Towards Low Energy Stream Ciphers

Subhadeep Banik[1], Vasily Mikhalev[2], Frederik Armknecht[2], Takanori Isobe[3], Willi Meier[4], Andrey Bogdanov[5], Yuhei Watanabe[6] and Francesco Regazzoni[7]

[1] LASEC, École Polytechnique Fédérale de Lausanne, Switzerland, subhadeep.banik@epfl.ch
[2] University of Mannheim, Germany {mikhalev,armknecht}@uni-mannheim.de
[3] University of Hyogo, Japan, takanori.isobe@ai.u-hyogo.ac.jp
[4] FHNW Switzerland, willi.meier@fhnw.ch
[5] DTU Compute, Technical University of Denmark, Lyngby anbog@dtu.dk
[6] National Institute of Advanced Industrial Science and Technology, Osaka, Japan yuhei.watanabe@aist.go.jp
[7] University of Lugano, Switzerland regazzoni@alari.ch

Abstract. Energy optimization is an important design aspect of lightweight cryptography. Since low energy ciphers drain less battery, they are invaluable components of devices that operate on a tight energy budget such as handheld devices or RFID tags. At Asiacrypt 2015, Banik et al. presented the block cipher family `Midori` which was designed to optimize the energy consumed per encryption and which reduces the energy consumption by more than 30% compared to previous block ciphers. However, if one has to encrypt/decrypt longer streams of data, i.e. for bulk data encryption/decryption, it is expected that a stream cipher should perform even better than block ciphers in terms of energy required to encrypt.
In this paper, we address the question of designing low energy stream ciphers. To this end, we analyze for common stream cipher design components their impact on the energy consumption. Based on this, we give arguments why indeed stream ciphers allow for encrypting long data streams with less energy than block ciphers and validate our findings by implementations. Afterwards, we use the analysis results to identify energy minimizing design principles for stream ciphers.

Keywords: Lightweight block cipher, Low energy, Stream Cipher

1 Introduction

1.1 Motivation

The field of lightweight cryptography has seen a number of cipher proposals in the past few years, with block ciphers like `CLEFIA` [SSA+07], `KATAN` [CDK09], `Klein` [GNL11], `LED` [GPPR11], Midori [BBI+15], `PRESENT` [BKL+07], `Piccolo` [SIH+11], `PRINCE` [BCG+12], `SIMON/SPECK` [BSS+13] and stream ciphers like Lizard [HKM17] and Plantlet [MAM16] to name a few. However, the Advanced Encryption Standard (`AES`) [DR02] still remains the de-facto standard when it comes to practical lightweight encryption, also due to the numerous low-power/area architectures for `AES` being reported in literature [MPL+11, SMTM01, FWR05].

However, we argue that for battery driven devices that run on tight battery budgets like handheld devices, medical implants or RFID tags, a more relevant parameter is the *energy consumption*. In a nutshell, it is a measure of the total electrical work done by the battery source during the execution of any operation. In fact, some previous works have investigated the energy efficiency of block ciphers. In [BDE+13, KDH+12], an evaluation

IACR Transactions on Symmetric Cryptology ISSN 2519-173X, Vol. 2018, No. 2, pp. 1–19
DOI:10.13154/tosc.v2018.i2.1-19

of several lightweight block ciphers with respect to various hardware performance metrics, with a particular focus on the energy cost, was done. In [BBR15], the authors looked at design strategies like serialization and round unrolling and the effect it has on the energy consumption required to encrypt a single block of data. Serialization stretches out the execution of each round function over a number of clock cycles and hence was found to be unsuitable for energy efficiency. The authors then proposed a formal model for energy consumption in any r-round unrolled block cipher architecture. The authors concluded that the energy consumed for encrypting one block of plaintext for any r-round unrolled implementation had a quasi-quadratic form (a, b, c are constants and R is the number of iterations of the round function prescribed for the design):

$$E_r = (ar^2 + br + c) \cdot \left(1 + \left\lceil \frac{R}{r} \right\rceil\right), \tag{1}$$

where $ar^2 + br + c$ denotes the energy consumed per cycle and $\left(1 + \lceil \frac{R}{r} \rceil\right)$ is the total clock cycles required to encrypt. Although an r-round unrolled cipher consumes *more energy per cycle* for increasing values of r, it takes *fewer cycles to complete the encryption operation* itself. This makes the determination of the values of r at which the design has the lowest energy consumption an interesting and important optimization problem. The authors concluded that for block ciphers with lightweight round functions like `PRESENT` and `SIMON`, $r = 2$ was the optimal configuration, whereas for "heavier" round functions like in `AES` and `Noekeon`, $r = 1$ was optimal. Building on these ideas, the block cipher family Midori was proposed in [BBI+15] that optimized the energy consumption per encryption.

However, previous work in this field [BBR15, BBI+15, BDE+13, KDH+12] has focused on the energy consumption for encrypting *one block* of data. While this is reasonable for scenarios that require the encryption of short data bursts, we show that when it comes to encrypting significantly large data, a stream cipher may be energy-wise a better solution than a block cipher. Stream ciphers like Grain [HJM07] and Trivium [CP08] use an extremely simple state update operation that typically involves to compute multiple boolean functions and state rotation. As a result, unrolling multiple rounds of a stream cipher usually only involves to realize additional copies of the boolean function circuit (if the number of rounds unrolled is small). This has the consequence that the power consumption in stream cipher circuits increases very slowly with the number of rounds unrolled. On the other hand, the number of clock cycles required to encrypt a given amount of plaintext drops linearly with the level of unrolling and so does the energy required to perform the encryption operation. This makes stream cipher promising candidates for low energy encryption.

As an instructive example, we compare the energy consumptions of the single and two-round unrolled Grain v1 circuits.

- A single round implementation of the Grain v1 circuit synthesized with the standard cell library of the STM 90nm logic process, takes around 1164 GE and has an average power consumption of 40.567 uW at a clock frequency of 10 MHz. In order to encrypt 64 bits of data, the circuit has to operate for 1 (loading the Key-IV) + 160 (for Key-IV mixing) + 64 = 225 clock cycles. Therefore the energy required for the operation is approximately $40.567 * 225 \approx 912.8$ pJ.

- A two-round unrolled Grain v1 circuit, which performs 2 round operations in one clock cycle, has an area of around 1200 GE and an average power consumption of around 41 uW. However this circuit requires only 1+80+32=113 clock cycles to encrypt 64-bit data, and so the energy requirement is only around 463 pJ. So a 2x unrolling results in approximately a 2x reduction in energy.

Consequently, for a cipher like Grain v1 which was specifically designed to allow for efficient

unrolling of up to 16 rounds, we expect the trend to persist for at least up to 16th degree of unrolling and perhaps beyond that as well.

1.2 Contribution

In this paper we investigate the energy consumption traits of stream ciphers. For our analysis, we select the stream ciphers Trivium [CP08], Grain v1 [HJM07], Grain-128 [HJMM06], Lizard [HKM17], Plantlet [MAM16], and Kreyvium [CCF+16]. We take a look at all implementation level aspects that are likely to affect the energy consumption of stream ciphers and then draw conclusions from our studies. Our principal finding from these experiments was that the 160x unrolled implementation of Trivium is about 9 times more energy efficient than any block cipher based solution for encrypting long data streams, and that unrolled stream ciphers in general outperform block ciphers in this domain.

1.3 Organization

The paper is organized as follows. In Section 2, we take a look at the factors that may affect the energy consumption of stream ciphers. We try to identify parameters that result in increase/decrease the energy consumption and try to draw necessary conclusions. Section 3 concludes the paper.

2 Energy-Impact of Design Components

In [BBI+15], it was pointed out that for any given block cipher, the three main factors that determine the quantity of energy dissipated in the circuit are:

(a) The clock frequency,

(b) the architecture of the individual components, and

(c) the number of unrolled rounds.

Since stream ciphers possess the same basic architecture as block ciphers in the sense that both are round-based, the same is likely to be true (to some extent) for a stream cipher as well. In this section, we investigate factors that may affect the energy consumption of stream ciphers. The aim is to identify design principles and parameters that a designer can choose to increase/decrease the energy consumption. To this end, we perform several experiments with respect to the three factors mentioned above from which we derive characteristics that an energy efficient stream cipher should possess. In all the simulations reported in the paper, we maintained the following design flow. First, the design was implemented at RTL level. A functional verification of the VHDL code was then done using *Mentorgraphics ModelSim*. Thereafter, *Synopsys Design Compiler* was used to synthesize the RTL design using the standard cell library of the STM 90nm CMOS logic process. The switching activity of each gate of the circuit was collected by running post-synthesis simulation. The average power was obtained using *Synopsys Power Compiler*, using the back annotated switching activity. The energy was then computed as the product of the average power and the total time taken for the encryption process.

2.1 Frequency of Operation

Note that the total energy dissipation for a CMOS gate can be written as $E_{gate} = E_{dynamic} + E_{static}$ where

- $E_{dynamic}$ refers to the dynamic dissipation which is due to the charging and discharging of load capacitances and the short-circuit current and

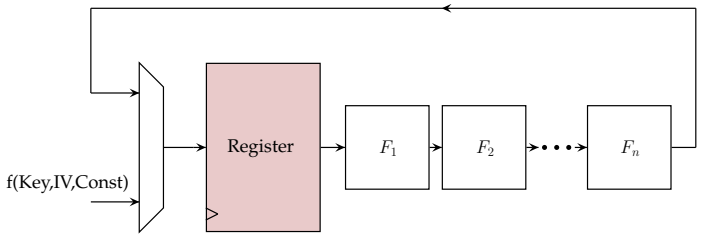

Figure 1: n round unrolled implementation of a stream cipher

- E_{static} denotes the static dissipation which is due to leakage current and other current drawn continuously from the power supply.

As pointed out in [KDH+12, BBR15], as the energy consumption is measured by the total number of switching activities of a circuit during the encryption process, it should be independent of the frequency of operation. While this is true at high frequencies where dynamic energy $E_{dynamic}$ consumed is significantly larger than the total static energy E_{static} consumed by the system, the situation changes at lower frequencies. It was shown in [BBR15] that for circuits designed with the standard cell library of the STM 90nm CMOS process, at frequencies lower than 1 MHz the static energy gets a higher impact. To remedy this effect, we fixed at our experiments the frequency of operation to 10 MHz (this corresponds to a clock period of 100 ns), so that the leakage power plays minimal role in the energy consumption.

2.2 Architecture

Often, there are different options for implementing a stream cipher. We will take a detailed look at a few of them:

A. Scan Flip-Flops vs Regular Flip-Flops Figure 1 depicts the diagram of a stream cipher which has been unrolled n times. Unrolling in stream ciphers refers to implementations where we include logic gates for several instantiations of the update function such that multiple rounds can be executed within a single clock cycle. In a stream cipher, the storage elements, commonly realized by flip-flops, are usually preceded by a multiplexer, which in the initial clock cycle filters a combination of the key and IV on to the register and the output of the round function thereafter. The combination of flip-flop and multiplexer can be replaced with a *scan flip-flop* which provides the same logical functionality while occupying less area and less power. Hence the intuition is that designs using scan flip-flops would be more energy-efficient than those based on combining flip-flops and multiplexer.

To investigate this, we executed simulations for several hardware-based stream ciphers. The results are tabulated in Table 1. The table shows simulation results for the six hardware-based stream ciphers Grain v1, Grain 128, Trivium, Plantlet, Lizard [1] and Kreyvium, synthesized with the standard cell library of the STM 90nm logic process. It displays the energy consumptions for encrypting for both 1 and 1000 blocks of plaintext where one **block** is taken to be equal to 64 bits.

[1] According to the designers, LIZARD is supposed to be implemented with serialized key/IV loading. In [HKM17], they show that such an implementation of LIZARD requires less area and power than Grain v1 (with and without serialized key/IV loading). However a serialized loading can not easily be adopted to make a round unrolled structure which is critical to energy minimization, as we will shortly see. So we evaluate all ciphers under one cycle key-IV loading process.

Table 1: A comparison of energy consumptions when using regular flip-flops (R) and scan flip-flops (S).

#	Cipher	FF	Area (GE)	Power (uW) @ 10 MHz	Energy (pJ) 1 block	Energy (nJ) 1000 Blocks
1	Grain v1	R	1164	40.6	912.8	260.28
		S	1005	38.9	874.8	249.47
2	Grain 128	R	1700	71.5	2287.1	459.23
		S	1455	57.8	1855.4	371.41
3	Trivium	R	1870	78.4	9527.6	510.48
		S	1584	75.6	9194.9	492.26
4	Plantlet	R	886	35.4	1364.7	227.99
		S	785	34.4	1363.1	227.73
5	Lizard[2]	R	1481	51.8	1663.2	332.93
		S	1360	50.4	1617.5	323.78
6	Kreyvium	R	3433	146.2	17792.5	952.53
		S	2892	140.8	17135.4	917.35

The results confirm the intuition formulated above. For example, Grain v1 takes 1 (loading key-IV) + 160 (initialization) + 64 = 225 cycles to encrypt 1 block of plaintext. As can be seen in Table 1, in case of using regular flip-flops this results into requiring energy of 225×100 ns $\times$ 40.6 uW $\approx$ 912.8 pJ. Similarly, 161+64000 = 64161 cycles are required to encrypt 1000 blocks, and so the energy required for it can be estimated as 64161×100 ns $\times$ 40.6 uW $\approx$ 260.28 nJ. In contrast, when using scan flip-flops the energy requirement are $\approx$ 874.8 pJ and $\approx$ 249.47 nJ, respectively.

Main Conclusions: We can draw the following conclusions from the results reported in Table 1. Designs implemented with scan flip-flops are shown to be better both with respect to energy consumption and circuit area. Since all other factors remain equal, reduced area when using scan flip-flops results into a reduced power consumption. Since energy is of the power over time, this also results into reduced energy consumption.

B. Fibonacci vs Galois Configuration Practically all stream ciphers deploy feedback shift registers (FSR), either with linear update function (LFSR) or non-linear update function (NLFSR). For these, a designer has the choice between the Fibonacci and the Galois configuration. For instance, the designs of stream ciphers Grain v1, Grain 128 and Trivium consider the Fibonacci configuration of the deployed LFSRs. As shown in Figure 2A, a shift register in Fibonacci configuration updates its states by shifting all bits by one position and by inserting at the final position a bit that has been computed by the round function from the current state (before shifting). In comparison, in a shift register in Galois configuration each state bit is updated using a function separate f_i applied to the entire current state (cf. Figure 2B).[3] Galois equivalent implementations for Grain v1 and Grain 128 were proposed in [MD10, Dub09]. The authors showed that Galois configurations usually have lower circuit latency and thus can allow for higher throughput. In Table 2, we tabulate for several ciphers a comparison between Galois and Fibonacci configurations.

Note that we have to omit the Plantlet stream cipher as a realization using Galois configuration is not possible. This is due to the fact that the non-linear register

[2]We use the implementation of Lizard that loads key-IV in one clock cycle

[3]Note that although it is not shown explicitly in Figure 2, each of the functions f_i are computed over the entire state and not just the preceding bit.

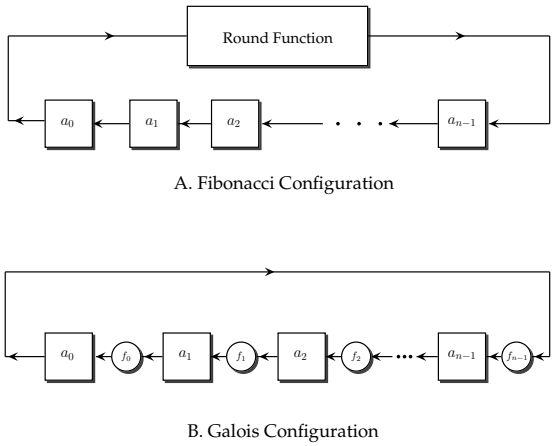

Figure 2: Fibonacci and Galois Configurations for Shift registers

Table 2: Results for Fibonacci vs Galois configurations, 1 block=64 bits, G: Galois, F: Fibonacci configuration

#	Cipher	Conf	Area (GE)	Power (uW) @ 10 MHz	Energy (pJ) 1 block	Energy (nJ) 1000 Blocks
1	Grain v1	G	1016	39.8	894.4	255.05
		F	1005	38.9	874.8	249.47
2	Grain 128	G	1466	58.9	1890.9	378.52
		F	1455	57.8	1855.4	371.41
3	Trivium	G	1592	76.0	9253.6	495.40
		F	1584	75.6	9194.9	492.26
4	Lizard	G	1366	50.7	1626.0	325.49
		F	1360	50.4	1617.5	323.78
5	Kreyvium	G	2898	141.3	17196.2	920.61
		F	2892	140.8	17135.4	917.35

update function takes inputs from the 39-th, being the last bit, which doesn't allow to apply the transformation given in [MD10].

Note that both configurations provide the same logical functionality and hence do not offer any significant advantages over the other with respect to classical cryptanalysis. However in [CMM14], it was shown that Galois registers are more vulnerable to power attacks than Fibonacci registers: They were able to find the initial state of the Galois register using approximately half the number of power traces as compared to Fibonacci registers.

Main Conclusions: A Galois configuration does not seem to offer any significant advantage over its Fibonacci counterpart with respect to energy consumption or area size. Moreover, most ciphers designs primarily consider FSRs in Fibonacci configuration. As we will see later, a more energy efficient realization of a stream cipher needs to unroll it a multiple number of times. However, implementations in Galois configuration for ciphers that were primarily designed for the Fibonacci configuration cannot be unrolled beyond a certain limit (see [MD10]). All these make Galois configurations unattractive for low energy designs.

C. Architecture of Round Function The round functions F_i in hardware-based stream

ciphers are generally very simple. They involve a one bit shift (that can be efficiently implemented by shift registers) and one or multiple small boolean functions to update the terminal bit of the register. We look at three possible ways of realizing these.

1. The first approach is to use a look-up table. For an n-variable boolean function this is a table of $2^n \times 1$ entries. For obvious reasons, although effective for small n, this kind of circuit style is inadvisable for larger values of n.
2. The second approach is to feed the functional description (in terms of the algebraic normal form) of the boolean function to the synthesizer and instructing it to optimize for area and power. In this approach, we depend on the ability of the circuit synthesizer.
3. The third approach is to use a Decoder-Switch-Encoder (DSE) style configuration. This approach was previously considered in [BBR15, BBI+15] for designing the 8-bit Rijndael S-box and was shown to be energy efficient. For implementing boolean functions, the first step is the same as for realizing an S-box circuit. We implement the decoder first i.e., for the case of an n-bit input we construct a set of 2^n wires, where logically, each wire represents one of the 2^n possible minterms of n variables. It is easy to see that only one of the wires would hold a logical HIGH signal for any given input value. Since there is one wire corresponding to every minterm, we simply logically OR all the wires whose minterms result in a logical HIGH in the truth table of the function. In fact it is clear, that we don't even need to expend hardware for constructing all 2^n wires: we can do with constructing only those wires whose minterms are present in the canonical normal from of the function we are implementing.

   However, the circuit size is still exponential in the input length, so we adopted a simple tweak. Whenever the number of input variables of a function exceeded 10, we split the function into the sum of two component functions of roughly equal size with an input size of less than 10 and constructed the circuits for each of the component functions. Breaking up the function into components is directly possible for some ciphers. For example, in Plantlet the NFSR update function g is given as

$$\begin{aligned} g = {} & n_0 + n_{13} + n_{19} + n_{35} + n_{39} + n_2 \cdot n_{25} + n_3 \cdot n_5 + n_7 \cdot n_8 + n_{14} \cdot n_{21} + \\ & n_{16} \cdot n_{18} + n_{22} \cdot n_{24} + n_{26} \cdot n_{32} + n_{33} \cdot n_{36} \cdot n_{37} \cdot n_{38} + \\ & n_{10} \cdot n_{11} \cdot n_{12} + n_{27} \cdot n_{30} \cdot n_{31} \end{aligned}$$

   Although this is a function of 29 variables, each variable occurs only once and hence there is no intersection of terms between any 2 monomials. Hence it is easy to break up g as a sum of five functions (say g_1 to g_5) each of 5 or 6 variables, such that no two component functions depend on the same input variable. However, this is not always the case. The NFSR update function of Grain v1 for instance, has 13 variables, and breaking it up into functions of disjoint variables is not straightforward. However the DSE construction does not explicitly require that the inputs of the component functions be disjoint. For example, the Grain v1 NFSR function can be written as the sum of four non-disjoint functions of 8, 7, 6, 3 variables each.

Main Conclusions: In Table 3, we list the simulation results for the three realizations of round function that we discussed. It is clear from the table that LUT or DSE style constructions of the boolean function have no significant advantage over the circuit optimized by the synthesizer.

Table 3: Results for different realizations of the round functions. LUT: Lookup table, FUN: Functional synthesis using Synopsys tool, DSE: DSE configuration

#	Cipher	Conf	Area (GE)	Power (uW) @ 10 MHz	Energy (pJ) 1 block	Energy (nJ) 1000 Blocks
1	Grain v1	LUT	1071	43.3	973.7	277.68
		FUN	1005	38.9	874.8	249.47
		DSE	1088	41.7	938.4	267.61
2	Grain-128	LUT	1449	57.9	1858.3	371.98
		FUN	1455	57.8	1855.4	371.41
		DSE	4165	76.3	2449.0	490.23
3	Trivium	LUT	1589	75.7	9211.1	493.12
		FUN	1584	75.6	9194.9	492.26
		DSE	1680	78.4	9542.8	510.88
4	Plantlet	LUT	785	34.5	1326.3	221.58
		FUN	785	34.4	1324.6	221.30
		DSE	1143	42.7	1644.1	274.68
5	Lizard	LUT	1327	49.9	1601.8	320.64
		FUN	1360	50.4	1617.5	323.78
		DSE	1946	58.5	1878.5	376.03
6	Kreyvium	LUT	2897	141.2	17184.0	919.96
		FUN	2892	140.8	17135.4	917.35
		DSE	2988	144.0	17524.8	938.20

2.3 Unrolling Rounds

Unrolling rounds is a design technique which aims to speed up the circuit throughput at the cost of area. The core idea is to replace the round function designed for one round by an augmented function that implements several rounds within one function. For example, a two round unrolled AES circuit consists of two sequentially placed circuits for the round functions, that computes the ciphertext in only 5 clock cycles (i.e. half the time as compared to a single round circuit). We discussed in Section 1.1 already that unrolling turned out to be an effective method for realizing low energy block ciphers and that we expect similar benefits for stream ciphers. In fact, most hardware stream ciphers have very simple round functions (consisting of a logical shift and a boolean function computation). Consequently, we do not expect a significant increase in the algebraic complexity when unrolling the design at least for the first few rounds. This would translate into a rather small increase in the hardware complexity, a reasonable number of additional logic gates. This would naturally limit the transient switching activity (signal glitches) from one round to the next (cf. [BBR15]). Since less glitches results into a lower power consumption, it is quite often the case that unrolling stream ciphers by one round does not significantly increase the power consumption whereas it always decreases the number of clock cycles required to encrypt a given amount of data. Thus, the overall energy consumption decreases with unrolling. A good example for this effect is the Grain v1 cipher that we discussed already in Section 1.1. The single round and the 2 round unrolled circuits have an average power consumptions of 40.567 and 41 uW respectively, at a clock frequency of 10 MHz. Since the number of clock cycles required to encrypt data in the 2 round circuit is approximately half as compared to the single round circuit, a 2x unrolling results in approximately a 2x reduction in energy as well.

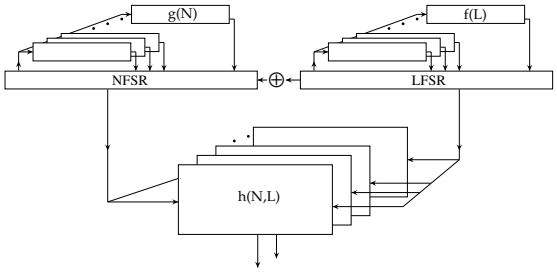

Figure 3: Up to 16x implementation of Grain v1

2.3.1 Unrolling in RTL[4]

Stream ciphers like Grain v1, Grain 128 and Trivium were specifically designed to easily allow unrolling. In Grain v1 for example, the last 16 bit positions in both the linear and non-linear register are used neither in the round update function nor the output keystream function. This implies that a 16x unrolling of Grain v1 is straightforward [HJM07], and only requires 16 additional copies of the round and update functions to be added to the circuit as shown in Figure 3. The same is true for Grain 128 (up to 32x unrolling) and Trivium (up to 64x unrolling).

For degrees of unrolling higher than that specified in the design, the algebraic structure of the resulting round update function gets more and more complicated, since simply adding more copies of round functions will no longer lead to correct functionality. In RTL however, unrolling beyond this specified limit is not very difficult to realize, and an example of this is shown in Appendix A.

In Table 4, we list the simulation results for energy consumptions for different degrees of unrolling. We use scan based flip-flops to construct the memory element and use functional optimization of the round function circuit as motivated before. Next, we discuss several aspects of the simulation and observations in details.

Comparison with block ciphers: We compare our results for stream ciphers with the block ciphers `PRESENT` and `Midori64`. `PRESENT` has been included as a standard in ISO/IEC 29192-2 and was shown in [BBR15] to be extremely energy efficient, while the `Midori` block cipher family was designed specifically for low energy consumption. Although a subspace attack [GJN+16, TLS16] that exploits a class of weak keys of `Midori64` has been reported, we keep the cipher in our comparisons as it sheds some light on lower energy limits achievable with block ciphers.

As opposed to the case of block ciphers, it is difficult to express the energy consumption of an r-round unrolled stream cipher by a simple equation as Equation (1). The reason is that unlike to block ciphers, unrolling a stream cipher by an additional round does not increase the circuit complexity uniformly. As a result the transient signals do not increase uniformly across round functions as in block ciphers, and so it is difficult to algebraically model the energy consumption.

A discussion on modes of operation: Usually for block ciphers, an additional layer of mode of operation is a must before usage, while stream ciphers in general do not require such a layer. For example, encrypting data using the CBC mode requires

[4] In digital circuit design, register-transfer level (RTL) is a design abstraction which models a synchronous digital circuit in terms of the flow of digital signals (data) between hardware registers, and the logical operations performed on those signals.

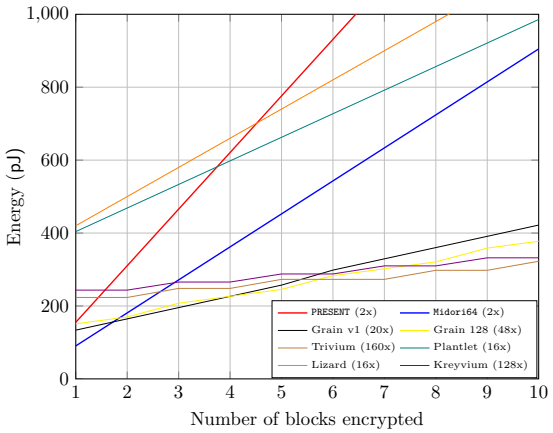

Figure 4: Energy consumptions for up to 10 blocks for most energy-efficient implementations

an additional number of xor gates equal to the blocksize of the block cipher. And encrypting data using the CTR mode requires an additional counter. It is clear that additional hardware amounts to additional power consumption and hence additional energy requirement. In Table 4, we tabulate the energy required for encrypting data in ECB mode for `PRESENT` and `Midori64`. Which is to say, the tabulated values reflect the energy consumed in the block cipher circuit only. Hence, using the above ciphers combined with a mode of operation will therefore consume some more energy than the values tabulated in the table. From Table 4 and Figure 4, our findings are that a suitably unrolled version of Trivium or Grain v1 consumes energy much less than most energy efficient stand-alone block cipher, as we increase the total amount of encrypted data. Therefore we conclude that Trivium or Grain v1 would perform better than a block cipher combined with a mode of operation.

Shorter vs. longer data lengths: Note that while for encrypting a single block of data, block cipher outperform stream ciphers, the opposite is true for larger data. For shorter lengths of data, the energy consumed by the stream cipher is dominated by the key initialization phase. For example, the 1x implementation of Trivium would take 1217 clock cycles to encrypt 64 bits, of which 1152 is used up by the key initialization function. A one round implementation of `Midori64` would take only 17 cycles to encrypt 64 bits. For longer data, the effect of key initialization on the energy consumption becomes less significant, since it is computed only once. To encrypt 1000 blocks (64000 bits) of data, Trivium 1x would require only $64000 + 1152 + 1 = 65153$ cycles. Clearly 1152 is a much smaller fraction of 65153 than of 1217. Multiple unrolling decreases the time to encrypt even further. For example, the 160x implementation of Trivium can encrypt 160 bits in a single clock cycle, and so around $1 + \lceil \frac{64000+1152}{160} \rceil = 409$ cycles are required for 1000 blocks. On the other hand, the most energy-efficient version of `Midori` (2x) would take $9 * 1000 = 9000$ cycles to encrypt 1000 blocks. As a result we see that for the most energy-efficient configuration of Trivium (160x) is around 9 times more energy efficient than the most energy efficient version of `Midori64`. In Figure 4 we plot the energy consumptions for encrypting up to 10 blocks of data with the most energy efficient unrolled configurations of the ciphers. While for a single block of data `Midori64` performs best, for 6 blocks of data or more Trivium performs best.

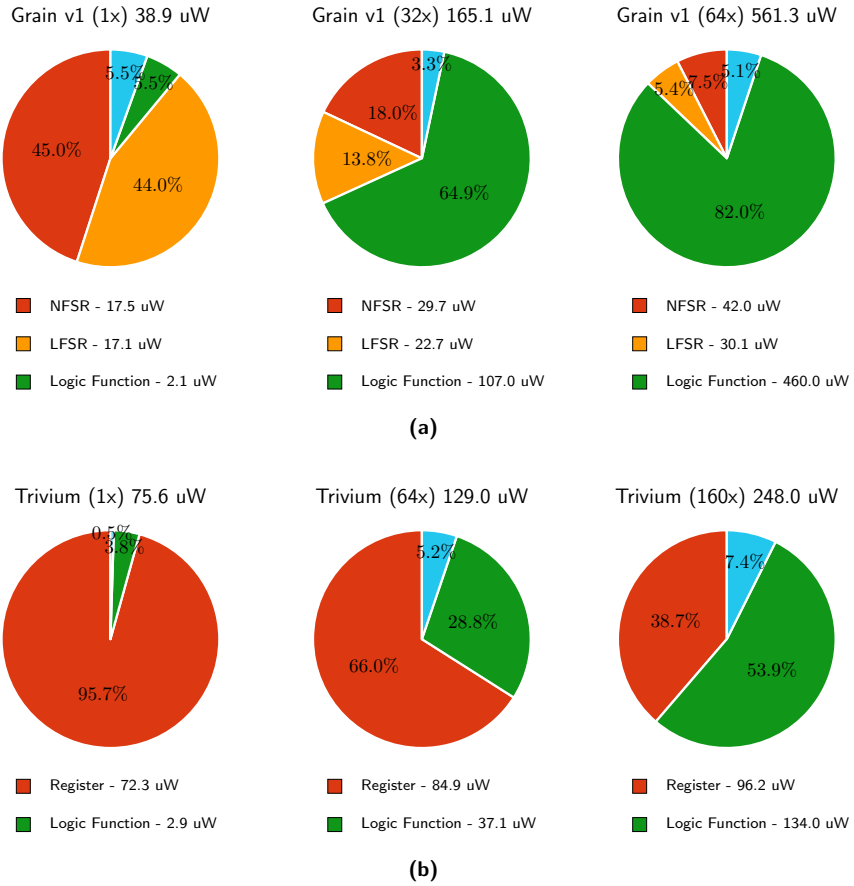

Figure 5: Power consumption shares of Grain v1 for 1, 32, 64 degrees of unrolling, Trivium for 1, 64, 160 degrees of unrolling (the sectors in blue denote power consumed in counters, and other control logic)

Parabolic behavior with unrolling With respect to unrolling, the energy consumption for stream ciphers follows the same parabolic behavior as block ciphers [BBR15], particularly for longer lengths of data. Which is to say that for smaller degrees of unrolling the energy consumption is very high, the energy consumption comes to a minimum at some fixed degree of unrolling, and the energy consumption increases again if the cipher is unrolled beyond this point. This is the result of two conflicting effects. For lower degrees of unrolling, the energy consumption is obviously high due to **1)** a comparatively large number of initialization rounds and **2)** a lower number of bits encrypted per clock cycle. For example, a single round unrolled version of Grain v1 encrypts one bit of plaintext per clock cycle. This means that to encrypt 32 bits, the design has to pay for the energy consumption of the 160-bit register and the associated logic functions for 1 (key loading) + 160 (initialization) + 32 (keystream) = 193 clock cycles.

Consequently, the energy consumption decreases when the degree of unrolling increases. Larger degree of unrolling implies less time spent in initialization and more bits encrypted per cycle. For example, a 32x unrolled version of Grain v1, would need only 5 clock cycles for initialization. To encrypt 32 bits of data, the system would have to pay for the energy consumption of the 160-bit register and logic functions for

$1 + 5 + 1 = 7$ clock cycles. However, the logic functions in a 32x unrolled version are more than 32 times more complex than in a 1x design, and it is true that more power is consumed in the hardware circuit of the logic functions. Despite of that, we can see that a 32x implementation of Grain v1 is around 6.5 to 7 times more energy efficient than the 1x version for short data lengths, and around 7.5 times better for longer data lengths.

However, beyond a certain degree of unrolling, increasing the unrolling results into an increase of energy consumption. The reason for that is the power consumed in the logic functions increases sharply at that point. This happens due to the reasons which are similar for block ciphers [BBI+15]. In [BBI+15, Figure 2], it was shown that power consumption in sequentially placed logic functions increases uniformly because of increased circuit latency which leads to increased glitch propagation. Because of this, it was shown that each additional unrolled round results in quadratic increase in power consumption (due to the term $ar^2 + br + c$ in Eq (1)), but only a linear decrease in the computation time (due to $\left(1 + \lceil \frac{R}{r} \rceil\right)$). As a result, unrolling the round functions beyond a fixed number usually proves counter-productive. Figure 5, demonstrates the increasing share of power consumed by the logic functions in Grain v1 over 1, 32 and 64 degrees of unrolling. It is easy to see that at 64x, the most power hungry element of the design is the round function.

Light/heavy round function A further conclusion one can draw from Table 4 is that the "lightness" of the round functions in stream ciphers has a similar effect as in the case of block ciphers. It was shown in [BBR15] that block ciphers with light round functions like `PRESENT, Twine, SIMON` produce less glitches when the circuits for more than one round function are connected serially. Hence, block ciphers with light round functions achieve energy optimality when unrolled twice, in contrast with heavy round functions whose single round versions are most energy efficient. In Table 4, it can be seen that for ciphers like Grain v1, Lizard and Plantlet whose update functions are more algebraically complex, the energy optimality is achieved at small degrees of unrolling. In contrast, it holds for Trivium which has an extremely simple round update function consisting of 3 and gates and 6 xor gates only that energy optimality is achieved at 160x unrolling.

There is also a distinct advantage for unrolled stream ciphers with simple update functions. This is due to reasons similar as in block ciphers. Simpler/lighter round functions themselves produce less glitches, and thus even when the circuits for these functions are unrolled several times, the propagation of glitches across circuits is not significant enough to escalate the power consumed. Heavier round functions produce more glitches, and their propagation becomes significant even for smaller degrees of unrolling. Figures 5a,b provide a useful comparison between Grain v1 (heavy) and Trivium (light) round functions. At 160x unrolling, the round function in Trivium consumes only 134 uW which is only around 54% of the total power. This is in contrast with the 64x Grain v1 implementation, which consumes around 460 uW, which is 82 % of the total power.

Comparison with Kreyvium Since Kreyvium builds upon the Trivium structure by adding two additional registers for key and IV rotation and two additional xor gates, we can see that an 1x unrolled version of Kreyvium consumes 1.5 to 2 times more energy as Trivium – even for longer data lengths. This trend can be seen for higher degrees of unrolling except for implementations where the number of unrolled rounds is a multiple of 128. These versions do not need additional registers to implement key and IV rotation since they can be assumed to be available on the wires, and hence these implementations have lower energy consumption. Nonetheless, the additional

Table 4: Comparison of energy for different degrees of unrolling, r denotes # unrolled rounds, Energy/bit figure calculated over 1000 blocks.

#	Cipher	r	Area (GE)	Power (uW) @ 10 MHz	Energy (pJ) 1 block	Energy (nJ) 1000 Blocks	Energy/bit (pJ)
1	Grain v1	1	1005	38.9	874.8	249.47	3.90
		16	2673	86.6	129.9	34.73	0.54
		20	2888	102.9	133.8	33.02	0.52
		24	3293	129.4	142.3	34.61	0.54
		28	3711	156.5	140.8	35.88	0.56
		32	3934	165.1	132.1	33.12	0.52
		48	5751	343.1	205.9	45.91	0.72
		64	7474	561.3	280.7	56.30	0.88
2	Grain-128	1	1455	57.8	1855.4	371.41	5.80
		32	3579	126.8	139.4	25.47	0.40
		40	4178	158.1	142.3	25.42	0.40
		48	4749	188.8	151.0	25.29	0.40
		56	5321	235.2	164.6	27.02	0.42
		64	6336	282.7	169.6	28.41	0.44
		80	7078	407.7	203.7	32.81	0.51
3	Trivium	1	1870	78.4	9527.6	510.48	7.97
		64	3051	128.7	257.4	13.11	0.20
		80	3457	148.1	251.7	12.08	0.19
		96	3839	169.4	237.1	11.51	0.18
		112	4241	189.3	227.1	11.04	0.17
		128	4593	207.1	227.8	10.56	0.17
		160	5409	248.2	223.4	10.15	0.16
		192	6179	306.2	244.9	10.44	0.16
		256	7755	419.5	251.7	10.73	0.17
		288	8584	490.0	294.0	11.17	0.17
4	Plantlet	1	785	34.4	1324.6	221.30	3.46
		8	1630	88.5	433.7	71.15	1.11
		16	2254	161.6	404.0	64.98	1.02
		32	3451	651.5	847.0	131.02	2.05
5	Lizard	1	1360	50.4	1617.5	323.78	5.06
		8	2565	101.7	417.0	81.70	1.28
		16	3954	200.0	420.0	80.34	1.26
		32	6778	672.4	739.6	135.09	2.11
6	Kreyvium	1	2892	140.8	17135.4	917.35	14.33
		64	4579	202.8	405.6	20.66	0.32
		80	5045	224.0	380.8	18.28	0.29
		96	5480	248.8	348.3	16.92	0.26
		112	5939	273.1	327.7	15.92	0.25
		128	5050	221.4	243.5	11.29	0.18
		160	7268	364.7	328.2	14.92	0.23
		192	8149	430.7	344.6	14.69	0.23
		256	8612	452.6	271.6	11.59	0.18
		288	10836	696.1	417.7	15.87	0.25
7	`PRESENT`	1	1440	52.2	172.3	172.3	2.69
		2	1968	91.3	155.2	155.2	2.43
		3	2500	149.0	178.8	178.8	2.79
8	`Midori64`	1	1542	60.6	103.0	103.0	1.61
		2	2017	100.6	90.5	90.5	1.41
		3	2826	273.8	191.7	191.7	3.00

complexity of 2 XOR gates in the round function implies that even for multiples of 128, the most energy-efficient configuration of Kreyvium consumes around 10% more energy than Trivium.

2.4 Lessons learnt

From the discussion in this section, it becomes clear for encrypting longer data streams, stream ciphers with a **simple** update functions have a distinct advantage. These are easier and more energy-efficient to unroll for **higher degrees of unrolling**. Higher degrees of unrolling allows to encrypt more bits in one clock cycle, which is crucial in bringing down the number of clock cycles required to encrypt a given length of data, and hence the energy consumption. On the other hand, higher degree of unrolling results into a more complex logic of the update function and hence needs more power to operate. Thus, a sufficiently simple update function ensures that the additional power consumption resulting from unrolling remains small enough to not outweigh the natural advantages obtained from unrolling. Lastly, the number of initialization rounds does affect the energy numbers for shorter data packets, but its effect becomes minimal with the increase in the length of plaintext to be encrypted.

3 Conclusion

In this paper, we investigated the design of low energy ciphers. We conducted experiments on various design parameters that affect the energy consumption of the encryption process and were able to draw several conclusions out of it. Our initial investigations showed that although block ciphers are more energy-efficient solutions for encryption of short data streams, for longer data streams multiple round unrolled stream ciphers perform better. Stream ciphers with simple update functions were found to be more energy-efficient since these were easy to unroll without increasing the circuit complexity and power consumption too much.

We found that the Trivium structure was best suited for this purpose. The 160x unrolled implementation of Trivium was not only around 9 times better than the best block cipher based solution in terms of energy consumption of 1000 data blocks.

Acknowledgements

Subhadeep Banik was supported by Commission for Technology and Innovation (Confédération Suisse) grant no CTI 19339.1. Takanori Isobe was supported in part by Grant-in-Aid for Young Scientist (B) (KAKENHI 17K12698) for Japan Society for the Promotion of Science.

References

[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 - 21st International Conference on the Theory and Application of Cryptology and Information Security, Auckland, New Zealand, November 29 - December 3, 2015, Proceedings, Part II*, volume 9453 of *Lecture Notes in Computer Science*, pages 411–436. Springer, 2015.

[BBR15] Subhadeep Banik, Andrey Bogdanov, and Francesco Regazzoni. Exploring energy efficiency of lightweight block ciphers. 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*, volume 9566 of *Lecture Notes in Computer Science*, pages 178–194. Springer, 2015.

[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 Yalçin. PRINCE - A low-latency block cipher for pervasive computing applications - extended abstract. 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*, volume 7658 of *Lecture Notes in Computer Science*, pages 208–225. Springer, 2012.

[BDE+13] Lejla Batina, Amitabh Das, Baris Ege, Elif Bilge Kavun, Nele Mentens, Christof Paar, Ingrid Verbauwhede, and Tolga Yalçin. Dietary recommendations for lightweight block ciphers: Power, energy and area analysis of recently developed architectures. In Michael Hutter and Jörn-Marc Schmidt, editors, *Radio Frequency Identification - Security and Privacy Issues 9th International Workshop, RFIDsec 2013, Graz, Austria, July 9-11, 2013, Revised Selected Papers*, volume 8262 of *Lecture Notes in Computer Science*, pages 103–112. Springer, 2013.

[BKL+07] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. 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*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer, 2007.

[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. *IACR Cryptology ePrint Archive*, 2013:404, 2013.

[CCF+16] Anne Canteaut, Sergiu Carpov, Caroline Fontaine, Tancrède Lepoint, María Naya-Plasencia, Pascal Paillier, and Renaud Sirdey. Stream ciphers: A practical solution for efficient homomorphic-ciphertext compression. In Thomas Peyrin, editor, *Fast Software Encryption - 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers*, volume 9783 of *Lecture Notes in Computer Science*, pages 313–333. Springer, 2016.

[CDK09] Christophe De Cannière, Orr Dunkelman, and Miroslav Knezevic. 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*, volume 5747 of *Lecture Notes in Computer Science*, pages 272–288. Springer, 2009.

[CMM14] Abhishek Chakraborty, Bodhisatwa Mazumdar, and Debdeep Mukhopadhyay. Fibonacci LFSR vs. galois LFSR: which is more vulnerable to power attacks? In Rajat Subhra Chakraborty, Vashek Matyas, and Patrick Schaumont, editors, *Security, Privacy, and Applied Cryptography Engineering - 4th International Conference, SPACE 2014, Pune, India, October 18-22, 2014. Proceedings*,

volume 8804 of *Lecture Notes in Computer Science*, pages 14–27. Springer, 2014.

[CP08] Christophe De Cannière and Bart Preneel. Trivium. In Matthew J. B. Robshaw and Olivier Billet, editors, *New Stream Cipher Designs - The eSTREAM Finalists*, volume 4986 of *Lecture Notes in Computer Science*, pages 244–266. Springer, 2008.

[DR02] Joan Daemen and Vincent Rijmen. *The Design of Rijndael: AES - The Advanced Encryption Standard*. Springer Verlag, Berlin, Heidelberg, New York, 2002.

[Dub09] Elena Dubrova. A transformation from the Fibonacci to the Galois NLFSRs. *IEEE Trans. Information Theory*, 55(11):5263–5271, 2009.

[FWR05] M. Feldhofer, J. Wolkerstorfer, and V. Rijmen. AES implementation on a grain of sand. *IEE Proceedings - Information Security*, 152(1):13–20, Oct 2005.

[GJN+16] Jian Guo, Jérémy Jean, Ivica Nikolic, Kexin Qiao, Yu Sasaki, and Siang Meng Sim. Invariant Subspace Attack Against Midori64 and The Resistance Criteria for S-box Designs. *IACR Trans. Symmetric Cryptol.*, 2016(1):33–56, 2016.

[GNL11] Zheng Gong, Svetla Nikova, and Yee Wei Law. KLEIN: A new family of lightweight block ciphers. In Ari Juels and Christof Paar, editors, *RFID. Security and Privacy - 7th International Workshop, RFIDSec 2011, Amherst, USA, June 26-28, 2011, Revised Selected Papers*, volume 7055 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2011.

[GPPR11] Jian Guo, Thomas Peyrin, Axel Poschmann, and Matthew J. B. Robshaw. The LED block cipher. In Preneel and Takagi [PT11], pages 326–341.

[HJM07] Martin Hell, Thomas Johansson, and Willi Meier. Grain: a stream cipher for constrained environments. *IJWMC*, 2(1):86–93, 2007.

[HJMM06] M. Hell, T. Johansson, A. Maximov, and W. Meier. A Stream Cipher Proposal: Grain-128. In *2006 IEEE International Symposium on Information Theory*, pages 1614–1618, July 2006.

[HKM17] Matthias Hamann, Matthias Krause, and Willi Meier. LIZARD - A Lightweight Stream Cipher for Power-constrained Devices. *IACR Trans. Symmetric Cryptol.*, 2017(1):45–79, 2017.

[KDH+12] Stéphanie Kerckhof, François Durvaux, Cédric Hocquet, David Bol, and François-Xavier Standaert. Towards green cryptography: A comparison of lightweight ciphers from the energy viewpoint. In Emmanuel Prouff and Patrick Schaumont, editors, *Cryptographic Hardware and Embedded Systems - CHES 2012 - 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings*, volume 7428 of *Lecture Notes in Computer Science*, pages 390–407. Springer, 2012.

[MAM16] Vasily Mikhalev, Frederik Armknecht, and Christian Müller. On Ciphers that Continuously Access the Non-Volatile Key. *IACR Trans. Symmetric Cryptol.*, 2016(2):52–79, 2016.

[MD10] Shohreh Sharif Mansouri and Elena Dubrova. An improved hardware implementation of the grain stream cipher. In Sebastián López, editor, *13th Euromicro Conference on Digital System Design, Architectures, Methods and Tools, DSD 2010, 1-3 September 2010, Lille, France*, pages 433–440. IEEE Computer Society, 2010.

[MPL+11] Amir Moradi, Axel Poschmann, San Ling, Christof Paar, and Huaxiong Wang. Pushing the limits: A very compact and a threshold implementation of AES. In Kenneth G. Paterson, editor, *Advances in Cryptology - EUROCRYPT 2011 - 30th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tallinn, Estonia, May 15-19, 2011. Proceedings*, volume 6632 of *Lecture Notes in Computer Science*, pages 69–88. Springer, 2011.

[PT11] Bart Preneel and Tsuyoshi Takagi, editors. *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 - October 1, 2011. Proceedings*, volume 6917 of *Lecture Notes in Computer Science*. Springer, 2011.

[SIH+11] Kyoji Shibutani, Takanori Isobe, Harunaga Hiwatari, Atsushi Mitsuda, Toru Akishita, and Taizo Shirai. Piccolo: An ultra-lightweight blockcipher. In Preneel and Takagi [PT11], pages 342–357.

[SMTM01] Akashi Satoh, Sumio Morioka, Kohji Takano, and Seiji Munetoh. A compact rijndael hardware architecture with s-box optimization. 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*, volume 2248 of *Lecture Notes in Computer Science*, pages 239–254. Springer, 2001.

[SSA+07] Taizo Shirai, Kyoji Shibutani, Toru Akishita, Shiho Moriai, and Tetsu Iwata. The 128-bit blockcipher CLEFIA (extended abstract). In Alex Biryukov, editor, *Fast Software Encryption, 14th International Workshop, FSE 2007, Luxembourg, Luxembourg, March 26-28, 2007, Revised Selected Papers*, volume 4593 of *Lecture Notes in Computer Science*, pages 181–195. Springer, 2007.

[TLS16] Yosuke Todo, Gregor Leander, and Yu Sasaki. Nonlinear invariant attack - practical attack on full scream, iscream, and midori64. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part II*, volume 10032 of *Lecture Notes in Computer Science*, pages 3–33, 2016.

Appendices

A Unrolling in RTL

The VHDL code snippet in Listing 1 shows an implementation of the round function of Grain 128 with degree of unrolling equal to any integer `mul`. Note that `NxDI, LxDI` are 128 bit signals denoting the current round NFSR and LFSR state bits respectively. We define the two sets of wires `IxD, JxD` of width `128+mul` each. For $0 \leq i \leq 127$, `IxD(i), JxD(i)` will simply carry the NFSR, LFSR signals. Thereafter we functionally define `IxD(128+i), JxD(128+i)` as the i^{th} update of the NFSR and LFSR respectively (for $0 \leq i \leq$`mul-1`). It is easy to see that `IxD(128 to 128+mul-1), JxD(128 to 128+mul-1)` are the update values of the NFSR, LFSR respectively in the next clock cycle. The input signal `KSAxSI` is HIGH during the initialization process, and so the `ZUxD` is the output signal fed back during this period. Finally, `OupxD` is the signal denoting the output keystream. The code can be used to achieve any multiplicity of unrolling and can easily be extended to other Grain or Trivium-like ciphers.

Listing 1: Unrolling Grain 128

```
IxD(0 to 127) <= NxDI;
JxD(0 to 127) <= LxDI;

a1: for i in 0 to mul-1 generate

      JxD(128+i) <= JxD(i) xor JxD(i+7) xor JxD(i+38) xor JxD(i+70) xor
                    JxD(i+81) xor JxD(i+96)  xor ZUxD(i);

      IxD(128+i) <= IxD(i+26) xor IxD(i+56) xor IxD(i+91) xor IxD(i+96) xor
                    IxD(i) xor (IxD(i+3) and IxD(i+67)) xor (IxD(i+11) and IxD(i+13)) xor
                    (IxD(i+17) and IxD(i+18)) xor (IxD(i+27) and IxD(i+59)) xor
                    (IxD(i+40) and IxD(i+48)) xor (IxD(i+61) and IxD(i+65)) xor
                    (IxD(i+68) and IxD(i+84)) xor JxD(i) xor ZUxD(i);

      OupxD(i)   <= IxD(i+2) xor  IxD(i+15) xor  IxD(i+36) xor IxD(i+45) xor IxD(i+64) xor
                    IxD(i+73) xor  IxD(i+89) xor JxD(i+93) xor (IxD(i+12) and JxD(i+8)) xor
                    (JxD(i+13) and JxD(i+20)) xor (IxD(i+95) and JxD(i+42)) xor
                    (JxD(i+60) and JxD(i+79)) xor (IxD(i+12) and IxD(i+95) and JxD(i+95));

      ZUxD(i)    <= OupxD(i) and KSAxSI(i);

end generate a1;

LopxDO <= JxD(128 to 128+mul-1);
NopxDO <= IxD(128 to 128+mul-1);
OupxDO <= OupxD;
```

For Trivium-like ciphers in which the 3 registers are mutually interdependent on each other, the unrolling follows the same principle. We present a VHDL code snippet in Listing 2, that shows an implementation of the Trivium round function with degree of unrolling `mul`. The signal `NxDI` represents the 288 bits of the current state. We define. similarly as above three sets of wires `R1xD, R2xD, R3xD`, so that `R1xD(mul to 92+mul), R2xD(mul to 83+mul), R3xD(mul to 110+mul)` would simply carry the `NxDI` signal. The concatenation of the bits `R1xD(0 to 92), R2xD(0 to 83), R3xD(0 to 110)` represents the updated state bits for the next clock cycle.

Listing 2: Unrolling Trivium

```
R1xD(mul to 92+mul) <= NxDI(0 to 92);

R2xD(mul to 83+mul) <= NxDI(93 to 176);

R3xD(mul to 110+mul) <= NxDI(177 to 287);



a1: for i in 0 to mul-1 generate

       A1xD(i) <= (R1xD(65 + mul - i)          xor  R1xD(92+ mul -i));
       A2xD(i) <= (R2xD(161 + (mul-93)  -i)    xor  R2xD(176 + (mul-93) -i));
       A3xD(i) <= (R3xD(242 + (mul-177) -i)    xor  R3xD(287 + (mul-177) -i));


       R2xD(mul - i-1) <= (R1xD(90+  mul -i  )       and R1xD(91+ mul -i )        )   xor
                          A1xD(i)  xor  R2xD(170 + (mul-93) -i );

       R3xD(mul - i-1) <= (R2xD(174 + (mul-93) -i )  and R2xD(175 + (mul-93) -i ) )   xor
                          A2xD(i)  xor  R3xD(263+ (mul-177) -i );

       R1xD(mul - i-1) <= (R3xD(285 + (mul-177) -i ) and R3xD(286 + (mul-177) -i) )   xor
                          A3xD(i)  xor  R1xD(68+ mul - i  );


       ZxDO(i) <= A1xD(i) xor A2xD(i) xor A3xD(i);

end generate a1;

NextxDO <= R1xD(0 to 92) & R2xD(0 to 83) & R3xD(0 to 110);
```