, \ldots, V_L$ have a special form which allows to recover their specification using a learning algorithm for *RandomSelect*(L, n, ρ) (see [KH11] for further details). Based on the knowledge of $V_1, \ldots, V_L$, it is then shown how to fully break the $(n, k, L)^+$-protocol. Note that while a learning algorithm for *RandomSelect*(L, n, ρ) analogously allows to obtain the specifications of $V_1, \ldots, V_L$ for the $(n, k, L)^{++}$-protocol, it has not yet been shown how this information can then be used to actually break the protocol. Hence, as pointed out previously, the hardness of *RandomSelect*(L, n, ρ) is actually conjectured to be a lower bound for the hardness of breaking the $(n, k, L)^{++}$-protocol.

In the same spirit, we will assume an even more powerful attacker (subsequently called *strong attacker*) for the following algebraic attack for weak generators w.r.t. our new $(n, k, L)^{\langle 80 \rangle}$-protocol. More precisely, we consider a passive attacker who actually knows the secret nonces $b \in \{0,1\}^{n/2}$ and thus, as the challenges $a \in \{0,1\}^{n/2}$ are public, input-output tuples $(u, w) = (f(a, b), F_l(f(a, b)) \in \{0,1\}^n \times \{0,1\}^{n+k}$ for the set of secret functions $F_1, \ldots, F_L$.[5] In consequence, the following results are not meant to suggest

[4] The meaning of the term *secure* in this context will be defined more precisely during the respective security reduction in the second part of this subsection.

[5] Note that this strong attacker scenario is not fully implausible as the example of the MIFARE Classic's weak RNG in Subsection 3.2.2 has shown.

that the $(n, k, L)^{\langle 80 \rangle}$-protocol can definitely be broken in practice if the used generator G produces a bitstream with a short, known linear complexity. Nevertheless, they definitely suggest that, as a measure of caution, such generators should be avoided (in particular, given that superior building blocks, such as the self-shrinking generator used for our concrete protocol instantiations in Subsection 4.2.2, are available at the cost of only few additional hardware gates).

The $(n, k, L)^{++}$-Protocol versus a Strong Attacker. Let $F_1, \ldots, F_L : \mathrm{GF}(2)^n \longrightarrow \mathrm{GF}(2)^{n+k}$ denote the secret key consisting of L injective GF(2)-linear mappings, where n, k, L are appropriately chosen. A *strong attacker* tries to compute specifications of these functions on the basis of pairs (u, w), where $w = F_l(u)$ for some secret index l, which is randomly and uniformly chosen from $\{1, \ldots, L\}$ for each new pair.

In this scenario, the learning algorithm for *RandomSelect*(L, n, ρ) from [KH11] can now be applied straightforwardly by choosing appropriate parameters μ, ρ such that $\mu \cdot \rho = n + k$, considering the secret functions F_l as vectors of μ component functions mapping from $\mathrm{GF}(2)^n$ into $\mathrm{GF}(2)^\rho$, identifying $\mathrm{GF}(2)^\rho$ with the finite field $K = \mathrm{GF}(2^\rho)$, and computing the component functions by means of the following algebraic attack approach:

Suppose we are given secret linear functions $f_1, \ldots, f_L : K^n \longrightarrow K$ and we want to compute specifications of these functions on the basis of known plaintext pairs (x, y), where x is randomly and uniformly chosen from $\{0, 1\}^n \subseteq K^n$ and it holds that $y = f_l(x)$ for some secret index l, which is randomly and uniformly chosen from $\{1, \ldots, L\}$ for each new pair.

We were done if we could compute the values $x_{i,l} = f_l(e_i)$ for $i = 1, \ldots, n$ and $l = 1, \ldots, L$, where $e_i \in K^n$ denotes the standard vector having one at position i and zero at all other positions.

Note that each known plaintext pair (x, y) yields a degree-L equation in the $x_{i,l}$-variables of the form

$$\prod_{l=1}^{L} \left(\bigoplus_{i \in I} x_{i,l} \oplus y \right) = 0,$$

where $x = \bigoplus_{i \in I} e_i$.

In [KH11], it is shown that systems built of degree-L equations of this kind can be solved by a nontrivial application of the technique of linearization, which implies to solve a system of linear equations over $O(n^L)$ variables (see the corresponding outline in Subsection 3.5.2). To the best of our knowledge, no faster approach has been suggested so far. So, even in the case of a *strong attacker*, who actually knows the secret nonces b and, hence, the inputs to the secret functions F_l, the $(n, k, L)^{++}$-protocol remains secure against passive attacks for parameter choices like $n = 128$ and $L = 8$ or $n = 64$ and $L = 16$.

The $(n,k,L)^{\langle 80\rangle}$-Protocol with a Weak G versus a Strong Attacker. We analyze now the case that the $((n+k)\cdot n)\cdot L$ bits characterizing the secret linear functions $F_1,\ldots,F_L$ are generated by one maximum-length LFSR of length $m+M$, where $L=2^M$. Remember that the secret symmetric key $\kappa=(\kappa_0,\ldots,\kappa_{m-1})$ and the M random bits $l_{M-1},\ldots,l_0$ forming the binary representation of the secret index $l\in\{1,\ldots,L=2^M\}$ serve as the initial state of the LFSR.

We show in the following that when facing a *strong attacker* (who knows the secret nonces b and, hence, the inputs $f(a,b)$ to the secret functions), this construction opens the door to an algebraic attack allowing to compute the secret key bits much more efficiently as compared to the general case described in the previous part about the $(n,k,L)^{++}$-protocol.

For demonstrating this, we consider the algebraic attack against the $(n,k,L)^{++}$-protocol described above and suppose that ρ is chosen by the attacker such that $\rho=M+1$. Our construction implies that each bit of the function matrices of $F_1,\ldots,F_L$ and, consequently, each bit of the secret K-elements $x_{i,l}$, is the output of a publicly known GF(2)-linear mapping in the κ-bits and the random l-bits.

Hence, the secret K-elements $x_{i,l}$ can be written as

$$x_{i,l}=\bigoplus_{s=0}^{m-1}c_{i,s}\kappa_s\oplus\bigoplus_{t=0}^{M-1}C_{i,t}l_t, \tag{4.3}$$

where the bits $l_{M-1},\ldots,l_0$ of l and the vectors $c_{i,s},C_{i,t}\in\mathrm{GF}(2)^\rho$ are publicly known.[6] Thus, each known plaintext pair (x,y), $x=\bigoplus_{i\in I}e_i$, translates into the statement that

$$\bigoplus_{s=0}^{m-1}\left(\bigoplus_{i\in I}c_{i,s}\right)\kappa_s\ \in\ W(y),$$

where the set $W(y)\subseteq\mathrm{GF}(2)^\rho$ is defined by $W(y)=\{y\oplus C_{I,1},\ldots,y\oplus C_{I,L}\}$ and for each $(l_{M-1},\ldots,l_0)$ representing an element $l\in\{1,\ldots,L\}$ it holds that

$$C_{I,l}=\bigoplus_{t=0}^{M-1}\left(\bigoplus_{i\in I}C_{i,t}\right)l_t.$$

Now we can compute a nonzero Boolean function $g:\{0,1\}^\rho\longrightarrow\{0,1\}$ which annihilates $W(y)$. This is possible as $W(y)$ is a proper subset of $\{0,1\}^\rho$ due to $|W(y)|\le 2^M=2^{\rho-1}$.

More concretely, we compute a square free polynomial $p=p\,(z_1,\ldots,z_\rho)$ which yields g. This can be done by solving a system of at most L GF(2)-linear equations in at most

[6] To avoid potential misconceptions, note that, while the indices l are secretly chosen at random during the execution of the protocol, in the context of Eq. (4.3) the binary representation $l_{M-1}\ldots,l_0$ of l for $x_{i,l}$ is publicly known.

2^ρ variables corresponding to the square free monomials over $z_1, \ldots, z_\rho$. As M and ρ are small numbers in practice, this is feasible. Note that the degree of p is at most ρ.

Consequently, the known plaintext pair (x, y) yields the following nonlinear equation in the key bits:

$$p\left(\bigoplus_{s=0}^{m-1}\left(\bigoplus_{i\in I} c_{i,s}\right)\kappa_s\right) = 0.$$

The degree of this equation is at most $\rho = \log_2(L) + 1$, which is much smaller than L, the degree of the algebraic attack for the general case as used above against the $(n, k, L)^{++}$-protocol. This shows that generators whose bitstream exhibits a small linear complexity should be avoided for instantiations of our new $(n, k, L)^{\langle 80 \rangle}$-protocol.

Security Reduction for Pseudorandom-Bit-Generators G.

Next, we consider the case that G is instantiated by a bitstream generator which produces a bitstream (z_i) of pseudorandom bits given a seed $\alpha \in \mathrm{GF}(2)^r$. More precisely, let $q = ((n + k) \cdot n) \cdot L$ be the number of bits that characterize the secret linear functions $F_1, \ldots, F_L$. For simplicity, we assume that the first q outputs of G eventually define the linear functions. Now let G be a (q, t, ε)-secure pseudorandom bit generator and let $z = (z_0, \ldots, z_{q-1})$ be a bitstring of length q. This means that for any algorithm D which accepts q bits input and which runs in time t, it holds

$$|\Pr[1 \leftarrow D(z) \mid z \leftarrow G(\alpha), \alpha \in_U \mathrm{GF}(2)^r] - \Pr[1 \leftarrow D(z) \mid z \in_U \mathrm{GF}(2)^q]| \leq \varepsilon. \quad (4.4)$$

Using a standard argument, one can show that the success probability of any attacker A against the protocol using G deviates at most by ε from the success probability if the linear functions are characterized by uniformly and independently sampled bits. More precisely, let A denote any attacker against the $(n, k, L)^{\langle 80 \rangle}$-protocol which runs in time t at most. We define a corresponding security experiment Exp_A, which is equal to 1 if A has been successful. Moreover, we consider two games. In Game 0, the linear functions F_l have been determined by the output of G based on a secret seed, while in Game 1, they are characterized by independently and uniformly sampled bits. The latter corresponds to a situation where the linear functions are randomly chosen, as suggested in the context of the $(n, k, L)^{++}$-protocol. It follows from Eq. (4.4) that

$$|\Pr[\mathsf{Exp}_A = 1 \mid \mathsf{Game\ 0}] - \Pr[\mathsf{Exp}_A = 1 \mid \mathsf{Game\ 1}]| \leq \varepsilon.$$

Otherwise, A could be used directly as a distinguisher for telling apart random bits from outputs of G, hence violating Eq. (4.4). Summing up, if G is a (q, t, ε)-secure pseudorandom bit generator for a sufficiently small value ε, we can practically restrict to the case that the linear functions are randomly chosen. In particular, using a corresponding generator G yields at most a negligible difference w.r.t. the security against the algebraic attack from [KH11] and the active MITM attack from [KS09] in comparison to the original $(n, k, L)^{++}$-protocol.

4.3.2 Impact of Splitting the Connection Function

In this subsection, we investigate any impact on the security caused by splitting the connection function. As the algebraic attack from [KH11] in the *strong attacker* setting discussed above is independent of the connection function, the resistance against this attack remains unchanged. However, as we elaborate below, the situation is different for the active MITM attack against (n, k, L)-type protocols explained in [KS09]. This MITM attack has been called (x, y)-*equality* attack and was used to break, e.g., the CKK^2-protocol by Cichoń, Klonowski and Kutyłowski [CKK08]. We show that splitting the connection function implies an (for the attacker better) upper bound of about $2^{-n/4}$ for the success probability of this kind of attack against the $(n, k, L)^{\langle 80 \rangle}$-protocol. One consequence is that for the parameters suggested in Subsection 4.2.2 (e.g., $n = 64$, $k = 32$, $L = 16$), a reasonable level of security can be reached by running the protocol a few times (e.g., four independently executed rounds would reduce the upper bound to $2^{-16 \cdot 4}$ if $n = 64$).

We start with a description of the (x, y)-equality as given in [KS09]:

> "The aim of an (x, y)-equality attacker Eve is to generate two messages $w \neq w' \in \mathrm{GF}(2)^{n+k}$ and to efficiently test by MITM-access to the protocol if w and $w \oplus w'$ belong to the same linear subspace V_l for some $l \in [L]$. As described above [i.e., in [KS09]], such an attack can be used to efficiently compute specifications of the subspaces $V_1, \ldots, V_L$.
>
> Eve works in three phases:
>
> 1. Send a message $y \in \mathrm{GF}(2)^N$ to Bob and receive $w' = F_l(f(y, b'))$.
> 2. Observe a challenge $a \in \mathrm{GF}(2)^N$ sent by Alice to Bob.
> 3. Compute a value $x = x(y, w', a) \in \mathrm{GF}(2)^N$, send it to Bob, receive the message $w = F_r(f(x, b))$ and send $w \oplus w'$ to Alice.
>
> The success probability of the attack is given by the probability that Alice accepts $w \oplus w'$ if $l = r$." [KS09]

As pointed out above, for appropriately chosen n and authentication instances composed of multiple rounds, the new connection function of the $(n, k, L)^{\langle 80 \rangle}$-protocol still yields provable security against (x, y)-equality attacks (like the original $(n, k, L)^{++}$-protocol as shown in [KS09]). In the following, we identify $\mathrm{GF}(2)^4$ with the finite field $K = \mathrm{GF}(2^4)$ and denote by $+$ and $\cdot$ the addition and multiplication in K, respectively. Let the function value $f(a, b)$ for all $a, b \in \mathrm{GF}(2)^{n/2}$ be defined by

$$f(a, b) = \left(\left(a_1 b_1, a_1 b_1^3 \right), \ldots, \left(a_{n/8} b_{n/8}, a_{n/8} b_{n/8}^3 \right) \right),$$

where $a_i, b_i \in K$, $i = 1, \ldots, n/8$, are obtained by partitioning a and b into blocks of 4 bits, respectively (cf. Eq. (4.1) in Subsection 4.2.2). Remember that, according to the

specification of the $(n,k,L)^{\langle 80\rangle}$-protocol, the prover Bob will only reply to challenges a (and choose nonces b) which satisfy $a_i, b_i \neq 0$ for all $i = 1, \ldots, n/8$.

The proof of the following theorem works analogous to the proof of Theorem 1 in [KS09] for the connection function $f(a,b) = (ab, ab^3)$ of the $(n,k,L)^{++}$-protocol.

Theorem 4.1

The success probability of an (x,y)-equality attack against the $(n,k,L)^{\langle 80\rangle}$-protocol is at most $0.2^{n/8}$.

Proof: For a challenge $a \in (K^*)^{n/8}$, Alice accepts the response $w \oplus w' \in \mathrm{GF}(2)^{n+k}$ with

$$w = F_l(f(x,b)) = F_l\Big(\Big(x_1b_1, x_1b_1^3\Big), \ldots, \Big(x_{n/8}b_{n/8}, x_{n/8}b_{n/8}^3\Big)\Big),$$
$$w' = F_l(f(y,b')) = F_l\Big(\Big(y_1b'_1, y_1b_1'^3\Big), \ldots, \Big(y_{n/8}b'_{n/8}, y_{n/8}b_{n/8}'^3\Big)\Big)$$

and

$$\Big((u_1,v_1), \ldots, \Big(u_{n/8}, v_{n/8}\Big)\Big) = F_l^{-1}(w \oplus w')$$

if for all $i = 1, \ldots, n/8$ it holds that $(a_i^{-1}u_i)^3 = a_i^{-1}v_i$ or, equivalently, $u_i^3 = a_i^2 v_i$, where the GF(2)-linearity of F_l implies that $u_i = x_ib_i + y_ib'_i$ and $v_i = x_ib_i^3 + y_ib_i'^3$.

In consequence, for known $a, y \in (K^*)^{n/8}$ and secret $b, b' \in (K^*)^{n/8}$, Eve has to choose an element $x \in (K^*)^{n/8}$ such that

$$(x_ib_i + y_ib'_i)^3 = a_i^2\left(x_ib_i^3 + y_ib_i'^3\right)$$

or, equivalently,

$$\left(x_i + y_i\frac{b'_i}{b_i}\right)^3 = a_i^2\left(x_i + y_i\left(\frac{b'_i}{b_i}\right)^3\right) \tag{4.5}$$

for all $i = 1, \ldots, n/8$.

Let $c_i = b'_i \cdot b_i^{-1}$. Then Eq. (4.5) is equivalent to $P_i(x_i, c_i) = 0$, where, for $\tilde{x}, d \in K^*$, the polynomial $P_i(\tilde{x}, d)$ is defined as

$$P_i(\tilde{x}, d) = \tilde{x}^3 + (y_id)\,\tilde{x}^2 + \left(y_i^2d^2 + a_i^2\right)\tilde{x} + d^3\left(y_i^3 + y_ia_i^2\right).$$

For arbitrarily fixed $a_i, y_i \in K^*$, there are $|K^*| = 15$ different polynomials of type $P_i(\tilde{x}, d)$ w.r.t. the variable $\tilde{x}$, because for each $d \in K^*$, the coefficient y_id of $\tilde{x}^2$ takes a different value in K^*. Let $N_i(\tilde{x}) = \{d \in K^* \mid P_i(\tilde{x}, d) = 0\}$. As, also w.r.t. the variable d, the polynomial $P_i(\tilde{x}, d)$ is of degree 3, it holds that $|N_i(\tilde{x})| \leq 3$ for any $\tilde{x} \in K^*$.

Remember that in order for having Alice accept the response $w \oplus w'$ in state a, Eve needs to choose $x \in (K^*)^{n/8}$ such that Eq. (4.5) is satisfied for all $i = 1, \ldots, n$. This is equivalent to choosing $x \in (K^*)^{n/8}$ such that $c_i \in N_i(x_i)$ for all $i = 1, \ldots, n/8$. However, as the nonces b' and b are secret, Eve has no information about $c_i = b'_i \cdot b_i^{-1}$, $i = 1, \ldots, n/8$. In consequence, her success probability is bounded from above by

$$\prod_{i=1}^{n/8} \frac{3}{15} = 0.2^{n/8}. \qquad \square$$

4.4 Hardware Efficiency

Preamble. As pointed out in Section 4.1, this chapter represents a major rewrite (and a significant extension) of our underlying publication [AHK14]. The main reason for these modifications (three years after presenting [AHK14]) is our improved knowledge in the field of cryptographic hardware design, which we gained in the process of developing the new lightweight stream cipher Lizard (see Chapter 8). This experience helped us not only to increase the hardware efficiency of the implementation (which, in turn, allowed for using larger parameters and a full 80-bit secret key now), but it also lead to a more detailed analysis of the implied hardware costs. For example, in our original publication [AHK14], no information about the power consumption of the preliminary implementation of our scheme was given, which, as we know now, is a crucial metric when comparing lightweight cryptographic designs (see Subsection 2.3.4 and Subsection 8.5.1). In a nutshell, the update provided in this thesis, which we indicate by using the new name $(n, k, L)^{\langle 80 \rangle}$-protocol, now turns the 'feasibility study' provided in [AHK14] into a full-fledged protocol suggestion, which comes along with a concrete specification for suggested parameters (see Subsection 4.2.2), corresponding sets of test vectors (see Appendix 4.A), and a reference implementation written in Verilog (see Appendix 4.B). As a final remark with respect to the contents of this subsection, we would like to 'warn' the reader that in the description of the targeted hardware metrics and the applied techniques, there will be some redundancies with the information provided in Chapter 2 and Section 8.5. This redundancy is intended, as, given the extent of this thesis, we do not want that a reader who is solely interested in the new $(n, k, L)^{\langle 80 \rangle}$-protocol has to consult several other chapters, e.g., in order to find out what a *gate equivalent* is or how power consumption is properly estimated for ASIC implementations of cryptographic designs. We try, however, to keep the extent of such explanations at a level still tolerable to the more experienced reader.

In order to assess the efficiency of our hardware implementation and to allow for comparing the results with other cryptographic protocols, generally accepted cost metrics are needed. In Chapter 2, an overview over the most relevant of them in the field of ultra-constrained RFIDs (with a focus on ASICs) was given. In the following, we will

focus on *area*, *power*, and *timing* (in the form of clock cycles needed on the prover's side during a full authentication instance).[7] The information provided for FPGAs is rather a 'byproduct' of our Verilog implementation, because, as pointed out in Chapter 2, ASICs are the prevalent component for ultra-constrained devices in the price range of \$0.05–\$0.10 as targeted in this thesis. Nevertheless, for the sake of completeness, we wanted to include at least the area requirements of a corresponding FPGA implementation, because these numbers are given for a few other lightweight cryptographic schemes (such as the block cipher PRESENT [BKL+07]; see below) in the respective publications, as well.

Before presenting our implementation results for FPGAs and ASICs in the following two subsections, we would like to share our impression that despite the multitude of allegedly lightweight authentication protocols which have been suggested so far (see, e.g., [JW05], [BC08], [GRS08] or, more recently, [KPC+11]), none of the respective works contains details about an actual hardware implementation for ASICs. In contrast, newly introduced lightweight block ciphers like PRESENT [BKL+07] or KATAN [DCDK09] always come with an extensive assessment of their real-world hardware cost. This (and the fact that, as pointed out in Section 3.4, our paper [AHM14] has shown that currently there does not seem to be a single unbroken LPN-based authentication protocol feasible for ultra-constrained devices) is why in Subsections 4.4.1 and 4.4.2, we compare the numbers of the $(n, k, L)^{\langle 80 \rangle}$-protocol rather with those of PRESENT (key length: 80 bits, block length: 64 bits), assuming its use as part of the following simple authentication scheme already described in Section 3.3: both parties share a secret 80-bit key for PRESENT and in order to prove his identity, the prover needs to correctly encrypt a random 64-bit challenge provided the verifier.

We hope that our hardware results presented in this subsection will encourage other designers of lightweight authentication protocols to also go trough the process of actually implementing their schemes to allow for easier efficiency comparison in the future.

4.4.1 The $(n, k, L)^{\langle 80 \rangle}$-Prover on ASICs

As pointed out previously, ASICs are a typical component in the context of RFID applications. They are (ex ante) tailored to a very specific need and subsequently produced in large quantities, allowing for low unit cost and making them perfectly suitable for pervasive devices like ultra-constrained RFID tags. In the field of ASICs, area is usually measured in μm^2. However, as area requirements in μm^2 strongly depend on the used standard cell library, it is common to use a metric called *gate equivalents*

[7] Note that this is not cherry picking. Instead, other metrics like *transmission bandwidth* (cf. Subsection 2.3.1), *delay* (cf. Subsection 2.3.7), *random number generation* (cf. Subsection 2.3.8), and *key storage* (cf. Subsections 2.3.9 and 2.3.10) are obviously not a bottle neck w.r.t. the $(n, k, L)^{\langle 80 \rangle}$-protocol, so it would not be of much use to treat them here. The only exception is *energy* (cf. Subsection 2.3.5). However, as pointed out previously, we are focusing on passively powered RFIDs in this thesis, where energy consumption over time is irrelevant.

(GE) instead. In short, one GE is equivalent to the area of a two-input drive-strength-one NAND gate. This at least allows for a rough comparison of area requirements derived using different technologies (see Subsection 2.3.3 for further details).

For improved comparability of results, we employ the UMCL18G212T3 (0.18 μm, 1.8 V) standard cell library that was also used by Poschmann in [Pos09] for implementing the block cipher PRESENT. Our results (see Table 4.1) are obtained via Cadence Encounter RTL Compiler RC12.22 [Cad17] and are based on the netlist generated through the command `synthesize -to_placed -effort high`. As common in the field of ultra-constrained devices (see, e.g., [Fel07]), we target a 100 kHz clock (cf. Subsection 2.3.6) and employ clock gating (cf. Subsection 2.3.4). The switching activity for power estimation (recorded with Mentor ModelSim SE-64 6.5b [Men17] and fed back to RTL compiler) covers the generation of an authentication token of length $n + k$ bits at a clock rate of 100 kHz and includes loading of the inputs (key κ, challenge a, nonce b, secret function index l) as well as the state initialization of the underlying self-shrinking generator. To improve the accuracy of the results, switching activity for 25 different random input combinations is considered and the arithmetic mean of the respective power estimations is computed. For all power values given in Table 4.1, the largest deviation of a single estimation from the computed average was below one percent.[8]

It is important to note that, as also pointed out in Subsections 2.3.4 and 8.5.1, while the area requirement of cryptographic designs can be compared over different standard cell libraries by using the measure gate equivalents, "[p]ower cannot be scaled reliably between different processes and libraries" [GB08]. Consequently, despite the fact that the power numbers of the $(n, k, L)^{\langle 80 \rangle}$-protocol as presented in Table 4.1 are below those of a serialized (and, hence, least area- and power-consuming) implementation of PRESENT (see below), we will not directly claim that our protocol is superior in this respect. However, as the employed standard cell library is the same, it is safe to say that the numbers are at least in the same ballpark.

The values in Table 4.1 show that w.r.t. area requirements, power consumption and communication complexity, both variants of our new $(n, k, L)^{\langle 80 \rangle}$-protocol operate well within the limits of ultra-constrained RFIDs as described in Chapter 2. Only the authentication time may, at first sight, seem to be a problem when compared to the upper bound of 150 ms told to us by several industrial sources (cf. Subsection 2.3.2). More precisely, at a clock speed of 100 kHz, the variant with $n = 128$ would need about 820 ms to perform its two rounds, while the the variant with $n = 64$ would still need about 493 ms to perform its four rounds. One may now argue that these authentication times will

[8] Please note that this deviation below *one percent* is not a copy-and-paste error from Subsection 8.5.1, where the power analysis for our new stream cipher Lizard is presented and where the power consumptions for different, randomly chosen inputs are similarly close to the average. This is, in fact, not surprising, because in both cases, the dominating component is an FSR which, for reasons of cryptographic security, is supposed to 'behave rather randomly' over time, leading to similar estimated power consumptions for different inputs.

Table 4.1: Hardware results w.r.t. ASICs for a clock speed of 100 kHz. *Clock Cycles* denotes the total number of clock cycles needed on the prover's side to perform a full authentication instance consisting of multiple rounds. Due to the nature of the self-shrinking generator used in our implementation, the values in the respective column may vary slightly for different keys κ and function indices l.

Parameters				**ASIC**			**Com. (Tag)**
n	k	L	Rnds.	Area [GE]	Power [nW]	Clock Cycles	Bits IN/OUT
128	32	16	2	1925	2194	$\approx 82\,000$	128/320
64	32	16	4	1366	2138	$\approx 49\,300$	128/384

still be tolerable for certain applications, or simply assume devices with a slightly higher clock speed than 100 kHz. However, for fairness reasons, we feel bound to the limits as given in Subsection 2.3.2 when assessing the suitability of a cryptographic scheme for ultra-constrained RFIDs. Fortunately, for the $(n, k, L)^{\langle 80 \rangle}$-protocol, authentication times can, in fact, be improved quite easily and efficiently. This is due to the fact that the dominating component w.r.t. token generation speed is the underlying LFSR-based self-shrinking generator, for which, as explained in Subsection 4.2.2, parallelization is rather straightforward and requires only few additional hardware gates, owing to the very simple feedback function. In particular, the feedback polynomial which we chose for our concrete instantiations provides, despite its good diffusion properties, the possibility for parallelization up to the degree 15 (where, admittedly, a maximum of 14 makes more sense in practice, due to the '2-bit-based' nature of the self-shrinking generator).[9] Consequently, as our current instantiations only use parallelization of degree 2 (cf. Subsection 4.2.2 and the reference implementation in Appendix 4.B), reducing the numbers of clock cycles given in Table 4.1 at a further factor of 7 is possible at very moderate hardware costs. Hence, even the larger $(n, k, L)^{\langle 80 \rangle}$-protocol variant with $n = 128$ can be considered for general application in the context of ultra-constrained RFIDs.[10]

[9] The degree of straightforwardly possible parallelization is determined by the position of the 'youngest' (w.r.t. the generator's internal bitstream) tap, which, in the case of our feedback polynomial $x^{84} + x^{82} + x^{62} + x^{49} + x^{30} + x^{15} + 1$, corresponds to the linear monomial x^{15}.

[10] Note that, while the underlying LFSR of our self-shrinking generator can be parallelized extremely efficiently up to degree 15, some additional logic in our protocol implementation would be required for parallelization degrees larger than 2, in order to handle the fact that, e.g., a self-shrinking generator parallelized at degree 14 can output between 0 and 7 bits per clock cycle. In the case of our instantiation with $n = 64$, the margin between the required 1366 GE and the virtual limit of 2000 GE is so large that the additional logic required for processing the self-shrinking generator's variable-length output can definitely be realized still within our strict boundaries for ultra-constrained RFIDs. For the instantiation with $n = 128$, on the other hand, it might be an alternative to use an easy parallelizable NFSR (like that of Grain v1 [HJM06]; see Subsection 5.2.4) instead of the self-shrinking generator, in order to save the logic for handling variable-length output and, thus, stay close to the postulated 2000 GE limit.

So, due to the fact that, as pointed out above, none of the (unbroken) LPN-based authentication protocol suggestions seems to be suitable for such devices, to the best of our knowledge, the $(n, k, L)^{\langle 80 \rangle}$-protocol is actually the first dedicated authentication approach feasible for ultra-constrained RFIDs. Nevertheless, as a matter of candor, we also need to compare our scheme to the classical (lightweight) block cipher-based authentication approaches. As explained in Section 3.3, according to [RPLP08], a serialized implementation of PRESENT can be realized at 1080 GE and requires 563 clock cycles to encrypt a single 64-bit block of data. The corresponding power consumption is given as 2.52 μW for a clock speed of 100 kHz. Apart from the slightly higher power requirements, these numbers may suggest that block cipher-based authentication is still to be preferred. And, in fact, we do not claim that our approach is superior. However, it offers an important feature that the assumed straightforward application of a lightweight block cipher like PRESENT does not: *privacy preservation*. More precisely, due to the use of the secret nonce $b \in \{0,1\}^{n/2}$ randomly chosen by the prover (cf. Subsection 4.2.2), identical challenges will be answered with different authentication tokens. In Section 3.1, we described in a basic setting how this can also be achieved for block-cipher based constructions. Note, however, that the corresponding approach would effectively half the challenge space to size 2^{32} for a cipher with block length 64 bits like PRESENT. In order to circumvent this limitation, it would additionally be required to implement a suitable mode of operation (i.e., not electronic codebook mode (ECB mode)), which comes at significant additional costs. The $(n, k, L)^{\langle 80 \rangle}$-protocol instantiation with $n = 128$ (resp. $n = 64$), on the other hand, offers a challenge space of size 2^{64} (resp. 2^{32}) and already includes privacy-preservation capabilities. Given that, as explained above, both parameter choices can satisfy the limits of ultra-constrained RFIDs, we hence consider our protocol a viable alternative to prevalent block cipher-based constructions.

4.4.2 The $(n, k, L)^{\langle 80 \rangle}$-Prover on FPGAs

In order to allow for an easy comparison on FPGAs, we target the same platform which Poschmann used in [Pos09] for the encryption unit of PRESENT: the *Spartan-3 XC3S400* (Package FG456, Speed -5) from Xilinx [Xil17]. Synthesis and implementation (including *Place & Route*) is performed using Xlinix ISE Design Suite 14.7 and the Verilog code for our modules is the same as that for the ASIC implementation discussed in the previous Subsection 4.4.1 (see also our reference implementation for $n = 128$ in Appendix 4.B).

While actually aimed at ASICs, the area footprint of the $(n, k, L)^{\langle 80 \rangle}$-protocol is also very moderate on FPGAs. More precisely, it amounts to 237 FFs and 307 4-input LUTs for the instantiation with $n = 128$, $k = 32$, $L = 16$ and to 165 FFs and 236 4-input LUTs for the instantiation with $n = 64$, $k = 32$, $L = 16$.[11] This compares to 152 FFs and

[11] In a nutshell, on FPGAs, combinatorial logic is realized through lookup tables (LUTs). For example, a 4-input LUT can be programmed to realize an arbitrary Boolean function over the input space $\{0,1\}^4$. More complex logic is then realized by appropriately combining these building blocks.

253 LUTs given in [Pos09] for the encryption unit of PRESENT on the same platform and with an *espresso*-optimized [Uni94] S-box. Without this latter optimization, the respective numbers are 154 FFs and 350 LUTs.

Overall, these numbers suggest that not only on ASICs, but also on FPGAs, the $(n, k, L)^{\langle 80 \rangle}$-protocol can compete with prevalent block cipher-based constructions.

4.5 Conclusion and Outlook

In this chapter, we introduced the $(n, k, L)^{\langle 80 \rangle}$ authentication protocol, which is a modification of the already investigated $(n, k, L)^{++}$-protocol made in order to improve hardware efficiency. The respective implementations confirm the suitability of our protocol for use cases which demand for low hardware size and power consumption, e.g., ultra-constrained RFID systems, making it interesting for practice. Moreover, the fact that the security of the $(n, k, L)^{\langle 80 \rangle}$-protocol relies on a different paradigm than the alternative approaches based on block ciphers or the LPN problem, i.e., the random selection of secret functions, makes our new scheme likewise interesting for the cryptography community. A further advantage as compared to, e.g., straightforward block cipher-based approaches, is that privacy-preservation capabilities are already inherent in the $(n, k, L)^{\langle 80 \rangle}$-protocol.

One major modification w.r.t. the original $(n, k, L)^{++}$-protocol is that the internal linear functions are now generated by a bitstream generator G in order to save memory regarding key storage. Our analysis shows that while using a single publicly known LFSR renders the protocol insecure (in a strong adversary model), deploying a secure pseudorandom bit generator is sufficient. For our concrete protocol specifications with parameters $n = 128$, $k = 32$, $L = 16$ and $n = 64$, $k = 32$, $L = 16$, respectively, we used the self-shrinking generator by Meier and Staffelbach [MS94] on top of a simple maximum-length LFSR, as it provably generates a bitstream with high linear complexity and hence protects against the aforementioned attack targeting protocol instantiations that rely on a mere LFSR.

However, it remains an open question whether other, intermediate approaches, e.g., using an NFSR, might be viable alternatives. In general, given that the underlying problem is relatively new, its hardness and possible connections to other problems need to be investigated further. Moreover, despite the popularity of lightweight authentication protocols, it turns out that only few actual implementations exist. This aspect represents an important next step towards a better understanding and comparison of existing design approaches.

In the following chapters of this thesis, we will leave the path of searching for dedicated authentication protocols and, instead, treat a fundamental question which arose to us while designing the $(n, k, L)^{\langle 80 \rangle}$-protocol: 'Why use a bitstream generator only to produce the specifications of the secret functions, but not for generating the authentication token right away?' To this end, in the following Chapter 5, we will briefly revisit some prominent

examples of classical stream ciphers. Afterwards, in Chapter 6, we are then going to study the most recent approaches for designing so-called *small-state stream ciphers*. Based on the theoretical considerations in Chapter 7, we will subsequently introduce our own lightweight stream cipher called Lizard in Chapter 8. Finally, in Chapter 9, we will forge a bridge to this and the previous Chapter 3, by suggesting how Lizard can be used to realize hardware-efficient, privacy-preserving authentication on ultra-constrained RFIDs.

Appendix 4.A Test Vectors

Key κ (80 bits), secret function index $l \in \{1, \ldots, 16\}$ as a bitstring (4 bits; $l = 1 \to 0000$, $l = 2 \to 0001$ etc.), challenge a ($n/2$ bits), nonce b ($n/2$ bits), and the corresponding $(n + k)$-bit authentication token w in hexadecimal notation. To avoid ambiguity, note that, e.g., the key

`0x01234FFFFFFFFFFFFFFF`

corresponds to

$$(\kappa_0, \ldots, \kappa_{79}) = (0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, \ldots, 1)\,.$$

Similarly, for a 96-bit authentication token w, the example

`0x010000000000000000000000`

would mean that the first seven bits of w (i.e., $w_0, \ldots, w_6$) are zero, followed by a one and 88 more zeros.

4.A.1 The $(n, k, L)^{\langle 80 \rangle}$-Protocol with Parameters $n = 128$, $k = 32$, $L = 16$

```
Key:            0x00000000000000000001
Function Index: 0x0
Challenge:      0x1F1F1F1F1F1F1F1F
Nonce:          0xF1F1F1F1F1F1F1F1
Token:          0xFF2740B656B0E63B7336BE440B45C0E908A864B7

Key:            0xFFFFFFFFFFFFFFFFFFFF
Function Index: 0xF
Challenge:      0xF1F1F1F1F1F1F1F1
Nonce:          0x1F1F1F1F1F1F1F1F
Token:          0x77CCF6242487A643275993F4B681C86373F43630

Key:            0x0123456789ABCDEF0123
Function Index: 0xA
Challenge:      0x123456789ABCDEF1
Nonce:          0x21FEDCBA98765432
Token:          0x98C6E14FB66F42C47B440FE5D3A68998BB8D8A06
```

4.A.2 The $(n, k, L)^{\langle 80 \rangle}$-Protocol with Parameters $n = 64$, $k = 32$, $L = 16$

```
Key:            0x00000000000000000001
Function Index: 0x0
Challenge:      0x1F1F1F1F
Nonce:          0xF1F1F1F1
Token:          0x59590415D03CBA2BE11876FC

Key:            0xFFFFFFFFFFFFFFFFFFFF
Function Index: 0xF
Challenge:      0xF1F1F1F1
Nonce:          0x1F1F1F1F
Token:          0xE9E59C93591BC71F081CBF16

Key:            0x0123456789ABCDEF0123
Function Index: 0xA
Challenge:      0x12345678
Nonce:          0x1FEDCBA9
Token:          0x367A40C2532B1173B446C716
```

Appendix 4.B Reference Implementation

Listing 4.1 shows our reference implementation in Verilog for the $(n, k, L)^{\langle 80 \rangle}$-prover with parameters $n = 128$, $k = 32$, $L = 16$. An implementation for the parameter combination $n = 64$, $k = 32$, $L = 16$, whose hardware costs are also presented in Section 4.4, works completely analogous. The major difference implied by choosing $n = 64$ is the smaller size of the challenge a and the nonce b, each of which then comprises of 32 bits (instead of 64 bits for $n = 128$). The necessary adaptions w.r.t. the indices and the counter limits (e.g., for matrix multiplication) are obvious. Hence, for the sake of shortness, we only give the reference implementation for the case $n = 128$ in this thesis.

The main module `nkl80_128` uses synchronous reset (taking one clock cycle) and all operations are triggered by the rising edge of the clock. Setting `reset` high resets the module. Once `reset` is set low again, state initialization begins. After state initialization has finished, the values of `key` (secret key), `a` (challenge), `b` (nonce), and `l` (secret function index) do not need to be present on the corresponding wires any longer, i.e., outer modules driving these wires, e.g., via registers, can now use these registers for other purposes. Once the module has completed state initialization, it enters the authentication token generation phase. Each time a new token bit is ready to be read (via `tokenBit`) by an outer module, `tokenBitFlag` is raised from from low to high. At the next rising edge of the clock, `tokenBitFlag` is set to low again and the output over the wire `tokenBit` becomes invalid. In order not to leak sensitive information about the computation of the

token bits to outer modules, the signal provided by the wire `tokenBit` is constantly low (i.e., zero) during intermediate steps of the computation. Once the full token has been generated (in this case, $128 + 32 = 160$ bits), `doneFlag` is raised from low to high and no further operations are performed. In particular, also the underlying LFSR (the module `mlfsr84para`) is paused in order not to waste precious power.

Listing 4.1: Reference implementation in Verilog of the $(n, k, L)^{\langle 80 \rangle}$-prover with parameters $n = 128$, $k = 32$, $L = 16$.

```
1  `timescale 1us / 1ps
2
3  //************************************
4
5  module nkl80_128(
6    input wire clk,
7    input wire reset,
8    input wire [0:79] key,
9    input wire [0:63] a,
10   input wire [0:63] b,
11   input wire [3:0] l,
12   output wire tokenBit,
13   output wire tokenBitFlag,
14   output wire doneFlag
15   );
16
17 ////////////////
18
19 reg [1:0] proverFSM;
20 //                                  v lfsr_enable
21 localparam S_INIT              = 2'b01;
22 localparam S_MULT_NEXT_ROW     = 2'b10;
23 localparam S_MULT_SCALAR_PROD = 2'b11;
24 localparam S_DONE              = 2'b00;
25
26 ////////////////
27
28 assign tokenBitFlag = (proverFSM == S_MULT_NEXT_ROW);
29 assign doneFlag = (proverFSM == S_DONE);
30
31 wire lfsr_enable;
32 assign lfsr_enable = proverFSM[0];
33
34 ////////////////
35
36 // used for
37 // - step counting during mixing
38 // - row counting during token generation
39 reg [7:0] shared_ctr;
40
41 // column counter for matrix multiplication
```

```
42 reg [6:0] j_ctr;
43
44 wire done_mixing;
45 assign done_mixing = (shared_ctr == 8'd127) ? 1'b1 : 1'b0;
46
47 wire done_scalarProduct;
48 assign done_scalarProduct = (j_ctr == 7'd127) ? 1'b1 : 1'b0;
49
50 wire done_matrixMult;
51 assign done_matrixMult = (shared_ctr == 8'd159) ? 1'b1 : 1'b0;
52
53 ////////////////
54
55 // used to store the intermediate result of the
56 // token bit computation, will finally contain the
57 // actual token bit
58 reg tokenBit_temp;
59
60 // is set to false (0) if the challenges do not fulfill the
61 // requirements explained in Chapter 4 of the thesis
62 // -> leads to all-zero token
63 reg securityStatusFlag;
64
65 // done_scalarProduct is included in order not to leak
66 // secret information about the computation process of
67 // the token bit to outer modules
68 assign tokenBit = done_scalarProduct & securityStatusFlag & tokenBit_temp;
69
70 ////////////////
71
72 // the value f(a,b) serves as input for the secret
73 // function F_l; see Sec. 4.2.2 of the thesis
74
75 reg [0:127] f_ab;
76
77 wire [3:0] leftFactor;
78 assign leftFactor = f_ab[124:127];
79
80 wire [3:0] rightFactor;
81 assign rightFactor = b[(shared_ctr[5:2] * 4) +: 4];
82
83 wire [3:0] f_ab_chunk;
84
85 multgf16 mult1(
86   .factor1(leftFactor),
87   .factor2(rightFactor),
88   .product(f_ab_chunk)
89   );
90
91 ////////////////
```

```
92
93  // a parallelized (2 steps per clock tick) version of the
94  // 84-bit maximum-length LFSR which our self-shrinking
95  // generator is based on; see Sec. 4.2.2 of the thesis
96
97  wire lfsr_outputBit1;
98  wire lfsr_outputBit2;
99
100 mlfsr84para lfsr(
101   .clk(clk),
102   .lfsr_reset(reset),
103   .lfsr_enable(lfsr_enable),
104   .key(key),
105   .l(l),
106   .lfsr_outputBit1(lfsr_outputBit1),
107   .lfsr_outputBit2(lfsr_outputBit2)
108   );
109
110 ////////////////
111
112 always @(posedge clk)
113   begin
114     if (reset)
115       begin
116         shared_ctr <= 8'd0;
117         j_ctr <= 7'd0;
118
119         securityStatusFlag <= 1'b1;
120         tokenBit_temp <= 1'b0;
121
122         proverFSM <= S_INIT;
123       end
124     else if (!doneFlag)
125       begin
126         if (proverFSM == S_INIT)
127           begin
128             // computation of (a'b',a'b'^3) over GF(2^4)
129             // in four clock ticks
130             if (shared_ctr[7:6] == 2'b00)
131               case (shared_ctr[1:0])
132                 2'b00 :
133                   begin
134                     if (shared_ctr > 0)
135                       begin
136                         f_ab[0:119] <= f_ab[8:127];
137                       end
138
139                     // (0,a')
140                     f_ab[124:127] <= a[(shared_ctr[5:2] * 4) +: 4];
141                   end
```

```
142                 2'b01 :
143                   begin
144                     // check for security violation due to
145                     // either a' == 0 or b' == 0
146                     // (and, hence, f_ab_chuck == 0 at this point)
147                     if (f_ab_chunk == 4'b0000)
148                       securityStatusFlag <= 1'b0;
149
150                     // (a'b',a'b')
151                     f_ab[120:123] <= f_ab_chunk;
152                     f_ab[124:127] <= f_ab_chunk;
153                   end
154                 2'b10 :
155                   begin
156                     // (a'b',a'b'^2)
157                     f_ab[124:127] <= f_ab_chunk;
158                   end
159                 2'b11 :
160                   begin
161                     // (a'b',a'b'^3)
162                     f_ab[124:127] <= f_ab_chunk;
163                   end
164               endcase
165
166               if (done_mixing)
167                 begin
168                   shared_ctr <= 8'd0;
169
170                   proverFSM <= S_MULT_SCALAR_PROD;
171                 end
172               else
173                 shared_ctr <= shared_ctr + 1;
174           end
175
176         if (proverFSM == S_MULT_NEXT_ROW)
177           begin
178             if (done_matrixMult)
179               proverFSM <= S_DONE;
180             else
181               begin
182                 shared_ctr <= shared_ctr + 1;
183                 j_ctr <= 7'd0;
184
185                 tokenBit_temp <= 1'b0;
186
187                 proverFSM <= S_MULT_SCALAR_PROD;
188               end
189           end
190
191         if (proverFSM == S_MULT_SCALAR_PROD)
```

```
192             begin
193               if (lfsr_outputBit1 == 1'b1)
194                 begin
195                   tokenBit_temp <= tokenBit_temp ^ (lfsr_outputBit2 &
                          f_ab[j_ctr]);
196
197                   if (done_scalarProduct)
198                     proverFSM <= S_MULT_NEXT_ROW;
199                   else
200                     j_ctr <= j_ctr + 1;
201                 end
202             end
203         end
204     end
205
206 ////////////////
207
208 endmodule
209
210 //************************************
211
212 module multgf16(
213     input wire [3:0] factor1,
214     input wire [3:0] factor2,
215     output wire [3:0] product
216     );
217
218 ////////////////
219
220 wire [3:0] x, y;
221
222 assign x = factor1;
223 assign y = factor2;
224
225 assign product[0] = (x[0] & y[0]) ^ (x[1] & y[3]) ^ (x[2] & y[2]) ^ (x[3]
      & y[1]);
226 assign product[1] = (x[0] & y[1]) ^ (x[1] & y[0]) ^ (x[1] & y[3]) ^ (x[2]
      & y[2]) ^ (x[3] & y[1]) ^ (x[2] & y[3]) ^ (x[3] & y[2]);
227 assign product[2] = (x[0] & y[2]) ^ (x[1] & y[1]) ^ (x[2] & y[0]) ^ (x[2]
      & y[3]) ^ (x[3] & y[2]) ^ (x[3] & y[3]);
228 assign product[3] = (x[0] & y[3]) ^ (x[1] & y[2]) ^ (x[2] & y[1]) ^ (x[3]
      & y[0]) ^ (x[3] & y[3]);
229
230 ////////////////
231
232 endmodule
233
234 //************************************
235
236 module mlfsr84para(
```

```
237   input wire clk,
238   input wire lfsr_reset,
239   input wire lfsr_enable,
240   input wire [0:79] key,
241   input wire [3:0] l,
242   output wire lfsr_outputBit1,
243   output wire lfsr_outputBit2
244   );
245
246 ////////////////
247
248 reg [0:83] lfsr_state;
249
250 // corresponding to the primitive feedback polynomial
251 // x^84 + x^82 + x^62 + x^49 + x^30 + x^15 + 1 (-> feedbackBit1)
252 // resp. a shifted version of its taps (-> feedbackBit2)
253 assign lfsr_feedbackBit1 = lfsr_state[(84 - 84)] ^ lfsr_state[(84 - 82)] ^
      lfsr_state[(84 - 62)] ^ lfsr_state[(84 - 49)] ^ lfsr_state[(84 - 30)]
      ^ lfsr_state[(84 - 15)];
254 assign lfsr_feedbackBit2 = lfsr_state[(84 - 84) + 1] ^ lfsr_state[(84 -
      82) + 1] ^ lfsr_state[(84 - 62) + 1] ^ lfsr_state[(84 - 49) + 1] ^
      lfsr_state[(84 - 30) + 1] ^ lfsr_state[(84 - 15) + 1];
255
256 assign lfsr_outputBit1 = lfsr_state[0];
257 assign lfsr_outputBit2 = lfsr_state[1];
258
259 always @(posedge clk)
260   begin
261     if (lfsr_reset)
262       begin
263         lfsr_state <= {l,key};
264       end
265     else if (lfsr_enable)
266       begin
267         lfsr_state[0:81] <=  lfsr_state[2:83];
268         lfsr_state[82] <= lfsr_feedbackBit1;
269         lfsr_state[83] <= lfsr_feedbackBit2;
270       end
271   end
272
273 ////////////////
274
275 endmodule
276
277 //*************************************
```

If you wait by the [key]stream long enough,
the bodies of your enemies will float by.

unknown TMD-TO cryptanalyst citing
Sun Tzu's 'The Art of War'

CHAPTER 5

Classical Stream Ciphers

ABSTRACT

In this chapter, we leave the path of searching for dedicated authentication protocols suitable for ultra-constrained RFIDs (cf. Chapter 2) and, instead, lay the foundation for treating a fundamental question which arose to us while designing the $(n, k, L)^{\langle 80 \rangle}$-protocol introduced in Chapter 4: 'Why use a bitstream generator only to produce the specifications of the secret functions, but not for generating the authentication token right away?'

To this end, as a first step, we revisit some prominent examples of classical stream ciphers here and, in particular, analyze their respective state initialization algorithms. Later, in Chapter 7, the corresponding insights will then serve as a basis for our new stream cipher design principle named LIZARD-construction.

Declaration of Origin: This chapter is based on the paper *Stream Cipher Operation Modes with Improved Security against Generic Collision Attacks* [HK15] and the paper *On Stream Ciphers with Provable Beyond-the-Birthday-Bound Security against Time-Memory-Data Tradeoff Attacks* [HK18], both written together with Matthias Krause and published at the *Cryptology ePrint Archive* and in *Cryptography and Communications* (Springer US journal), respectively.

5.1 Introduction

Stream ciphers have a long history when it comes to protecting digital communication. In 1987, Ronald L. Rivest designed RC4 [Sch95], which was later used in SSL/TLS [DR08] and the wireless network security protocols WEP [Ins97] and TKIP (often called WPA) [Ins04]. Other well-known stream cipher examples are E_0 of the Bluetooth standard [Blu14] and A5/1 of GSM [BGW99]. Unfortunately, E_0 and A5/1 have been shown to be highly insecure (see, e.g., [LMV05] and [BB06]) and RC4 also shows severe vulnerabilities, which led to its removal from the TLS protocol [Pop15] and rendered other protocols like WEP insecure [FMS01]. In 2004, the eSTREAM project [ECR08] was started in order to identify new stream ciphers for two application profiles:

> "Profile 1 contains stream ciphers more suitable for software applications with high throughput requirements. Profile 2 stream ciphers are particularly suitable for hardware applications with restricted resources such as limited storage, gate count, or power consumption." [ECR08]

The competition ended in 2008 and, for Profile 2, the resulting eSTREAM portfolio contained four ciphers, one of which (i.e., F-FCSR-H v2 [BAL06]) was removed shortly after due to new cryptanalytic results [HJ08]. After the latest review in 2012 [BBV12], the remaining three ciphers for Profile 2 are still part of the portfolio: Grain v1 [HJM06], MICKEY 2.0 [BD06], and Trivium [CP05]. In this chapter, we will revisit two of them, namely Grain v1 and Trivium, as they archetypically represent two different extremes of the common tradeoff between inner state size and complexity of the involved operations. This tradeoff will play an important role in the context of *small-state stream ciphers* as discussed in the following Chapter 6.

From a technical perspective, stream ciphers are symmetric encryption algorithms intended for protecting, in an online manner, *plaintext bitstreams* X which have to pass an insecure channel. Encryption is performed via bitwise addition of a *keystream* S to X, which depends on a *secret symmetric key* k and a *public initialization vector* IV. The legal recipient, who also knows k, decrypts the *encrypted bitstream* $Y = X \oplus S$ by generating S and computing $X = Y \oplus S$. Straightforward, non-parallelized implementations of stream ciphers typically produce one keystream bit per clock cycle, thus allowing to encrypt and transmit data 'as it appears'. Stream ciphers are hence particularly suited for time-critical communication scenarios where potentially small amounts of data need to be processed instantly (such as over-the-air communication for mobile devices).[1]

In practice, the communication between legal users is usually organized in *sessions*, where in the first phase of each session, the secret *session key* k is generated by executing a *key establishment protocol* (see, e.g., Subsection 5.2.1 for that of E_0 in Bluetooth). This

[1] While block ciphers can be used in such scenarios, too, they are usually less efficient, because for data pieces smaller than their block size, *padding* is required, which leads to a wasteful overhead in terms of computation and communication.

session key generation phase will not be treated in detail in this thesis, as it represents a large, independent research area of its own. Note that a session can, e.g., be a phone call, where at the beginning of the call, a key establishment protocol between the mobile phone and the nearest base station is performed.

Following [BG07], each stream cipher is associated with a well-defined set of inner states and its keystream generation process can be divided into the following two phases:

(A) The *key and IV setup phase*, where an initial state is derived from the secret session key k and an initialization vector IV.

(B) The *keystream generation phase*, in which the keystream is generated based on the initial state derived in phase (A).

In this thesis, we focus on what we call *KSG-based stream ciphers*, for which the main algorithmic component for performing phases (A) and (B) is a *keystream generator (KSG)*. KSGs are clock-controlled devices which can be formally specified by finite automata, defined by an inner state length n, the set of inner states $\{0,1\}^n$, a state update function $\pi : \{0,1\}^n \longrightarrow \{0,1\}^n$, and an output function $outbit : \{0,1\}^n \longrightarrow \{0,1\}$. Starting from an initial state $q_{\text{init}} = q_0$, in each clock cycle $t \geq 0$, the KSG produces an output bit $z_t = outbit(q_t)$ and changes the inner state according to $q_{t+1} = \pi(q_t)$. The keystream $S(q_{\text{init}})$ corresponding to the initial state q_{init} is defined by concatenating all the outputs bits $z_0 z_1 z_2 \cdots$.

As pointed out above, the key and IV setup phase (A) of a KSG-based stream cipher is performed by a KSG-based *state initialization algorithm*, which computes, from the session key k and the initialization vector IV, the initial state q_{init}. It typically contains the following two subphases:

(A.1) The *loading phase* defines how the session key k and the initialization vector IV are loaded into the inner state registers and results in a *loading state* $q_{\text{load}} = q_{\text{load}}(k, IV)$.

(A.2) The *mixing phase* runs an appropriate KSG-based *mixing algorithm*

$$MIX : \{0,1\}^n \longrightarrow \{0,1\}^n$$

on q_{load} and results in a state $q_{\text{mixed}} = MIX(q_{\text{load}})$.

The aim of the mixing phase (A.2) is to generate a sufficient amount of diffusion, confusion, high algebraic degree etc. in the dependencies of the initial state bits from the session key bits and the IV bits. In many cases, an essential part of the mixing algorithm consists in running the KSG a certain number of times without producing keystream bits. Moreover, as we will show in Section 5.3, for many ciphers (such as Trivium, Grain v1, A5/1, E_0) it holds that $q_{\text{init}} = q_{\text{mixed}}$. Furthermore, we will also explain that for Trivium, Grain v1, and A5/1, the mixing algorithm MIX can be inverted efficiently.

One can distinguish the following two *operation modes of stream ciphers*. In the *one-stream mode*, the key and IV setup phase (A) is performed only once at the beginning of the session and produces an initial state $q_{\text{init}} = q_{\text{init}}(k, IV)$. The corresponding keystream $S = S(q_{\text{init}})$ is used for the whole session. Note that due to their extremely large limits (e.g., 2^{64} bits for Trivium) on the amount of keystream generated under a single key/IV pair, Trivium and Grain v1 can be considered to work in one-stream mode.[2]

In contrast to this, in the *packet mode*, the communication and encryption process during a session is divided into packet steps $i = 1, 2, \ldots$, where in each packet step, a piece of message of a certain maximal packet length R is encrypted and sent. Corresponding to this, the keystream of a session is the concatenation of the keystream packets $S^1, S^2, \ldots$, where for all $i \geq 1$, S^i denotes the keystream packet generated in packet step i. Corresponding stream cipher instantiations are equipped with a mechanism providing, for each packet step i, an initialization vector IV^i (such as the frame counter in A5/1; cf. Subsection 5.2.2). Each packet step i starts with performing the key and IV setup phase (A), yielding a *packet initial state* $q^i_{\text{init}} = q^i_{\text{init}}(k, IV^i)$, followed by the generation of the keystream packet S^i, which is defined to be the prefix of length R of $S(q^i_{\text{init}})$. Typical examples for stream ciphers operating in packet mode are the Bluetooth cipher E_0 (see Subsection 5.2.1) and the GSM cipher A5/1 (see Subsection 5.2.2). Also note that in the network protocols of many important digital communication scenarios, data streams are transmitted packet-wise (Ethernet, WLAN, Bluetooth, cellular networks etc.). It thus seems natural to consider stream ciphers running packet mode and, in particular, to look for corresponding design optimizations (see Section 8.1 for more examples of stream ciphers used in packet mode and for further information about their practical relevance).

Structure of this chapter: In Section 5.2, we revisit some of the most important *classical stream ciphers*, where the term 'classical' is meant to distinguish them from *small-state stream ciphers* as discussed in the following Chapter 6. Furthermore, in Subsection 5.2.5, we also briefly describe how block ciphers can be used to create (though not bitwise working) stream ciphers. In Section 5.3, we then analyze how the state initialization algorithms of the concrete examples in Subsections 5.2.1 to 5.2.4 map to the abstract model introduced above. The respective insights will be of particular importance in the context of the Lizard-construction introduced in Chapter 7.

5.2 Some Prominent Stream Cipher Examples

This section provides an overview of some prominent stream ciphers, with a focus on their respective state initialization algorithms and operation modes. It includes E_0 (used in Bluetooth) and A5/1 (used in GSM) as well as the eSTREAM hardware portfolio members Trivium and Grain v1. Please observe in the following that E_0 and $A5/1$

[2] Clearly, Trivium and Grain v1 could also be used in packet mode, but in contrast to, e.g., Lizard (see Chapter 8), their design is not specifically optimized for such scenarios.

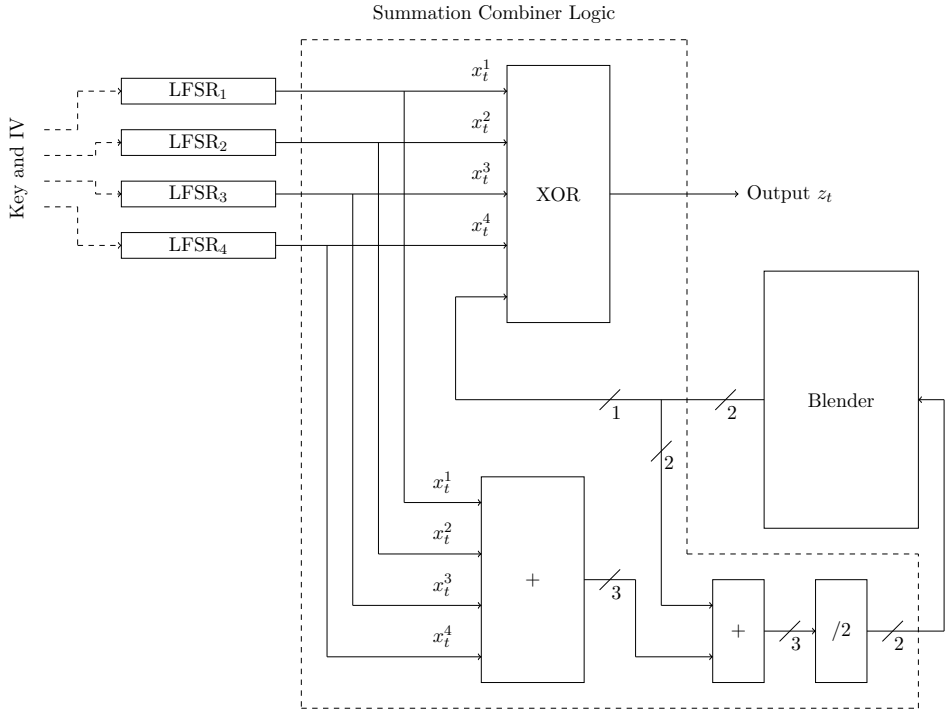

Figure 5.1: The E_0 encryption engine in step t. $x_t^1, \ldots, x_t^4$ denote the output bits of the four LFSRs (see Fig. 5.2 for tap positions), respectively. z_t denotes the bit generated by E_0's output function in this step. (cf. [Blu14], p. 1323)

are both operated in packet mode (cf. Section 5.1) in the sense that each packet in a session is encrypted under the same session key but using different, publicly known IVs. Furthermore, keep in mind that, as pointed out above, there are actually practical attacks on E_0 and $A5/1$ (see, e.g., [LMV05] and [BB06], respectively), whereas Trivium and Grain v1 can be considered still unbroken. There is, however, no connection between the fact that E_0 and $A5/1$ are operated in packet mode and the fact that they have been broken. From a security perspective, they simply belong to an older generation of stream ciphers as, for example, their feedback shift registers (FSRs) are all linear, in contrast to the nonlinear feedback shift registers (NFSRs) used in Trivium and Grain v1.

5.2.1 E_0 (used in Bluetooth)

The classical way of ensuring data confidentiality for Bluetooth connections between a master device A and a slave device B utilizes stream cipher-based encryption on a

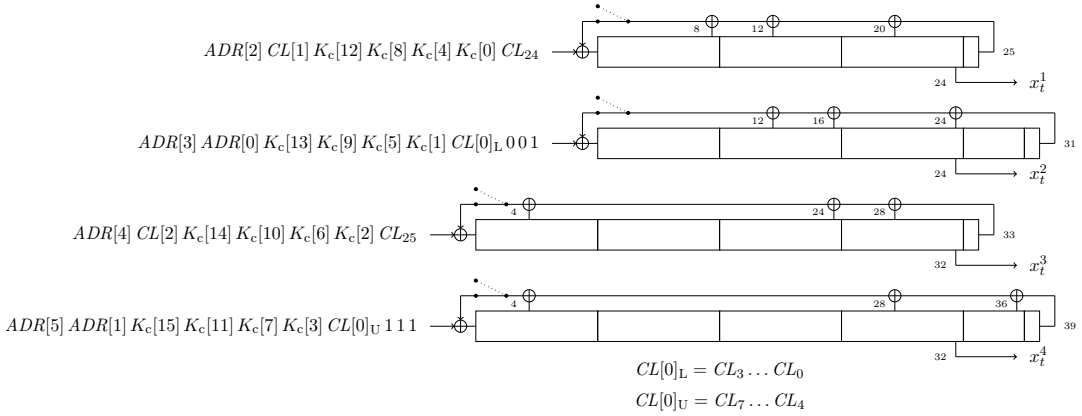

Figure 5.2: Initializing the LFSRs of E_0. $ADR[i]$, $K_c[i]$, and $CL[i]$ denote the bytes of ADR, K_c, and CL, respectively. (cf. [Blu14], p. 1327)

per-packet basis. More precisely, when encryption is activated, the payload of each packet (at most 2790 bits for the so-called *basic rate* [Blu14]) is encrypted under a separate payload key using the algorithm E_0, which produces a keystream of appropriate length that is XORed to the plaintext in a bitwise fashion. The term *payload key*, which is used in the original Bluetooth specification [Blu14], refers to the 132-bit inner state of the E_0-KSG right before the first keystream bit for payload encryption is produced (i.e., in terms of our abstract model, to the result q^i_{init} of the state initialization algorithm). It is derived on the basis of the current 128-bit encryption key K_c, the 48-bit Bluetooth address ADR of the master, and 26 bits $CL_0, \ldots, CL_{25}$ of the master's clock (to which both devices are synchronized) at the time of the first transmission slot of the packet.[3]

Before we continue with the respective details, please note that the encryption key K_c corresponds to a *session key*, i.e., K_c is constant for a potentially very large number of successive packets. Consequently, the difference between the payload keys q^i_{init} and q^j_{init}, $i \neq j$, of two different packets i and j during such a session solely arises from the difference between the corresponding, publicly known clock values at encryption time.

In the following, we will describe how the individual payload key q^i_{init} for a packet i is generated by E_0 based on the inputs K_c, ADR, and $CL_0, \ldots, CL_{25}$. The general structure of the encryption engine of E_0 is depicted in Fig. 5.1. The internal state of the four LFSRs has a total size of 128 bits and the respective feedback polynomials are all primitive (cf. [Blu14], p. 1322). Figure 5.2 shows the positions of the feedback taps. For further details like the exact definition of the *blender* finite-state machine (4 bits), we refer the reader to the official Bluetooth specification (cf. [Blu14], p. 1322–1324).

[3] Unlike the symmetric encryption key K_c, the values ADR and $CL_0, \ldots, CL_{25}$ are considered to be public. For more details on the generation of K_c, we refer the reader to the corresponding paragraph at the end of this subsection.

In our context (i.e., w.r.t. our abstract model and the conclusions we will draw from it), it is sufficient to understand that the inputs K_c, ADR, and $CL_0, \ldots, CL_{25}$ are split up and, together with the two 3-bit constants 111 and 001, arranged as depicted in Fig. 5.2 prior to payload key generation, which then works as follows (cf. [Blu14], p. 1325–1326):[4]

Step $t = 0$**:** Disable the feedback switches of the four LFSRs and set all registers to zero.

Steps $t = 1, \ldots, 39$**:** Start shifting K_c, ADR, $CL_0, \ldots, CL_{25}$, and the six bits 111 and 001 into the four LFSRs as depicted in Fig. 5.2 (e.g., at $t = 1$, CL_{24} is shifted into LFSR_1, 1 into LFSR_2, CL_{25} into LFSR_3, and 1 into LFSR_4). For each LFSR, close its feedback switch at the end of the step in which the first input bit has reached this LFSR's rightmost cell. When the last of the four switches (i.e., the switch of LFSR_4) is closed (i.e., at the end of step $t = 39$), reset the blender register cells (4 bits) to zero.[5]

Steps $t = 40, \ldots, 55$**:** Shift in the rest of the input bits. (Once an LFSR has no input bits left, it is henceforth operated in the conventional way, i.e., it is clocked with closed feedback switch and without any external interference.)

Steps $t = 56, \ldots, 111$**:** Clock the E_0-KSG. (Keep in mind that, in the steps $t = 0, \ldots, 111$, the result z_t of E_0's output function (see Fig. 5.1) is discarded completely.)

Steps $t = 112, \ldots, 239$**:** Clock the E_0-KSG and save the 128 bits $z_{112}, \ldots, z_{239}$ generated by E_0's output function during those steps to some temporary internal location. (Those bits are *not* used as actual keystream!)

Step $t = 240$**:** Copy the 128 bits $z_{112}, \ldots, z_{239}$, which were generated during the steps $t = 112, \ldots, 239$, from the aforementioned temporary location into the register cells of the four LFSRs (whose combined size is $25 + 31 + 33 + 39 = 128$ bits), overwriting their contents (for details, again, see [Blu14]). The contents of the blender register cells are not modified during this step.

At this point, the combined contents of the register cells of the four LFSRs (128 bits) and of the register cells of the blender component (4 bits) now represent the initial state q^i_{init} (132 bits) of the E_0-KSG (called payload key in the Bluetooth specification [Blu14]). Starting with this initial state q^i_{init}, the output bits $z_{240}, z_{241}, z_{242}, \ldots$, which serve as

[4] Keep in mind that, during the following steps $t = 0, \ldots, 239$, no keystream is produced. The first keystream bit for encryption will be z_{240}, which is generated by E_0's output function on the basis of the inner state of the E_0-KSG at the end of step $t = 240$ (i.e., on the basis of the *initial state* q^i_{init}).

[5] In each step t (except $t = 240$), the blender finite-state machine (FSM) is clocked as a part of the E_0 encryption engine (cf. Fig. 5.1). However, as the blender register cells are reset to zero at the end of $t = 39$ and as the output bits $z_0, \ldots, z_{111}$ are discarded by E_0, the summation combiner logic (cf. Fig. 5.1) and the blender FSM can be regarded as inactive during the steps $t = 0, \ldots, 39$. (see [Blu14], p. 1322–1324, for further details)

keystream bits for the encryption of the payload of packet i, are then generated by the E_0-KSG.[6] If another packet needs to be encrypted, the whole initialization process of the cipher is repeated with a new clock value as part of the IV for the state initialization algorithm of E_0, leading to a different payload key.

Bluetooth: Generating the Encryption Key K_c

When two Bluetooth devices connect for the first time (called *pairing*), an *initialization key* K_{init} is computed using the algorithm E_{22}, based on a common personal identification number (PIN), the Bluetooth address of one of the devices, and a 128-bit random number. Subsequently, K_{init} is used together with two random 128-bit values and the 48-bit Bluetooth addresses of both devices to create the the so-called *link key* (or *combination key*) K_{ab} using the algorithm E_{21}. This link key is sometimes also referred to as the *authentication key*, as it is used together with a (changing) 128-bit random number during the subsequent authentications of the respective devices using the algorithm E_1. As a result of such an authentication during connection establishment, a 32-bit authentication token *SRES* and a 96-bit value called *ACO* (Authenticated Ciphering Offset) are produced. If authentication has been successful (i.e., the authentication tokens match), a new *encryption key* K_{c} will be generated each time encryption is enabled by the communicating parties.[7] The generation of K_{c} is performed using the algorithm E_3, which is based on a hash function with the following inputs: the link key K_{ab}, a 128-bit random number *EN_RAND*, and the 96-bit value *ACO* generated during authentication. To account for legal export restrictions, the Bluetooth specification provides means of shortening the effective key length of K_{c} to less than 128 bits. The resulting keys are often referred to as K'_{c}. However, as the absolute key length will still be 128 bits, we omit this shortening step for the sake of clarity here and simply speak of the encryption key K_{c} in our description of E_0.

5.2.2 A5/1 (used in GSM)

Though considered outdated for security reasons (see, e.g., [BBK03] or the TMD-TO attack using the FPGA cluster COPACOBANA in [GKNP08]), the stream cipher A5/1 is still widely used to encrypt GSM communication. Upon connection establishment, the mobile device is authenticated and a 64-bit symmetric session key K_{c} is generated using

[6] To avoid any misconceptions, let us point out that the first keystream bit z_{240} is computed on the basis of the *new* LFSR contents in step $t = 240$, i.e., on the basis of (four of) the bits $z_{112}, \ldots, z_{239}$, which are copied into the register cells of the four LFSRs in this step. (The contents of the blender register cells, which are also involved in the computation of z_{240}, do not change in step $t = 240$.) The next keystream bit z_{241} is then obtained by clocking the E_0-KSG once and computing the output bit on the basis of the resulting register contents of the LFSRs and the blender component, and so on.

[7] According to the Bluetooth specification ([Blu14], p. 1308), when using E_0, "the encryption keys shall be refreshed by the Link Manager at least once every 2^{28} Bluetooth Clocks (about 23.3 hours)".

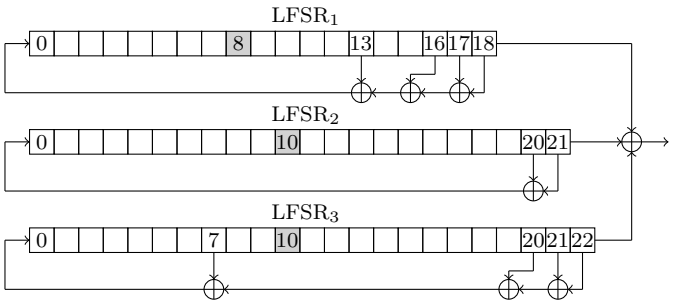

Figure 5.3: Structure of A5/1, which consists of three LFSRs of total state length 64 bits. The gray register cells denote the majority bits of the LFSRs.

one of the COMP128 [Bru04] algorithms, which take a publicly known 128-bit challenge (i.e., a random number produced by the authentication center of the network) and a common secret 128-bit key (stored on the SIM card and known to the authentication center) as inputs. For each data frame of 114 downlink and 114 uplink bits, 228 keystream bits are generated based on the 64-bit session key K_c and a (frame-individual) IV.

The LFSRs in A5/1 are either clocked all three in parallel or in so-called *majority mode*, where, at each clock cycle, the majority function over the three majority bits in Fig. 5.3 is computed and each LFSR is clocked if and only if its majority bit equals the output of the majority function. Please note that the feedback polynomials of the three LFSRs are all primitive (cf. [BD00]). The corresponding taps are depicted in Fig. 5.3.

Let $K_c = (K_{63}, \ldots, K_0)$ denote the secret session key and let $F = (F_{21}, \ldots, F_0)$ denote the 22-bit representation of the publicly known frame number. The 228 keystream bits which will be XORed to the corresponding 114 downlink bits and 114 uplink bits, respectively, are generated as follows (cf. [BD00]):

1. Set all register cells of the three LFSRs to 0.
2. For $i = 0$ to 63 do:
   - 2.1 $\text{LFSR}_1[0] := \text{LFSR}_1[0] \oplus K_i$
     $\text{LFSR}_2[0] := \text{LFSR}_2[0] \oplus K_i$
     $\text{LFSR}_3[0] := \text{LFSR}_3[0] \oplus K_i$
   - 2.2 Clock all three LFSRs (i.e., no majority clocking).
3. For $i = 0$ to 21 do:
   - 3.1 $\text{LFSR}_1[0] := \text{LFSR}_1[0] \oplus F_i$
     $\text{LFSR}_2[0] := \text{LFSR}_2[0] \oplus F_i$
     $\text{LFSR}_3[0] := \text{LFSR}_3[0] \oplus F_i$
   - 3.2 Clock all three LFSRs (i.e., no majority clocking).

4. Clock the KSG 100 times using majority clocking and discard the output bits, which are each computed as the XOR of the three bits $\text{LFSR}_1[18]$, $\text{LFSR}_2[21]$, and $\text{LFSR}_3[22]$ as depicted in Fig. 5.3.

5. Generate 114 keystream bits for downlink encryption clocking the KSG 114 times (using majority clocking).

6. Generate 114 keystream bits for uplink encryption clocking the KSG 114 times (using majority clocking).

For the enrcyption of the next frame, the above process is repeated with the same session key K_c but a different frame number F. Please note that the state of the LFSRs after step 4 (i.e., right before the first keystream bit is produced) corresponds to what was called *payload key* in the context of Bluetooth (cf. Subsection 5.2.1) or, in terms of our abstract model introduced in Section 5.1, to the *packet initial state* q^i_{init} of a packet i, where packets are called *frames* for A5/1 in GSM.

5.2.3 Trivium

Trivium [CP05] is one of the three members of the eSTREAM hardware portfolio. It is designed to produce a keystream of up to 2^{64} bits based on an 80-bit symmetric key and an 80-bit IV as inputs. Its inner structure consists of three shift registers of total length 288 bits, which are interwoven nonlinearly as depicted in Fig. 5.4. Each keystream bit is computed as the XOR of six bits from the current internal state, two from each shift register.

Before the first keystream bit is output, the following *state initialization algorithm* is performed based on the key $K = (K_1, \ldots, K_{80})$ and the initialization vector $IV = (IV_1, \ldots, IV_{80})$ (cf. [CP05]):

1. $(s_1, s_2, \ldots, s_{93}) := (K_1, \ldots, K_{80}, 0, \ldots, 0)$
   $(s_{94}, s_{95}, \ldots, s_{177}) := (IV_1, \ldots, IV_{80}, 0, \ldots, 0)$
   $(s_{178}, s_{179}, \ldots, s_{288}) := (0, \ldots, 0, 1, 1, 1)$

2. For $i = 1$ to $4 \cdot 288$ do:

   2.1 $t_1 := s_{66} \oplus s_{91}s_{92} \oplus s_{93} \oplus s_{171}$
   $t_2 := s_{162} \oplus s_{175}s_{176} \oplus s_{177} \oplus s_{264}$
   $t_3 := s_{243} \oplus s_{286}s_{287} \oplus s_{288} \oplus s_{69}$

   2.2 $(s_1, s_2, \ldots, s_{93}) := (t_3, s_1, \ldots, s_{92})$
   $(s_{94}, s_{95}, \ldots, s_{177}) := (t_1, s_{94}, \ldots, s_{176})$
   $(s_{178}, s_{179}, \ldots, s_{288}) := (t_2, s_{178}, \ldots, s_{287})$

In terms of our model, the 288-bit state $s_1, \ldots, s_{288}$ of the three shift registers right after stage 2 of the above initialization procedure has been completed corresponds to the

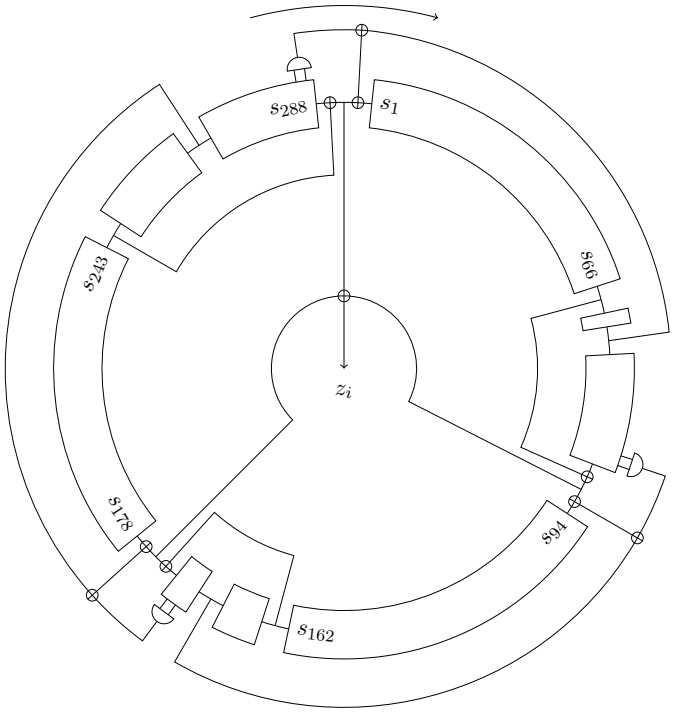

Figure 5.4: Structure of Trivium (total state length 288 bits). z_i denotes the output bit in step i. (cf. [CP05])

initial state q_{init}. Based on this state, the KSG is then further clocked using the same operations with the only difference that, from now on, in each step i the keystream bit $z_i = s_{66} \oplus s_{93} \oplus s_{162} \oplus s_{177} \oplus s_{243} \oplus s_{288}$ is output in order to encrypt one bit of plaintext.

Please observe that the state update of Trivium is reversible (see also [CP05]) during state initialization and keystream generation. In particular, the knowledge of an arbitrary inner state during one of these phases will allow an attacker to efficiently compute the underlying secret key by clocking the cipher backwards.

5.2.4 Grain v1

Grain v1 [HJM06], like Trivium, is a member of the eSTREAM hardware portfolio. It takes an 80-bit key and a 64-bit IV as inputs and is composed of two shift registers, one an 80-bit NFSR and one an 80-bit LFSR, which are connected as depicted in Fig. 5.5.

Let us denote the 80-bit state of the NFSR by $(b_0, \ldots, b_{79})$ and the 80-bit state of the LFSR by $(s_0, \ldots, s_{79})$. Then the state update of the NFSR during keystream generation

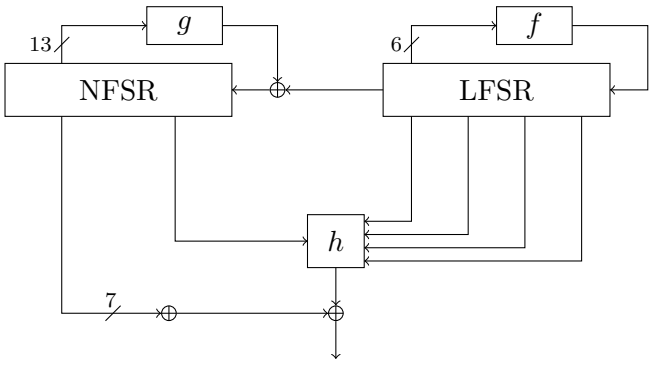

Figure 5.5: Structure of Grain v1 (keystream generation phase). f denotes the linear feedback function of the LFSR, g denotes the nonlinear feedback function of the NFSR. (cf. [HJM06])

is defined by $b'_i := b_{i+1}$, $i = \{0, \ldots, 78\}$, and

$$\begin{aligned} b'_{79} := {} & s_0 \oplus b_{62} \oplus b_{60} \oplus b_{52} \oplus b_{45} \oplus b_{37} \oplus b_{33} \oplus b_{28} \oplus b_{21} \oplus b_{14} \oplus b_9 \oplus b_0 \\ & \oplus b_{63}b_{60} \oplus b_{37}b_{33} \oplus b_{15}b_9 \\ & \oplus b_{60}b_{52}b_{45} \oplus b_{33}b_{28}b_{21} \\ & \oplus b_{63}b_{45}b_{28}b_9 \oplus b_{60}b_{52}b_{37}b_{33} \oplus b_{63}b_{60}b_{21}b_{15} \\ & \oplus b_{63}b_{60}b_{52}b_{45}b_{37} \oplus b_{33}b_{28}b_{21}b_{15}b_9 \\ & \oplus b_{52}b_{45}b_{37}b_{33}b_{28}b_{21}, \end{aligned}$$

where $(b'_0, \ldots, b'_{79})$ denotes the new state of the NFSR.

Analogously, the state update of the LFSR, whose feedback polynomial is primitive (cf. [HJM06]), can be defined by $s'_i := s_{i+1}$, $i = \{0, \ldots, 78\}$, and

$$s'_{79} := s_{62} \oplus s_{51} \oplus s_{38} \oplus s_{23} \oplus s_{13} \oplus s_0,$$

where $(s'_0, \ldots, s'_{79})$ denotes the new state of the LFSR.

The result $h(s_3, s_{25}, s_{46}, s_{64}, b_{63})$ for the nonlinear function $h : \{0,1\}^5 \longrightarrow \{0,1\}$,

$$\begin{aligned} h(x_0, x_1, x_2, x_3, x_4) = {} & x_1 \oplus x_4 \oplus x_0x_3 \oplus x_2x_3 \oplus x_3x_4 \\ & \oplus x_0x_1x_2 \oplus x_0x_2x_3 \oplus x_0x_2x_4 \oplus x_1x_2x_4 \oplus x_2x_3x_4, \end{aligned}$$

is XORed with the seven NFSR bits $b_1, b_2, b_4, b_{10}, b_{31}, b_{43}, b_{56}$ to produce the output of the cipher. Observe that neither s_0 nor b_0 are among the inputs of h, as this implies that the state update is not only reversible during the keystream generation phase[8] depicted in Fig. 5.5, but also during the initialization phase, which is depicted in Fig. 5.6 and will now be described in further detail.

[8]The LFSR state update is trivially reversible and the NFSR update function contains b_0 only as a linear term.

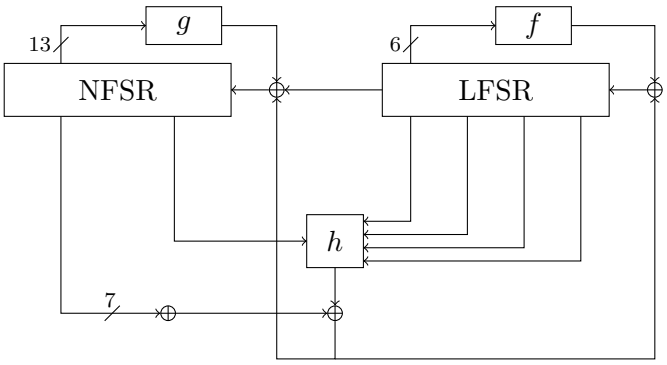

Figure 5.6: Structure of Grain v1 (initialization phase). (cf. [HJM06])

At the beginning of the initialization phase, the 80-bit secret key $k = (k_0, \ldots, k_{79})$ is parallel loaded (i.e., copied) to the 80-bit NFSR and the 64-bit public initialization vector $IV = (IV_0, \ldots, IV_{63})$ is parallel loaded to the first 64 register cells of the LFSR. The 16 remaining LFSR register cells are filled with ones (thus, avoiding initialization with the all-zero state). More precisely,

$$(b_0, \ldots, b_{79}) := (k_0, \ldots, k_{79}) \,,$$
$$(s_0, \ldots, s_{79}) := (IV_0, \ldots, IV_{63}, 1, \ldots, 1) \,.$$

After this, the cipher is clocked 160 times but instead of producing keystream, the output is XORed to the previously described state update for the register cells b_{79} and s_{79} as depicted in Fig. 5.6.

The resulting content of the two feedback shift registers (160 bits in total) corresponds to the initial state q_{init}. Henceforth, the KSG is operated in keystream generation mode (cf. Fig. 5.5), producing one keystream bit per clock cycle.

5.2.5 Excursus: Block Cipher-based Constructions

Before we move on to Section 5.3, where we will analyze in detail how the state initialization algorithms of the stream ciphers in Subsections 5.2.1 to 5.2.4 can be described in terms of our abstract model, we now briefly discuss another way for realizing stream ciphers, which will play a role in Chapter 6 in the context of distinguishing attacks.

Instead of designing a KSG from scratch, stream ciphers can also be built on the basis of block ciphers, e.g., by using them in *counter mode (CTR mode)* as depicted in Fig. 5.7.[9] Notably, "one of the requirements imposed on all eSTREAM stream cipher

[9] A prominent example for this approach is A5/3 [3GP03], the 'stronger' alternative to A5/1 regarding GSM encryption. A5/3 uses the block cipher KASUMI [3GP17] in a mixture between CTR mode and output feedback mode (OFB mode). However, the effective key length in A5/3 is still only 64 bits and there are already attacks with complexity below exhaustive key search (see, e.g., [BDK05] and [DKS10]).

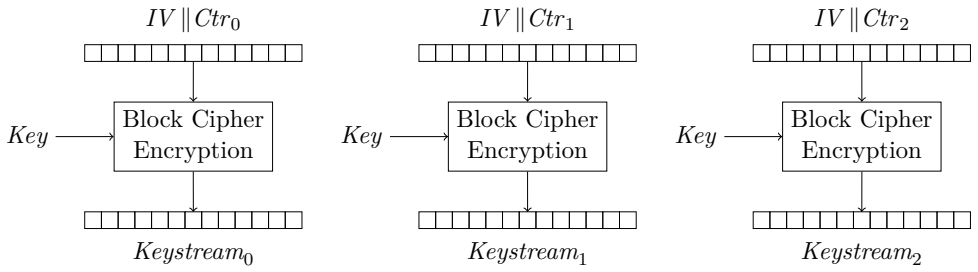

Figure 5.7: Example of block cipher-based keystream generation using counter mode. $IV||Ctr_i$ denotes the concatenation of the initialization vector IV and the counter value Ctr_i in step i. $Keystream_i$ denotes the keystream fragment produced in step i.

submissions was that they should demonstrate the potential to be superior to the AES in at least one significant aspect" [ECR05] and the respective testing framework included AES in CTR mode.

Please observe that using a block cipher in CTR mode to generate a keystream as shown in Fig. 5.7 can also be interpreted as a kind of *packet mode* (cf. Section 5.1), similar to what we described for E_0 and A5/1. In this case, however, the packet size is not determined by the underlying technology (i.e., the maximum payload size for Bluetooth packets or the fixed frame size in the case of GSM), but by the block size of the employed block cipher instead. Concretely, the concatenation $IV||Ctr_i$ used as the input for the block cipher in Fig. 5.7 can be interpreted as a publicly known initialization vector IV^i on the basis of which the keystream $Keystream_i$ (i.e., S^i in terms of our model in Section 5.1) for the packet i, whose size equals the block length of the underlying cipher, is generated.

A related approach in order to realize a stream cipher on the basis of a block cipher was taken for LEX [Bir05], a phase 3 candidate in the eSTREAM software profile. The way AES (with key and block length 128 bits) is used in LEX (see Fig. 5.8) strongly resembles OFB mode, with the difference that instead of using the AES ciphertext blocks as keystream blocks, in each AES round, four bytes are extracted from the intermediary state and taken as keystream bytes (i.e., 320 bits of keystream during one invocation of AES with a 128-bit key). By generalizing a distinguishing attack of Englund, Hell and Johansson against LEX [EHJ07], in Chapter 6, we will be able to present a generic distinguisher for the new class of so-called *small-state stream ciphers* as well.

5.3 Modeling the State Initialization of the Examples

In this section, we are now going to explore how the state initialization algorithms of the 'classical' stream ciphers in Subsections 5.2.1 to 5.2.4 can be described in terms of our abstract model introduced in Section 5.1. The resulting insights will then help us in

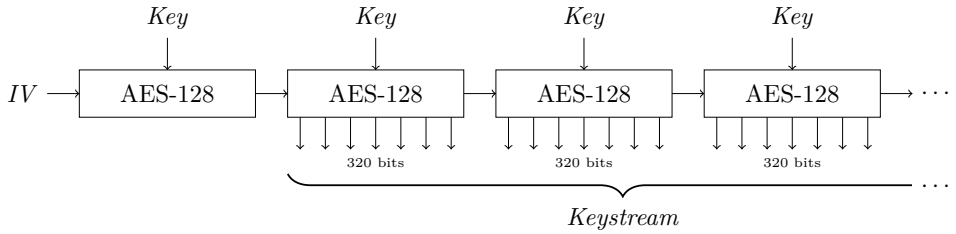

Figure 5.8: Initialization and keystream generation of LEX. (cf. [Bir05])

Chapter 7 to suggest and analyze a new type of state initialization in the form of the LIZARD-construction.

Let us recall that we suppose the key and IV setup phase (A) of a KSG-based stream cipher to contain the following two subphases:

(A.1) The *loading phase* defines how the session key k and the initialization vector IV are loaded into the inner state registers and results in a *loading state* $q_{\text{load}} = q_{\text{load}}(k, IV)$.

(A.2) The *mixing phase* runs an appropriate KSG-based *mixing algorithm*

$$MIX : \{0,1\}^n \longrightarrow \{0,1\}^n$$

on q_{load} and results in a state $q_{\text{mixed}} = MIX(q_{\text{load}})$.

In the following, we will see that for all of the examples in Subsections 5.2.1 to 5.2.4, it holds that $q_{\text{init}} = q_{\text{mixed}}$, where q_{init} denotes the *initial state* on the basis of which the keystream is eventually computed by the KSG as described in Section 5.1. Moreover, we will show that for A5/1, Trivium, and Grain v1, the mixing algorithm MIX can be represented by an efficiently invertible, public function $P : \{0,1\}^n \longrightarrow \{0,1\}^n$, where, in the case of Trivium and Grain v1, P is a permutation. For E_0, a large part of the state initialization algorithm can be described by an efficiently invertible, 'nearly bijective' function P' as well (see below).

5.3.1 E_0 (used in Bluetooth)

For the sake of simplicity, in this subsection, we denote the combination of publicly known inputs (including constants) for E_0's state initialization (cf. Subsection 5.2.1) by a single initialization vector IV. The way IV (80 bits) and the secret key k (128 bits) are loaded into the four LFSRs at the beginning of the initialization procedure is completely GF(2)-linear. Hence, the combined state of the four LFSRs right after IV and k have been shifted in completely (i.e., at $t = 55$) can be described as $f(IV) \oplus g(k)$, where $f : \{0,1\}^{80} \longrightarrow \{0,1\}^{128}$ and $g : \{0,1\}^{128} \longrightarrow \{0,1\}^{128}$ are GF(2)-linear functions. The 4-bit state $h(IV, k)$ of the blender FSM at $t = 55$ is determined by a nonlinear function

$h : \{0,1\}^{80} \times \{0,1\}^{128} \longrightarrow \{0,1\}^4$ depending on the 80-bit initialization vector IV and the 128-bit key k. Based on $f(IV)$, $g(k)$, and $h(IV,k)$, we can now denote the full inner state of E_0 at $t = 55$ as

$$q^i_{\text{load}} = (f(IV) \oplus g(k)) \,||\, h(IV,k).$$

Note, however, that, at this point in time, 32 bits of the secret session key k have been shifted into LFSR_1 (and nowhere else), which is only 25 bits wide (see Fig. 5.2 in Subsection 5.2.1). Consequently, any attack recovering q^i_{load} will 'only' reveal a linear combination with reduced information about k, which may induce further effort, e.g., by requiring to search for the right 4-bit state of the blender engine, when trying to decrypt other packets encrypted under the same session key.

In E_0, the state update of the four LFSRs (128 bits) is fully bijective as they are completely independent of each other and the blender engine (cf. Fig. 5.1). Only the state update of the blender FSM (4 bits) possibly isn't. Hence, we will describe the phase between $t = 56$ and $t = 111$, during which, as part of the state initialization, the cipher is clocked and its output is discarded, using an efficiently invertible and 'nearly bijective' function $P' : \{0,1\}^{132} \longrightarrow \{0,1\}^{132}$.

A distinctive property of E_0 is that, as the final step of the state initialization algorithm, 128 output bits are produced (between $t = 112$ and $t = 239$), which are not used as keystream (i.e., they are kept internal), but instead (at $t = 240$) are parallel loaded to the register cells of the four LFSRs. Let $\tilde{F} : \{0,1\}^{132} \longrightarrow \{0,1\}^{128}$ denote the one-way function which, given an inner state $y \in \{0,1\}^{132}$, computes the first 128 output bits of the E_0-KSG based on y. Moreover, let $\tilde{h} : \{0,1\}^{132} \longrightarrow \{0,1\}^4$ denote the one-way function which, given an inner state $y \in \{0,1\}^{132}$, computes the new 4-bit state of E_0's blender FSM after 128 clock cycles. We can now describe the step between $t = 112$ and $t = 240$ using the function $F' : \{0,1\}^{132} \longrightarrow \{0,1\}^{132}$ with $F'(y) = \tilde{F}(y) || \tilde{h}(y)$. Subsequently, after $t = 240$, the keystream for the corresponding Bluetooth packet i (of length at most 2790 bits) is produced based on the initial state

$$q^i_{\text{init}} = q^i_{\text{mixed}} = F'\Big(P'\Big(q^i_{\text{load}}\Big)\Big).$$

In terms of our abstract model, this means that for E_0, the mixing algorithm $MIX : \{0,1\}^{132} \longrightarrow \{0,1\}^{132}$ has the form $MIX = F' \circ P'$, where P' is an efficiently invertible, 'nearly bijective' function and F' is a one-way function. As a consequence, in contrast to A5/1, Trivium, and Grain v1 (see below), MIX is *not* efficiently invertible in the case of E_0. Nevertheless, we will show in Theorem 7.2 of Chapter 7 that this does not thwart birthday-bound time-memory-data tradeoff attacks.

5.3.2 A5/1 (used in GSM)

The way in which the secret session key k (64 bits) and the initialization vector IV (22 bits) are loaded to the LFSRs of the A5/1 engine is rather similar to the Bluetooth

cipher E_0. A major difference, however, is that during the first 64 clock cycles of the state initialization algorithm, each of the 64 key bits is introduced to *each* of the three LFSRs of total size 64 bits.

The way we model A5/1 makes use of following two "key ideas" from [BSW01] by Biryukov, Shamir, and Wagner:

- "Key idea 3: A5/1 can be efficiently inverted",
- "Key idea 4: The key can be extracted from the initial state of any frame"[10].

As in the case of E_0, the introduction of IV and k to three LFSRs of A5/1 at the beginning of the initialization procedure is completely GF(2)-linear. Hence, the combined state of the LFSRs right after IV and k have been shifted in completely (i.e., at the end of step 3 of the algorithm in Subsection 5.2.2) can be described as

$$q^i_{\text{load}} = f(IV) \oplus g(k),$$

where $f : \{0,1\}^{22} \longrightarrow \{0,1\}^{64}$ and $g : \{0,1\}^{64} \longrightarrow \{0,1\}^{64}$ are GF(2)-linear functions. Due to "key idea 3" from [BSW01], the subsequent stepping of the KSG for 100 times using majority clocking (step 4 of the algorithm in Subsection 5.2.2) can be described by an efficiently invertible function $P : \{0,1\}^{64} \longrightarrow \{0,1\}^{64}$. Based on the initial state

$$q^i_{\text{init}} = q^i_{\text{mixed}} = P\Big(q^i_{\text{load}}\Big)$$

after step 4, the 228 keystream bits for the corresponding frame are then produced. Hence, in terms of our abstract model, the mixing algorithm $MIX : \{0,1\}^{64} \longrightarrow \{0,1\}^{64}$ of A5/1 has the form $MIX = P$ and MIX is efficiently invertible.

5.3.3 Trivium

For Trivium, the way the the secret key k (80 bits) and the initialization vector IV (80 bits) are introduced to the state registers (of total length 288 bits) is even simpler than in the case of E_0 or A5/1, as the respective values are simply parallel loaded (see step 1 of the state initialization algorithm in Subsection 5.2.3), resulting in the loading state

$$q_{\text{load}} = \sigma(k||IV||CONST),$$

where $CONST$ denotes the initialization constants (208 bits) and $\sigma : \{0,1\}^{288} \longrightarrow \{0,1\}^{288}$ denotes a fixed, publicly known, *bitwise* permutation, which determines the exact placement of the key bits, IV bits, and constant bits.

[10]Note that Biryukov, Shamir, and Wagner refer to the state of the LFSRs right after the frame counter has been introduced (i.e., after step 3 of the algorithm in Subsection 5.2.2) as the "initial state", whereas we denote by *initial state* the output of the state initialization procedure (i.e., the state at the end of step 4 of the algorithm in Subsection 5.2.2). This, however, does not harm our argumentation as "key idea 4" can be combined with "key idea 3".

As pointed out previously, the subsequent clocking of the Trivium engine without producing output (step 2 of the state initialization algorithm) is easily reversible. Thus, we can describe this step (comprising $4 \cdot 288$ clock cycles) by an efficiently invertible permutation $P : \{0,1\}^{288} \longrightarrow \{0,1\}^{288}$. Based on the resulting initial state

$$q_{\text{init}} = q_{\text{mixed}} = P(q_{\text{load}}),$$

a keystream of up to 2^{64} bits is then produced by the KSG. Hence, in terms of our abstract model, the mixing algorithm $MIX : \{0,1\}^{288} \longrightarrow \{0,1\}^{288}$ of Trivium has the form $MIX = P$ and MIX is an efficiently invertible permutation.

5.3.4 Grain v1

The state initialization of Grain v1 works similar to the one of Trivium. That is, the secret key k (80 bits) and the initialization vector IV (64 bits) are parallel loaded to the register cells of the NFSR and the LFSR, respectively, resulting in the loading state

$$q_{\text{load}} = \sigma(k||IV||CONST),$$

where $CONST$ denotes the initialization constants (16 bits) and $\sigma : \{0,1\}^{160} \longrightarrow \{0,1\}^{160}$ denotes a fixed, publicly known, *bitwise* permutation, which determines the exact placement of the key bits, IV bits, and constant bits.

Subsequently, the cipher is clocked 160 times without producing keystream. Even though the output of the nonlinear function $h : \{0,1\}^5 \longrightarrow \{0,1\}$, XORed with further seven bits from the NFSR, is fed back to both FSRs during this phase, we have seen in Subsection 5.2.4 that the state update of Grain v1 is still straightforwardly reversible. We can thus describe this stage of the state initialization algorithm using an efficiently invertible permutation $P : \{0,1\}^{160} \longrightarrow \{0,1\}^{160}$. Just like in the case of Trivium, the resulting initial state can now be written as

$$q_{\text{init}} = q_{\text{mixed}} = P(q_{\text{load}}).$$

Hence, in terms of our abstract model, the mixing algorithm $MIX : \{0,1\}^{160} \longrightarrow \{0,1\}^{160}$ of Grain 1 has the form $MIX = P$ and MIX is an efficiently invertible permutation.

5.4 Conclusion and Outlook

In this chapter, we revisited some prominent examples of classical stream ciphers and, in particular, analyzed their respective state initialization algorithms. Moreover, we explained that in practical applications, stream ciphers are usually used in packet mode. In Chapter 7, the corresponding insights will serve as a basis for our new stream cipher design principle named Lizard-construction. But prior to this, as a counterpart to

'classical' stream ciphers, we will introduce the new group of so-called *small-state stream ciphers* in the following Chapter 6. In a nutshell, for an inner state of size n bits, these ciphers try to reach a security level larger than $n/2$, which is the limit imposed on classical stream ciphers by the birthday bound. Note that Grain v1, which has a 160-bit inner state and aims for 80-bit security, exactly hits this classical 'frontier'. In particular, regarding the common tradeoff between inner state size and complexity of the involved operations, we have seen in Subsections 5.2.3 and 5.2.4, respectively, that while Trivium uses a comparatively large state but extremely simple functions, Grain v1 actually works the other way around. It is henceforth not surprising that all of the small-state stream ciphers described in the following Chapter 6 are based on the general structure of the Grain family [HJMM08].

Small is beautiful!

Fritz Schumacher (economist)

CHAPTER 6

Small-State Stream Ciphers

ABSTRACT

Time-memory-data tradeoff (TMD-TO) attacks limit the security level of many classical stream ciphers to the birthday bound. Very recently, a new field of research has emerged, which searches for so-called *small-state stream ciphers* that try to overcome this limitation. In this chapter, existing designs and known analysis of small-state stream ciphers are revisited and new insights on distinguishers and key recovery are derived based on TMD-TO attacks.

A particular result is the transfer of a generic distinguishing attack suggested in 2007 by Englund et al. to this new class of lightweight ciphers. Our analysis shows that the initial hope of achieving full security against TMD-TO attacks by continuously using the secret key has failed. In particular, we provide generic distinguishers for Plantlet and Fruit with complexity significantly smaller than that of exhaustive key search. However, by studying the assumptions underlying the applicability of these attacks, we are able to come up with a new design idea for small-state stream ciphers, which might allow to finally achieve full security against TMD-TO attacks by continuously using not only the key but also the IV during keystream generation.

Another contribution of this chapter is the first key recovery attack against Fruit v1. We show that there are at least 2^{64} weak keys, each of which does not provide 80-bit security as promised by designers. As a consequence of our attack, the designers of Fruit have updated their scheme to the new version Fruit v2.

Declaration of Origin: This chapter is based on the paper *Design and Analysis of Small-State Grain-like Stream Ciphers* [HKMZ18], written together with Matthias Krause, Willi Meier, and Bin Zhang and published in *Cryptography and Communications* (Springer US journal).

6.1 Introduction

As explained in the previous Chapter 5, for more than three decades now, stream ciphers have been an essential component of many digital communication solutions. In 2004, after the discovery of weaknesses in several practically used ciphers such as E_0 (cf. Subsection 5.2.1) and A5/1 (cf. Subsection 5.2.2), the eSTREAM project [ECR08] was started in order to identify new stream ciphers for different application profiles. In the hardware category, aiming at devices with restricted resources, three ciphers are still part of the eSTREAM portfolio after the latest revision in 2012: Trivium [CP05] (cf. Subsection 5.2.3), Grain v1 [HJM06] (cf. Subsection 5.2.4), and MICKEY 2.0 [BD06].

Common to these three ciphers is that they have an inner state length of at least twice the size of the targeted security level against key recovery attacks. This is due to the inherent vulnerability of classical stream ciphers against TMD-TO attacks like those of Babbage [Bab95] and Biryukov and Shamir [BS00], which allow to recover some inner state during keystream generation (and, usually, also the corresponding initial state by clocking the cipher backwards) with an overall attack complexity of $2^{n/2}$, where n denotes the inner state length of the underlying KSG. If the state initialization algorithm, which computes the initial state from a given key/IV pair, is efficiently invertible (as it is, e.g., for Trivium and Grain; see Section 5.3), knowing the initial state immediately reveals the secret key. And even if the state initialization algorithm is not efficiently invertible, variants of such TMD-TO attacks can allow for key recovery, e.g., by targeting the inner state at $t = 0$, which often contains the secret key (see Section 5.2). A generic view on these attacks is provided in Chapter 7 along with a corresponding complexity analysis. Furthermore, building on the aforementioned classical TMD-TO attacks, a large body of work (see, e.g., [HS05] and [Bjø08]) about improved techniques/applications and resulting design implications for stream ciphers has been published since.

In 2015, a new line of research emerged with the publication of Sprout [AM15] by Armknecht and Mikhalev, which pursues the goal of reducing the size of the volatile inner state of lightweight stream ciphers below this 'magic' boundary formerly induced by TMD-TO attacks. We will refer to such ciphers, whose volatile inner state size is less than twice the targeted security level against key recovery, by the term *small-state stream ciphers*. Sprout has a Grain-like structure and uses two 40-bit feedback shift registers. In comparison to conventional stream ciphers like Grain v1, the characteristic difference of Sprout is that the 80-bit key is not only accessed during the state initialization but also continuously used as part of the state update during the subsequent keystream generation phase. Even though Sprout was broken shortly after publication (see, e.g., [LNP15], [ZG15], [Ban15], [EK16]), it has sparked interest in the underlying design principle and related ciphers like Fruit [GHX16] have been suggested since. It is observed that the designers of Fruit have changed the specification of their cipher several times in the past (e.g., ePrint versions 20160521:111224, 20161124:115414, 20170304:073404 of [GHX16] all contain different algorithms). We will refer to the most recent version of Fruit at the

time of our attacks (i.e., ePrint version 20170304:073404) as Fruit v1 in the rest of this chapter. Note that, in reaction to our results, the designers of Fruit have updated their scheme to the new version Fruit v2 (ePrint version 20170724:053140).

Although continuously accessing the secret key is elegant in theory, it often comes at a heavy price in practice. For example, if the key is stored in an EEPROM (see Subsection 2.3.9), the corresponding access times may significantly slow down the operation speed of the KSG. This is particularly true if the key bits are not accessed sequentially (as in the case of Sprout and its successor Plantlet [MAM17]) but at random positions (as in the case of Fruit v1). In fact, Fruit v1 needs to access six different, non-sequential key bits per clock cycle. Another drawback to continuously reading key bits from an EEPROM is the associated increase in power consumption, which is especially problematic for low-cost RFID tags, where the power budget is often as low as 10 μW (cf. Subsection 2.3.4). For comparison, Ranasinghe and Cole state that "[a] tag performing a read operation will require about 5 μW – 10 μW, while a tag attempting to perform a write operation to its E^{2}PROM will require about 50 μW or more." [CR08] A way to circumvent the disadvantages of continuously accessing the key in the EEPROM might be to buffer it in volatile memory. This, however, would significantly increase the hardware costs of the corresponding implementation due the additionally required flip-flops (cf. Subsection 2.3.3).

On the other hand, in an application where the key is fixed (e.g., by using key-dependent masks or via 'burning' fuses/antifuses; cf. Subsection 2.3.10), continuously accessing key bits (in potentially random order) is clearly feasible. Hence the question whether a more complicated[1] key schedule as in the case of Fruit v1 does provide additional security when compared to, e.g., Plantlet, is not only interesting from a theoretical point of view but of actual practical relevance.

As a consequence of this discussion, we see a number of issues that motivate small-state stream ciphers and their further study:

- Resource limitations defined by the notions "ultra-constrained RFIDs" [AHM14] (cf. Chapter 2) and "ultra-lightweight cryptography" [BP17].
- Optimizing the design (w.r.t. hardware efficiency) for certain application scenarios. See, for example, our new stream cipher Lizard in Chapter 8, which works in *packet mode* (cf. Section 5.1) and allows for a more power-efficient implementation than, e.g., Grain v1; or Sprout-like ciphers, which target scenarios where the secret key is permanently available to the encryption engine.
- The theoretical question, how the 'established' $n/2$ security bound implied by classical TMD-TO attacks like those of Babbage [Bab95] or Biryukov and Shamir

[1] Please note that when speaking of a *more complicated key schedule*, we are referring to the way in which the key bits are used in order to compute the round key bit (e.g., using more than on key bit as input to an elaborate round key function with potentially high algebraic degree).

[BS00] can be improved w.r.t. key recovery and/or distinguishing.

- Which of these 'alleged improvements' do actually provide *provable security*? (we refer to our security proof for Lizard in Chapter 7 and to the contents of Section 6.3)

In direction of cryptanalysis, while Plantlet has not been broken in the meantime, this chapter presents a key recovery attack against Fruit v1 that makes use of the cipher's insecure key schedule. In fact, our results seem to raise the question whether there is actually a need for (respectively a benefit of) a more complicated key schedule than the basic one used in Plantlet. Because not only Fruit v1 has now been broken due to its nonlinear key schedule, but also Sprout initially suffered from this fate (see Subsection 6.2.1). It seems that simplicity may actually be preferable here. After all, the original idea underlying the use of key bits for the state update during keystream generation was simply to protect against TMD-TO attacks aiming at inner state recovery. Sequentially using one key bit per clock cycle as done by Plantlet already seems to do this job.

Another contribution of this chapter is the discussion of TMD-TO distinguishing attacks against ciphers like Sprout, Fruit v1, and Plantlet, which continuously use the secret key as part of the state update. The existence of such attacks with a complexity below that of exhaustive key search was considered an exclusion criterion in the eSTREAM contest. Making use of a result by Englund, Hell and Johansson [EHJ07], we give a generic keystream-based TMD-TO distinguisher with attack complexity way below the complexity of exhaustive key search for Plantlet and Fruit v1. In fact, Banik already showed in 2015 [Ban15] that there is a similar distinguisher for Sprout. His attack can also be seen as a specific variant of the generic distinguisher by Englund, Hell and Johansson from 2007 (i.e., long before Sprout, Fruit, and Plantlet were introduced).

We would like to point out that, in our opinion, the existence of distinguishing attacks with a complexity below that of exhaustive key search should not be a knock-out criterion for stream ciphers targeting ultra-lightweight applications, because such attacks might actually be tolerable depending on the application scenario. Nonetheless, the user needs to know about the corresponding complexities in order to be able to decide on this question. In [GHX16] (version 20170304:073404), however, the designers of Fruit v1 do not discuss the topic of distinguishing attacks at all, which might give potential users the impression that Fruit v1 is considered 80-bit secure (the general security level claimed in the abstract of the paper) against this type of attack. The designers of Plantlet, on the other hand, recognize the possibility of distinguishing attacks and discuss them as part of their cryptanalysis in [MAM17]. In particular, they point out the aforementioned distinguishing attack of Banik against Sprout and acknowledge that it would also be applicable to Plantlet ("the memory complexity of this distinguishing attack against Plantlet is at least 2^{58} which is about 32,768 terabytes" [MAM17]). However, they also state: "This attack is possible due to the simplicity of the round key function. It would be possible to make the design resistant against this attack by either choosing a more

complicated key-selection function or by further increasing internal state size." [MAM17] While increasing the size of the inner state would in fact be a valid countermeasure (though it would somehow counteract the initial idea of having a smaller state), employing a more complicated key schedule as suggested by the Plantlet designers would not help.

The fact that the designers of Fruit v1 do not discuss the topic of distinguishing attacks at all together with the above misconception of the Plantlet authors w.r.t. the effects of a more complicated key schedule seem to make it beneficial for future research in the direction of small-state stream ciphers to revisit the distinguishing attack by Englund, Hell, and Johansson and transfer it, along with a discussion of the now necessary assumptions, to this new application context. In particular, by analyzing those assumptions, we will then be able to come up with a new design idea for small-state stream ciphers, which finally delivers what continuously using the secret key alone cannot: security against TMD-TO-based key recovery *and* distinguishing attacks.

Before we sketch the structure of this chapter, let us mention that, besides our results about Sprout, Fruit and Plantlet, we will also provide a brief description of our new stream cipher Lizard here, despite the fact that it will be treated in extensive detail in Chapter 8. Our reason for doing so is that in Section 6.2, we want to draw the full picture regarding the current state of the art in small-state stream cipher development. This holds in particular as Lizard is currently the only design which strives for beyond-the-birthday-bound security w.r.t. key recovery based on a different approach than Sprout-like ciphers.

Structure of this chapter: In Section 6.2, the existing small-state stream ciphers and their known analysis are revisited. In Subsection 6.3.1, we present a generic distinguishing attack against ciphers which continuously use the secret key, like Sprout, Fruit and Plantlet. It is based on a result by Englund, Hell and Johansson [EHJ07] from 2007 and also related to the distinguishing attack of Banik [Ban15] against Sprout from 2015. The generic nature of the attack will clarify that the security against distinguishing of ciphers which continuously use the secret key cannot be increased by choosing a more complicated key schedule. In Subsection 6.3.2, we demonstrate that Fruit v1 has at least 2^{64} weak keys, each of which does not provide the 80-bit security promised by the designers. The corresponding attack exploits the cipher's vulnerable key schedule and uses a variant of Babbage's TMD-TO attack [Bab95] to recover a subset of the unknown key bits together with a certain state of the feedback shift registers (FSRs) during state initialization. Based on this information, the cipher is then clocked back and the rest of the key bits is read from the FSRs at $t = 0$. In Section 6.4, we indicate recent advances and potential directions for further developments of small-state stream ciphers. In Section 6.5, a new design idea on stream ciphers which continuously use the initialization vector is presented, which not only allows to achieve security against TMD-TO-based key recovery but also against distinguishing. Section 6.6 concludes the chapter and provides an outlook on potential future work as well as on the subsequent contents of this thesis.

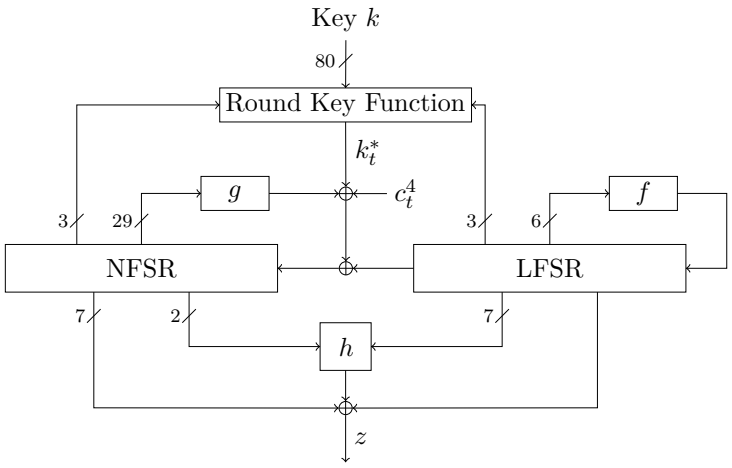

Figure 6.1: Keystream generation of Sprout. (cf. [AM15] and [ZG15])

6.2 Small-State Stream Ciphers

In the following, we will provide an overview over all current small-state stream ciphers, along with their respective specification and a summary of cryptanalytic results.

6.2.1 Sprout

Sprout [AM15] utilizes an 80-bit secret key $k = (k_0, \ldots, k_{79})$ and a 70-bit public initialization vector $IV = (IV_0, \ldots, IV_{69})$. Its structure, depicted in Fig. 6.1, is based on that of the Grain family of stream ciphers [HJMM08] (cf. Subsection 5.2.4). The major innovation of Sprout is that, through a round key function, the secret key is continuously (i.e., also during keystream generation) used for updating parts of the volatile inner state, which consists of a 40-bit NFSR, a linked 40-bit LFSR with a primitive feedback polynomial, and a 9-bit counter.

As done in [AM15], we denote the feedback function of the NFSR, the feedback function of the LFSR, and the nonlinear filter function by g, f, and h, respectively. The lower 7 bits of the 9-bit counter, denoted by $(c_t^6, c_t^5, c_t^4, c_t^3, c_t^2, c_t^1, c_t^0)$ at time t, realize a modulo 80 counter. As depicted in Fig. 6.1, the 5th LSB c_t^4 of this counter is involved in the NFSR's state update. Being part of a modulo 80 counter, the largest possible cycle length for c_t^4 is obviously 80 and, as can be checked easily, it in fact reaches this maximum.

Let $L^t = (l_t, l_{t+1}, \ldots, l_{t+39})$ and $N^t = (n_t, n_{t+1}, \ldots, n_{t+39})$ denote the inner states of the LFSR and the NFSR at time t, respectively. The 40-bit LFSR is updated by f as

$$l_{t+40} = f\left(L^t\right) = l_t \oplus l_{t+5} \oplus l_{t+15} \oplus l_{t+20} \oplus l_{t+25} \oplus l_{t+34}.$$

The 40-bit NFSR is updated by g, together with the bits k_t^*, l_t, and c_t^4, as

$$\begin{aligned} n_{t+40} &= k_t^* \oplus l_t \oplus c_t^4 \oplus g\Big(N^t\Big) \\ &= k_t^* \oplus l_t \oplus c_t^4 \oplus n_t \oplus n_{t+13} \oplus n_{t+19} \oplus n_{t+35} \oplus n_{t+39} \oplus n_{t+2}n_{t+25} \\ &\quad \oplus n_{t+3}n_{t+5} \oplus n_{t+7}n_{t+8} \oplus n_{t+14}n_{t+21} \oplus n_{t+16}n_{t+18} \oplus n_{t+22}n_{t+24} \\ &\quad \oplus n_{t+26}n_{t+32} \oplus n_{t+10}n_{t+11}n_{t+12} \oplus n_{t+27}n_{t+30}n_{t+31} \\ &\quad \oplus n_{t+33}n_{t+36}n_{t+37}n_{t+38}, \end{aligned}$$

where the round key bit k_t^* is defined through

$$k_t^* = \begin{cases} k_t, & 0 \leq t \leq 79, \\ k_{(t \bmod 80)} \cdot u_t, & t \geq 80, \end{cases} \tag{6.1}$$

with $u_t = l_{t+4} \oplus l_{t+21} \oplus l_{t+37} \oplus n_{t+9} \oplus n_{t+20} \oplus n_{t+29}$.

Given the FSR states L^t and N^t at time t, the keystream bit z_t is generated as

$$\begin{aligned} z_t = {} & h(n_{t+4}, l_{t+6}, l_{t+8}, l_{t+10}, l_{t+32}, l_{t+17}, l_{t+19}, l_{t+23}, n_{t+38}) \\ & \oplus \Big(\bigoplus\nolimits_{i \in B} n_{t+i}\Big) \oplus l_{t+30}, \end{aligned}$$

where $B = \{1, 6, 15, 17, 23, 28, 34\}$ and the nonlinear filter function is

$$h(\cdot) = n_{t+4}l_{t+6} \oplus l_{t+8}l_{t+10} \oplus l_{t+32}l_{t+17} \oplus l_{t+19}l_{t+23} \oplus n_{t+4}l_{t+32}n_{t+38}.$$

The state initialization algorithm of Sprout works as follows. As a first step, the IV and a 10-bit constant are loaded to the FSR cells:

$$\begin{aligned} n_i &= IV_i, \ 0 \leq i \leq 39, \\ l_j &= \begin{cases} IV_{j+40}, & 0 \leq j \leq 29, \\ 1, & 30 \leq j \leq 38, \\ 0, & j = 39. \end{cases} \end{aligned}$$

Note that, as the key will be continuously involved in the state update, it is not loaded to the FSRs at this point in time (in contrast to, e.g., Trivium and Grain v1; cf. Subsections 5.2.3 and 5.2.4, respectively). Subsequent to the IV loading, the cipher is then clocked 320 times under the modified update relations

$$\begin{aligned} l_{t+40} &= z_t \oplus f\Big(L^t\Big), \\ n_{t+40} &= z_t \oplus k_t^* \oplus l_t \oplus c_t^4 \oplus g\Big(N^t\Big). \end{aligned}$$

Following this stage, during which z_t is kept internal, the keystream generation phase starts and z_t is output for encryption purposes instead of being fed back to the FSRs (just like in the case of Grain v1).

Shortly after the new design had been proposed, various cryptanalytic results for Sprout were published in quick succession. It started with a related key attack by Hao [Hao15], which, however, cannot be considered a break of the cipher, because Armknecht and Mikhalev had stated in [AM15] that "as the key is assumed to be fixed, related-key attacks are out of scope". The first real attack against Sprout, targeting a single key scenario, is due to Lallemand and Naya-Plasencia [LNP15]. It has a time complexity of around 2^{69} Sprout encryptions and is based on a divide-and-conquer approach using list merging techniques. In [MSBD15], an attack using SAT solvers (see Section 9.4) is described by Maitra et al. It uses a guess-and-determine approach requiring 2^{54} attempts (i.e., it guesses 54 bits of the volatile inner state), where each of these trials results in a SAT instance that can be solved within about one minute on a standard PC. Given the speed of modern CPUs, it is unclear, however, whether the results in [MSBD15] actually lead to a real attack with computational cost below that of exhaustive key search.

Particularly interesting is the approach of Esgin and Kara in [EK16], as they mount a TMD-TO attack with overall complexity below 2^{45} against Sprout, whose major design promise was, in fact, to protect against this type of attacks by continuously using the secret key. Esgin and Kara circumvent the corresponding challenges through exploiting the nonlinearity of Sprout's round key function. More precisely, in those steps t, $t \geq 80$, where the bit u_t in the above Eq. (6.1) takes the value 0, no key bit is involved in the state update. Note that our new TMD-TO-based key recovery attack against Fruit v1, which is presented in Subsection 6.3.2, makes use of the cipher's complicated, nonlinear key schedule as well.

In [Ban15], another SAT solver-based attack is proposed by Banik, which requires to guess slightly less bits (i.e., 50 instead of 54 bits) of Sprout's FSRs than the aforementioned attack by Maitra et al. Moreover, in the same paper, Banik proposes a key recovery attack with time complexity about $2^{66.7}$ Sprout encryptions, requiring negligible memory. The currently best attack against Sprout is due to Zhang and Gong [ZG15]. Like Esgin and Kara, they use a TMD-TO approach and claim that "[w]ith carefully chosen parameters, the new attack is at least 2^{20} times faster than Lallemand/Naya-Plasencia attack at Crypto 2015, Maitra et al. attack and Banik attack, 2^{10} times faster than Esgin/Kara attack with much less memory."

6.2.2 Fruit

Like Sprout, Fruit [GHX16] adopts a Grain-like structure and utilizes an 80-bit secret key $k = (k_0, \ldots, k_{79})$ together with a 70-bit public initialization vector $IV = (IV_0, \ldots, IV_{69})$. As pointed out in Section 6.1, the designers of Fruit have changed their cipher's specification several times in the past (mostly in reaction to attacks). In this subsection,

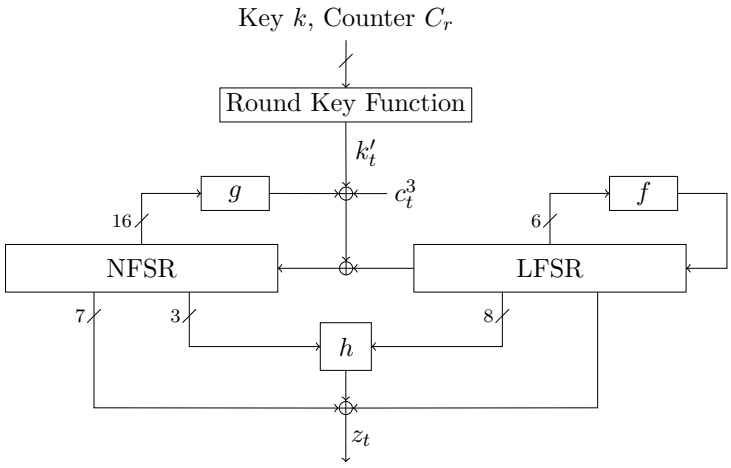

Figure 6.2: Keystream generation of Fruit v1 (i.e., ePrint version 20170304:073404 of [GHX16]).

we describe the variant given in ePrint version 20170304:073404 of [GHX16] (calling it Fruit v1), as it was the most recent at the time of our key recovery attack presented in Subsection 6.3.2.[2] Moreover, we will also explain how Fruit v1 was patched to Fruit v2.

Following the new design paradigm of Sprout, in Fruit v1, the secret key is continuously used for updating parts of the volatile inner state, which consists of a 37-bit NFSR, a linked 43-bit LFSR with a primitive feedback polynomial, and two counters of size 7 bits and 8 bits, respectively. The cipher's general structure is depicted in Fig. 6.2.

As done for Sprout and Grain v1, we denote the feedback function of the NFSR, the feedback function of the LFSR, and the nonlinear filter function by g, f, and h, respectively. Following the original cipher description in [GHX16], at time t, the two counters (which work independently) are denoted by $C_r = (c_t^0, \ldots, c_t^6)$ and $C_c = (c_t^7, \ldots, c_t^{14})$ and their LSBs are c_t^6 and c_t^{14}, respectively. While C_r is used for the round key function and as part of the NFSR's state update, C_c only serves to count the steps during state initialization.

Let $L^t = (l_t, l_{t+1}, \ldots, l_{t+42})$ and $N^t = (n_t, n_{t+1}, \ldots, n_{t+36})$ denote the inner states of the LFSR and the NFSR at time t, respectively. The 43-bit LFSR is updated by f as

$$l_{t+43} = f\left(L^t\right) = l_t \oplus l_{t+8} \oplus l_{t+18} \oplus l_{t+23} \oplus l_{t+28} \oplus l_{t+37}.$$

[2]Note that, in reaction to our key recovery attack, the designers of Fruit have updated their scheme to Fruit v2 (ePrint version 20170724:053140 of [GHX16]). While, prior to that, they had simply called it "Fruit", in the appendix of 20170724:053140, they now refer to all earlier versions as "Fruit v1". In this thesis, however, we use the name Fruit v1 strictly for the cipher specification given in ePrint version 20170304:073404, just as we have done in our main publication [HKMZ18] and our underlying ePrint paper [HKMZ17b]. By this, we also mean to clarify that our attack in Subsection 6.3.2 was the only one working for Fruit prior to its update to Fruit v2; i.e., due to corresponding patches, previous attacks (like those in [DS17] and [HKMZ17a]) could not be applied anymore against what we call Fruit v1 here.

The 37-bit NFSR is updated by g, together with the bits k'_t, l_t, and c_t^3, as

$$\begin{aligned} n_{t+37} &= k'_t \oplus l_t \oplus c_t^3 \oplus g\left(N^t\right) \\ &= k'_t \oplus l_t \oplus c_t^3 \oplus n_t \oplus n_{t+10} \oplus n_{t+20} \oplus n_{t+12}n_{t+3} \oplus n_{t+14}n_{t+25} \\ &\quad \oplus n_{t+5}n_{t+23}n_{t+31} \oplus n_{t+8}n_{t+18} \oplus n_{t+28}n_{t+30}n_{t+32}n_{t+34}, \end{aligned}$$

where k'_t denotes the output of the round key function at time t.

As in the original description [GHX16], let the (decimal) values of s, y, u, p, q, r be defined through the bits of the counter C_r as

$$\begin{aligned} s &= \left(c_t^0, c_t^1, c_t^2, c_t^3, c_t^4, c_t^5\right),\ y = \left(c_t^3, c_t^4, c_t^5\right),\ u = \left(c_t^4, c_t^5, c_t^6\right), \\ p &= \left(c_t^0, c_t^1, c_t^2, c_t^3, c_t^4\right),\ q = \left(c_t^1, c_t^2, c_t^3, c_t^4, c_t^5\right),\ r = \left(c_t^3, c_t^4, c_t^5, c_t^6\right), \end{aligned}$$

where, in each case, the rightmost bit is the LSB of the binary representation of the corresponding natural number. Then for all $t \geq 0$, the round key bit k'_t of Fruit v1 is computed as

$$k'_t = k_s k_{y+64} \oplus k_p k_{u+72} \oplus k_{q+32} \oplus k_{r+64}. \tag{6.2}$$

Given the FSR states L^t and N^t, the nonlinear filter function h computes the value

$$h_t = l_{t+6}l_{t+15} \oplus l_{t+1}l_{t+22} \oplus n_{t+35}l_{t+27} \oplus l_{t+11}l_{t+33} \oplus n_{t+1}n_{t+33}l_{t+42}.$$

The keystream bit z_t at time t is then generated as

$$z_t = h_t \oplus n_t \oplus n_{t+7} \oplus n_{t+13} \oplus n_{t+19} \oplus n_{t+24} \oplus n_{t+29} \oplus n_{t+36} \oplus l_{t+38}.$$

The state initialization algorithm of Fruit v1 works as follows. As a first step, the 80-bit key is loaded to the FSR cells through

$$\begin{aligned} n_i &= k_i,\ 0 \leq i \leq 36, \\ l_j &= k_{j+37},\ 0 \leq j \leq 42 \end{aligned}$$

and both counters are set to 0. Then, the 70-bit initialization vector IV is extended to 130 bits as

$$IV' = (IV'_0, \ldots, IV'_{129}) = (\underbrace{1,0,0,0,0,0,0,0,0,0}_{10}, \underbrace{IV_0, \ldots, IV_{69}}_{70}, \underbrace{0, \ldots, 0}_{50})$$

and the cipher is clocked 130 times under the modified update relations

$$l_{t+43} = z_t \oplus IV'_t \oplus f\left(L^t\right),$$

$$n_{t+37} = z_t \oplus IV'_t \oplus k'_t \oplus l_t \oplus c_t^3 \oplus g\Big(N^t\Big).$$

Following this, the seven (henceforth secret) bits of the counter C_r are overwritten as

$$c_{130}^0 = n_{130}, c_{130}^1 = n_{131}, \ldots, c_{130}^5 = n_{135}, c_{130}^6 = l_{130}$$

based on six bits from N^{130} and one bit from L^{130}. Subsequently, the bit l_{130} of L^{130} is overwritten with 1 in order to avoid the all-zero state for the LFSR. As the final step of the state initialization algorithm, Fruit v1 is then clocked 80 times in keystream generation mode (i.e., without feeding z_t and IV' any longer to the FSRs; cf. Fig. 6.2), but without outputting z_t. After that, the actual keystream generation phase starts and the first keystream bit used for plaintext encryption is z_{210}.

Two attacks against the prior versions of what we call Fruit v1 here are known. The first is due to Dey and Sarkar [DS17]. Using a divide-and-conquer approach and sieving techniques, they are able to recover the secret key on the basis of about 2^{75} Fruit encryptions. The second attack is a fast correlation attack by Meier and Zhang [HKMZ17a] and was presented at ESC 2017.[3] It allows to recover the initial states of the two FSRs and the 128-bit *round key* with complexity about 2^{69}. As the counter C_r underlying the round key function (cf. Eq. (6.2)) is only 7 bits wide, the maximum possible cycle length for the round key bit k'_t during keystream generation is obviously 128. Hence, knowing the initial FSR states and this 128-bit round key allows for decrypting the respective ciphertext. Besides the actual attack against Fruit, an important contribution of [HKMZ17a] was the derivation of corresponding design principles for Grain-like small-state stream ciphers. In Section 6.4, we will treat these new criteria in further detail.

As pointed out above, both of these attacks do not work for Fruit v1 any more due to corresponding design patches. However, in Subsection 6.3.2, we demonstrate that Fruit v1 has at least 2^{64} weak keys, each of which does not provide the 80-bit security promised by the designers. Moreover, our generic distinguisher for small-state stream ciphers which continuously use the secret key, presented in Subsection 6.3.1, can be applied as well.

As a consequence of our key recovery attack, the designers of Fruit patched Fruit v1, resulting in the new specification Fruit v2 (ePrint version 20170724:053140 of [GHX16]). More precisely, they changed the round key function to

$$k'_t = k_s k_{y+32} \oplus k_p k_{u+64} \oplus k_{q+16} \oplus k_{r+48}$$

with $s = (c_t^0, c_t^1, c_t^2, c_t^3, c_t^4)$, $y = (c_t^5, c_t^6, c_t^0, c_t^1, c_t^2)$, $u = (c_t^3, c_t^4, c_t^5, c_t^6)$, $p = (c_t^0, c_t^1, c_t^2, c_t^3)$, $q = (c_t^4, c_t^5, c_t^6, c_t^0, c_t^1)$, $r = (c_t^2, c_t^3, c_t^4, c_t^5, c_t^6)$. The new version is not susceptible to our key recovery attack any longer, but, as we discuss in Section 6.4, the question remains, in what respect the complicated key schedule of Fruit is actually superior to the simple one

[3] Note that, very recently, the results of Meier and Zhang given in [HKMZ17a] were published as a full paper by Zhang, Gong, and Meier [ZGM17]. However, for reasons of chronological consistency, we will keep referring to the original presentation [HKMZ17a] in the rest of this chapter.

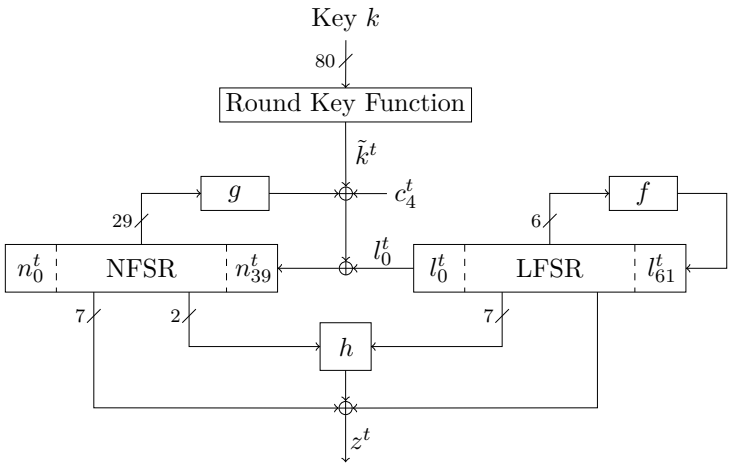

Figure 6.3: Keystream generation of Plantlet. (cf. [MAM17])

used in Plantlet (see below). It should also be noted that our generic distinguisher in Subsection 6.3.1 still works for Fruit v2.

6.2.3 Plantlet

In reaction to the aforementioned attacks against Sprout (cf. Subsection 6.2.1), the designers presented a modified version at FSE 2017 [MAM17], calling it Plantlet. The structure of Plantlet, which takes 80-bit keys and 90-bit IVs, is depicted in Fig. 6.3. For the sake of brevity, we will focus on the changes compared to Sprout in this subsection. Moreover, as the designers of Sprout and Plantlet have changed their notation between the corresponding works, we will also use their new notation for Plantlet here.

Plantlet consists of a 40-bit NFSR, a 61-bit LFSR, and a 9-bit counter, the latter of which works exactly like the counter in Sprout (cf. Subsection 6.2.1). In particular, like in Sprout, the 5th LSB c_4^t of the counter is employed in the state update during keystream generation. The most prominent change w.r.t. the predecessor Sprout is the simplified round key function of Plantlet: $\tilde{k}^t = k_{(t \bmod 80)}$, $t \geq 0$. In Section 6.4, we will conjecture that despite its simplicity, this key schedule is actually the optimal one w.r.t. countering TMD-TO key recovery attacks for ciphers that continuously use the non-volatile key.

During initialization (which, like in Sprout, takes 320 clock cycles), the LFSR of Plantlet is updated as

$$\begin{aligned} l_{60}^{t+1} &= 1, \\ l_{59}^{t+1} &= l_{54}^t \oplus l_{43}^t \oplus l_{34}^t \oplus l_{20}^t \oplus l_{14}^t \oplus l_0^t \oplus z^t, \\ l_i^{t+1} &= l_{i+1}^t,\ 0 \leq i \leq 58, \end{aligned} \tag{6.3}$$

for $0 \leq t \leq 319$. For keystream generation (i.e., $t \geq 320$), it then changes to

$$\begin{aligned} l_{60}^{t+1} &= l_{54}^t \oplus l_{43}^t \oplus l_{34}^t \oplus l_{20}^t \oplus l_{14}^t \oplus l_0^t, \\ l_i^{t+1} &= l_{i+1}^t, \ 0 \leq i \leq 59. \end{aligned}$$

Note that during initialization, the LSB of the 61-bit LFSR is ignored, turning it effectively into a 60-bit LFSR. However, though none of the taps change between both stages, the corresponding feedback polynomials of the respective LFSRs are both primitive. The designers of Plantlet call this property *double-layer LFSR*.[4]

The NFSR feedback is identical to that of Sprout, i.e.,

$$\begin{aligned} n_{39}^{t+1} &= \tilde{k}^t \oplus l_0^t \oplus c_4^t \oplus n_0^t \oplus n_{13}^t \oplus n_{19}^t \oplus n_{35}^t \oplus n_{39}^t \\ &\quad \oplus n_2^t n_{25}^t \oplus n_3^t n_5^t \oplus n_7^t n_8^t \oplus n_{14}^t n_{21}^t \oplus n_{16}^t n_{18}^t \oplus n_{22}^t n_{24}^t \oplus n_{26}^t n_{32}^t \\ &\quad \oplus n_{10}^t n_{11}^t n_{12}^t \oplus n_{27}^t n_{30}^t n_{31}^t \oplus n_{33}^t n_{36}^t n_{37}^t n_{38}^t, \\ n_i^{t+1} &= n_{i+1}^t, \ 0 \leq i \leq 38, \end{aligned}$$

for $t \geq 320$. During initialization (i.e., for $0 \leq t \leq 319$), the linear term z^t is XORed additionally to the NFSR update (and the LFSR update; see Eq. (6.3) above), just like in Sprout.

The output function of Plantlet has the form

$$z^t = h^t \oplus l_{30}^t \oplus \left(\bigoplus\nolimits_{i \in B} n_i^t \right),$$

where $B = \{1, 6, 15, 17, 23, 28, 34\}$ and

$$h^t = n_4^t l_6^t \oplus l_8^t l_{10}^t \oplus l_{32}^t l_{17}^t \oplus l_{19}^t l_{23}^t \oplus n_4^t l_{32}^t n_{38}^t.$$

After loading the 90-bit IV via $n_i^0 = IV_i$, $0 \leq i \leq 39$, and $l_{i-40}^0 = IV_i$, $40 \leq i \leq 89$, to the FSR cells and setting $l_{50}^0 = \ldots = l_{58}^0 = 1$, $l_{59}^0 = 0$, $l_{60}^0 = 1$ at $t = 0$, the state initialization and the keystream generation work analogously to the corresponding phases in Sprout based on the above components (see Subsection 6.2.1 for further details).

So far, the only cryptanalytic result for Plantlet is a differential fault attack by Maitra and Siddhanti [SM17]. As the designers of Plantlet do not make any claims about this scenario and given that such attacks are hard to realize, Plantlet can currently be considered the only unbroken Sprout-like cipher.

In Subsection 6.3.1, we present a generic TMD-TO distinguishing attack against Plantlet with overall complexity 2^{61}. While, based on a similar distinguishing attack

[4] While, from a theoretical point of view, this looks very elegant, it's not really clear whether the security of Plantlet does actually benefit from the fact that the LFSR feedback polynomial is also primitive during initialization. After all, due to the additional feedback of the output bit z^t, the LFSR's period between $t = 0$ and $t = 319$ is unclear anyhow.

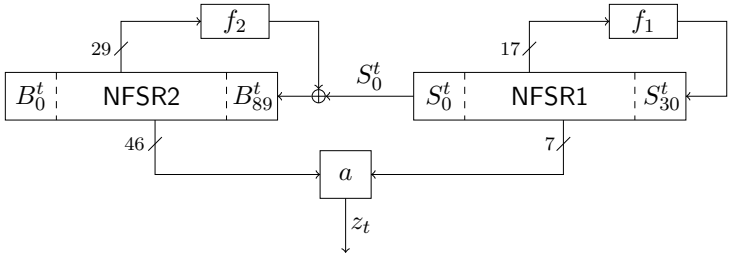

Figure 6.4: Keystream generation of Lizard.

against Sprout by Banik [Ban15], the designers of Plantlet acknowledge the existence of such attacks (hence, we do not claim to break Plantlet), they also point out in [MAM17] that a more complicated round key function would allow to thwart them. The generic nature of our distinguishing attack will show that this is not the case.

6.2.4 LIZARD

While, w.r.t. key recovery attacks, Sprout-like stream ciphers in fact allow to reduce the size of the volatile inner state below the birthday bound, the underlying principle of continuously accessing the key from a separate non-volatile memory like an EEPROM has two severe drawbacks as pointed out in Subsection 2.3.9 and Section 6.1: (i) reading from an EEPROM is costly in terms of power, hence continuously accessing it during keystream generation can, e.g., have a devastating impact on the energy budget of a battery-powered RFID device (like a pace maker); (ii) accessing the EEPROM is also a bottle neck w.r.t. the maximum possible speed for implementations of such ciphers.

Our lightweight stream cipher Lizard [HKM17b] uses another approach to reduce the size of the inner state without suffering from the above drawbacks. Its state initialization algorithm is based on our new design principle named Lizard-*construction* [HK18], which allows for provable $(2n/3)$-security against generic TMD-TO key recovery attacks. For an in-depth description and analysis of the Lizard-construction and its concrete instantiation Lizard, we refer the reader to Chapters 7 and 8, respectively. As pointed out in Section 6.1, in this chapter, we only provide a brief outline of Lizard's structure for the sake of drawing the full picture regarding the current state of the art in small-state stream cipher development.

Lizard takes 120-bit keys, 64-bit IVs and has an inner state length of 121 bits. It allows to generate up to 2^{18} keystream bits per key/IV pair and is supposed to provide 80-bit security against key recovery attacks and 60-bit security against distinguishing. Also note that, unlike Sprout, Plantlet, and Fruit, the tap positions in Lizard retain the original Grain option of parallelization (up to a factor of 6) via duplicating the (comparatively cheap) circuits of the feedback and output functions (cf. Subsection 8.3.4).

Lizard has a Grain-like structure (see Fig. 6.4) with the difference that instead of

Grain's maximum-length LFSR, a maximum-length NFSR is used (NFSR1), which is 31 bits wide and has the update relation

$$\begin{aligned}
S_{30}^{t+1} &= S_0^t \oplus S_2^t \oplus S_5^t \oplus S_6^t \oplus S_{15}^t \oplus S_{17}^t \oplus S_{18}^t \oplus S_{20}^t \oplus S_{25}^t \\
&\oplus S_8^t S_{18}^t \oplus S_8^t S_{20}^t \oplus S_{12}^t S_{21}^t \oplus S_{14}^t S_{19}^t \oplus S_{17}^t S_{21}^t \oplus S_{20}^t S_{22}^t \\
&\oplus S_4^t S_{12}^t S_{22}^t \oplus S_4^t S_{19}^t S_{22}^t \oplus S_7^t S_{20}^t S_{21}^t \oplus S_8^t S_{18}^t S_{22}^t \\
&\oplus S_8^t S_{20}^t S_{22}^t \oplus S_{12}^t S_{19}^t S_{22}^t \oplus S_{20}^t S_{21}^t S_{22}^t \oplus S_4^t S_7^t S_{12}^t S_{21}^t \\
&\oplus S_4^t S_7^t S_{19}^t S_{21}^t \oplus S_4^t S_{12}^t S_{21}^t S_{22}^t \oplus S_4^t S_{19}^t S_{21}^t S_{22}^t \\
&\oplus S_7^t S_8^t S_{18}^t S_{21}^t \oplus S_7^t S_8^t S_{20}^t S_{21}^t \oplus S_7^t S_{12}^t S_{19}^t S_{21}^t \\
&\oplus S_8^t S_{18}^t S_{21}^t S_{22}^t \oplus S_8^t S_{20}^t S_{21}^t S_{22}^t \oplus S_{12}^t S_{19}^t S_{21}^t S_{22}^t, \\
S_i^{t+1} &= S_{i+1}^t,\ 0 \le i \le 29.
\end{aligned}$$

NFSR2 (90 bits) is based on Grain-128a [gHJM11] and has the update relation

$$\begin{aligned}
B_{89}^{t+1} &= S_0^t \oplus B_0^t \oplus B_{24}^t \oplus B_{49}^t \oplus B_{79}^t \oplus B_{84}^t \oplus B_3^t B_{59}^t \oplus B_{10}^t B_{12}^t \\
&\oplus B_{15}^t B_{16}^t \oplus B_{25}^t B_{53}^t \oplus B_{35}^t B_{42}^t \oplus B_{55}^t B_{58}^t \oplus B_{60}^t B_{74}^t \\
&\oplus B_{20}^t B_{22}^t B_{23}^t \oplus B_{62}^t B_{68}^t B_{72}^t \oplus B_{77}^t B_{80}^t B_{81}^t B_{83}^t, \\
B_j^{t+1} &= B_{j+1}^t,\ 0 \le j \le 88.
\end{aligned}$$

The output function of Lizard builds on the construction scheme introduced in [MJSC16] as part of the FLIP family of stream ciphers. The output bit z_t at time t is computed as $z_t = \mathcal{L}_t \oplus \mathcal{Q}_t \oplus \mathcal{T}_t \oplus \tilde{\mathcal{T}}_t$, where

$$\begin{aligned}
\mathcal{L}_t &= B_7^t \oplus B_{11}^t \oplus B_{30}^t \oplus B_{40}^t \oplus B_{45}^t \oplus B_{54}^t \oplus B_{71}^t, \\
\mathcal{Q}_t &= B_4^t B_{21}^t \oplus B_9^t B_{52}^t \oplus B_{18}^t B_{37}^t \oplus B_{44}^t B_{76}^t, \\
\mathcal{T}_t &= B_5^t \oplus B_8^t B_{82}^t \oplus B_{34}^t B_{67}^t B_{73}^t \oplus B_2^t B_{28}^t B_{41}^t B_{65}^t \\
&\quad \oplus B_{13}^t B_{29}^t B_{50}^t B_{64}^t B_{75}^t \oplus B_6^t B_{14}^t B_{26}^t B_{32}^t B_{47}^t B_{61}^t \\
&\quad \oplus B_1^t B_{19}^t B_{27}^t B_{43}^t B_{57}^t B_{66}^t B_{78}^t, \\
\tilde{\mathcal{T}}_t &= S_{23}^t \oplus S_3^t S_{16}^t \oplus S_9^t S_{13}^t B_{48}^t \oplus S_1^t S_{24}^t B_{38}^t B_{63}^t.
\end{aligned}$$

The state initialization of Lizard, during which no keystream is output, proceeds in four phases. At $t = 0$, the key $K = (K_0, \ldots, K_{119})$ and the initialization vector $IV = (IV_0, \ldots, IV_{63})$ are loaded to the registers of the KSG as follows:

$$B_j^0 = \begin{cases} K_j \oplus IV_j, & 0 \le j \le 63, \\ K_j, & 64 \le j \le 89, \end{cases}$$

$$S_i^0 = \begin{cases} K_{i+90}, & 0 \le i \le 28, \\ K_{119} \oplus 1, & i = 29, \\ 1, & i = 30. \end{cases}$$

Then, the KSG is clocked 128 times in the usual Grain-like manner, i.e., the output bit z_t is XORed to the feedback of both FSRs. After that, the secret key is used a second (and last) time as follows:

$$B_j^{129} = B_j^{128} \oplus K_j,\ 0 \leq j \leq 89,$$
$$S_i^{129} = \begin{cases} S_i^{128} \oplus K_{i+90}, & 0 \leq i \leq 29, \\ 1, & i = 30. \end{cases}$$

The final phase of Lizard's state initialization consists in clocking the KSG 128 times already in keystream generation mode (i.e., without XORing the output bit z_t to the FSRs' feedback), but discarding the corresponding keystream bits. The first keystream bit actually used for encryption is z_{257}.

For a summary of 'external' cryptanalytic results for Lizard, we refer the reader to Subsection 8.4.11. Let us, however, emphasize here that Lizard is currently unbroken.

6.3 TMD-TO Attacks against Sprout-like Stream Ciphers

Time-memory-data tradeoff (TMD-TO) attacks like those of Babbage [Bab95] and Biryukov and Shamir [BS00] are probably the most powerful generic attacks against stream ciphers. For classical stream ciphers like Trivium (cf. Subsection 5.2.3) and Grain (cf. Subsection 5.2.4), which generate the keystream exclusively based on a so-called *initial state* (see our model in Section 5.1) and have an efficiently invertible state initialization algorithm (cf. Section 5.3), such attacks reduce the maximum possible security level to half the size of the inner state of the KSG. As a consequence, stream ciphers used to have comparatively large inner states. In 2015, Armknecht and Mikhalev suggested Sprout (cf. Subsection 6.2.1), which tried to beat this birthday bound.

While classical TMD-TO attacks aiming at inner state recovery indeed cannot be applied anymore (straightforwardly) against Sprout-like ciphers, we will now show that they are still susceptible to another type of generic TMD-TO distinguishing attack. Moreover, we will also present the first key recovery attack against Fruit v1 (our name for the specification in ePrint version 20170304:073404 of [GHX16]). Our key recovery attack makes use of the complicated round key function of Fruit v1 and raises the question, whether simplicity (as present in Plantlet) wouldn't actually be preferable here.

We would like to point out that, in this chapter, no particular familiarity with the concept of TMD-TO-based key recovery and distinguishing attacks is required on the reader's side. In particular, all relevant details will be introduced, as required, in the course of our respective attacks. For a more formal exposition, we refer the reader to the following Chapters 7 and 8 (esp. Section 7.3 and Subsection 8.4.2, respectively).

6.3.1 A Generic Distinguishing Attack against Stream Ciphers which Continuously use the Non-volatile Key

In this subsection, we present a generic distinguishing attack against stream ciphers which continuously use the non-volatile key. Our description is based on a result by Englund, Hell, and Johansson [EHJ07] from 2007 and hence also related to the distinguishing attack of Banik against Sprout [Ban15] from 2015.

Revisiting the Generic Attack of Englund, Hell, and Johansson

In 2007, Englund, Hell, and Johansson published a note [EHJ07], in which they presented a "new distinguishing attack scenario for stream ciphers, allowing a resynchronization collision attack" and pointed out that "[t]he attack can succeed if the part of the state that depends on both the key and the IV is smaller than twice the key size." At the time of publication, the authors did not have ciphers like Sprout, Fruit, or Plantlet in mind (the first of which was only published in 2015), but they were targeting block ciphers in output feedback mode (OFB mode). These can also be interpreted as (though not bitwise working) stream ciphers and, in fact, the eSTREAM phase-3 candidate LEX [Bir05] (see Subsection 5.2.5) is very similar to a block cipher in OFB mode, so the attack of Englund, Hell, and Johansson could be applied. They start the description of their idea by:

> "Let us divide the internal state of the cipher into two parts, $State = (State_K, State_{K+IV})$, where $State_K$ is the part of the state that statically holds the key and $State_{K+IV}$ is the part of the state that is affected by both the key and the IV. Also, let N be the size of $State_{K+IV}$ in bits. If the key size $|K| > N/2$, then the new distinguishing attack will always succeed with complexity below exhaustive key search." [EHJ07]

Note that this corresponds exactly to the scenario we are facing for Sprout, Fruit and Plantlet. The attack description then continues:

> "Consider a resynchronization scenario. We assume that the key is fixed and that the cipher is reinitialized using many different IVs. Furthermore, we assume that we have access to one long keystream sequence produced from one of the IVs, denoted IV_0. We then intercept many short ciphertext messages, each initialized using a different IV, and we assume that we know the first N plaintext bits in every ciphertext message. Our goal is to recover the rest of the plaintext for one of the messages. [...] We now proceed as follows. Let a keystream block be N consecutive bits in a considered keystream sequence. We first store $2^{N/2}$ different keystream blocks from the keystream generated from IV_0 in a sorted table. Thus, we have a table covering a fraction of $2^{-N/2}$ of all possible keystream blocks of length N. If the cipher is reinitialized with a new IV and we know the N first bits of the corresponding keystream,

> then the number of reinitializations needed in order to have a collision, i.e., receiving N bits that are present in the table, is geometrically distributed with expected value $2^{N/2}$. Denote the IV producing the collision by IV_c. If a collision is found, then with high probability the states are the same and the sequences following the colliding blocks of IV_0 and IV_c will be identical. That means that if we know only the first N keystream bits generated by IV_c, then we can predict future keystream bits from IV_c. In other words, by knowing only the first N plaintext bits of the message, we can decrypt the rest of the ciphertext without knowing the key." [EHJ07]

Depending on the application context and the attack complexity, the fact that the attacker is not only able to distinguish the cipher from a random bitstream but also obtains a potentially large amount of previously unknown keystream for the collision initialization vector IV_c, may pose a serious security risk.

Also note that what this attack effectively does is to look for two different IVs which map to shifted versions of the same keystream. The same idea underlies, e.g., Banik's distinguishing attack against Sprout [Ban15]. However, the attack by Englund, Hell, and Johansson cannot be applied carelessly to stream ciphers like Sprout, Fruit, and Plantlet. The reason for this lies in their assumption that

> "[i]f the cipher is reinitialized with a new IV and we know the N first bits of the corresponding keystream, then the number of reinitializations needed in order to have a collision, i.e., receiving N bits that are present in the table, is geometrically distributed with expected value $2^{N/2}$." [EHJ07]

This statement is obviously motivated by their application context of block ciphers in OFB mode. There, the IV serves as the (volatile) initial state and thus, the IV space and the space of (volatile) inner states have the same size. In fact, at a later point, the authors even write:

> "This resynchronization scenario will not give an attack on a block cipher in counter mode. [...] However, the above scenario applies to block ciphers used in OFB mode since the output block z_i can be viewed as the part of the state that depends on both the key and the IV, $State_{K+IV}$. Similarly, the key used in the block cipher can be viewed as the part of the state that statically holds the key, $State_K$. By reinitializing the OFB mode stream cipher with a new IV, the cipher will enter a new random state after every encryption." [EHJ07]

Note that for stream ciphers it is common to have the IV size below the total size of the volatile inner state (see, e.g., Trivium and Grain, but also Sprout, Fruit, Plantlet, and Lizard). Hence, reinitializing with a new IV will only allow us to randomly draw elements from a *subset* of the set of all volatile inner states. We will thus see that for a generic attack description against general stream ciphers, we need additional assumptions.

Such assumptions are also necessary because, for a block cipher in OFB mode, the mapping of IVs to initial states is the identity and, hence, obviously injective. While, under an arbitrarily fixed key, the respective mappings for Sprout and Plantlet are injective, too, this is not clear for Fruit due to the fact that the IV is introduced 'from aside' during the state initialization. As a consequence, in our experiment, we would not draw *uniformly* at random from the set of all possible initial states as some of those states would be more likely due to the fact that they are the result of more than one IV.

Another important difference between a block cipher in OFB mode and common stream ciphers results from the fact that a block cipher realizes a permutation between plaintext blocks and ciphertext blocks. Hence, for a block cipher in OFB mode, the mapping of inner states (corresponding to the block cipher's input blocks) to keystream blocks (corresponding to the block cipher's output blocks) is also a permutation. For common stream ciphers, however, we do not have this guarantee as different inner states of size n can actually lead to identical keystream blocks of size n. As a consequence, when straightforwardly applying the attack of Englund, Hell, and Johansson to ciphers like Sprout, Fruit, and Plantlet, we might actually experience keystream block collisions which are not the result of colliding inner states but of collisions in the output function. To avoid such 'false positives', we need to slightly increase the size of the keystream blocks which we save (for practical attacks, usually by only a few bits; see below).

We will now describe a generic attack scheme for stream ciphers that continuously use the non-volatile key, which takes the above considerations into account. In particular, explicitly specifying the required assumptions will enable us to come up with a new design idea in Section 6.5 that thwarts this type of distinguishing attack. More precisely, we will extend the 2-tuple $State = (State_K, State_{K+IV})$ underlying the model of Englund, Hell, and Johansson (and also Sprout-like ciphers) by a third component: $State_{IV}$.

A Generic Attack Scheme

Definition 6.1: Continuous-Key-Use (CKU) Stream Cipher

A *CKU stream cipher* is a standard KSG-based stream cipher (cf. Section 5.1) with the additional tweak that, even after the state initialization has been completed, the secret key is still used as an input to the state update function. The key schedule, which determines the way in which the secret key influences the state update, can depend on any part of the volatile inner state (FSRs, counters etc.).

Note that, e.g., Sprout, Fruit, and Plantlet are all CKU stream ciphers. Sprout's key schedule depends on a separate counter along with bits from the FSRs, while those of Fruit and Plantlet only depend on a counter.

In the following definition, we do not not speak of any counters, but only of a *volatile inner state*, which we consider potential counters to be part of (along with the FSRs

etc.). The reason for doing so is that we want to stay as generic as possible in our attack description in order to show its applicability for a wide range of CKU stream ciphers.

Definition 6.2

Let CIPHER be a CKU stream cipher with the following properties:

- n: size of the volatile inner state in bits;
- l: IV length in bits;
- 2^λ: limit of keystream bits per IV.

We define:

$$I_k = \left\{ \mathit{InitialState}_k(v) \mid v \in \{0,1\}^l \right\} \subseteq \{0,1\}^n ,$$

where $\mathit{InitialState}_k(v)$ denotes the initial state which the state initialization algorithm of CIPHER computes on the basis of the key k and the IV v.

For our generic distinguishing attack against CIPHER, we need the following two assumptions to hold.

Assumption 6.1: IV Near-Injectivity

CIPHER fulfills the *IV Near-Injectivity* assumption for the secret key k if $|I_k| \approx 2^l$.

Assumption 6.2: Initial State Randomness

Let $\sigma := \max\{0, (n/2 - \lambda)\}$, and let T be a set of about $2^{n/2}$ different $\tilde{n}$-bit keystream blocks (with $\tilde{n}$ slightly larger than n) that were obtained on the basis of $\lceil 2^\sigma \rceil$ different IVs under an arbitrarily fixed key k by sliding an $\tilde{n}$-bit window over each of the respective $\lceil 2^\sigma \rceil$ keystreams of length $\leq 2^\lambda$ bits without experiencing any collisions. Furthermore, let $S_k(T)$ denote the set of volatile inner states which underlie the keystream blocks in T for the secret key k.

CIPHER fulfills the *Initial State Randomness* assumption if for such sets T, it holds with high probability that

$$\frac{|S_k(T) \cap I_k|}{|S_k(T)|} \gtrapprox \frac{|I_k|}{|\{0,1\}^n|}.$$

Note that if a stream cipher violates Assumption 6.1, this itself opens the door for a distinguishing attack that looks for IVs which produce identical keystreams. For example,

if the IV size is smaller than the key size, even a single such IV collision allows for a distinguishing attack with complexity lower than that of exhaustive key search.[5]

Assumption 6.2 ensures that there will actually be a sufficient amount of collision candidates to look for. Banik makes a similar assumption in his distinguishing attack against Sprout in [Ban15] by expecting that for a randomly chosen inner state, there is a probability of 2^{-10} that an IV exists for which this state is "the 80th keystream phase state". We will now describe our generic distinguishing attack against CKU stream ciphers and explain its consequences for Fruit v1 and Plantlet afterwards.

So let Cipher be a CKU stream cipher with an n-bit volatile inner state, an IV length of l bits, and a limit of 2^λ keystream bits per IV. Moreover, let Cipher fulfill Assumptions 6.1 and 6.2, and let $\sigma := \max\{0, (n/2 - \lambda)\}$. Then the following algorithm allows to distinguish the keystream produced by Cipher from a random bitstream with high probability.

Distinguisher:

Step (1) Obtain about $2^{n/2}$ $\tilde{n}$-bit keystream blocks (with $\tilde{n}$ slightly larger than n) based on $\lceil 2^\sigma \rceil$ different IVs from the oracle by sliding an $\tilde{n}$-bit window over each of the respective $\lceil 2^\sigma \rceil$ keystreams of length $\leq 2^\lambda$ bits and save the keystream blocks in an efficiently searchable data structure like a hash table. If, during this step, a collision occurs, we can already distinguish Cipher and stop.

Step (2) For $2^{n/2}$ different IVs, obtain the corresponding $\tilde{n}$-bit keystream prefix from the oracle and look for a collision in the data structure created in *Step (1)*. Once such a collision is found, we can distinguish Cipher and stop.

[Output] If, during *Step (1)* or *Step (2)*, a collision was found, we assume that we are in the *pseudorandom scenario* of the distinguishing game (i.e., the oracle provides its responses based on the actual outputs of Cipher under some arbitrarily fixed secret key) and answer `pseudorandom`. Else, we assume that we are in the *random scenario* of the distinguishing game (i.e., the oracle answers our queries with random bitstrings) and answer `random`.

The success probability of the attack can be derived from the birthday paradox. Under an arbitrarily fixed key k, the universe of the corresponding experiment is the set of all possible initial states I_k, which, according to Assumption 6.1, has size $|I_k| \approx 2^l$. Assumption 6.2 ensures that, w.h.p., a subset of size about $2^{n/2-(n-l)} = 2^{l-n/2}$ of the volatile inner states underlying the $2^{n/2}$ keystream blocks collected in *Step (1)* will belong to the set I_k. Assumption 6.1 guarantees that in *Step (2)*, we draw uniformly and at random $2^{n/2}$ elements from I_k. According to the birthday paradox, when drawing

[5] A small amount of IV collisions may be tolerable depending on the security claims; see, e.g., Lizard, which claims 80-bit security against key recovery and 60-bit security against distinguishing.

uniformly and at random $2^{n/2}$ elements from a set of size 2^l, we are likely to find a collision with an arbitrarily fixed subset of size $2^{l-n/2}$.

The complexity of the above attack is:

(1) Obtain $2^{n/2}$ keystream blocks of size $\tilde{n}$ bits each and store them in an efficiently searchable data structure:

- Data complexity (keystream): $2^{n/2}$;
- Memory complexity (keystream blocks): $2^{n/2} \cdot \tilde{n}$;
- Time complexity: $2^{n/2}$.

(2) Obtain $2^{n/2}$ keystream prefixes of size $\tilde{n}$ each and search for a collision in the data structure created in *Step (1)*.

- Data complexity (keystream prefixes): $2^{n/2} \cdot \tilde{n}$;
- Memory complexity: negligible;
- Time complexity: $2^{n/2}$.

Note that, for the time complexity, we consider the computation of each $\tilde{n}$-bit keystream block to be an atomic operation, independent of whether it is a keystream prefix or a block that appears in the middle of a keystream. This is motivated by the comparison to the complexity of exhaustive key search, where, per key candidate, a keystream prefix needs to be generated and compared to the given keystream prefix for the secret key. In consequence, *Step (1)* and *Step (2)* each have a time complexity of $2^{n/2}$. Note that for *Step (1)*, where we slide an $\tilde{n}$-bit window over the keystream, we actually need less encryption operations when we assume that each atomic encryption operation gives $\tilde{n}$ new keystream bits. However, inserting the $2^{n/2}$ keystream blocks into the hash table takes $2^{n/2} \cdot O(1)$ time. In our assessment of the corresponding time costs w.r.t. search structure operations, we follow Hellman, who writes in his famous paper titled *A Cryptanalytic Time-Memory Trade-Off* [Hel80]: "Because [...] is sorted by ciphertext, the cryptanalyst can find C_0 and its associated key in at most $\log_2 N$ operations using a binary search. Either by neglecting logarithmic factors of through hash coding [4], this will be counted as one operation."

So the overall complexity of our generic TMD-TO distinguishing attack against CKU stream ciphers is dominated by its data and memory complexities, which are both about $2^{n/2} \cdot \tilde{n}$. Consequently, if the key size of the attacked CKU stream cipher is larger than $(n/2) + \log_2(\tilde{n})$, we obtain a distinguishing attack complexity below the complexity of exhaustive key search.[6]

[6] Englund, Hell, and Johansson similarly concluded: "If the key size $|K| > N/2$, then the new distinguishing attack will always succeed with complexity below exhaustive key search." [EHJ07] However, they left out the logarithmic summand, which we chose to include as exhaustive key search has negligible data and memory complexity, whereas in the above distinguishing attack, these complexities are actually each at a factor of $\tilde{n}$ higher than the time complexity and dominate the overall cost of the attack.

Applying the Attack to Plantlet and Fruit v1

In the following, we will discuss the consequences of the above distinguishing attack for the ciphers Plantlet (cf. Subsection 6.2.3) and Fruit v1 (cf. Subsection 6.2.2). Sprout will not be treated due to the respective distinguishing result already presented by Banik [Ban15]. The reason for including Plantlet despite the fact that the designers already acknowledged that Banik's attack against Sprout could also be transferred to Plantlet is that they point out at the same time: "It would be possible to make the design resistant against this attack by either choosing a more complicated key-selection function or by further increasing internal state size." [MAM17] We will show that the above attack is completely independent of how complicated the key-selection function is. Fruit v1, on the other hand, is discussed as the designers do not mention the possibility of distinguishing attacks at all in their paper, which might give the impression that the security level against distinguishing is supposed to equal that against key recovery via exhaustive key search. We will show that this is not the case.

The main question when applying our generic attack scheme is obviously whether the targeted cipher fulfills the necessary Assumptions 6.1 (*IV Near-Injectivity*) and 6.2 (*Initial State Randomness*). In Appendix 6.A, we show for Plantlet that, under an arbitrarily fixed key, the mapping of IVs to the corresponding volatile initial state is injective; hence, Assumption 6.1 is fulfilled. For Fruit v1, we will also suppose Assumption 6.1 to be fulfilled, despite the fact that here, IV collisions could actually occur. However, as pointed out above, the existence of a large number of IV collisions would constitute a weakness on its own. In particular, as the IV space of Fruit has only size 2^{70} whereas the key size is 80 bits, the existence of even a single IV collision would be sufficient to launch a distinguishing attack with complexity below that of exhaustive key search by exhaustively searching the complete IV space for this collision. More precisely, for each of the 2^{70} IVs, one would ask the oracle in the corresponding distinguishing game for the respective keystream prefix (of sufficient length; see discussion below) and store these keystream prefixes in an efficiently searchable data structure. If some prefix should occur twice, one answers `pseudorandom` as, w.h.p., one is in the *pseudorandom scenario* (i.e., the oracle gives its answers based on the actual output of Fruit v1). If all prefixes are distinct, one answers `random`, assuming that one is in the *random scenario* (i.e., the oracle answers each query with a random bitstring). Under the premise that Fruit v1 has IV collisions, this strategy, which, in the worst case, requires 2^{70} oracle queries, will be successful with probability close to 1 (and, hence, constitute a valid distinguisher, which has to be better than the 0.5 success probability of pure guessing), as in both scenarios (each of which occurs with probability 0.5) one will give the correct answer with probability close to 1 if the keystream prefixes are sufficiently long (see, again, the corresponding discussion about the required length of keystream prefixes below). Note that if it is unclear whether Fruit v1 actually has IV collisions, one can also combine the two approaches. That is, one first assumes that Fruit v1 has no (or only few) IV collisions and uses our new generic

distinguishing algorithm for CKU stream ciphers. If the algorithm is not successful in finding a collision of keystream blocks, one then runs the above IV collision-based distinguisher, assuming that there are either many IV collisions for Fruit v1 or that the oracle is in the *random scenario*. The worst-case attack complexity of such a combined distinguisher would obviously be dominated by the 2^{70} oracle queries of the second part. However, it would still be below the cost of exhaustive key search.

The applicability of Assumption 6.2, on the other hand, can neither be proved for Plantlet nor for Fruit v1. Instead, we have to refer to a plausibility argument based on the structure of the respective cipher (similarly to what, e.g., Banik did for Sprout [Ban15]):

- **Plantlet:** The IV space of Plantlet has size 2^{90} and the corresponding mapping (under an arbitrarily fixed key) to the set of initial states is injective (see Appendix 6.A). As two bits of the 9-bit counter will not be used after the state initialization is completed, the relevant volatile inner state of Plantlet has size $61 + 40 + 7 = 108$ bits. From the cipher definition (cf. Subsection 6.2.3), we know that the remaining 7-bit counter will have the same binary value $0 \dots 0$ for all *initial* states. During keystream generation, this counter, interpreted as a natural number, stepwise takes all values *mod 80*. In particular, in every 80th clock cycle, the counter takes binary value $0 \dots 0$. Based on general security assumptions for stream ciphers, we can also expect that the FSR states will evolve randomly during keystream generation. As a consequence, every time the counter takes value $0 \dots 0$, we have a chance of 2^{90-101} (the combined size of the FSRs is 101 bits) that there is an IV which has this inner state as an initial state. So when picking an arbitrary $(108 + \epsilon)$-bit keystream block, we have a chance of $80^{-1} \cdot 2^{90-101} > 2^{-18}$ that the underlying 108-bit inner state is also an initial state for some IV under the given key. This implies that Assumption 6.2 holds.

- **Fruit v1:** The IV space of Fruit has size 2^{70}. Due to the way the IV is introduced to the inner state, there is a possibility for IV collisions leading to the same initial state. However, as explained above, for Fruit v1 we will also suppose Assumption 6.1 to be fulfilled. So let 2^{70} be the approximate size of the set of initial states that are actually possible under the given secret key. Like Plantlet, Fruit v1 has a (7-bit wide) counter that influences the key bit selection. The volatile inner state of Fruit v1 has size $37 + 43 + 7 = 87$ bits. As in the case of Plantlet, the 7-bit counter is stepwise incremented during keystream generation. However, there are two differences: (1) the counter will actually cycle through all possible 2^7 values; (2) the counter value of the initial states will not be known as it is set based on certain (secret) FSR values as part of the state initialization (cf. Subsection 6.2.2). We know, however, that each initial state has the property that, when clocked back 80 times (the respective state transition function is bijective), a certain position of the LFSR has to be 1 and six positions of the NFSR have to equal six positions of the counter. One in 2^7 inner states of size 87 bits will fulfill this property. Using

the same security argument about FSR state randomness as in the case of Plantlet, we can hence expect that when picking an arbitrary $(87+\epsilon)$-bit keystream block, we have a chance of about $2^{-7} \cdot (2^{70-80}) = 2^{-17}$ that the underlying 87-bit inner state is also an initial state for some IV under the given key. This implies that Assumption 6.2 holds.

In order to substantiate our assumption w.r.t. the randomly evolving FSR states during keystream generation and the corresponding implications on the existence of a sufficient amount of collision candidates, we created a reduced (i.e., 'halved') version of Fruit v1, which we call *Shrunk Fruit v1* (see Appendix 6.B for a specification). Shrunk Fruit v1 has a volatile inner state of size 46 bits (19-bit NFSR, 21-bit LFSR, 6-bit counter) and is operated with 40-bit keys and 35-bit IVs. Assuming that Shrunk Fruit v1 fulfills Assumptions 6.1 and 6.2, our new generic distinguisher for CKU stream ciphers should find a collision pair w.h.p. based on $2^{46/2}$ keystream *blocks* collected in *Step (1)* and $2^{46/2}$ keystream *prefixes* requested in *Step (2)*. We created a corresponding simulation with the computer algebra system Magma [BCP97]. In each iteration of the experiment, a random key was chosen and 2^{23} 50-bit keystream blocks were generated and stored as described in *Step (1)* of the above distinguishing algorithm. Then, in line with *Step (2)*, 50-bit keystream prefixes were computed for randomly chosen IVs until the first collision with the set generated in *Step (1)* occurred. We performed 25 such iterations and the average number of trials needed in *Step (2)* was $2^{22.3}$, which is in line with our theoretical results. The simulations also showed that as few as four additional bits per keystream block (i.e., 50 instead of 46) were sufficient to avoid false positives, i.e., keystream block collisions caused by collisions in the output function instead of by colliding inner states.

The complexities (*Step (1)* + *Step (2)*) of the above distinguishing attack when applied to Plantlet and Fruit v1 are as follows:

- **Plantlet:**
  - Data complexity: $2^{108/2} + 2^{108/2} \cdot (108+20) \approx 2^{61}$;
  - Memory complexity: $2^{108/2} \cdot (108+20) = 2^{61}$;
  - Time complexity: $2^{108/2} + 2^{108/2} = 2^{55}$.
- **Fruit v1:**
  - Data complexity: $2^{87/2} + 2^{87/2} \cdot (87+20) \approx 2^{50.2}$;
  - Memory complexity: $2^{87/2} \cdot (87+20) \approx 2^{50.2}$;
  - Time complexity: $2^{87/2} + 2^{87/2} = 2^{44.5}$.

In both cases, we added a 20-bit security margin to the keystream block size in order to avoid false positives as explained previously. Note that for classical TMD-TO attacks like those of Babbage [Bab95] and Biryukov and Shamir [BS00], the length of the keystream

segments which have to be stored is usually completely ignored when describing the attack complexities (i.e., the memory complexity is given as $2^{n/2}$ instead of $2^{n/2} \cdot n$, the latter of which is actually required for storing $2^{n/2}$ keystream blocks of size n bits each). The same holds for the original description of Englund, Hell, and Johansson's attack in [EHJ07]. In our distinguishing attacks against Plantlet and Fruit v1, however, we want to be as precise (and as fair) as possible. Therefore, we also not only included the factor n itself, but further added a 'global' security margin against false positives of 20 bits for both ciphers, despite their different inner state sizes.[7] We believe that 20 bits are a sufficient upper bound in both cases for the following reason. One of the most important design goals for stream ciphers is that their mapping *inner state* $\longrightarrow$ *keystream prefix* should be as close to a random function as possible (see, e.g., Chapter 7 for a corresponding *random oracle model*) in order to thwart attacks that make use of the cipher's particular structure (i.e., non-generic attacks as compared to our generic distinguisher). For a random function $F : \{0,1\}^n \longrightarrow \{0,1\}^{n+7}$, however, the following holds: Let $x \in \{0,1\}^n$ be chosen uniformly and at random. Then the probability that there is no second preimage $y \in \{0,1\}^n$, $y \neq x$, with $F(x) = F(y)$ is

$$\left(\frac{2^{n+7}-1}{2^{n+7}}\right)^{2^n-1} \approx \left(\left(1-\frac{1}{2^{n+7}}\right)^{2^{n+7}}\right)^{2^{-7}} \approx \left(1-e^{-1}\right)^{2^{-7}} \approx 0.996$$

and, hence, with only 7 bits of security margin, already way larger than actually necessary for our attack to succeed with high probability. We would also like to point out that, even in the unlikely case that our 20-bit security margin should turn out to be slightly too optimistic, this would have a negligible influence on the corresponding attack complexities for Plantlet and Fruit v1 given above (e.g., even using $108 + 50$ instead of $108 + 20$ bits would increase the overall attack complexity for Plantlet only from 2^{61} to $2^{61.3}$).

Note that for Plantlet, complexities could also easily be improved by taking the cipher specifics into account. For example, the attacker actually knows which of the keystream blocks collected in *Step (1)* are impossible to produce a collision (because he knows the underlying counter values). These values would not have to be stored in the first place, reducing the memory complexity of the attack. Moreover, the above attack complexities are based on the assumption that all values can occur in the counter register. For Plantlet, this is actually not the case, which would further reduce the attack complexity. However, as pointed out before, we wanted to have a scheme for distinguishing attacks against CKU stream ciphers which is as generic as possible.

Thereby, we hope to clarify that a more complicated key schedule (e.g., by having a nonlinear round key function, making use of FSR bits etc.) will not protect against the

[7] As pointed out previously, due to the focus on block ciphers in OFB mode, this security margin was not necessary for the attack of Englund, Hell, and Johansson. It would, however, be required in the well-known TMD-TO attacks by Babbage [Bab95] and Biryukov and Shamir [BS00], where, like the additional factor n itself, it is usually not included in the description of the respective attack complexities.

above distinguishing attack. So, while continuously using the secret key may protect against TMD-TO *inner state recovery* attacks, it does not provide security (equal to the security against key recovery) against TMD-TO *distinguishing* attacks.

A trivial way to thwart the above distinguishing attack would be to increase the size of the volatile inner state. However, this would clearly counteract the initial idea of having a smaller state. In Section 6.5, we introduce a new design idea for providing full security against TMD-TO distinguishing attacks without increasing the volatile inner state.

6.3.2 A Key Recovery Attack against Fruit v1

The following attack against Fruit v1 (our name for the cipher specification in ePrint version 20170304:073404 of [GHX16]; see Section 6.1 for further remarks) works for the subset $\{k_0, \ldots, k_{63}, 0, \ldots, 0\} \subset \{0,1\}^{80}$ (i.e., for every 65 536th key), for which it allows key recovery faster than exhaustive key search (i.e., with complexity $< 2^{64}$) against the remaining bits $k_0, \ldots, k_{63}$. In other words: Fruit has (at least) 2^{64} weak keys, which do not offer the promised 80-bit security.

Attack Description

Our attack makes use of Fruit v1's complicated key schedule. More precisely:

Observation 6.1

The key schedule of Fruit v1 has the property that if $k_{64} = k_{65} = \ldots = k_{79} = 0$ holds, then the key schedule bit k'_t is always computed as $k'_t = k_{q+32}$, where $0 \leq q \leq 31$ and q depends on t as described in Subsection 6.2.2. In particular, this means that if $k_{64} = k_{65} = \ldots = k_{79} = 0$ holds, then the key bits $k_0, \ldots, k_{31}$ are never involved in the computation of k'_t.

So let us suppose that $k_{64} = k_{65} = \ldots = k_{79} = 0$ holds. We will use a variant of Babbage's TMD-TO attack [Bab95] to recover the inner state (composed of the 80 bits of the FSRs *and* the 32 key bits $k_{32}, \ldots, k_{63}$, i.e., 112 bits in total) at $t = 130$ *before* the counter C_r and the value of l_{130} are overwritten. Observe that if we know this inner state and the IV underlying it, then we will be able to clock the cipher back and recover the inner state at $t = 0$, which contains the full 80-bit key. Also note that the counter values at $t = 130$ *before* C_r is overwritten are publicly known and always the same.

At the beginning of the corresponding *key recovery game*, the oracle chooses a random key $k = (k_0, \ldots, k_{79})$ satisfying $k_{64} = k_{65} = \ldots = k_{79} = 0$. Thereafter, each query performed by our algorithm (asking 'What is the keystream prefix for IV x?') is correctly answered by the oracle according to the respective Fruit v1 output under k. Note that our algorithm does not follow any particular strategy w.r.t. which IVs actually serve as query inputs. The only condition is that the same IV is not used twice. Consequently, this

oracle-based key recovery game actually corresponds to a classical known-plaintext setting, where we know the plaintext and ciphertext prefixes (and, hence, also the keystream prefixes) for 2^{52} different public IVs (see below).

For the sake of brevity, in the following, we will omit the 'security margin' ϵ w.r.t. the length of keystream fragments that we included in our above distinguishing attack. As explained there, the corresponding number of additionally required bits is actually negligible. In particular, it would only affect the data and memory complexities of our key recovery attack, both of which are below the overall complexity (see below) and would still not exceed it if we included a generous security margin of, e.g., 16 bits.

Attack algorithm:

Step (1) For 2^{52} different IVs, obtain the 112-bit keystream prefix from the oracle and save the tuples (*keystream prefix*, IV) in an appropriate (i.e., efficiently searchable with respect to the keystream prefixes) data structure like a hash table.

Step (2) For 2^{60} random choices of elements

$$((k^*_{32}, \ldots, k^*_{63}), (l^*_{130}, \ldots, l^*_{172}), (n^*_{130}, \ldots, n^*_{166})) \in \{0,1\}^{112},$$

evaluate the function $F : \{0,1\}^{112} \longrightarrow \{0,1\}^{112}$, which computes the first 112 keystream bits based on this random KSG state at $t = 130$, and look for a collision in the data structure created in *Step (1)*.[8] Once a collision is found, clock the cipher back based on the discovered (IV, *inner state*) pair and obtain the full secret key as the FSR contents at $t = 0$.

[Output] If, during *Step (2)*, a collision was found (which happens with high probability according to the following Theorem 6.1), the algorithm provides the corresponding secret key. Else, it answers `SearchFailed` and stops (or, alternatively, repeats *Step (2)* until a collision is eventually found).

Theorem 6.1

According to the birthday paradox, with high probability, we will find a collision between one of the keystream prefixes from *Step (1)* with a keystream prefix from *Step (2)*.

Proof. Shifted to the end of this subsection for reasons of clarity. □

[8] Note that the 112-bit input $((k^*_{32}, \ldots, k^*_{63}), (l^*_{130}, \ldots, l^*_{172}), (n^*_{130}, \ldots, n^*_{166}))$ contains all necessary information to compute this keystream prefix, because: the counter C_r at $t = 130$ is overwritten with bits from these assumed FSR states, and we suppose the last 16 key bits to be 0, and thus the first 32 key bits are never needed again for the state update (and the keystream generation) after $t = 0$.

The complexity of the above attack is:

(1) Obtain 2^{52} keystream prefixes of size 112 bits each and store them, together with the respective IV of size 70 bits, in an efficiently searchable data structure:

- Time complexity: 2^{52};
- Data complexity (keystream prefixes): $2^{52} \cdot 112 \approx 2^{58.8}$;
- Memory complexity ((*keystream prefix*, IV) tuples):

$$2^{52} \cdot (112 + 70) \approx 2^{59.5}.$$

(2) Evaluate the function F 2^{60} times (we will consider this as 2^{60} atomic operations, just like each encryption under a different key during exhaustive key search is usually treated as an atomic operation; actually, we are even faster as we do not need to perform the first 130 clock cycles of the initialization) and look for a collision:

- Time complexity: 2^{60}.

Obviously, for the tradeoff parameters that we chose (i.e., collecting 2^{52} keystream prefixes during *Step (1)* and performing 2^{60} evaluations of F for different random inputs in *Step (2)*), the overall attack cost is dominated by the time complexity 2^{60}. (Note that we needed to get below 2^{64} to show that for the targeted subset of 2^{64} keys, the remaining unknown 64 key bits can be recovered with complexity below exhaustive key search, i.e., below 2^{64}.)

We close this subsection with some final remarks:

- For the related ciphers Grain and Plantlet, it can be shown that, under an arbitrarily fixed key, the mapping of IVs to initial states is injective. Due to the way the IV is introduced (i.e., 'from aside') in Fruit, it is not clear to what extent this property actually holds here. This leads to the situation that during *Step (1)* of the algorithm, we could need some more attempts to arrive at a subset of 2^{52} different keystream prefixes corresponding to a set S, $|S| = 2^{52}$, of different inner states as explained above. This, however, is not a problem as we could collect up to 2^{57} keystream prefixes during *Step (1)* and still stay below the overall target complexity 2^{64} of exhaustive key search against the bits $k_0, \ldots, k_{63}$ (note that only the data complexity would increase as we only save different keystream prefixes, which leaves the memory complexity unchanged). Moreover, we could additionally tweak the tradeoff parameters.

- For an older version of Fruit (i.e., ePrint version 20161124:115414 of [GHX16]), the design description actually contained an upper bound (2^{15}) on the number of IVs that should be used under the same secret key. In the newer ePrint version 20170304:073404, which underlies what we call Fruit v1 here, this restriction has

been dropped.[9] Nonetheless, we would like to point out that the above attack could be easily adapted to a scenario with this restriction still in place. Instead of targeting the inner state at $t = 130$, one could also target an arbitrary inner state during keystream generation. However, in this case the counter would not be known any longer and it (more precisely, only the six bits $c_t^1, \ldots, c_t^6$) would have to be included in our TMD-TO attack. This would raise the data, memory and time complexities each by a factor of at most 2^3 (the counter is also 'halved' via the birthday paradox), which would still fall in the boundaries of a successful attack.[10] In *Step (1)* of the attack, one would then obtain 2^{55} 118-bit keystream blocks based on 2^{12} keystreams each of size 2^{43} bits (the limit set by the designers of Fruit v1) for 2^{12} different IVs. In *Step (2)*, one would pick 2^{63} random inputs for a modified function $F : \{0,1\}^{118} \longrightarrow \{0,1\}^{118}$. The rest of the attack then works analogously to what we described before.

- An easy way to thwart our key recovery attack against Fruit v1 would obviously be to add the linear term '$\oplus\, k_s$', with s cyclically taking the values $0, \ldots, 31$, to the cipher's round key function. However, the question remains in what sense (i.e., w.r.t. which kind of attacks) a more complicated round key function like that of Fruit v1 is actually supposed to be superior to the basic one used in the yet unbroken Plantlet.

Proof of Theorem 6.1

In *Step (1)*, we collect keystream prefixes which correspond to a subset S of size 2^{52} of the set $\Omega = \{0,1\}^{32} \times \{0,1\}^{43} \times \{0,1\}^{37}$ of size 2^{112}. More precisely, the 112-bit inner states underlying the keystream prefixes collected in *Step (1)* all belong to the same secret key. This implies that S has a special structure:

$$\begin{aligned}
&\left(\left(\tilde{k}_{32}, \ldots, \tilde{k}_{63}\right), \left(\tilde{l}_{130}, \ldots, \tilde{l}_{172}\right), (\tilde{n}_{130}, \ldots, \tilde{n}_{166})\right) \in S \\
&\text{and } \left(\left(\hat{k}_{32}, \ldots, \hat{k}_{63}\right), \left(\hat{l}_{130}, \ldots, \hat{l}_{172}\right), (\hat{n}_{130}, \ldots, \hat{n}_{166})\right) \in S \\
&\Rightarrow \tilde{k}_i = \hat{k}_i \text{ for } i = 32, \ldots, 63.
\end{aligned}$$

However, we will see that this structure is completely irrelevant for the success probability of our attack. In particular, S does not have to be a random subset of Ω. The only important aspect is that during *Step (2)*, we are able to randomly select elements from

[9] Note that limiting a cipher which is designed for fixed-key scenarios (due to the continuous use of non-sequential key bits) to 2^{15} IVs per secret key, seems rather unrealistic as this would effectively mean that a corresponding device would have to be dumped after at most 32 768 keystream generations.

[10] In fact, e.g., the data complexity would be increased by a factor below 2^3 (and possibly even decrease), as now, the keystream blocks can be derived via sliding a 118-bit window over the keystream, just like in the classical TMD-TO attack.

the whole of Ω (which is the case in our attack as we are free to choose the inputs for $F : \{0,1\}^{112} \longrightarrow \{0,1\}^{112}$; see above).

So let S with $|S| = 2^{52}$ be an arbitrarily fixed subset of Ω with $|\Omega| = 2^{112}$. The probability of finding a collision during *Step (2)* can be computed as $1 - p$, where p denotes the probability that the 2^{60} elements which we draw during *Step (2)* all belong to the set $\Omega \setminus S$. For simplicity, we will assume here that our algorithm is 'stupid' in the sense that it does not remember which elements it has already drawn uniformly and at random during *Step (2)*, i.e., the same non-collision elements in $\Omega \setminus S$ can be drawn multiple times during this step. (Note that this assumption is obviously a disadvantage for the attacker.) Then we get

$$p = \left(\frac{2^{112} - 2^{52}}{2^{112}}\right)^{2^{60}} = \left(1 - \frac{1}{2^{60}}\right)^{2^{60}} \approx e^{-1},$$

which proves our Theorem 6.1 as $1 - p \approx 0.63$. □

6.4 The Future of Small-State Stream Ciphers

Discussing recent advances in the field of small-state stream ciphers, we have omitted the most fundamental question so far: is *small-state* actually a meaningful design target? From a theoretical point of view, small-state designs are certainly interesting as new concepts like the continuous use of the secret key in Sprout-like ciphers or the Even-Mansour-like [EM93] state initialization algorithm of Lizard now seem to allow to 'beat the birthday bound' and hence to explore more extreme tradeoffs between state size and, e.g., the complexity of feedback and output functions. The ultimate question here seems to be whether we can achieve n-bit security against key recovery *and* distinguishing with an n-bit volatile inner state? Our conjecture is that this is indeed possible by using a Sprout-like approach which additionally includes the IV in the state update during keystream generation (see Section 6.5).

But does a small-state approach also make sense from a practical perspective? Clearly, registers are costly in terms of area (cf. Subsection 2.3.3) and power (cf. Subsection 2.3.4), so the fewer state bits the better; but area and power are also consumed by the combinatorial logic, e.g., for the feedback and output function. This conflict is also represented in the eSTREAM portfolio, where Grain v1 (cf. Subsection 5.2.4) has an inner state size of 'only' 160 bits but rather involved feedback and output functions, whereas Trivium (cf. Subsection 5.2.3) has an inner state of 288 bits but extremely simple feedback and output functions. After the eSTREAM contest had finished (see Section 5.1 for further details), the focus in the cryptographic community shifted from lightweight stream ciphers to lightweight block ciphers like PRESENT [BKL+07] and KATAN/KTANTAN [DCDK09]. In 2009, Poschmann (one of the designers of PRESENT) concluded his PhD thesis about lightweight cryptography with: "The widespread assumption that stream-ciphers can

be implemented more efficiently in hardware compared to block ciphers does not hold anymore, since the block cipher PRESENT requires only 1,000 GE." [Pos09] A simple estimation shows that the small-state design principle is apparently the only way to close this efficiency gap (at least in terms of chip area measured in GE; cf. Subsection 2.3.3) between lightweight block and stream ciphers: The FSR storage cells of Grain v1, which achieves 80-bit security with a 160-bit inner state, consume alone (i.e., without any additional counters, combinatorial logic etc.) at least $160 \cdot 5.33 \approx 853$ GE using the cheapest type of flip-flop (i.e., D flip-flops) with the UMCL18G212T3 standard cell library that was also used by Poschmann for implementing PRESENT. Given that PRESENT (which, like Grain v1, has a key size of 80 bits) can be implemented using this cell library for only 1075 GE in total [RPLP08], it becomes clear that the only way for stream ciphers to catch up here is to reduce the size of their volatile inner state below the birthday bound. And in fact, ciphers like Plantlet and Lizard (both of which were introduced at FSE 2017) indicate that the small-state design approach has made stream ciphers 'exciting' again. After all, both of them surpass Grain v1, the most hardware-efficient member of the eSTREAM portfolio, in important metrics for lightweight ciphers like chip area (Plantlet and Lizard) and power consumption (Lizard). Moreover, Plantlet can be implemented with a chip area below 1000 GE and is hence even superior to PRESENT in this respect.[11]

But the competitive efficiency w.r.t. block ciphers is not the only novelty of small-state stream ciphers. Motivated by the attacks against Sprout (cf. Subsection 6.2.1), we developed Lizard, whose full details are presented in Chapter 8 and which offers *provable* security against generic TMD-TO key recovery attacks. Though corresponding proofs are yet missing for Sprout-like ciphers, this is certainly a very promising direction of future research. Moreover, all of the new small-state stream ciphers are still susceptible to distinguishing attacks (see Subsection 8.4.2 for Lizard and Section 6.3 for Sprout-like ciphers, respectively). For Trivium and Grain, only recently, security proofs w.r.t. generic TMD-TO distinguishing attacks have been published by Krause [Kra17]. In Section 6.5, we present a new design idea for small-state stream ciphers, for which we conjecture full security against such attacks and expect that a corresponding security proof in a *random oracle model* (see Chapter 7) will be available soon.

In Subsection 6.2.2, we have seen that fast correlation attacks, which were first introduced by Meier and Staffelbach [MS88] in 1988, remain to be a real threat to contemporary stream ciphers. In particular, the corresponding attack of Meier and Zhang against Fruit [HKMZ17a] presented at ESC 2017 makes use of the fact that the nonlinear filter function h becomes linear in the NFSR bits if the contents of the LFSR are known. As pointed out in [HKMZ17a], the lessons learned here for the design of new small-state

[11] Also do not forget that, unlike stream ciphers, block ciphers additionally need an appropriate mode of operation (if the problems of electronic codebook mode (ECB mode) are to be avoided), increasing hardware costs in terms of, e.g., area and power consumption.

Grain-like stream ciphers are that the output function needs to remain "[s]trong even when one of the registers is known" and that the feedback function of the NFSR has to be "[o]f high enough nonlinearity (to prevent good linear approximations)." Note that, also for ciphers that continuously use the secret key (such as Sprout, Fruit, and Plantlet), both of these conditions need to hold irrespective of the specifics of round key generation.

Very recently, important progress has also been made in the field of applying cube attacks (introduced by Dinur and Shamir [DS09] in 2009) against NFSR-based stream ciphers. More precisely, at Crypto 2017, two papers introduced new approaches for using cubes beyond an experimentally feasible range: In [Liu17], a framework for evaluating the algebraic degree of NFSR-based cryptosystems is suggested and cube-based distinguishing attacks against round-reduced versions of Trivium and related ciphers are presented. In [TIHM17], the so-called *division property*, which has already been exploited in the context of block ciphers [Tod15], is used to allow for the currently best cube-based key recovery attacks against round-reduced versions of Grain-128a [gHJM11], Trivium [CP05], and ACORN [Wu16]. These advances show that stream cipher designers need to take special care w.r.t. design decisions that have an impact on the algebraic degree of their cipher.[12] This is particularly important in the field of lightweight stream ciphers, where designers might be tempted to implement feedback and output functions of low degree and, in order to reduce latency and energy consumption, have only a small number of initialization rounds.

An important implication of fast correlation and cube attacks is consequently that, as strong feedback and output functions are of vital importance anyway (esp. if a large number of initialization rounds (like in Trivium, cf. Subsection 5.2.3) is to be avoided due to its unfavourable effects w.r.t. latency and energy consumption), designers of lightweight stream ciphers should rather save on state cells.

Another lesson learned from existing attacks (e.g., those for Sprout summarized in Subsection 6.2.1, or the new key recovery attack for Fruit v1 presented in Subsection 6.3.2) against Sprout-like stream ciphers, which continuously use the secret key, is that a complicated key schedule / round key function has so far only led to security problems. In fact, we conjecture that the basic round key function used by Plantlet is actually the optimal one (see also Section 6.5). More precisely, it is already sufficient to thwart TMD-TO key recovery attacks. Other classical attack techniques like algebraic attacks, correlation attacks etc. should be countered by established design measures like the choice of feedback and output functions of a high enough degree.

In line with these insights, we will now conclude this chapter by presenting a new design approach, which may not only allow to reduce the inner state size, but also to provide full security against generic TMD-TO distinguishing attacks, something which is yet missing for existing small-state stream ciphers like Sprout, Fruit, Plantlet, or Lizard.

[12] More precisely, in the context of cube attacks, the algebraic degree of the Boolean function that maps secret key and IV to the first keystream bit.

6.5 New Design Idea: Stream Ciphers which Continuously use the IV

The assumption underlying the applicability of the distinguishing attack for CKU stream ciphers like Sprout, Fruit v1, and Plantlet presented in Subsection 6.3.1 (and also, e.g., that of Banik [Ban15]), is that it is possible to find two IVs which, under an arbitrarily fixed key, lead to shifted versions of the same keystream. In this section, we suggest a potential countermeasure, which ensures that, under an arbitrarily fixed key, any two IVs will always map to different keystreams.

The prominent innovation of Sprout (cf. Subsection 6.2.1) was to continuously (i.e., also during keystream generation) use the secret key as part of the state update in order to protect against TMD-TO *inner state recovery* attacks. Our suggestion is now to also continuously use the public IV as part of the state update in order to protect against TMD-TO *distinguishing* attacks. By doing so, the public IV would become part of the inner state, just like the secret key became part of the inner state in Sprout.

In consequence, resynchronization attacks that try to discover some collision in the volatile part of the inner state based on collisions in the corresponding keystream segments (such as our distinguishing attack presented in Subsection 6.3.1) would be thwarted. This is due to the fact that through the continuous involvement of the IV in the state update, two identical volatile inner states will subsequently evolve differently under different IVs, thereby leading to different keystream segments, which deprives an attacker of the corresponding possibility to efficiently detect collisions in the volatile inner state through comparing segments of the observed keystreams.[13] Hence, for a properly designed stream cipher which continuously uses the key and the IV as part of its state update, neither the classical TMD-TO attacks (due to the continuous key involvement) nor our new distinguishing attack (due to the continuous IV involvement) will work any longer.

Now one may argue from a hardware perspective that, while the secret key has to be stored anyhow (e.g., also for Trivium, Grain etc.) in order to be reused with other IVs, this would not be the case for the IV. Hence, at first sight, assuming that the IV is still accessible after state initialization might be considered cheating. However, we do not think that this is the case for many application scenarios. Look, for example, at A5/1 (cf. Subsection 5.2.2). There, the IV used in the encryption of a data packet is the respective (sequentially incremented) 22-bit frame number. Hence, any A5/1 device needs some memory containing this frame number anyhow. In general, especially for ciphers with small IV spaces, there always has to be a mechanism like a stepwise incremented IV register to make sure that the same IV is not accidentally used twice under the same secret key. Similarly, in all communication scenarios like A5/1, where the packet counter serves at the same time as an IV source, we will always have this information.

[13] In other words, under an arbitrarily fixed key, two IVs will never lead to shifted versions of the same keystream.

Actually, any stream cipher definition which strictly requires 'Never use the same IV twice under a single key!' also needs a mechanism to enforce this requirement (though stream cipher designers seem to hardly talk about this issue in their suggestions and instead leave the problem of IV uniqueness to user). Independent of whether this mechanism is to keep a stepwise incremented IV in a writable storage location like an EEPROM or another register on the device, or whether the device actually keeps a table of already used IVs, there will always be a source where a cipher can continuously get the current IV from.

In this thesis, we will not suggest a specific instantiation of such a cipher which continuously uses the secret key *and* the IV, but leave the development to future work. However, it might be as easy as changing the round key function of Plantlet (cf. Subsection 6.2.3) from $\tilde{k}^t = k_{(t \bmod 80)}$, $t \geq 0$, to

$$\tilde{k}^t = k_{(t \bmod 80)} \oplus IV_{(t \bmod 90)},\ t \geq 0.$$

As a final note, we would like to point out that while our new design idea of continuously using the IV as part of the state update was originally targeted at CKU stream ciphers (see Definition 6.1 in Subsection 6.3.1), we conjecture its applicability also for other contexts. In particular, as we will explain in Section 9.3 of the chapter *Future Research Directions*, combining continuous IV use and *packet mode*[14] can even thwart classical TMD-TO attacks like those of Babbage [Bab95] and Biryukov and Shamir [BS00], without requiring any additional measures such as using the secret key more than once.

6.6 Conclusion and Outlook

Our results show that the search for small-state stream ciphers which completely 'defeat' the birthday bound, is far from being finished. The initial hope that continuously using the secret key would fully solve this problem has been shattered by TMD-TO distinguishing attacks. While our new design principle of continuously using key *and* IV might actually lead to ciphers that resist TMD-TO key recovery *and* distinguishing attacks, it is obvious that the necessary hardware conditions will not be present in all application scenarios. Hence, the search for alternative solutions remains a field which is not only interesting from a theoretical point of view, but also of actual practical relevance. The results presented in this chapter seem to indicate that a more complicated key schedule (as used by Fruit) will probably not be the way to go. Another interesting direction for future research might be the search for a non-Grain-like small-state stream cipher.

In the following Chapter 7, we will present the Lizard-*construction*, which underlies the provable $(2n/3)$-security against generic TMD-TO-based key recovery of our new

[14] See Section 5.1 for a formal introduction of this term, and Chapter 8 for a practical instantiation in the form of our new lightweight stream cipher Lizard.

lightweight stream cipher Lizard. While, for the sake of providing a complete overview w.r.t. the current state of the art in small-state stream cipher design, we have already briefly outlined the structure of Lizard in Subsection 6.2.4, Chapter 8 will then contain the full details, including an in-depth security analysis and an assessment of the cipher's suitability for ultra-constrained RFIDs.

Appendix 6.A Plantlet: Injectivity of IV → Initial State

The following algorithm computes (under an arbitrarily fixed key) for any given FSR content at $t = 320$ a corresponding FSR content at $t = 0$, which, according to the Plantlet algorithm (cf. Subsection 6.2.3), would lead to the given FSR content at $t = 320$. (Moreover, it returns the IV contained in the FSRs at $t = 0$.) This shows that the corresponding mapping of FSR contents at $t = 0$ to FSR contents at $t = 320$ under an arbitrarily fixed key is surjective. As domain and codomain have the same size, this also implies injectivity and shows that, under an arbitrarily fixed key, different IVs will always be mapped to different initial states by Plantlet's state initialization algorithm.

Given:

- 80-bit Key: $(k_0, \ldots, k_{79})$
- 60-bit LFSR part of the initial state: $(l_0^{320}, \ldots, l_{59}^{320})$
  Note that l_{60} is irrelevant for us as it is not used during initialization.
- 40-bit NFSR part of the initial state: $(n_0^{320}, \ldots, n_{39}^{320})$
- $(c_t^0, \ldots, c_t^8)$ is known for all t, $t \geq 0$.
  This is particularly important for c_4^t (see below).

Algorithm:

for $t = 319$ down to 0 **do**
  for $i = 0$ to 58 **do**
    $l_{i+1}^t \leftarrow l_i^{t+1}$
  end for
  for $i = 0$ to 38 **do**
    $n_{i+1}^t \leftarrow n_i^{t+1}$
  end for
  $(x_0^t, \ldots, x_8^t) \leftarrow (n_4^t, l_6^t, l_8^t, l_{10}^t, l_{32}^t, l_{17}^t, l_{19}^t, l_{23}^t, n_{38}^t)$
  $h^t \leftarrow x_0^t x_1^t \oplus x_2^t x_3^t \oplus x_4^t x_5^t \oplus x_6^t x_7^t \oplus x_0^t x_4^t x_8^t$
  $z^t \leftarrow h^t \oplus l_{30}^t \oplus (n_1^t \oplus n_6^t \oplus n_{15}^t \oplus n_{17}^t \oplus n_{23}^t \oplus n_{28}^t \oplus n_{34}^t)$
  $l_0^t \leftarrow l_{59}^{t+1} \oplus l_{54}^t \oplus l_{43}^t \oplus l_{34}^t \oplus l_{20}^t \oplus l_{14}^t \oplus z^t$
  $\tilde{k}^t \leftarrow k_{(t \bmod 80)}$
  $n_0^t \leftarrow n_{39}^{t+1} \oplus \tilde{k}^t \oplus l_0^t \oplus c_4^t \oplus n_{13}^t \oplus n_{19}^t \oplus n_{35}^t \oplus n_{39}^t \oplus n_2^t n_{25}^t \oplus n_3^t n_5^t \oplus n_7^t n_8^t \oplus n_{14}^t n_{21}^t \oplus n_{16}^t n_{18}^t \oplus n_{22}^t n_{24}^t \oplus n_{26}^t n_{32}^t \oplus n_{10}^t n_{11}^t n_{12}^t \oplus n_{27}^t n_{30}^t n_{31}^t \oplus n_{33}^t n_{36}^t n_{37}^t n_{38}^t \oplus z^t$
end for
for $i = 0$ to 39 **do**
  $IV_i \leftarrow n_i^0$
end for
for $i = 40$ to 89 **do**

$IV_i \leftarrow l^0_{i-40}$
end for
return $(IV_0, \ldots, IV_{89})$

Appendix 6.B Shrunk Fruit v1

In the following, we present an overview over *Shrunk Fruit v1*, our 'halved' variant of Fruit v1 (cf. Subsection 6.2.2) that we used for the experiments described in Subsection 6.3.1. As compared to Fruit v1, which has a key size of 80 bits and an IV size of 70 bits, Shrunk Fruit v1 uses 40-bit keys and 35-bit IVs. The sizes of the NFSR and the LFSR were also shrunk from 37 to 19 bits and from 43 to 21 bits, respectively. For the specifications of the new FSRs as well as for the key schedule and the output function, we did our very best to retain the properties of the original cipher and, in particular, not to introduce new weaknesses. To that end, we actually even kept the number and degree of terms in the key schedule, output and feedback functions of the original Fruit v1. Instead, we just squeezed the corresponding tap indices to fit the new FSRs. In consequence, Shrunk Fruit v1 is probably even stronger (against non-generic attacks) than one would expect from a truly halved variant. Other important properties such as the use of a maximum-length LFSR were also transferred from Fruit v1 to Shrunk Fruit v1. The size of the counter C_r was only reduced by one bit, because in Fruit v1, seven bits are required to index 80 key bits and, consequently, in Shrunk Fruit v1, six bits are now required to index the 40 key bits. Please find below a full specification of Shrunk Fruit v1 in bullet point form:

- **Input:** 40-bit key $k = (k_0, \ldots, k_{39})$, 35-bit IV $IV = (IV_0, \ldots, IV_{34})$
- **Keystream Limit per IV:** 2^{21} bits (due to the 21-bit maximum-length LFSR; corresponds to the limit of 2^{43} bits and the 43-bit maximum-length LFSR in Fruit v1)
- **6-bit Counter:**
$$C_r = \left(c_t^1, c_t^2, c_t^3, c_t^4, c_t^5, c_t^6\right)$$
- **Key Schedule:**
$$\begin{aligned} k'_t &= k_s k_{y+32} \oplus k_p k_{u+36} \oplus k_{q+16} \oplus k_{r+32} \\ s &= \left(c_t^1, c_t^2, c_t^3, c_t^4, c_t^5\right) \\ y &= \left(c_t^4, c_t^5\right) \\ u &= \left(c_t^5, c_t^6\right) \\ p &= \left(c_t^1, c_t^2, c_t^3, c_t^4\right) \end{aligned}$$

$$q = \left(c_t^2, c_t^3, c_t^4, c_t^5\right)$$
$$r = \left(c_t^4, c_t^5, c_t^6\right)$$

- **19-bit NFSR:**

$$n_{t+19} = k'_t \oplus l_t \oplus c_t^4 \oplus n_t \oplus n_{t+5} \oplus n_{t+10} \oplus n_{t+6}n_{t+2} \oplus n_{t+8}n_{t+13} \\ \oplus n_{t+3}n_{t+11}n_{t+15} \oplus n_{t+4}n_{t+9} \oplus n_{t+14}n_{t+15}n_{t+16}n_{t+17}$$

- **21-bit LFSR (a maximum-length LFSR like in Fruit v1):**

$$l_{t+21} = l_t \oplus l_{t+4} \oplus l_{t+9} \oplus l_{t+12} \oplus l_{t+14} \oplus l_{t+17}$$

- **Keybit z_t:**

$$h_t = l_{t+3}l_{t+7} \oplus l_{t+1}l_{t+11} \oplus n_{t+18}l_{t+13} \oplus l_{t+5}l_{t+16} \oplus n_{t+1}n_{t+17}l_{t+20}$$
$$z_t = h_t \oplus n_t \oplus n_{t+3} \oplus n_{t+7} \oplus n_{t+9} \oplus n_{t+12} \oplus n_{t+14} \oplus n_{t+18} \oplus l_{t+19}$$

- **IV' (extension of the 35-bit IV to 65 bits; corresponds to the extension of the 70-bit IV to 130 bits in Fruit v1):**

$$IV' = (IV'_0, \ldots, IV'_{64}) = (\underbrace{1, 0, 0, 0, 0}_{5}, \underbrace{IV_0, \ldots, IV_{34}}_{35}, \underbrace{0, \ldots, 0}_{25})$$

- **Key Loading:**

$$(n_0, \ldots, n_{18}) = (k_0, \ldots, k_{18})$$
$$(l_0, \ldots, l_{20}) = (k_{19}, \ldots, k_{39})$$

- **Key Schedule Counter Initialization:**

$$\left(c_0^1, c_0^2, c_0^3, c_0^4, c_0^5, c_0^6\right) = (0, \ldots, 0)$$

- **Initialization Procedure:**
  1. 65 IV loading and mixing steps as described for Fruit v1 in Subsection 6.2.2 (there: 130 steps).
  2. Set
  $$\left(c_{65}^1, c_{65}^2, c_{65}^3, c_{65}^4, c_{65}^5, c_{65}^6\right) := (n_{65}, n_{66}, n_{67}, n_{68}, n_{69}, l_{65})$$
  and then $l_{65} := 1$.
  3. Clock 40 times in keystream generation mode (i.e., without feeding z_t and IV' any longer to the FSRs), but without outputting z_t, as described for Fruit v1 (there: 80 times).

- **Output:** The first keystream bit that is used for plaintext encryption is z_{105}.

Es ist noch nicht genug, eine Sache zu beweisen, man muß die Menschen zu ihr auch noch verführen oder zu ihr erheben.

Friedrich Wilhelm Nietzsche

CHAPTER 7

The LIZARD-Construction

ABSTRACT

In this chapter, we propose and analyze the LIZARD-construction, a new way to build KSG-based stream ciphers (cf. Chapter 5). For an inner state size of n bits, we prove a tight $(2n/3)$-bound on its security against TMD-TO *key recovery* attacks, while the security against TMD-TO *distinguishing* attacks remains at the birthday-bound level $n/2$. The lower bound of the $(2n/3)$-result refers to a random oracle model which allows to derive formal security statements w.r.t. generic TMD-TO attacks. While similar frameworks have already been widely used for analyzing the security of block cipher, MAC, and hash function constructions, to the best of our knowledge this is the first time that such a model is considered in a stream cipher context.

The security analysis presented in this chapter is also of immediate practical relevance as, with the stream cipher LIZARD (see Chapter 8), a first instantiation of our new design principle, which we hence named LIZARD-construction, was introduced at FSE 2017. Though aiming for 80-bit security against key recovery, LIZARD has an inner state size of only 121 bits and surpasses Grain v1, the most hardware-efficient member of the eSTREAM portfolio, in important metrics for lightweight ciphers (cf. Chapter 2) such as chip area and power consumption.

Declaration of Origin: This chapter is based on the paper *On Stream Ciphers with Provable Beyond-the-Birthday-Bound Security against Time-Memory-Data Tradeoff Attacks* [HK18], written together with Matthias Krause and published in *Cryptography and Communications* (Springer US journal).

7.1 Introduction

As pointed out in Chapter 5, most classical stream ciphers are vulnerable against generic time-memory-data tradeoff (TMD-TO) attacks, which reduce their effective key length to the birthday bound $n/2$, where n denotes the inner state size of the underlying keystream generator (KSG). In consequence, e.g., the eSTREAM portfolio members Trivium and Grain v1 (both targeting 80-bit security) have comparatively large inner states of $n = 288$ bits and $n = 160$ bits, respectively. In this chapter, we propose and analyze the LIZARD-construction, a new way to build KSG-based stream ciphers. Given an inner state size of n bits, we prove a tight $(2n/3)$-bound on its security against TMD-TO *key recovery* attacks, while the security against TMD-TO *distinguishing* attacks remains at the birthday-bound level $n/2$.

For reasons of clarity, this introduction is divided into three parts. In the first part (Subsection 7.1.1), we will recall our model for KSG-based stream ciphers from Section 5.1. Those readers, who still have all details of Chapter 5 present, can safely skip this subsection up to the new Definition 7.1 at its very end. As we will strongly build on this model in this chapter (and also on our findings with w.r.t. the state initialization algorithms of classical stream ciphers as studied in Section 5.3), we still deemed it useful to repeat the corresponding details here. In the second part of this introduction (Subsection 7.1.2), we will briefly explain the role of TMD-TO attacks in general, and w.r.t. our framework in particular. Based on this, in the third part (Subsection 7.1.3), we will then outline the contribution of this chapter.

7.1.1 A Model for KSG-based Stream Ciphers

In our framework, we suppose that the communication between legal users is organized in *sessions*, where in the first phase of each session, the *secret session* key k is generated by executing a *key establishment protocol* (see, e.g., Subsection 5.2.1 for that of E_0 in Bluetooth). As pointed out in Section 5.1, this *session key generation phase* will not be treated in detail in this thesis.

Following [BG07], each stream cipher is associated with a well-defined set of inner states and its keystream generation process can be divided into the following two phases:

(A) The *key and IV setup phase*, where an initial state is derived from the secret session key k and an initialization vector IV.

(B) The *keystream generation phase*, in which the keystream is generated based on the initial state derived in phase (A).

In this thesis, we focus on what we call *KSG-based stream ciphers*, for which the main algorithmic component for performing phases (A) and (B) is a *keystream generator (KSG)*.

KSGs are clock-controlled devices which can be formally specified by finite automata, defined by an inner state length n, the set of inner states $\{0,1\}^n$, a state update function $\pi : \{0,1\}^n \longrightarrow \{0,1\}^n$, and an output function $outbit : \{0,1\}^n \longrightarrow \{0,1\}$. Starting from an initial state $q_{\text{init}} = q_0$, in each clock cycle $t \geq 0$, the KSG produces an output bit $z_t = outbit(q_t)$ and changes the inner state according to $q_{t+1} = \pi(q_t)$. The keystream $S(q_{\text{init}})$ corresponding to the initial state q_{init} is defined by concatenating all the outputs bits $z_0 z_1 z_2 \cdots$.

As pointed out above (and already in Chapter 5), the key and IV setup phase (A) of a KSG-based stream cipher is performed by a KSG-based *state initialization algorithm*, which computes, from the session key k and the initialization vector IV, the initial state q_{init}. It typically contains the following two subphases:

(A.1) The *loading phase* defines how the session key k and the initialization vector IV are loaded into the inner state registers and results in a *loading state* $q_{\text{load}} = q_{\text{load}}(k, IV)$.

(A.2) The *mixing phase* runs an appropriate KSG-based *mixing algorithm*

$$MIX : \{0,1\}^n \longrightarrow \{0,1\}^n$$

on q_{load} and results in a state $q_{\text{mixed}} = MIX(q_{\text{load}})$.

The aim of the mixing phase (A.2) is to generate a sufficient amount of diffusion, confusion, high algebraic degree etc. in the dependencies of the initial state bits from the session key bits and the IV bits. In many cases, an essential part of the mixing algorithm consists in running the KSG a certain number of times without producing keystream bits. Moreover, as we have shown in Section 5.3, for many ciphers (such as Trivium, Grain v1, A5/1, E_0) it holds that $q_{\text{init}} = q_{\text{mixed}}$. Furthermore, we have also explained that for Trivium, Grain v1, and A5/1, the mixing algorithm *MIX* can be inverted efficiently.

Note that also Sprout-like ciphers as discussed in Chapter 6 could be plugged into this framework by treating the secret key as part of the inner (and, hence, also *initial*) state. However, in the context of this chapter, we will confine our notion to how the inner state is used in classical stream ciphers (cf. Chapter 5). In particular, we suppose that *MIX* behaves like a random function with respect to standard security properties (see Section 7.2 for further details).

One can distinguish the following two *operation modes of stream ciphers*. In the *one-stream mode*, the key and IV setup phase (A) is performed only once at the beginning of the session and produces an initial state $q_{\text{init}} = q_{\text{init}}(k, IV)$. The corresponding keystream $S = S(q_{\text{init}})$ is used for the whole session. As pointed out in Chapter 5, due to their extremely large limits (e.g., 2^{64} bits for Trivium) on the amount of keystream generated under a single key/IV pair, Trivium and Grain v1 can be considered to work in one-stream mode.[1]

[1] Clearly, Trivium and Grain v1 could also be used in packet mode, but in contrast to, e.g., LIZARD (see Chapter 8), their design is not specifically optimized for such scenarios.

In contrast to this, in the *packet mode*, the communication and encryption process during a session is divided into packet steps $i = 1, 2, \ldots$, where in each packet step, a piece of message of a certain maximal packet length R is encrypted and sent. Corresponding to this, the keystream of a session is the concatenation of the keystream packets $S^1, S^2, \ldots$, where for all $i \geq 1$, S^i denotes the keystream packet generated in packet step i. Corresponding stream cipher instantiations are equipped with a mechanism providing, for each packet step i, an initialization vector IV^i (such as the frame counter in A5/1; cf. Subsection 5.2.2). Each packet step i starts with performing the key and IV setup phase (A), yielding a *packet initial state* $q^i_{\text{init}} = q^i_{\text{init}}(k, IV^i)$, followed by the generation of the keystream packet S^i, which is defined to be the prefix of length R of $S(q^i_{\text{init}})$. As we have seen in Chapter 5, typical examples for stream ciphers operating in packet mode are the Bluetooth cipher E_0 (cf. Subsection 5.2.1) and the GSM cipher A5/1 (cf. Subsection 5.2.2). We also pointed out that in the network protocols of many important digital communication scenarios, data streams are transmitted packet-wise (Ethernet, WLAN, Bluetooth, cellular networks etc.). It thus seems natural to consider stream ciphers running packet mode and, in particular, to look for corresponding design optimizations (see Section 8.1 for more examples of stream ciphers used packet mode and for further information about their practical relevance).

As in [BS00], [BG07], and many other papers, we consider the keystream generation phase (B) of KSG-based stream ciphers as being defined by the output block function $OUTBLOCK : \{0,1\}^n \longrightarrow \{0,1\}^n$, which assigns to each inner state $q \in \{0,1\}^n$ the first n keystream bits produced on q. Here, the exact definition of $OUTBLOCK$:

Definition 7.1: *OUTBLOCK*

For all $q \in \{0,1\}^n$, let

$$OUTBLOCK(q) = (\tilde{z}_0, \ldots, \tilde{z}_{n-1}),$$

where for all r, $0 \leq r \leq n-1$,

$$\tilde{z}_r = outbit(\pi^r(q)),$$

where $\pi : \{0,1\}^n \longrightarrow \{0,1\}^n$ and $outbit : \{0,1\}^n \longrightarrow \{0,1\}$ denote the state transition function and the output function of the corresponding KSG, respectively.

Note that the keystream $S(q_{\text{init}}(k, IV)) = (z_0, z_1, \ldots)$ generated on a key/IV pair (k, IV) can now be expressed as follows (see also Fig. 7.1). For each n-bit subblock $(z_r, \ldots, z_{r+n-1})$, it holds

$$(z_r, \ldots, z_{r+n-1}) = OUTBLOCK(\pi^r(q_{\text{init}})).$$

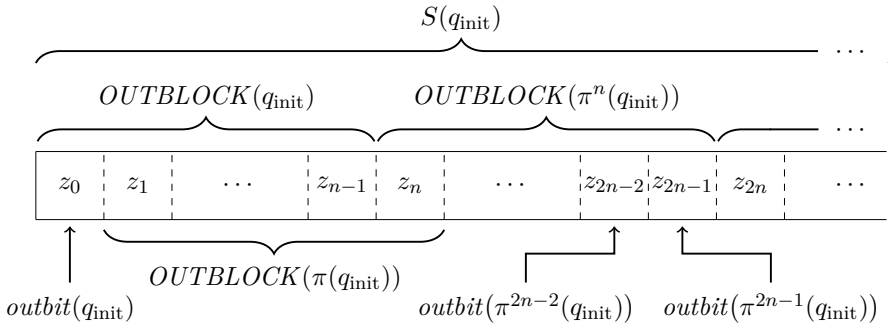

Figure 7.1: The keystream generation phase (B) in terms of our model.

7.1.2 The Role of TMD-TO Attacks

During the last decades, many stream ciphers have been suggested and many different techniques for cryptanalyzing them have been developed (correlation attacks, fast correlation attacks, guess-and-determine attacks, BDD attacks, cube attacks; see, e.g., Sections 6.2 and 8.4). Attacks on stream ciphers typically suppose that the attacker knows a piece S' of keystream which was generated under a secret session key k and a set of known or actively chosen initialization vectors. Standard goals are to distinguish S' from a truly random bitstream, to recover the inner states responsible for S', to predict a new keystream packet on the basis of S', or to recover the secret session key.

Having already successfully applied them against Sprout-like ciphers in Section 6.3, in this chapter, we once more focus on so-called *time-memory-data tradeoff (TMD-TO) attacks*, which are for many stream ciphers the most powerful known attacks. TMD-TO attacks have a generic nature in the sense that they access the security-relevant components *MIX* and *OUTBLOCK* only in a black-box manner. This implies that from the attacker's point of view, these components are *ideally designed* in the spirit of [GT15]. Hence, the target of TMD-TO attacks is the way how the components *MIX* and *OUTBLOCK* interact in computing the keystream from the secret session key k and an initialization vector IV.

In consequence, such TMD-TO attacks can usually be formulated for variable inner state length n. Correspondingly, we express upper and lower security bounds for stream cipher constructions against TMD-TO attacks in an asymptotic manner. For instance, we say that a stream cipher construction has, for some number a, $0 \leq a \leq 1$, the security level $a \cdot n$ w.r.t. TMD-TO attacks if there is a TMD-TO attack of cost behavior $O(2^{a \cdot n})$ with significant success probability and, for all $\alpha < a$, all TMD-TO attacks of cost behavior $O(2^{\alpha \cdot n})$ have only negligibly small success probability. We will discuss the cost behavior of TMD-TO attacks in more detail at the beginning of Section 7.3.

The vulnerability against generic TMD-TO attacks such as those of Babbage [Bab95] or Biryukov and Shamir [BS00] represents an inherent weakness of KSG-based stream

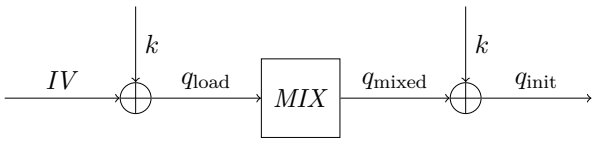

Figure 7.2: Key and IV setup phase (A) of the LIZARD-construction. The XOR symbol denotes the addition of the corresponding n-bit vectors over GF(2).

ciphers.[2] This vulnerability implies that for KSG-based stream ciphers working in one-stream mode, the effective key length is bounded by $n/2$, where n denotes the inner state length of the underlying KSG. As a consequence, classical stream ciphers have comparatively large inner state lengths; e.g., 288 bits and 160 bits for the eSTREAM portfolio members Trivium (see Subsection 5.2.3) and Grain v1 (see Subsection 5.2.4), respectively.

7.1.3 Our Contribution

In this chapter, we propose a construction principle for designing KSG-based stream ciphers with a provable beyond-the-birthday-bound security of $2n/3$ against generic TMD-TO key recovery attacks: taking a stream cipher with a state initialization algorithm of the new type

$$q_{\text{init}} = MIX(q_{\text{load}}) \oplus k, \tag{7.1}$$

where $q_{\text{load}} = k \oplus IV$ (see Fig. 7.2), and using it in packet mode. We give this construction principle the name LIZARD-construction, as it underlies the stream cipher LIZARD (see Chapter 8) introduced at FSE 2017.

The LIZARD-construction can be motivated as follows. Babbage's TMD-TO attack [Bab95] implies that if a KSG-based stream cipher runs in one-stream mode, then it is possible to predict the keystream of the whole session with a TMD-TO attack of cost behavior $O(2^{n/2})$ (see Theorem 7.1 in Section 7.3). Moreover, if the state initialization algorithm is efficiently invertible (as it is the case, e.g., for Trivium, Grain v1, and A5/1; cf. Section 5.3), then this attack even yields the secret session key.

Consequently, for obtaining KSG-based stream ciphers with beyond-the-birthday-bound resistance against TMD-TO attacks, one has to consider stream ciphers running in packet mode. But this alone is not enough. More precisely, many stream ciphers (such as Trivium, Grain v1, A5/1, and E_0; see, again, Section 5.3) employ a state initialization algorithm of type

$$q_{\text{init}} = MIX(q_{\text{load}}) \tag{7.2}$$

[2] Let us once more emphasize that, in this chapter, when speaking of *KSG-based stream ciphers*, we are referring to classical constructions; i.e., continuous-key-use stream ciphers (cf. Definition 6.1) such as Sprout-like designs are *not* treated here.

with $q_{\text{load}} = q_{\text{load}}(k, IV)$, instead of

$$q_{\text{init}} = MIX(q_{\text{load}}) \oplus k$$

with $q_{\text{load}} = q_{\text{load}}(k, IV) = k \oplus IV$, as used by the LIZARD-construction (cf. Eq. (7.1)). We show in Theorem 7.2 that even if a stream cipher runs in packet mode and even if the state initialization algorithm is not efficiently invertible (as, e.g., that of E_0), a state initialization algorithm of the type in Eq. (7.2) provides only a security level of $n/2$ w.r.t. session key recovery attacks.

In contrast to this, we are able to prove a tight $(2n/3)$-bound on the security of the LIZARD-construction against TMD-TO attacks. More precisely, in Theorem 7.3 we first describe a TMD-TO session key recovery attack of TMD cost $\tilde{O}(2^{2n/3})$ against the LIZARD-construction, which is based on the Slidex attack of Dunkelman, Keller, and Shamir [DKS12] against the one-key Even-Mansour cipher [EM93].

The main contribution of this chapter is to then show that for the LIZARD-construction, this security bound of $2n/3$ is sharp. The proof of the matching security lower bound result is done, in the spirit of [GT15], in a *random oracle model* corresponding to the components *MIX* and *OUTBLOCK* of the LIZARD-construction. We prove an information-theoretic lower bound on the security of the LIZARD-construction against generic chosen-IV attackers, who have black-box access to the component primitives *MIX* and *OUTBLOCK*, and to the stream cipher construction itself. Due to their generic nature, all known TMD-TO attacks against stream ciphers can be formulated as attacks in this model in a straightforward way.

The proof of our security lower bound follows the typical structure of similar information-theoretic proofs in the context of iterated Even-Mansour ciphers (see, e.g., [EM93], [BKL+12], [ABD+13], [CS14], [CLL+14], [HT16]). In particular, it is inspired by the (much shorter) security lower bound proof in [EM93]. As in [EM93], the lower bound against key recovery attacks follows from a lower bound against the weaker type of attack in which the goal of the attacker is to predict a new keystream packet, and which we call *packet prediction attack* in the following. The rough idea consists in proving that if an attacker Eve poses significantly less than at most $2^{2n/3}$ component and construction queries, then, with high probability, the entropy of the secret session key will still be at least $n - 1$. This immediately implies an exponentially small success probability for recovering the session key and we will show that this is also the case for predicting a correct new packet.

Note that the security bound of $2n/3$ for the LIZARD-construction cannot hold against distinguishing attacks. We show in Corollary 7.1 that there is a TMD-TO distinguishing attack of TMD cost $\tilde{O}(2^{n/2})$ against any KSG-based stream cipher working in packet mode if the packet length exceeds the inner state length n.

To the best of our knowledge, this is the first time that a formal random oracle model for the security of stream ciphers against generic TMD-TO attacks is considered. So

far, similar models were used, e.g., for analyzing the security of operation modes of key-alternating block cipher constructions (see the framework of iterated Even-Mansour ciphers), or of cryptographic hash functions, or of MAC algorithms, but not for stream ciphers. Note that in [BG07], another way of formally analyzing the security of stream cipher constructions was proposed, namely in the complexity-theoretic framework of pseudorandom number generators and pseudorandom function generators.

In 2015, Armknecht and Mikhalev suggested with Sprout (see Subsection 6.2.1) another construction method for obtaining stream ciphers with beyond-the-birthday-bound security against TMD-TO attacks. In Sprout, the symmetric secret key is not only accessed during the state initialization but also continuously used as part of the state update during the subsequent keystream generation phase. The hope here was to obtain stream ciphers with the maximum possible resistance against TMD-TO attacks.

Although Sprout was broken soon after publication via non-generic attacks (see our corresponding summary in Subsection 6.2.1), it has raised interest in the design principle and, as outlined in Chapter 6, a number of related ciphers have been suggested since, including Fruit [GHX16] and Plantlet [MAM17]. In Subsection 6.3.1, however, we have shown that this whole class of Sprout-like ciphers is susceptible to generic TMD-TO distinguishing attacks and, hence, does not meet the original expectation of providing *full* TMD-TO security. This emphasizes the importance of *provable* resistance against TMD-TO attacks as a design criterion for new stream cipher constructions. In particular, for Sprout-like ciphers, also generic TMD-TO *key recovery* attacks are still a real possibility as corresponding security proofs are missing yet.

As already mentioned, the Lizard-construction, offering provable beyond-the-birthday-bound security against TMD-TO key recovery attacks, has inspired the design of our recently published lightweight stream cipher Lizard [HKM17b]. Lizard works in packet mode with a packet length of $R \leq 2^{18}$ bits, has a state initialization algorithm of type Eq. (7.1), and an inner state length of 121 bits. The design features of Lizard, presented in Chapter 8, show that the Lizard-construction allows for practical instantiations which are competitive w.r.t. important hardware metrics for lightweight devices such as chip area (cf. Subsection 2.3.3) and power consumption (cf. Subsection 2.3.4).

Recently, interesting cryptanalytic results for Lizard have been published in [BICG17], [MSS+17], and [SSMC17]. Please note, however, that none of these papers violates the security claims made in the Lizard specification and, hence, breaks the cipher. In particular, the analysis provided in [BICG17], [MSS+17], and [SSMC17] does not indicate any weakness of the general Lizard-construction design principle. To avoid potential misconceptions, it is especially important to realize that the algorithms in [BICG17] for computing key/IV pairs which produce identical initial states (and, hence, identical keystreams) do not lead to actual attacks. This is due to the fact that in these algorithms, the attacker chooses the keys himself. This way, he is able to invert Phase 3 (the second key addition) of Lizard's state initialization and generate *some* key/IV pair that leads to a given initial state. However, the algorithms in [BICG17] do not provide

any indication on how to efficiently find the *actual* secret key if the attacker is only given an initial state together with the IV that was used to generate it (under this secret key). In Subsection 8.4.11, we will provide further details w.r.t. the results of [BICG17], [MSS+17], and [SSMC17] in the context of our in-depth security analysis for LIZARD.

Structure of this chapter: In Section 7.2, we discuss some security-relevant properties of the stream cipher components *MIX* and *OUTBLOCK*. In Section 7.3, we then start our analysis by describing three generic TMD-TO attacks against KSG-based stream ciphers, including one against the LIZARD-construction. In Section 7.4, we introduce our random oracle model for stream ciphers. Section 7.5 then contains the corresponding lower bound results. In Section 7.6, we conclude the chapter by summarizing our findings, showing some directions of further research, and providing an outlook on the remaining parts of this thesis.

7.2 More on Stream Ciphers

In this section, we discuss some concrete security-relevant issues of the components $MIX, OUTBLOCK : \{0,1\}^n \longrightarrow \{0,1\}^n$ of KSG-based stream ciphers. Let us fix a KSG of inner state length n, and let π and *outbit* define its state transition and output bit function, respectively. Observe first that the corresponding function *OUTBLOCK* is π-iterative in the following sense:

Definition 7.2: π-iterative Function

A function $F : \{0,1\}^n \longrightarrow \{0,1\}^n$ is called *π-iterative* if for all inputs $y \in \{0,1\}^n$ it holds that the suffix of length $n-1$ of $F(y)$ equals the prefix of length $n-1$ of $F(\pi(y))$.

Observe next that *OUTBLOCK* should be *preimage resistant* in the sense that it is infeasible to compute, for given $z \in \{0,1\}^n$, a value $y \in \{0,1\}^n$ fulfilling $OUTBLOCK(y) = z$. Otherwise, it would be feasible to predict, on the basis of the first n keystream bits of a packet, all remaining keystream bits of this packet.

Concerning *MIX*, observe that standard efficiency and security assumptions on KSGs imply that *MIX* should be an efficiently computable function which behaves like a random function with respect to important properties such as correlation immunity, algebraic degree, and resistance against conditional differential cryptanalysis. Moreover, as shown in Section 5.3, for many stream ciphers *MIX* is bijective and can be inverted efficiently.

These properties will be reflected in our security analysis in the way that *OUTBLOCK* is assumed to be a randomly chosen π-iterative function and that *MIX* is assumed to be a randomly chosen permutation which can be inverted efficiently. Note, however, that for many practical stream ciphers, the component *OUTBLOCK* may deviate from

behaving like a random function in the following respect. For efficiency reasons, the state transition function π and the output bit function *outbit* are often of rather low degree. According to Definition 7.1, this implies that a small number of output bits of $OUTBLOCK$ (more specific, the leftmost ones) can have a lower degree than expected for a random function. In consequence, the stream cipher can have a lowered *sampling resistance* in the sense of [BS00] and [BSW01]. As described there, this could be used for reducing the cost of certain TMD-TO attacks in a non-generic manner. For example, a corresponding approach (now known as *BSW-sampling*) allowed for a very efficient attack [BSW01] against A5/1 (cf. Subsection 5.2.2). As, for the asymptotic behavior of TMD-TO attacks, the cost reductions of techniques like BSW-sampling are rather negligible, we do not consider this effect in our security model. However, the possibility of a lowered sampling resistance has to be considered in the design of concrete stream ciphers (see Subsection 8.4.2).

7.3 Time-Memory-Data Tradeoff Attacks

In this section, we first make some general remarks on TMD-TO attacks against KSG-based stream ciphers and then describe three such attacks. As explained in Subsection 7.1.1, we consider KSG-based stream ciphers to be defined by the inner state length n, the key length KL, the IV length IVL, and the algorithmic components $LOAD : \{0,1\}^{KL} \times \{0,1\}^{IVL} \longrightarrow \{0,1\}^n$, $MIX : \{0,1\}^n \longrightarrow \{0,1\}^n$, and $OUTBLOCK : \{0,1\}^n \longrightarrow \{0,1\}^n$, where

$$q_{\text{load}} = q_{\text{load}}(k, IV) = LOAD(k, IV)$$

and $q_{\text{mixed}} = MIX(q_{\text{load}})$.

A TMD-TO attacker is supposed to know a certain amount of keystream (usually called *data*), which has its origin in one session, i.e., it was generated under one secret session key k. TMD-TO attacks are generic in the sense that they assume that the security-relevant components MIX and $OUTBLOCK$ are ideally designed, which is reflected by the assumption that the attacker has only black-box access to MIX and $OUTBLOCK$. One distinguishes *passive* TMD-TO attacks, in which the attacker knows a certain amount of keystream generated w.r.t. to one or several IVs known to him, and *active* TMD-TO attacks, in which the attacker gets keystream packets generated w.r.t. to IVs of his choice.[3] Note that the TMD-TO attacks discussed in this section are passive, while our security lower bound in Section 7.5 refers to active TMD-TO attacks. Consequently, the security guarantee of the LIZARD-construction also holds w.r.t. the strong class of *chosen-IV attackers*.

[3] In the spirit of this definition, our TMD-TO attacks against Sprout-like ciphers in Section 6.3 were passive as they did not require to choose particular IVs.

Following [BS00] and [BSW01], TMD-TO attacks are associated with the cost metrics *time*, *memory*, and *data*, which are scalable in the sense that a smaller amount of data can be compensated by a larger amount of time and/or memory. This implies that the cost behavior of TMD-TO attacks is usually expressed as a so-called *tradeoff curve* of type $f(T, M, D) = B$, where B is some number. The interpretation is that if one invests time T, memory M, and data D such that $f(T, M, D) = B$, then the attack reaches its goal with significant success probability. In this chapter, we rather understand the cost of a TMD-TO attack to be the minimal number C for which the relation $f(T, M, D) = B$ can be fulfilled under the condition that none of the metrics T, M, D exceeds C. For instance, the tradeoff curve of Babbage's attack is $T \cdot D = 2^n$, which implies *TMD cost* of $2^{n/2}$.

Note here that TMD-TO attacks are sometimes considered to be divided into a (key-independent) *precomputation phase*, where, based on the components of the cipher, some search data structure is constructed, and the *online phase*, in which the now given data (e.g., passively or actively obtained keystream) is used for reaching the attack goal based on the previously computed search data structure (see, e.g., the attacks in [BS00] and [BSW01]). The size of this search data structure is usually counted as memory. In this chapter, we do not distinguish between the costs of these two phases and take as cost the *overall TMD cost* of both phases. This is in line with our TMD-TO attacks against Sprout-like ciphers (see Section 6.3) and our security analysis for Lizard (see Section 8.4), where we also focus on the overall cost (as the maximum of time, memory, and data complexity) of the complete attacks. In particular, we do not treat higher precomputation costs as 'acceptable', as is done in some other works (see, e.g., [MSS+17]).

We use the notation $\tilde{O}(f(n))$ for expressing the asymptotic cost behavior of algorithms, which means that operations which rather need time (or memory) $\Theta(n)$, such as binary search in a sorted array of size 2^n, are counted with cost $\tilde{O}(1)$ only. Moreover, at some places we use the relation $\lim_{N\to\infty}(1 - 1/N)^N = e^{-1}$, which implies that the probability that at least one of N independent trials with success probability $1/N$ is successful is around $1 - e^{-1}$ if N is large enough.

Let us start with the traditional attack of Babbage in [Bab95]:

Theorem 7.1

We consider a KSG-based stream cipher of inner state length n working in packet mode or in one-stream mode. Suppose that the attacker Eve knows a collection of pieces of keystream, which can have their origin in different packets of one session, and which contain D different subsequences of n consecutive keystream bits. Then Eve can compute the initial state of at least one packet in time $\tilde{O}(2^n/D)$ with success probability around $1 - e^{-1}$ (which implies a tradeoff curve $T \cdot D = 2^n$ and minimum TMD cost $2^{n/2}$).

Sketch of the Proof of Theorem 7.1: Eve generates, based on randomly and independently chosen inner states $y \in \{0,1\}^n$, $T = 2^n/D$ times a pair $(y, OUTBLOCK(y))$. As $T \cdot D = 2^n$, with probability around $1 - e^{-1}$ there is some pair $(y, OUTBLOCK(y))$ such that y equals one of the inner states behind the T keystream subsequences of length n known to Eve. As the state transition function π is efficiently invertible, this allows to efficiently compute the secret initial state q^i_{init} of the corresponding packet, respectively of the only initial state q_{init} if the cipher runs in one-stream mode. Setting $T = D = 2^{n/2}$ implies an attack of time and data $\tilde{O}(2^{n/2})$. □

In one-stream mode, the attack in Theorem 7.1 discovers the only initial state q_{init} and, thus, the whole keystream of this session. In packet mode, the initial state of at least one packet is discovered and, thus, the whole packet can be computed. Consequently, it holds:

Corollary 7.1

We consider a KSG-based stream cipher of inner state length n working in packet mode with packet length $R = n + 1$. Suppose that Eve knows data consisting of D different keystream packets. Then Eve can distinguish the *pseudorandom scenario*, where the data is generated by the cipher, from the *random scenario*, where the data comes from a truly random source, in time $\tilde{O}(2^n/D)$ with advantage around $\sqrt{e^{-1}} - e^{-1}$.

Sketch of the Proof of Corollary 7.1: For each inner state $y \in \{0,1\}^n$, we denote by $Z(y) \in \{0,1\}^{n+1}$ the sequence of the first $n+1$ keystream bits generated on initial state y. Moreover, let $\mathcal{D}^*$ denote the set of all those packets $Z \in \{0,1\}^{n+1}$ contained in the data for which there is some inner state $y \in \{0,1\}^n$ with $Z(y) = Z$. We know that in the pseudorandom case all D packets contained in the data belong to $\mathcal{D}^*$, while in the random case the probability that $|\mathcal{D}^*|$ deviates significantly from $D/2$ is negligibly small.

Eve now generates at most $T = 2^n/D$ times a pair $(y, Z(y))$ for randomly and independently chosen inner states $y \in \{0,1\}^n$ and stops with output `pseudorandom` if she gets a collision, i.e., if she generated some y for which $Z(y)$ coincides with one of the D packets contained in the data. If after $T = 2^n/D$ rounds no collision happened, she stops and outputs `random`.

By the same arguments as in the proof of Theorem 7.1, it follows that in the pseudorandom case, the probability that Eve outputs `pseudorandom` is around $1 - e^{-1}$. In the random case, the probability that Eve outputs `pseudorandom` is around $1 - (1 - ((D/2)/2^n))^{2^n/D} \approx 1 - \sqrt{e^{-1}}$. This implies an advantage of

$$\left|\left(1 - e^{-1}\right) - \left(1 - \sqrt{e^{-1}}\right)\right| = \sqrt{e^{-1}} - e^{-1}. \qquad \square$$

For achieving a higher resistance than $\tilde{O}(2^{n/2})$ against packet prediction attacks and session key recovery attacks, it must be hard to compute the session key from a given packet initial state. Note that this is *not the case* for Trivium, Grain, and A5/1, as for all these ciphers, *MIX* is efficiently invertible and its result is taken directly as the initial state (see Section 5.3). Here, *packet prediction attacks* refer to stream ciphers working in packet mode. The goal is to compute the keystream packet corresponding to a new initialization vector IV^* under the secret session key, where *new* means that no keystream bit of the packet to IV^* was known to Eve.

But even if the mixing algorithm is presumably preimage resistant (as in the case of the Bluetooth cipher E_0; cf. Subsection 5.3.1), the security against session key recovery attacks will be only $n/2$ if the state initialization algorithm implies that the packet initial states q^i_{init} are equal to the respective packet mixing states q^i_{mixed} (as they are for E_0). More precisely, it holds:

Theorem 7.2

We consider a KSG-based stream cipher working in packet mode with a state initialization for which the packet initial states are equal to the packet mixing states. Then, with probability around $1 - e^{-1}$, an attacker Eve can compute the secret session key in time $\tilde{O}(2^n/D)$ if she knows D different n-bit keystream packet prefixes. This implies a tradeoff curve $T \cdot D = 2^n$ and thus minimum TMD cost $2^{n/2}$.

Proof of Theorem 7.2: By the assumption it holds that

$$q^i_{\text{init}} = MIX\Big(q_{\text{load}}\Big(k, IV^i\Big)\Big)$$

for all packets i.

Eve generates $T = 2^n/D$ times a pair $(q, OUTBLOCK(MIX(q)))$ for randomly and independently chosen inner states $q \in \{0,1\}^n$. As $T \cdot D = 2^n$, with probability around $1 - e^{-1}$, for some q, $MIX(q)$ equals the initial state of one of the D keystream packet prefixes known to Eve, which implies $q = q_{\text{load}}(k, IV^i)$. This allows to compute k from q as IV^i is public. Here we assumed that k can be efficiently computed from $q_{\text{load}}(k, IV^i)$ and IV^i, which is true for all KSG-based stream ciphers which are known to us (see, e.g., Section 5.3). □

Consequently, to achieve beyond-the-birthday-bound security against generic TMD-TO attacks, the state initialization algorithm has to provide $q^i_{\text{init}} \neq MIX(q_{\text{load}}(k, IV^i))$.

The next theorem shows that if the packet initial states are computed according to the LIZARD-construction (see Eq. (7.1)), i.e., as $q^i_{\text{init}} = MIX(q^i_{\text{load}}) \oplus k$, where $q^i_{\text{load}} = k \oplus IV^i$, then there is a TMD-TO attack of TMD cost $\tilde{O}(2^{(2/3)n})$.

Theorem 7.3

We consider a KSG-based stream cipher working in packet mode for which the inner state length n equals the IV length and the session key length, and we assume that for all $i \geq 1$, the packet initial states q_{init}^i are generated according to Eq. (7.1), see above. Suppose that Eve knows a set of D packet prefixes of length n, i.e., a set of pairs $\{(IV^i, PREFIX(IV^i)) \mid i \in I^*\}$ for a set $I^* \subseteq \mathbb{N}$, where $D = |I^*| \geq 2^{n/2}$ and, for each $i \in I^*$,

$$PREFIX\Big(IV^i\Big) = OUTBLOCK\Big(MIX\Big(IV^i \oplus k\Big) \oplus k\Big).$$

Then, with constant positive probability, Eve can compute the secret session key in time $\tilde{O}(D + \frac{2^{2n}}{D^2})$, which implies TMD cost of $\tilde{O}(2^{(2/3)n})$.

Sketch of the Proof of Theorem 7.3: Our attack uses the idea of the *Slidex attack* of Dunkelman, Keller, and Shamir [DKS12] against the one-key Even-Mansour cipher. We describe the attack but only sketch the analysis of the success probability. For an exact specification of the success probabilities we refer to [DKS12].

Let Q^* denote the set of all initial states corresponding to the indices in I^*, i.e.,

$$Q^* = \left\{ MIX\Big(IV^i \oplus k\Big) \oplus k \;\middle|\; i \in I^* \right\}.$$

Observe that before starting the attack, Q^* is unknown to Eve.

In the first phase of the attack, Eve generates D times a pair $(q, OUTBLOCK(q))$ for randomly and independently chosen inner states $q \in \{0,1\}^n$. Whenever q falls into Q^*, which happens with probability $\frac{D}{2^n}$, Eve sees a collision of $OUTBLOCK(q)$ with $PREFIX(IV^i)$ for some $i \in I^*$ and it holds $MIX(IV^i \oplus k) \oplus k = q$.

Consequently, after the first phase, a standard Chernoff bound argument yields that Eve knows with constant positive probability a set of pairs $\{(IV^i, q_{\text{init}}(IV^i)) \mid i \in I^{**}\}$ for some $I^{**} \subseteq I^*$ with $|I^{**}| \geq \frac{D^2}{2^n}$.

In a second phase, Eve generates $\frac{2^n}{D^2/2^n} = \frac{2^{2n}}{D^2}$ times a pair $(u, MIX(u))$ for randomly and independently chosen inner states $u \in \{0,1\}^n$. Eve stops with u if

$$u \oplus IV^i = MIX(u) \oplus q_{\text{init}}\Big(IV^i\Big) \tag{7.3}$$

for some $i \in I^{**}$ and publishes the hypothesis that $u \oplus IV^i$ equals the session key k.

In [DKS12], it is shown that the event that Eq. (7.3) holds implies the event $k = u \oplus IV^i$ with positive constant probability if MIX is supposed to behave like a random permutation. Note further that, as $\frac{2^{2n}}{D^2} \cdot |I^{**}| \geq \frac{2^{2n}}{D^2} \cdot \frac{D^2}{2^n} = 2^n$, the event that Eq. (7.3) is fulfilled during the second phase happens with probability around $1 - e^{-1}$. □

7.4 A Random Oracle Model for the LIZARD-Construction

In this section, we introduce a formal framework for analyzing the security of the LIZARD-construction against generic TMD-TO attacks and start with a formal definition.

Definition 7.3: The LIZARD-Construction

A KSG-based stream cipher is designed according to the LIZARD-*construction* if it fulfills the following criteria:

(L1) The construction refers to three auxiliary parameters n, π, R, where n denotes the inner state length, $\pi : \{0,1\}^n \longrightarrow \{0,1\}^n$ denotes the state transition function of the underlying KSG, and R denotes the packet length. It is required that $R \geq n$, that π is bijective, and that for all inner states $q \in \{0,1\}^n$ the period of the sequence $(\pi^r(q))_{r=0}^{\infty}$ is greater than R.

(L2) The construction refers to a set of secret session keys and a set of public initialization vectors which are both defined to be $\{0,1\}^n$.

(L3) The construction consists in the following rules (see (L5)) how to generate from a secret key $k \in \{0,1\}^n$ and packet initialization vectors $IV^i \in \{0,1\}^n$ the packet initial states $q_{\text{init}}(k, IV^i) \in \{0,1\}^n$ and the corresponding keystream packets

$$PACKET\Big(k, IV^i\Big) = PACKET\Big(q_{\text{init}}\Big(k, IV^i\Big)\Big) \in \{0,1\}^R .$$

(L4) These rules in (L5) refer to a bijective state mixing function $MIX : \{0,1\}^n \longrightarrow \{0,1\}^n$ and a π-iterative output block function $OUTBLOCK : \{0,1\}^n \longrightarrow \{0,1\}^n$, which are considered to be the main components of the cipher (see Definition 7.2 for the definition of π-iterativeness).

(L5) For all secret keys $k \in \{0,1\}^n$ and initialization vectors $IV^i \in \{0,1\}^n$, it holds that

$$q_{\text{init}}\Big(k, IV^i\Big) = MIX\Big(k \oplus IV^i\Big) \oplus k \tag{7.4}$$

and

$$PACKET\Big(q_{\text{init}}\Big(k, IV^i\Big)\Big) = (z_0, z_1, \ldots, z_{R-1}) ,$$

where for all r, $0 \leq r \leq R - n$, it holds

$$(z_r, z_{r+1}, \ldots, z_{r+n-1}) = OUTBLOCK\Big(\pi^r\Big(q_{\text{init}}\Big(k, IV^i\Big)\Big)\Big). \tag{7.5}$$

Note that Eq. (7.5) corresponds to the usual keystream generation definition (see Section 7.2, especially Definition 7.1). Note further that the stream cipher LIZARD, as defined in Chapter 8, differs from the design features of the LIZARD-construction in some minor points, which do not harm our security bounds. For instance, in contrast to condition (L2), the IV length of LIZARD is smaller than the inner state length. Observe that in our situation, a smaller IV length lowers the power of a chosen-IV attacker, i.e., our security lower bounds also hold for a modified LIZARD-construction of IV length smaller than n. We refer the reader to Subsection 8.3.5 for a detailed description of how LIZARD deviates from the general LIZARD-construction and an in-depth explanation of why these modifications do not harm the cipher's security.

In our security analysis we consider, in the spirit of [GT15], the LIZARD-construction to be defined over an *ideal* state mixing function MIX, behaving like a random permutation over $\{0,1\}^n$, and over an *ideal* output block function $OUTBLOCK$, behaving like a random π-iterative function. We model the security of the LIZARD-construction against generic TMD-TO attacks by the adversary Eve's success probability to win the following *packet prediction game* (see Definition 7.4) with a limited number of oracle queries against Alice, who holds a secret session key k. Eve has black-box access to the ideal components MIX and $OUTBLOCK$, and is allowed to ask for keystream packets $PACKET(k, IV^i)$ generated w.r.t. the secret session key k held by Alice and IVs IV^i of Eve's choice. Eve wins the game if, after asking a certain number of oracle queries, she is able to predict the keystream packet w.r.t. to a new IV, which has not been asked before.

From now on, for the sake of shortness of the denotations, we denote the component functions MIX and $OUTBLOCK$ by P and F, respectively, the construction function $PACKET$ by E, and initialization vectors by $x \in \{0,1\}^n$ (respectively x', x^* etc.).

Definition 7.4: The Packet Prediction Game

(i) The game depends on the global parameters n, π, R, which satisfy the rules in Definition 7.3, and a parameter M, which bounds the number of oracle queries. The game is divided into a *query phase* and a *prediction phase*.

(ii) At the beginning, Alice chooses randomly and w.r.t. the uniform distribution a secret triple $\omega = (k_\omega, P_\omega, F_\omega)$, where

- $k_\omega \in \{0,1\}^n$ denotes the secret session key,
- $P_\omega : \{0,1\}^n \longrightarrow \{0,1\}^n$ denotes a random permutation (correponding to MIX),
- $F_\omega : \{0,1\}^n \longrightarrow \{0,1\}^n$ denotes a random π-iterative function (corresponding to $OUTBLOCK$).

We denote by Ω the corresponding probability space of all such triples together with the uniform distribution.

(iii) The adversary Eve is supposed to be a randomized oracle algorithm of potentially unbounded computational power, who is allowed to pose *component queries* of type '$P(u) = ?$', or '$P^{-1}(v) = ?$', or '$F(y) = ?$' for inputs $u, v \in \{0,1\}^n$ and $y \in \{0,1\}^n$, which are correctly answered by Alice by $P_\omega(u)$, $(P_\omega)^{-1}(v)$, or $F_\omega(y)$, respectively.

(iv) Moreover, Eve is allowed to pose *construction queries* of the form '$E(x) = ?$', where $x \in \{0,1\}^n$, which will be answered by Alice with the keystream packet $E_\omega(x)$ corresponding to the initial state

$$y := P_\omega(x \oplus k_\omega) \oplus k_\omega$$

induced by the session key k_ω and the initialization vector x (see Eq. (7.4)). Note that this keystream packet $E_\omega(x)$ is the concatenation of R/n F-values. In particular,

$$E_\omega(x) = F_\omega(y) \Big|\Big| F_\omega(\pi^n(y)) \Big|\Big| F_\omega\Big(\pi^{2n}(y)\Big) \Big|\Big| \cdots \Big|\Big| F_\omega\Big(\pi^{(R/n-1)n}(y)\Big) \ .$$

W.l.o.g., for the sake of simplicity we assume in our proof that n divides R.

(v) In the query phase, Eve poses exactly M oracle queries. In the prediction phase, Eve has to submit a pair $(x^*, z^*) \in \{0,1\}^n \times \{0,1\}^n$, where x^* does not occur as input of an E-query in the query phase. Eve wins if $z^* = F_\omega(P_\omega(x^* \oplus k_\omega) \oplus k_\omega)$, i.e., if z^* equals the block of the first n bits of the keystream packet $E_\omega(x^*)$ corresponding to the initial state $P_\omega(x^* \oplus k_\omega) \oplus k_\omega$, i.e., the keystream packet corresponding to session key k_ω and initialization vector x^*.

(vi) Besides the number M of oracle queries, the essential cost parameter is the winning probability of Eve, which is measured with respect to the uniform distribution on Ω and the internal randomization of Eve.

Observe that generic TMD-TO attacks (as described in Section 7.3) against the LIZARD-construction can be formulated in a straightforward way as packet prediction or key recovery games in the sense of Definition 7.4. Here, the cost metric *data* corresponds to the number of E-queries (possibly multiplied by the packet length), while each evaluation of the cipher components *MIX* and *OUTBLOCK* corresponds to a P-, P^{-1}-, or F-query in the sense of Definition 7.4. Hence, the overall number of oracle queries in our game constitutes a lower bound for the cost metric *time* of corresponding generic TMD-TO attacks against the LIZARD-construction. Note here that evaluations of the state transition function π by Eve do not count in our security analysis; it is supposed that π is completely

known to Eve. The function π has to satisfy rule (L1) of Definition 7.3, but apart from that, it can be arbitrarily easy. Our lower bound arguments even work if π is linear.

We conclude this section by describing how a random uniformly distributed π-iterative function can be generated.

Generating a random π-iterative function F: Note that, as π is bijective, the strongly connected components of the directed graph $G_\pi = (\{0,1\}^n, E_\pi)$, where $E_\pi = \{(v, \pi(v)) \mid v \in \{0,1\}^n\}$, are simple cycles $C^1, \ldots, C^s$ of sizes $d_1, \ldots, d_s$, which we call *π-cycles*.

For each π-cycle C^j, $1 \leq j \leq s$, fix a starting point $v_0^j \in C^j$. Note that $C^j = \{v_0^j, \ldots, v_{d_j-1}^j\}$, where for all i, $1 \leq i \leq d_j - 1$, it holds $v_i^j = \pi^i(v_0^j)$.

A uniformly distributed π-iterative function F can be defined by choosing for all j, $1 \leq j \leq s$, randomly and independently a uniformly distributed bitstring

$$b^j = \left(b_0^j, \ldots, b_{d_j-1}^j\right) \in \{0,1\}^{d_j}$$

and defining $F(v_i^j)$ for all i, $0 \leq i \leq d_j - 1$, by

$$F\left(v_i^j\right) = \left(b_i^j, b_{(i+1) \bmod d_j}^j, \ldots, b_{(i+n-1) \bmod d_j}^j\right).$$

Here we took into account that, by Definition 7.3, the sizes of the cycles are each larger than $R \geq n$. Note that the entropy of a random π-iterative function is 2^n.

7.5 The Security Lower Bound Proof

In this section, we show the main result of this chapter, a sharp security lower bound for the LIZARD-construction. At several places, our lower bound proof uses a combinatorial result proved by Chen et al. in [CLL+14], namely Theorem 1 in Section 3, which is known as the *Sum-Capture Theorem*.

For motivating the use of this result, let us consider the situation that Alice holds a secret triple $\omega = (k_\omega, P_\omega, F_\omega)$ and that Eve asked a number of queries, where $U, X, Y \subseteq \{0,1\}^n$ denote the sets of inputs of the P-queries, E-queries and F-queries asked by Eve so far, respectively. For Eve it is desirable that the choice of U, X, Y implies a sufficiently large set of triples $(u, x, y) \in U \times X \times Y$ for which $x \oplus u = P_\omega(u) \oplus y$, as such triples give nontrivial information about the secret session key k_ω.

In particular, if $F_\omega(y)$ is not equal to the prefix of length n of $E_\omega(x)$, then $x \oplus u \neq k_\omega$. If $F_\omega(y)$ equals the prefix of length n of $E_\omega(x)$, then one can show that $x \oplus u = k_\omega$ holds with constant positive probability.

Thus, Eve wins the game with high probability if she manages to pose the queries in such a way that for all keys $k \in \{0,1\}^n$ there is some $(u, x, y) \in U \times X \times Y$ fulfilling $x \oplus u = P_\omega(u) \oplus y = k$. It is easy to show that this goal can be reached by Eve with high probability with $\tilde{O}(2^{(2/3)n})$ oracle queries.

The Sum-Capture Theorem from [CLL+14], which we present in a slightly modified form, shows that reaching this goal with significantly less than $\tilde{O}(2^{(2/3)n})$ oracle queries succeeds only with exponentially small success probability.

Theorem 7.4: Sum-Capture Theorem

Let P denote a uniformly random permutation over $\{0,1\}^n$, let $N = 2^n$, and fix an arbitrary number M, $9n \leq M \leq N/2$. Suppose that Eve (who is supposed to be a probabilistic algorithm) poses a sequence $U = \{u_1, \ldots, u_M\}$ of M P-queries. For any subsets $X, Y \subseteq \{0,1\}^n$ let

$$\mu(P, U, X, Y) = |\{(u, x, y) \in U \times X \times Y \mid x \oplus u = y \oplus P(u)\}|\,.$$

Then the probability for the event that there are subsets $X, Y \subseteq \{0,1\}^n$ such that

$$\mu(P, U, X, Y) \geq \frac{M \cdot |X| \cdot |Y|}{N} + \frac{2M^2 \cdot \sqrt{|X| \cdot |Y|}}{N} + 3\sqrt{n \cdot M \cdot |X| \cdot |Y|} \qquad (7.6)$$

is at most $2/N$, where the probability is taken over the random choice of P and the internal randomization of Eve. □

7.5.1 The Main Theorem

We will now formulate our main result (Theorem 7.5) based on this technical definition:

Definition 7.5: $B(M, R, n)$

For natural numbers M, R, n, where $R \geq n$, let

$$B(M, R, n) = 2^{-n} \cdot M^3 \cdot \left(R + n - 1 + 2\sqrt{R + n - 1}\right) + 3 \cdot \sqrt{n \cdot M^3 \cdot (R + n - 1)}.$$

Note that $B(M, R, n)$ equals the term on the right-hand side of Eq. (7.6) for $|X| = M$ and $|Y| = (R + n - 1)M$.

In the formulation of the following Theorem 7.5, there occurs some balancedness parameter Δ, which will be needed in the proof for identifying computational transcripts which have some critical combinatorial properties.

Theorem 7.5: The Main Theorem

Suppose that the parameters M, R, n satisfy the following three rules for some number Δ:

(1) $B(M,R,n) + 2 \cdot \Delta \cdot M + \frac{(R+n)\cdot M^2}{\Delta} \leq \left(1 - \frac{1}{\sqrt{2}}\right) \cdot 2^n$,

(2) $22 \cdot 2^{-(n-1)} \cdot R \cdot M^2 + \sqrt{\frac{n \cdot M}{2}} \leq \frac{\Delta - (R+n-1)}{R+n-1}$,

(3) $\Delta \cdot ((n+R) \cdot M) \leq \ln(2) \cdot 2^{n-3}$.

Then Eve's success probability to win the packet prediction game with parameters R, n, Δ with M oracle queries is bounded by

$$34 \cdot 2^{-n} + M \cdot e^{-n} + M \cdot (\Delta + 2) \cdot 2^{-(n-1)} + 11 \cdot (R + 4n) \cdot M \cdot 2^{-(n-1)}.$$

This implies the following asymptotic lower bound result, which can be derived straightforwardly from Theorem 7.5.

Corollary 7.2

Let $\epsilon > 0$ and $a > 1$ be constants and suppose that $M \leq 2^{(2/3-\epsilon)n}$, $R \leq n^a$, and $\Delta = \lfloor 2^{n/3} \rfloor$. Then M, R, n, Δ satisfy all rules in Theorem 7.5 and Eve's success probability to win the packet prediction game with parameters R, n, Δ with M oracle queries is bounded by $3 \cdot 2^{-\epsilon \cdot n}$ if n is large enough. □

Getting started with the proof of Theorem 7.5: The remaining part of this chapter is devoted to the proof of Theorem 7.5. It is divided into twelve subsections and accompanied by Appendix 7.A, in which some useful basics on Chernoff bound techniques are presented. In the first three subsections (Subsections 7.5.2 to 7.5.4), we describe and formalize the computational behavior of Eve in a way which is similar to the framework of security lower bounds for iterated Even-Mansour ciphers. Based on this formalism, we are able to describe the main ideas for proving Theorem 7.5 in Subsection 7.5.4. At the end of this subsection, we state Lemma 7.1, from which Theorem 7.5 can be directly concluded. The remaining nine subsections (Subsections 7.5.5 to 7.5.13) are more technical and devoted to the proofs of the four claims of Lemma 7.1, which directly correspond to the additive terms of the probability bound stated in Theorem 7.5.

7.5.2 The Friendly Alice, Structural Collisions, and Sudden Death

We will prove our security bound for a modified game, in which Alice is supposed to be *friendly* to Eve in the sense that in certain situations Alice provides some additional information to Eve. In particular, Alice informs Eve if Eve managed to discover a so-called *structural collision* (see Definition 7.6). Moreover, she follows a *sudden-death*

rule (see Definition 7.8), which has to do with structural collisions.

Definition 7.6: Structural Collisions

- A pair (x, y), where $x, y \in \{0,1\}^n$, is called a *structural EF-collision* w.r.t. to an elementary event $\omega = (k_\omega, P_\omega, F_\omega)$ if

$$y = \pi^r(P_\omega(x \oplus k_\omega) \oplus k_\omega)$$

  for some r, $-(n-1) \leq r \leq R-1$. Note that this implies that the n-bit block $F_\omega(y)$ is a subblock of packet $E_\omega(x)$ or has at least a nonempty intersection with packet $E_\omega(x)$.

- If (x, y) is a structural EF-collision w.r.t. ω, then the point $\bar{y} = P_\omega(x \oplus k_\omega) \oplus k_\omega$ is called the *reference point* of this collision.

- A pair (x, x'), where $x \neq x' \in \{0,1\}^n$, is called a *structural EE-collision* w.r.t. to ω if the initial states of the packets $E_\omega(x)$ and $E_\omega(x')$ come so close that these packets have a nonempty intersection, i.e., there is some number r, $1 \leq r \leq R-1$, such that

$$\pi^r(P_\omega(x \oplus k_\omega) \oplus k_\omega) = P_\omega(x' \oplus k_\omega) \oplus k_\omega \text{ or}$$
$$\pi^r(P_\omega(x' \oplus k_\omega) \oplus k_\omega) = P_\omega(x \oplus k_\omega) \oplus k_\omega.$$

  Note that this implies that the suffix of packet $E_\omega(x)$ starting at position r equals the prefix of packet $E_\omega(x')$, or that the suffix of packet $E_\omega(x')$ starting at position r equals the prefix of packet $E_\omega(x)$.

Note here that structural EF-collisions are exactly those collisions which are exploited in the classical TMD-TO attacks against stream ciphers. Note further that there may occur collisions which are non-structural but caused by internal collisions of F_ω.

Suppose that Alice holds the elementary event $\omega = (k_\omega, P_\omega, F_\omega)$ and communicates with Eve. The friendly Alice does the following:

Definition 7.7: The Friendly Alice

- Whenever Eve poses an F-query with some input $y \in \{0,1\}^n$ which forms a structural EF-collision (x, y) w.r.t. ω for some $x \in \{0,1\}^n$ which already occurred as input of an E-query posed before, then, besides publishing $F_\omega(y)$, Alice confirms a structural collision, publishes a pointer to the input x and publishes the reference point $P_\omega(x \oplus k_\omega) \oplus k_\omega$ of this collision.

- Whenever Eve poses an E-query with some input $x \in \{0,1\}^n$ which forms a structural EF-collision (x, y) w.r.t. ω for some y which already occurred as input of an F-query posed before, then, besides publishing $E_\omega(x)$, Alice confirms a structural collision, publishes a pointer to y and publishes the reference point $P_\omega(x \oplus k_\omega) \oplus k_\omega$ of this collision.
- Suppose that Eve poses an E-query with some input $x \in \{0,1\}^n$ which forms a structural EE-collision (x, x') w.r.t. ω for some x' which already occurred as input of another E-query posed before. Suppose w.l.o.g. that $\pi^r(P_\omega(x \oplus k_\omega) \oplus k_\omega) = P_\omega(x' \oplus k_\omega) \oplus k_\omega$ for some r, $1 \leq r \leq R-1$. Then, besides publishing $E_\omega(x)$, Alice confirms a structural EE-collision and publishes a pointer to x'. Moreover, Alice publishes the value $y = \pi^r(P_\omega(x \oplus k_\omega) \oplus k_\omega) = P_\omega(x' \oplus k_\omega) \oplus k_\omega$, the value $F_\omega(y)$, and the reference points $\bar{y} = P_\omega(x \oplus k_\omega) \oplus k_\omega$ and y of the resulting structural EF-collisions (x, y) and (x', y).

Next, we formulate the *sudden-death rule*.

Definition 7.8: Sudden Death

Suppose that Alice holds an elementary event $\omega = (k_\omega, P_\omega, F_\omega)$ and consider a situation in which Eve already posed a number of queries. A pair (x, u), where $x, u \in \{0,1\}^n$, is called a *sudden-death pair* w.r.t. ω if the following conditions are fulfilled:

- Eve has already discovered a structural EF-collision (x, y) (which implies that Eve has asked an E-query with input x).
- Eve has already asked a P-query with input u or a P^{-1}-query with output u.
- It holds $x \oplus u = k_\omega$.

Whenever Eve asks a query which causes a sudden-death pair w.r.t. to the secret ω held by Alice, then Alice immediately gives up, the game stops, and Eve wins.

Note that the friendliness of Alice increases Eve's chances to win the prediction game. Consequently, it is sufficient to show the security lower bound of Theorem 7.5 for an adversary Eve who plays the packet prediction game with a friendly Alice. Note further that the TMD-TO attack described in Theorem 7.3 consists in generating a sudden-death pair.

7.5.3 Formalizing the Computational Behavior of Eve

First note the well-known fact, proved, e.g., in [CLL+14] and many other papers, that it is sufficient to prove our security lower bound for deterministic adversaries. For showing this, suppose that Eve is randomized and that the randomization is organized by a number B of random bits. Then Eve's success probability can be written as

$$\Pr[\text{Eve successful}] = \sum_{b \in \{0,1\}^B} 2^{-B} \Pr[\text{Eve successful} \mid b], \tag{7.7}$$

where $\Pr[\text{Eve successful} \mid b]$ denotes the success probability of the deterministic algorithm obtained by assigning b to Eve's random bits.

Consequently, if we show an upper bound on the success probability of all deterministic adversaries, then by Eq. (7.7), this bound also holds for randomized adversaries. Therefore, we assume from now on that Eve is deterministic.

Remember that, during each computation, Eve poses at most M oracle queries, where she either wins via sudden death of Alice or she stops after M queries with the publication of a pair consisting of an initialization vector $x^* \in \{0,1\}^n$ and a keystream prefix $z^* \in \{0,1\}^n$ as final output.

We identify such computations with transcripts

$$\tau = \Big((\mathit{type}_1, \mathit{input}_1, \mathit{output}_1), \ldots, (\mathit{type}_j, \mathit{input}_j, \mathit{output}_j) \Big),$$

$j \leq M$, which are defined to be sequences of query triples corresponding to the oracle queries posed during the computation. Here, $\mathit{type}_r \in \{F, P^{-1}, P, E\}$, input_r, and output_r denote the type, the input, and the output of the r-th oracle query, $r = 1, \ldots, j$, respectively. Note that the output of an oracle query can, besides the output function values of P_ω, or P_ω^{-1}, or F_ω, or E_ω, contain additional information about structural collisions discovered by this query (see Definition 7.7).

If τ has length M, then $(x^*(\tau), z^*(\tau)) \in \{0,1\}^n \times \{0,1\}^n$ denotes the (*initialization vector*, *keystream prefix*) pair published after τ based on τ. For transcripts τ of length j, $1 \leq j \leq M$, and numbers i, $1 \leq i \leq j$, we denote by $\tau^{\leq i}$ the subtranscript corresponding to the first i queries along τ.

Each elementary event $\omega \in \Omega$ defines a unique transcript τ_ω corresponding to the computation of Eve on ω. The length of τ_ω can be smaller than M. This is the case if and only if the next query after the last query of τ_ω produces a sudden-death pair w.r.t. ω (see Definition 7.8). In this case, this next query is not counted to be a part of τ_ω.

Definition 7.9: j-alive Elementary Event

For all j, $1 \leq j \leq M$, an elementary event $\omega \in \Omega$ is called *j-alive* if the transcript τ_ω has length at least j.

Let us denote by $\Omega^{\text{s.death}}$ the set of all elementary events ω for which τ_ω leads to the generation of a sudden-death pair w.r.t. ω, and note that this is equivalent to τ_ω having length smaller than M. Eve's computation τ_ω on an elementary event ω is successful if and only if either the length of τ_ω is smaller than M (i.e., $\omega \in \Omega^{\text{s.death}}$) or the first n bits of the keystream packet corresponding to $x^*(\tau_\omega)$ via ω coincide with $z^*(\tau_\omega)$, i.e.,

$$F_\omega(P_\omega(x^*(\tau_\omega) \oplus k_\omega) \oplus k_\omega) = z^*(\tau_\omega).$$

We denote by $\Omega^{\text{succ}} \subseteq \Omega$ the set of all elementary events leading to a successful computation. Note that $\Omega^{\text{s.death}} \subseteq \Omega^{\text{succ}}$.

7.5.4 Basic Definitions and the Idea of the Proof of Theorem 7.5

For all j, $1 \leq j \leq M$, we denote by $\mathcal{T}^j$ the set of all transcripts τ of length j (i.e., consisting of j query triples) which occur with positive probability, i.e., for which there is some $\omega \in \Omega$ such that τ is the prefix of length j of τ_ω. For each j, $1 \leq j \leq M$, and each transcript $\tau \in \mathcal{T}^j$, we define the following sets corresponding to the queries along τ:

- $X(\tau) = \{x \in \{0,1\}^n \mid \tau \text{ contains an } E\text{-query with input } x\}$,
- $Y(\tau) = \{y \in \{0,1\}^n \mid \tau \text{ contains an } F\text{-query with input } y\}$,[4]
- $U(\tau) = \{u \in \{0,1\}^n \mid \tau \text{ contains a } P\text{-query with input } u, \text{ or a } P^{-1}\text{-query with output } u\}$,
- $V(\tau) = \{v \in \{0,1\}^n \mid \tau \text{ contains a } P\text{-query with output } v, \text{ or a } P^{-1}\text{-query with input } v\}$,
- $X^*(\tau) = \{x \in X(\tau) \mid x \text{ occurs at the left-hand side of some structural } EF\text{-collision discovered during } \tau\}$,
- $\bar{Y}^*(\tau) = \{\bar{y} \in \{0,1\}^n \mid \bar{y} \text{ is the reference point of some structural } EF\text{-collision discovered during } \tau\}$,
- $\mathit{Coll}(\tau) = \{(x, \bar{y}) \mid x \in X^*(\tau), \text{ and } \bar{y} \in \bar{Y}^*(\tau) \text{ is the reference point of a structural } EF\text{-collision } (x, y) \text{ discovered during } \tau\}$,
- $\bar{Y}^{(r)}(\tau) = \{\bar{y} \in \{0,1\}^n \mid \pi^r(\bar{y}) \in Y(\tau)\}$,
- $\bar{Y}(\tau) = \bigcup_{r=-(n-1)}^{R-1} \bar{Y}^{(r)}(\tau)$.

[4]Note that we put also those y to $Y(\tau)$ which occur at the right-hand side of a structural EF-collision that was disclosed by Alice as additional information to an EE-collision, see Definition 7.7.

Remember that we suppose Alice to be friendly in the sense of Definition 7.7. This implies that the oracle answers of Alice along τ yield the complete knowledge about $\mathit{Coll}(\tau)$, $\bar{Y}^*(\tau)$, and $X^*(\tau)$.

Observe that $X(\tau)$ corresponds to the set of all initialization vectors for which Eve gets the corresponding keystream packet from Alice during τ, and that $X^*(\tau)$ corresponds to the set of all those initialization vectors for which Eve even knows the initial state of the corresponding packet. These known initial states are contained in the set $\bar{Y}^*(\tau)$. The set $\bar{Y}(\tau)$ corresponds to the set of all initial states for which a part of the corresponding keystream packet has been discovered during τ.

Note further that $\mathit{Coll}(\tau)$ yields all information also about structural EE-collisions discovered during τ. This is because, due to Definition 7.7, for each structural EE-collisions (x, x') discovered during τ, there is some $y \in Y(\tau)$ such that (x, y) and (x', y) are structural EF-collisions discovered during τ. Moreover, $\mathit{Coll}(\tau)$ defines a one-to-one correspondence between $X^*(\tau)$ and $\bar{Y}^*(\tau)$, which is established by the bijection $P_\omega(x \oplus k_\omega) \oplus k_\omega$ for an $\omega \in \Omega(\tau)$ (see Definition 7.10 below). Note that, by definition, this bijection is the same for all τ-consistent elements $\omega \in \Omega(\tau)$.

Definition 7.10: τ-consistent Elementary Events / Keys

For all j, $1 \leq j \leq M$, and transcripts $\tau \in \mathcal{T}^j$, we denote by $\Omega(\tau)$ the set of all *τ-consistent elementary events* ω, i.e.,

$$\Omega(\tau) = \left\{ \omega \in \Omega \;\middle|\; \omega \text{ is } j\text{-alive and } \tau_\omega^{\leq j} = \tau \right\}.$$

$\Omega(\tau)$ defines the set $K(\tau) \subseteq \{0,1\}^n$ of *τ-consistent keys*, i.e.,

$$K(\tau) = \{k_\omega \mid \omega \in \Omega(\tau)\}.$$

Note that $\Omega(\tau)$ defines a probability distribution $\Pr_\tau$ on $K(\tau)$. For all $k \in K(\tau)$, it holds

$$\Pr_\tau[k] = \frac{|\{\omega \in \Omega(\tau) \mid k_\omega = k\}|}{|\Omega(\tau)|}.$$

After the computation τ has happened, Eve's knowledge about the secret ω can be identified with the probability space $\Omega(\tau)$ with the uniform distribution. Note that the induced probability distribution $\Pr_\tau$ on $K(\tau)$ does not need to be uniform. Actually, the analysis of the distribution $\Pr_\tau$ on $K(\tau)$ will be the key ingredient for proving Theorem 7.5.

For describing the main idea of the proof, let us consider the situation that Alice holds a secret $\omega = (k_\omega, P_\omega, F_\omega)$ and that Eve performed M queries without generating a sudden-death pair. Let us denote by τ the corresponding transcript τ_ω.

Clearly, during τ the set of τ-consistent keys becomes smaller and smaller. For getting

a first impression of how key candidates $k \in \{0,1\}^n$ are discarded during τ, suppose that τ contains query triples (P,u,v), (E,x,p), (F,y,z) for which $x \oplus u = k$ and $\pi^r(v \oplus k) = y$ for some r, $0 \leq r \leq R-1$. Then there are three possibilities:

1.) z does not equal the substring of packet $p \in \{0,1\}^R$ starting at position $r+1$, or, if $r > R-n$, the prefix of z of length $R-r$ does not equal the suffix of length $R-r$ of packet p. Then k cannot be the right key, i.e., $k \notin K(\tau)$.

2.) z equals the substring of packet $p \in \{0,1\}^R$ starting at position $r+1$, or, if $r > R-n$, the prefix of z of length $R-r$ equals the suffix of length $R-r$ of packet p, but $(x, v \oplus k)$ does not belong to $Coll(\tau)$. This implies that (x,y) does not form a structural EF-collision (which would be the case if k was the right key k_ω) and that the collision of z with p is caused by a (non-structural) internal collision of F_ω. Consequently, $k \notin K(\tau)$.

3.) z equals the substring of packet $p \in \{0,1\}^R$ starting at position $r+1$, or, if $r > R-n$, the prefix of z of length $R-r$ equals the suffix of length $R-r$ of packet p, and $(x, v \oplus k) \in Coll(\tau)$. Then it also holds that $k \notin K(\tau)$. Otherwise, if $k = k_\omega$, then (x,u) would form a sudden-death pair and the computation would have stopped before τ was completed.

After τ is completed, Eve has to choose a pair $(x^*(\tau), z^*(\tau))$. She is in a promising position if one of the following two conditions is fulfilled.

Condition 7.1

$K(\tau)$ contains only a small number of keys. In this case, Eve can choose one of the few keys in $K(\tau)$, say k', and construct an (*initialization vector*, *keystream prefix*) pair by choosing some input u of a P-query of τ for which $u \oplus k'$ is not input of an E-query. Then $k' = k_\omega$ implies that $(u \oplus k', P_\tau(u) \oplus k')$ is a successful (*initialization vector*, *keystream prefix*) pair.

Condition 7.2

$\Pr_\tau[k_\omega]$ is nontrivially large. In this case, a Bayes decision is successful with nontrivially high probability.

Our proof starts with a combinatorial characterization of τ-consistency of elementary events and keys (see Subsection 7.5.5). This characterization will lead to the formulation of three properties of elementary events ω, for which the following holds. Event ω has one of these properties if and only if $K(\tau_\omega)$ satisfies Condition 7.1 or Condition 7.2 (see Lemma 7.4). We will identify these three properties with the colors *black*, *red*, and *blue* and denote by Ω^{black}, resp. Ω^{red}, resp. Ω^{blue} the sets of elementary events having the

corresponding property (see Subsection 7.5.6). We will further define an elementary event ω to be *green*, if it is neither red, nor black, nor blue, nor belongs to $\Omega^{\text{s.death}}$, and will denote the set of all green elementary events by Ω^{green}.

We prove Theorem 7.5 by using the following relation:

$$\begin{aligned}\Pr_{\Omega}[\Omega^{\text{succ}}] \leq\ & \Pr_{\Omega}\Big[\Omega^{\text{black}} \cap \Omega^{\text{succ}}\Big] + \Pr_{\Omega}\Big[\Big(\Omega^{\text{red}} \cup \Omega^{\text{blue}}\Big) \cap \Omega^{\text{succ}}\Big] \\ & + \Pr_{\Omega}\Big[\Big(\Omega^{\text{s.death}} \setminus \Big(\Omega^{\text{black}} \cup \Omega^{\text{red}} \cup \Omega^{\text{blue}}\Big)\Big) \cap \Omega^{\text{succ}}\Big] \\ & + \Pr_{\Omega}[\Omega^{\text{green}} \cap \Omega^{\text{succ}}].\end{aligned}$$

Consequently, $\Pr_{\Omega}[\Omega^{\text{succ}}]$ can be upper bounded by

$$\begin{aligned}\Pr_{\Omega}[\Omega^{\text{succ}}] \leq\ & \Pr_{\Omega}\Big[\Omega^{\text{black}}\Big] + \Pr_{\Omega}\Big[\Omega^{\text{red}} \cup \Omega^{\text{blue}}\Big] \\ & + \Pr_{\Omega}\Big[\Omega^{\text{s.death}} \setminus \Big(\Omega^{\text{black}} \cup \Omega^{\text{red}} \cup \Omega^{\text{blue}}\Big)\Big] \\ & + \Pr_{\Omega}[\Omega^{\text{succ}} \cap \Omega^{\text{green}}].\end{aligned} \tag{7.8}$$

Black and red elementary events will have the property that for all transcripts τ the following holds: if one event $\omega \in \Omega(\tau)$ is black (resp. red), then all events in $\Omega(\tau)$ are black (resp. red). This justifies to define transcripts τ to be *black* (resp. *red*) if at least one τ-consistent elementary event is black (resp. red). All transcripts which are neither red nor black are called *green*. Note that for green transcripts τ, the set $\Omega(\tau)$ can contain green elementary events and blue elementary events.

This allows to rewrite the probability $\Pr_{\Omega}[\Omega^{\text{succ}} \cap \Omega^{\text{green}}]$ as

$$\Pr_{\Omega}[\Omega^{\text{succ}} \cap \Omega^{\text{green}}] = \Pr_{\Omega}[\Omega^{\text{green}}] \cdot \Pr_{\Omega^{\text{green}}}[\Omega^{\text{succ}}] \leq \Pr_{\Omega^{\text{green}}}[\Omega^{\text{succ}}], \tag{7.9}$$

where $\Pr_{\Omega^{\text{green}}}[\Omega^{\text{succ}}]$ can be written as

$$\begin{aligned}\Pr_{\Omega^{\text{green}}}[\Omega^{\text{succ}}] &= \sum_{\tau \in \mathcal{T}^{M}_{\text{green}}} \Pr_{\Omega^{\text{green}}}[\Omega^{\text{succ}} \cap \Omega^{\text{green}}(\tau)] \\ &= \sum_{\tau \in \mathcal{T}^{M}_{\text{green}}} \Pr_{\Omega^{\text{green}}}[\tau] \cdot \Pr_{\Omega^{\text{green}}(\tau)}[\Omega^{\text{succ}}].\end{aligned} \tag{7.10}$$

Here, $\mathcal{T}^{M}_{\text{green}}$ denotes the set of all green transcripts of length M and $\Omega^{\text{green}}(\tau)$ denotes the set of all green elementary events in $\Omega(\tau)$.

Note that we occasionally use the following denotation for conditional probabilities. Let A, B be subsets (events) of the probability space Ω. Then we write

$$\Pr_{B}[A] := \Pr_{\Omega}[A \mid B] = \frac{\Pr_{\Omega}[A \cap B]}{\Pr_{\Omega}[B]}.$$

By Eqs. (7.8) to (7.10), Theorem 7.5 follows directly from the following Lemma 7.1.

Lemma 7.1

(i) It holds that $\Pr_\Omega[\Omega^{\text{black}}] \leq 34 \cdot 2^{-n}$.

(ii) It holds that $\Pr_\Omega[\Omega^{\text{red}} \cup \Omega^{\text{blue}}] \leq M \cdot e^{-n}$.

(iii) It holds that

$$\Pr_\Omega\Big[\Omega^{\text{s.death}} \setminus \Big(\Omega^{\text{black}} \cup \Omega^{\text{red}} \cup \Omega^{\text{blue}}\Big)\Big] \leq 2^{-(n-1)} \cdot (\Delta + 2) \cdot M.$$

(iv) For all $\tau \in \mathcal{T}^M_{\text{green}}$, it holds that

$$\Pr_{\Omega^{\text{green}}(\tau)}[\Omega^{\text{succ}}] \leq 11 \cdot (R + 4n) \cdot M \cdot 2^{-(n-1)}.$$

We will prove Lemma 7.1 in Subsection 7.5.7 and the subsections following it.

7.5.5 The Characterization of τ-Consistency

Definition 7.11: (τ, k)-critical Point

Let $k \in \{0,1\}^n$. A point $u \in U(\tau)$ is called (τ, k)-*critical* if at least one of the following conditions is fulfilled.

(C1) $u \oplus k \in X(\tau) \setminus X^*(\tau)$ and $P_\tau(u) \oplus k \in \bar{Y}(\tau) \setminus \bar{Y}^*(\tau)$.

(C2) $u \oplus k \in X^*(\tau)$ or $P_\tau(u) \oplus k \in \bar{Y}^*(\tau)$.

Here, $P_\tau(u)$ denotes the output of the P-query on input u along τ, resp. the input of the P^{-1}-query with output u.

The notion of (τ, k)-critical points allows to characterize τ-consistency.

Lemma 7.2

A key $k \in \{0,1\}^n$ is not τ-consistent if and only if there is a (τ, k)-critical point $u \in U(\tau)$.

Proof of Lemma 7.2: We first prove the **if-direction**. Let $k \in \{0,1\}^n$ and suppose that there is some $u \in U(\tau)$ which is (τ, k)-critical. For deriving a contradiction, we assume that $k \in K(\tau)$, i.e., that there is some $\omega \in \Omega(\tau)$ with $k_\omega = k$.

Suppose first that u is (τ, k)-critical via condition (C1) of Definition 7.11. By definition, $P_\tau(u) \oplus k = P_\omega(u) \oplus k_\omega \in \bar{Y}(\tau)$, implying the existence of some r, $-(n-1) \leq r \leq R-1$, such that $\pi^r(P_\omega(u) \oplus k_\omega) \in Y(\tau)$. This implies that $(u \oplus k_\omega, \pi^r(P_\omega(u) \oplus k_\omega))$ has to be classified as a structural EF-collision with reference point $P_\omega(u) \oplus k_\omega$ along τ. But this cannot be true, as by Definition 7.8, $(u \oplus k, u)$ would form a sudden-death pair w.r.t. ω, which implies that $\omega \notin \Omega(\tau)$.

Suppose now that u is (τ, k)-critical via condition (C2) of Definition 7.11. If $u \oplus k = u \oplus k_\omega \in X^*(\tau)$, then $(u \oplus k, u)$ is again a sudden-death pair w.r.t. ω, which implies that $\omega \notin \Omega(\tau)$. If $P_\tau(u) \oplus k \in \bar{Y}^*(\tau)$, then $(u \oplus k, P_\tau(u) \oplus k) \in \mathit{Coll}(\tau)$, which again implies that $(u \oplus k, u)$ is a sudden-death pair and that $\omega \notin \Omega(\tau)$.

Let us now show the **only-if direction** of Lemma 7.2. We fix some j, $1 \leq j \leq M$, some transcript $\tau \in \mathcal{T}^j$ with $\Pr_\Omega[\tau] > 0$, and some key $k \in \{0,1\}^n$ for which there do not exist (τ, k)-critical points $u \in U(\tau)$ in the sense of Definition 7.11.

We have to show that k is τ-consistent. We do this by constructing a permutation P' over $\{0,1\}^n$ and a π-iterative function $F' : \{0,1\}^n \longrightarrow \{0,1\}^n$ such that $\omega' = (k, P', F') \in \Omega(\tau)$.

For all inputs $x \in X(\tau)$, $u \in U(\tau)$, $v \in V(\tau)$, and $y \in Y(\tau)$ of oracle queries posed during τ, we denote by $E_\tau(x)$, $P_\tau(u)$, $P_\tau^{-1}(v)$, and $F_\tau(y)$, respectively, the corresponding oracle answers given by Alice during τ. P' and F' have to satisfy the condition that $P'(u) = P_\tau(u)$ and $F'(y) = F_\tau(u)$ for all $u \in U(\tau)$ and $y \in Y(\tau)$, respectively.

We now have to define P' and F' outside of $U(\tau)$ and $Y(\tau)$, respectively, in such a way that ω' is τ-consistent. We do this by defining P' and F' along the (k, P')-*paths* $(u \oplus k, u, P'(u) \oplus k)$ for all $u \in \{0,1\}^n$, where we go with u through $\{0,1\}^n$ in a certain order. In this process, we dynamically maintain a set $\mathit{Target}(P')$, which is initially set to $\{0,1\}^n \setminus V(\tau)$. Whenever we define $P'(u)$ for a new u, we delete $P'(u)$ from $\mathit{Target}(P')$.

Phase 1 considers all $u \in \{0,1\}^n$ for which $u \oplus k \in X^*(\tau)$.

Then it holds $u \notin U(\tau)$, as otherwise u would be (τ, k)-critical via condition (C2) of Definition 7.11.

We define $P'(u) = \bar{y} \oplus k$, where $\bar{y}$ denotes the unique point in $\bar{Y}^*(\tau)$ for which $(u \oplus k, \bar{y}) \in \mathit{Coll}(\tau)$. Note that the point $\bar{y} \oplus k$ does not belong to $V(\tau)$. Otherwise, if $\bar{y} \oplus k$ equaled $P_\tau(u')$ for some $u' \neq u \in U(\tau)$, then u' would be (τ, k)-critical.

We define F' on the set $\{\pi^r(P'(u) \oplus k) \mid r = -(n-1), \ldots, R-1\}$ according to the packet $E_\tau(u \oplus k)$. Note here that if $-(n-1) \leq r < 0$, then $E_\tau(u \oplus k)$ determines only a suffix of $F'(\pi^r(P'(u) \oplus k))$, and that if $R-n-1 < r \leq R-1$, then $E_\tau(u \oplus k)$ determines only a prefix of $F'(\pi^r(P'(u) \oplus k))$.

Phase 2 treats the (k, P')-paths through those $u \in U(\tau)$ for which $u \oplus k \in X(\tau) \setminus X^*(\tau)$.

Note that for these $u \in U(\tau)$, as they are not (τ, k)-critical, it holds $\bar{y} := P_\tau(u) \oplus k \notin \bar{Y}(\tau)$. This implies that for all r, $-(n-1) \leq r \leq R-1$, it holds that $\pi^r(\bar{y})$ is not in $Y(\tau)$, which allows us to define $F'(\pi^r(\bar{y}))$ according to the packet $E_\tau(u \oplus k)$.

Phase 3 considers all $u \notin U(\tau)$ for which $u \oplus k \in X(\tau) \setminus X^*(\tau)$.

Here, $P'(u)$ has to be chosen in such a way that $P'(u) \oplus k \notin \bar{Y}(\tau)$. Otherwise, there would exist some r, $-(n-1) \leq r \leq R-1$, such that $(u \oplus k, \pi^r(P'(u) \oplus k))$ is an EF-collision discovered during τ, which would imply that $u \oplus k \in X^*(\tau)$ and contradict the assumption made for phase 3.

Corresponding to this, we define a set

$$\mathit{Forbidden}(u) = \left\{ v \in \{0,1\}^n \;\middle|\; v \oplus k \in \bar{Y}(\tau) \right\}$$

and choose

$$P'(u) \in \mathit{Target}(P') \setminus \mathit{Forbidden}(u).$$

Note that for all remaining $u \in \{0,1\}^n$, the values of $P'(u)$ can be freely chosen in $\mathit{Target}(P')$. For all remaining $y \in \{0,1\}^n$, the values of $F'(y)$ can also be freely chosen in $\{0,1\}^n$ in such a way that the π-iterativeness of F' is maintained. □

7.5.6 Assigning Colors to Elementary Events, Transcripts, and Keys

We will now assign the colors *black*, *red*, *blue*, and *green* to elementary events, transcripts, and keys. There will be three colors, namely black, red, and blue, which have to be considered as *bad* in the sense that if ω has a bad color, then τ_ω yields some nontrivial information which helps Eve to win the game. Let us start with the definition of black elementary events, which is partly based on considering the following equivalence relation $\equiv_P$, induced by a permutation P over $\{0,1\}^n$.

Definition 7.12: The Equivalence Relation $\equiv_P$

Let P denote a permutation of $\{0,1\}^n$ and let U be an arbitrary subset of $\{0,1\}^n$.

- For all $u, u' \in U$, let $u \equiv_P u'$ if and only if $u \oplus P(u) = u' \oplus P(u')$.
- Let $\mathit{Max}(P, U)$ denote the maximal size of an equivalence class w.r.t. $\equiv_P$ in U.

Definition 7.13: τ-critical Key

A key $k \in \{0,1\}^n$ is called *τ-critical* if there is some $u \in U(\tau)$ such that $u \oplus k \in X(\tau)$ and $P_\tau(u) \oplus k \in \bar{Y}(\tau)$.

Note that $k \in \{0,1\}^n$ is τ-critical if there is some $u \in U(\tau)$ such that u is (τ, k)-critical with regard to condition (C1) in Definition 7.11.

Definition 7.14: Blackness

- For all j, $1 \leq j \leq M$, a transcript $\tau \in \mathcal{T}^j$ is called *black* if the number of τ-critical keys (see Definition 7.13) exceeds

$$B(M,R,n) = 2^{-n} \cdot M^3 \cdot \left(R+n-1+2\sqrt{R+n-1}\right) + 3\sqrt{n \cdot M^3 \cdot (R+n-1)}$$

  or if

$$Max(P_\tau, U(\tau)) > 5,$$

  where $P_\tau : U(\tau) \longrightarrow \{0,1\}^n$ denotes the injective mapping corresponding to the P- and P^{-1}-queries in τ.

- For all j, $1 \leq j \leq M$, an elementary event $\omega \in \Omega$ is called *j-black* if ω is j-alive (see Definition 7.9) and the transcript $\tau_\omega^{\leq j}$, corresponding to the first j queries along τ_ω, is black.
- Let Ω^j_{black} denote the set of all j-black elementary events and $\mathcal{T}^j_{\text{black}}$ the set of all black transcripts $\tau \in \mathcal{T}^j$.
- Let $\Omega^{\text{black}} = \bigcup_{j=1}^{M} \Omega^j_{\text{black}}$.

Let us next define red transcripts.

Definition 7.15: Redness

- For all j, $1 \leq j \leq M$, a transcript $\tau \in \mathcal{T}^j$ is called *red* if it is not black but it holds $|X^*(\tau)| > \Delta$. (Remember that Δ denotes some balancedness parameter, which was introduced in Theorem 7.5.)
- For all j, $1 \leq j \leq M$, an elementary event $\omega \in \Omega$ is called *j-red* if ω is j-alive (see Definition 7.9) and the transcript $\tau_\omega^{\leq j}$ is red.
- Let Ω^j_{red} denote the set of all j-red elementary events and $\mathcal{T}^j_{\text{red}}$ the set of all red transcripts $\tau \in \mathcal{T}^j$.
- Let $\Omega^{\text{red}} = \bigcup_{j=1}^{M} \Omega^j_{\text{red}}$.

Note that one strategy of Eve could be to pose queries in a first phase in such a way that for the resulting transcript τ it holds that the set $K(\tau)$ of τ-consistent keys is small, and then to try for each key in $K(\tau)$ whether it fits. Redness and blackness of transcripts τ cover exactly the case in which this strategy could be successful.

Lemma 7.3

For all j, $1 \leq j \leq M$, and $\tau \in \mathcal{T}^j$, the following holds. If τ is neither red nor black, then

$$|K(\tau)| \geq 2^n - B(M,R,n) - 2 \cdot \Delta \cdot j.$$

Proof of Lemma 7.3: From Definition 7.11 and Lemma 7.2 we know that $k \in \{0,1\}^n \setminus K(\tau)$ if and only if there is some $u \in U(\tau)$ such that u is (τ,k)-critical via condition (C1) or via condition (C2). Condition (C1) implies that k is τ-critical in the sense of Definition 7.13. As τ is not black, the number of such keys is bounded by $B(M,R,n)$. Condition (C2) implies that $k \in X^*(\tau) \oplus U(\tau)$ or $k \in \bar{Y}^*(\tau) \oplus V(\tau)$. As τ is not red, it holds that $|X^*(\tau) \oplus U(\tau)| \leq \Delta \cdot j$ and $|\bar{Y}^*(\tau) \oplus U(\tau)| \leq \Delta \cdot j$. □

The motivation for considering blue elementary events is as follows. We have seen above that $\Omega(\tau)$, the set of all possible events if Eve sees τ, defines a probability distribution on $K(\tau)$, the set of all keys which are consistent with τ. This distribution is known to Eve. Eve could make a Bayes decision and test the hypothesis that the secret key is the most probable key in $K(\tau)$.

Blue elementary events $\tilde{\omega} = (k_{\tilde{\omega}}, P_{\tilde{\omega}}, F_{\tilde{\omega}})$ will have the property that for $\tau = \tau_{\tilde{\omega}}$ it holds that $\Pr_{\Omega(\tau)}[k_{\tilde{\omega}}]$ is large, i.e., if Alice chooses a blue elementary event, then the success probability of a Bayes decision made by Eve will be nontrivially high.

Definition 7.16: Blueness

- For all j, $1 \leq j \leq M$, an elementary event $\omega \in \Omega$ is called *j-blue* if ω is j-alive (see Definition 7.9) and neither j-black nor j-red and if

$$\left|\left(X\left(\tau_\omega^{\leq j}\right) \oplus k_\omega\right) \cap U\left(\tau_\omega^{\leq j}\right)\right| > \Delta$$

  or

$$\left|\left(\bar{Y}\left(\tau_\omega^{\leq j}\right) \oplus k_\omega\right) \cap V\left(\tau_\omega^{\leq j}\right)\right| > \Delta.$$

- Let Ω^j_{blue} denote the set of all j-blue elementary events.
- Let $\Omega^{\text{blue}} = \bigcup_{j=1}^{M} \Omega^j_{\text{blue}}$.

Definition 7.17: Greenness

- For all j, $1 \leq j \leq M$, a transcript $\tau \in \mathcal{T}^j$ is called *green* if it is neither red nor black.

- For all j, $1 \leq j \leq M$, an elementary event $\omega \in \Omega$ is called *j-green* if ω is j-alive (see Definition 7.9) and neither j-blue, nor j-red, nor j-black.
- Let Ω^j_{green} denote the set of all j-green elementary events and $\mathcal{T}^j_{\text{green}}$ the set of all green transcripts $\tau \in \mathcal{T}^j$.
- Let $\Omega^{\text{green}} = \Omega^M_{\text{green}}$.

It is important to note the following difference between red and black events on the one side, and green and blue events on the other side. If, for a transcript τ, one elementary event in $\Omega(\tau)$ is black (resp. red), then all elementary events in $\Omega(\tau)$ are black (resp. red), which justifies to define τ to be black (resp. red).

In contrast, if a transcript τ is green, then the elementary events in $\Omega(\tau)$ are either blue or green. This is because blueness of an elementary event $\omega \in \Omega(\tau)$ does not only depend on τ but also on the component k_ω of ω.

We will prove Theorem 7.5 by showing that the probabilities of black, red, and blue elementary events are exponentially small, that the probability of sudden-death events is exponentially small, and that for green transcripts $\tau \in \mathcal{T}^M_{\text{green}}$, the probability that Eve publishes a correct (*initialization vector*, *keystream prefix*) pair is exponentially small (see Lemma 7.1 and the now following Subsection 7.5.7).

To this end, let us have a closer look at the structure of $\Omega(\tau)$ for green transcripts τ. We know that for a green transcript τ, the decision if an elementary event $\omega \in \Omega(\tau)$ is green or blue depends only on k_ω. This justifies the following definition:

Definition 7.18: τ-green/τ-blue Keys

- Let τ denote a green transcript. A τ-consistent key $k \in K(\tau)$ is called *τ-green* if $|(X(\tau) \oplus k) \cap U(\tau)| \leq \Delta$ and $|(\bar{Y}(\tau) \oplus k) \cap V(\tau)| \leq \Delta$, and *$\tau$-blue* otherwise.
- We denote by $K^{\text{green}}(\tau)$ (resp. $K^{\text{blue}}(\tau)$) the set of all τ-consistent keys which are τ-green (resp. τ-blue).

Note that, by definition, for green transcripts τ it holds:

$$K(\tau) = K^{\text{green}}(\tau) \cup K^{\text{blue}}(\tau).$$

7.5.7 Starting with the Proof of Lemma 7.1

Remember that for proving Lemma 7.1, we have to show the following claims:

(i) It holds that $\Pr_\Omega[\Omega^{\text{black}}] \leq 34 \cdot 2^{-n}$.

(ii) It holds that $\Pr_{\Omega}[\Omega^{\text{red}} \cup \Omega^{\text{blue}}] \leq M \cdot e^{-n}$.

(iii) It holds that

$$\Pr_{\Omega}\Big[\Omega^{\text{s.death}} \setminus \Big(\Omega^{\text{black}} \cup \Omega^{\text{red}} \cup \Omega^{\text{blue}}\Big)\Big] \leq 2^{-(n-1)} \cdot (\Delta + 2) \cdot M.$$

(iv) For all $\tau \in \mathcal{T}^{M}_{\text{green}}$, it holds that

$$\Pr_{\Omega^{\text{green}}(\tau)}[\Omega^{\text{succ}}] \leq 11 \cdot (R + 4n) \cdot M \cdot 2^{-(n-1)}.$$

The proofs of parts (i), (ii), (iii), and (iv) will be given in Subsections 7.5.9, 7.5.11, 7.5.8, and 7.5.10, respectively. All these proofs use the following *Smoothness Lemma*, which shows that for all green transcripts τ, there is a sufficiently large number of green τ-consistent keys and that the probabilities of these green keys do not differ too much.

Lemma 7.4: Smoothness Lemma

For all green transcripts τ, the following is true if n is large enough:

(I) $|K^{\text{green}}(\tau)| \geq \frac{1}{\sqrt{2}} \cdot 2^{n}$.

(II) For all $k, k' \in K^{\text{green}}(\tau)$, it holds that

$$\Pr_{\Omega^{\text{green}}(\tau)}[k] \leq \sqrt{2} \cdot \Pr_{\Omega^{\text{green}}(\tau)}[k'].$$

Lemma 7.4 implies the following corollary, which will be an important tool for proving Lemma 7.1.

Corollary 7.3

For all green transcripts τ, the following is true:

(a) For all $k \in \{0,1\}^{n}$, it holds that

$$\Pr_{\Omega^{\text{green}}(\tau)}[k] \leq 2^{-(n-1)}.$$

(b) For all $x, \bar{y} \in \{0,1\}^{n}$, the following holds:

(b.1) If $(x, \bar{y}) \in \mathit{Coll}(\tau)$, then

$$\Pr_{\Omega^{\text{green}}(\tau)}[P_{\omega}(x \oplus k_{\omega}) \oplus k_{\omega} = \bar{y}] = 1.$$

(b.2) If $(x, \bar{y}) \notin \mathit{Coll}(\tau)$ but $x \in X^*(\tau)$ or $\bar{y} \in \bar{Y}^*(\tau)$, then

$$\Pr_{\Omega^{\text{green}}(\tau)}[P_\omega(x \oplus k_\omega) \oplus k_\omega = \bar{y}] = 0.$$

(b.3) If $x \in X(\tau) \setminus X^*(\tau)$ and $\bar{y} \in \bar{Y} \setminus \bar{Y}^*(\tau)$, then

$$\Pr_{\Omega^{\text{green}}(\tau)}[P_\omega(x \oplus k_\omega) \oplus k_\omega = \bar{y}] = 0.$$

(b.4) In all other cases, i.e., if $x \notin X^*(\tau)$ and $\bar{y} \notin \bar{Y}^*(\tau)$ and ($x \notin X(\tau)$ or $\bar{y} \notin \bar{Y}(\tau)$), it holds

$$\Pr_{\Omega^{\text{green}}(\tau)}[P_\omega(x \oplus k_\omega) \oplus k_\omega = \bar{y}] \leq 9 \cdot 2^{-(n-1)}.$$

(c) For all $x, x' \in \{0,1\}^n$, where $x \notin X(\tau)$ and $x' \in X(\tau)$, and all r, $-(R+n) \leq r \leq R+n$, it holds

$$\Pr_{\Omega^{\text{green}}(\tau)}\left[\pi^r(P_\omega(x \oplus k_\omega) \oplus k_\omega) = P_\omega(x' \oplus k_\omega) \oplus k_\omega\right] \leq 11 \cdot 2^{-(n-1)}.$$

Note here that part (a) of Corollary 7.3 follows directly from Lemma 7.4. Parts (b.1), (b.2), and (b.3) follow directly from the definition of τ-consistent keys. Parts (b.4) and (c) will be proved in Subsection 7.5.12.

In the following, we prove part (I) of Lemma 7.4. The proof of part (II) can be found in Subsection 7.5.13.

Proof of Part (I) of Lemma 7.4: We fix some number j, $1 \leq j \leq M$, and some transcript $\tau \in \mathcal{T}^j_{\text{green}}$.

By Lemma 7.3 it holds that

$$|K^{\text{green}}(\tau)| = |K(\tau)| - \left|K^{\text{blue}}(\tau)\right| \geq 2^n - B(M,R,n) - 2 \cdot \Delta \cdot j - \left|K^{\text{blue}}(\tau)\right|.$$

We show that

$$\left|K^{\text{blue}}(\tau)\right| \leq \frac{(R+n) \cdot j^2}{\Delta}.$$

This is because

$$\begin{aligned}\sum_{k \in \{0,1\}^n} |(X(\tau) \oplus k) \cap U(\tau)| &= \sum_{k \in \{0,1\}^n} |\{(x,u) \in X(\tau) \times U(\tau) \mid x \oplus u = k\}| \\ &= |X(\tau) \times U(\tau)| = |X(\tau)| \cdot |U(\tau)|,\end{aligned}$$

which implies that

$$|\{k \in \{0,1\}^n\,;\ |(X(\tau) \oplus k) \cap U(\tau)| > \Delta\}| \leq \frac{|X(\tau)| \cdot |U(\tau)|}{\Delta} \leq \frac{j^2}{\Delta}.$$

In exactly the same way, one can prove that

$$\left|\left\{k \in \{0,1\}^n\,;\ \left|\left(\bar{Y}(\tau) \oplus k\right) \cap V(\tau)\right| > \Delta\right\}\right| \leq \frac{\left|\bar{Y}(\tau)\right| \cdot |V(\tau)|}{\Delta} \leq \frac{(R+n-1) \cdot j^2}{\Delta}.$$

Consequently,

$$|K^{\text{green}}(\tau)| \geq 2^n - B(M,R,n) - 2 \cdot \Delta \cdot j - \frac{(R+n) \cdot j^2}{\Delta} \geq \frac{1}{\sqrt{2}} \cdot 2^n$$

if n is large enough. The last inequation follows from Theorem 7.5, (1). □

7.5.8 The Proof of Part (iii) of Lemma 7.1: Bounding the Probability of Sudden Death

In this subsection, we prove part (iii) of Lemma 7.1, namely that

$$\Pr_{\Omega}\left[\Omega^{\text{s.death}} \setminus \left(\Omega^{\text{black}} \cup \Omega^{\text{red}} \cup \Omega^{\text{blue}}\right)\right] \leq 2^{-(n-1)} \cdot (\Delta+2) \cdot M.$$

Let us denote by $\mathcal{T}^{\text{all}}_{\text{green}} = \bigcup_{j=1}^{M} \mathcal{T}^{j}_{\text{green}}$ the set of all green transcripts which occur with nonzero probability. Note that $\mathcal{T}^{\text{all}}_{\text{green}}$ has the structure of a partially ordered set, where a transcript τ' is *smaller* than τ if τ' is a prefix of τ.

We denote by $\mathcal{T}^{*}_{\text{green}}$ the set of maximal elements in this partially ordered set $\mathcal{T}^{\text{all}}_{\text{green}}$. Observe that

$$\mathcal{T}^{*}_{\text{green}} = \mathcal{T}^{M}_{\text{green}} \cup \mathcal{T}^{\text{strange}}_{\text{green}},$$

where $\mathcal{T}^{\text{strange}}_{\text{green}}$ contains all green transcripts τ' of length smaller than M for which all transcripts τ which contain τ' as a prefix are black or red.

Let us denote by $\tilde{\Omega}$ the set

$$\tilde{\Omega} = \left(\Omega^{\text{s.death}} \setminus \left(\Omega^{\text{black}} \cup \Omega^{\text{red}} \cup \Omega^{\text{blue}}\right)\right) \cup \bigcup_{\tau \in \mathcal{T}^{*}_{\text{green}}} \Omega^{\text{green}}(\tau),$$

where for all j, $1 \leq j \leq M$, and all $\tau \in \mathcal{T}^{j}_{\text{green}}$, it holds

$$\Omega^{\text{green}}(\tau) = \left\{\omega \in \Omega^{j}_{\text{green}} \;\middle|\; \tau_{\omega}^{\leq j} = \tau\right\}.$$

For all $\tau \in \mathcal{T}^{\text{all}}_{\text{green}}$, we denote by $\Omega^{\text{s.death}}_{\text{green}}(\tau)$ the set of all elementary events $\omega \in \Omega^{\text{green}}(\tau)$ for which the next query after τ generates a sudden-death pair w.r.t. ω. Note that $\Omega^{\text{s.death}}_{\text{green}}(\tau) = \emptyset$ if the length of τ is M.

Observe that $\omega \in \Omega^{\text{s.death}} \setminus (\Omega^{\text{black}} \cup \Omega^{\text{red}} \cup \Omega^{\text{blue}})$ if and only if there is some $\tau \in \mathcal{T}^{\text{all}}_{\text{green}}$ such that $\omega \in \Omega^{\text{s.death}}_{\text{green}}(\tau)$. Consequently,

$$\begin{aligned}\Pr_{\Omega}\Big[\Omega^{\text{s.death}} \setminus \Big(\Omega^{\text{black}} \cup \Omega^{\text{red}} \cup \Omega^{\text{blue}}\Big)\Big] &\leq \Pr_{\tilde{\Omega}}\Big[\Omega^{\text{s.death}} \setminus \Big(\Omega^{\text{black}} \cup \Omega^{\text{red}} \cup \Omega^{\text{blue}}\Big)\Big] \\ &= \sum_{\tau \in \mathcal{T}^{\text{all}}_{\text{green}}} \Pr_{\tilde{\Omega}}\Big[\Omega^{\text{s.death}}_{\text{green}}(\tau)\Big].\end{aligned}$$

For all transcripts $\tau \in \mathcal{T}^{*}_{\text{green}}$, we fix a natural number $i(\tau)$, $1 \leq i(\tau) \leq j$, where j denotes the length of τ. We do this in such a way that the sets

$$T(\tau) = \left\{\tau^{\leq i(\tau)}, \tau^{\leq i(\tau)+1}, \ldots, \tau^{\leq j}\right\}$$

form a partition of the set $\mathcal{T}^{\text{all}}_{\text{green}}$ into pairwise disjoints subsets.[5] Note that the sets $T(\tau)$ correspond to prefixes of the transcript τ.

Now define for all transcripts $\tau \in \mathcal{T}^{*}_{\text{green}}$ subsets $A(\tau)$ and $B(\tau)$ of $\tilde{\Omega}$:

$$A(\tau) = \bigcup_{\tilde{\tau} \in T(\tau)} \Omega^{\text{s.death}}_{\text{green}}(\tilde{\tau}),$$

$$B(\tau) = \Omega^{\text{green}}(\tau) \cup A(\tau).$$

Note that the set system $\{B(\tau) \mid \tau \in \mathcal{T}^{*}_{\text{green}}\}$ defines a partition of $\tilde{\Omega}$ into pairwise disjoint subsets, that the set system $\{A(\tau) \mid \tau \in \mathcal{T}^{*}_{\text{green}}\}$ defines a partition of the set $\Omega^{\text{s.death}} \setminus (\Omega^{\text{black}} \cup \Omega^{\text{red}} \cup \Omega^{\text{blue}})$ into pairwise disjoint subsets, and that for all $\tau \in \mathcal{T}^{*}_{\text{green}}$ it holds $A(\tau) \subseteq B(\tau)$. Consequently,

$$\begin{aligned}\Pr_{\tilde{\Omega}}\Big[\Omega^{\text{s.death}} \setminus \Big(\Omega^{\text{black}} \cup \Omega^{\text{red}} \cup \Omega^{\text{blue}}\Big)\Big] &= \sum_{\tau \in \mathcal{T}^{*}_{\text{green}}} \Pr_{\tilde{\Omega}}[A(\tau) \cap B(\tau)] \\ &= \sum_{\tau \in \mathcal{T}^{*}_{\text{green}}} \Pr_{\tilde{\Omega}}[B(\tau)] \cdot \Pr_{B(\tau)}[A(\tau)] \qquad (7.11) \\ &\leq \max_{\tau \in \mathcal{T}^{*}_{\text{green}}} \Pr_{B(\tau)}[A(\tau)].\end{aligned}$$

We fix some arbitrary $\tau \in \mathcal{T}^{*}_{\text{green}}$ and denote by j the length of τ. Note that for all transcripts $\tilde{\tau} \in T(\tau)$, it holds that $\omega \in \Omega^{\text{s.death}}_{\text{green}}(\tilde{\tau})$ if and only if $\omega \in \Omega^{\text{green}}(\tilde{\tau})$ and the key k_ω falls into the set $D(\tilde{\tau})$, which is defined as follows:

$$D(\tilde{\tau}) = \left(X^{*}_{\text{new}}(\tilde{\tau}) \oplus U_{\text{new}}(\tilde{\tau})\right) \setminus \left(X^{*}(\tilde{\tau}) \oplus U(\tilde{\tau})\right),$$

[5] One way of constructing the numbers $i(\tau)$ is as follows. We enumerate the transcripts in $\mathcal{T}^{*}_{\text{green}}$, take the first transcript τ, set $i(\tau) = 1$, and label all transcripts $\tau^{\leq s}$, for $s = i(\tau), \ldots, j$, where j denotes the length of τ. For all other transcripts $\tau \in \mathcal{T}^{*}_{\text{green}}$, define $i(\tau)$ to be the smallest number i for which $\tau^{\leq i}$ has not been labeled so far and label all transcripts in the corresponding set $T(\tau)$.

where $X^*_{\text{new}}(\tilde{\tau})$ and $U_{\text{new}}(\tilde{\tau})$ denote the new sets $X^*(\cdot)$ and $U(\cdot)$ after posing the uniquely determined next query after $\tilde{\tau}$. According to Corollary 7.3, part (a), the probability of this event is bounded by $2^{-(n-1)} \cdot |D(\tilde{\tau})|$.

Now observe that $\bigcup_{\tilde{\tau} \in T(\tau)} D(\tilde{\tau})$ is a subset of $X^*_{\text{new}}(\tau) \oplus U_{\text{new}}(\tau)$ if $j < M$ and of $X^*(\tau) \oplus U(\tau)$ if $j = M$. If $j < M$, then $|X^*_{\text{new}}(\tau)| \leq |X^*(\tau)| + 2 \leq \Delta + 2$, as τ is green, and $|U_{\text{new}}(\tau)| \leq |U(\tau)| + 1 \leq M$. We obtain that

$$\Pr_{B(\tau)}[A(\tau)] \leq 2^{-(n-1)} \cdot |X^*_{\text{new}}(\tau) \oplus U_{\text{new}}(\tau)| \leq 2^{-(n-1)} \cdot (\Delta + 2) \cdot M,$$

which proves part (iii) of Lemma 7.1 by Eq. (7.11). □

7.5.9 The Proof of Part (i) of Lemma 7.1: Bounding the Probability of Black Elementary Events

In this subsection, we prove part (i) of Lemma 7.1, namely that

$$\Pr_{\Omega}\left[\Omega^{\text{black}}\right] \leq 34 \cdot 2^{-n}. \tag{7.12}$$

Proof of Eq. (7.12): From Definition 7.14, it follows straightforwardly that for any elementary event $\omega \in \Omega$, it holds that the transcript τ_ω is black if and only if it has some black prefix (where τ_ω is considered to be its own prefix). This, in turn, implies that $\omega \in \Omega^{\text{black}}$ if and only if τ_ω is black. Consequently, it is sufficient here to assess the probability that for an $\omega \in \Omega$ chosen uniformly and at random (see Definition 7.4), the number of τ_ω-critical keys exceeds $B(M, R, n)$ or it holds $Max(P_{\tau_\omega}, U(\tau_\omega)) > 5$.

Remember from Definition 7.13 that a key $k \in \{0,1\}^n$ is called τ_ω-critical if there is some $u \in U(\tau_\omega)$ such that $x := u \oplus k \in X(\tau_\omega)$ and $y := P_{\tau_\omega}(u) \oplus k \in \bar{Y}(\tau_\omega)$, which implies that for the corresponding triple (u, x, y) it holds that $x \oplus u = y \oplus P_{\tau_\omega}(u)$. Moreover, remember from Theorem 7.4 the definition of $\mu(P, U, X, Y)$ for permutations P over $\{0,1\}^n$ and subsets U, X, Y of $\{0,1\}^n$:

$$\mu(P, U, X, Y) = \left|\{(u, x, y) \in U \times X \times Y \mid x \oplus u = y \oplus P(u)\}\right|.$$

Consequently, $\mu(P_{\tau_\omega}, U(\tau_\omega), X(\tau_\omega), \bar{Y}(\tau_\omega))$ is an upper bound for the number of τ_ω-critical keys.

Theorem 7.4 implies that the probability that for a randomly chosen $\omega \in \Omega$, it holds that

$$\mu\Big(P_{\tau_\omega}, U(\tau_\omega), X(\tau_\omega), \bar{Y}(\tau_\omega)\Big) \geq B(M, R, n),$$

is at most $2 \cdot 2^{-n}$. Here, we took into account that $|U(\tau_\omega)| \leq M$, $|X(\tau_\omega)| \leq M$, and $|\bar{Y}(\tau_\omega)| \leq M \cdot (R + n - 1)$.

So, the probability that for a randomly chosen $\omega \in \Omega$, ω falls into Ω^{black} because the number of τ_ω-critical keys exceeds $B(M, R, n)$, is bounded from above by $2 \cdot 2^{-n}$.

We complete the proof by showing that

$$\Pr_{\Omega}[Max(P_{\tau_\omega}, U(\tau_\omega)) \geq 6] \leq 32 \cdot 2^{-n}.$$

According to Definition 7.12, the event $Max(P_{\tau_\omega}, U(\tau_\omega)) \geq 6$ implies the existence of some $U' \subseteq U(\tau_\omega)$, $|U'| = 6$, such that $u'_1 \oplus P_{\tau_\omega}(u'_1) = u'_2 \oplus P_{\tau_\omega}(u'_2)$ for all $u'_1, u'_2 \in U'$. Given a subset $U' \subseteq U(\tau_\omega)$, $|U'| = 6$, the probability that $u'_1 \oplus P_{\tau_\omega}(u'_1) = u'_2 \oplus P_{\tau_\omega}(u'_2)$ holds for all $u'_1, u'_2 \in U'$, equals

$$\prod_{i=1}^{5} \frac{1}{2^n - i} \leq \left(\frac{1}{1/2 \cdot 2^n}\right)^5 = 2^5 \cdot 2^{-5 \cdot n}.$$

Consequently,

$$\begin{aligned}\Pr_{\Omega}[Max(P_{\tau_\omega}, U(\tau_\omega)) \geq 6] &\leq |U(\tau_\omega)|^6 \cdot 2^5 \cdot 2^{-5 \cdot n} \\ &\leq 2^{6 \cdot (2/3)n} \cdot 2^5 \cdot 2^{-5 \cdot n} = 32 \cdot 2^{-n}.\end{aligned}$$

Here, for the sake of simplicity, we upper bounded the number of subsets with six elements of $U(\tau_\omega)$ by $|U(\tau_\omega)|^6$. $|U(\tau_\omega)|$, in turn, is upper bounded by $2^{(2/3)n}$ as the underlying transcript τ_ω consists of at most $2^{(2/3)n}$ queries. □

7.5.10 The Proof of Part (iv) of Lemma 7.1

Let τ be a green transcript of length M, i.e., $\tau \in \mathcal{T}^M_{\text{green}}$. We have to bound the probability that Eve is successful under the condition that Alice has chosen a green elementary event $\omega = (k_\omega, P_\omega, F_\omega) \in \Omega^{\text{green}}(\tau)$.

Depending on τ, Eve publishes a pair $(x^*(\tau), z^*(\tau)) \in \{0,1\}^n \times \{0,1\}^n$, where $x^*(\tau) \notin X(\tau)$. Eve wins if and only if $z^*(\tau)$ equals the block of the first n keystream bits of the packet generated on input $x^*(\tau)$ under ω, i.e.,

$$z^*(\tau) = F_\omega(P_\omega(x^*(\tau) \oplus k_\omega) \oplus k_\omega).$$

For all $\omega \in \Omega^{\text{green}}(\tau)$, let y_ω denote the value

$$y_\omega = P_\omega(x^*(\tau) \oplus k_\omega) \oplus k_\omega.$$

We have to bound the probability

$$\Pr_{\Omega^{\text{green}}(\tau)}[F_\omega(y_\omega) = z^*(\tau)].$$

We do this by dividing $\Omega^{\text{green}}(\tau)$ into two disjoint subsets *IND* and *DEP*, where *IND* contains all those elementary events $\omega \in \Omega^{\text{green}}(\tau)$ for which $F_\omega(y_\omega)$ is independent from the queries and answers contained in τ, and $DEP = \Omega^{\text{green}}(\tau) \setminus IND$.

Note that $\omega \in DEP$ if and only if

(I) there is some i, $-(n-1) \leq i \leq n-1$, such that $\pi^i(y_\omega) \in Y(\tau)$, or

(II) there is some i, $-(n-1) \leq i \leq n-1$, some $x \in X(\tau)$, and some r, $0 \leq r \leq R-1$, such that $\pi^i(y_\omega) = \pi^r(P_\omega(x \oplus k_\omega) \oplus k_\omega)$.

In case (I), $F_\omega(y_\omega)$ is not independent from the answer of the F-query with input $\pi^i(y_\omega)$; in case (II), $F_\omega(y_\omega)$ is not independent from the answer of the E-query with input x (in particular, from the block starting at position r in packet $E_\omega(x)$).

Corresponding to this, DEP can be written as

$$DEP = DEP_1 \cup DEP_2,$$

where DEP_1 contains all $\omega \in \Omega^{\text{green}}(\tau)$ for which case (I) is fulfilled and DEP_2 contains all $\omega \in \Omega^{\text{green}}(\tau)$ for which case (II) is fulfilled. Note that

$$\begin{aligned}\Pr_{\Omega^{\text{green}}(\tau)}[F_\omega(y_\omega) = z^*(\tau)] &= \Pr_{\Omega^{\text{green}}(\tau)}[DEP] \cdot \Pr_{\Omega^{\text{green}}(\tau)}[F_\omega(y_\omega) = z^*(\tau) \mid DEP] \\ &\quad + \Pr_{\Omega^{\text{green}}(\tau)}[IND] \cdot \Pr_{\Omega^{\text{green}}(\tau)}[F_\omega(y_\omega) = z^*(\tau) \mid IND] \\ &\leq \Pr_{\Omega^{\text{green}}(\tau)}[DEP] + \Pr_{\Omega^{\text{green}}(\tau)}[F_\omega(y_\omega) = z^*(\tau) \mid IND],\end{aligned}$$

i.e.,

$$\begin{aligned}\Pr_{\Omega^{\text{green}}(\tau)}[F_\omega(y_\omega) = z^*(\tau)] \leq \Pr_{\Omega^{\text{green}}(\tau)}[DEP_1] + \Pr_{\Omega^{\text{green}}(\tau)}[DEP_2] \\ + \Pr_{\Omega^{\text{green}}(\tau)}[F_\omega(y_\omega) = z^*(\tau) \mid IND].\end{aligned} \tag{7.13}$$

It is quite obvious that

$$\Pr_{\Omega^{\text{green}}(\tau)}[F_\omega(y_\omega) = z^*(\tau) \mid IND] = 2^{-n}, \tag{7.14}$$

as $\omega \in IND$ implies that $F_\omega(y_\omega)$ can take all values in $\{0,1\}^n$ with the same probability.

Next, observe that for all $\omega \in \Omega^{\text{green}}(\tau)$, it holds that $\omega \in DEP_1$ if and only if

$$y_\omega \in \bigcup_{y \in Y(\tau)} \left\{ \pi^i(y) \;\middle|\; -(n-1) \leq i \leq n-1 \right\},$$

where the set at the right-hand side has size at most $(2n-1) \cdot M$. As $x^*(\tau) \notin X(\tau)$, it follows by Corollary 7.3, part (b), that

$$\Pr_{\Omega^{\text{green}}(\tau)}[DEP_1] \leq (2n-1) \cdot M \cdot 9 \cdot 2^{-(n-1)}. \tag{7.15}$$

Observe further that for all $\omega \in \Omega^{\text{green}}(\tau)$, it holds that $\omega \in DEP_2$ if and only if

$$\pi^i(P_\omega(x^*(\tau) \oplus k_\omega) \oplus k_\omega) = P_\omega(x \oplus k_\omega) \oplus k_\omega$$

for some $x \in X(\tau)$ and some number i, $-(R+n-2) \leq i \leq n-1$. As $x^*(\tau) \notin X(\tau)$, it follows by Corollary 7.3, part (c), that

$$\Pr_{\Omega^{\text{green}}(\tau)}[DEP_2] \leq (R+2n-2) \cdot M \cdot 11 \cdot 2^{-(n-1)}. \tag{7.16}$$

Putting Eqs. (7.13) to (7.16) together yields

$$\begin{aligned}\Pr_{\Omega^{\text{green}}(\tau)}[\Omega^{\text{succ}}] &\leq (2 + (2n-1) \cdot M \cdot 9 + (R+2n-2) \cdot M \cdot 11) \cdot 2^{-(n-1)} \\ &< 11 \cdot (R+4n) \cdot M \cdot 2^{-(n-1)}.\end{aligned}$$

□

7.5.11 The Proof of Part (ii) of Lemma 7.1: Bounding the Probability of Red and Blue Elementary Events

We have to show that

$$\Pr_{\Omega}\left[\Omega^{\text{red}} \cup \Omega^{\text{blue}}\right] \leq M \cdot e^{-n}. \tag{7.17}$$

In the proof, we will use a Chernoff bound argument, which is described in Appendix 7.A.

Proof of Eq. (7.17): Note first that for all $\omega \in \Omega^{\text{red}} \cup \Omega^{\text{blue}}$, there is some j, $1 \leq j \leq M$, such that the j-th query makes ω red or blue. Consequently,

$$\Omega^{\text{red}} \cup \Omega^{\text{blue}} = \bigcup_{j=1}^{M} \Omega^{j-1}_{\text{green}} \cap \left(\Omega^{j}_{\text{red}} \cup \Omega^{j}_{\text{blue}}\right),$$

which implies

$$\begin{aligned}\Pr_{\Omega}\left[\Omega^{\text{red}} \cup \Omega^{\text{blue}}\right] &\leq \sum_{j=1}^{M} \Pr_{\Omega}\left[\Omega^{j-1}_{\text{green}} \cap \left(\Omega^{j}_{\text{red}} \cup \Omega^{j}_{\text{blue}}\right)\right] \\ &= \sum_{j=1}^{M} \Pr_{\Omega}\left[\Omega^{j}_{\text{red}} \cup \Omega^{j}_{\text{blue}} \;\middle|\; \Omega^{j-1}_{\text{green}}\right] \cdot \Pr_{\Omega}\left[\Omega^{j-1}_{\text{green}}\right] \\ &\leq \sum_{j=1}^{M} \Pr_{\Omega}\left[\Omega^{j}_{\text{red}} \cup \Omega^{j}_{\text{blue}} \;\middle|\; \Omega^{j-1}_{\text{green}}\right].\end{aligned}$$

Hence, for proving Eq. (7.17), it is sufficient to show that for all j, $1 \leq j \leq M$, it holds

$$\Pr_{\Omega^{j-1}_{\text{green}}}\left[\Omega^{j}_{\text{red}} \cup \Omega^{j}_{\text{blue}}\right] = \Pr_{\Omega}\left[\Omega^{j}_{\text{red}} \cup \Omega^{j}_{\text{blue}} \;\middle|\; \Omega^{j-1}_{\text{green}}\right] < e^{-n}. \tag{7.18}$$

Note first that Eq. (7.18) is true if $j < \frac{\Delta}{R+n-1}$, as then for all transcripts τ with j queries, it holds that the cardinalities of $X(\tau)$ and $\bar{Y}(\tau)$ are smaller than Δ.

We fix some arbitrary number j, $\frac{\Delta}{R+n-1} \leq j \leq M$. For all J, $1 \leq J \leq j-1$, we define a random variable $DB_J \in \{0,1\}$ over Ω, where $DB_J(\omega) = 1$ if and only if ω is J-alive (see Definition 7.9) and the J-th query along τ_ω increases $(X(\tau_\omega) \oplus k_\omega) \cap U(\tau_\omega)$ or increases $(\bar{Y}(\tau_\omega) \oplus k_\omega) \cap V(\tau_\omega)$ or increases $X^*(\tau_\omega)$. Formally,

$$\begin{aligned} &DB_J(\omega) = 1 \iff \\ &\left|\left(X\left(\tau_\omega^{\leq J}\right) \oplus k_\omega\right) \cap U\left(\tau_\omega^{\leq J}\right)\right| > \left|\left(X\left(\tau_\omega^{\leq J-1}\right) \oplus k_\omega\right) \cap U\left(\tau_\omega^{\leq J-1}\right)\right| \text{ or} \\ &\left|\left(\bar{Y}\left(\tau_\omega^{\leq J}\right) \oplus k_\omega\right) \cap V\left(\tau_\omega^{\leq J}\right)\right| > \left|\left(\bar{Y}\left(\tau_\omega^{\leq J-1}\right) \oplus k_\omega\right) \cap V\left(\tau_\omega^{\leq J-1}\right)\right| \text{ or} \\ &\left|X^*\left(\tau_\omega^{\leq J}\right)\right| > \left|X^*\left(\tau_\omega^{\leq J-1}\right)\right|. \end{aligned}$$

Note that the event $\omega \in \Omega^j_{\text{red}} \cap \Omega^j_{\text{blue}}$ implies the event that

$$\sum_{J=1}^{j-1} DB_J(\omega) \geq \frac{\Delta - (R+n-1)}{R+n-1}. \tag{7.19}$$

This is because each query along τ_ω increases $(X(\tau_\omega) \oplus k_\omega) \cap U(\tau_\omega)$ by at most one, $(\bar{Y}(\tau_\omega) \oplus k_\omega) \cap V(\tau_\omega)$ by at most $R+n-1$, and $X^*(\tau_\omega)$ by at most two. In particular, each E-query can increase $(X(\tau_\omega) \oplus k_\omega) \cap U(\tau_\omega)$ by at most one and $X^*(\tau_\omega)$ by at most two, each P- or P^{-1}-query can increase $(X(\tau_\omega) \oplus k_\omega) \cap U(\tau_\omega)$ and $(\bar{Y}(\tau_\omega) \oplus k_\omega) \cap V(\tau_\omega)$ by at most one, and each F-query can increase $(\bar{Y}(\tau_\omega) \oplus k_\omega) \cap V(\tau_\omega)$ by at most $R+n-1$ and $X^*(\tau_\omega)$ by at most one.

We bound the probability of the event in Eq. (7.19) over $\Omega^{j-1}_{\text{green}}$. We do this by bounding the probability of the event $DB_J(\omega) = 1$ over $\Omega^{j-1}_{\text{green}}$ for all $J = 1, \ldots, j-1$. Let us fix a number J, $1 \leq J \leq j-1$. Note that

$$\Pr_{\Omega^{j-1}_{\text{green}}}[DB_J(\omega) = 1] = \sum_{\tau \in \mathcal{T}^J_{\text{green}}} \Pr_{\Omega^{j-1}_{\text{green}}}[\tau] \cdot \Pr_{\Omega^{j-1}_{\text{green}}(\tau)}[DB_J(\omega) = 1].$$

Note further that for all $\tau \in \mathcal{T}^J_{\text{green}}$ and $\omega \in \Omega^{j-1}_{\text{green}}(\tau)$, it holds that $DB_J(\omega) = 1$ if and only if at least one of the following conditions is fulfilled:

(A) The J-th query in τ is a P-query with input u or a P^{-1}-query with output u and $k_\omega \in u \oplus X(\tau)$.

(B) The J-th query in τ is a P-query with output v or a P^{-1}-query with input v and $k_\omega \in v \oplus \bar{Y}(\tau)$.

(C) The J-th query in τ is an F-query with input y and there is some r, $-(n-1) \leq r \leq R-1$, such that $k_\omega \in \pi^{-r}(y) \oplus V(\tau)$.

(D) The J-th query in τ is an E-query with input x and $k_\omega \in x \oplus U(\tau)$.

(E) The J-th query in τ is an E-query with input x and $P_\omega(x \oplus k_\omega) \oplus k_\omega = \bar{y}$ for some $\bar{y} \in \bar{Y}(\tau)$.

(F) The J-th query in τ is an F-query with input y and $y = \pi^r(P_\omega(x \oplus k_\omega) \oplus k_\omega)$ for some $x \in X(\tau) \setminus X^*(\tau)$ and some number r, $-(n-1) \leq r \leq R-1$.

(G) The J-th query in τ is an E-query with input x and there is some r, $-(R-1) \leq r \leq R-1$, and some $x' \in X(\tau)$ such that $\pi^r(P_\omega(x \oplus k_\omega) \oplus k_\omega) = P_\omega(x' \oplus k_\omega) \oplus k_\omega$.

Note that (A) and (D) are the situations in which query J increases $(X(\tau_\omega) \oplus k_\omega) \cap U(\tau_\omega)$, that (B) and (C) are the situations in which query J increases $(\bar{Y}(\tau_\omega) \oplus k_\omega) \cap V(\tau_\omega)$, that (E) and (F) are the situations in which query J generates a new structural EF-collision (i.e., increases $X^*(\tau_\omega)$ by one), and that (G) is the situation in which query J generates a new structural EE-collision (i.e., increases $X^*(\tau_\omega)$ by one or two).

Note further that conditions (A,D) imply that k_ω belongs to a set of at most $J-1$ elements. Conditions (B,C) imply that k_ω belongs to a set of at most $(R+n-1) \cdot (J-1)$ elements. From Corollary 7.3, part (a), it follows that these events have probability at most $2^{-(n-1)} \cdot (R+n-1) \cdot (J-1)$.

From Corollary 7.3, part (b), it follows that condition (E) has probability at most $9 \cdot |\bar{Y}(\tau_\omega)| \cdot 2^{-(n-1)} \leq 9 \cdot (R+n-1) \cdot (J-1) \cdot 2^{-(n-1)}$ and that condition (F) has probability at most $9 \cdot (R+n-1) \cdot |X(\tau_\omega)| \cdot 2^{-(n-1)} \leq 9 \cdot (R+n-1) \cdot (J-1) \cdot 2^{-(n-1)}$.

From Corollary 7.3, part (c), it follows that condition (G) has probability at most $11 \cdot (2R-1) \cdot |X(\tau_\omega)| \cdot 2^{-(n-1)} \leq 11 \cdot (2R-1) \cdot (J-1) \cdot 2^{-(n-1)}$.

We obtain that for all J, $1 \leq J \leq j-1$,

$$\begin{aligned} \Pr_{\Omega^{j-1}_{\text{green}}}[DB_J(\omega) = 1] &\leq 11 \cdot 2^{-(n-1)} \cdot (2R-1) \cdot (J-1) \\ &\leq 22 \cdot 2^{-(n-1)} \cdot R \cdot (j-1). \end{aligned} \tag{7.20}$$

Eq. (7.20) allows us to apply the Chernoff bound method from Lemma 7.7 in Appendix 7.A with $N = j-1$, $p = 22 \cdot 2^{-(n-1)} \cdot R \cdot (j-1)$, and $d = \sqrt{n/(2 \cdot (j-1))}$ to obtain directly that

$$\Pr_{\Omega^{j-1}_{\text{green}}}\left[\sum_{J=1}^{j-1} DB_J(\omega) > 22 \cdot 2^{-(n-1)} \cdot R \cdot (j-1)^2 + \sqrt{\frac{n \cdot (j-1)}{2}}\right] < e^{-n}. \tag{7.21}$$

Note that item (2) of Theorem 7.5 yields

$$\begin{aligned} &22 \cdot 2^{-(n-1)} \cdot R \cdot (j-1)^2 + \sqrt{\frac{n \cdot (j-1)}{2}} \\ &< 22 \cdot 2^{-(n-1)} \cdot R \cdot M^2 + \sqrt{\frac{n \cdot M}{2}} \\ &\leq \frac{\Delta - (R+n-1)}{R+n-1}. \end{aligned} \tag{7.22}$$

Thus, Eq. (7.21) together with Eq. (7.22) proves Eqs. (7.18) and (7.17), and, consequently, Lemma 7.1, part (ii). □

7.5.12 The Proof of Corollary 7.3, Parts (b.4) and (c)

Let us fix an arbitrary number j, $1 \leq j \leq M$, and a green transcript $\tau \in \mathcal{T}^j_{\text{green}}$. We assume that part (a) of Corollary 7.3 holds, i.e., that for all $k \in K^{\text{green}}(\tau)$, we have

$$\Pr_{\Omega^{\text{green}}(\tau)}[k] \leq 2^{-(n-1)}. \tag{7.23}$$

Let us first prove part (b.4) of Corollary 7.3. We fix some $x, \bar{y} \in \{0,1\}^n$, where $x \notin X^*(\tau)$ and $\bar{y} \notin \bar{Y}^*(\tau)$ and ($x \notin X(\tau)$ or $\bar{y} \notin \bar{Y}(\tau)$). We have to show

$$\Pr_{\Omega^{\text{green}}(\tau)}[P_\omega(x \oplus k_\omega) \oplus k_\omega = \bar{y}] \leq 9 \cdot 2^{-(n-1)}. \tag{7.24}$$

Proof of Eq. (7.24): We divide $\Omega^{\text{green}}(\tau)$ into two subsets Ω_1 and Ω_2, where

$$\begin{aligned} \Omega_1 &= \{\omega \in \Omega^{\text{green}}(\tau) \mid x \oplus k_\omega \notin U(\tau)\}, \\ \Omega_2 &= \{\omega \in \Omega^{\text{green}}(\tau) \mid x \oplus k_\omega \in U(\tau)\}, \end{aligned} \tag{7.25}$$

and denote

$$\begin{aligned} K_1 &= \{k \in K^{\text{green}}(\tau) \mid x \oplus k \notin U(\tau)\}, \\ K_2 &= \{k \in K^{\text{green}}(\tau) \mid x \oplus k \in U(\tau)\}. \end{aligned} \tag{7.26}$$

The sets Ω_2 and K_2 define another set $W \subseteq \{0,1\}^n$ by

$$W = \{P_\omega(x \oplus k_\omega) \oplus k_\omega \mid \omega \in \Omega_2\} = \{P_\tau(x \oplus k) \oplus k \mid k \in K_2\}.$$

Here, P_τ denotes the restriction of P_ω to $U(\tau)$, which, by definition, is the same for all $\omega \in \Omega(\tau)$. Note that $|W| \leq |K_2| \leq |U(\tau)| \leq j \leq M$.

Let us now define an equivalence relation on K_2. For keys $k, k' \in K_2$, we define that $k \equiv k'$ if $P_\tau(x \oplus k) \oplus k = P_\tau(x \oplus k') \oplus k'$. Let $L_1, \ldots, L_s$ denote the equivalence classes corresponding to the equivalence relation $\equiv$ on K_2. Clearly, $s = |W|$ and for each class L_l, $1 \leq l \leq s$, there is exactly one $w \in W$ such that $P_\tau(x \oplus k) \oplus k = w$ for all $k \in L_l$.

Note that $k \equiv k'$ implies that $x \oplus k \equiv_{P_\tau} x \oplus k'$ in the sense of Definition 7.12 and remember that, as τ is not black, $Max(P_\tau, U(\tau)) \leq 5$. This implies:

Lemma 7.5

For all $w \in W$, the number of keys $k \in K_2$ for which $P_\tau(x \oplus k) \oplus k = w$ is at most five. □

Note that

$$\begin{aligned}
&\Pr_{\Omega^{\mathrm{green}}(\tau)}[P_\omega(x \oplus k_\omega) \oplus k_\omega = \bar{y}] \\
&= \Pr_{\Omega^{\mathrm{green}}(\tau)}[\Omega_1] \cdot \Pr_{\Omega^{\mathrm{green}}(\tau)}[P_\omega(x \oplus k_\omega) \oplus k_\omega = \bar{y} \mid \Omega_1] \\
&\quad + \Pr_{\Omega^{\mathrm{green}}(\tau)}[\Omega_2] \cdot \Pr_{\Omega^{\mathrm{green}}(\tau)}[P_\omega(x \oplus k_\omega) \oplus k_\omega = \bar{y} \mid \Omega_2],
\end{aligned}$$

i.e.,

$$\begin{aligned}
&\Pr_{\Omega^{\mathrm{green}}(\tau)}[P_\omega(x \oplus k_\omega) \oplus k_\omega = \bar{y}] \\
&\leq \Pr_{\Omega^{\mathrm{green}}(\tau)}[P_\omega(x \oplus k_\omega) \oplus k_\omega = \bar{y} \mid \Omega_1] \\
&\quad + \Pr_{\Omega^{\mathrm{green}}(\tau)}[\Omega_2] \cdot \Pr_{\Omega^{\mathrm{green}}(\tau)}[P_\omega(x \oplus k_\omega) \oplus k_\omega = \bar{y} \mid \Omega_2].
\end{aligned} \tag{7.27}$$

For estimating $\Pr_{\Omega^{\mathrm{green}}(\tau)}[P_\omega(x \oplus k_\omega) \oplus k_\omega = \bar{y} \mid \Omega_1]$, note that if $x \oplus k_\omega \notin U(\tau)$ and $x \in X(\tau)$, then $P_\omega(x \oplus k_\omega)$ takes all values in $\{0,1\}^n$ which are outside of $V(\tau)$ and which are outside of $\bar{Y}(\tau) \oplus k_\omega$ with the same probability (see the proof of Lemma 7.2).[6] If $x \notin X(\tau)$, then $P_\omega(x \oplus k_\omega)$ takes all values in $\{0,1\}^n$ which are outside of $V(\tau)$ with the same probability (see the proof of Lemma 7.2). This implies that

$$\Pr_{\Omega^{\mathrm{green}}(\tau)}[P_\omega(x \oplus k_\omega) \oplus k_\omega = \bar{y} \mid \Omega_1] \leq \frac{1}{2^n - (R+n)M} \leq 2^{-(n-1)} \tag{7.28}$$

if n is large enough.

Observe next that by Eq. (7.23) it holds that

$$\Pr_{\Omega^{\mathrm{green}}(\tau)}[\Omega_2] \leq 2^{-(n-1)} \cdot |K_2|\,. \tag{7.29}$$

For estimating $\Pr_{\Omega^{\mathrm{green}}(\tau)}[P_\omega(x \oplus k_\omega) \oplus k_\omega = \bar{y} \mid \Omega_2]$, we first consider the case that $\bar{y} \notin W$. Then, by the definition of W, it holds that

$$\Pr_{\Omega^{\mathrm{green}}(\tau)}[P_\omega(x \oplus k_\omega) \oplus k_\omega = \bar{y} \mid \Omega_2] = 0.$$

Assume now that $\bar{y} \in W$. From Lemma 7.5 and the Smoothness Lemma (Lemma 7.4) it follows that

$$\Pr_{\Omega^{\mathrm{green}}(\tau)}[P_\omega(x \oplus k_\omega) \oplus k_\omega = \bar{y} \mid \Omega_2] \leq \sqrt{2} \cdot \frac{5}{|K_2|} < \frac{8}{|K_2|}. \tag{7.30}$$

[6] Here, $P_\omega(x \oplus k_\omega) \in \bar{Y}(\tau) \oplus k_\omega$ would imply that $x \in X^*(\tau)$, which contradicts the assumption.

Inserting Eqs. (7.28) to (7.30) into Eq. (7.27) yields

$$\Pr_{\Omega^{\text{green}}(\tau)}[P_\omega(x \oplus k_\omega) \oplus k_\omega = \bar{y}] \leq 2^{-(n-1)} + 2^{-(n-1)} \cdot |K_2| \cdot \frac{8}{|K_2|} = 9 \cdot 2^{-(n-1)}. \qquad \square$$

Let us now prove part (c) of Corollary 7.3. We fix some $x \neq x' \in \{0,1\}^n$, where $x \notin X(\tau)$ and $x' \in X(\tau)$, and some number i, $-(R+n) \leq i \leq R+n$. We have to show

$$\Pr_{\Omega^{\text{green}}(\tau)}\big[Ev(x,x',i)\big] \leq 11 \cdot 2^{-(n-1)}, \tag{7.31}$$

where the event $Ev(x,x',i) \subseteq \Omega^{\text{green}}(\tau)$ is defined as

$$Ev(x,x',i) = \left\{\omega \in \Omega^{\text{green}}(\tau) \;\middle|\; \pi^i(P_\omega(x \oplus k_\omega) \oplus k_\omega) = P_\omega(x' \oplus k_\omega) \oplus k_\omega\right\}.$$

Proof of Eq. (7.31): Let us first handle the case that $x' \in X^*(\tau)$ and denote by y' the unique value for which $(x',y') \in \mathit{Coll}(\tau)$. Then, by the definition of structural EF-collisions, it holds that

$$P_\omega(x' \oplus k_\omega) \oplus k_\omega = y'$$

for all $\omega \in \Omega^{\text{green}}(\tau)$.

Consequently, for all $\omega \in \Omega^{\text{green}}(\tau)$ it holds that $\omega \in Ev(x,x',i)$ if and only if

$$P_\omega(x \oplus k_\omega) \oplus k_\omega = \pi^{-i}(y').$$

From part (b) of Corollary 7.3 it follows that if $x' \in X^*(\tau)$, then

$$\Pr_{\Omega^{\text{green}}(\tau)}\big[Ev(x,x',i)\big] \leq 9 \cdot 2^{-(n-1)} < 11 \cdot 2^{-(n-1)}.$$

Now let us consider the case that $x' \in X(\tau) \setminus X^*(\tau)$. For arbitrary points $z \in \{0,1\}^n$, we define

$$\Omega^{\text{green}}(\tau,z) = \{\omega \in \Omega^{\text{green}}(\tau) \mid P_\omega(x \oplus k_\omega) \oplus k_\omega = z\},$$
$$K^{\text{green}}(\tau,z) = \{k \in \{0,1\}^n \mid \exists \omega \in \Omega^{\text{green}}(\tau,z) : k_\omega = k\}.$$

Moreover, for $b \in \{1,2\}$, we define

$$\Omega_b(z) = \Omega^{\text{green}}(\tau,z) \cap \Omega_b,$$
$$K_b(z) = K^{\text{green}}(\tau,z) \cap K_b,$$

where the sets Ω_1 and Ω_2 and the sets K_1 and K_2 are defined as in Eqs. (7.25) and (7.26).

Let us clarify how the keys of elementary events in $\Omega_1(z)$ and $\Omega_2(z)$ and the keys in $K_1(z)$ and $K_2(z)$ look like. It can be easily checked that for all $\omega = (k_\omega, P_\omega, F_\omega) \in \Omega_1$, it

holds $\omega \in \Omega_1(z)$ if and only if $z \oplus k_\omega \notin V(\tau)$ and $P_\omega(x \oplus k_\omega) = z \oplus k_\omega$. Moreover, for all $\omega = (k_\omega, P_\omega, F_\omega) \in \Omega_2$, it holds $\omega \in \Omega_2(z)$ if and only if $P_\tau(x \oplus k_\omega) \oplus k_\omega = z$, which implies by Lemma 7.5 that

$$|K_2(z)| \leq 5. \tag{7.32}$$

We obtain that

$$\begin{aligned} |K^{\text{green}}(\tau, z)| \geq |K_1(z)| &\geq |K_1| - |V(\tau)| \\ &= |K^{\text{green}}(\tau)| - |K_2| - |V(\tau)| \\ &\geq |K^{\text{green}}(\tau)| - 2 \cdot |V(\tau)| \\ &\geq |K^{\text{green}}(\tau)| - 2M \geq \frac{1}{\sqrt{2}} \cdot 2^n \end{aligned} \tag{7.33}$$

if n is large enough. The last inequation follows from $M < 2^{(2/3)n}$ and the proof of the Smoothness Lemma (Lemma 7.4): It is a straightforward consequence of Eqs. (7.42) and (7.43) in Subsection 7.5.13 that for all green transcripts τ and all constants $\delta < 1$, it holds that $|K^{\text{green}}(\tau)| \geq \delta \cdot 2^n$ if n is large enough. Lemma 7.4 states this just for $\delta = \frac{1}{\sqrt{2}}$.

By exactly the same arguments as in the Smoothness Lemma (Lemma 7.4), one can show that for all $k, k' \in K^{\text{green}}(\tau, z)$, it holds that

$$\Pr_{\Omega^{\text{green}}(\tau,z)}[k] \leq \sqrt{2} \cdot \Pr_{\Omega^{\text{green}}(\tau,z)}[k'] \tag{7.34}$$

if n is large enough.

Eqs. (7.33) and (7.34) imply directly that for all $k \in K^{\text{green}}(\tau, z)$, it holds that

$$\Pr_{\Omega^{\text{green}}(\tau,z)}[k] \leq 2^{-(n-1)} \tag{7.35}$$

if n is large enough.

Note that

$$\begin{aligned} &\Pr_{\Omega^{\text{green}}(\tau)}\big[Ev(x, x', i)\big] \\ &= \sum_{z \in \{0,1\}^n} \Pr_{\Omega^{\text{green}}(\tau)}[P_\omega(x \oplus k_\omega) \oplus k_\omega = z] \cdot \Pr_{\Omega^{\text{green}}(\tau,z)}\big[Ev(x, x', i)\big]. \end{aligned} \tag{7.36}$$

For deriving an upper bound for $\Pr_{\Omega^{\text{green}}(\tau,z)}[Ev(x, x', i)]$, we write

$$\begin{aligned} \Pr_{\Omega^{\text{green}}(\tau,z)}\big[Ev(x, x', i)\big] &= \Pr_{\Omega^{\text{green}}(\tau,z)}[\Omega_1(z)] \cdot \Pr_{\Omega_1(z)}\big[Ev(x, x', i)\big] \\ &\quad + \Pr_{\Omega^{\text{green}}(\tau,z)}[\Omega_2(z)] \cdot \Pr_{\Omega_2(z)}\big[Ev(x, x', i)\big] \\ &\leq \Pr_{\Omega_1(z)}\big[Ev(x, x', i)\big] + \Pr_{\Omega^{\text{green}}(\tau,z)}[\Omega_2(z)] \\ &\leq \Pr_{\Omega_1(z)}\big[Ev(x, x', i)\big] + 5 \cdot 2^{-(n-1)}, \end{aligned} \tag{7.37}$$

where the last inequation follows from Eqs. (7.32) and (7.35).

We write $K_1(z)$ as

$$K_1(z) = K_3(z) \cup K_4(z) \cup K_5(z),$$

where

- $K_3(z) = \{k \in K_1(z) \mid x' \oplus k \in U(\tau), P_\tau(x' \oplus k) \oplus k = \pi^i(z)\}$,
- $K_4(z) = \{k \in K_1(z) \mid x' \oplus k \in U(\tau), P_\tau(x' \oplus k) \oplus k \neq \pi^i(z)\}$,
- $K_5(z) = \{k \in K_1(z) \mid x' \oplus k \notin U(\tau)\}$.

From Lemma 7.5, we know that $|K_3(z)| \leq 5$.[7] By exactly the same arguments as in the Smoothness Lemma (Lemma 7.4), one can show that for all $k, k' \in K_1(z)$, it holds

$$\Pr_{\Omega_1(z)}[k] \leq \sqrt{2} \cdot \Pr_{\Omega_1(z)}[k'] \tag{7.38}$$

if n is large enough. Eqs. (7.33) and (7.38) imply directly that for all $k \in K_1(z)$, it holds

$$\Pr_{\Omega_1(z)}[k] \leq 2^{-(n-1)} \tag{7.39}$$

if n is large enough. From $|K_3(z)| \leq 5$ and Eq. (7.39) it now follows that

$$\Pr_{\Omega_1(z)}\big[Ev(x, x', i) \cap (k_\omega \in K_3(z))\big] \leq 5 \cdot 2^{-(n-1)}.$$

Furthermore, w.r.t. $K_4(z)$, it obviously holds that

$$\Pr_{\Omega_1(z)}\big[Ev(x, x', i) \cap (k_\omega \in K_4(z))\big] = 0.$$

Consequently,

$$\begin{aligned} \Pr_{\Omega_1(z)}\big[Ev(x, x', i)\big] &\leq \Pr_{\Omega_1(z)}[\Omega_5(z)] \cdot \Pr_{\Omega_5(z)}\big[Ev(x, x', i)\big] + 5 \cdot 2^{-(n-1)} \\ &\leq \Pr_{\Omega_5(z)}\big[Ev(x, x', i)\big] + 5 \cdot 2^{-(n-1)}, \end{aligned} \tag{7.40}$$

where $\Omega_5(z) = \{\omega \in \Omega_1(z) \mid k_\omega \in K_5(z)\}$.

Note that for all $\omega \in \Omega_5(z)$, the condition that $\omega \in Ev(x, x', i)$ is equivalent to

$$P_\omega(x' \oplus k_\omega) = \pi^i(z) \oplus k_\omega,$$

[7] Lemma 7.5 refers to the keys in K_2, whereas we now refer to the keys in $K_1(z) \subseteq K_1$. However, note that the corresponding equivalence classes of keys are formed based on x for Lemma 7.5, but based on x' here. Hence, as $k \in K_3(z)$ implies $x' \oplus k \in U(\tau)$, we can use the same arguments as for Lemma 7.5 in order to conclude $|K_3(z)| \leq 5$.

which has probability 0 if $\pi^i(z) \oplus k_\omega \in V(\tau)$ and probability at most

$$\frac{1}{2^n - (|V(\tau)| + 1) - \left|\bar{Y}(\tau)\right|} \leq 2^{-(n-1)}$$

if $\pi^i(z) \oplus k_\omega \notin V(\tau)$ and n is large enough; see Eq. (7.28) and the comment before Eq. (7.28) and note that, here, $P_\omega(x' \oplus k_\omega)$ is not only disallowed to take the values in $V(\tau)$, but also additionally disallowed to take the value $z \oplus k_\omega = P_\omega(x \oplus k_\omega) \notin V(\tau)$ (as $x \neq x'$). We obtain that

$$\Pr_{\Omega_5(z)}\big[Ev(x, x', i)\big] \leq 2^{-(n-1)} \tag{7.41}$$

if n is large enough.

Putting Eqs. (7.37), (7.40), and (7.41) together, we obtain that for all $z \in \{0,1\}^n$, it holds

$$\Pr_{\Omega^{\text{green}}(\tau,z)}\big[Ev(x, x', i)\big] \leq 11 \cdot 2^{-(n-1)},$$

which implies by Eq. (7.36) that

$$\Pr_{\Omega^{\text{green}}(\tau)}\big[Ev(x, x', i)\big] \leq 11 \cdot 2^{-(n-1)}. \qquad \square$$

7.5.13 The Proof of the Smoothness Lemma (Lemma 7.4), Part (II)

We fix an arbitrary number j, $1 \leq j \leq M$, and a green transcript $\tau \in \mathcal{T}^j_{\text{green}}$. We analyze the probability distribution $\Pr_{\Omega(\tau)}$ on $K^{\text{green}}(\tau)$ by showing that for all $k \in K^{\text{green}}(\tau)$, it holds that this distribution is close to the uniform distribution on $K^{\text{green}}(\tau)$. More precisely,

$$\begin{aligned} &\Pr_{\Omega(\tau)}[k] \leq \delta \cdot |K^{\text{green}}(\tau)|^{-1} \\ &\text{for } \delta = \left(\frac{2^{n-1}}{2^{n-1} - (R+n)j}\right)^{2\Delta}. \end{aligned} \tag{7.42}$$

Note that part (3) of Theorem 7.5 implies $\delta \leq \sqrt{2}$. This is because for $\theta = (R+n)j$ and $N = 2^{n-1}$, we can write δ as

$$\delta = \left(\frac{N}{N-\theta}\right)^{2\Delta} = \left(\frac{1}{1-\theta/N}\right)^{2\Delta} = \left(\left(\frac{1}{1-\theta/N}\right)^{N/\theta}\right)^{\frac{2\Delta\theta}{N}} \approx e^{\frac{2\Delta\theta}{N}}. \tag{7.43}$$

Part (3) of Theorem 7.5 implies that $\Delta \cdot \theta \leq \frac{\ln(2) \cdot N}{4}$, which is equivalent to

$$\frac{2\Delta\theta}{N} \leq \frac{\ln(2)}{2}.$$

Proof of Eq. (7.42): The proof of Lemma 7.2 shows how, for keys $k \in K(\tau)$, completions P' of P_τ on $\{0,1\}^n \setminus U(\tau)$ and F' of F_τ on $\{0,1\}^n \setminus Y(\tau)$ have to be constructed such that (k, P', F') belongs to $\Omega(\tau)$. In particular:

(1) The function values of P' on $X^*(\tau) \oplus k$, a set of size $|X^*(\tau)|$, are determined.

(2) The function values of P' on the set $((X(\tau) \setminus X^*(\tau)) \oplus k) \setminus U(\tau)$ are forbidden to fall into the set $\bar{Y}(\tau) \oplus k$.

(3) We assume that P' is an arbitrarily fixed completion of P_τ satisfying (1) and (2) and describe how a completion F' of F_τ has to be constructed in such a way that (k, P', F') belongs to $\Omega(\tau)$. The function values $F'(y)$ are determined or partly determined on a set $Z(k) = Z_1(k) \cup Z_2(k)$, where $Z_1(k)$ contains all those $y \in \{0,1\}^n$ for which there is some r, $-(n-1) \leq r \leq R-1$, such that $\pi^{-r}(y) = P'(x \oplus k) \oplus k$ for some $x \in X(\tau) \setminus X^*(\tau)$, and $Z_2(k)$ contains all those $y \in \{0,1\}^n$ for which there is some i, $-(n-1) \leq i \leq n-1$, such that $\pi^i(y) \in Y(\tau)$. Note that $Z_1(k) \cap Z_2(k) = \emptyset$ and $|Z_1(k)| = |Z_1(k')|$ and $|Z_2(k)| = |Z_2(k')|$ for all $k, k' \in K^{\text{green}}(\tau)$. This implies that $Z(k)$ has the same size for all $k \in K^{\text{green}}(\tau)$.

We use the following fact. Let A_1, B_1, A_2, B_2 be finite sets fulfilling $A_1 \cap A_2 = B_1 \cap B_2 = \emptyset$, $|A_1 \cup A_2| = |B_1 \cup B_2|$, $|A_2| < |B_1|$, and $|B_2| < |A_1|$. Then the number of bijective mappings $f : A_1 \cup A_2 \longrightarrow B_1 \cup B_2$ for which $f(A_2) \subseteq B_1$ is

$$\frac{|A_1|! \cdot |B_1|!}{(|A_1| - |B_2|)!} = \frac{|A_1|! \cdot |B_1|!}{(|B_1| - |A_2|)!}. \tag{7.44}$$

In the following, A_1 corresponds to the set $\{0,1\}^n \setminus (U(\tau) \cup (X(\tau) \oplus k))$, A_2 to the set $((X(\tau) \setminus X^*(\tau)) \oplus k) \setminus U(\tau)$, B_1 to the set $\{0,1\}^n \setminus (V(\tau) \cup (\bar{Y}(\tau) \oplus k))$, and B_2 to the set $((\bar{Y}(\tau) \setminus \bar{Y}^*(\tau)) \oplus k) \setminus V(\tau)$. We denote

- $T = 2^n - |X(\tau)| - |U(\tau)|$,
- $t = |X(\tau) \setminus X^*(\tau)|$,
- $S = 2^n - |\bar{Y}(\tau)| - |V(\tau)|$,
- $s = |\bar{Y}(\tau) \setminus \bar{Y}^*(\tau)|$.

It holds that $S > 2^{n-1}$ and $T > 2^{n-1}$ if n is large enough.

Note that for proving Lemma 7.4, it is sufficient to show that for all pairs $k, k' \in K^{\text{green}}(\tau)$, it holds

$$\frac{\Pr_{\Omega(\tau)}[k]}{\Pr_{\Omega(\tau)}[k']} \leq \delta.$$

This is because τ is green, which, together with $\Pr_{\Omega^{\text{blue}}(\tau)}[k] = \Pr_{\Omega^{\text{blue}}(\tau)}[k'] = 0$ for $k, k' \in K^{\text{green}}(\tau)$, implies

$$\frac{\Pr_{\Omega(\tau)}[k]}{\Pr_{\Omega(\tau)}[k']} = \frac{\Pr_{\Omega(\tau)}[\Omega^{\text{green}}(\tau)] \cdot \Pr_{\Omega^{\text{green}}(\tau)}[k] + \Pr_{\Omega(\tau)}\left[\Omega^{\text{blue}}(\tau)\right] \cdot \Pr_{\Omega^{\text{blue}}(\tau)}[k]}{\Pr_{\Omega(\tau)}[\Omega^{\text{green}}(\tau)] \cdot \Pr_{\Omega^{\text{green}}(\tau)}[k'] + \Pr_{\Omega(\tau)}[\Omega^{\text{blue}}(\tau)] \cdot \Pr_{\Omega^{\text{blue}}(\tau)}[k']} = \frac{\Pr_{\Omega^{\text{green}}(\tau)}[k]}{\Pr_{\Omega^{\text{green}}(\tau)}[k']}.$$

Let us denote by $\mathit{ConsP}_\tau(k)$ the set of all τ-consistent completions P' of P_τ on $\{0,1\}^n \setminus U(\tau)$, i.e., of all completions P' of P_τ for which there is some completion F' of F_τ on $\{0,1\}^n \setminus Y(\tau)$ such that $(k, P', F') \in \Omega(\tau)$. The above statement (3) implies that for all $k \in K(\tau)$ and completions $P' \in \mathit{ConsP}_\tau(k)$, the number of such completions F' is the same, i.e., does not depend on k. This implies that

$$\frac{\Pr_{\Omega(\tau)}[k]}{\Pr_{\Omega(\tau)}[k']} = \frac{|\mathit{ConsP}_\tau(k)|}{|\mathit{ConsP}_\tau(k')|}.$$

Note that due to requirement (2) (see above), the size of $\mathit{ConsP}_\tau(k)$ depends on the sizes of the sets $((X(\tau) \setminus X^*(\tau)) \oplus k) \setminus U(\tau)$ and $((\bar{Y}(\tau) \setminus \bar{Y}^*(\tau)) \oplus k) \setminus V(\tau)$. As k is not blue, the sizes of the corresponding intersections $((X(\tau) \setminus X^*(\tau)) \oplus k) \cap U(\tau)$ and $((\bar{Y}(\tau) \setminus \bar{Y}^*(\tau)) \oplus k) \cap V(\tau)$ can vary only between 0 and Δ (see Definition 7.18).

Thus, for $k \in K^{\text{green}}(\tau)$, the value $|\mathit{ConsP}_\tau(k)|$ is minimal if

$$|((X(\tau) \setminus X^*(\tau)) \oplus k) \cap U(\tau)| = \left|\left(\left(\bar{Y}(\tau) \setminus \bar{Y}^*(\tau)\right) \oplus k\right) \cap V(\tau)\right| = 0.$$

By Eq. (7.44) this implies

$$|\mathit{ConsP}_\tau(k)| = \frac{S! \cdot T!}{(T-s)!} = \frac{S! \cdot T!}{(S-t)!}. \tag{7.45}$$

The value $|\mathit{ConsP}_\tau(k)|$ is maximal if

$$\begin{aligned} |((X(\tau) \setminus X^*(\tau)) \oplus k) \cap U(\tau)| &= \min\{\Delta, t\} \text{ and} \\ \left|\left(\left(\bar{Y}(\tau) \setminus \bar{Y}^*(\tau)\right) \oplus k\right) \cap V(\tau)\right| &= \min\{\Delta, s\}. \end{aligned} \tag{7.46}$$

We now have to distinguish three cases corresponding to whether $|X(\tau) \setminus X^*(\tau)| > \Delta$ or not and whether $|\bar{Y}(\tau) \setminus \bar{Y}^*(\tau)| > \Delta$ or not.

Case 1: $|X(\tau) \setminus X^*(\tau)| > \Delta$ and $|\bar{Y}(\tau) \setminus \bar{Y}^*(\tau)| > \Delta$.

In this case, it follows from Eqs. (7.44) and (7.46) that

$$|\mathit{ConsP}_\tau(k)| \leq \frac{(S+\Delta)! \cdot (T+\Delta)!}{(S+\Delta-(t-\Delta))!} = \frac{(S+\Delta)! \cdot (T+\Delta)!}{(S-t+2\Delta)!}. \tag{7.47}$$

Eqs. (7.45) and (7.47) imply that the $\Pr_{\Omega(\tau)}$-values of elements from $K^{\text{green}}(\tau)$ can differ by a factor δ which is at most $\delta_1 \cdot \delta_2$, where

$$\delta_1 = \frac{(T+1)(T+2)\cdots(T+\Delta)}{(S-t+1)(S-t+2)\cdots(S-t+\Delta)} \leq \left(\frac{T}{S-t}\right)^{\Delta} \tag{7.48}$$

and

$$\begin{aligned}\delta_2 &= \frac{(S+1)(S+2)\cdots(S+\Delta)}{(S-t+\Delta+1)(S-t+\Delta+2)\cdots(S-t+2\Delta)} \\ &\leq \left(\frac{S}{S-t+\Delta}\right)^{\Delta} = \left(\frac{S}{S-(t-\Delta)}\right)^{\Delta} \leq \left(\frac{S}{S-j}\right)^{\Delta} \leq \left(\frac{2^{n-1}}{2^{n-1}-j}\right)^{\Delta}.\end{aligned}$$

Here we used the fact that from $a \geq b$ it follows that $\frac{a}{b} \geq \frac{a+1}{b+1}$.

For upper bounding δ_1, we have to distinguish the two cases $S \geq T$ and $S < T$. If $S \geq T$, then, by Eq. (7.48),

$$\delta_1 \leq \left(\frac{S}{S-t}\right)^{\Delta} \leq \left(\frac{2^{n-1}}{2^{n-1}-j}\right)^{\Delta}. \tag{7.49}$$

If $S < T$, then observe that $S \geq T-(R+n-1)j$. This holds because $|\bar{Y}(\tau)| \leq (R+n-1)j$. Consequently,

$$\begin{aligned}\delta_1 \leq \left(\frac{T}{T-t-(R+n-1)j}\right)^{\Delta} &\leq \left(\frac{T}{T-j-(R+n-1)j}\right)^{\Delta} \\ &= \left(\frac{T}{T-(R+n)j}\right)^{\Delta} \leq \left(\frac{2^{n-1}}{2^{n-1}-(R+n)j}\right)^{\Delta}.\end{aligned} \tag{7.50}$$

Case 2: $|X(\tau) \setminus X^*(\tau)| \leq \Delta$, i.e., $t \leq \Delta$.

Here, $|ConsP_\tau(k)|$ is maximal if $(X(\tau) \setminus X^*(\tau)) \oplus k$ is a subset of $U(\tau)$, which implies $|ConsP_\tau(k)| = (T+t)!$ and that, by Eq. (7.45), the $|ConsP_\tau(k)|$-values for $k \in K^{\text{green}}(\tau)$ cannot differ by a factor larger than

$$\begin{aligned}\frac{(T+t)! \cdot (S-t)!}{T! \cdot S!} &= \frac{(T+1) \cdot \ldots \cdot (T+t)}{(S-t+1) \cdot \ldots \cdot S} \\ &\leq \left(\frac{T}{S-t}\right)^{t} \leq \left(\frac{T}{S-t}\right)^{\Delta} \leq \left(\frac{2^{n-1}}{2^{n-1}-(R+n)j}\right)^{\Delta}.\end{aligned}$$

Note that the last inequation follows from the same case distinction (i.e., $S \geq T$ and $S < T$) that was already performed as part of *Case 1* above (see Eqs. (7.49) and (7.50)).

Case 3: $|\bar{Y}(\tau) \setminus \bar{Y}^*(\tau)| \leq \Delta$, i.e., $s \leq \Delta$.

Here, $|ConsP_\tau(k)|$ is maximal if $(\bar{Y}(\tau) \setminus \bar{Y}^*(\tau)) \oplus k$ is a subset of $V(\tau)$, which implies $|ConsP_\tau(k)| = (S+s)!$ and that, by Eq. (7.45), the $|ConsP_\tau(k)|$-values for $k \in K^{\text{green}}(\tau)$ cannot differ by a factor larger than

$$\frac{(S+s)! \cdot (T-s)!}{T! \cdot S!} = \frac{(S+1) \cdot \ldots \cdot (S+s)}{(T-s+1) \cdot \ldots \cdot T} \leq \left(\frac{S}{T-s}\right)^s \leq \left(\frac{S}{T-s}\right)^\Delta \leq \left(\frac{S}{S-j-s}\right)^\Delta$$
$$\leq \left(\frac{S}{S-j-(R+n-1)j}\right)^\Delta = \left(\frac{S}{S-(R+n)j}\right)^\Delta \leq \left(\frac{2^{n-1}}{2^{n-1}-(R+n)j}\right)^\Delta.$$

Summarizing all three cases, we obtain that

$$\delta \leq \left(\frac{2^{n-1}}{2^{n-1}-(R+n)j}\right)^{2\Delta}. \qquad \square$$

Note: This concludes the proof of Lemma 7.4, thereby completing the proof of Lemma 7.1 and, in turn, also the proof of our main result (Theorem 7.5).

7.6 Conclusion and Outlook

In this chapter, we introduced, for the first time, a random oracle model for KSG-based stream ciphers and proved a sharp asymptotic $(2n/3)$-bound on the security of the Lizard-construction against generic chosen-IV key recovery and packet prediction TMD-TO attacks. We hope that the security model and the lower bound techniques developed here will help to prove similar sharp security bounds for other stream cipher constructions.

We have further shown that for a packet length $R > n$, where n denotes the inner state length of the underlying KSG, KSG-based stream ciphers can be only $(n/2)$-secure w.r.t. generic TMD-TO distinguishing attacks (see Corollary 7.1). From a theoretical point of view, it would be very interesting to analyze the case $R = n$. Our conjecture is that for $R = n$, the Lizard-construction is $(2n/3)$-secure even against distinguishing attacks.

In the following Chapter 8, we will present our new lightweight stream cipher Lizard, which is the first practical instantiation of the Lizard-construction and was introduced at FSE 2017. The results of our corresponding hardware implementation for ASICs will show that the Lizard-construction is a very promising design principle for the creation of small-state stream ciphers (cf. Chapter 6) suitable for ultra-constrained RFIDs (cf. Chapter 2). In particular, employing the Lizard-construction allows for *provable* security against generic TMD-TO key recovery attacks, something which is yet missing for Sprout-like approaches. Based on these insights, in Section 9.2, we will then suggest how Lizard

(resp. the LIZARD-construction) can be used to realize lightweight, privacy-preserving authentication, thus constituting a viable alternative to prevalent block cipher-based constructions (cf. Chapter 3). We consider this use case particularly interesting due to our above conjecture about the $(2n/3)$-security of the LIZARD-construction against generic TMD-TO distinguishing attacks for the case $R = n$.

Appendix 7.A A Short Excursion to Chernoff Bounds

At several places of our proof, we have to apply a technique called *Chernoff bounds* in the literature. The basic Chernoff bound argument is the following.

Theorem 7.6

Let N be a positive integer, $p \in (0,1)$, and $A_1, \ldots, A_N$ be a set of mutually independent random variables, where for all $i = 1, \ldots, N$, it holds that $\Pr[A_i = 1-p] = p$ and $\Pr[A_i = -p] = 1-p$. Let $A = \sum_{i=1}^{N} A_i$. Then, for all $a > 0$, it holds

$$\Pr[A > a] < e^{-2 \cdot a^2 / N}. \tag{7.51}$$

Proof of Theorem 7.6: See, e.g., Theorem A4 on page 235 of [ASE92]. □

We derive from Theorem 7.6 a corresponding result for random $\{0,1\}$-variables.

Lemma 7.6

Let p, N, and A_i for $i = 1, \ldots, N$ be defined as in Theorem 7.6, and let $B_i = A_i + p$. Note that $B_i \in \{0,1\}$ and $\Pr[B_i = 1] = p$. Let $B = \sum_{i=1}^{N} B_i$. Then, for all $d > 0$, it holds

$$\Pr[B > (p+d)N] < e^{-2 \cdot d^2 \cdot N}.$$

Proof of Lemma 7.6: By definition, $B = A + N \cdot p$. The proof is completed by putting $a = d \cdot N$ into Eq. (7.51) in Theorem 7.6. □

We will apply Chernoff bound arguments in the following modified scenario and start with some denotation.

Definition 7.19

Let $N \geq 1$ and let $X_1, \ldots, X_N$ denote a collection of random $\{0,1\}$-variables. For all i, $1 \leq i \leq N$, and for all $b = (b_1, \ldots, b_i) \in \{0,1\}^i$, let $X(b)$ denote the event that $X_j = b_j$ for all $j = 1, \ldots, i$.

Lemma 7.7

Let $C_1, \ldots, C_N$ denote a collection of random $\{0,1\}$-variables fulfilling the following two conditions for some probability bound p, $0 < p < 1$:

- $\Pr[C_1 = 1] = p_1 \leq p$.
- For all i, $2 \leq i \leq N$, and all $b \in \{0,1\}^{i-1}$, there is some number $p(b) \leq p$, which can be computed from b, and for which it holds that

$$\Pr[C_i = 1 \mid C(b)] = p(b).$$

Let $C = \sum_{i=1}^{N} C_i$. Then, for all $d > 0$, it holds

$$\Pr[C > (p+d)N] < e^{-2 \cdot d^2 \cdot N}.$$

Note that for any $D > 0$ and $d = \sqrt{D/(2N)}$, we obtain

$$\Pr[C > (p+d)N] = \Pr\left[C > pN + \sqrt{\frac{D \cdot N}{2}}\right] < e^{-D}.$$

Proof of Lemma 7.7: We construct a collection of mutually independent binary random variables $B_1, \ldots, B_N$ satisfying

- $C_i = 1$ implies $B_i = 1$,
- $\Pr[B_i = 1] = p$

for all i, $1 \leq i \leq N$. This proves our Lemma 7.7, as $\sum_{i=1}^{N} C_i \leq \sum_{i=1}^{N} B_i$ with probability one, and as Lemma 7.6 can be applied to $B = \sum_{i=1}^{N} B_i$.

The experiments Exp_i behind B_i are for all i, $1 \leq i \leq N$, defined as follows. We start with Exp_1: Exp_1 performs the experiment behind C_1. If $C_1 = 1$, then it outputs $B_1 = 1$. If $C_1 = 0$, then it starts an independent experiment with output $E_1 \in \{0,1\}$ and $\Pr[E_1 = 1] = q_1 = \frac{p - p_1}{1 - p_1}$.

For $i > 1$, the experiment Exp_i is defined as follows:

- Look at the outcome $b \in \{0,1\}^{i-1}$ of the experiments behind $C_1, \ldots, C_{i-1}$ and compute $p(b)$.
- Perform the experiment behind C_i (note that $\Pr[C_i = 1 \mid C(b)] = p(b)$). If $C_i = 1$, then output $B_i = 1$. If $C_i = 0$, then start an independent experiment E_i with output $E_i \in \{0,1\}$ and $\Pr[E_i = 1] = q(b) = \frac{p - p(b)}{1 - p(b)}$.

Note that $\Pr[B_1 = 1] = p$, as

$$\begin{aligned} &\Pr[B_1 = 1 \mid C_1 = 1] \cdot \Pr[C_1 = 1] + \Pr[B_1 = 1 \mid C_1 = 0] \cdot \Pr[C_1 = 0] \\ &= 1 \cdot p_1 + q_1 \cdot (1 - p_1) = p_1 + (p - p_1) = p. \end{aligned}$$

A similar argument shows that for all i, $2 \leq i \leq N$, and all $b \in \{0,1\}^{i-1}$, it holds that $\Pr[B_i = 1 \mid C(b)] = p$. From this, it follows directly that $\Pr[B_i = 1] = p$.

We now show the mutual independence of $B_1, \ldots, B_N$. Remember that for all i, $1 \leq i \leq N$, and all vectors $b = (b_1, \ldots, b_i) \in \{0,1\}^i$, $B(b)$ denotes the event that $B_j = b_j$ for all $j = 1, \ldots, i$, and that $C(b)$ denotes the event that $C_j = b_j$ for all $j = 1, \ldots, i$.

For showing the mutual independence of $B_1, \ldots, B_N$, it is obviously sufficient to prove that for all $b \in \{0,1\}^N$, it holds

$$\Pr[B(b)] = p^{|b|} \cdot (1-p)^{N-|b|}, \tag{7.52}$$

where $|b|$ denotes the number of 1-components in b.

We show Eq. (7.52) by induction on N. The case $N = 1$ follows directly from above, where it was shown that $\Pr[B_1 = 1] = p$.

Now let us assume that $N > 1$ and that Eq. (7.52) holds for $i = 1, \ldots, N-1$. We fix some vector $b \in \{0,1\}^N$ and denote $\tilde{b} = (b_1, \ldots, b_{N-1})$. Note that by the induction hypothesis, it holds

$$\begin{aligned}\Pr[B(b)] &= \Pr\Big[B\Big(\tilde{b}\Big) \cap (B_N = b_N)\Big] = \Pr\Big[B\Big(\tilde{b}\Big)\Big] \cdot \Pr\Big[B_N = b_N \;\Big|\; B\Big(\tilde{b}\Big)\Big] \\ &= p^{|\tilde{b}|} \cdot (1-p)^{N-1-|\tilde{b}|} \cdot \Pr\Big[B_N = b_N \;\Big|\; B\Big(\tilde{b}\Big)\Big].\end{aligned}$$

Consequently, it is sufficient to show that for all $\tilde{b} = (b_1, \ldots, b_{N-1}) \in \{0,1\}^{N-1}$, it holds $\Pr[B_N = 1 \mid B(\tilde{b})] = p$.

Note that the event $B(\tilde{b})$ equals the disjoint union of all events $B(\tilde{b}) \cap C(b')$, where b' ranges over all vectors from $\{0,1\}^{N-1}$ for which $b' \leq \tilde{b}$, i.e., for which $b'_j \leq \tilde{b}_j$ for all $j = 1, \ldots, N-1$. It hence suffices to show that $\Pr[B_N = 1 \mid C(b')] = p$ for all $b' \leq \tilde{b}$. But this follows directly from the definition of the experiment behind B_N (see above). □

I am the Lizard King,
I can do anything!

Jim Morrison (The Doors)

CHAPTER 8

Lizard – A Lightweight Stream Cipher for Power-constrained Devices

ABSTRACT

In the previous chapters, we have seen that TMD-TO attacks limit the security level of many classical stream ciphers (like E_0, A5/1, Trivium, Grain) to $n/2$, where n denotes the inner state length of the underlying KSG. Seeking to overcome this limitation, very recently the new field of so-called *small-state stream ciphers* has emerged. As part of the respective overview in Chapter 6, we already sketched the design of Lizard, our new lightweight stream cipher for power-constrained devices like passive RFID tags. In this chapter, we now provide the full details, including an in-depth security analysis and an assessment of its hardware characteristics in terms of the metrics introduced in Chapter 2.

The cipher's efficiency results from combining a Grain-like design with the Lizard-construction (cf. Chapter 7), our new construction principle for the state initialization of stream ciphers, which offers provable $(2n/3)$-security against TMD-TO attacks aiming at key recovery. Lizard uses 120-bit keys, 64-bit IVs, and has an inner state length of 121 bits. It is supposed to provide 80-bit security against key recovery attacks. Lizard allows to generate up to 2^{18} keystream bits per key/IV pair, which would be sufficient for many existing communication scenarios like Bluetooth, WLAN, or HTTPS.

Declaration of Origin: This chapter is based on the paper *LIZARD – A Lightweight Stream Cipher for Power-constrained Devices* [HKM17b], written together with Matthias Krause and Willi Meier and presented at *FSE 2017*.

8.1 Introduction

As explained in Chapter 5, stream ciphers have a long history when it comes to protecting digital communication. Due to the mentioned security flaws of older designs such as RC4 [Sch95], E_0 (Bluetooth) [Blu14], and A5/1 (GSM) [BGW99], in 2004, the eSTREAM project [ECR08] was started in order to identify new stream ciphers for different application profiles. In the hardware category, aiming at devices with restricted resources, three ciphers are still part of the eSTREAM portfolio after the latest revision in 2012 [BBV12]: Grain v1 [HJM06], MICKEY 2.0 [BD06], and Trivium [CP05].

Grain v1 uses 80-bit keys, 64-bit IVs and the authors do not give an explicit limit on the number of keystream bits that should be generated for each key/IV pair (cf. Subsection 5.2.4). MICKEY 2.0 uses 80-bit keys, IVs of variable length up to 80 bits and the maximum amount of keystream bits for each key/IV pair is 2^{40}. Trivium uses 80-bit keys, 80-bit IVs and at most 2^{64} keystream bits should be generated for each key/IV pair (cf. Subsection 5.2.3).

Interestingly, all three ciphers of the eSTREAM hardware portfolio are obviously designed for potentially very large keystream sequences per key/IV pair. In contrast, common transmission standards use much smaller packet sizes. For example, A5/1 of GSM produces only 228 keystream bits per key/IV pair, where the session key is 64 bits long and the IV corresponds to 22 bits of the publicly known frame (i.e., packet) number (cf. Subsection 5.2.2). Similarly, Bluetooth packets contain at most 2790 bits for the so-called *basic rate*. The Bluetooth cipher E_0 takes a 128-bit session key and uses 26 bits of the master's clock, which is assumed to be publicly known, as the packet-specific IV (cf. Subsection 5.2.1). For wireless local area networks (WLANs), the currently active IEEE 802.11-2012 standard [Ins12] implies that at most 7943 bytes (i.e., $< 2^{16}$ bits) are encrypted under the same key/IV pair using CCMP. Another widespread example for encryption on a per-packet basis is SSL/TLS, which underlies Hypertext Transfer Protocol Secure (HTTPS) and thus plays a vital role in securing the World Wide Web. In the most recent version, TLS 1.2 [DR08], the maximum amount of data encrypted under the same key/IV pair is $2^{14} + 2^{10}$ bytes (i.e., $2^{17} + 2^{13} < 2^{18}$ bits), as long as RC4, which is now forbidden for all TLS versions by RFC 7465 [Pop15], is not used.

Considering the above examples, the natural question arises whether a stream cipher with attractive features regarding security and efficiency could be designed by specifically targeting such *packet mode* scenarios (cf. Section 5.1), where only moderately long pieces of keystream are generated under the same key/IV pair. In this chapter, we answer the question in the affirmative by presenting Lizard, a lightweight stream cipher for power-constrained devices. It takes 120-bit keys, 64-bit IVs and generates up to 2^{18} keystream bits per key/IV pair. (Note that a maximum packet size of 2^{18} bits covers all of the real-world protocol examples given in the previous paragraph.) Lizard is designed to provide 80-bit security against key recovery attacks including generic TMD-TO attacks. This is remarkable insofar as Lizard's inner state is only 121 bits wide.

In contrast, Trivium and Grain v1, for example, have an inner state length of at least twice the size of the targeted security level against key recovery attacks. This is due to the inherent vulnerability of classical stream ciphers (i.e., stream ciphers which compute the keystream based on a so-called *initial state*) against TMD-TO attacks like those of Babbage [Bab95] and Biryukov and Shamir [BS00], which allow to recover some inner state during keystream generation (and, usually, also the corresponding initial state by clocking the cipher backwards) with an overall attack complexity of $2^{n/2}$, where n denotes the inner state length of the underlying KSG (see Theorem 7.1 in Section 7.3). As the state initialization algorithm, which computes the initial state from a given key/IV pair, is efficiently invertible for Trivium and Grain v1 (cf. Section 5.3), knowing the initial state immediately reveals the secret key. And even if the state initialization algorithm is not efficiently invertible, variants of such TMD-TO attacks often allow for key recovery (see Subsection 8.4.2 for further details). Lizard, on the other hand, represents the first practical instantiation of the Lizard-construction introduced in Chapter 7, which provides a method for designing stream ciphers in such a way that, for packet mode scenarios, a beyond-the-birthday-bound security level of $2n/3$ w.r.t. generic TMD-TO attacks aiming at key recovery can be proved. (TMD-TO attacks aiming at recovering some initial state still work for Lizard, but they do not allow to straightforwardly derive the underlying secret key.)

The design of Lizard is closely related to that of Grain v1 (cf. Subsection 5.2.4), which turned out to be the most hardware-efficient member of the eSTREAM portfolio and, hence, can be considered as a benchmark for new hardware-oriented designs. When compared to Grain v1, the major differences of Lizard are the smaller inner state (121 bits vs. 160 bits), the larger key (120 bits vs. 80 bits) and the fact that Lizard introduces the secret key not only once but twice during its state initialization. All of these modifications are a direct consequence of implementing the generic Lizard-construction as explained in Subsection 8.3.5. Naturally, reducing the size of the inner state also required to change the underlying feedback shift registers (FSRs) and, for security reasons, to choose a heavier output function.

In Chapter 6, where we already sketched the structure of Lizard, we also gave an overview over some other recent small-state stream ciphers, namely Sprout [AM15], Plantlet [MAM17], and Fruit [GHX16]. Common to these three ciphers is that they continuously use the secret key during keystream generation, which, as explained in Section 6.1, comes with certain efficiency drawbacks. In particular, continuously accessing the secret key from EEPROM can have a severe impact on an RFID tag's power budget (see Subsection 2.3.9 for further details) as well as limit the maximum possible encryption speed due to latency issues. Also note that, unlike Lizard, neither Sprout nor Plantlet nor Fruit offer *provable* security against TMD-TO attacks aiming at key recovery.

We would like to point out that the above arguments by no means imply that we reject the idea of stream ciphers that continuously access the secret key. However, they illustrate the need for alternative ideas that allow to reduce the size of the inner state

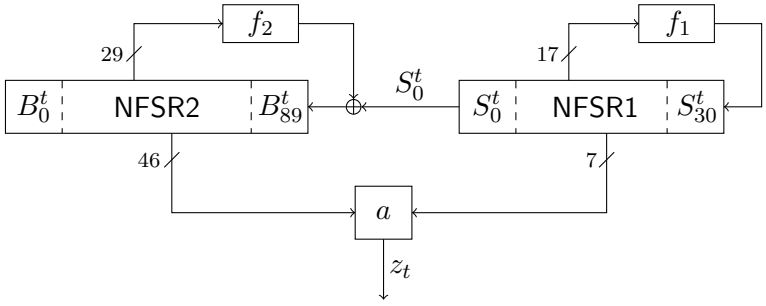

Figure 8.1: LIZARD in keystream generation mode.

even in scenarios where continuous key access is not feasible.

LIZARD has been designed with low-cost scenarios like passively powered RFID tags in mind. In particular, it outperforms Grain v1 in important hardware metrics like cell area and power consumption. Most notably, the estimated power consumption is about 16 percent below that of Grain v1, making LIZARD particularly suitable for power-constrained devices. This shows that in scenarios where plaintext packets of moderate length are to be encrypted under individual IVs, the LIZARD-construction provides an interesting alternative to conventional state initialization algorithms of stream ciphers.

Structure of this chapter: In Section 8.2, we specify the components of LIZARD (Subsection 8.2.1) and describe how it is operated during state initialization (Subsection 8.2.2) and keystream generation (Subsection 8.2.3). Building on this, Section 8.3 then provides the corresponding design considerations. In particular, Subsection 8.3.5 contains a detailed explanation of how the generic LIZARD-construction introduced in Chapter 7 has been implemented for LIZARD. Section 8.4 treats the cryptanalysis of LIZARD and in Section 8.5, we present the details of our hardware implementation along with a comparison of the corresponding performance metrics between LIZARD and Grain v1. Section 8.6 concludes the chapter and provides an outlook on potential future work as well as on the remaining parts of this thesis. Test vectors and a reference implementation can be found in Appendix 8.A and Appendix 8.C, respectively.

8.2 Design Specification

The design of LIZARD is similar to (and was in fact inspired by) that of the Grain family [HJMM08] of stream ciphers (see Subsection 5.2.4). In particular, the inner state of LIZARD is distributed over two interconnected FSRs as depicted in Fig. 8.1.

Note, however, that while Grain uses one LFSR and one NFSR, which, moreover, are of the same length, LIZARD uses two NFSRs of different lengths instead. The reasons for this design choice will be explained in Section 8.3. Like in Grain, the third important

building block besides the two FSRs is a nonlinear output function, which takes inputs from both shift registers and is also used as part of the state initialization algorithm.

In the following, we describe the components of the cipher in detail (Subsection 8.2.1) and specify how it is operated during state initialization (Subsection 8.2.2) and keystream generation (Subsection 8.2.3). For the sake of clarity, Subsections 8.2.1 to 8.2.3 contain only the technical aspects of Lizard. Explanations of important design choices are given separately in Section 8.3, along with a discussion of the security properties of the particular components (e.g., the algebraic properties of the feedback functions).

8.2.1 Components

The 121-bit inner state of Lizard is distributed over two NFSRs, NFSR1 and NFSR2, whose contents at time $t = 0, 1, \ldots$ we denote by $(S_0^t, \ldots, S_{30}^t)$ and $(B_0^t, \ldots, B_{89}^t)$, respectively (cf. Fig. 8.1). As NFSR1 and NFSR2 are Fibonacci-type, for $t \in \mathbb{N} \setminus \{0, 128\}$ (cf. Subsection 8.2.2) it holds that $S_i^{t+1} := S_{i+1}^t$, $i = 0, \ldots, 29$, and $B_j^{t+1} := B_{j+1}^t$, $j = 0, \ldots, 88$.

NFSR1

In Lizard, NFSR1 replaces the LFSR of the Grain family of stream ciphers. NFSR1 is 31 bits wide and corresponds to the NFSR A_{10} of the eSTREAM Phase 2 (hardware portfolio) candidate ACHTERBAHN-128/80 [GGK06]. For non-zero starting states, it has a guaranteed period of $2^{31} - 1$ (i.e., maximal) and can be specified by the following update relation (during keystream generation), defining f_1 depicted in Fig. 8.1:

$$\begin{aligned} S_{30}^{t+1} := {} & S_0^t \oplus S_2^t \oplus S_5^t \oplus S_6^t \oplus S_{15}^t \oplus S_{17}^t \oplus S_{18}^t \oplus S_{20}^t \oplus S_{25}^t \\ & \oplus S_8^t S_{18}^t \oplus S_8^t S_{20}^t \oplus S_{12}^t S_{21}^t \oplus S_{14}^t S_{19}^t \oplus S_{17}^t S_{21}^t \oplus S_{20}^t S_{22}^t \\ & \oplus S_4^t S_{12}^t S_{22}^t \oplus S_4^t S_{19}^t S_{22}^t \oplus S_7^t S_{20}^t S_{21}^t \oplus S_8^t S_{18}^t S_{22}^t \\ & \oplus S_8^t S_{20}^t S_{22}^t \oplus S_{12}^t S_{19}^t S_{22}^t \oplus S_{20}^t S_{21}^t S_{22}^t \oplus S_4^t S_7^t S_{12}^t S_{21}^t \\ & \oplus S_4^t S_7^t S_{19}^t S_{21}^t \oplus S_4^t S_{12}^t S_{21}^t S_{22}^t \oplus S_4^t S_{19}^t S_{21}^t S_{22}^t \\ & \oplus S_7^t S_8^t S_{18}^t S_{21}^t \oplus S_7^t S_8^t S_{20}^t S_{21}^t \oplus S_7^t S_{12}^t S_{19}^t S_{21}^t \\ & \oplus S_8^t S_{18}^t S_{21}^t S_{22}^t \oplus S_8^t S_{20}^t S_{21}^t S_{22}^t \oplus S_{12}^t S_{19}^t S_{21}^t S_{22}^t. \end{aligned}$$

NFSR2

NFSR2 is 90 bits wide and uses a modified version of g from Grain-128a [gHJM11] as its feedback polynomial. More precisely, f_2 of NFSR2 (cf. Fig. 8.1) squeezes the taps of g to fit a 90-bit register, resulting in the following update relation (during keystream generation):

$$B_{89}^{t+1} := S_0^t \oplus B_0^t \oplus B_{24}^t \oplus B_{49}^t \oplus B_{79}^t \oplus B_{84}^t \oplus B_3^t B_{59}^t \oplus B_{10}^t B_{12}^t$$

$$\oplus B_{15}^t B_{16}^t \oplus B_{25}^t B_{53}^t \oplus B_{35}^t B_{42}^t \oplus B_{55}^t B_{58}^t \oplus B_{60}^t B_{74}^t$$
$$\oplus B_{20}^t B_{22}^t B_{23}^t \oplus B_{62}^t B_{68}^t B_{72}^t \oplus B_{77}^t B_{80}^t B_{81}^t B_{83}^t.$$

Note that this update relation for B_{89}^{t+1} additionally contains the masking bit S_0^t from NFSR1 (analogously to the Grain family).

Output Function a

$a : \{0,1\}^{53} \longrightarrow \{0,1\}$ builds on the construction scheme introduced in [MJSC16] as part of the FLIP family of stream ciphers (see Section 8.3 for details). For the sake of clarity, we define a through the output bit z_t of LIZARD at time t, which is computed as

$$z_t := \mathcal{L}_t \oplus \mathcal{Q}_t \oplus \mathcal{T}_t \oplus \tilde{\mathcal{T}}_t,$$

where

$$\begin{aligned}
\mathcal{L}_t &:= B_7^t \oplus B_{11}^t \oplus B_{30}^t \oplus B_{40}^t \oplus B_{45}^t \oplus B_{54}^t \oplus B_{71}^t,\\
\mathcal{Q}_t &:= B_4^t B_{21}^t \oplus B_9^t B_{52}^t \oplus B_{18}^t B_{37}^t \oplus B_{44}^t B_{76}^t,\\
\mathcal{T}_t &:= B_5^t \oplus B_8^t B_{82}^t \oplus B_{34}^t B_{67}^t B_{73}^t \oplus B_2^t B_{28}^t B_{41}^t B_{65}^t\\
&\quad \oplus B_{13}^t B_{29}^t B_{50}^t B_{64}^t B_{75}^t \oplus B_6^t B_{14}^t B_{26}^t B_{32}^t B_{47}^t B_{61}^t\\
&\quad \oplus B_1^t B_{19}^t B_{27}^t B_{43}^t B_{57}^t B_{66}^t B_{78}^t,\\
\tilde{\mathcal{T}}_t &:= S_{23}^t \oplus S_3^t S_{16}^t \oplus S_9^t S_{13}^t B_{48}^t \oplus S_1^t S_{24}^t B_{38}^t B_{63}^t.
\end{aligned}$$

8.2.2 State Initialization

The state initialization process can be divided into four phases. Phases 1–3 constitute an instantiation of the LIZARD-construction design principle introduced in Chapter 7, which, in our case, provides 80-bit security against generic TMD-TO attacks aiming at key recovery. Phase 4 is a consequence of the necessity to make sure that NFSR1 is not in the all-zero state after phase 3 (see below for details).

Phase 1: Key and IV Loading

Let $K = (K_0, \ldots, K_{119})$ denote the 120-bit key and $IV = (IV_0, \ldots, IV_{63})$ the 64-bit public IV. The registers of the KSG are initialized as follows:

$$B_j^0 := \begin{cases} K_j \oplus IV_j, & \text{for } j \in \{0, \ldots, 63\}, \\ K_j, & \text{for } j \in \{64, \ldots, 89\}, \end{cases}$$

$$S_i^0 := \begin{cases} K_{i+90}, & \text{for } i \in \{0, \ldots, 28\}, \\ K_{119} \oplus 1, & \text{for } i = 29, \\ 1, & \text{for } i = 30. \end{cases}$$

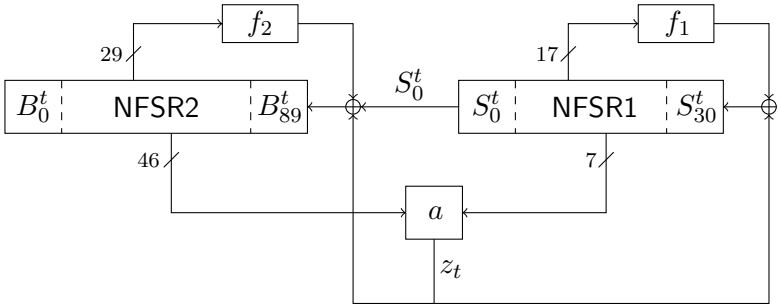

Figure 8.2: LIZARD in phase 2 of the state initialization.

Phase 2: Grain-like Mixing

Clock the cipher 128 times without producing actual keystream. Instead, at time $t = 0, \ldots, 127$, the output bit z_t is fed back into both FSRs as depicted in Fig. 8.2. To avoid ambiguity, we now give the full update relations that will be used for NFSR2 and NFSR1 in phase 2. For $t = 0, \ldots, 127$, compute

$$
\begin{aligned}
B_j^{t+1} &:= B_{j+1}^t, \quad \text{for } j \in \{0, \ldots, 88\}, \\
B_{89}^{t+1} &:= z_t \oplus S_0^t \oplus B_0^t \oplus B_{24}^t \oplus B_{49}^t \oplus B_{79}^t \oplus B_{84}^t \oplus B_3^t B_{59}^t \oplus B_{10}^t B_{12}^t \\
&\quad \oplus B_{15}^t B_{16}^t \oplus B_{25}^t B_{53}^t \oplus B_{35}^t B_{42}^t \oplus B_{55}^t B_{58}^t \oplus B_{60}^t B_{74}^t \\
&\quad \oplus B_{20}^t B_{22}^t B_{23}^t \oplus B_{62}^t B_{68}^t B_{72}^t \oplus B_{77}^t B_{80}^t B_{81}^t B_{83}^t, \\
S_i^{t+1} &:= S_{i+1}^t, \quad \text{for } i \in \{0, \ldots, 29\}, \\
S_{30}^{t+1} &:= z_t \oplus S_0^t \oplus S_2^t \oplus S_5^t \oplus S_6^t \oplus S_{15}^t \oplus S_{17}^t \oplus S_{18}^t \oplus S_{20}^t \oplus S_{25}^t \\
&\quad \oplus S_8^t S_{18}^t \oplus S_8^t S_{20}^t \oplus S_{12}^t S_{21}^t \oplus S_{14}^t S_{19}^t \oplus S_{17}^t S_{21}^t \oplus S_{20}^t S_{22}^t \\
&\quad \oplus S_4^t S_{12}^t S_{22}^t \oplus S_4^t S_{19}^t S_{22}^t \oplus S_7^t S_{20}^t S_{21}^t \oplus S_8^t S_{18}^t S_{22}^t \\
&\quad \oplus S_8^t S_{20}^t S_{22}^t \oplus S_{12}^t S_{19}^t S_{22}^t \oplus S_{20}^t S_{21}^t S_{22}^t \oplus S_4^t S_7^t S_{12}^t S_{21}^t \\
&\quad \oplus S_4^t S_7^t S_{19}^t S_{21}^t \oplus S_4^t S_{12}^t S_{21}^t S_{22}^t \oplus S_4^t S_{19}^t S_{21}^t S_{22}^t \\
&\quad \oplus S_7^t S_8^t S_{18}^t S_{21}^t \oplus S_7^t S_8^t S_{20}^t S_{21}^t \oplus S_7^t S_{12}^t S_{19}^t S_{21}^t \\
&\quad \oplus S_8^t S_{18}^t S_{21}^t S_{22}^t \oplus S_8^t S_{20}^t S_{21}^t S_{22}^t \oplus S_{12}^t S_{19}^t S_{21}^t S_{22}^t,
\end{aligned}
$$

where $z_t := \mathcal{L}_t \oplus \mathcal{Q}_t \oplus \mathcal{T}_t \oplus \tilde{\mathcal{T}}_t$ is computed as described in Subsection 8.2.1.

Phase 3: Second Key Addition

In this phase, the 120-bit key is XORed in a bitwise fashion to the inner state of the KSG, similar to how it was introduced in phase 1. More precisely, the following update relations for NFSR2 and NFSR1 constitute phase 3:

$$B_j^{129} := B_j^{128} \oplus K_j, \quad \text{for } j \in \{0, \ldots, 89\},$$

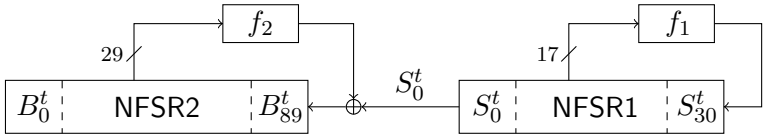

Figure 8.3: LIZARD in phase 4 of the state initialization. The output function a is omitted as all output bits generated in this phase are discarded.

$$S_i^{129} := \begin{cases} S_i^{128} \oplus K_{i+90}, & \text{for } i \in \{0, \ldots, 29\}, \\ 1, & \text{for } i = 30. \end{cases}$$

Note that, like in phase 1, the rightmost cell of NFSR1 is here again set to 1 in order to avoid the all-zero state, which would practically render our maximum-length FSR NFSR1 ineffective as, after phase 3, the only input to it will come from its feedback function (i.e., for $t \geq 129$, S_i^{t+1}, $i = 0, \ldots, 30$, is computed as described in Subsection 8.2.1). In contrast to phase 1, the key bit K_{119} is not inverted in phase 3.

Phase 4: Final Diffusion

As pointed out previously, phases 1–3 constitute an instantiation of the key loading and mixing steps of the generic LIZARD-construction (cf. Chapter 7), which, in our case, yields 80-bit security w.r.t. generic TMD-TO attacks aiming at key recovery. Phase 4 is additionally required in order to obtain the necessary diffusion w.r.t. the bit at the rightmost position of NFSR1, which was set to 1 in phase 3. We achieve this by stepping the whole KSG 128 times in keystream generation mode as phase 4 (see Fig. 8.3). Note that the 128 output bits produced during phase 4 are discarded, i.e., they are not used as actual keystream.

Once more, to avoid ambiguity, we give the full update relations that will be used for NFSR2 and NFSR1 in phase 4. For $t = 129, \ldots, 256$, compute

$$\begin{aligned}
B_j^{t+1} &:= B_{j+1}^t, \quad \text{for } j \in \{0, \ldots, 88\}, \\
B_{89}^{t+1} &:= S_0^t \oplus B_0^t \oplus B_{24}^t \oplus B_{49}^t \oplus B_{79}^t \oplus B_{84}^t \oplus B_3^t B_{59}^t \oplus B_{10}^t B_{12}^t \\
&\quad \oplus B_{15}^t B_{16}^t \oplus B_{25}^t B_{53}^t \oplus B_{35}^t B_{42}^t \oplus B_{55}^t B_{58}^t \oplus B_{60}^t B_{74}^t \\
&\quad \oplus B_{20}^t B_{22}^t B_{23}^t \oplus B_{62}^t B_{68}^t B_{72}^t \oplus B_{77}^t B_{80}^t B_{81}^t B_{83}^t, \\
S_i^{t+1} &:= S_{i+1}^t, \quad \text{for } i \in \{0, \ldots, 29\}, \\
S_{30}^{t+1} &:= S_0^t \oplus S_2^t \oplus S_5^t \oplus S_6^t \oplus S_{15}^t \oplus S_{17}^t \oplus S_{18}^t \oplus S_{20}^t \oplus S_{25}^t \\
&\quad \oplus S_8^t S_{18}^t \oplus S_8^t S_{20}^t \oplus S_{12}^t S_{21}^t \oplus S_{14}^t S_{19}^t \oplus S_{17}^t S_{21}^t \oplus S_{20}^t S_{22}^t \\
&\quad \oplus S_4^t S_{12}^t S_{22}^t \oplus S_4^t S_{19}^t S_{22}^t \oplus S_7^t S_{20}^t S_{21}^t \oplus S_8^t S_{18}^t S_{22}^t \\
&\quad \oplus S_8^t S_{20}^t S_{22}^t \oplus S_{12}^t S_{19}^t S_{22}^t \oplus S_{20}^t S_{21}^t S_{22}^t \oplus S_4^t S_7^t S_{12}^t S_{21}^t \\
&\quad \oplus S_4^t S_7^t S_{19}^t S_{21}^t \oplus S_4^t S_{12}^t S_{21}^t S_{22}^t \oplus S_4^t S_{19}^t S_{21}^t S_{22}^t
\end{aligned}$$

$$\oplus\, S_7^t S_8^t S_{18}^t S_{21}^t \oplus S_7^t S_8^t S_{20}^t S_{21}^t \oplus S_7^t S_{12}^t S_{19}^t S_{21}^t$$
$$\oplus\, S_8^t S_{18}^t S_{21}^t S_{22}^t \oplus S_8^t S_{20}^t S_{21}^t S_{22}^t \oplus S_{12}^t S_{19}^t S_{21}^t S_{22}^t.$$

See Subsection 8.3.5 for further remarks on this design choice.

8.2.3 Keystream Generation

At the end of the state initialization, the 31-bit (initial) state of NFSR1 is $(S_0^{257}, \ldots, S_{30}^{257})$ and the 90-bit (initial) state of NFSR2 is $(B_0^{257}, \ldots, B_{89}^{257})$. The first keystream bit that is used for plaintext encryption is z_{257}. For $t \geq 257$, the states $(S_0^{t+1}, \ldots, S_{30}^{t+1})$ and $(B_0^{t+1}, \ldots, B_{89}^{t+1})$ and the output bit z_t are computed using the relations given in Subsection 8.2.1. Figure 8.1 on page 218 depicts the structure of Lizard during keystream generation.

As Lizard is designed to be operated in packet mode, the maximum size of a plaintext packet encrypted under the same key/IV pair is 2^{18} bits and no key/IV pair may be used more than once, i.e., for more than one packet. Let $X = (x_0, \ldots, x_{|X|-1})$ denote such a plaintext packet and let $z_{257}, z_{258} \ldots$ be the keystream generated for it as described before. Then the corresponding ciphertext packet $Y = (y_0, \ldots, y_{|X|-1})$ can be produced via $y_i := x_i \oplus z_{i+257}$, $i = 0, \ldots, |X| - 1$. Decryption (given that the secret session key and the public IV are known) works analogously.

Note that, though we use the terms *plaintext/ciphertext packet* here, Lizard is really a (synchronous) stream cipher. That is, the keystream bits $z_{257}, z_{258} \ldots$ are generated in a bitwise fashion (and independently of the plaintext/ciphertext) and, consequently, the individual plaintext bits x_i can be encrypted and then (in the form of y_i) transmitted as they arrive. The same obviously holds for the decryption of the ciphertext bits y_i on the receiver's side.

8.3 Design Considerations

In this section, we provide additional explanations w.r.t. our design, which were omitted in Section 8.2 for the sake of clarity. Based on several of the following properties, we will then argue in Section 8.4, why we believe that Lizard resists the currently known types of attacks against stream ciphers.

Please note that in the following subsections (and also in Section 8.4), we will repeatedly use standard cryptographic terms such as *nonlinearity*, *balancedness*, *resiliency*, *algebraic immunity* etc. in the description of the employed feedback and output functions. For corresponding definitions and further explanations, we refer the reader to, e.g., *Boolean Functions for Cryptography and Error Correcting Codes* [Car12] by Claude Carlet (esp. *Section 4.1: Cryptographic criteria for Boolean functions*). A nice overview is also given in the paper [MJSC16] by Méaux et al.

8.3.1 NFSR1

As mentioned in Subsection 8.2.1, NFSR1 is 31 bits wide and corresponds to the NFSR A_{10} of the eSTREAM Phase 2 (hardware portfolio) candidate ACHTERBAHN-128/80 [GGK06]. We chose the NFSR A_{10} out of the set of all maximum-length NFSRs used in ACHTERBAHN due to the fact that the two NFSRs which would have offered even slightly longer periods (namely, A_{11} and A_{12}) both use inconvenient taps w.r.t. potential speedup measures (see Subsection 8.3.3 for details). Moreover, we did not choose one of the smaller maximum-length NFSRs of ACHTERBAHN as (under the condition that we want to keep the speedup option as described below) those would have led to the situation that we would have had to use certain taps of NFSR1 at the same time in its feedback function and in the output function. The reason why we used the ACHTERBAHN design as a source for our NFSR1 in the first place is twofold.

First, we wanted NFSR1 to have guaranteed period $2^{31} - 1$, as, in LIZARD, NFSR1 replaces the maximum-length LFSR of the Grain family. Unfortunately, apart from special cases like constructing n-bit NFSRs with period $2^n - 1$ from n-bit maximum-length LFSRs (cf. [Dub13]), not much is known yet about how to construct large NFSRs with maximum period (see, e.g., [Dub12] or [Dub13] for further remarks and references to corresponding approaches).[1] However, due to the restriction to packet mode (with a maximum of 2^{18} keystream bits per key/IV pair), LIZARD actually does not need as large guaranteed periods as the Grain family, which allowed us to replace the maximum-length LFSR of Grain by a suitable maximum-length NFSR. The design document of ACHTERBAHN provides a collection of such maximum-length NFSRs of sufficient size.

The second reason for using A_{10} from ACHTERBAHN as our NFSR1 is its hardware efficiency. Despite A_{10}'s comparatively large algebraic normal form, the designers of ACHTERBAHN are able provide a compact hardware realization of the feedback function consuming only 31.75 gate equivalents (GE) and having logical depth three (see Chapter 2 and Section 8.5 for further details w.r.t. hardware complexity and an explanation of corresponding units of measure like GE).

When operated in a self-contained manner (i.e., after phase 3 of the state initialization), NFSR1 has a guaranteed period (for non-zero starting states) of $2^{31} - 1$. The following properties of A_{10} of ACHTERBAHN (and, hence, also of NFSR1 of LIZARD) were given in [GGK06]: a nonlinearity of 61 440, an order of correlation immunity of 6, and a diffusion parameter[2] of 61. Moreover, it is easy to see that the feedback function f_1 of NFSR1 is balanced and, thus (as it is 6th order correlation-immune), 6-resilient. For comparison, in

[1] In fact, a de Bruijn sequence of length 2^n can be generated with an n-bit NFSR. Still, for simplicity, we also call our 31-bit NFSR1 with period $2^{31} - 1$ *maximum-length*, in analogy to the case of LFSRs (where a period of 2^n is obviously impossible to achieve with an n-bit LFSR).

[2] The diffusion parameter λ was determined experimentally in [GGK06] and denotes "the minimum number of clock cycles needed in order to transform any two initial states of the shift register A_j of Hamming distance 1 into shift register states of Hamming distance close to $N_j/2$" (N_j denotes the size of the shift register A_j).

Grain v1 as well as in Grain-128a, the feedback function of the LFSR (which is replaced by NFSR1 in Lizard) has resiliency 5. Finally, the algebraic degree of f_1 is 4.

8.3.2 NFSR2

NFSR2 is 90 bits wide and its feedback polynomial is a modified version of g from Grain-128a [gHJM11], which, after years of intense cryptanalysis, is still considered unbroken. In contrast to NFSR1, the period of NFSR2 during keystream generation is unknown because, due to the masking bit from NFSR1, NFSR2 is actually a filter instead of a real NFSR (cf. corresponding remark for the Grain family in [HJMM08]).

As described in Subsection 8.2.1, f_2 of Lizard squeezes the taps of g from Grain-128a in such a way that the property of g that no tap appears more than once is preserved in f_2. In consequence, several important properties of g like its balancedness, its resiliency of 4, its nonlinearity of 267 403 264 and its security w.r.t. linear approximations (2^{14} best linear approximations with bias $63 \cdot 2^{-15}$) carry over to f_2 (see [gHJM11] for further details regarding the security of g of Grain-128a). The algebraic degree of f_2 is 4.

Beyond these properties, an important requirement for NFSR2 is that it should provide resistance against cube-like attacks on the initialization. The NFSR as selected for Lizard meets this condition as unpublished results suggest, [Tod17].

8.3.3 Output Function a

An important question in FSR-based stream cipher design is how to share the load of ensuring security between the driving register(s) and the output function. To compensate for the fact that the inner state of Lizard is smaller than that of Grain v1, we decided that the output function should have more inputs and larger algebraic degree instead. It builds on the construction scheme introduced in [MJSC16] as part of the FLIP family of stream ciphers. More precisely, Lizard's output function a can be written as the direct sum of a linear function with seven monomials, a quadratic function with four monomials, a triangular function with seven monomials, and another triangular function with four monomials, where each tap of NFSR1 and NFSR2 appears at most once in a.

As a consequence, the output function of Lizard is defined over 53 variables, balanced, and has, according to lemmata 3–6 in [MJSC16], the following security properties: a nonlinearity of 4 476 506 321 453 056 ($\approx 2^{51}$), a resiliency of 8, an algebraic immunity of at least 7, and a fast algebraic immunity of at least 8. The algebraic degree of a is 7.

If the content of NFSR1 at time t should be known to the attacker (e.g., as part of a guess-and-determine attack), the output function still depends on at least 43 variables and 'gracefully degrades' into the direct sum of a linear function with seven or eight (depending on the values of S_9^t and S_{13}^t) monomials, a quadratic function with four or five (depending on the values of S_1^t and S_{24}^t) monomials, and a triangular function with seven monomials, which again conforms to the construction principle introduced in [MJSC16]

and leads to the following worst-case security properties for that situation: a nonlinearity of $4\,317\,411\,672\,064$ ($\approx 2^{41}$), a resiliency of 7, an algebraic immunity of at least 7, and a fast algebraic immunity of at least 8.

While the choice of tap positions for state update functions is often already restricted by the need to guarantee a certain period (e.g., as in the case of NFSR1), the choice of tap positions for an output function is commonly less substantiated. For example, the design documents introducing the members of the Grain family (cf. [HJM06], [gHJM11], [HJMM08]) mainly focus on the conceptual question whether certain taps used in the output function should be from the NFSR or the LFSR (and how many of each). The more concrete question of *which* tap positions within each FSR are actually chosen for the output function is almost exclusively discussed in the context of hardware acceleration or when it comes to mitigate issues of previous versions arising from actual attacks (e.g., the attack of Dinur and Shamir [DS11] on Grain-128 [HJMM06], which lead to a change of tap positions in the output function of Grain-128a [gHJM11]).

In the absence of canonical criteria for the selection of tap positions for Grain-like constructions, we mainly resort to the concept of (full) positive difference sets that was used by Golić in [Gol96] to assess the security of nonlinear filter generators consisting of a single LFSR and a nonlinear output function: "for a positive integer λ, call Γ a λth-order positive difference set if λ is the maximum number of pairs of its elements with the same mutual difference (for $\lambda = 1$, we get a full positive difference set)" [Gol96].

In particular, the output function a of LIZARD has the following properties:

- The set
$$\{1, 3, 9, 13, 16, 23, 24\}$$
of output function taps from NFSR1 is a 2nd-order positive difference set. Moreover, no taps from NFSR1 are used at the same time for its feedback function and the output function. (In Grain-128a, the feedback function of the LFSR, which corresponds to our NFSR1, and the output function do not share any taps, either.)

- No taps from NFSR2 are used at the same time for its feedback function and the output function. (In Grain-128a, the feedback function of the NFSR, which corresponds to our NFSR2, and the output function share only a single tap called "b_{i+95}" in [gHJM11].)

- The direct sum $\mathcal{L}_t \oplus \mathcal{Q}_t \oplus \mathcal{T}_t$ uses only taps from NFSR2. (To maintain a sufficient security level even when the content of the smaller NFSR1 is known to the attacker, e.g., due to guessing; cf. Subsection 8.4.5.)

- The set
$$\{5, 7, 11, 30, 40, 45, 54, 71\}$$
of the tap indices (all from NFSR2) of the linear monomials of $\mathcal{L}_t \oplus \mathcal{Q}_t \oplus \mathcal{T}_t$ is a full positive difference set.

- The set
$$\{4, 8, 9, 18, 21, 37, 44, 52, 76, 82\}$$
of the tap indices (all from NFSR2) of the quadratic monomials of $\mathcal{L}_t \oplus \mathcal{Q}_t \oplus \mathcal{T}_t$ is a full positive difference set. One consequence of this is that each two bits of the internal bitstream of NFSR2 can form at most once a quadratic monomial together.

- The sets
$$\{|4-21|, |8-82|, |9-52|, |18-37|, |44-76|\}$$
of differences between the two taps (all from NFSR2) of each quadratic monomial in $\mathcal{L}_t \oplus \mathcal{Q}_t \oplus \mathcal{T}_t$ and
$$\{|3-59|, |10-12|, |15-16|, |25-53|, |35-42|\}$$
of differences between the two taps (all from NFSR2) of each quadratic monomial in the feedback function of NFSR2 are disjoint. Hence, even during phase 2 of the state initialization, each two bits of the internal bitstream of NFSR2 can form at most once a quadratic monomial together.

- None of the differences
$$\{|4-21|, |8-82|, |9-52|, |18-37|, |44-76|\}$$
between the two taps (all from NFSR2) of each quadratic monomial in $\mathcal{L}_t \oplus \mathcal{Q}_t \oplus \mathcal{T}_t$ appears as a difference between two taps of a higher degree monomial of $\mathcal{L}_t \oplus \mathcal{Q}_t \oplus \mathcal{T}_t$.

- Each of the sets
$$\{34, 67, 73\},$$
$$\{2, 28, 41, 65\},$$
$$\{13, 29, 50, 64, 75\},$$
$$\{6, 14, 26, 32, 47, 61\},$$
$$\{1, 19, 27, 43, 57, 66, 78\}$$
of tap indices (all from NFSR2) of the monomials of degree $3, \ldots, 7$ of $\mathcal{L}_t \oplus \mathcal{Q}_t \oplus \mathcal{T}_t$ is a full positive difference set. Consequently, each two bits of the internal bitstream of NFSR2 never appear more than once together as part of each (i.e., the same) of those monomials.

Another cryptanalytic result motivating our selection of tap positions for the output function of Lizard are attacks based on binary decision diagrams (BDDs). A direct consequence of this type of attack against stream ciphers, which was introduced by Krause in [Kra02] and applied to Grain-128 by Stegemann in [Ste07a], is that (roughly speaking) the distance between the smallest and the largest tap index of a monomial should be large for as many monomials as possible (see Subsection 8.4.10 for further details).

8.3.4 Speedup Options

Our update relation for NFSR1 does not involve the five taps corresponding to S_{26}^t, S_{27}^t, S_{28}^t, S_{29}^t, S_{30}^t in the generation of S_{30}^{t+1}. Analogously, B_{89}^{t+1} of NFSR2 is computed without the involvement of B_{85}^t, B_{86}^t, B_{87}^t, B_{88}^t, B_{89}^t. These ten taps are also not used in the output function of the cipher, which allows for a speedup of the keystream generation (up to a factor of 6) simply by multiplying the (comparatively cheap) hardware for the two feedback functions and the output function. Grain-128a uses the same idea to allow for a speedup of up to a factor of 32 (which is obviously not possible here as NFSR1 has only 31 register cells in total).

8.3.5 State Initialization Algorithm

As pointed out in Subsection 8.2.2, the state initialization algorithm of Lizard represents an instantiation of the generic Lizard-construction introduced in Chapter 7, which, in our case, provides 80-bit security against generic TMD-TO attacks aiming at key recovery (cf. Subsection 8.4.2). In a nutshell, the Lizard-construction design principle for stream ciphers can be described as follows:

- Choose an appropriate[3] KSG of inner state length n and an appropriate packet length R.
- Define the state initialization algorithm for packet i as follows:
  (1) **Loading:** Given the symmetric session key $k \in \{0,1\}^n$ and the packet initialization vector $IV^i \in \{0,1\}^n$, load $IV^i \oplus k$ into the inner state registers of the KSG, yielding the inner state $q_{\text{load}}^i = IV^i \oplus k$.
  (2) **Mixing:** Run an appropriate KSG-based mixing algorithm[4] on q_{load}^i, yielding the inner state q_{mixed}^i.
  (3) **Hardening:** XOR k bitwise to this inner state q_{mixed}^i, yielding the initial state $q_{\text{init}}^i = q_{\text{mixed}}^i \oplus k$.
- Starting with the initial state q_{init}^i, the keystream for packet i is generated by the KSG in the traditional way.

[3] *Appropriate* means that the KSG satisfies standard requirements w.r.t. design goals like security and hardware efficiency. $(2n/3)$-security against TMD-TO attacks aiming at key recovery is then added by the Lizard-construction.

[4] Such a mixing algorithm can, e.g., simply consist in repeatedly stepping the KSG in keystream generation mode without producing output (cf. Trivium [CP05], Subsection 5.2.3) or involve more sophisticated measures like generating a piece of *internal* keystream that is not output but instead loaded in parallel to the inner state registers of the KSG as the initial state based on which the actual keystream is then generated (cf. the E_0-cipher of the Bluetooth standard [Blu14], Subsection 5.2.1).

The state initialization algorithm of Lizard as described in Subsection 8.2.2 implements the above scheme with some minor adaptions, which we will describe in the following. Note that none of these modifications reduces the provable $(2n/3)$-security against TMD-TO key recovery attacks provided by the generic Lizard-construction.

When compared to the generic scheme, the first obvious difference of Lizard is that the IV length is now smaller than the key length. Even in the context of packet mode, we currently do not see the need for IVs larger than 64 bits, hence, this design choice was made to reduce the hardware costs of an implementation. (Grain v1 also uses 64-bit IVs.) For extreme situations, where more than 2^{64} packets need to be encrypted, Lizard should be operated in a setting that allows the use of session keys.

W.r.t. potential security implications, observe that phase 1 of the state initialization of Lizard (cf. Subsection 8.2.2) could equivalently be written as

$$B_j^0 := \begin{cases} K_j \oplus IV_j, & \text{for } j \in \{0,\ldots,63\}, \\ K_j \oplus 0, & \text{for } j \in \{64,\ldots,89\}, \end{cases}$$

$$S_i^0 := \begin{cases} K_{i+90} \oplus 0, & \text{for } i \in \{0,\ldots,28\}, \\ K_{119} \oplus 1, & \text{for } i = 29, \\ 1, & \text{for } i = 30, \end{cases}$$

which (ignoring S_{30}^0 for now, see below) can be interpreted as the loading phase of the generic Lizard-construction under the restriction that the rightmost 56 bits of the 120-bit IV (in the generic scheme with $n = 120$) are fixed to $(0,\ldots,0,1)$. In the random oracle model that was used to prove the $(2n/3)$-security of the Lizard-construction against TMD-TO attacks aiming at key recovery (see Sections 7.4 and 7.5), such a restriction of the choice of IVs in fact means a limitation of an attacker's capabilities. Hence, the security claims for the generic Lizard-construction still hold for Lizard (cf. Subsection 8.4.2). The reason for setting $S_{29}^0 := K_{119} \oplus 1$ is to avoid the "sliding property" of Grain v1 and Grain-128 that was pointed out in [DCKP08] (see Subsection 8.4.8 for further details).

Another difference compared to the generic scheme is that in Lizard, the size of the inner state is one bit larger than the size of the key. This extension by one bit is necessary because of the Grain-like, FSR-based structure of Lizard. More precisely, XORing a 120-bit key to a 120-bit inner state in phase 3 could result in a situation, where both FSRs get stuck in the all-zero state, leading to an all-zero keystream for the corresponding packet. Extending the inner state to 121 bits and setting $S_{30}^{129} := 1$ prevents this. Again, in the context of the security proof for the Lizard-construction, a TMD-TO attack on Lizard (with 120-bit keys and a 121-bit inner state) will have at least the same complexity as an attack on the generic scheme with 120-bit keys and a 120-bit inner state.

The third notable difference between the generic Lizard-construction and the state

initialization of LIZARD is the additional phase 4 (cf. Subsection 8.2.2) in LIZARD. The purpose of this phase is twofold. First, without it, due to setting $S_{30}^{129} := 1$ in phase 3, one bit of the inner state of NFSR1 would be known to an attacker during the first 31 steps of keystream generation. Though we do not consider this an immediate threat (remember that the output function of LIZARD takes 53 inner state bits as input), it is nonetheless an undesirable property that we feel should be avoided. The second motivation behind phase 4 (and, in fact, also the reason for stepping the whole KSG in phase 4 instead of only stepping NFSR1, which would have been sufficient to counter the first problem) is to avoid certain related key/IV properties, which are discussed in detail as part of Subsection 8.4.8.

8.4 Cryptanalysis

Beyond question, the essential feature of a cipher is its security. In particular, new schemes need to resist those attacks which weakened or even broke other ciphers in the past. The term *resist* here refers to a certain security level targeted by the designers, where the common approach is to require that no attack with a complexity lower than that of exhaustive key search exists. For LIZARD, we deviate from this *one-security-level-fits-it-all* approach to allow for a hardware-efficient implementation in low-cost scenarios, where certain types of attacks might be of lesser importance than, e.g., strictly limiting factors like power consumption. More precisely, LIZARD offers 80-bit security against key recovery and 60-bit security against distinguishing. Due to the key length of 120 bits, the complexity of exhaustive key search is even significantly higher than that of key recovery via TMD-TO attack (cf. Subsection 8.4.1 and Subsection 8.4.2).

Note that LIZARD is only one of many cryptographic schemes that deviate from the design paradigm *key length = security level*. Most prominently, in asymmetric cryptography, key sizes significantly larger than the claimed security level are generally accepted. A well-known example from the field of symmetric cryptography is the lightweight authentication protocol by Hopper and Blum [HB01] (commonly known as HB protocol) with its numerous variants (e.g., the HB$^+$ protocol of Juels and Weis [JW05]), some of which need key sizes of several hundreds of bits to be operated securely (see Section 3.4 and [AHM14] for further details).

Trading some security against distinguishing attacks in favor of hardware efficiency seems a plausible option to us as there are many practical scenarios where an attacker will know anyhow that he is dealing with an encrypted data stream. Moreover, the fact that there is a distinguishing attack does not mean that there are necessarily other, more practically relevant attacks of the same complexity as well. Finally, we would like to point out that for prominent lightweight block ciphers like PRESENT [BKL$^+$07] and KATAN64 [DCDK09], which both have 80-bit key size and 64-bit block size, a complete code book can be built with complexity lower than 2^{80} without considering the ciphers

broken. In the same spirit, we consider a TMD-TO-based distinguishing attack of overall attack complexity 2^{60} against LIZARD acceptable for many use cases.

In the following subsections, we will argue for several types of attacks against stream ciphers (and the Grain family in particular) why we believe that LIZARD will resist them.

8.4.1 Exhaustive Key Search

Based on experiments using the computer algebra system Magma [BCP97], the number of possible initial states (under an arbitrarily fixed IV) after phase 4 of the state initialization of LIZARD is expected to be around $2^{119.34}$, if we assume that phase 2 of the state initialization realizes a random, bijective mapping of $(B_0^0, \ldots, B_{89}^0, S_0^0, \ldots, S_{30}^0)$ to $(B_0^{128}, \ldots, B_{89}^{128}, S_0^{128}, \ldots, S_{30}^{128})$. Consequently, an attacker who knows a (sufficiently long) piece of keystream of some packet will need at most around $2^{119.34}$ key guesses to find the corresponding initial state, which then allows to generate the complete keystream of this packet. Note that in Trivium and all members of the Grain family, for an arbitrarily fixed IV, the state initialization realizes an injective mapping of the key space to the set of initial states. In our case, this property is lost due to the second key addition and setting $S_{30}^{129} := 1$ in phase 3 of the state initialization. Still, around $2^{119.34}$ possible initial states for an arbitrarily fixed IV seems more than enough, given that LIZARD is supposed to be a power-saving alternative to Grain v1, whose key space has only size 2^{80}. Moreover, only if the actual key was found by the attacker (and not just one that produces the same initial state for the given IV), he will be able to decrypt other packets as well.

In [DCKP08], an attack on Grain v1 and Grain-128 (but not Grain-128a) was introduced that allowed to reduce the cost of exhaustive key search by a factor of two by making use of what the authors call the "sliding property" of those ciphers. In Subsection 8.4.8, we explain how this sliding property was avoided for LIZARD.

8.4.2 Time-Memory-Data Tradeoff Attacks

In the previous parts of this thesis, we have already seen that the vulnerability against TMD-TO attacks like those of Babbage [Bab95] and Biryukov and Shamir [BS00] represents an inherent weakness of existing practical stream ciphers. This yields the well-known rule that for achieving l-bit security by a stream cipher (in the standard, non-LIZARD-construction-type operation mode), one has to choose an inner state length of at least $2l$ for the underlying KSG. As a consequence, modern practical stream ciphers have comparatively large inner state lengths (e.g., 288 bits for Trivium [CP05] or 160 bits for Grain v1 [HJM06]).[5]

[5]But also w.r.t. small-state stream ciphers, we have shown in Chapter 6 by means of our generic TMD-TO distinguisher for Sprout-like ciphers and the TMD-TO key recovery attack against Fruit v1 that TMD-TO attacks are still among the major threats in contemporary stream cipher design.

Also remember that, e.g., for Trivium and all members of the Grain family, not only the state update function during keystream generation but also the state initialization algorithm, which computes the initial state from a key/IV pair, is efficiently invertible (cf. Subsections 5.3.3 and 5.3.4). As a consequence, if an attacker manages to recover any inner state during keystream generation, he will also be able to recover the corresponding initial state, and, by inverting the state initialization algorithm, the underlying secret key. While computing the initial state from any of the later inner states is also possible for Lizard, the secret key cannot be computed efficiently from the initial state. This is due to the fact that Lizard represents an instantiation of the general Lizard-construction design principle as described in Chapter 7 and Subsection 8.3.5.

Hence, for Lizard, we have to treat the case of TMD-TO attacks aiming at key recovery and the case of TMD-TO attacks aiming at recovering the initial state of some packet separately. Before we go into the respective details, let us recall some terminology, which we will make extensive use of.

Definition 8.1

In this subsection, we consider the following complexities w.r.t. TMD-TO attacks against stream ciphers:

P: the preprocessing time of the attack,

T: the online time of the attack,

M: the memory required for the attack,

D: the (keystream) data required for the attack.

When we speak of the *overall complexity* of a TMD-TO attack, we refer to the maximum of the above four cost factors (including preprocessing time). Correspondingly, we assume attackers whose goal is to keep the overall complexity of their attack as low as possible.

TMD-TO Attacks aiming at Recovering the Initial State of some Packet

For stream ciphers like Trivium, Grain, and also Lizard, which work as a finite-state machines that compute keystream exclusively based on an initial state, recovering this state allows an attacker to generate the whole keystream for the underlying key/IV pair (i.e., in the case of Lizard, for one packet of length up to 2^{18} bits). Furthermore, if the cipher's state update function is efficiently invertible (as for Trivium, Grain, and Lizard), recovering one of the subsequent inner states is obviously also sufficient for a successful attack.

Let us denote by N the number of all possible inner states of the targeted KSG. The

TMD-TO attack of Babbage [Bab95] has the tradeoff curve $TM = N$ with $P = M$ and $T \leq D$ (where $T < D$ means that some of the available data is ignored during the online phase of the attack). Obviously, the best overall complexity that can be achieved here is $N^{1/2}$ by choosing $T = M = N^{1/2}$. The TMD-TO attack of Biryukov and Shamir [BS00] is based on Hellman's time-memory tradeoff attack for block ciphers [Hel80] and has the tradeoff curve $TM^2D^2 = N^2$ with $P = N/D$ and $T \geq D^2$. As, in our model, the overall complexity includes preprocessing, $P = N/D$ implies here that $N^{1/2}$ is a lower bound for the attack's overall complexity. This lower bound, however, cannot be achieved as $P = D = N^{1/2}$ would imply $T \geq N$ due to the restriction $T \geq D^2$. Note that even if we did not consider preprocessing to be part of the overall attack complexity, the original attack of Biryukov and Shamir would not have an overall complexity lower than $N^{1/2}$ due to the condition $T \geq D^2$. Because choosing $T < N^{1/2}$ would imply $D < N^{1/4}$ and, in order to satisfy $TM^2D^2 = N^2$, also $M > N^{1/2}$.

In [BS00], Biryukov and Shamir also discuss a technique called *BSW-sampling*, which was originally used by Biryukov, Shamir, and Wagner in [BSW01] to attack the GSM cipher A5/1. While BSW-sampling allows to relax the restriction $T \geq D^2$ in the above attack, the tradeoff curve $TM^2D^2 = N^2$ and the relation $P = N/D$ remain unchanged. Hence, if one considers precomputation to be part of the overall attack complexity (as we do), even the use of BSW-sampling does not allow for attacks with overall complexity lower than $N^{1/2}$. Moreover, the applicability of BSW-sampling is highly cipher specific (see [BS00] for further details). The designers of Grain v1 state:

> "The sampling resistance of $h(x)$ is reasonable: This function does not become linear in the remaining variables by fixing less than 3 of its 5 variables. Similarly, the variables occurring in monomials of $g(x)$ are sufficiently disjoint. Hence the resulting sampling resistance is large, and thus time/memory/data tradeoff attacks are expected to have complexity not lower than $O(2^{80})$." [HJM06]

Note that the output function of Lizard is even much more complex than those of Grain v1 or Grain-128a and, e.g., does not become linear in the remaining variables by fixing less than 13 of its 53 variables. Moreover, the variables occurring in monomials of f_2 of Lizard (which corresponds to g in the Grain family) are also sufficiently disjoint. Additionally, in contrast to the Grain family, both FSRs of Lizard are now nonlinear. Consequently, we believe that, like Grain, Lizard is sufficiently resistant w.r.t. sampling.[6]

[6]Note that, after the presentation of Lizard at FSE 2017, Maitra et al. published [MSS+17], in which a BSW-sampling TMD-TO attack against Lizard is suggested. In Subsection 8.4.11, we will discuss these results in further detail and conclude that, as expected, Lizard does not perform worse than Grain v1 w.r.t. such attacks. Note further that the attack of Maitra et al. has a precomputation complexity significantly above the cipher's targeted security level and, hence, does not violate any of our security claims.

We will now briefly describe the two extreme cases of a TMD-TO inner state recovery attack against LIZARD, both of which have overall attack complexity of (at least) 2^{60}. In the first variant, an attacker knows 121 keystream bits of each of approx. 2^{60} keystream packets. A TMD-TO attack like that of Babbage or Biryukov and Shamir will then allow him to learn one of the corresponding 2^{60} initial states of these keystream packets, based on which he will be able to generate the complete keystream of this packet. Note, however, that the attacker has no control about *which one* of these 2^{60} keystream packets he will eventually be able to generate completely. Moreover, the attacker must know at least 121 bits of the respective keystream packet beforehand, i.e., he will not be able to generate a keystream packet that was not part of his TMD-TO attack. In the second extreme case, the attacker tries to keep the set of keystream packets used in his TMD-TO attack as small as possible. This is achieved if he knows $2^{18} - 1$ (i.e., one bit less than the maximum packet size of LIZARD) keystream bits of each of approx. 2^{42} keystream packets. Then, a TMD-TO attack (still with overall attack complexity 2^{60}) will allow him to learn one of the corresponding 2^{42} initial states of these keystream packets, based on which he will be able to generate the complete keystream of this packet. Again, the attacker has no control about *which one* of these 2^{42} keystream packets he will eventually be able to generate completely and, to launch his attack, he must know $2^{18} - 1$ keystream bits of each of those 2^{42} packets. In particular, the attacker will only learn a single keystream bit that he didn't know beforehand.

Obviously, the state recovery attack with overall complexity of (at least) 2^{60} described in the previous paragraph would also allow an attacker to distinguish LIZARD from a truly random function, i.e., the security against TMD-TO-based distinguishing attacks is not increased by implementing the generic LIZARD-construction. The reason why we do not claim 60.5-bit security against distinguishing despite LIZARD's 121-bit state is the following: An attacker could also target the inner state at the end of phase 3 of the state initialization using a TMD-TO attack on the basis of 2^{60} keystream data points obtained from 2^{60} different IVs. Such an attack could be realized with overall complexity 2^{60} as, due to setting $S_{30}^{129} := 1$ in phase 3, there are actually only 2^{120} states possible at $t = 129$.

Before we move on to discussing TMD-TO attacks that aim at recovering the secret key, let us remind that actually all current small-state stream ciphers are susceptible to TMD-TO distinguishing attacks with complexity significantly below that of key recovery. In particular, we have shown in Subsection 6.3.1 that, e.g., the Sprout-like ciphers Plantlet and Fruit v1, both of which use 80-bit keys, can be distinguished with complexities 2^{61} and 2^{50}, respectively. For situations where this is not acceptable, we refer the reader to our new design ideas presented in Sections 6.5 and 9.3.

TMD-TO Tradeoff Attacks aiming at Key Recovery

As pointed out in Subsection 8.3.5, LIZARD's state initialization represents an instantiation of the generic LIZARD-construction introduced in Chapter 7, which, in our case, offers

80-bit security against key recovery via TMD-TO attacks. It is important to keep in mind that the attacker model underlying the provable security of the Lizard-construction (and also all other security claims w.r.t. Lizard) treats any precomputations as part of the overall attack. In particular, our claims do not conflict with the TMD-TO key recovery attack of Dunkelman and Keller presented in [DK08]: Let P, T, M, D be defined as introduced at the beginning of this subsection with the additional condition that all D keystream data points have to be obtained on the basis of different IVs (i.e., D keystream prefixes of sufficient length, generated under the same secret key but D different IVs). Moreover, let K denote the number of all possible keys and let V denote the number of all possible IVs. The key recovery attack of Dunkelman and Keller has the tradeoff curve $TM^2D^2 = (KV)^2$ with $D \leq V$ and $P = (KV)/D$.[7] In the case of Lizard, where $K = 2^{120}$ and $V = 2^{64}$, this means that $P = 2^{184}/D$ with $D \leq 2^{64}$. Hence, the key recovery attack of Dunkelman and Keller would have a precomputation complexity greater or equal exhaustive key search and thus way larger than the 80-bit security against key recovery claimed for Lizard.

Finally, note that even when ignoring precomputation time (which we do not in our security model), the attack of Dunkelman and Keller would not break Lizard's 80-bit security against key recovery. More generally, any TMD-TO attack with tradeoff curve $TM^2D^2 = (KV)^2$ and restriction $D \leq V$ will have an *online complexity* larger or equal 2^{80} in the case of Lizard. Because when choosing $D = 2^{64}$ (i.e., as large as possible), one obtains $TM^2 2^{2 \cdot 64} = (2^{120} \cdot 2^{64})^2$ or $TM^2 = 2^{240}$, for which $T < 2^{80}$ and $M < 2^{80}$ cannot hold at the same time. Similarly, any TMD-TO attack with tradeoff curve $TM = KV$ will have an online complexity larger or equal $2^{(120+64)/2} = 2^{92}$ for Lizard, even without any additional restrictions on the tradeoff parameters.

8.4.3 Correlation Attacks, Linear Approximations

The designers of Grain v1 state:

> "Due to the statistical properties of maximum-length LFSR sequences, the bits in the LFSR are (almost) exactly balanced. This may not be the case for a NFSR when it is driven autonomously. However, as the feedback $g(x)$ is xored with an LFSR-state, the bits in the NFSR are balanced. Moreover, recall that $g(x)$ is a balanced function. Therefore, the bits in the NFSR may be assumed to be uncorrelated to the LFSR bits." [HJM06]

The same, in fact, applies to Lizard, where (the maximum-length FSR) NFSR1 corresponds to Grain's LFSR, NFSR2 corresponds to Grain's NFSR, and f_2 corresponds to g

[7] The relation $P = (KV)/D$ is not given explicitly in [DK08] but can be derived from the description of the attack algorithm. Also note that, in contrast to the aforementioned (inner state recovery) attack of Biryukov and Shamir, the condition $T \geq D^2$ does not have to be satisfied in the (key recovery) attack of Dunkelman and Keller.

(cf. Subsection 8.2.1 and Subsections 8.3.1 and 8.3.2).

In [MJSC16], Méaux et al. point out the importance of "good balancedness, nonlinearity and resiliency properties" of the filtering function in order to withstand correlation attacks [Sie85] and fast correlation attacks [MS89]. As explained in Subsection 8.3.3, LIZARD features a rather heavy output function to compensate for the smaller inner state compared to the original Grain family. It is defined over 53 variables and has nonlinearity 4 476 506 321 453 056, whereas the output function of Grain v1 is defined over 12 variables and has nonlinearity 1536 and the output function of Grain-128a is defined over 17 variables and has nonlinearity 61 440. The resiliency of LIZARD's output function is 8 compared to 7 for that of Grain v1 and Grain-128a, respectively.[8]

In [BGM06], Berbain, Gilbert, and Maximov present an attack on Grain v0 that combines linear approximations of the NFSR's feedback function and of the output function in order to recover the initial state of the LFSR given a sufficient amount of keystream bits. Once the initial state of the LFSR is known, each keystream bit can be expressed as a linear function in one or two (depending on the LFSR state) bits of the internal bitstream of the NFSR, which allows to efficiently recover the initial state of the NFSR using "a technique which consists of building chains of keystream bits" [BGM06]. Two variants (differing in the LFSR derivation method) of this key recovery attack[9] against Grain v0 are described in [BGM06], the best of which has time complexity 2^{43}, memory complexity 2^{42}, and data (i.e., keystream) complexity 2^{38}.

As possible countermeasures, Berbain, Gilbert, and Maximov proposed the following modifications [BGM06]: "Introduce several additional masking variables from the NFSR in the keystream bit computation", "replace g by a 2-resilient function", "[m]odify the filtering function h in order to make it more difficult to approximate", and "[m]odify the function g and h to increase the number of inputs". The Grain designers revised their eSTREAM submission accordingly (in particular, seven bits from the NFSR were added linearly to the output function) and suggested Grain v1 [HJM06], for which the authors of [BGM06] acknowledge that "[t]his novel version of Grain appears to be much stronger and is immune against the statistical attacks presented in this paper".

For Grain-128a, the feedback function g of the NFSR was constructed with the above attack in mind. The designers state: "The best linear approximation of g is of considerable

[8] The resiliency of the output functions of Grain v1 [HJM06] and Grain-128a [gHJM11] is not specified explicitly in the respective papers. However, these values can be computed rather easily: For Grain v1, the designers state that the function h, which is part of the full output function, is balanced and correlation immune of the first order. Thus, h is 1-resilient (but not 2-resilient as can be checked easily). By adding seven linear monomials, whose tap positions are disjoint from those used in h, the resiliency of the full output function of Grain v1 increases to 7. (This can be shown using lemmata 3 and 4 from [MJSC16].) In Grain-128a, the function h is unbalanced and, hence, has resiliency -1 according to Definition 5 in [MJSC16]. By adding eight linear monomials, whose tap positions are disjoint from those used in h, the resiliency of the full output function of Grain-128a increases to 7.

[9] Remember that for the original Grain family, initial state recovery and key recovery are equivalent due to the efficiently invertible state initialization algorithm (cf. Subsection 5.3.4).

interest, and for it to contain many terms, we need the resiliency of the function g to be high. We also need a high nonlinearity in order to obtain a small bias." [gHJM11] As a consequence, g was chosen such that it has nonlinearity 267 403 264 and resiliency 4.

As explained in Subsection 8.3.2, the feedback function f_2 of NFSR2 in Lizard squeezes the taps of g of Grain-128a in a way that preserves its balancedness, resiliency, and nonlinearity. Moreover, in accordance with the above suggestions from [BGM06] and the construction principle underlying g of Grain-128a (see previous paragraph), the output function of Lizard has more than three times as many inputs, a much higher nonlinearity, and a higher resiliency than those of Grain v1 and Grain-128a (cf. values at the beginning of this subsection) in order to strengthen it against linear approximations.

Note that even if the initial state of NFSR1 of Lizard (corresponding to the LFSR in the original Grain family) could be recovered by means of linear approximation, the subsequent NFSR initial state recovery procedure for Grain v0 described in Section 5 of [BGM06] would not work against NFSR2 of Lizard. This is due to the fact that the output function of Lizard will still contain at least ten nonlinear monomials (of degrees $2, \ldots, 7$) over bits from NFSR2 if the content of NFSR1 should be known (cf. Subsection 8.3.3), making the aforementioned chaining technique from [BGM06] not applicable any more.

8.4.4 Algebraic Attacks

As pointed out previously, unlike for the members of the original Grain family, the state initialization algorithm of Lizard is not efficiently invertible. Hence, we would actually have to differentiate here between algebraic attacks aiming at key recovery and algebraic attacks trying to recover the initial state of some keystream packet. However, an attempt to express the observed keystream bits as functions of the 120 key bits and then solve the corresponding system of equations would require to include all state transitions down to $t = 0$. Given that both FSRs are nonlinear and considering the high algebraic degree of the output function (which is used as part of the state update in phase 2 of the state initialization), this is clearly more complex than expressing the observed keystream bits as functions of the 121 bits of the initial state at $t = 257$ and then solving the corresponding system of equations. Consequently, for the remainder of this subsection, we will focus on algebraic attacks that try to recover the initial state of some keystream packet.

First of all, note that, to the best of our knowledge, no successful (i.e., having complexity lower than 2^{80} (Grain v1) or 2^{128} (Grain-128a)) algebraic attacks that can recover arbitrary initial states for Grain v1 or Grain-128a have been reported so far.[10] Due to the smaller

[10]The currently best result seems to be an algebraic attack by Berbain, Gilbert, and Joux against a modified version of Grain-128, which requires 2^{115} computations and 2^{39} keystream bits [BGJ09]. They point out, however, that "[t]his attack is not applicable to the original Grain-128". Moreover, note that the required amount of keystream bits (belonging to a single initial state) would not be available for Lizard due to the maximum packet size of 2^{18} bits.

inner state of LIZARD, the number of variables of the corresponding system of equations in such an attack would now in fact be lower. This, however, is compensated for by the larger degree of the output function, which is now 7 as compared to 3 for Grain v1 and Grain-128a. As pointed out in Subsection 8.3.3, LIZARD's output function builds on the construction scheme introduced in [MJSC16] and has algebraic immunity of at least 7 and fast algebraic immunity of at least 8. In addition, now both FSRs are nonlinear and NFSR1, which corresponds to the LFSR of the original Grain family, has algebraic degree 4. Based on these properties, we expect that algebraic attacks against LIZARD will not be efficient (i.e., will not have complexity lower than 2^{80}, which is that of TMD-TO-based key recovery as described in Subsection 8.4.2).

8.4.5 Guess-and-determine Attacks

The output function of LIZARD depends on 53 variables, compared to 12 in Grain v1 and 17 in Grain-128a. As pointed out in Subsection 8.3.3, when guessing the shorter NFSR1, the output function still depends on at last 43 variables and has the following worst-case security properties: nonlinearity 4 317 411 672 064, resiliency 7, algebraic immunity at least 7, and fast algebraic immunity at least 8. For comparison, the full output function of Grain v1 (Grain-128a) has nonlinearity 1536 (61 440) and algebraic degree 3 (3). Thus an algebraic attack on NFSR2 similar to the one in [BGJ09] will have large enough complexity.

Note that guessing the content of NFSR1 at $t = 0$ corresponds to guessing the bits $K_{90}, \ldots, K_{119}$ of the secret key. An attack that guesses the content of NFSR1 at $t = 257$ (i.e., the respective part of the initial state) in order to recover the full initial state will not automatically reveal the secret key because of the use of the LIZARD-construction design principle, due to which the state initialization of LIZARD (in contrast to the original Grain family) is not efficiently invertible.

More precisely, when considered individually, phase 2 and phase 4 of the state initialization are obviously efficiently invertible. In particular, the knowledge of $(S_0^{257}, \ldots, S_{30}^{257})$ (i.e., the NFSR1-related part of the initial state) is already sufficient to efficiently recover $(S_0^{129}, \ldots, S_{30}^{129})$ (i.e., the content of NFSR1 after the second key addition in phase 3). However, when treated as a whole, phases 1–3 (i.e., the state transition from $t = 0$ to $t = 129$) cannot be inverted efficiently.

8.4.6 Conditional Differentials, Cube Distinguishers

Lehmann and Meier studied the security of Grain-128a against dynamic cube attacks and differential attacks. They came to the following conclusion:

> "To analyse the security of the cipher, we study the monomial structure and use high order differential attacks on both the new and old versions. The comparison of symbolic expressions suggests that Grain-128a is immune

against dynamic cube attacks. Additionally, we find that it is also immune against differential attacks as the best attack we could find results in a bias at round 189 out of 256." [LM12]

Lizard has 128 rounds in phase 2 of the state initialization, where the Grain-like mixing is performed as described in Subsection 8.2.2 and further explained in Subsection 8.3.5. On top of that, the key is added again in phase 3 of the state initialization and, finally, the KSG is stepped 128 additional times (now in keystream generation mode) before the first keystream bit is output.

Note that the inner state of Lizard (121 bits) is smaller than that of Grain v1 (160 bits) and significantly smaller than that of Grain-128a (256 bits), whereas the output function is more dense. It depends on 53 variables as compared to 12 in Grain v1 and 17 in Grain-128a. The output function of Lizard also has more nonlinear monomials (13) than Grain v1 (8) and Grain-128a (5). Moreover, now both FSRs are nonlinear.

The combination of a smaller state and a more dense output function causes a faster diffusion of differentials and of the monomial structure for Lizard. Therefore, we expect that Lizard is at least as resistant against differential attacks and dynamic cube attacks as Grain v1 and Grain-128a, which seem to be already sufficiently secure in that respect.

8.4.7 IV Collisions

A consequence of setting $S_{30}^{129} := 1$ in phase 3 of the state initialization of Lizard (cf. Subsection 8.2.2) is that, in contrast to the original Grain family, what we call *IV collisions* can now occur, i.e., two key/IV pairs (K, IV) and (K, IV') with $IV \neq IV'$ can map to the same initial state and, hence, result in identical keystream packets. The implications of this are twofold.

First, an attacker could try to exploit this fact to launch a distinguishing attack. More precisely, if the corresponding oracle answers with identical, sufficiently long keystream packets for two different key/IV pairs (K, IV) and (K, IV'), where $IV \neq IV'$, then the attacker can distinguish the pseudo-random from the random the scenario. However, as we will prove at the end of this subsection, it holds that:

Theorem 8.1

The number of oracle queries necessary for finding an IV collision for Lizard with probability greater than $1/2$ exceeds 2^{60}.

In particular, the complexity of this attack is not lower than that of the generic TMD-TO distinguishing attack discussed in Subsection 8.4.2.

The second implication of potential IV collisions is that an attacker might get hold of a (partial) keystream packet x generated for the key/IV pair (K, IV) and a partial keystream packet y generated for (K, IV'), where $IV \neq IV'$. If $|x| > |y| > 121$ and y

fully coincides with the corresponding part of x, then the attacker can conclude that, with high probability, both (partial) keystream packets x and y where generated based on the same initial state. Consequently, he would now know a larger piece (i.e., x) of the keystream packet for (K, IV'). However, the expected number of (partial) keystream packets needed for finding such a collision is larger than 2^{60} and the attacker would have no choice w.r.t. *for which one* of these packets he would obtain further keystream bits. Overall, this scenario has at least the complexity of the TMD-TO-based initial state recovery attack described in Subsection 8.4.2 and seems even less realistic.

Proof of Theorem 8.1

In order to show that the number of oracle queries necessary for finding an IV collision with probability $> 1/2$ exceeds 2^{60} as claimed in Theorem 8.1, we first need to observe that, for an arbitrarily fixed key, each initial state can be the result of at most two different IVs.

Let (K, IV) and $(K, \widetilde{IV})$ with $IV \neq \widetilde{IV}$ be an IV collision, i.e., the corresponding initial states

$$\left(\left(B_0^{257}, \ldots, B_{89}^{257}\right), \left(S_0^{257}, \ldots, S_{30}^{257}\right)\right)$$

for (K, IV) and

$$\left(\left(\widetilde{B}_0^{257}, \ldots, \widetilde{B}_{89}^{257}\right), \left(\widetilde{S}_0^{257}, \ldots, \widetilde{S}_{30}^{257}\right)\right)$$

for $(K, \widetilde{IV})$ coincide. Then

$$\left(\left(B_0^{129}, \ldots, B_{89}^{129}\right), \left(S_0^{129}, \ldots, S_{30}^{129}\right)\right) = \left(\left(\widetilde{B}_0^{129}, \ldots, \widetilde{B}_{89}^{129}\right), \left(\widetilde{S}_0^{129}, \ldots, \widetilde{S}_{30}^{129}\right)\right)$$

must hold, as phase 4 of the state initialization algorithm of Lizard (cf. Subsection 8.2.2) implements a bijective mapping over the set of all inner states. As the key is (arbitrarily) fixed, we also know that

$$\left(B_0^{128}, \ldots, B_{89}^{128}\right) = \left(\widetilde{B}_0^{128}, \ldots, \widetilde{B}_{89}^{128}\right)$$

together with either

$$\left(S_0^{128}, \ldots, S_{30}^{128}\right) = \left(\widetilde{S}_0^{128}, \ldots, \widetilde{S}_{30}^{128}\right)$$

or

$$\left(S_0^{128}, \ldots, S_{30}^{128}\right) = \left(\widetilde{S}_0^{128}, \ldots, \widetilde{S}_{30}^{128} \oplus 1\right)$$

must hold.

The case

$$\left(\left(B_0^{128},\ldots,B_{89}^{128}\right),\left(S_0^{128},\ldots,S_{30}^{128}\right)\right)=\left(\left(\widetilde{B}_0^{128},\ldots,\widetilde{B}_{89}^{128}\right),\left(\widetilde{S}_0^{128},\ldots,\widetilde{S}_{30}^{128}\right)\right),$$

however, is impossible due to the fact that, like phase 4, phase 2 of the state initialization algorithm also implements a bijective mapping over the set of all inner states and, as $IV \neq \widetilde{IV}$,

$$\left(\left(B_0^{0},\ldots,B_{89}^{0}\right),\left(S_0^{0},\ldots,S_{30}^{0}\right)\right)\neq\left(\left(\widetilde{B}_0^{0},\ldots,\widetilde{B}_{89}^{0}\right),\left(\widetilde{S}_0^{0},\ldots,\widetilde{S}_{30}^{0}\right)\right).$$

This now also shows immediately that there cannot be a third key/IV combination $(K,\widehat{IV})$ with $IV \neq \widehat{IV} \neq \widetilde{IV} \neq IV$ that results in the same initial state as the IV collision (K, IV) and $(K,\widetilde{IV})$, because

$$S_{30}^{128} \neq \widetilde{S}_{30}^{128} \neq \widehat{S}_{30}^{128} \neq S_{30}^{128}$$

is a contradiction.

How many oracles queries (K, IV^i), $i = 1,\ldots$, with $IV^i \neq IV^j$ for $i \neq j$, are now necessary in order to find an IV collision with probability 1/2? Based on the above observations, we can model this as a simple urn problem. The urn contains 2^{120} different pairs of balls (i.e., 2^{121} balls in total), where each pair of balls corresponds to a different pair of inner states at $t = 128$ fulfilling

$$\left(\left(B_0^{128},\ldots,B_{89}^{128}\right),\left(S_0^{128},\ldots,S_{30}^{128}\right)\right)=\left(\left(\widetilde{B}_0^{128},\ldots,\widetilde{B}_{89}^{128}\right),\left(\widetilde{S}_0^{128},\ldots,\widetilde{S}_{30}^{128}\oplus 1\right)\right).$$

The question is how many balls need to be drawn (randomly and without putting them back) in order to have at least one pair with probability 1/2?

The probability p_i that at attempt $i = 1,\ldots$, a pair is completed, can be upper bounded by

$$p_i \leq \frac{i-1}{2^{121}-(i-1)}.$$

Consequently, the probability $Pr[k]$ that after k attempts, at least one pair has been found, can be upper bounded by

$$Pr[k] \leq \sum_{i=1}^{k} p_i = \sum_{i=1}^{k} \frac{i-1}{2^{121}-(i-1)} < k \cdot \frac{k}{2^{121}-k}.$$

For $k \leq 2^{60}$, we have $Pr[k] < 1/2$, which proves the claim.[11] □

[11] Note that, using the more precise estimate $\sum_{i=1}^{k} \frac{i-1}{2^{121}-(i-1)} \leq \int_0^k \frac{x}{2^{121}-x}\,\mathrm{d}x$, one can even show that $Pr[k] \leq 1/2$ for all $k \leq 2^{60.5}$.

8.4.8 Related Key(/IV) Attacks, Slide Attacks

In [Küç06], Küçük first pointed out a *sliding property* of the state initialization of Grain v1, which was later formally published by De Cannière, Küçük, and Preneel in [DCKP08] as: "For a fraction of $2^{-2 \cdot n}$ of pairs (K, IV), there exists a related pair (K^*, IV^*) which produces an identical but n-bit shifted key stream." In the same paper, the authors describe how this property can be exploited to speed up exhaustive key search for Grain v1 (and also for Grain-128) by a factor of two. In addition, they also suggest a related-key slide attack, for which they note: "As is the case for all related key attacks, the simple attack just described is admittedly based on a rather strong supposition." [DCKP08] As a reaction, the designers of Grain-128a changed the 22-bit constant $(1, \ldots, 1)$ that was used in state initialization of Grain-128 to $(1, \ldots, 1, 0)$.

For LIZARD, a speed-up of exhaustive key search by a factor of two would actually be tolerable due to the key length of 120 bits together with the fact that we aim for 80-bit security against key recovery. Still, we consider such a sliding property undesirable as it might pave the way for other attacks. To avoid it, we set $S_{29}^0 := K_{119} \oplus 1$ in phase 1 of the state initialization (cf. Subsection 8.3.5). As a result, for a key/IV pair (K, IV), a related key/IV pair (K^*, IV^*) in the sense of [DCKP08] would have to satisfy $K_{118}^* = K_{119} \oplus 1$. This, however, would then lead to a bad (i.e., inverted) key bit being added in phase 3 of the state initialization, where $K_{118}^* = K_{119}$ must hold for the attack to work. Note that without inverting K_{119} in phase 1 of the state initialization, LIZARD would in fact suffer from a variant of the sliding property, despite the second key addition in phase 3.

Another undesirable related key/IV property, which would arise out of using the generic LIZARD-construction with a Grain-like KSG, is avoided by phase 4 of the state initialization of LIZARD. More precisely, given some key/IV pair (K, IV) together with the corresponding keystream packet, an attacker might know (or suspect) for some other keystream packet that it was generated under the related key/IV pair (K', IV'), where $K_2' = K_2 \oplus 1$, $IV_2' = IV_2 \oplus 1$, $K_i' = K_i$ for $0 \leq i \leq 79$, $i \neq 2$, and $IV_i' = IV_i$ for $0 \leq i \leq 63$, $i \neq 2$. Without phase 4 (i.e., if z_{129} instead of z_{257} was the first keystream bit used for plaintext encryption), the attacker could immediately conclude that the first 11 keystream bits generated for (K', IV') are identical to the first 11 keystream bits $z_{129}, \ldots, z_{139}$ generated for (K, IV). This is due to the fact that if $K_i \oplus IV_i = K_i' \oplus IV_i'$ for $i = 0, \ldots, 63$ and $K_i = K_i'$ for $i = 64, \ldots, 119$, then the corresponding inner states of LIZARD will be identical after phase 2 of the state initialization. In the above example, where key bit K_2 and IV bit IV_2 are flipped for K_2' and IV_2', respectively, this would lead to identical inner states after the second key addition at $t = 129$ except the bit B_2^{129} of NFSR2, which would be flipped now as well. However, it takes ten further steps of the KSG for this information to reach a tap of the output function. While the described scenario might seem rather far-fetched, we still wanted to avoid this kind of nonrandom behavior in order to thwart other attacks that might make use of it. Moreover, as pointed out in Subsection 8.3.5, phase 4 also became necessary due to setting $S_{30}^{129} := 1$ in phase 3.

8.4.9 Weak Key/IV Pairs

In [ZW09], Zhang and Wang introduced the notion of *weak key/IV pairs* for the Grain family of stream ciphers. They use such pairs, which lead to an all-zero initial state of the LFSR, to mount distinguishing attacks and initial state recovery attacks.[12] The designers of Grain-128a point out: "We note that the IV is normally assumed to be public, and that the probability of using a weak key/IV pair is 2^{-128}. Any attacker guessing this to happen and then launching a rather expensive attack, is much better off just guessing a key." [gHJM11] For Grain v1, which has 2^{64} weak key/IV pairs among a total of 2^{144} key/IV pairs, the corresponding probability would be 2^{-80}, leading to a similar conclusion.

In analogy to the definition of Zhang and Wang, weak key/IV pairs for Lizard would lead to an all-zero initial state of NFSR1. This, however, is impossible due to setting $S_{30}^{129} := 1$ in phase 3 of the state initialization (cf. Subsection 8.2.2) together with the fact that, after phase 3, the only input to the maximum-length FSR NFSR1 comes from its own feedback function.

Note that without setting $S_{30}^{129} := 1$ in phase 3 of the state initialization, there would have been about 2^{153} weak key/IV pairs out of 2^{184} total key/IV pairs for Lizard, leading to a probability of 2^{-31} for using a weak pair. Consequently, without setting $S_{30}^{129} := 1$ in phase 3 of the state initialization, attacks based on weak key/IV pairs might have posed a real threat to Lizard.

8.4.10 BDD-based Attacks

In [Kra02], Krause introduced the idea of using binary decision diagrams (BDDs) to attack LFSR-based stream ciphers like A5/1 of the GSM standard or E_0 of Bluetooth. Stegemann later showed in [Ste07a], how this approach can be transferred to NFSR-based stream ciphers like Trivium and Grain. In contrast to TMD-TO attacks or correlation attacks, which potentially require a lot of keystream or ciphertext data, BDD attacks are *short-keystream attacks* in the sense that only the information-theoretic minimum of keystream bits (i.e., often only few more than n bits of keystream for a KSG of inner state length n) is required to recover the corresponding initial state. As pointed out in Subsection 8.4.2, for Trivium and all members of the Grain family, an attack that recovers the initial state is equivalent to an attack that recovers the underlying secret key as the state initialization of these stream ciphers is efficiently invertible (cf. Subsections 5.3.3 and 5.3.4, respectively). For Lizard, this is not the case due to the use of the Lizard-construction design principle. Hence, for each keystream packet produced by Lizard, a separate BDD attack would be required to recover the corresponding initial state, which would then allow to generate the remaining, unknown keystream bits of that

[12] Keep in mind that for all members of the original Grain family, initial state recovery is equivalent to key recovery as pointed out in Subsection 8.4.2.

particular packet.

While we are currently not aware of any BDD attack faster than exhaustive key search against any member of the Grain family, the major design consequence of the BDD-related cryptanalytic results that Stegemann obtained for Grain-like stream ciphers is that the maximum number of what he calls *active monomials* of the feedback functions and the output function should be as large as possible (see [Ste07a] for further details). In the setting of Stegemann, for Grain v1, the maximum number of active monomials would be 0 for the LFSR, 6 for the NFSR, and 5 for the output function. For Grain-128a, the maximum number of active monomials would be 0 for the LFSR, 3 for the NFSR, and 3 for the output function. In comparison, for LIZARD, the maximum number of active monomials would be 21 for NFSR1, 3 for NFSR2, and 10 for the output function a. Consequently, we expect that, despite the smaller inner state, LIZARD will also perform sufficiently well against BDD attacks.

Note that in Section 9.4 of the chapter *Future Research Directions*, we will treat the concept of BDD-based cryptanalysis in further detail.

8.4.11 External Cryptanalysis

Since the presentation of LIZARD at FSE 2017, several 'external' cryptanalytic results have appeared, which we are now going to summarize. But before we do so, let us point out that none of these interesting findings conflict with our security claims for LIZARD.

In [BICG17], Banik et al. describe two strategies to generate key/IV pairs leading to identical keystreams. Their respective observations, however, do not translate into attacks with a complexity lower than our security claims made for LIZARD. But, in addition, the authors also suggest a distinguisher against (full) LIZARD and a key recovery attack against a round-reduced version. For their distinguisher, they state that "[t]he process takes around $2^{51.5}$ random IV encryptions (with encryption required to produce 2^{18} keystream bits) and around $2^{76.6}$ bits of memory." Note here that as each of those "random IV encryptions" consists in the generation of 2^{18} keystream bits, this would translate into a data complexity of $2^{69.5}$ in terms of the classical TMD-TO attacks. Moreover, the distinguisher of Banik et al. is exclusively based on finding collisions in keystreams generated under the same key but different IVs (similar to what we describe in Subsection 6.3.1) and, hence, takes a 'detour' that is actually not necessary for LIZARD (but for Sprout-like ciphers). More precisely, as already acknowledged in our original cipher specification [HKM17b] (see also Subsection 8.4.2), classical TMD-TO attacks like those of Babbage [Bab95] or Biryukov and Shamir [BS00] can be applied directly to LIZARD. As those attacks do not underlie the additional assumptions that attacks like that of Banik against Sprout (cf. Subsection 6.3.1) and that of Banik et al. against LIZARD require, and as the overall complexity of these classical attacks (time $2^{60.5}$, data $2^{60.5}$, memory $2^{60.5} \cdot 121$) is also better than that of Banik et al., it is unclear to what extent the distinguisher presented in [BICG17] is actually relevant for the security of

Lizard. Very interesting, however, is their key recovery attack against round-reduced Lizard (226 out of 256 rounds), which is presented in the same paper and uses the fact that, for an arbitrarily fixed key, there are on average 2^6 IV pairs which lead to identical keystreams. It should be noted, however, that in a first version of their paper (i.e., [BI17]), where the attack targeted 223 (instead of now 226) of the full 256 rounds, Banik and Isobe pointed out that it is "very difficult to extend the attack" to a higher number of rounds and that, in consequence, "any attack under 2^{80} computations seems infeasible." It thus seems questionable whether the remaining gap of 30 rounds can be closed within the complexities set by the security claims of the cipher. To understand this conjecture, it is important to know that the attack of Banik et al. is actually independent of the 128 rounds of phase 4 of Lizard's state initialization algorithm. I.e., the 'quality' of the attack is solely determined by the number of rounds of phase 2 that can be coped with. Given that the 128 Grain-like mixing steps of phase 2, which take place between the two key additions, can be considered 'the heart of Lizard's state initialization', it seems quite some way to go to extend Banik et al.'s attack from currently 98 to the full 128 rounds. Finally, note that Banik et al. state in the abstract of their paper that their "results do not affect the security claims of the designers." [BICG17]

In [MSS+17], Maitra et al. present a TMD-TO attack against Lizard. While, in the paper's conclusion, the authors state that "our observation does not imply breaking Lizard", their results are very interesting nonetheless. More precisely, they mount a technique called *conditional BSW-sampling*, where the term *conditional* reflects the "need to fix a few state bits with a specific pattern". The authors' goal is to recover some inner state during the online phase. Consequently, as Lizard's state initialization algorithm is designed according to the generic Lizard-construction, this attack does not lead to key recovery. Moreover, as Lizard is operated in packet mode with a maximum packet size of 2^{18} bits, the effects of inner state recovery are further limited as described in Subsection 8.4.2. In particular, the online complexity $T = M = D = 2^{54}$ of the attack implies that at least 2^{36} keystream packets are required as part of *data* and, similar to our own example in Subsection 8.4.2, the attacker has no control about for *which one* of these packets he will eventually obtain some inner state. Finally and most importantly, the preprocessing complexity of the attack of Maitra et al. is $P = 2^{67}$ and, hence, does not violate our claim of 60-bit security against distinguishing for Lizard in the original cipher specification (see [HKM17b] and Subsection 8.4.2), as we clearly refer to the *overall complexity* of any attack as the relevant limit. In our opinion, the attack of Maitra et al. also confirms our original conjecture that "we believe that, like Grain, Lizard is sufficiently resistant w.r.t. sampling" (cf. Subsection 8.4.2). This conclusion can be drawn from the fact that, even while the eSTREAM contest was still running, a similar BSW-sampling-based TMD-TO attack was applied against Grain v1 by Bjørstad [Bjø08] in 2008, having the complexities $P = 2^{106.5}$, $T = 2^{71}$, $M = 2^{71}$, $D = 2^{53.5}$. Given that the attack of Bjørstad targets a cipher with an inner state of 160 bits (and, hence, a security level of 80 bits w.r.t. inner state recovery), whereas Lizard, as a small-state

cipher, has only a 121-bit state, scaling down the respective complexities shows that Lizard does not perform significantly worse than Grain v1 here. And remember that, despite the results of Bjørstad, Grain v1 was finally selected as (and still is) a member of the eSTREAM portfolio.

In [SSMC17], Siddhanti et al. present a *differential fault analysis (DFA)* attack on Grain v1, ACORN v3, and Lizard. In a nutshell, this type of attack is based on inducing (by technical means such as electromagnetic or laser beam injection) faults (i.e., bit flips) in the inner state of the cipher. By restarting the cipher with the same key/IV combination and comparing correct and 'faulty' keystreams, the attacker seeks to deduce information about the correct secret inner state. Note that we (just like most other cipher designers) do not make any claims about resistance against DFA in our cipher specification. The reason for doing so is that, similar to side-channel resistance, protection against DFA is usually considered a problem which has to be taken care of by technical means or, after the 'core cipher' has been designed, added on top in the form of established algorithmic modifications. For example, in [PMK+11] a side-channel resistant, serialized implementation of the lightweight block cipher PRESENT [BKL+07] is suggested, which, depending on the level of resistance, requires between 2282 GE and 3582 GE of chip area (cf. Subsection 2.3.3), as compared to an unprotected serialized implementation for only 1111 GE. An implementation of PRESENT which is not only side-channel but also fault resistant, is suggested in [CN17] and comes at enormous hardware costs of 45 843 GE. Nevertheless, for the sake of completeness, we still want to briefly summarize the findings of Siddhanti et al. here. Like for Grain v1, the authors require 5 faults to successfully recover the targeted secret inner state of Lizard. They acknowledge themselves, however: "As mentioned before, we can only solve for the secret state and not for the secret key in case of Lizard. However, we can obtain the secret key once the secret state is known in case of Grain v1 and ACORN v3." [SSMC17] That is, implementing the Lizard-construction design principle significantly limits the impact of this kind of DFA. In fact, given the attack's preconditions, one could question its sense in the case of Lizard at all. More precisely, the attack assumes that it is possible to repeatedly generate the keystream for a specific key/IV pair (even under injecting faults). And 'all it does' is to provide one of the secret inner states underlying this particular (already known) keystream. Due to the Lizard-construction, the recovered inner state is of no use for generating keystream belonging to other key/IV pairs and, hence, actually 'useless' from the viewpoint of decrypting secret communication.

8.5 Hardware Implementation

In the following, we present the hardware results for our new stream cipher Lizard and compare them to those of Grain v1 [HJM06]. The reasons for focusing on Grain v1 are twofold. First, it is a natural choice for comparison due to the close structural

relation between Lizard and Grain v1 as explained in Sections 8.2 and 8.3. Second, and more importantly, Grain v1 can be considered as a benchmark for new lightweight stream cipher designs as it turned out to be the most hardware-efficient member of the eSTREAM [ECR08] portfolio.[13] This conclusion can be drawn, e.g., from tables 1–4 and figures 1–3 in [GB08], where Good and Benaissa evaluate the hardware performance of the phase-3, profile-2 candidates of the eSTREAM competition. (Note that, for the sake of comparability, we are referring to the standard, non-parallelized implementations of the ciphers, which, after initialization, produce one keystream bit per clock cycle.)

8.5.1 Performance

When comparing the hardware performance of ciphers, the first apparent task is to specify which hardware is actually targeted. In line with papers like [Fel07] and [GB08], we focus on application-specific integrated circuits (ASICs) with standard CMOS libraries.[14] ASICs are an essential component in RFID technology, which, as Feldhofer puts it, "allows giving a digital identity to nearly every object in the world" [Fel07]. While especially low-cost RFID tags may be subject to severe resource limits as explained in Chapter 2, in many cases, they still need to provide security features like confidentiality or privacy. The two main restrictions imposed on the design of cryptographic protocols for RFID tags are the circuit size (cf. Subsection 2.3.3) and the power budget (cf. Subsection 2.3.4). The circuit size strongly influences the manufacturing costs of an RFID tag and is commonly specified in gate equivalents (GE), where one GE corresponds to the area of a two-input drive-strength-one NAND gate. The power consumption is crucial as low-cost RFID tags are usually passively powered (i.e., via an electromagnetic field radiated by the reader). Given that the transmission power of an RFID reader is limited by factors like legal regulations, the more power a tag consumes, the smaller the maximum reading distance becomes. As done by Feldhofer in [Fel07] for his comparison of low-power implementations of Trivium and Grain, we will focus on these two values, cell area and power consumption, in our comparison of Lizard and Grain v1.

In addition, we provide the critical path delay of the circuit and the number of clock cycles required to perform the state initialization in Table 8.1. As explained in Subsection 2.3.7, the metric *delay* plays a rather negligible role in the context of ultra-constrained devices commonly running at clock speeds of about 100 kHz (cf. Subsection 2.3.6). For comparison, the delay of our implementation of Lizard is 2474 ps, which would allow for a maximum clock frequency of about 404 MHz. Still, for the

[13] See Chapter 5 for more information on the eSTREAM project and the hardware portfolio members Trivium [CP05] and Grain v1 [HJM06].

[14] The testing framework for eSTREAM profile-2 candidates [BKL+06] additionally considers low-cost FPGAs and a corresponding performance evaluation can be found in, e.g., [GCB06] and [BKSQ07]. However, as ASICs are certainly more common in ultra-constrained, low-cost environments (targeted by Lizard), we focus on this technology and leave an FPGA-related evaluation of our new cipher as future work.

sake of completeness and as several other works like [GB08] do so as well, we decided to provide the corresponding data. For further information about the technical details of this metric and the related term *critical path*, we refer the reader to Subsection 2.3.7, where, at the example of Lizard, we also explain how a delay of 2474 ps translates into a maximum clock frequency of about 404 MHz.

Like Feldhofer, we will not provide compound metrics (e.g., the power-area-time product given in [GB08]) but leave the computation (as appropriate for the application scenario) to the reader. Moreover, it is important to note that while the area requirement of cipher designs can be compared over different standard cell libraries by using the measure gate equivalents, "[p]ower cannot be scaled reliably between different processes and libraries" [GB08]. Consequently, it is inevitable to use the same design flow for all implementations that are to be compared. As done by Good and Benaissa in [GCB06] and [GB08] in their hardware comparison of eSTREAM candidates, we use Cadence tools [Cad17] for synthesis and Mentor ModelSim [Men17] for generating switching activity. While Feldhofer uses 0.35 μm and Good and Benaissa use 0.13 μm CMOS process technology in the cited papers, we employ the UMCL18G212T3 (0.18 μm, 1.8 V) standard cell library that was also used by Poschmann in [Pos09] for implementing the block cipher PRESENT and by Beierle et al. for the `SKINNY` family of block ciphers [BJK+16]. Our results (see Table 8.1) are obtained via Cadence Encounter RTL Compiler RC12.22 [Cad17] and are based on the netlist generated through the command `synthesize -to_placed -effort high`. Like Feldhofer in [Fel07], we target a 100 kHz clock (cf. Subsection 2.3.6) and employ clock gating (cf. Subsection 2.3.4). The switching activity for power estimation (recorded with Mentor ModelSim SE-64 6.5b [Men17] and fed back to RTL compiler) covers the generation of 10 kbit of keystream (as done in [GCB06]) at a clock rate of 100 kHz and includes the state initialization of the compared cipher modules. To improve the accuracy of the results, switching activity for 25 different random key/IV combinations is considered and the arithmetic mean of the respective power estimations is computed. For all power values given in Table 8.1, the largest deviation of a single estimation from the computed average was below one percent.

As pointed out above, we had to implement not only our new cipher Lizard but also Grain v1 in order to obtain a meaningful comparison of power consumptions on the basis of the same design flow. Moreover, as Lizard targets low-power environments, we decided to serialize phases 1 (i.e., the initial key and IV loading) and 3 (i.e., the second key addition) of its state initialization (see Subsection 8.5.2 for details). This allows to take full advantage of the FSR-based structure of the KSG by using simple D flip-flops (without additional costly features like scan, set/reset or enable functionality) for storing the cipher's inner state. Naturally, for reasons of fairness, we considered the positive effects of serializing the key/IV loading for Grain v1 as well. The corresponding values can be found in Table 8.1 along with those for the straightforward implementation of Grain v1. We do not suggest to use non-serialized key/IV loading for Lizard and, hence, Table 8.1 only contains the hardware metrics for Lizard with phases 1 and 3

Table 8.1: Hardware results for a clock speed of 100 kHz. The symbol * indicates that the respective implementation uses serialized key/IV loading. Latency is given as the number of clock cycles needed to perform the state initialization (including module reset and key/IV loading), denoted *Load/Ini* in the table. After state initialization, all designs produce one keystream bit per clock cycle, corresponding to a throughput of 100 kbit/s.

Design	**Area** [GE]	**Power** [nW]	**Delay** [ps]	**Load/Ini** [clk. cyc.]
Lizard*	1161	2110	2474	499
Grain v1*	1268	2517	2155	241
Grain v1	1221	3578	2166	161

implemented as described in Subsection 8.5.2.[15]

Serializing parts of the state initialization comes at a price, however, as it increases latency. In the case of Lizard, 240 additional clock cycles are required for computing the initial state, based on which, subsequently, one keystream bit per clock cycle is produced. We consider this increase tolerable considering that, even with serialized key/IV loading, the state initialization (including module reset) of Lizard takes only 499 clock cycles, as compared to, e.g., 1153 clock cycles for the straightforward (i.e., non-serialized) implementation of the state initialization of the eSTREAM hardware portfolio member Trivium. For Grain v1, serializing the key/IV loading can be realized at the cost of only 80 additional clock cycles as the key and the IV can be shifted separately into the NFSR and the LFSR, respectively, at the same time. In our non-serialized (i.e., straightforward) implementation of Grain v1, the key/IV loading is performed as part of the module reset (which takes one clock cycle). Hence, the state initialization takes $1 + 160 = 161$ clock cycles. In the serialized implementation, the key/IV loading of Grain v1 is performed in a separate stage, leading to a state initialization effort of $1 + 80 + 160 = 241$ clock cycles.

Table 8.1 shows that the estimated power consumption of Lizard during the generation of 10 kbit of keystream (including state initialization) is about 16 percent lower than that of Grain v1 with serialized key/IV loading. Moreover, Lizard also allows to save on chip area and, hence, production costs.

At first glance, the reduction in chip area might seem surprisingly small, considering that Lizard's inner state is about 25 percent smaller than that of Grain v1. Remember, however, that Lizard needs additional logic for loading the (larger) key (twice) and, moreover, we chose a much heavier output function, which is defined over 53 variables.

[15]The reason for not describing phases 1 and 3 of the state initialization of Lizard in its serialized form in the first place in Subsection 8.2.2 is that we wanted the specification of the algorithm to be as concise as possible. Moreover, the way we introduce Lizard in Subsection 8.2.2 hopefully facilitates to understand the relation to the generic Lizard-construction as described in Subsection 8.3.5.

As a consequence of the larger key, LIZARD would benefit more than Grain v1 from an 'external' key source (like an EEPROM; cf. Subsection 2.3.9) that takes over the task of key bit selection based on an index or supplies the key bits sequentially. However, for reasons of fairness, we assumed the (from LIZARD's point of view) worst situation that all key and IV bits are provided via separate wires to the respective cipher modules, which then have to take care of key bit selection themselves. To avoid ambiguity about the capabilities of the implementations which the values in Table 8.1 are based on, we provide the interfaces of the respective modules along with a short description in Appendix 8.B and a reference implementation of LIZARD written in Verilog in Appendix 8.C.

Finally, note that while LIZARD explicitly targets power-constrained devices like passively powered RFID tags, it can also help to save energy (cf. Subsection 2.3.5) on battery powered devices. For example, producing 10 kbit of keystream (including state initialization) at 100 kHz would consume 0.22 μJ with LIZARD*, 0.26 μJ with Grain v1*, and 0.36 μJ with Grain v1. Keep in mind, however, that due to the larger number of state initialization cycles of LIZARD*, Grain v1* will consume less total energy for very small keystream pieces.

For application scenarios where only few hundred (continuous) bits of plaintext are to be encrypted under a single IV and the optimization target is energy consumption, it might actually be preferable to not use stream ciphers at all, but instead resort to lightweight block ciphers like `SKINNY` [BJK+16], SIMON [BSS+13], or PRESENT [BKL+07]. For example, a round-based implementation of `SKINNY-64-128` (block size 64 bits, key size up to 128 bits) takes 1696 GE and achieves a throughput of 177.78 kbit/s at 100 kHz and a nibble-serial implementation takes 1399 GE and achieves 8.12 kbit/s at 100 kHz. As block ciphers do not need a state initialization in the style of stream ciphers, these throughput rates are achievable for all plaintext amounts whose length is a multiple of the cipher's block size. For stream ciphers, the effective throughput (including state initialization) depends on the amount of keystream that is produced under a single IV. This shows that efficiency comparison of stream ciphers and block ciphers is dependent on targeted applications. For example, our implementations of LIZARD and Grain v1 may be paused after each single keystream bit, whereas block ciphers can only operate on whole blocks of data. Hence, in a scenario where only few bits (i.e., less than the block size) need to be encrypted from time to time (under the same IV), the use of block ciphers might lead to a large overhead w.r.t. computation and communication (due to padding). In other scenarios, as described above, block ciphers may be preferable.

8.5.2 Serialization of Phases 1 and 3 of LIZARD's State Initialization

As pointed out in the previous subsection, we decided to serialize phases 1 (i.e., the initial key and IV loading) and 3 (i.e., the second key addition) of LIZARD's state initialization in order to allow for a more power-efficient implementation.

More precisely, phase 1 of the state initialization is distributed over 121 clock cycles

$c = 0, \ldots, 120$ as follows:

$$B_j^{0,c+1} := \begin{cases} B_{j+1}^{0,c}, & \text{for } j \in \{0, \ldots, 88\}, \\ S_0^{0,c}, & \text{for } j = 89, \end{cases}$$

$$S_i^{0,c+1} := \begin{cases} S_{i+1}^{0,c}, & \text{for } i \in \{0, \ldots, 29\}, \\ p_c, & \text{for } i = 30, \end{cases}$$

where

$$p_c := \begin{cases} K_c \oplus IV_c, & \text{for } c \in \{0, \ldots, 63\}, \\ K_c, & \text{for } c \in \{64, \ldots, 118\}, \\ K_c \oplus 1, & \text{for } c = 119, \\ 1, & \text{for } c = 120. \end{cases}$$

To avoid ambiguity, we point out that $B_j^{0,121} = B_j^0$, $j = 0, \ldots, 89$, and $S_i^{0,121} = S_i^0$, $i = 0, \ldots, 30$, where B_j^0 and S_i^0 are defined as explained in Subsection 8.2.2. The initial content of the FSRs (i.e., $B_j^{0,0}$, $j = 0, \ldots, 89$, and $S_i^{0,0}$, $i = 0, \ldots, 30$) is undefined and algorithmically irrelevant as, what effectively happens here, is that the bitstring

$$K_0 \oplus IV_0, \ldots, K_{63} \oplus IV_{63}, K_{64}, \ldots, K_{118}, K_{119} \oplus 1, 1$$

is shifted into the KSG's driving registers 'from the right' (in terms of Fig. 8.1 in Section 8.2).

Analogously, phase 3 of the state initialization is also distributed over 121 clock cycles $c = 0, \ldots, 120$ as follows:

$$B_j^{129,c+1} := \begin{cases} B_{j+1}^{129,c}, & \text{for } j \in \{0, \ldots, 88\}, \\ S_0^{129,c}, & \text{for } j = 89, \end{cases}$$

$$S_i^{129,c+1} := \begin{cases} S_{i+1}^{129,c}, & \text{for } i \in \{0, \ldots, 29\}, \\ q_c, & \text{for } i = 30, \end{cases}$$

where

$$B_j^{129,0} := B_j^{128} \text{ for } j \in \{0, \ldots, 89\},$$

$$S_i^{129,0} := S_i^{128} \text{ for } i \in \{0, \ldots, 30\},$$

$$q_c := \begin{cases} B_0^{129,c} \oplus K_c, & \text{for } c \in \{0, \ldots, 119\}, \\ 1, & \text{for } c = 120, \end{cases}$$

and B_j^{128} and S_i^{128} are defined as explained in Subsection 8.2.2.

Again, to avoid ambiguity, we point out that $B_j^{129,121} = B_j^{129}$, $j = 0, \ldots, 89$, and $S_i^{129,121} = S_i^{129}$, $i = 0, \ldots, 30$, where B_j^{129} and S_i^{129} are defined as explained in Subsection 8.2.2.

8.6 Conclusion and Outlook

We presented Lizard, a new lightweight stream cipher for power-constrained devices like passive RFID tags. Its hardware efficiency results from combining a Grain-like design with the generic Lizard-construction introduced in Chapter 7, which offers provable $(2n/3)$-security against TMD-TO attacks aiming at key recovery. Lizard uses 120-bit keys, 64-bit IVs, and has an inner state length of 121 bits. It is supposed to provide 80-bit security against key recovery attacks and 60-bit security against distinguishing attacks. Lizard allows to generate up to 2^{18} keystream bits per key/IV pair, which would be sufficient for many existing communication scenarios like Bluetooth, WLAN, or HTTPS.

Hardware implementations for Lizard and Grain v1 were created using the same design flow in order to allow for a meaningful comparison of performance metrics. The results show that Lizard consumes about 16 percent less power than Grain v1 at slightly reduced area requirements. This indicates that in scenarios where plaintext packets of moderate length are to be encrypted under individual IVs, the Lizard-construction design principle provides an interesting alternative to conventional state initialization algorithms of stream ciphers.

As future work, we suggest to evaluate the performance of Lizard on other hardware platforms like FPGAs or microcontrollers. Moreover, it might be interesting to investigate, whether, under the current security guarantees, even more lightweight variants of Lizard are possible (e.g., by choosing a less heavy output function). With respect to the generic Lizard-construction, it would be desirable to have key sizes of $2n/3$ (instead of currently n) and still maintain provable $(2n/3)$-security against TMD-TO attacks aiming at key recovery. One way to achieve this could be to derive an n-bit Lizard-*construction key* (or even two separate n-bit keys for the respective phases of the state initialization) on-the-fly from the actual $(2n/3)$-bit key, e.g., by using a component like the round key function of Fruit [GHX16].

Another important open problem is the guaranteed keystream period of Lizard (as well as of the Grain family and other stream ciphers like Trivium; see, e.g., [HG11]). In the design document of Grain v1 [HJM06], it is stated that "the LFSR guarantees a minimum period for the keystream and it also provides balancedness in the output." But in fact, in the setting of Grain (and also Lizard), a large guaranteed period of the internal state (composed of both FSRs) is necessary but not sufficient for a large guaranteed period of the keystream, because we only know that the period of the keystream divides the length of the corresponding internal state cycle.

Finally, we also suggest to study and compare the suitability of recent (lightweight) stream and block ciphers for different application scenarios. Such a comparison could be in the style of [GB08], where Good and Benaissa evaluated the hardware performance of eSTREAM candidates, but would also need to establish a reasonable common test setting for the different concepts stream and block cipher.

In the following Chapter 9, we will discuss some more directions of potential future

research, which we consider especially promising, in further detail. In particular, we will also forge a bridge to our starting point of lightweight authentication protocols (see Chapters 3 and 4) by sketching how LIZARD could be used to realize hardware-efficient, privacy-preserving authentication on ultra-constrained RFIDs, thus constituting a viable alternative to prevalent block cipher-based constructions (cf. Section 3.3).

Appendix 8.A Test Vectors

Key (120 bits), IV (64 bits), and the corresponding first 128 keystream bits in hexadecimal notation. To avoid ambiguity, note that, e.g., the key

`0x01234FFFFFFFFFFFFFFFFFFFFFFF`

corresponds to

$$(K_0, \ldots, K_{119}) = (0,0,0,0,0,0,0,1,0,0,1,0,0,0,1,1,0,1,0,0,1,\ldots,1)\,.$$

Similarly, for the keystream, the example

`0x01000000000000000000000000000000`

would mean that the first seven keystream bits (i.e., $z_{257}, \ldots, z_{263}$) are zero, followed by a one and 120 more zeros.

```
Key:       0x00000000000000000000000000000000
IV:        0x0000000000000000
Keystream: 0xB6304CA4CA276B3355EC2E10968E84B3

Key:       0x00000000000000000FFFFFFFFFFFFFFF
IV:        0xFFFFFFFFFFFFFFFF
Keystream: 0x4D190941816F942358F0D164F4ECEB09

Key:       0x0123456789ABCDEF0123456789ABCD
IV:        0xABCDEF0123456789
Keystream: 0x983311A97831586548209DAFBF26FC93
```

LIZARD (online) in PHP. Note that Christian A. Gorke offers an open-source implementation of LIZARD in PHP at `https://th.informatik.uni-mannheim.de/lizard/php/`. On the respective website, users also have the possibility to generate keystreams of length up to 10 000 bits for key/IV combinations of their choice via a convienient online form. Moreover, this online service can also be accessed through a straighforward application programming interface (API), which allows, e.g., for employing it as part of back-end solutions (such as server-side verification in the case of LIZARD-based authentication, cf. Section 9.2).

Appendix 8.B Module Interfaces/Capabilities

Listing 8.1: Verilog module port declaration for LIZARD.

```
1  module lizard(
2    input wire clk,
3    input wire reset,
4    input wire enable,
5    input wire [0:119] key,
6    input wire [0:63] iv,
7    output wire keystreamBit,
8    output wire keystreamFlag
9    );
```

Listing 8.2: Verilog module port declaration for Grain v1.

```
1  module grainv1(
2    input wire clk,
3    input wire reset,
4    input wire enable,
5    input wire [0:79] key,
6    input wire [0:63] iv,
7    output wire keystreamBit,
8    output wire keystreamFlag
9    );
```

Listings 8.1 and 8.2 show the interfaces of the implemented cipher modules. For Grain v1, the straightforward implementation and the variant with serialized key/IV loading (denoted as Grain v1* in Table 8.1) have identical interfaces.

The modules use synchronous reset (taking one clock cycle) and all operations are triggered by the rising edge of the clock. By setting the `enable` flag low, the respective cipher module can be paused at any time during state initialization or keystream generation. Setting `reset` high resets a module. Once `reset` is set low again, state initialization begins. For all modules, it is assumed that key and IV are available (via `key` and `iv`) until key/IV loading has finished. Once a cipher module has completed state initialization and enters the keystream generation phase, it changes `keystreamFlag` from low to high and outputs one keystream bit per clock cycle via `keystreamBit`.

Appendix 8.C Reference Implementation

Listing 8.3: Reference implementation of LIZARD in Verilog.

```
1  `timescale 1us / 1ps
2
3  //************************************
4
5  module lizard(
6    input wire clk,
7    input wire reset,
8    input wire enable,
9    input wire [0:119] key,
10   input wire [0:63] iv,
11   output wire keystreamBit,
12   output wire keystreamFlag
13   );
14
15 ////////////////
16
17 reg[2:0] cipherFSM;
18
19 //                          keystreamFlag
20 //                          v
21 localparam S_PHASE1 = 3'b00_0;
22 localparam S_PHASE2 = 3'b01_0;
23 localparam S_PHASE3 = 3'b10_0;
24 localparam S_PHASE4 = 3'b11_0;
25 localparam S_GENOUT = 3'b11_1;
26
27 ////////////////
28
29 assign keystreamFlag = cipherFSM[0];
30
31 ////////////////
32
33 reg [0:30] nfsr1_state;
34 reg [0:89] nfsr2_state;
35
36 ////////////////
37
38 wire nfsr1_feedbackBit;
39
40 // efficient implementation of NFSR1's feedback function as decribed in
       the ACHTERBAHN paper (cf. design specs)
41
42 wire nfsr1_maj, nfsr1_mux1, nfsr1_mux2, nfsr1_mux3, nfsr1_mux4;
43
44 assign nfsr1_maj = (nfsr1_state[4] & nfsr1_state[12]) | (nfsr1_state[4] &
       nfsr1_state[19]) | (nfsr1_state[12] & nfsr1_state[19]);
```

```
45 assign nfsr1_mux1 = (nfsr1_state[21]) ? nfsr1_state[12] : nfsr1_state[17];
46 assign nfsr1_mux2 = (nfsr1_state[8]) ? nfsr1_state[18] : nfsr1_state[20];
47 assign nfsr1_mux3 = (nfsr1_state[21]) ? nfsr1_state[7] : nfsr1_state[22];
48 assign nfsr1_mux4 = (nfsr1_mux3) ? nfsr1_maj : nfsr1_mux2;
49
50 assign nfsr1_feedbackBit_achterbahn = nfsr1_state[0] ^ nfsr1_state[2] ^
       nfsr1_state[5] ^ nfsr1_state[6] ^ nfsr1_state[15] ^ nfsr1_state[18] ^
       nfsr1_state[25] ^ (nfsr1_state[14] & nfsr1_state[19]) ^ nfsr1_mux1 ^
       nfsr1_mux4;
51
52 assign nfsr1_feedbackBit = nfsr1_feedbackBit_achterbahn;
53
54 ////////////////
55
56 wire nfsr2_feedbackBit;
57
58 assign nfsr2_feedbackBit =  nfsr2_state[0] ^ nfsr2_state[24] ^
       nfsr2_state[49] ^ nfsr2_state[79] ^ nfsr2_state[84] ^ (nfsr2_state[3]
       & nfsr2_state[59]) ^ (nfsr2_state[10] & nfsr2_state[12]) ^
       (nfsr2_state[15] & nfsr2_state[16]) ^ (nfsr2_state[25] &
       nfsr2_state[53]) ^ (nfsr2_state[35] & nfsr2_state[42]) ^
       (nfsr2_state[55] & nfsr2_state[58]) ^ (nfsr2_state[60] &
       nfsr2_state[74]) ^ (nfsr2_state[20] & nfsr2_state[22] &
       nfsr2_state[23]) ^ (nfsr2_state[62] & nfsr2_state[68] &
       nfsr2_state[72]) ^ (nfsr2_state[77] & nfsr2_state[80] &
       nfsr2_state[81] & nfsr2_state[83]);
59
60 ////////////////
61
62 wire outLin, outQuad, outTri1, outTri2;
63
64 assign outLin = nfsr2_state[7] ^ nfsr2_state[11] ^ nfsr2_state[30] ^
       nfsr2_state[40] ^ nfsr2_state[45] ^ nfsr2_state[54] ^ nfsr2_state[71];
65
66 assign outQuad = (nfsr2_state[4] & nfsr2_state[21]) ^ (nfsr2_state[9] &
       nfsr2_state[52])^ (nfsr2_state[18] & nfsr2_state[37]) ^
       (nfsr2_state[44] & nfsr2_state[76]);
67
68 assign outTri1 = nfsr2_state[5] ^ (nfsr2_state[8] & nfsr2_state[82]) ^
       (nfsr2_state[34] & nfsr2_state[67] & nfsr2_state[73]) ^
       (nfsr2_state[2] & nfsr2_state[28] & nfsr2_state[41] & nfsr2_state[65])
       ^ (nfsr2_state[13] & nfsr2_state[29] & nfsr2_state[50] &
       nfsr2_state[64] & nfsr2_state[75]) ^ (nfsr2_state[6] & nfsr2_state[14]
       & nfsr2_state[26] & nfsr2_state[32] & nfsr2_state[47] &
       nfsr2_state[61]) ^ (nfsr2_state[1] & nfsr2_state[19] & nfsr2_state[27]
       & nfsr2_state[43] & nfsr2_state[57] & nfsr2_state[66] &
       nfsr2_state[78]);
69
70 assign outTri2 = nfsr1_state[23] ^ (nfsr1_state[3] & nfsr1_state[16]) ^
       (nfsr1_state[9] & nfsr1_state[13] & nfsr2_state[48]) ^ (nfsr1_state[1]
```

```
    & nfsr1_state[24] & nfsr2_state[38] & nfsr2_state[63]);
71
72  assign keystreamBit = outLin ^ outQuad ^ outTri1 ^ outTri2;
73
74  ////////////////
75
76  reg [6:0] ctr;
77
78  wire done_keyAddition;
79  assign done_keyAddition = (ctr == 7'd120) ? 1'b1 : 1'b0;
80
81  wire done_ivAddition;
82  assign done_ivAddition = ctr[6];
83
84  wire done_mixing;
85  assign done_mixing = (ctr == 7'd127) ? 1'b1 : 1'b0;
86
87  wire [6:0] keyIndex;
88  assign keyIndex = ctr[6:0];
89
90  wire [5:0] ivIndex;
91  assign ivIndex = ctr[5:0];
92
93  ////////////////
94
95  always @(posedge clk)
96    begin
97      if (reset)
98        begin
99          ctr <= 7'd0;
100         cipherFSM <= S_PHASE1;
101       end
102     else if (enable)
103       begin
104         nfsr1_state[0:28] <= nfsr1_state[1:29];
105         nfsr2_state[0:88] <= nfsr2_state[1:89];
106
107         case (cipherFSM)
108           S_PHASE1:
109             begin
110               nfsr2_state[89] <= nfsr1_state[0];
111
112               if (done_keyAddition)
113                 begin
114                   nfsr1_state[29] <= ~nfsr1_state[30];
115                   nfsr1_state[30] <= 1'b1;
116
117                   ctr <= 7'd0;
118                   cipherFSM <= S_PHASE2;
119                 end
```

```
120                 else
121                   begin
122                     nfsr1_state[29] <= nfsr1_state[30];
123
124                     if (done_ivAddition)
125                       nfsr1_state[30] <= key[keyIndex];
126                     else
127                       nfsr1_state[30] <= key[keyIndex] ^ iv[ivIndex];
128
129                     ctr <= ctr + 1;
130                   end
131               end
132
133             S_PHASE2:
134               begin
135                 nfsr1_state[29] <= nfsr1_state[30];
136
137                 nfsr1_state[30] <= nfsr1_feedbackBit ^ keystreamBit;
138                 nfsr2_state[89] <= nfsr2_feedbackBit ^ nfsr1_state[0] ^
                        keystreamBit;
139
140                 if (done_mixing)
141                   begin
142                     ctr <= 7'd0;
143                     cipherFSM <= S_PHASE3;
144                   end
145                 else
146                   ctr <= ctr + 1;
147               end
148
149             S_PHASE3:
150               begin
151                 nfsr1_state[29] <= nfsr1_state[30];
152
153                 nfsr2_state[89] <= nfsr1_state[0];
154
155                 if (done_keyAddition)
156                   begin
157                     nfsr1_state[30] <= 1'b1;
158
159                     ctr <= 7'd0;
160                     cipherFSM <= S_PHASE4;
161                   end
162                 else
163                   begin
164                     nfsr1_state[30] <= nfsr2_state[0] ^ key[keyIndex];
165
166                     ctr <= ctr + 1;
167                   end
168               end
```

```
169
170             S_PHASE4:
171               begin
172                 nfsr1_state[29] <= nfsr1_state[30];
173
174                 nfsr1_state[30] <= nfsr1_feedbackBit;
175                 nfsr2_state[89] <= nfsr2_feedbackBit ^ nfsr1_state[0];
176
177                 if (done_mixing)
178                   cipherFSM <= S_GENOUT;
179                 else
180                   ctr <= ctr + 1;
181               end
182
183             S_GENOUT:
184               begin
185                 nfsr1_state[29] <= nfsr1_state[30];
186
187                 nfsr1_state[30] <= nfsr1_feedbackBit;
188                 nfsr2_state[89] <= nfsr2_feedbackBit ^ nfsr1_state[0];
189               end
190           endcase
191         end
192     end
193
194 ////////////////
195
196 endmodule
197
198 //************************************
```

Future forever and ever!

Graffiti at a university wall

CHAPTER 9

Future Research Directions

ABSTRACT

Throughout this thesis, we have already identified various directions for potential future research in the broad field of lightweight cryptography. Here, we now present three (in our opinion) particularly interesting projects in further detail.

First, we describe a straightforward Lizard-based (cf. Chapter 8) authentication scheme and present a corresponding basic FPGA prototype. We also discuss the relevance of *privacy preservation* in this context and suggest, how the original Lizard design could be adapted for making use of the full potential which the general Lizard-construction (cf. Chapter 7) offers w.r.t. lightweight authentication.

Second, we take our idea of continuously using the IV as part of a stream cipher's state update one step further. While in Section 6.5, this new approach was 'only' meant to protect ciphers which continuously use the secret key (i.e., Sprout-like ciphers) against TMD-TO *distinguishing* attacks, we here show that by combining *continuous IV use* and *packet mode* (see, e.g., Section 5.1 and Chapter 7), classical TMD-TO *inner state recovery* attacks can be thwarted, as well.

Third, motivated by our application context of ultra-constrained RFID devices (cf. Chapter 2), where (e.g., due to their restricted communication capabilities; cf. Subsection 2.3.1) often only few keystream bits may actually be available to an attacker, we suggest to increase the study of *short-keystream attacks*. In particular, we believe that the field of BDD-based cryptanalysis has been neglected for too long and provide several new ideas how the efficiency of such attacks could be improved in the future.

Declaration of Origin: Section 9.3 is based on the paper *A Note on Stream Ciphers that Continuously Use the IV* [HKM17a], written together with Matthias Krause and Willi Meier and presented at *Dagstuhl Seminar 18021: Symmetric Cryptography, 2018* [Mei18].

9.1 Introduction

As part of the previous chapter conclusions, we have already pointed out numerous open problems and potential directions for fruitful future work. Adding to this picture, in this chapter, we will explicitly discuss three new ideas, which we consider particularly promising, in further detail.

First, in Section 9.2, we sketch how our new lightweight stream cipher Lizard (cf. Chapter 8) could be used (and further optimized) to realize hardware-efficient, privacy-preserving authentication, thereby connecting our two main topics: *lightweight authentication* (cf. Chapters 3 and 4) and *lightweight stream ciphers* (cf. Chapters 6 to 8). As a real-world example, we also describe our implementation of straightforward Lizard-based authentication between two connected FPGA boards, which are supposed to serve as targets in future side-channel cryptanalysis.

Second, in Section 9.3, we present a new idea for designing stream ciphers which resist TMD-TO inner state recovery attacks. It combines the concept of explicitly targeting *packet mode* scenarios (as introduced in Chapter 7) and the idea of *continuously using the IV* as part of the state update (as introduced in Section 6.5). While in Section 6.5, continuously using the IV was suggested to protect Sprout-like ciphers (i.e., ciphers which continuously use the secret key) against TMD-TO *distinguishing* attacks, we will explain here that for stream ciphers working in packet mode, it can also thwart TMD-TO *inner state recovery* attacks. In particular, our approach alleviates the need for additional countermeasures such as the continuous key use of Sprout-like ciphers (cf. Subsections 6.2.1 to 6.2.3) or the second key addition in Lizard's state initialization algorithm (cf. Subsection 8.3.5).

Third, in Section 9.4, we suggest to 'revive' the cryptanalytic technique of BDD attacks, originally introduced by Krause [Kra02] in 2002. After successful attacks against then widespread stream ciphers such as E_0 of the Bluetooth standard [Blu14] and A5/1 of GSM [BGW99], and despite several follow-up publications (see, e.g., [KS06] and [Ste07a]), the technique subsequently somehow sank into oblivion with the advent of eSTREAM portfolio [BBV12] members such as Trivium [CP05] and Grain v1 [HJM06], whose state size is at least twice the key size. However, we will explain that the lack of corresponding research and respective publications after 2008 is very unfortunate for several (up-to-date) reasons. To 'reignite the flame', we will also make several suggestions how the efficiency of BDD attacks can be improved, especially in the important context of parallelization. As SAT[1] attacks have seen much attention recently and as they share strong relations to BDD-based cryptanalysis, these will be discussed (and compared to BDD attacks) in Section 9.4, as well.

Section 9.5 concludes the chapter.

[1] SAT is the common abbreviation for the Boolean satisfiability problem.

9.2 LIZARD-based Authentication

Remember that, in Chapter 5, we left the path of searching for dedicated authentication protocols suitable for ultra-constrained RFIDs (cf. Chapter 2) and, instead, turned to a fundamental question which arose to us while designing the $(n, k, L)^{\langle 80 \rangle}$-protocol introduced in Chapter 4: 'Why use a bitstream generator only to produce the specifications of the secret functions, but not for generating the authentication token right away?'

With our new lightweight stream cipher Lizard (cf. Chapter 7), we now have the necessary 'tool' to do exactly this. Lizard surpasses Grain v1 (cf. Subsection 5.2.4), the most hardware-efficient member of the latest eSTREAM portfolio [BBV12], in important metrics for lightweight ciphers such as chip area (cf. Subsection 2.3.3) and power consumption (cf. Subsection 2.3.4). Given its hardware cost of only 1161 GE (see Table 8.1 in Subsection 8.5.1), it can even compete with lightweight block ciphers like PRESENT [BKL+07], whose straightforward implementation requires 1570 GE (and 32 clock cycles to encrypt one 64-bit block of data) [BKL+07] and whose serialized implementation requires 1080 GE (and 563 clock cycles to encrypt one 64-bit block of data) [RPLP08]. In fact, we will explain in the course of this section that Lizard-based authentication can even be superior to the prevalent (lightweight) block cipher-based schemes (cf. Section 3.3) if important additional features such as *privacy preservation* are required.

9.2.1 Employing LIZARD 'as it is'

Lizard can actually be used straightforwardly for realizing a basic *challenge-response authentication protocol*. In the respective scenario, the *verifier* (an RFID reader) chooses some 64-bit IV as the challenge and transmits it to the *prover* (an RFID tag), who then responds with the corresponding keystream prefix of length 120 bits for the given (public) IV and the common secret key. Knowing the IV and the key, the verifier is able to generate the correct keystream prefix and, for a valid prover, both bitstrings will match, leading to a successful authentication. Just like for general stream cipher-based encryption, it is obviously crucial here that the same IV is never used twice. This can be guaranteed if the verifier keeps a log of already consumed IVs or simply uses a sequential (or, e.g., maximum-length LFSR-driven) IV counter. Alternatively, for each authentication instance, the verifier can also simply choose a random IV, given that the underlying IV space is large enough. In our targeted context of ultra-constrained RFIDs, Lizard's IV space of size 2^{64} can be expected to be appropriate for nearly every application scenario. And, in extreme cases, there is still the possibility to switch the secret key after a predefined number of authentication events.

Also note that revealing the keystream should not lead to security issues here, as our complete cryptanalysis for Lizard (see Section 8.4) was performed in a *known-keystream setting*. Moreover, for the case of a malicious verifier, remember that the security proof

of the LIZARD-construction against generic TMD-TO key recovery attacks even holds for *chosen-IV adversaries* (see Chapter 7, esp. Section 7.4).

An FPGA Implementation of classical LIZARD-based Authentication

Field-programmable gate arrays (FPGAs) are an important component in hardware development. In particular, even if the final product is supposed to be an ASIC (as in the case of ultra-constrained RFIDs, cf. Chapter 2), they play a major role during the stage of testing/prototyping. This is due to the fact that, as their name suggests, FPGAs actually allow for reprogramming their respective circuits, e.g., in the course of applying further optimizations or fixing bugs as part of the development process. The manufacturing of ASICs, on the other hand, has large 'entry costs'. That is, even to produce a single ASIC, e.g., corresponding photolithographic *masks* (and *wafers*) are required, whose costs amount to thousands of U.S. dollars (see, e.g., Subsection 2.3.10 for further details). So, in a nutshell, if only few devices are required (as in the case of testing/prototyping), FPGAs are the hardware of choice, whereas, when it comes to large-scale production for the final product, ASICs can play their strengths in the context of ultra-constrained RFIDs due to the then low per-unit costs (as, e.g., the aforementioned masks can be used to produce thousands or millions of devices).

In consequence, we decided to use FPGAs to realize our basic prototype of LIZARD-based authentication depicted in Fig. 9.1. The reasons for building this real-world example at all are twofold.

First, in the context of this thesis, we wanted to complete the picture spanning from laying the theoretical foundations with the security proof of the LIZARD-construction (cf. Chapter 7), over designing a concrete instantiation in the form our new lightweight stream cipher LIZARD (cf. Chapter 8), to now presenting an actual hardware device.

Second, as future work, we suggest to evaluate (and potentially improve) LIZARD w.r.t. its resistance against side-channel attacks such as power analysis. In this context, our basic FPGA prototype is supposed to serve as a first target.

Before we give a brief description of our hardware example, we would like to warn the reader not to be mislead by the sheer sizes of the FPGA boards depicted in Fig. 9.1. These are general-purpose testing boards, which contain a multitude of additional features (such as LAN and VGA ports) that we make no use of here. The actual FPGA core is a very small component on these boards and even 'within' this core, the gate array of LIZARD occupies only a tiny fraction of space.

In the setup depicted in Fig. 9.1, the left FPGA board plays the role of the verifier, while the right board acts as the prover. On both boards, the FPGA core contains a gate array built on the basis of our LIZARD reference implementation provided in Appendix 8.C. Around these 'LIZARD modules', we implemented a simple challenge-response authentication logic (as described above) and the respective communication is performed through a basic, self-written communication protocol using the six small,

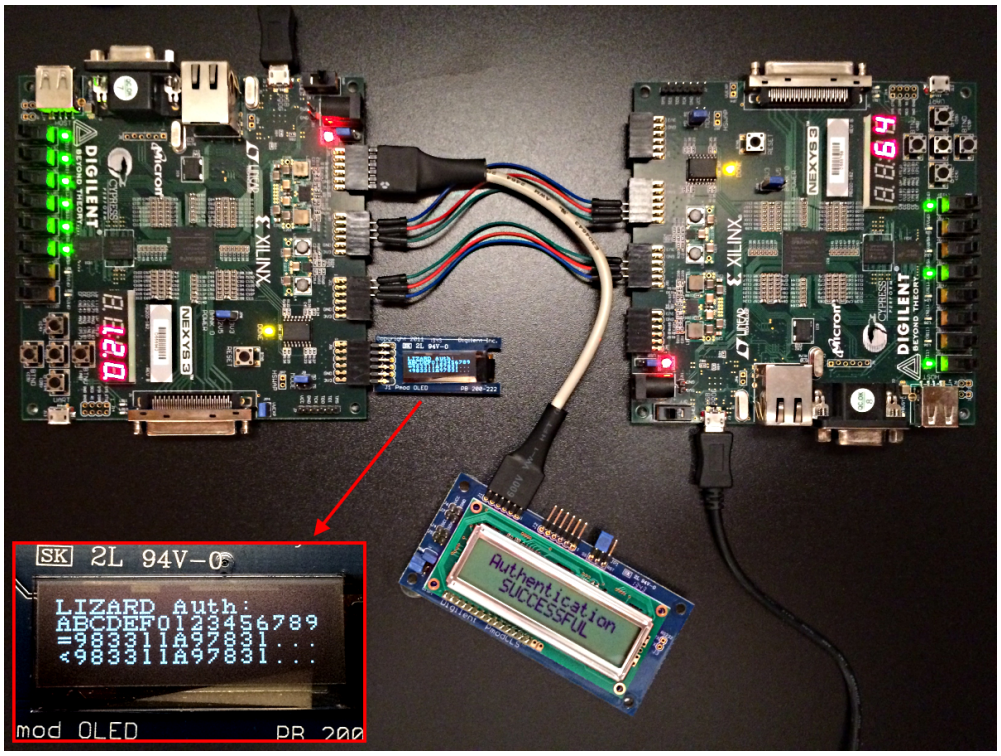

Figure 9.1: A basic FPGA prototype of LIZARD-based authentication.

colored wires connecting the two boards.[2] The verifier (i.e., the FPGA board on the left) is additionally equipped with two attached displays delivering information about the state of the authentication process. Moreover, the red numbers on the on-board seven-segment displays indicate, how many bits the respective board has already received in the course of the current authentication instance.

Figure 9.1 shows the state of a successfully completed authentication. For this example, we used the secret key `0x0123456789ABCDEF0123456789ABCD` and the public challenge (i.e., IV) `0xABCDEF0123456789`, corresponding to the third test vector which we provided for LIZARD in Appendix 8.A. The enlarged extract of the verifier's OLED display in the lower left corner of Fig. 9.1 is to be interpreted as follows: The second line shows the challenge sent to the prover. The third line (starting with '=') contains the first six bytes of the correct 120-bit keystream prefix computed by the verifier on the basis of the challenge and the secret key. The last line (starting with '<') contains the first six bytes

[2]The reason why we neither used the built-in communication capabilities of the FPGA test boards nor decided to attach some additional wireless communication hardware, is that, with regard to future analysis (e.g., concerning side-channel or man-in-the-middle attacks), we wanted to keep the setting as simple as possible.

of the 120-bit response received from the (here: valid) prover. As both bitstrings of size 120 bits matched, the verifier indicates 'Authentication SUCCESSFUL'.

Towards Privacy Preservation for classical LIZARD-based Authentication

In Section 3.1, where we explained the different concepts of *identification*, *authentication*, and *authorization*, we also briefly touched the attached problem of so-called *privacy preservation*. One particular aspect of this large field of research is the problem of *tracing*. Nowadays, most of us usually carry several RFID devices with us when we leave our homes, knowingly (as in the case of membership cards like the ecUM, cf. Subsection 3.2.2) or unknowingly (such as RFID tags contained in unsuspicious products like clothing). If, upon request by an RFID reader, corresponding tags freely reveal their identity (see, e.g., the UID of the MIFARE Classic-based ecUM), then an attacker could place malicious reading devices at various places in order to trace the movement of individuals (who, e.g., often tend to hold their whole wallet in front of a reader, allowing the respective device to potentially communicate with all contained RFID-based cards). Note, however, that this step of freely giving away an RFID tag's identity often actually serves the purpose of simplifying the subsequent authentication. As, if no group keys are used, it gives the corresponding RFID reader (i.e., the *verifier*) the possibility to load the respective 'tag-individual' secret key from its database. But even if group keys are used or, more simple, there is only one valid RFID tag (i.e., *prover*) for a certain reader (imagine, e.g., a single RFID key for your home's front door), thus removing the need for a tag to reveal its identity in advance, there is still the possibility of such tracing attacks through the information revealed in the course of authentication.

So let us consider the simple case that no explicit identification is necessary as there is only one valid prover for some verifier, i.e., this verifier holds only one secret key and expects correct answers in a challenge-response protocol based on this certain key. Let us further assume that the authentication is performed using the simple LIZARD-based scheme described above. Then an attacker could trace the prover (e.g., an RFID tag) easily by always sending the same challenge, as the tag would always answer with the same 120-bit keystream prefix belonging to this challenge (i.e., IV) and its secret key. That is, even though the attacker would not be able to recover the secret key or impersonate the prover in some other way, he would at least always know that he is communicating with the same RFID tag, e.g., allowing to trace its owner's movements.

As explained in Section 3.1 (at the example of block cipher-based authentication), one way to deal with this problem is to additionally introduce some random nonce x on the prover's side. But how should this nonce be used properly in the case of stream cipher-based authentication? We first describe two tempting, but *bad* ways:

- The random nonce could be XORed to the keystream in order to produce constantly changing responses (under the same challenge). Then, however, to allow for correct

verification, the prover would need to (publicly) communicate this nonce to the verifier for each authentication instance. But trivially, this would now also enable an attacker to cancel the effects of the nonce by XORing it to the response himself and obtaining the underlying keystream. Irrespective of the used nonces, this underlying keystream would still always be the same, as long as the challenge does not change.

- A more sophisticated approach could be to XOR the challenge and the nonce. In the case of our above LIZARD-based example, this would mean that the 64-bit IV is computed as $IV := c \oplus x$ for a 64-bit challenge c and a 64-bit nonce x. Again, the nonce x would have to be publicly transmitted to the verifier together with the corresponding 120-bit keystream prefix to allow for correct verification. But while this scheme can in fact thwart tracing, a much more severe security problem would arise here. More precisely, observing a single successful authentication instance would now allow an attacker to impersonate the valid prover in the future. As for some new challenge c', he could now simply compute his nonce as $x' := IV \oplus c'$, and his response consisting of the initially observed 120-bit keystream prefix (for the old challenge-nonce tuple (c, x) with $IV = c \oplus x$) together with his new nonce x' would be accepted by the verifier for c'.

In consequence, we suggest to realize privacy preservation for LIZARD-based authentication similar to how it was described for block cipher-based schemes in Section 3.1. That is, the IV on the basis of which the 120-bit keystream prefix for authentication is generated, should be computed as the concatenation $IV := c||x$ of a challenge c and a nonce x of 'suitable' sizes. The drawback of this approach is obviously that challenge and nonce now need to share the IV length of the stream cipher, just like they needed to share the block length of the block cipher-based example in Section 3.1. In our context of ultra-constrained RFIDs, such a solution, where, e.g., the challenge size is set to 44 bits and the nonce size is set to 20 bits (adding up to LIZARD's IV length of 64 bits), could in fact be sufficient for many application scenarios. Nonetheless, in the following Subsection 9.2.2, we suggest how LIZARD could be tuned especially for authentication purposes, thereby making full use of the possibilities offered by the underlying LIZARD-construction.

9.2.2 Further Optimizing LIZARD for Authentication

First of all, remember that the general LIZARD-construction (cf. Chapter 7) actually assumes *IV length = key length = inner state length.* In particular, the corresponding proof of the $(2n/3)$-security against generic TMD-TO key recovery attack was done in this scenario and the only reason why our instantiation LIZARD (cf. Chapter 8) with key size 120 bits has an IV size of 'only' 64 bits is that, as explained in Subsection 8.3.5, we saw no need for larger IVs in the context of lightweight stream ciphers and shortening the IV allowed to save a little bit of chip area. But given the new application scenario of

privacy-preserving authentication described above, the few additional gates required for using the full possible IV length of 120 bits in Lizard should be well worth the gain in security and, hence, broadened applicability.

Another property of the general Lizard-construction gaining particular relevance in the context of authentication is our conjecture made in Section 7.6 that for $R = n$ (i.e., if the packet size equals the size of the inner state), the Lizard-construction is $(2n/3)$-secure even against distinguishing attacks. However, we cannot use this property for optimizing our concrete instantiation Lizard w.r.t. authentication right away, as in Lizard, there is the possibility of so-called *IV collisions* (cf. Subsection 8.4.7), which result out of setting $S_{30}^{129} := 1$ in phase 3 of the state initialization of Lizard (cf. Subsection 8.2.2) and allow for corresponding distinguishing attacks (even though with a complexity larger than the claimed 60-bit security level against distinguishing). The reason for setting $S_{30}^{129} := 1$ in the first place was to avoid that, after phase 3 of the state initialization, NFSR1 (cf. Section 8.2) could have otherwise gotten stuck in the all-zero state. In order to resolve this issue for an authentication-focused variant of Lizard, we suggest that instead of setting $S_{30}^{129} := 1$, the term

$$\oplus \left(S_1^t \oplus 1\right) \left(S_2^t \oplus 1\right) \cdots \left(S_{30}^t \oplus 1\right)$$

should be added to the feedback function of the 31-bit NFSR NFSR1 of the original Lizard.[3] Thereby, NFSR1 obtains the full possible period of 2^{31} (instead of currently $2^{31} - 1$, cf. Subsection 8.3.1) and the need for overwriting one of its cells with 1 during phase 3 of the state initialization to avoid the all-zero state becomes obsolete. In consequence, also the possibility of IV collisions and corresponding distinguishing attacks is canceled. The reason why we did not do this already in the original Lizard lies in the fact that if S_{26}^t, S_{27}^t, S_{28}^t, S_{29}^t, S_{30}^t are involved in the generation of S_{30}^{t+1}, the speedup option as described in Subsection 8.3.4 is completely lost. Here, in the context of *authentication*, we believe that such a speedup option is not necessary, because in each instance, only a short, 120-bit long keystream prefix needs to be generated. Remember that, in Subsection 2.3.2, we explained that the common notion is that the whole process of authentication should not take more than 150 ms. Furthermore, as described, e.g., in Subsection 2.3.6, the prevalent clock rate for ultra-constrained RFIDs is 100 kHz. As Lizard's state initialization requires only 499 clock cycles (see Table 8.1 in Subsection 8.5.1), the generation of a 120-bit long keystream prefix takes only 619 clock cycles in total. Given a 100 kHz clock, this translates into only 6.19 ms, which is way below the 150 ms bound for authentication mentioned above.

[3] Note that the addition of this term can be expected to have negligible influence (if any influence at all) on the security of Lizard. This is due to the fact that the order of states of the old NFSR1's state cycle of length $2^{31}-1$ is completely preserved with only one exception: the all-zero state is now 'glued into' this state cycle between the states $(1, 0, \ldots, 0)$ and $(0, \ldots, 0, 1)$ in the form $(1, 0, \ldots, 0) \to (0, \ldots, 0) \to (0, \ldots, 0, 1)$.

We would like to point out that this is even faster than encrypting two 64-bit blocks of data with a serialized implementation of PRESENT, which would take $2 \cdot 563 = 1126$ clock cycles (see the numbers at the very beginning of this section). As explained at the end of Subsection 4.4.2, for realizing privacy-preserving authentication with PRESENT for a challenge size of 64 bits, two blocks of data would be necessary, which, in turn, would then also require to implement a suitable mode of operation (i.e., not ECB mode), which comes at significant additional costs. Given that (without this required mode of operation) a serialized implementation of PRESENT needs 1080 GE [RPLP08], whereas standard Lizard amounts to 1161 GE (see Table 8.1 in Subsection 8.5.1), it is hard to estimate which solution will finally be cheaper in terms of chip area (cf. Subsection 2.3.3) after our above suggestions of using a full 120-bit IV and extending the period of NFSR1 to 2^{31} have been implemented. However, it is safe to say that both numbers will be in the same ballpark, making our modified variant of Lizard a viable alternative to prevalent block cipher-based schemes when it comes to lightweight authentication.

In summary, as future work, we suggest to adapt standard Lizard as described above, i.e., to extend the IV to 120 bits and to modify NFSR1 such that it reaches full period 2^{31}, which makes setting $S_{30}^{129} := 1$ obsolete and removes the corresponding IV collision issues. In a privacy-preserving authentication scheme, this would, e.g., allow to choose a challenge of size 80 bits and a nonce of size 40 bits[4], which are then concatenated to obtain the 120-bit IV, on the basis of which a 120-bit keystream prefix for authentication is generated and transmitted to the verifier (together with the public nonce).

With regard to the respective security analysis, we suggest to particularly investigate whether the conjectured $(2n/3)$-security of the Lizard-construction against generic TMD-TO *distinguishing attacks* for $R = n$ applies to this scheme. Moreover, special care has also to be taken w.r.t. the effects that an increased IV length may have in the context of other cryptanalytic techniques than TMD-TO.

9.3 More on Stream Ciphers that Continuously Use the IV

In Chapter 6, we gave an overview over the new class of so-called of *small-state stream ciphers*, whose volatile inner state size is less than twice the key size (resp. the targeted security level against key recovery). At the moment, this class consists of the three Sprout-like ciphers (i.e., ciphers that continuously use the secret key) Sprout [AM15], Plantlet [MAM17], and Fruit [GHX16] as well as our new stream cipher Lizard [HKM17b], which, with the Lizard-construction (cf. Chapter 7), uses a different, non-Sprout-like approach to achieve increased security against TMD-TO *key recovery* attacks. Remember that, in Subsection 6.3.1, we also presented a generic distinguisher against Sprout-like ciphers with a complexity significantly smaller than that of exhaustive key search. To thwart this

[4]Remember that the nonce only serves to thwart tracing, for which 40 bits should already be sufficient.

kind of attack, we suggested in Section 6.5 that these ciphers should not only continuously use the secret key, but also continuously use the public IV in their state update.

However, in our original publication [HKMZ18] underlying Chapter 6, we concluded for other, non-Sprout-like stream ciphers:

> "Other small-state ciphers like Lizard [27] would hardly benefit for the following reason: If the secret key is not used after the state initialization, there is always the possibility of a TMD tradeoff inner state recovery attack like those by Babbage [3] or Biryukov and Shamir [10]. Such an attack will have complexity half the size of the volatile inner state, independent of whether the IV is continuously used during state update, because the IV is public and the attacker will be able to evaluate the function $(\textit{volatile inner state}, IV) \longrightarrow \textit{keystream block}$ for randomly chosen volatile inner states and the proper IV. The only advantage of continuously using the IV would be that TMD tradeoff precomputations could be prevented as, if it is not a chosen-IV attack scenario, the attacker would have to wait for which IV he actually obtains the required/attacked keystream." [HKMZ18]

In this section, we will argue why we believe that this restriction of continuous IV use to the context of Sprout-like ciphers was probably premature. More precisely, we will explain that, under certain conditions, general stream cipher constructions can benefit from this new design idea as well.

9.3.1 Continuous IV Use with Stream Ciphers working in Packet Mode

At FSE 2017, our new small-state stream cipher Lizard (cf. Chapter 8) was introduced. Apart from its provable security against TMD-TO-based key recovery attacks, one of its prominent characteristics is that it explicitly targets *packet mode* scenarios, i.e., application contexts where only a moderate number of keystream bits needs to be generated per key/IV pair. More precisely, Lizard allows to generate up to 2^{18} keystream bits per key/IV pair, which would be sufficient for many existing communication scenarios like Bluetooth, WLAN, or HTTPS as explained in Section 8.1.

For Lizard, we claim 80-bit security against key recovery and 60-bit security against distinguishing. The lower security level against distinguishing results from the fact that, as pointed out in Subsection 8.4.2, TMD-TO-based inner state recovery attacks like those of Babbage [Bab95] or Biryukov and Shamir [BS00] are still possible, because after the state initialization of the cipher has been completed, the keystream is then generated solely in dependence of the resulting (121-bit wide) initial state, exactly as it is done by classical stream ciphers like Trivium [CP05] (cf. Subsection 5.2.3) and Grain [HJMM08] (cf. Subsection 5.2.4).

In the following, we will explain that by using a stream cipher in packet mode together with continuously involving the IV in the state update, TMD-TO-based inner state

recovery attacks can actually be thwarted. For a technical discussion w.r.t. why the assumption of continuous IV availability for encryption purposes is actually plausible in many cases, we refer the reader to Section 6.5.

For the sake of brevity, let us only quickly recall the tradeoff curves and restrictions of the relevant attacks. The TMD-TO attack of Babbage [Bab95] has the tradeoff curve $TM = N$ with $P = M$ and $T \leq D$ (where $T < D$ means that some of the available data is ignored during the online phase of the attack). The TMD-TO attack of Biryukov and Shamir [BS00] is based on Hellman's time-memory tradeoff attack for block ciphers [Hel80] and has the tradeoff curve $TM^2D^2 = N^2$ with $P = N/D$ and $T \geq D^2$. While the use of so-called *BSW-sampling* [BSW01] allows to relax the restriction $T \geq D^2$ in the Biryukov-Shamir attack, the tradeoff curve $TM^2D^2 = N^2$ and the relation $P = N/D$ remain unchanged. For further details (also w.r.t. the involved cost metrics) please see, e.g., Subsection 8.4.2. Keep in mind, however, that, as explained before, in this thesis we always refer to the *overall complexity* of an attack (including precomputation) as the relevant benchmark w.r.t. our security claims.

Attacking a Packet-Mode, Continuous-IV-Use Stream Cipher via Classical TMD-TO Attacks

Let us now consider an arbitrary KSG-based stream cipher with a volatile inner state size of 100 bits, a key size of 80 bits, and an IV size of 80 bits. Moreover, let us assume that this stream cipher is used in packet mode with a limit of 2^{20} keystream bits per key/IV pair and that, during keystream generation, the IV is continuously employed in the state update (like the key is continuously employed in the state update with Sprout-like ciphers). Then, a generic TMD-TO attacker in the spirit of [Bab95] and [BS00] has the following two possibilities:

1. He uses the fact that the IV is public and tries to invert the resulting function $F_{IV}(\textit{volatile inner state}) \longrightarrow \textit{keystream block}$ by randomly choosing volatile inner states of 100 bits and looking for a collision in the keystream of the attacked packet. As the data there is limited to 2^{20} bits, both Babbage and Biryukov-Shamir TMD-TO attacks will lead to overall complexities of at least 2^{80} (the cost of exhaustive key search).[5] In particular, observe that using data from another packet (generated under another initialization vector IV') leads to an independent birthday experiment, as the attacker then has to evaluate the different function $F_{IV'}$. That is, data obtained on the basis of different IVs cannot be used here to achieve an 'overall birthday bound-based' advantage.

[5] More precisely, for the Babbage attack, the restrictions $T \leq D$ and $P = M$ together with the tradeoff curve $TM = N$ imply $M \geq 2^{80}$ and, hence, also $P \geq 2^{80}$ in our example. Similarly, for the Biryukov-Shamir attack, if $D \leq 2^{20}$, with or without BSW-sampling, due to $P = N/D$ the precomputation complexity is at least 2^{80} for $N = 2^{100}$.

2. He ignores the information about the IV and uses data from various packets, i.e., he tries to invert the function $F(\textit{volatile inner state}, IV) \longrightarrow \textit{keystream block}$. Then, however, he has to sample randomly from the space *Volatile Inner States* $\times$ *IVs* of size 2^{100+80}, leading to an overall attack complexity above exhaustive key search.

Also note that the less common approach of attacking the function $F(\textit{Key}, IV) \longrightarrow \textit{keystream prefix}$ for a collection of keystream prefixes obtained on the basis of different IVs (as taken in, e.g., [HS05] and [DK08]), would not work, as well, because the key size and the IV size add up to 160 here, leading to an overall complexity of at least 2^{80}.[6]

We would like to point out that for scenarios where different (e.g., session) keys are used, it is important to deprive an attacker of the possibility to collect more data based on a situation where the same IVs are used in different sessions, as this would eventually allow him to recover a secret inner state (and, possibly, by clocking the cipher back, even the session key) of one of these sessions. Potential countermeasures are here to keep the IV counter over sessions (instead of resetting it) or to choose the IVs at random. If IV reset is inevitable, one could also increase the volatile inner state from 100 bits to 120 bits in our above example and limit the number of session keys to a (in our opinion plausible) number of 2^{20}.[7] This would allow an attacker to obtain at most 2^{40} keystream bits generated under the same IV, and, for N now being 2^{120}, again lead to overall complexities of at least 2^{80} for the above classical TMD-TO attacks.

Towards Practical Instantiations of our Design Idea

While the target of this section is mainly to stimulate a general discussion about our new design idea, we still would like to point out an important aspect which has to be treated with special care in potential future instantiations: the concrete way in which the IV is involved in the state update.

As we have seen above, the security of our approach against generic TMD-TO attacks is based on the assumption that an attacker will not be able to use the same function F for attacking different packets. However, certain correlations between the functions F_{IV} and $F_{IV'}$ for different initialization vectors IV and IV' are unavoidable. For example, given some arbitrary volatile inner state S, the keystream blocks $F_{IV}(S)$ and $F_{IV'}(S)$ will have the same first (i.e., leftmost) bit, as in our new design approach, the IV is continuously involved in the state update, but does not enter the cipher's output function directly. Note that the same applies to Sprout-like ciphers w.r.t. the continuous use of the secret key (cf. Subsections 6.2.1 to 6.2.3).

[6] To avoid any misconceptions, we would like to point out that also Grain v1 [HJM06], which has key size 80 bits but IV size only 64 bits, would not succumb to this attack in a *single-key scenario*. This is due to the fact that under a fixed key, an attacker would only be able to collect at most 2^{64} keystream prefixes, leading to an overall complexity above 2^{80} for this type of TMD-TO attack. But by choosing key length = IV length = 80 bits in our example, we can resist this attack even for *multiple-key scenarios*.

[7] With such state sizes, we are in the area of lightweight cryptography and corresponding applications.

In this section, we do not discuss the potential security implications of such 'subtle' correlations, as we are focusing on generic TMD-TO attacks, which treat the function F as a black-box. In particular, we also do not consider cipher-specific approaches which might perform on-the-fly adaptions of F in order to make it applicable also to data obtained from other packets. However, in concrete instantiations, it might be required to fend off such non-generic attacks, which have yet to be developed.

But one important rule for future instantiations of our new design approach can be formulated right away: it is of vital importance that each bit of an initialization vector IV must always have the potential to influence the keystream block $F_{IV}(S)$ generated on the basis of IV and a volatile inner state S. To illustrate this rule, imagine the bad *round IV function* $IV^{\text{round}} = IV_{(2t) \bmod 80} \cdot IV_{(2t+1) \bmod 80}$ for an initialization vector $IV = (IV_0, \ldots, IV_{79})$ of length 80 bits and t denoting the corresponding keystream generation step of the KSG. In this case, a (chosen-IV) attacker could, e.g., focus on the set (of size about $2 \cdot 2^{40}$) of all IVs which either have only zeros at their even index positions or have only zeros at their odd index positions. Given the above bad round IV function, all these IVs would never influence the state update during keystream generation, allowing an attacker to target all the corresponding keystream packets with a single function F (using the classical, generic TMD-TO attacks), thereby defeating our assumption about the limit of data available to him.

Also less extreme scenarios could be imagined, where an attacker knows that for certain blocks of a number of keystream packets, not all IV bits are involved in the underlying state updates. Note that, due to their 'complicated' *round key functions*, the continuous-key-use stream ciphers Sprout and Fruit have already succumbed to similar TMD-TO attacks in the past w.r.t. the involvement of the key bits (see, e.g., [EK16] and Subsection 6.3.2). In fact, in Section 6.4 we hence conjectured that the round key function of Plantlet, which, in each step, simply cyclically XORs one key bit to the feedback bit of the NFSR, is actually the optimal one for thwarting generic TMD-TO attacks. In the same spirit, we also believe that for instantiations of our new design approach, cyclically XORing one IV bit per step to the volatile inner state should be the way to go.

Summing up, creating a concrete, *secure* instantiation of our new design idea of combining *continuous IV use* and *packet mode* will definitely be a challenging task, as, besides TMD-TO attacks, a wide range of other established cryptanalytic techniques (such as correlation attacks, algebraic attacks etc.) will have to be considered, as well. Nonetheless, we are convinced that the result will be worth the effort, not only from the practical viewpoint of obtaining an even more resource-efficient stream cipher, but also with regard to an improved theoretical understanding of the boundaries of lightweight (small-state) design. In this respect, we particularly suggest to investigate whether for the principle of continuously using the IV, similar security proofs as for the Lizard-construction (cf. Chapter 7) or for the hardness of Trivium and Grain w.r.t. generic TMD-TO attacks (see [Kra17]) can be found.

9.4 BDD and SAT Attacks

As we have seen in the previous chapters, *long-keystream attacks* such as TMD-TO attacks or correlation attacks, which require an attacker to get hold of very large keystream pieces (in the dimension of millions/billions of bits), play a dominating role in the design and analysis of contemporary stream ciphers. This is reflected in a multitude of corresponding publications (see, e.g., [MS88], [Bab95], [BS00], [LMV05], [HS05], [Bjø08], [BGJ09], [EK16], [MSS+17], [ZGM17]) and also in the fact that we considered it particularly important that our new lightweight stream cipher Lizard (cf. Chapter 8) is *provably* resistant against generic TMD-TO key recovery attacks due to implementing the Lizard-construction design principle (cf. Chapter 7). However, especially in the context of ultra-constrained RFID devices (cf. Chapter 2), a real-world attacker may easily run into a situation where only few bits of known keystream are actually available to him. For this reason, we consider *short-keystream attacks*, which often require only the information-theoretic minimum of keystream bits, an important topic of future research.

Two prominent examples of such short-keystream attacks are attacks based on binary decision diagrams (BDDs) and SAT solvers. In the context of stream cipher analysis, the goal of corresponding attacks is usually to recover the KSG's secret initial state. As pointed out previously, for popular stream ciphers like Trivium [CP05] and Grain [HJMM08], the state initialization algorithm is efficiently invertible (see Section 5.3) and, hence, initial state recovery is equivalent to key recovery. The reason for targeting the initial state instead of trying to directly recover the secret key lies in the fact that the latter approach requires to additionally consider those state transitions which are performed as part of the ciphers' initialization algorithms. This, however, massively increases the size of the corresponding BDD or SAT instances.

The first application of BDDs in the field of cryptanalysis is due to Krause, who showed in [Kra02] that, e.g., for an A5/1-type (cf. Subsection 5.2.2) KSG of state size n, the secret initial state can be computed from the first $\lceil 1.14n \rceil$ bis of the keystream in time $n^{O(1)}2^{0.6403n}$. Similar attack complexities were shown for E_0-type (cf. Subsection 5.2.1) KSGs and the self-shrinking generator [MS94] in the same paper. Despite the popularity of [Kra02] (measured by the number of citations) and a follow-up publication by Krause and Stegemann in 2006 [KS06], the technique of BDD-based cryptanalysis passed again out of the community's mind in the following years. To the best of our knowledge, the most recent relevant contributions in this field now date back to 2007 and 2008, respectively. More precisely, in 2007 Stegemann [Ste07a] performed a BDD attack against Grain-128 [HJMM06], and in 2008, Eibach, Pilz, and Völkel [EPV08] considered BDD attacks (for comparing efficiencies) in their SAT-based attack against a reduced version of Trivium [CP05] called *Bivium*.

The reason why BDD attacks subsequently sank into oblivion is probably already contained in Stegemann's slides [Ste07b], with which he presented his attack [Ste07a]

against Grain-128 at SAC 2007. There, he gives the complexity of his BDD-based initial state recovery attack as "time and memory $\approx 2^{187}$" and compares it to the classical TMD-TO initial state recovery attack (see Theorem 7.1 in Chapter 7) requiring "2^{128} operations and keystream bits" (note that Grain-128 has an inner state size of 256 bits). Stegemann concludes in his presentation that "[the] BDD-attack [is the] best non-trivial short keystream attack" [Ste07b]. While this statement was undoubtedly correct, it also showed that, at that time, BDD attacks started to become outdated. The initial BDD attacks of Krause [Kra02] targeted stream ciphers whose inner state size equaled (or was very close) to the key size. Therefore, given a corresponding cipher with an efficiently invertible state initialization algorithm (such as A5/1; cf. Subsection 5.2.2), an initial state recovery attack with complexity below that of 'exhaustive initial state search' immediately yielded a key recovery attack with complexity below exhaustive key search and, hence, broke the cipher. But as a consequence of these attacks (and, in particular, the aforementioned TMD-TO attacks of Babbage [Bab95] and Biryukov and Shamir [BS00]), newer ciphers such as Trivium [CP05] and Grain [HJMM08] now featured inner states whose size was at least twice the key size. In this new context, BDD attacks were not able to deliver 'real cipher breaks' any longer and, as they were also inferior (w.r.t. the overall attack complexity) to generic TMD-TO attacks, could now only be marketed as the "best non-trivial short keystream attack[s]" [Ste07b]. Given the huge complexities of, e.g., 2^{187} for Stegemann's BDD attack against Grain-128, the *short-keystream* argument was obviously not interesting enough for the community to further study this type of attack.

In our opinion, this lack of corresponding research and respective publications after 2008 is very unfortunate. The reasons for this are twofold. First, with the advent of ultra-constrained RFIDs (cf. Chapter 2), which, due to their application context, often produce only very moderate amounts of keystream during their whole lifetime, short-keystream attacks are becoming increasingly important now. Furthermore, the extremely restricted resources of such devices have inspired the new class of *small-state stream ciphers* as discussed in Chapter 6. As their name suggests, these ciphers try to minimize the size of the internal state, thereby once more closing the aforementioned gap between state size and key size and thus making BDD attacks interesting again.

Second, and even more important, SAT attacks have already seen a huge revival in the field of stream cipher cryptanalysis (see, e.g., [SNC09], [DA14], [MSBD15], [Ban15], [SSMC17]). This is partly due to the fact that (unlike, e.g., in the above paper of Eibach, Pilz, and Völkel [EPV08] from 2008) SAT attacks are now not only used as a 'standalone method' any longer, but are also employed as a tool in other, usually non-generic attacks (see, e.g., [DA14] or [SSMC17]). In Subsection 9.4.2, we will explain why we actually believe BDD attacks to be even superior to SAT attacks in that respect, making them a promising subject of future research.

The basic idea behind SAT attacks is very similar to that of BDD-based attacks (see Subsection 9.4.1). In a nutshell, the attacker makes use of two different types of

information: (I) From the publicly available definition of the stream cipher, he knows the KSG's feedback function(s) and, thus, how the later bits of the secret internal bitstream depend on the secret initial state. (II) He also knows the KSG's output function and, by observing keystream bits, can narrow down the possible inputs (i.e., combinations of bits of the internal bitstream) responsible for these output bits. In a SAT attack, these two types of information are coded into one large conjunctive normal form (CNF) formula such that, once enough keystream has been observed, there is only satisfying assignment to the corresponding variables, revealing the secret initial state (see, e.g., [BCJ07] for further information).

Obviously, for a properly designed stream cipher, building and solving such a CNF formula straightforwardly should be infeasible. In consequence, Eibach, Pilz, and Völkel [EPV08] consider the case that certain parts of the secret initial state are guessed in advance and then used to simplify the resulting SAT instances. Note that this also provides a trivial way of parallelizing the attack by distributing SAT instances with different pre-guessed parts of the initial state to different SAT solvers running on separate CPUs. As pointed out above, another way of reducing the complexity of the respective SAT instances to a feasible level is to additionally use information about the internal bitstream obtained as part of another, non-generic attack (such as differential fault analysis; see, e.g., [SSMC17]).

In our opinion, the further optimization potential for SAT-based cryptanalysis is rather limited. This is due to the fact that corresponding attacks usually employ SAT solvers in a black-box manner. That is, all necessary information is coded into one large CNF formula, which is then fed to a SAT solver (such as CryptoMiniSat 4 [Soo15]) in the hope that its 'secret inner workings' will find the satisfying assignment in feasible time. The cryptanalyst's scope of influence is more or less confined to slightly optimizing the CNF formula in advance (e.g., through introducing new variables in order to restrict the size of clauses) or, in the context of pre-guessing bits, to choosing the respective bit positions. In [EPV08], where both of these techniques are applied, several guessing strategies are compared experimentally, but the theoretical explanations of the corresponding results are limited to some few conjectures. So this might in fact be a point where future research could prove fruitful. However, as we will argue in the following, we are convinced that in the field of BDD-based attacks, much more interesting work is to be done and, correspondingly, much more improvements can be made.

9.4.1 A Hands-on Introduction to (O)BDD Attacks

As a lengthy introduction to the theory behind BDD attacks would go far beyond the scope of this *Future Research Directions* chapter, we will instead refer to a simple 'toy example' of BDD-based cryptanalysis, which is supposed to contain just enough information such that also a non-expert reader will be able to understand our suggestions for future improvements provided in Subsection 9.4.2. For a more formal introduction to

the field of BDDs in general and BDD-based cryptanalysis in particular, we refer the reader to [Weg00] and [Kra02], respectively.

Very important, however, is to keep in mind that, in the following, we will refer to a special type of BDDs called *ordered binary decision diagrams (OBDDs)*. In a nutshell, OBDDs have two distinctive properties: (I) For each path from the root to a sink of the OBDD, it holds that each variable may occur at most once on this path. (II) The variables occur in the same order on all these paths of the OBDD.[8]

In consequence of this, OBDDs have several nice properties which we can benefit from in our cryptanalysis of stream ciphers. In particular, crucial operations such as OBDD minimization and OBDD conjunction can be performed efficiently. Moreover, OBDDs allow to efficiently enumerate the set of all satisfying assignments, something which will be very important later on in Subsection 9.4.2.

To fill the above notions about OBDDs with life and to get an idea of how OBDD-based cryptanalysis basically works, let us consider the following 'toy example' of a simple KSG with inner state size five, whose secret inner bitstream $x_0, x_1, \ldots, x_4, x_5, \ldots$ (after state initialization has been completed) is defined by the feedback relation

$$x_{t+5} := x_t \oplus x_{t+2} \quad \text{for } t \geq 0, \tag{9.1}$$

where $(x_0, \ldots, x_4)$ denotes the secret initial state. Moreover, let the output function of this KSG be given as

$$z_t := x_{t+2} \cdot x_{t+4} \quad \text{for } t \geq 0. \tag{9.2}$$

It can be easily checked that for this KSG, the initial state $(x_0, \ldots, x_4) = (0, 1, 1, 0, 1)$ leads to the keystream prefix

$$(z_0, z_1, \ldots, z_9) = (1, 0, 1, 0, 1, 0, 1, 0, 0, 0)\,.$$

OBDD-based cryptanalysis now proceeds as follows. An attacker who gets hold of the above keystream prefix and wants to recover the underlying secret initial state starts by turning his information about step $t = 0$ into an OBDD. From Eq. (9.1), he knows that

$$x_0 \oplus x_2 \oplus x_5 = 0 \tag{9.3}$$

must hold w.r.t. the newly generated inner state stream bit x_5. This knowledge is represented by the OBDD R_0 depicted in Fig. 9.2. Note that those assignments to the variables x_0, x_2, x_5 satisfying Eq. (9.3) are exactly the satisfying assignments of OBDD R_0 (i.e., those assignments which lead to the OBDD's 1-sink).

[8] Note that when we work with several OBDDs in our cryptanalysis, all of these OBDDs will be defined over the same set of variables (even though they do not necessarily contain all of them), and the variable order will be the same (i.e., 'global') for all OBDDs.

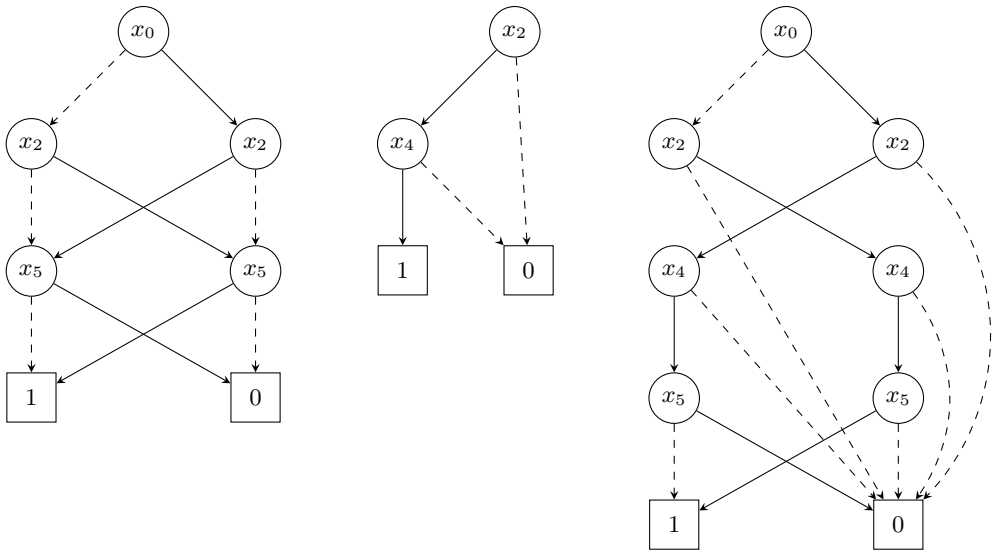

Figure 9.2: The OBDDs R_0 (left), Q_0 (middle), and $P_0 = R_0 \wedge Q_0$ (right). The solid (resp. dashed) edges denote that the variable corresponding to the source node of the edge takes the value 1 (resp. 0).

Also at $t = 0$, the attacker learns from the observed first keystream bit $z_0 = 1$ together with Eq. (9.2) that

$$x_2 \cdot x_4 = 1 \tag{9.4}$$

must hold. This knowledge is represented by the OBDD Q_0 depicted in Fig. 9.2.

Through AND-synthesis of the two OBDDs R_0 and Q_0, the attacker finally obtains the OBDD P_0 depicted in Fig. 9.2, whose satisfying assignments are exactly those assignments to the variables x_0, x_2, x_4, x_5 which simultaneously fulfill Eqs. (9.3) and (9.4).

For the next step, $t = 1$, the attacker proceeds analogously. More precisely, he builds the OBDDs R_1 and Q_1 corresponding to the relations

$$x_1 \oplus x_3 \oplus x_6 = 0 \tag{9.5}$$

and

$$x_3 \cdot x_5 = 0 \tag{9.6}$$

(as $z_1 = 0$), respectively. The new main OBDD P_1 is computed as

$$P_1 := P_0 \wedge R_1 \wedge Q_1.$$

Its satisfying assignments are exactly those assignments to the variables x_0, x_1, x_2, x_3, x_4, x_5, x_6 which simultaneously fulfill Eqs. (9.3) to (9.6). Unfortunately, the OBDD P_1 is already too large to be depicted here.

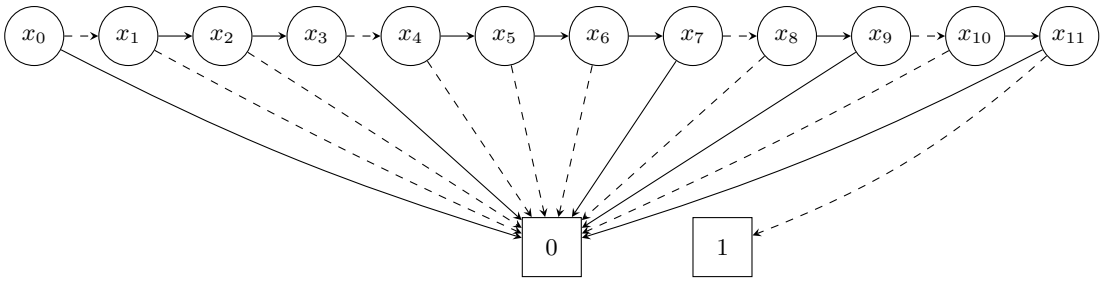

Figure 9.3: The OBDD P_6, which contains the solution of our cryptanalysis. For space reasons, we rotated the OBDD (except the sinks) by 90 degrees. The root of the tree is the node labeled with x_0.

The general attack strategy is now as follows. The attacker will treat the subsequent rounds $t = 2, 3, \ldots$ accordingly, obtaining further growing OBDDs P_2, P_3, and so on. However, as explained in detail in [Kra02], at some point, the size of the OBDDs P_t will eventually reach a maximum and henceforth (usually quickly) decrease. Note that this maximum actually dominates the overall complexity of the attack.

The reason, why, from some point on, the size of the OBDDs P_t starts to decrease again, lies, roughly speaking, in the fact that the more keystream bits are considered, the smaller becomes the number of inner state streams which are consistent with these keystream bits as well as with the KSG's feedback relation (for further details, again, we refer to [Kra02]). In the case of our above 'toy example', after only seven steps, the main OBDD P_6 has degraded into a list as depicted in Fig. 9.3. The only satisfying assignment to the first twelve bits of the inner state stream can be derived from P_6 directly as

$$(x_0, x_1, x_2, x_3, x_4, \ldots, x_{11}) = (0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0)\,.$$

The first five bits $(x_0, x_1, x_2, x_3, x_4) = (0, 1, 1, 0, 1)$ are the initial state that underlies the attacked keystream prefix and, hence, represent the solution of our OBDD-based cryptanalysis. From this initial state, an attacker can now generate the full keystream.

9.4.2 Towards More Efficient OBDD Attacks

Based on the notion of OBDD-based cryptanalysis established in Subsection 9.4.1, we are now ready to sketch our new ideas of how to improve such attacks.

OBDDs as a Tool in Combined Attacks

As pointed out at the beginning of this section, SAT attacks have recently seen increased use as tool in other, non-generic attacks. We believe that OBDDs are even more suitable in this respect. This is due to the fact that, as explained, SAT solvers are usually employed in a black-box manner. OBDDs, on the other hand, can efficiently provide

valuable information even in the midst of an ongoing attack. Look, for example, at the OBDD P_0 in Fig. 9.2 of Subsection 9.4.1, whose satisfying assignments are exactly those assignments to the variables x_0, x_2, x_4, x_5 which simultaneously fulfill Eqs. (9.3) and (9.4). Enumerating these satisfying assignments is efficiently possible by (virtually) flipping the direction of all edges and computing all paths which lead from the 1-sink to the root (here: the node labeled x_0). Each of these paths (and no other) represents a satisfying assignment. W.r.t. P_0, we hence immediately see that $(x_0, x_2, x_4, x_5) = (1, 1, 1, 0)$ and $(x_0, x_2, x_4, x_5) = (0, 1, 1, 1)$ are the only assignments simultaneously satisfying Eqs. (9.3) and (9.4). The same approach is obviously possible also during the later stages of the attack, e.g., for P_4. We consider this a very interesting property in the context of future *combined attacks*, where OBDDs are 'only' used as a tool together with other, non-generic attack techniques (such as differential fault analysis or BSW-sampling; see, e.g., Subsection 8.4.11). That is, during the attack, such intermediary information about satisfying assignments can be supplied to other 'attack components' in order to optimize their workings. In turn, corresponding results can subsequently be fed back to the OBDD. This kind of constant interaction in both directions is very hard (if not impossible) to efficiently realize with SAT solvers.

Systemizing and Evaluating Reordering Heuristics for Cryptanalysis

The advantages of OBDDs (e.g., w.r.t. the efficient enumeration of satisfying assignments) come at a price, however. This price is their memory consumption. While even for large problem instances, SAT solvers usually require very few memory, OBDDs can grow heavily in the course of a real-world attack (see below). One way to counter this, is to use *variable reordering*. To give the reader and idea of the results, we used Rudell's *sifting algorithm* [Rud93] with the CUDD package [Som15] for BDD manipulation of Somenzi in order to reduce the size of the OBDD P_0 from Fig. 9.2 in Subsection 9.4.1 based on a new order of variables. The result is depicted in Fig. 9.4 and has now only five inner nodes instead of originally seven.

In the theory-focused papers of Krause [Kra02] and Stegemann [Ste07a], such reordering techniques were not considered, as the effects of the corresponding heuristics are very hard to grasp in a theoretical model. In their BDD-based attack against Bivium, the authors Eibach, Pilz, and Völkel state, however:

> "The CUDD library does not support an explicit minimisation operation, but the resulting BDD of an AND operation is already minimised (for a fixed variable order). The reordering operation tries to further minimize the BDD by reordering the variables. This operation requires most of the time in the BDD attack and is performed every few steps (based on a heuristic). The reordering operation is also just a heuristic and does usually not find the best ordering of the variables. However it significantly reduces the size of the BDD (up to a factor of 10) and makes future operations faster." [EPV08]

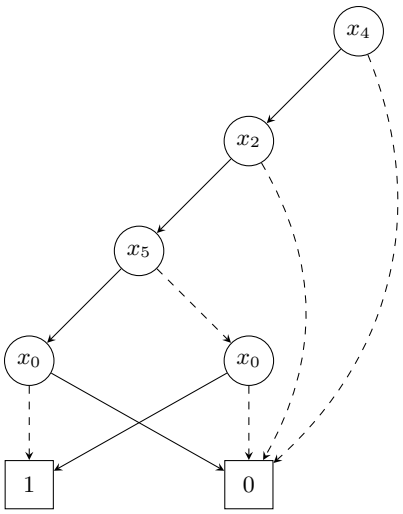

Figure 9.4: The OBDD P_0 after a reordering of variables using Rudell's *sifting algorithm* [Rud93] with the CUDD package [Som15] of Somenzi. The new root is the node labeled x_4.

Unfortunately, this is the only information w.r.t. reordering provided in the above paper. Given the enormous significance of space reduction for the future applicability of OBDD attacks together with the large body of work on corresponding reordering techniques (see, e.g., [FMK91], [ISY91], [Rud93], [PSP94], [PS95], [BLW95]), we consider it a very important next step to perform an experimental evaluation of the suitability of the respective approaches in the context of cryptanalysis. Clearly, it would also be very beneficial to integrate the general concept of variable reordering into the theoretical model of BDD attacks by Krause [Kra02], e.g., in order to identify promising variable orders for the respective attack instances in advance and to obtain corresponding bounds on the sizes of the resulting optimized BDDs. We expect, however, that this will constitute an extremely challenging task.

But not only the question of *which* reordering algorithm should be used is of importance, but also the *when* matters. The above statement "The reordering operation tries to further minimize the BDD by reordering the variables. This operation requires most of the time in the BDD attack and is performed every few steps (based on a heuristic)." [EPV08] of Eibach, Pilz, and Völkel suggests that they used CUDD's automatic reordering feature, which autonomously triggers reordering every few steps based on certain parameters. In our preliminary experiments, we made the experience that manually choosing the execution point of reordering leads to significantly better results. In particular, triggering reordering just once a few steps (in terms of t) before the main OBDD P_t would have reached its maximum size proved clearly superior to constant reordering. However, as these are just some first observations, we believe that more research (and, in particular, a corresponding theoretical foundation) is required in this respect.

Reordering in the Context of Parallelization

When it comes to real-world attacks, their potential to be parallelized is of enormous practical relevance. In our experiments, we found that SAT attacks are currently superior in that respect. The reasons for this are twofold. First, as already mentioned before, SAT attacks consume very few memory. Second, to the best of our knowledge, the only parallelization strategy suggested for BDD and SAT attacks so far is the trivial approach of splitting the problem up into completely independent instances by pre-guessing certain bits of the initial state (see, e.g., [KS06] and [EPV08]). While this way of sharing the workload over different computational units is very easy to realize, it is, at the same time, clearly suboptimal, as all but one unit will work on the basis of wrong information.[9] In a later paragraph of this subsection, we will suggest a new, non-trivial strategy for parallelizing BDD attacks, which allows the computational units to work *together* (on the correct information) in order to find the solution.

Despite the subobtimality of parallelization via pre-guesssing, SAT attacks (unlike BDD attacks) still perform rather well in this scenario, as our experiments have shown. More precisely, we considered a simple KSG of size 39 bits, whose inner state stream is defined by the relation

$$x_{t+39} := x_t \oplus x_{t+13} \oplus x_{t+5} \cdot x_{t+17} \oplus x_{t+24} \cdot x_{t+29} \quad \text{for } t \geq 0, \tag{9.7}$$

where $(x_0, \ldots, x_{38})$ denotes the secret initial state, and whose output function is

$$z_t := x_{t+9} \oplus x_{t+19} \oplus x_{t+19} \quad \text{for } t \geq 0. \tag{9.8}$$

Performing a keystream-only initial state recovery attack against this KSG using the SAT solver MiniSat 2.2 [ES10] consumes (even without any pre-guessed initial state bits) less than 2 MB of main memory on a standard PC. For comparison, a straightforwardly implemented OBDD attack using the CUDD package [Som15] requires, under the same conditions, almost 900 MB of main memory.

This extreme memory consumption of BDD attacks currently limits their potential to be used in parallelized cryptanalysis, because there, random-access memory (RAM) is often a scarce resource. For example, on a system equipped with four Intel Xeon Phi Coprocessor (Model: 31S1P) cards, we were still easily able to employ each cards full computational power (i.e., 57 cores with 1.10 GHz each) in the course of a parallelized SAT attack (using pre-guessing) against a slightly larger KSG than our above example.

[9] Consider, e.g., the case of two instances obtained on the basis of pre-guessing the solution's first bit. Thus, only one of both instances will work with the correct information, while the other one will practically spend its efforts in vain. The suboptimality of this strategy becomes obvious when considering the extreme case that actually all bits of the solution are pre-guessed, as then, one effectively ends up with an utterly inefficient variant of 'exhaustive initial state search', where each initial state test is performed by building a separate BDD or SAT instance, instead of simply running the KSG on this initial state candidate and comparing the resulting keystream prefix to the observed correct one.

A similar BDD-based attack, however, was not possible, as each of those Xeon Phi Coprocessor cards is only equipped with a total of 8 GB of memory.

The use of variable reordering may be one way to alleviate this problem. For example, by manually triggering reordering (using Rudell's *sifting algorithm* [Rud93]) just once a few steps before the OBDD in our attack against the above KSG of size 39 bits had reached its maximum size, we were able reduce the memory consumption from almost 900 MB to about 237 MB. The problem here, however, was that the respective reordering step alone took about 22 minutes, whereas the whole above OBDD attack without reordering had taken 'only' 215 seconds. As the CUDD package [Som15] allows to pre-set some own variable order, we were naturally interested in what would happen if we used the variable order resulting from the above reordering step already from the beginning (without any further reordering). And in fact, the whole attack now required only 158 MB and 24 seconds.[10]

Now executing an OBDD attack, thereby discovering a 'good' variable order, and then repeating this same attack more efficiently based on the new variable order is obviously not very exciting. What is, however, is an observation that we made during our experiments. In the course of a parallelized OBDD attack (using pre-guessing) against the above KSG, we discovered that a 'good' variable order found under an arbitrary choice of the initial state's pre-guessed part (i.e., even for a wrong one), also massively reduced the memory and time consumption of the other attack instances of the parallelized attack when pre-set there. In consequence, an optimized, parallel OBDD attack against a KSG of inner state size, e.g., 60 bits, could look as follows:

1. Determine a 10-bit area of the secret initial state that shall be pre-guessed, leading to 2^{10} parallel OBDD attack instances, each of them operating based on one of the corresponding 2^{10} possible guesses for that 10-bit area.

2. Execute one of these 2^{10} OBDD attack instances on a very potent system with a powerful CPU and fast memory. At the point where the corresponding OBDD reaches its maximum size, manually trigger variable reordering and save the corresponding variable order.[11]

3. Pre-set this variable order to the remaining $2^{10} - 1$ OBDD attack instances, which will benefit from this information as described above and, hence, can now even be executed on systems with less potent CPU cores and fewer memory, such as the aforementioned Intel Xeon Phi Coprocessor cards.

[10] That the memory requirement here was even lower than in the experiment where the actual reordering was performed, results from the fact that now, even the steps before the step during which the reordering in the initial experiment had taken place, profited from the new 'good' variable order.

[11] If this maximum size OBDD should be too large even for the potent system to handle, reordering can also be triggered already some steps before the maximum would have been reached and the results can still be expected to be sufficiently well as the above practical example for our 39-bit KSG has shown.

Our current explanation for the observation that, in practice, this strategy seems to work very well, is that despite the different pre-guessed parts, all of the respective parallel OBDD attack instances seem to still share enough 'common structure' for profiting from a common, 'pre-optimized' variable order. Keep in mind, however, that this is work in progress, which needs to be backed up by further experiments and much more theoretical study.

A New Approach for 'Truely' Parallelizing OBDD Attacks

As pointed out above, to the best of our knowledge, the only parallelization strategy suggested for BDD attacks so far is the trivial approach of splitting the problem up into completely independent instances by pre-guessing certain bits of the initial state (see, e.g., [KS06]). In the following, we suggest to study a new strategy, in which the involved CPU cores actually work *together* (based on the correct information solely coming from the observed keystream), instead of working completely separately (with all but one using wrong information due to the respective wrongly guessed parts of the initial state).

Our idea makes use of Theorem 3.3.6 in the book [Weg00] of Wegener, which implies that the AND-synthesis of two OBDDs G_f and G_g (with sizes $|G_f|$ and $|G_g|$) can be performed in "expected time and space $O(|G_h^*|)$", "where G_h^* is the graph consisting of the nodes in the product graph G_h reachable from the node representing h". Wegener continues to add that "[o]ften, $|G_h^*|$ is much smaller than $|G_f||G_g|$" and further points out that "[t]he synthesis algorithm can easily be generalized to the synthesis of m functions where we have to consider nodes of the product of the m corresponding OBDDs."

Based on these theoretical foundations, we came up with the following new strategy for OBDD-based cryptanalysis. Instead of building only one main OBDD P_t (see our example in Subsection 9.4.1) we actually generate now 'two main OBDDs' P_t^1 and P_t^2, where for every even step t, P_t^1 receives the information (comprising of the information about the state transition and the information from the output function with the respective observed keystream bit) of this step, and for every odd step t, P_t^2 is supplied with the respective information for that odd step. A more formal description of our new approach, using the terms of the introductory example in Subsection 9.4.1, is given as Algorithm 9.1.

Note that for $t \geq X$, the OBDD $\tilde{P}_t$ in Algorithm 9.1 will contain exactly the same information (in particular, have the same satisfying assignments) as if it had been built from the beginning as a single main OBDD P_t using the relations

$$\begin{aligned} P_0 &:= R_0 \wedge Q_0, \\ P_t &:= P_{t-1} \wedge R_t \wedge Q_t \quad \text{for } t \geq 1 \end{aligned}$$

of the classical approach explained in Subsection 9.4.1. Moreover, keep in mind that all OBDDs involved in Algorithm 9.1 are defined over the same set of variables and use a common variable order (cf. Footnote 8 on page 277).

Algorithm 9.1 A new approach for more efficient, parallelizable OBDD attacks.

$X \leftarrow$ *magic number* (see explanation below)
$P^1_{-1} \leftarrow$ 1-OBDD
$P^2_{-1} \leftarrow$ 1-OBDD
for $t = 0$ to X **do**
  if t is even **then**
    $P^1_t \leftarrow P^1_{t-1} \wedge R_t \wedge Q_t$
    $P^2_t \leftarrow P^2_{t-1}$
  else
    $P^1_t \leftarrow P^1_{t-1}$
    $P^2_t \leftarrow P^2_{t-1} \wedge R_t \wedge Q_t$
  end if
end for
$\tilde{P}_X \leftarrow P^1_X \wedge P^2_X$
$t \leftarrow X$
while $\tilde{P}_t$ has more than one satisfying assignments **do**
  $t \leftarrow t + 1$
  $\tilde{P}_t \leftarrow \tilde{P}_{t-1} \wedge R_t \wedge Q_t$
end while
  return $\tilde{P}_t$

The intuition behind the efficiency of our new approach is the following. Let $t^{\max}$ denote the step in which the single main OBDD P_t would have reached its maximum size $|P_{t^{\max}}|$ and remember that for $t > t^{\max}$, the sizes $|P_t|$ would henceforth constantly decrease until only one satisfying assignment is left. Then, for a properly chosen parameter X (in particular, $X > t^{\max}$), the result $\tilde{P}_X := P^1_X \wedge P^2_X$ in Algorithm 9.1 will already have much less nodes than $P_{t^{\max}}$, and, according to Theorem 3.3.6 of Wegener [Weg00], the respective synthesis operation will have advantageous (as compared to working with $P_{t^{\max}}$) expected time and memory consumption. However, for the new approach to be actually more efficient than classical one, X must be, on the other hand, still small enough such that the sizes of the OBDDs P^1_X and P^2_X still stay significantly below $|P_{t^{\max}}|$.

So, obviously, the choice of the parameter X is of crucial importance, which is why we called it the *magic number* at the beginning of Algorithm 9.1. Finding such a 'good' X is a challenging task, in particular, as it mainly depends on the value of $t^{\max}$, which we do not know in advance without first executing the attack in the classical, 'single main OBDD'-based way. Doing so, however, would trivially counteract the idea of being more efficient. This is why, so far, we have been experimenting with various heuristics to address this problem. As this, however, is still work in progress and as the description of the respective heuristics would certainly go beyond the scope of this *Future Research*

Directions chapter, we instead refer the reader to a potential later publication or, hopefully and even better, to his own ideas which we would be eager to hear about. The 'holy grail' in this context would obviously be a way of theoretically determining $t^{\max}$ in advance based on the details of the respective cryptanalytic problem. Note that the initial work on BDD-based attacks by Krause [Kra02] already contains corresponding considerations, but the resulting bounds are currently not tight enough to be actually used in a practical attack. Consequently, we strongly encourage future research in this direction.

To further motivate such research, note that our corresponding experimental results are currently extremely promising. More precisely, remember that our aforementioned, classical OBDD attack against the 39-bit KSG specified through Eqs. (9.7) and (9.8) had consumed almost 900 MB of memory and taken about 215 seconds. The single main OBDD used there reached its maximum size at $t^{\max} = 16$. In contrast, for an optimally chosen parameter X (here: $X = 21$), the same attack using our new approach in Algorithm 9.1 requires only about 110 MB of memory and is completed in as few as 8 seconds. Note that we did not apply any further 'tricks' (such as reordering) to achieve these vast improvements, i.e., they are exclusively based on the new algorithmic idea of working with 'two main OBDDs' P_t^1 and P_t^2, instead of performing the whole attack in the classical way with only one main OBDD P_t from start to end.

As a final remark, we would like to point out that we even did not physically parallelize the above new experimental attack. That is, the OBDDs P_t^1 and P_t^2 were generated by a single thread on a single CPU core. Observe, however, that as they are built on the basis of completely separate information, P_t^1 and P_t^2 could as well be easily computed in parallel on different CPUs in order to further speed up the attack. In fact, we are tempted to be bold enough to claim that, to the best of our knowledge, this new approach actually represents the first, non-trivial way of truly parallelizing BDD attacks. In this respect, also remember that, as pointed out by Wegener w.r.t. his Theorem 3.3.6 in [Weg00], "[t]he synthesis algorithm can easily be generalized to the synthesis of m functions where we have to consider nodes of the product of the m corresponding OBDDs." In consequence, a logical next step would be to extend our approach to using 'additional main OBDDs' $P_t^3, P_t^4, \ldots$ during the first phase of the attack, in order to further increase the speed and, at the same time, lower the overall memory consumption.

Further Improvements for OBDD Attacks

We want to conclude this (O)BDD-related subsection with two additional short suggestions of topics for potential future research in this area. Actually, both of them have also already been briefly mentioned in the paper [EPV08] of Eibach, Pilz, and Völkel from 2008, but unfortunately, we have not seen any corresponding academic progress, yet. This is probably due to a lack of respective research during the deep sleep that, as explained at the beginning of this subsection, BDD-based cryptanalysis seems to have taken since 2008. Nonetheless, we consider these two topics rather important, in particular, as they

are also connected to several of our suggestions above.

First, the order in which the information coming from the feedback and output relations (together with the observed keystream bits) is introduced to the OBDDs has never been analyzed systematically. Note that in this subsection, also for reasons of clarity and comprehensibility, we have simply used the classical order implied by the respective KSG's inner workings. However, using a more sophisticated approach (such as, per step, introducing feedback and output relations which share common variables) can safely be expected to produce better results w.r.t. OBDD sizes and overall computation time.

Second, w.r.t. different pre-guessing strategies, Eibach, Pilz, and Völkel concluded in 2008 (for their attack against the reduced Trivium [CP05] variant *Bivium*, consisting of two interconnected NFSRs):

> "It turned out that for SAT solvers and BDDs it is most useful to guess the internal bits close to the end of the 2 registers. The difference is that it is optimal for the SAT solvers to only guess the end of the second register and for the BDDs, it is optimal to share the guessed bits between the ends of both registers. Looking at the equation system describing Bivium, we notice that the variables close to the output occure rather frequent (those of the second register a little more than those of the first). This might be one reason, why these guessing strategies helped most." [EPV08]

Unfortunately, this is the only theoretical explanation w.r.t. the choice of pre-guessing strategies provided in [EPV08]. Given that, besides our new approach suggested above, the pre-guessing of internal state bits is currently the only other way to parallelize OBDD attacks, we believe that further research in this direction would prove very beneficial for future applications of (O)BDD-based cryptanalysis.

9.5 Conclusion and Outlook

In this chapter, we discussed three potential areas of future research in further detail and presented our corresponding ideas. First, we suggested how our new stream cipher Lizard (cf. Chapter 8) could be used (and optimized) to realize lightweight, privacy-preserving authentication for ultra-constrained RFID devices (cf. Chapter 2), hence representing a viable alternative to prevalent block cipher-based schemes. Second, we explained that by combining *continuous IV use* (cf. Section 6.5) and *packet mode* (see, e.g., Section 5.1 and Chapter 7), classical TMD-TO *inner state recovery* attacks like those of Babbage [Bab95] or Biryukov and Shamir [BS00] can actually be thwarted. Third, we investigated BDD- and SAT-based *short-keystream* cryptanalysis, and provided various suggestions how the efficiency of future BDD attacks could be improved, particularly, with regard to the increasingly important field of parallelization. As our most important contribution in this context, we see the new technique proposed with Algorithm 9.1, which, to the best of our knowledge, represents the first method for *truly* parallelizing BDD attacks.

As a final note, let us point out that the reason for discussing these three topics such prominently in the form of a separate chapter, is not solely that they are among the top priorities on our own research agenda. Instead, we hope that also other researchers will be interested in the presented ideas and, thus, we would like to encourage any kind of future collaboration on the respective projects.

In the following Chapter 10, we will conclude this thesis with a short summary of our results.

Sie sind geladen – es schlägt zwölfe! So sei es denn! – Lotte! Lotte, lebe wohl! Lebe wohl!

Werther (Goethe)

CHAPTER 10

Conclusion

After the AES block-cipher contest [Nat16] had finished in 2000/2001, much of the cryptographic community's attention shifted towards stream ciphers and, with the corresponding eSTREAM project's [ECR08] hardware-focused *Profile 2*, in particular also towards resource-constrained devices. The major catalyst of this development was the fact that around this time, in 2004, RFID technology was finally setting off to conquer the mass market, calling for more lightweight cryptographic implementations than, e.g., AES [DR02] was able to provide. Grain v1 [HJM06], the most hardware-efficient member of the final eSTREAM portfolio [BBV12], was already a first, important step in this direction, but it was soon caught up and, depending on the type of implementation, even surpassed in terms of low hardware requirements by the then new lightweight block ciphers PRESENT [BKL+07] and KATAN/KTANTAN [DCDK09]. Only few months after the eSTREAM contest had been launched in November 2004, and likewise motivated by the advent of cheap, mass market RFID devices, Juels and Weis presented their HB^+ authentication protocol [JW05], targeting so-called *Electronic Product Codes (EPCs)*, which are meant to replace classical barcodes on everyday goods. Though HB^+ was broken soon via active MITM attacks [GRS05], a multitude of related schemes (i.e., also grounding their security claims on the assumed hardness of the well-known *learning parity with noise (LPN) problem*) started to appear in quick succession, which, however, seemed to increasingly loose sight of the tight hardware bounds on low-cost RFID tags in the price range of \$0.05–\$0.10, as originally targeted by Juels and Weis in [JW05].

The situation described here thus brought up the following three questions:

1. When can a cryptographic scheme actually be called *lightweight*? In particular, which concrete limits does it have to satisfy in order to be suitable for ultra-constrained RFID devices?

2. Is it possible to design a (dedicated) lightweight authentication scheme for such ultra-constrained RFIDs as an alternative to prevalent block cipher-based constructions?

3. Can stream ciphers be made more competitive w.r.t. current lightweight block ciphers, or does Grain v1's 'birthday-based' inner state size of twice the key size imply a natural lower bound for the hardware costs of such designs?

In this thesis, we contributed to each of these three fields of interest. More precisely, in Chapter 2, we first laid the necessary hardware foundations by providing a comprehensive summary of relevant metrics and corresponding conditions that should be met by lightweight cryptographic schemes if deployed in low-cost RFID systems. Some of these conditions have been collected from open literature, but most of them are the result of various discussions with experts from industry.

In Chapter 3, we then gave an introduction to authentication in general and its lightweight forms in particular. An alarming result here was the insight that none of the currently unbroken LPN-based authentication protocols is actually suitable for ultra-constrained RFIDs. As a potential alternative, we identified authentication protocols

based on the *principle of random selection of secret linear functions* such as the $(n, k, L)^{++}$-protocol [KS09] suggested by Krause and Stegemann in 2009. However, a comparatively large key length and the use of involved operations had prevented these protocols from being deployed on resource-restricted devices, so far.

Motivated to solve these issues, in Chapter 4, we introduced our new $(n, k, L)^{\langle 80 \rangle}$ authentication protocol with key size 80 bits and, on the basis of hardware implementations for FPGAs and ASICs, demonstrated its suitability for ultra-constrained RFIDs.

In Chapter 5, we then left the path of searching for dedicated authentication protocols and, instead, laid the foundation for treating a fundamental question which arose to us while designing the $(n, k, L)^{\langle 80 \rangle}$-protocol: 'Why use a bitstream generator only to produce the specifications of the secret functions, but not for generating the authentication token right away?' To this end, as a first step, we revisited some prominent examples of classical stream ciphers and, in particular, analyzed their state initialization algorithms.

Subsequently, in Chapter 6, we turned towards the rather young research area of so-called *small-state stream ciphers*, which try to overcome the limit imposed by TMD-TO attacks (like those of Babbage [Bab95] or Biryukov and Shamir [BS00]) on the security of classical stream ciphers. The prevalent design principle here, introduced by Armknecht and Mikhalev in 2015 with their stream cipher Sprout [AM15], is to continuously involve the secret key in the state update. However, by providing a generic distinguisher with complexity significantly lower than that of exhaustive key search for such ciphers, we showed in this chapter that the initial hope of achieving full security against TMD-TO attacks raised by this new design approach has failed. But by studying the assumptions underlying the applicability of our attack, we were then able to come up with a new design idea for small-state stream ciphers, which might allow to finally thwart TMD-TO *key recovery* and *distinguishing* attacks by continuously using not only the key but also the IV during keystream generation. Another contribution of this chapter was the first key recovery attack against Fruit v1 [GHX16], indicating that a simple *round key function* is actually preferable for Sprout-like ciphers.

Paving the way for our own small-state stream cipher, in Chapter 7, we then proposed and analyzed the Lizard-construction, which combines the explicit use of *packet mode* with a new type of state initialization algorithm. For corresponding KSG-based designs of inner state length n, we proved a tight $(2n/3)$-bound on the security against TMD-TO key recovery attacks, while the security against TMD-TO distinguishing attacks remains at the birthday-bound level $n/2$. The lower bound of the $(2n/3)$-result refers to a random oracle model which allows to derive formal security statements w.r.t. generic TMD-TO attacks. While similar frameworks were already used for analyzing the security of block cipher, MAC, and hash function constructions, to the best of our knowledge this is the first time that such a model has been considered in a stream cipher context.

Building on these theoretical results, in Chapter 8, we finally presented Lizard, our new lightweight (small-state) stream cipher for power-constrained devices like passive RFID tags. Lizard uses 120-bit keys, 64-bit IVs, has an inner state length of 121 bits,

and its hardware efficiency and security result from combining a Grain-like design with the Lizard-construction. It is supposed to provide 80-bit protection against key recovery attacks and allows to generate up to 2^{18} keystream bits per key/IV pair, which would be sufficient for many existing communication scenarios such as Bluetooth, WLAN, or HTTPS. Most notably, besides lower area requirements, the estimated power consumption of Lizard is also about 16 percent below that of Grain v1, making it particularly suitable for passive RFID tags.

While in the respective conclusion sections of Chapters 2 to 8, we had already provided numerous suggestions for potential future work, with Chapter 9, we decided to discuss three particularly promising topics in further detail. First, we sketched how our new lightweight stream cipher Lizard could be used (and further optimized) to realize hardware-efficient, privacy-preserving authentication, thereby connecting our two main topics: *lightweight authentication* and *lightweight stream ciphers*. Second, we suggested how to combine our previous ideas of explicitly targeting *packet mode* scenarios and *continuously using the IV* as part of the state update, in order to now thwart classical TMD-TO *inner state recovery* attacks. Third, motivated by our application context of ultra-constrained RFID devices with their corresponding bandwidth limitations, we found that the field of *short-keystream attacks* should see more attention in future research. In particular, we reasoned that the field of BDD-based cryptanalysis has been neglected for too long and, in order to 'reignite the flame', provided several new ideas how the efficiency and applicability of such attacks could be improved.

Not only the large extent of our *Future Research Directions* chapter, but also the ongoing stream of corresponding publications (see, e.g., [BBI+15], [BJK+16], [SS16], [MAM17], [Liu17], [JPST17], [ZXM18]) clearly shows that the search for lightweight cryptographic solutions is far from being completed. Less than one month before the finalization of this thesis, Amazon opened its first, automated grocery store *Amazon Go* in Seattle [Win18], where, without any checkout lines or other interaction, customers can simply leave the shop with their selected goods, being billed electronically over their Amazon account. While, at first sight, this might seem like the perfect concluding example in this thesis for the apparently inevitable triumph of RFID technology and corresponding lightweight cryptographic solutions, unfortunately, it is not. For instead of attaching an RFID tag to each product, Amazon decided to equip the respective store with a vast amount of cameras and to employ corresponding object-recognition techniques for tracking items and customers. The reason for this choice is very simple. In the concerned product segment of milk boxes and the like, even low-cost RFID tags in the price range of \$0.05–\$0.10 are currently still too expensive, given the low profit margins which these goods offer to grocery retailers. In consequence, for staying competitive, even more restricted RFID devices are to be expected, putting further pressure on the resource demands of corresponding cryptographic solutions. However, we are absolutely confident that the research community will be, once more, ready to take up this challenge.

Bibliography

[3GP03] 3GPP: 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; 3G Security. Specification of the A5/3 Encryption Algorithms for GSM and ECSD, and the GEA3 Encryption Algorithm for GPRS; Document 1: A5/3 and GEA3 Specifications (Release 6). 3GPP TS 55.216 V6.2.0 (2003-09), 2003. `https://www.gsma.com/aboutus/wp-content/uploads/2014/12/a53andgea3specifications.pdf`.

[3GP17] 3GPP: 3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; 3G Security. Specification of the 3GPP confidentiality and integrity algorithms; Document 2: KASUMI specification (Release 14). 3GPP TS 35.202 V14.0.0 (2017-03), 2017. `http://www.3gpp.org/ftp/Specs/archive/35_series/35.202/35202-e00.zip`.

[ABD+13] Elena Andreeva, Andrey Bogdanov, Yevgeniy Dodis, Bart Mennink, and John P. Steinberger. On the Indifferentiability of Key-Alternating Ciphers. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology - CRYPTO 2013 - 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, volume 8042 of *Lecture Notes in Computer Science*, pages 531–550. Springer, 2013.

[AHK14] Frederik Armknecht, Matthias Hamann, and Matthias Krause. Hardware Efficient Authentication based on Random Selection. In Stefan Katzenbeisser, Volkmar Lotz, and Edgar R. Weippl, editors, *Sicherheit 2014: Sicherheit, Schutz und Zuverlässigkeit, Beiträge der 7. Jahrestagung des Fachbereichs Sicherheit der Gesellschaft für Informatik e.V. (GI), 19.-21. März 2014, Wien, Österreich*, volume 228 of *LNI*, pages 169–185. GI, 2014.

[AHM14] Frederik Armknecht, Matthias Hamann, and Vasily Mikhalev. Lightweight Authentication Protocols on Ultra-Constrained RFIDs - Myths and Facts. In Nitesh Saxena and Ahmad-Reza Sadeghi, editors, *Radio Frequency Identification: Security and Privacy Issues: 10th International Workshop, RFIDSec 2014, Oxford, UK, July 21-23, 2014, Revised Selected Papers*, pages 1–18. Springer International Publishing, Cham, 2014.

[AK03] Frederik Armknecht and Matthias Krause. Algebraic Attacks on Combiners with Memory. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003: 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003. Proceedings*, pages 162–175. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[AM15] Frederik Armknecht and Vasily Mikhalev. On Lightweight Stream Ciphers with Shorter Internal States. In Gregor Leander, editor, *Fast Software Encryption: 22nd International Workshop, FSE 2015, Istanbul, Turkey, March 8-11, 2015, Revised Selected Papers*, pages 451–470. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.

[ASE92] Noga Alon, Joel H. Spencer, and Paul Erdős. *The Probabilistic Method*. Wiley Interscience, New York, 1992.

[Bab95] Steve H. Babbage. Improved "exhaustive search" attacks on stream ciphers. In *Security and Detection, 1995., European Convention on*, pages 161–166, May 1995.

[BAL06] Thierry Berger, François Arnault, and Cédric Lauradoux. Update on F-FCSR Stream Cipher. eSTREAM: the ECRYPT Stream Cipher Project, 2006. `http://www.ecrypt.eu.org/stream/p3ciphers/ffcsr/ffcsr_p3.pdf`.

[Ban15] Subhadeep Banik. Some Results on Sprout. In Alex Biryukov and Vipul Goyal, editors, *Progress in Cryptology – INDOCRYPT 2015: 16th International Conference on Cryptology in India, Bangalore, India, December 6-9, 2015, Proceedings*, pages 124–139. Springer International Publishing, Cham, 2015.

[BB06] Elad Barkan and Eli Biham. Conditional Estimators: An Effective Attack on A5/1. In Bart Preneel and Stafford Tavares, editors, *Selected Areas in Cryptography: 12th International Workshop, SAC 2005, Kingston, ON, Canada, August 11-12, 2005, Revised Selected Papers*, pages 1–19. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[BB08] Ganesh K. Balachandran and Raymond E. Barnett. A 440-nA True Random Number Generator for Passive RFID Tags. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 55(11):3723–3732, Dec 2008.

[BBI+15] Subhadeep Banik, Andrey Bogdanov, Takanori Isobe, Kyoji Shibutani, Harunaga Hiwatari, Toru Akishita, and Francesco Regazzoni. Midori: A Block Cipher for Low Energy. In Tetsu Iwata and Jung Hee Cheon,

editors, *Advances in Cryptology – ASIACRYPT 2015*, pages 411–436, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[BBK03] Elad Barkan, Eli Biham, and Nathan Keller. Instant Ciphertext-Only Cryptanalysis of GSM Encrypted Communication. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 600–616. Springer Berlin Heidelberg, 2003.

[BBV12] Steve Babbage, Julia Borghoff, and Vesselin Velichkov. D.SYM.10 - The eSTREAM Portfolio in 2012. eSTREAM: the ECRYPT Stream Cipher Project, 2012. `http://www.ecrypt.eu.org/ecrypt2/documents/D.SYM.10-v1.pdf`.

[BC08] Julien Bringer and Hervé Chabanne. Trusted-HB: A low cost version of HB^+ secure against a man-in-the-middle attack. *IEEE Trans. Inform. Theor.*, 54:4339–4342, 2008.

[BCG+12] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçın. PRINCE – A Low-Latency Block Cipher for Pervasive Computing Applications. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology – ASIACRYPT 2012: 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6, 2012. Proceedings*, pages 208–225. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[BCI08] Julien Bringer, Hervé Chabanne, and Thomas Icart. Cryptanalysis of EC-RAC, a RFID Identification Protocol. In Matthew K. Franklin, Lucas Chi Kwong Hui, and Duncan S. Wong, editors, *Cryptology and Network Security: 7th International Conference, CANS 2008, Hong-Kong, China, December 2-4, 2008. Proceedings*, pages 149–161. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[BCJ07] Gregory V. Bard, Nicolas T. Courtois, and Chris Jefferson. Efficient Methods for Conversion and Solution of Sparse Systems of Low-Degree Multivariate Polynomials over GF(2) via SAT-Solvers. Cryptology ePrint Archive, Report 2007/024, 2007. `https://eprint.iacr.org/2007/024`.

[BCP97] Wieb Bosma, John Cannon, and Catherine Playoust. The MAGMA Algebra System I: The User Language. *J. Symb. Comput.*, 24(3-4):235–265, October 1997.

[BD00] Eli Biham and Orr Dunkelman. Cryptanalysis of the A5/1 GSM Stream Cipher. In Bimal Roy and Eiji Okamoto, editors, *Progress in Cryptology — INDOCRYPT 2000*, volume 1977 of *Lecture Notes in Computer Science*, pages 43–51. Springer Berlin Heidelberg, 2000.

[BD06] Steve Babbage and Matthew Dodd. The stream cipher MICKEY 2.0. eSTREAM: the ECRYPT Stream Cipher Project, 2006. `http://www.ecrypt.eu.org/stream/p3ciphers/mickey/mickey_p3.pdf`.

[BDK05] Eli Biham, Orr Dunkelman, and Nathan Keller. A Related-Key Rectangle Attack on the Full KASUMI. In Bimal Roy, editor, *Advances in Cryptology - ASIACRYPT 2005: 11th International Conference on the Theory and Application of Cryptology and Information Security, Chennai, India, December 4-8, 2005. Proceedings*, pages 443–461. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[BG07] Côme Berbain and Henri Gilbert. On the Security of IV Dependent Stream Ciphers. In Alex Biryukov, editor, *Fast Software Encryption: 14th International Workshop, FSE 2007, Luxembourg, Luxembourg, March 26-28, 2007, Revised Selected Papers*, pages 254–273. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[BGJ09] Côme Berbain, Henri Gilbert, and Antoine Joux. Algebraic and Correlation Attacks against Linearly Filtered Non Linear Feedback Shift Registers. In Roberto Maria Avanzi, Liam Keliher, and Francesco Sica, editors, *Selected Areas in Cryptography: 15th International Workshop, SAC 2008, Sackville, New Brunswick, Canada, August 14-15, Revised Selected Papers*, pages 184–198. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[BGM06] Côme Berbain, Henri Gilbert, and Alexander Maximov. Cryptanalysis of Grain. In Matthew Robshaw, editor, *Fast Software Encryption: 13th International Workshop, FSE 2006, Graz, Austria, March 15-17, 2006, Revised Selected Papers*, pages 15–29. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006.

[BGW99] Marc Briceno, Ian Goldberg, and David Wagner. A pedagogical implementation of A5/1, 1999. Available at `http://www.scard.org/gsm/a51.html`.

[BI17] Subhadeep Banik and Takanori Isobe. Some cryptanalytic results on Lizard. Cryptology ePrint Archive, Report 2017/346, 2017. `https://eprint.iacr.org/2017/346`.

[BICG17] Subhadeep Banik, Takanori Isobe, Tingting Cui, and Jian Guo. Some cryptanalytic results on Lizard. *IACR Transactions on Symmetric Cryptology*, 2017(4):82–98, 2017.

[Bir05] Alex Biryukov. LEX. eSTREAM: the ECRYPT Stream Cipher Project, 2005. `hhttp://www.ecrypt.eu.org/stream/lexp3.html`.

[BJK+16] Christof Beierle, Jérémy Jean, Stefan Kölbl, Gregor Leander, Amir Moradi, Thomas Peyrin, Yu Sasaki, Pascal Sasdrich, and Siang Meng Sim. The SKINNY Family of Block Ciphers and Its Low-Latency Variant MANTIS. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016: 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part II*, pages 123–153. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.

[Bjø08] Tor E. Bjørstad. Cryptanalysis of Grain using Time/Memory/Date Tradeoffs. eSTREAM, ECRYPT Stream Cipher Project, Report 2008/012, 2008. `http://www.ecrypt.eu.org/stream/papersdir/2008/012.pdf`.

[BKL+06] Lejla Batina, Sandeep Kumar, Joseph Lano, Nele Lemke, Kirstin Mentens, Christof Paar, Bart Preneel, Kazuo Sakiyama, and Ingrid Verbauwhede. Testing Framework for eSTREAM Profile II Candidates. eSTREAM, ECRYPT Stream Cipher Project, Report 2006/014, 2006. `http://www.ecrypt.eu.org/stream/papersdir/2006/014.pdf`.

[BKL+07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and Christine Vikkelsoe. PRESENT: An Ultra-Lightweight Block Cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007: 9th International Workshop, Vienna, Austria, September 10-13, 2007. Proceedings*, pages 450–466. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[BKL+12] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Francois-Xavier Standaert, John Steinberger, and Elmar Tischhauser. Key-Alternating Ciphers in a Provable Setting: Encryption Using a Small Number of Public Permutations. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 45–62. Springer Berlin Heidelberg, 2012.

[BKM+09] Erik-Oliver Blass, Anil Kurmus, Refik Molva, Guevara Noubir, and Abdullatif Shikfa. The F_f-Family of Protocols for RFID-Privacy and Authentication. In *5th Workshop on RFID Security, RFIDSec'09*, 2009.

[BKSQ07] Philippe Bulens, Kassem Kalach, Francois-Xavier Standaert, and Jean-Jacques Quisquater. FPGA Implementations of eSTREAM Phase-2 Focus Candidates with Hardware Profile. eSTREAM, ECRYPT Stream Cipher Project, Report 2007/024, 2007. `http://www.ecrypt.eu.org/stream/papersdir/2007/024.pdf`.

[BL13] Daniel J. Bernstein and Tanja Lange. Never Trust a Bunny. In *Proceedings of the 8th International Conference on Radio Frequency Identification: Security and Privacy Issues*, RFIDSec'12, pages 137–148, Berlin, Heidelberg, 2013. Springer-Verlag.

[Blu14] Bluetooth SIG. Bluetooth Core Specification 4.2, 2014. `https://www.bluetooth.org/DocMan/handlers/DownloadDoc.ashx?doc_id=286439`.

[BLW95] Beate Bollig, Martin Löbbing, and Ingo Wegener. Simulated annealing to improve variable orderings for OBDDs. In *International Workshop on Logic Synthesis, Granlibakken, CA, May 1995*, pages 5–5, 1995.

[BMvT78] Elwyn Berlekamp, Robert McEliece, and Henk van Tilborg. On the inherent intractability of certain coding problems (Corresp.). *IEEE Transactions on Information Theory*, 24(3):384–386, May 1978.

[BP17] Alex Biryukov and Leo Perrin. State of the Art in Lightweight Symmetric Cryptography. Cryptology ePrint Archive, Report 2017/511, 2017. `http://eprint.iacr.org/2017/511`.

[Bru04] Billy Brumley. A3/A8 & COMP128. Helsinki University of Technology, T-79.514 Special Course on Cryptology, 2004. `http://www.tcs.hut.fi/Studies/T-79.514/slides/S5.Brumley-comp128.pdf`.

[BS00] Alex Biryukov and Adi Shamir. Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers. In Tatsuaki Okamoto, editor, *Advances in Cryptology — ASIACRYPT 2000: 6th International Conference on the Theory and Application of Cryptology and Information Security Kyoto, Japan, December 3–7, 2000 Proceedings*, pages 1–13. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000.

[BSS+13] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. The SIMON and SPECK Families of

Lightweight Block Ciphers. Cryptology ePrint Archive, Report 2013/404, 2013. http://eprint.iacr.org/2013/404.

[BSS+15] Ray Beaulieu, Douglas Shors, Jason Smith, Stefan Treatman-Clark, Bryan Weeks, and Louis Wingers. SIMON and SPECK: Block Ciphers for the Internet of Things. Cryptology ePrint Archive, Report 2015/585, 2015. http://eprint.iacr.org/2015/585.

[BSW01] Alex Biryukov, Adi Shamir, and David Wagner. Real Time Cryptanalysis of A5/1 on a PC. In Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, and Bruce Schneier, editors, *Fast Software Encryption: 7th International Workshop, FSE 2000 New York, NY, USA, April 10–12, 2000 Proceedings*, pages 1–18. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[Cad17] Cadence Design Systems, Inc. Encounter RTL Compiler. Website (accessed on October 17, 2017), 2017. http://www.cadence.com/.

[Car12] Claude Carlet. Boolean Functions for Cryptography and Error Correcting Codes. LAGA, University of Paris 8, France, 2012. http://www.math.univ-paris13.fr/~carlet/chap-fcts-Bool-corr.pdf.

[Cis04] Cisco Systems. Cisco Security Notice: Cisco IPsec VPN Implementation Group Password Usage Vulnerability. archive.org – WayBack-Machine, Version: May 02, 2004 (accessed on September 6, 2017), 2004. https://web.archive.org/web/20040502183734/http://www.cisco.com/warp/public/707/cisco-sn-20040415-grppass.shtml.

[CKK08] Jacek Cichoń, Marek Klonowski, and Mirosław Kutyłowski. Privacy Protection for RFID with Hidden Subset Identifiers. In Jadwiga Indulska, Donald J. Patterson, Tom Rodden, and Max Ott, editors, *Pervasive Computing: 6th International Conference, Pervasive 2008 Sydney, Australia, May 19-22, 2008 Proceedings*, pages 298–314. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[CKPS00] Nicolas Courtois, Alexander Klimov, Jacques Patarin, and Adi Shamir. Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations. In Bart Preneel, editor, *Advances in Cryptology — EUROCRYPT 2000: International Conference on the Theory and Application of Cryptographic Techniques Bruges, Belgium, May 14–18, 2000 Proceedings*, pages 392–407. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000.

[CLL+14] Shan Chen, Rodolphe Lampe, Jooyoung Lee, Yannick Seurin, and John Steinberger. Minimizing the Two-Round Even-Mansour Cipher. In

Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014*, volume 8616 of *Lecture Notes in Computer Science*, pages 39–56. Springer Berlin Heidelberg, 2014.

[CM03] Nicolas T. Courtois and Willi Meier. Algebraic Attacks on Stream Ciphers with Linear Feedback. In Eli Biham, editor, *Advances in Cryptology — EUROCRYPT 2003: International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4–8, 2003 Proceedings*, pages 345–359. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[CN17] Thomas De Cnudde and Svetla Nikova. Securing the PRESENT Block Cipher Against Combined Side-Channel Analysis and Fault Attacks. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(12):3291–3301, Dec 2017.

[CNO08] Nicolas T. Courtois, Karsten Nohl, and Sean O'Neil. Algebraic Attacks on the Crypto-1 Stream Cipher in MiFare Classic and Oyster Cards. Cryptology ePrint Archive, Report 2008/166, 2008. `http://eprint.iacr.org/2008/166`.

[Cou03] Nicolas T. Courtois. Fast Algebraic Attacks on Stream Ciphers with Linear Feedback. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003: 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003. Proceedings*, pages 176–194. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[CP05] Christophe De Cannière and Bart Preneel. Trivium – Specifications. eSTREAM: the ECRYPT Stream Cipher Project, 2005. `http://www.ecrypt.eu.org/stream/p3ciphers/trivium/trivium_p3.pdf`.

[CR08] Peter H. Cole and Damith C. Ranasinghe. *Networked RFID Systems and Lightweight Cryptography: Raising Barriers to Product Counterfeiting*. Springer Berlin Heidelberg, first edition, 2008.

[CS14] Shan Chen and John Steinberger. Tight Security Bounds for Key-Alternating Ciphers. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 327–350. Springer Berlin Heidelberg, 2014.

[DA14] Prakash Dey and Avishek Adhikari. Improved Multi-Bit Differential Fault Analysis of Trivium. In Willi Meier and Debdeep Mukhopadhyay,

editors, *Progress in Cryptology – INDOCRYPT 2014*, pages 37–52, Cham, 2014. Springer International Publishing.

[DCDK09] Christophe De Cannière, Orr Dunkelman, and Miroslav Knežević. KATAN and KTANTAN — A Family of Small and Efficient Hardware-Oriented Block Ciphers. In Christophe Clavier and Kris Gaj, editors, *Cryptographic Hardware and Embedded Systems - CHES 2009: 11th International Workshop Lausanne, Switzerland, September 6-9, 2009 Proceedings*, pages 272–288. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

[DCKP08] Christophe De Cannière, Özgül Küçük, and Bart Preneel. Analysis of Grain's Initialization Algorithm. In Serge Vaudenay, editor, *Progress in Cryptology – AFRICACRYPT 2008: First International Conference on Cryptology in Africa, Casablanca, Morocco, June 11-14, 2008. Proceedings*, pages 276–289. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[DH15] Elena Dubrova and Martin Hell. Espresso: A stream cipher for 5G wireless communication systems. *Cryptography and Communications*, pages 1–17, 2015.

[DHW+12] Benedikt Driessen, Ralf Hund, Carsten Willems, Christof Paar, and Thorsten Holz. Don't Trust Satellite Phones: A Security Analysis of Two Satphone Standards. In *2012 IEEE Symposium on Security and Privacy*, pages 128–142, May 2012.

[DK08] Orr Dunkelman and Nathan Keller. Treatment of the Initial Value in Time-Memory-Data Tradeoff Attacks on Stream Ciphers. Cryptology ePrint Archive, Report 2008/311, 2008. `http://eprint.iacr.org/2008/311`.

[dKGHG08] Gerhard de Koning Gans, Jaap-Henk Hoepman, and Flavio D. Garcia. A Practical Attack on the MIFARE Classic. In Gilles Grimaud and François-Xavier Standaert, editors, *Smart Card Research and Advanced Applications: 8th IFIP WG 8.8/11.2 International Conference, CARDIS 2008, London, UK, September 8-11, 2008. Proceedings*, pages 267–282. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[DKS10] Orr Dunkelman, Nathan Keller, and Adi Shamir. A Practical-Time Attack on the A5/3 Cryptosystem Used in Third Generation GSM Telephony. Cryptology ePrint Archive, Report 2010/013, 2010. `http://eprint.iacr.org/2010/013`.

[DKS12] Orr Dunkelman, Nathan Keller, and Adi Shamir. Minimalism in Cryptography: The Even-mansour Scheme Revisited. In *Proceedings of the 31st Annual International Conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'12, pages 336–354, Berlin, Heidelberg, 2012. Springer-Verlag.

[DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.

[DR08] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008. Updated by RFCs 5746, 5878, 6176, 7465, 7507, 7568, 7627, 7685, 7905, 7919.

[DS09] Itai Dinur and Adi Shamir. Cube Attacks on Tweakable Black Box Polynomials. In Antoine Joux, editor, *Advances in Cryptology - EUROCRYPT 2009: 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*, pages 278–299, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[DS11] Itai Dinur and Adi Shamir. Breaking Grain-128 with Dynamic Cube Attacks. In Antoine Joux, editor, *Fast Software Encryption: 18th International Workshop, FSE 2011, Lyngby, Denmark, February 13-16, 2011, Revised Selected Papers*, pages 167–187. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[DS17] Sabyasachi Dey and Santanu Sarkar. Cryptanalysis of full round Fruit. Cryptology ePrint Archive, Report 2017/87, 2017. `http://eprint.iacr.org/2017/87.pdf`.

[Dub12] Elena Dubrova. A List of Maximum Period NLFSRs. Cryptology ePrint Archive, Report 2012/166, 2012. `http://eprint.iacr.org/2012/166`.

[Dub13] Elena Dubrova. A Scalable Method for Constructing Galois NLFSRs With Period $2^n - 1$ Using Cross-Join Pairs. *IEEE Transactions on Information Theory*, 59(1):703–709, January 2013.

[ECR05] ECRYPT - European Network of Excellence for Cryptology. eSTREAM Optimized Code HOWTO, 2005. `http://www.ecrypt.eu.org/stream/perf/`.

[ECR08] ECRYPT – European Network of Excellence for Cryptology. eSTREAM: the ECRYPT stream cipher project, 2008. `http://www.ecrypt.eu.org/stream/`.

[EHJ07] Håkan Englund, Martin Hell, and Thomas Johansson. A Note on Distinguishing Attacks. In *2007 IEEE Information Theory Workshop on Information Theory for Wireless Networks*, pages 1–4, July 2007.

[EK16] Muhammed F. Esgin and Orhun Kara. Practical Cryptanalysis of Full Sprout with TMD Tradeoff Attacks. In Orr Dunkelman and Liam Keliher, editors, *Selected Areas in Cryptography - SAC 2015: 22nd International Conference, Sackville, NB, Canada, August 12-14, 2015, Revised Selected Papers*, pages 67–85. Springer International Publishing, Cham, 2016.

[EM93] Shimon Even and Yishay Mansour. A construction of a cipher from a single pseudorandom permutation. In Hideki Imai, RonaldL. Rivest, and Tsutomu Matsumoto, editors, *Advances in Cryptology — ASIACRYPT '91*, volume 739 of *Lecture Notes in Computer Science*, pages 210–224. Springer Berlin Heidelberg, 1993.

[EPV08] Tobias Eibach, Enrico Pilz, and Gunnar Völkel. Attacking Bivium Using SAT Solvers. In Hans Kleine Büning and Xishun Zhao, editors, *Theory and Applications of Satisfiability Testing – SAT 2008*, pages 63–76, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg.

[ES10] Niklas Eén and Niklas Sörensson. MiniSat 2.2. Website (accessed on February 06, 2018), 2010. `http://minisat.se/`.

[FDW04] Martin Feldhofer, Sandra Dominikus, and Johannes Wolkerstorfer. Strong Authentication for RFID Systems Using the AES Algorithm. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 357–370. Springer Berlin Heidelberg, 2004.

[Fel07] Martin Feldhofer. Comparison of Low-Power Implementations of Trivium and Grain. eSTREAM, ECRYPT Stream Cipher Project, Report 2007/027, 2007. `http://www.ecrypt.eu.org/stream/papersdir/2007/027.pdf`.

[FMK91] Masahiro Fujita, Yusuke Matsunaga, and Taeko Kakuda. On variable ordering of binary decision diagrams for the application of multi-level logic synthesis. In *Proceedings of the European Conference on Design Automation.*, pages 50–54, Feb 1991.

[FMS01] Scott Fluhrer, Itsik Mantin, and Adi Shamir. Weaknesses in the Key Scheduling Algorithm of RC4. In Serge Vaudenay and Amr M. Youssef,

editors, *Selected Areas in Cryptography: 8th Annual International Workshop, SAC 2001 Toronto, Ontario, Canada, August 16–17, 2001 Revised Papers*, pages 1–24. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[FS09] Dmitry Frumkin and Adi Shamir. Untrusted-HB: Security Vulnerabilities of Trusted-HB. Cryptology ePrint Archive, Report 2009/044, 2009. `http://eprint.iacr.org`.

[FWR05] Martin Feldhofer, Johannes Wolkerstorfer, and Vincent Rijmen. AES implementation on a grain of sand. *Information Security, IEE Proceedings*, 152(1):13–20, Oct 2005.

[GB08] Tim Good and Mohammed Benaissa. Hardware performance of eStream phase-III stream cipher candidates. eSTREAM: the ECRYPT Stream Cipher Project, 2008. `http://www.ecrypt.eu.org/stream/docs/hardware.pdf`.

[GCB06] Tim Good, William Chelton, and Mohamed Benaissa. Review of stream cipher candidates from a low resource hardware perspective. eSTREAM, ECRYPT Stream Cipher Project, Report 2006/016, 2006. `http://www.ecrypt.eu.org/stream/papersdir/2006/016.pdf`.

[GdKGM+08] Flavio D. Garcia, Gerhard de Koning Gans, Ruben Muijrers, Peter van Rossum, Roel Verdult, Ronny Wichers Schreur, and Bart Jacobs. Dismantling MIFARE Classic. In Sushil Jajodia and Javier Lopez, editors, *Computer Security - ESORICS 2008: 13th European Symposium on Research in Computer Security, Málaga, Spain, October 6-8, 2008. Proceedings*, pages 97–114. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[GGK06] Berndt Gammel, Rainer Göttfert, and Oliver Kniffler. ACHTERBAHN-128/80. eSTREAM: the ECRYPT Stream Cipher Project, 2006. `http://www.ecrypt.eu.org/stream/p2ciphers/achterbahn/achterbahn_p2.pdf`.

[gHJM11] Martin Ågren, Martin Hell, Thomas Johansson, and Willi Meier. Grain-128a: A New Version of Grain-128 with Optional Authentication. *International Journal of Wireless and Mobile Computing*, 5(1):48–59, December 2011.

[GHX16] Vahid Amin Ghafari, Honggang Hu, and Chengxin Xie. Fruit: Ultra-Lightweight Stream Cipher with Shorter Internal State. Cryptology ePrint Archive, Report 2016/355, 2016. `http://eprint.iacr.org/2016/355`.

[GKNP08] Tim Güneysu, Timo Kasper, Martin Novotny, and Christof Paar. Cryptanalysis with COPACOBANA. *IEEE TRANSACTIONS ON COMPUTERS*, 57(11):1498–1513, 2008.

[GLS14] Lubos Gaspar, Gaëtan Leurent, and François-Xavier Standaert. Hardware Implementation and Side-Channel Analysis of Lapin. In Josh Benaloh, editor, *Topics in Cryptology – CT-RSA 2014: The Cryptographer's Track at the RSA Conference 2014, San Francisco, CA, USA, February 25-28, 2014. Proceedings*, pages 206–226. Springer International Publishing, Cham, 2014.

[Gol96] Jovan Dj. Golić. On the security of nonlinear filter generators. In Dieter Gollmann, editor, *Fast Software Encryption: Third International Workshop Cambridge, UK, February 21–23 1996 Proceedings*, pages 173–188. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996.

[GRS05] Henri Gilbert, Matt Robshaw, and Hervé Sibert. An Active Attack Against HB^+ - A Provably Secure Lightweight Authentication Protocol. Cryptology ePrint Archive, Report 2005/237, 2005. `http://eprint.iacr.org/2005/237`.

[GRS08] Henri Gilbert, Matthew J. B. Robshaw, and Yannick Seurin. $HB^\#$: Increasing the Security and Efficiency of HB^+. In Nigel Smart, editor, *Advances in Cryptology – EUROCRYPT 2008: 27th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, April 13-17, 2008. Proceedings*, pages 361–378. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[GRVS09] Flavio D. Garcia, Peter van Rossum, Roel Verdult, and Ronny Wichers Schreur. Wirelessly Pickpocketing a Mifare Classic Card. In *Proceedings of the 2009 30th IEEE Symposium on Security and Privacy*, SP '09, pages 3–15, Washington, DC, USA, 2009. IEEE Computer Society.

[GT15] Peter Gazi and Stefano Tessaro. Secret-key cryptography from ideal primitives: A systematic overview. In *Information Theory Workshop (ITW), 2015 IEEE*, pages 1–5, April 2015.

[Ham10] Matthias Hamann. On the Complexity of a Learning Problem Induced by a Lightweight Cryptographic Construction. Diploma thesis, University of Mannheim, Germany, 2010.

[Ham15] Matt Hamblen. Samsung expects 70M Galaxy S6 and Edge phones to be sold. Website (accessed on September 19, 2017), 2015. `https://www.computerworld.com/article/2912156/`

samsung-expects-70m-galaxy-s6-and-edge-phones-to-be-sold.html.

[Hao15] Yonglin Hao. A Related-key Chosen-IV Distinguishing Attack on Full Sprout Stream Cipher. Cryptology ePrint Archive, Report 2015/231, 2015. http://eprint.iacr.org/2015/231.pdf.

[HB01] Nicholas J. Hopper and Manuel Blum. Secure Human Identification Protocols. In Colin Boyd, editor, *Advances in Cryptology — ASIACRYPT 2001: 7th International Conference on the Theory and Application of Cryptology and Information Security Gold Coast, Australia, December 9–13, 2001 Proceedings*, pages 52–66. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[Hel80] Martin Hellman. A cryptanalytic time-memory trade-off. *IEEE Transactions on Information Theory*, 26(4):401–406, Jul 1980.

[HG11] Honggang Hu and Guang Gong. Periods on Two Kinds of Nonlinear Feedback Shift Registers with Time Varying Feedback Functions. Technical report, University of Waterloo, Canada, 2011. http://cacr.uwaterloo.ca/techreports/2011/cacr2011-08.pdf.

[His14] Rebecca Hiscott. RFID Tags Track Marijuana From Seed to Sale, in Colorado. Website of Mashable Inc. (accessed on October 01, 2017), 2014. http://mashable.com/2014/02/11/marijuana-rfid-tracking/.

[HJ08] Martin Hell and Thomas Johansson. Breaking the F-FCSR-H Stream Cipher in Real Time. In Josef Pieprzyk, editor, *Advances in Cryptology - ASIACRYPT 2008: 14th International Conference on the Theory and Application of Cryptology and Information Security, Melbourne, Australia, December 7-11, 2008. Proceedings*, pages 557–569. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[HJM06] Martin Hell, Thomas Johansson, and Willi Meier. Grain - A Stream Cipher for Constrained Environments. eSTREAM: the ECRYPT Stream Cipher Project, 2006. http://www.ecrypt.eu.org/stream/p3ciphers/grain/Grain_p3.pdf.

[HJMM06] Martin Hell, Thomas Johansson, Alexander Maximov, and Willi Meier. A Stream Cipher Proposal: Grain-128. In *2006 IEEE International Symposium on Information Theory*, pages 1614–1618, July 2006.

[HJMM08] Martin Hell, Thomas Johansson, Alexander Maximov, and Willi Meier. The Grain Family of Stream Ciphers. In Matthew Robshaw and Olivier

Billet, editors, *New Stream Cipher Designs: The eSTREAM Finalists*, pages 179–190. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[HK15] Matthias Hamann and Matthias Krause. Stream Cipher Operation Modes with Improved Security against Generic Collision Attacks. Cryptology ePrint Archive, Report 2015/757, 2015. `http://eprint.iacr.org/2015/757`.

[HK18] Matthias Hamann and Matthias Krause. On stream ciphers with provable beyond-the-birthday-bound security against time-memory-data tradeoff attacks. *Cryptography and Communications*, 10(5):959–1012, Sep 2018.

[HKL+12] Stefan Heyse, Eike Kiltz, Vadim Lyubashevsky, Christof Paar, and Krzysztof Pietrzak. Lapin: An Efficient Authentication Protocol Based on Ring-LPN. In Anne Canteaut, editor, *Fast Software Encryption*, volume 7549 of *Lecture Notes in Computer Science*, pages 346–365. Springer Berlin Heidelberg, 2012.

[HKM17a] Matthias Hamann, Matthias Krause, and Willi Meier. A Note on Stream Ciphers that Continuously Use the IV. Cryptology ePrint Archive, Report 2017/1172, 2017. `https://eprint.iacr.org/2017/1172`.

[HKM17b] Matthias Hamann, Matthias Krause, and Willi Meier. LIZARD – A Lightweight Stream Cipher for Power-constrained Devices. *IACR Transactions on Symmetric Cryptology*, 2017(1):45–79, 2017.

[HKMZ17a] Matthias Hamann, Matthias Krause, Willi Meier, and Bin Zhang. On Stream Ciphers with Small State. Early Symmetric Crypto (ESC), January 2017, Canach, Luxembourg, 2017.

[HKMZ17b] Matthias Hamann, Matthias Krause, Willi Meier, and Bin Zhang. Time-Memory-Data Tradeoff Attacks against Small-State Stream Ciphers. Cryptology ePrint Archive, Report 2017/384, 2017. `http://eprint.iacr.org/2017/384`.

[HKMZ18] Matthias Hamann, Matthias Krause, Willi Meier, and Bin Zhang. Design and analysis of small-state grain-like stream ciphers. *Cryptography and Communications*, 10(5):803–834, Sep 2018.

[HS05] Jin Hong and Palash Sarkar. New Applications of Time Memory Data Tradeoffs. In *Advances in Cryptology - ASIACRYPT 2005: 11th International Conference on the Theory and Application of Cryptology and Information Security, Chennai, India, December 4-8, 2005. Proceedings*, pages 353–372. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[HT16] Viet Tung Hoang and Stefano Tessaro. Key-alternating Ciphers and Key-length Extension: Exact Bounds and Multi-user Security. Cryptology ePrint Archive, Report 2016/578, 2016. `http://eprint.iacr.org/2016/578`.

[Ins97] Institute of Electrical and Electronics Engineers. IEEE Standard for Information Technology – Telecommunications and Information Exchange Between Systems – Local and Metropolitan Area Networks – Specific Requirements – Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. *IEEE Std 802.11-1997*, pages i–445, 1997.

[Ins04] Institute of Electrical and Electronics Engineers. IEEE Standard for Information Technology – Telecommunications and Information Exchange Between Systems – Local and Metropolitan Area Networks – Specific Requirements – Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications: Amendment 6: Medium Access Control (MAC) Security Enhancements. *IEEE Std 802.11i-2004*, pages 1–190, July 2004.

[Ins12] Institute of Electrical and Electronics Engineers. IEEE Standard for Information Technology – Telecommunications and Information Exchange Between Systems – Local and Metropolitan Area Networks – Specific Requirements – Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. *IEEE Std 802.11-2012 (Revision of IEEE Std 802.11-2007)*, pages 1–2793, March 2012.

[Int97] Interplay Productions. Atomic Bomberman. Website of Wikpedia (accessed on February 19, 2018), 1997. `https://en.wikipedia.org/wiki/Atomic_Bomberman`.

[ISY91] Nagisa Ishiura, Hiroshi Sawada, and Shuzo Yajima. Minimization of binary decision diagrams based on exchanges of variables. In *1991 IEEE International Conference on Computer-Aided Design Digest of Technical Papers*, pages 472–475, Nov 1991.

[JPST17] Jérémy Jean, Thomas Peyrin, Siang Sim, and Jade Tourteaux. Optimizing Implementations of Lightweight Building Blocks. *IACR Transactions on Symmetric Cryptology*, 2017(4):130–168, 2017.

[Jue06] Ari Juels. RFID Security and Privacy: A Research Survey. *IEEE Journal on Selected Areas in Communications*, 24(2):381–394, September 2006.

[JW05] Ari Juels and Stephen A. Weis. Authenticating Pervasive Devices with Human Protocols. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005. Proceedings*, pages 293–308. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[Ker83a] Auguste Kerckhoffs. La cryptographie militaire. *Journal des sciences militaires*, IX:5–38, Jan 1883.

[Ker83b] Auguste Kerckhoffs. La cryptographie militaire. *Journal des sciences militaires*, IX:161–191, Feb 1883.

[KH11] Matthias Krause and Matthias Hamann. The Cryptographic Power of Random Selection. In *Proceedings of SAC 2011*, volume 7118 of *LNCS*, pages 134–150. Springer, 2011.

[KJAB17] Ahmed Khattab, Zahra Jeddi, Esmaeil Amini, and Magdy Bayoumi. *RFID Security: A Lightweight Paradigm*. Analog Circuits and Signal Processing. Springer International Publishing, 2017.

[KPC+11] Eike Kiltz, Krzysztof Pietrzak, David Cash, Abhishek Jain, and Daniele Venturi. Efficient authentication from hard learning problems. In *Proceedings of Eurocrypt 2011*, volume 6632 of *LNCS*, pages 7–26. Springer, 2011.

[Kra02] Matthias Krause. BDD-Based Cryptanalysis of Keystream Generators. In Lars R. Knudsen, editor, *Advances in Cryptology — EUROCRYPT 2002: International Conference on the Theory and Applications of Cryptographic Techniques Amsterdam, The Netherlands, April 28 – May 2, 2002 Proceedings*, pages 222–237. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002.

[Kra17] Matthias Krause. On the Hardness of Trivium and Grain with respect to Generic Time-Memory-Data Tradeoff Attacks. Cryptology ePrint Archive, Report 2017/289, 2017. `http://eprint.iacr.org/2017/289`.

[KS06] Matthias Krause and Dirk Stegemann. Reducing the Space Complexity of BDD-Based Attacks on Keystream Generators. In Matthew Robshaw, editor, *Fast Software Encryption*, pages 163–178, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[KS09] Matthias Krause and Dirk Stegemann. More on the Security of Linear RFID Authentication Protocols. In *Proceedings of SAC 2009*, volume 5867 of *LNCS*, pages 182–196. Springer, 2009.

[Küç06] Özgül Küçük. Slide Resynchronization Attack on the Initialization of Grain 1.0. eSTREAM, ECRYPT Stream Cipher Project, Report 2006/044, 2006. `http://www.ecrypt.eu.org/stream`.

[LF06] Éric Levieil and Pierre-Alain Fouque. An improved LPN algorithm. In *Security and Cryptography for Networks*, pages 348–359. Springer, 2006.

[Liu17] Meicheng Liu. Degree Evaluation of NFSR-Based Cryptosystems. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017: 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20–24, 2017, Proceedings, Part III*, pages 227–249. Springer International Publishing, Cham, 2017.

[LM12] Michael Lehmann and Willi Meier. Conditional Differential Cryptanalysis of Grain-128a. In Josef Pieprzyk, Ahmad-Reza Sadeghi, and Mark Manulis, editors, *Cryptology and Network Security: 11th International Conference, CANS 2012, Darmstadt, Germany, December 12-14, 2012. Proceedings*, pages 1–11. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[LMV05] Yi Lu, Willi Meier, and Serge Vaudenay. The Conditional Correlation Attack: A Practical Attack on Bluetooth Encryption. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005. Proceedings*, pages 97–117. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005.

[LNP15] Virginie Lallemand and María Naya-Plasencia. Cryptanalysis of Full Sprout. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology – CRYPTO 2015: 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*, pages 663–682. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.

[MAM17] Vasily Mikhalev, Frederik Armknecht, and Christian Müller. On Ciphers that Continuously Access the Non-Volatile Key. *IACR Transactions on Symmetric Cryptology*, 2016(2):52–79, 2017.

[Mei18] Willi Meier. TMD tradeoffs on small-state stream ciphers. Dagstuhl Seminar 18021: Symmetric Cryptography, January 2018, Dagstuhl, Germany, 2018.

[Men17] Mentor. ModelSim. Website (accessed on October 17, 2017), 2017. `http://www.mentor.com/`.

[Mih96] Miodrag J. Mihaljević. A faster cryptanalysis of the self-shrinking generator. In Josef Pieprzyk and Jennifer Seberry, editors, *Information Security and Privacy: First Australasian Conference, ACISP'96 Wollongong, NSW, Australia, June 24–26, 1996 Proceedings*, pages 182–189. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996.

[MJSC16] Pierrick Méaux, Anthony Journault, François-Xavier Standaert, and Claude Carlet. Towards Stream Ciphers for Efficient FHE with Low-Noise Ciphertexts. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part I*, pages 311–343. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.

[MME+11] Honorio Martin, Enrique San Millán, Luis Entrena, Julio César Hernández Castro, and Pedro Peris-Lopez. AKARI-X: A pseudorandom number generator for secure lightweight systems. In *IOLTS*, pages 228–233, 2011.

[MPC04] Willi Meier, Enes Pasalic, and Claude Carlet. Algebraic Attacks and Decomposition of Boolean Functions. In Christian Cachin and Jan L. Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004: International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004. Proceedings*, pages 474–491. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004.

[MS88] Willi Meier and Othmar Staffelbach. Fast Correlation Attacks on Stream Ciphers. In D. Barstow, W. Brauer, P. Brinch Hansen, D. Gries, D. Luckham, C. Moler, A. Pnueli, G. Seegmüller, J. Stoer, N. Wirth, and Christoph G. Günther, editors, *Advances in Cryptology — EUROCRYPT '88: Workshop on the Theory and Application of Cryptographic Techniques Davos, Switzerland, May 25–27, 1988 Proceedings*, pages 301–314, Berlin, Heidelberg, 1988. Springer Berlin Heidelberg.

[MS89] Willi Meier and Othmar Staffelbach. Fast correlation attacks on certain stream ciphers. *Journal of Cryptology*, 1(3):159–176, 1989.

[MS94] Willi Meier and Othmar Staffelbach. The Self-Shrinking Generator. In *Advances in Cryptology - EUROCRYPT '94, Workshop on the Theory and Application of Cryptographic Techniques, Perugia, Italy, May 9-12, 1994, Proceedings*, volume 950 of *Lecture Notes in Computer Science*, pages 205–214. Springer, 1994.

[MSBD15] Subhamoy Maitra, Santanu Sarkar, Anubhab Baksi, and Pramit Dey. Key Recovery from State Information of Sprout. Cryptology ePrint Archive, Report 2015/236, 2015. `http://eprint.iacr.org/2015/236.pdf`.

[MSGAHJ13] Joan Melià-Seguí, Joaquin Garcia-Alfaro, and Jordi Herrera-Joancomartí. J3Gen: A PRNG for Low-Cost Passive RFID. *Sensors*, 13(3):3816–3830, 2013.

[MSS+17] Subhamoy Maitra, Nishant Sinha, Akhilesh Siddhanti, Ravi Anand, and Sugata Gangopadhyay. A TMDTO Attack Against Lizard. Cryptology ePrint Archive, Report 2017/647, 2017. `http://eprint.iacr.org/2017/647`.

[Nat16] National Institute of Standards and Technology (NIST). Cryptographic Standards and Guidelines: AES Development. Website (accessed on February 22, 2018), 2016. `http://csrc.nist.gov/archive/aes/`.

[NESP08] Karsten Nohl, David Evans, Starbug Starbug, and Henryk Plötz. Reverse-engineering a Cryptographic RFID Tag. In *Proceedings of the 17th Conference on Security Symposium*, SS'08, pages 185–193, Berkeley, CA, USA, 2008. USENIX Association.

[NKTZ12] Andrey Nuykin, Alexander Kravtsov, Sergey Timoshin, and Igor Zubov. A low cost EEPROM design for passive RFID tags. In *Communications and Electronics (ICCE), 2012 Fourth International Conference on*, pages 443–446, Aug 2012.

[NP07] Karsten Nohl and Henryk Plötz. Mifare: Little Security, Despite Obscurity. Website (accessed on September 16, 2017), 2007. `https://events.ccc.de/congress/2007/Fahrplan/events/2378.en.html`.

[NXP17] NXP Semiconductors. MIFARE. Website (accessed on September 16, 2017), 2017. `https://www.mifare.net/`.

[OOV08] Khaled Ouafi, Raphael Overbeck, and Serge Vaudenay. On the Security of HB# against a Man-in-the-Middle Attack. In Josef Pieprzyk, editor, *Advances in Cryptology - ASIACRYPT 2008: 14th International Conference on the Theory and Application of Cryptology and Information Security, Melbourne, Australia, December 7-11, 2008. Proceedings*, pages 108–124. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[OP11] David Oswald and Christof Paar. Breaking Mifare DESFire MF3ICD40: Power Analysis and Templates in the Real World. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems*

– *CHES 2011: 13th International Workshop, Nara, Japan, September 28 – October 1, 2011. Proceedings*, pages 207–222. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[PLHCETR09] Pedro Peris-Lopez, Julio Cesar Hernandez-Castro, Juan M. Estevez-Tapiador, and Arturo Ribagorda. LAMED - A PRNG for EPC Class-1 Generation-2 RFID Specification. *Comput. Stand. Interfaces*, 31(1):88–97, January 2009.

[Pli99] John Pliam. Authentication Vulnerabilities in IKE and Xauth with Weak Pre-Shared Secrets. archive.org – WayBackMachine, Version: March 08, 2005 (accessed on September 6, 2017), 1999. `https://web.archive.org/web/20050308093355/https://www.ima.umn.edu/~pliam/xauth/`.

[PMK+11] Axel Poschmann, Amir Moradi, Khoongming Khoo, Chu-Wee Lim, Huaxiong Wang, and San Ling. Side-Channel Resistant Crypto for Less than 2,300 GE. *Journal of Cryptology*, 24(2):322–345, Apr 2011.

[Pop15] A. Popov. Prohibiting RC4 Cipher Suites. RFC 7465 (Proposed Standard), February 2015.

[Pos09] Axel Poschmann. Lightweight Cryptography - Cryptographic Engineering for a Pervasive World. Cryptology ePrint Archive, Report 2009/516, 2009. `http://eprint.iacr.org/2009/516`.

[Pre03] Bart Preneel. NESSIE: New European Schemes for Signatures, Integrity, and Encryption. Website (accessed on October 19, 2017), 2003. `https://www.cosic.esat.kuleuven.be/nessie/`.

[PRO17] PROXMARK.org. Proxmark 3. Website (accessed on September 18, 2017), 2017. `http://www.proxmark.org/`.

[PS95] Shipra Panda and Fabio Somenzi. Who Are the Variables in Your Neighborhood. In *Proceedings of the 1995 IEEE/ACM International Conference on Computer-aided Design*, ICCAD '95, pages 74–77, Washington, DC, USA, 1995. IEEE Computer Society.

[PSP94] Shipra Panda, Fabio Somenzi, and Bernard F. Plessier. Symmetry Detection and Dynamic Variable Ordering of Decision Diagrams. In *Proceedings of the 1994 IEEE/ACM International Conference on Computer-aided Design*, ICCAD '94, pages 628–631, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.

[REC05] Damith C. Ranasinghe, Daniel W. Engels, and Peter H. Cole. Low-Cost RFID Systems: Confronting Security and Privacy. In *In: Auto-ID Labs Research Workshop*. Portal, 2005.

[Rep13] Craig A. Repec. Regulatory status for using RFID in the EPC Gen 2 band (860 to 960 MHz) of the UHF spectrum, 2013. `http://www.gs1.org/docs/epcglobal/UHF_Regulations.pdf`.

[Rob05] Mark Roberti. The History of RFID Technology. Website of RFID Journal (accessed on October 20, 2017), 2005. `http://www.rfidjournal.com/articles/view?1338/`.

[RPLP08] Carsten Rolfes, Axel Poschmann, Gregor Leander, and Christof Paar. Ultra-Lightweight Implementations for Smart Devices – Security for 1000 Gate Equivalents. In Gilles Grimaud and François-Xavier Standaert, editors, *Smart Card Research and Advanced Applications: 8th IFIP WG 8.8/11.2 International Conference, CARDIS 2008, London, UK, September 8-11, 2008. Proceedings*, pages 89–103. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[Rud93] Richard Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Proceedings of 1993 International Conference on Computer Aided Design (ICCAD)*, pages 42–47, Nov 1993.

[RUS04] RUS-CERT. RUS-CERT-1195: [Generic/IPsec] Risiken von XAUTH. Website (accessed on September 6, 2017), 2004. `https://cert.uni-stuttgart.de/ticker/article.php?mid=1195`.

[Sch95] Bruce Schneier. *Applied Cryptography (2nd Ed.): Protocols, Algorithms, and Source Code in C*. John Wiley & Sons, Inc., New York, NY, USA, 1995.

[SCU11] Jean-Ferdinand Susini, Harvé Chabanne, and Pascal Urien. *RFID and the Internet of Things*, chapter RFID and the Internet of Things, page 304. ISTE - John Wiley & Sons, 2011.

[SE12] Markku-Juhani O. Saarinen and Daniel Engels. A Do-It-All-Cipher for RFID: Design Requirements (Extended Abstract). Cryptology ePrint Archive, Report 2012/317, 2012. `http://eprint.iacr.org/2012/317`.

[Sie85] Thomas Siegenthaler. Decrypting a Class of Stream Ciphers Using Ciphertext Only. *IEEE Transactions on Computers*, 34(1):81–85, January 1985.

[SM17] Akhilesh Siddhanti Subhamoy Maitra. A differential fault attack on plantlet. Cryptology ePrint Archive, Report 2017/088, 2017. `http://eprint.iacr.org/2017/088`.

[SNC09] Mate Soos, Karsten Nohl, and Claude Castelluccia. Extending SAT Solvers to Cryptographic Problems. In Oliver Kullmann, editor, *Theory and Applications of Satisfiability Testing - SAT 2009*, pages 244–257, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[Som15] Fabio Somenzi. CUDD 3.0.0. Website (accessed on February 05, 2018), 2015. `http://vlsi.colorado.edu/~fabio/`.

[Soo15] Mate Soos. CryptoMiniSat 4. Website (accessed on February 02, 2018), 2015. `https://www.msoos.org/cryptominisat4/`.

[SS16] Sumanta Sarkar and Habeeb Syed. Lightweight Diffusion Layer: Importance of Toeplitz Matrices. *IACR Transactions on Symmetric Cryptology*, 2016(1), 2016.

[SSMC17] Akhilesh Anilkumar Siddhanti, Santanu Sarkar, Subhamoy Maitra, and Anupam Chattopadhyay. Differential Fault Attack on Grain v1, ACORN v3 and Lizard. Cryptology ePrint Archive, Report 2017/678, 2017. `http://eprint.iacr.org/2017/678`.

[Ste07a] Dirk Stegemann. Extended BDD-Based Cryptanalysis of Keystream Generators. In Carlisle Adams, Ali Miri, and Michael Wiener, editors, *Selected Areas in Cryptography: 14th International Workshop, SAC 2007, Ottawa, Canada, August 16-17, 2007, Revised Selected Papers*, pages 17–35. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[Ste07b] Dirk Stegemann. Extended BDD-based Cryptanalysis of Keystream Generators (Presentation). 14th Workshop on Selected Areas in Cryptography (SAC), 2007, Canada, 2007.

[STF05] Thorsten Staake, Frédéric Thiesse, and Elgar Fleisch. Extending the EPC network: the potential of RFID in anti-counterfeiting. In *Proceedings of the 2005 ACM symposium on Applied computing*, pages 1607–1612. ACM, 2005.

[Sul04] Philippe Sultan. An example of VPN server spoofing. Website (accessed on September 6, 2017), 2004. `https://who.rocq.inria.fr/Philippe.Sultan/vpn/spoofed_vpn_server.html`.

[TBM08] Carlos Tokunaga, David Blaauw, and Trevor Mudge. True Random Number Generator With a Metastability-Based Quality Control. *IEEE Journal of Solid-State Circuits*, 43(1):78–85, Jan 2008.

[TIHM17] Yosuke Todo, Takanori Isobe, Yonglin Hao, and Willi Meier. Cube Attacks on Non-Blackbox Polynomials Based on Division Property. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017: 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20–24, 2017, Proceedings, Part III*, pages 250–279. Springer International Publishing, Cham, 2017.

[Tod15] Yosuke Todo. Structural Evaluation by Generalized Integral Property. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015: 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part I*, pages 287–314, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[Tod17] Yosuke Todo. Personal communication, 2017.

[Uni94] University of California, Berkeley. Espresso, 1994. `http://embedded.eecs.berkeley.edu/pubs/downloads/espresso/index.htm`.

[van05] vantronix | secure systems GmbH. void11. archive.org – WayBackMachine, Version: February 06, 2005 (accessed on September 6, 2017), 2005. `https://web.archive.org/web/20050206100144/http://www.wlsec.net:80/void11/`.

[Vau07] Serge Vaudenay. On Privacy Models for RFID. In Kaoru Kurosawa, editor, *Advances in Cryptology – ASIACRYPT 2007: 13th International Conference on the Theory and Application of Cryptology and Information Security, Kuching, Malaysia, December 2-6, 2007. Proceedings*, pages 68–87. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[Ver13] Ingrid Verbauwhede. How much crypto in one microJoule? Real World Crypto 2013 – Stanford, CA, 2013. `https://crypto.stanford.edu/RealWorldCrypto/slides/ingrid.pdf`.

[vTJ11] Henk C. A. van Tilborg and Sushil Jajodia, editors. *Encyclopedia of Cryptography and Security*, page 675. Springer US, Boston, MA, 2011.

[Weg00] Ingo Wegener. *Branching Programs and Binary Decision Diagrams: Theory and Applications*. SIAM e-books. Society for Industrial and Applied Mathematics, 2000.

[Win18] Nick Wingfield. Inside Amazon Go, a Store of the Future. Website of The New York Times (accessed on February 16, 2018), 2018. `https://www.nytimes.com/2018/01/21/technology/inside-amazon-go-a-store-of-the-future.html`.

[Wu16] Hongjun Wu. Acorn v3. Submission to CAESAR competition., 2016.

[WZ11] Wenling Wu and Lei Zhang. LBlock: A Lightweight Block Cipher. In *Applied Cryptography and Network Security*, volume 6715 of *LNCS*, pages 327–344. Springer, 2011.

[Xel17] Xelerance Corp. Openswan. Website (accessed on September 6, 2017), 2017. `https://www.openswan.org/`.

[Xil17] Xilinx, Inc. Xilinx Website, 2017. `http://www.xilinx.com/`.

[ZG15] Bin Zhang and Xinxin Gong. Another Tradeoff Attack on Sprout-Like Stream Ciphers. In Tetsu Iwata and Hee Jung Cheon, editors, *Advances in Cryptology – ASIACRYPT 2015: 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 – December 3, 2015, Proceedings, Part II*, pages 561–585. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015.

[ZGM17] Bin Zhang, Xinxin Gong, and Willi Meier. Fast Correlation Attacks on Grain-like Small State Stream Ciphers. *IACR Transactions on Symmetric Cryptology*, 2017(4):58–81, 2017.

[ZKL01] Erik Zenner, Matthias Krause, and Stefan Lucks. Improved Cryptanalysis of the Self-Shrinking Generator. In Vijay Varadharajan and Yi Mu, editors, *Information Security and Privacy: 6th Australasian Conference, ACISP 2001 Sydney, Australia, July 11–13, 2001 Proceedings*, pages 21–35. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[ZW09] Haina Zhang and Xiaoyun Wang. Cryptanalysis of Stream Cipher Grain Family. Cryptology ePrint Archive, Report 2009/109, 2009. `http://eprint.iacr.org/2009/109`.

[ZXM18] Bin Zhang, Chao Xu, and Willi Meier. Fast Near Collision Attack on the Grain v1 Stream Cipher. Cryptology ePrint Archive, Report 2018/145 (accepted for IACR EUROCRYPT 2018), 2018. `https://eprint.iacr.org/2018/145`.

Erklärung der Urheberschaft

Eidesstattliche Versicherung gemäß § 7 Absatz 2 Buchstabe c) der Promotionsordnung der Universität Mannheim (Stand: 11. Juni 2012) zur Erlangung des Doktorgrades der Naturwissenschaften:

1. Bei der eingereichten Dissertation zum Thema
   Lightweight Cryptography on Ultra-Constrained RFID Devices
   handelt es sich um mein eigenständig erstelltes eigenes Werk.

2. Ich habe nur die angegebenen Quellen und Hilfsmittel benutzt und mich keiner unzulässigen Hilfe Dritter bedient. Insbesondere habe ich wörtliche Zitate aus anderen Werken als solche kenntlich gemacht.

3. Die Arbeit oder Teile davon habe ich bislang nicht an einer Hochschule des In- oder Auslands als Bestandteil einer Prüfungs- oder Qualifikationsleistung vorgelegt.

4. Die Richtigkeit der vorstehenden Erklärung bestätige ich.

5. Die Bedeutung der eidesstattlichen Versicherung und die strafrechtlichen Folgen einer unrichtigen oder unvollständigen eidesstattlichen Versicherung sind mir bekannt.

Ich versichere an Eides statt, dass ich nach bestem Wissen die reine Wahrheit erklärt und nichts verschwiegen habe.

Mannheim, 04. Juni 2018 Matthias Alexander Hamann