Comparison of Lightweight Stream Ciphers: MICKEY 2.0, WG-8, Grain and Trivium

Lennart Diedrich[1], Patrick Jattke[1], Lulzim Murati[1], Matthias Senker and Alexander Wiesmaier[1,2,3]

[1] Technische Universität Darmstadt, Germany
[2] AGT International, Germany
[3] Hochschule Darmstadt, Germany

Abstract. In this paper we compare four lightweight stream ciphers, MICKEY 2.0, WG-8, Grain and Trivium, which can be used for data encryption in the Internet of Things. At first we provide a brief overview of each algorithm. This includes their basic functional principle, their different specifications, possible hardware implementations and their complexity. We conclude the overview with a short security evaluation. Finally, we provide a comparison of the algorithms pointing out differences, similarities and their applicability in the Internet of Things.

Keywords: MICKEY 2.0, WG-8, Grain, Trivium, comparison, lightweight cryptography, lightweight data encryption, internet of things, IoT

1 Introduction

The Internet of Things (IoT) describes a paradigm of networked interconnection, consisting of objects in our everyday life. Mostly, the devices also have an integrated intelligence, which enables the communication with other IoT devices or even with humans. The main building blocks of the IoT, besides of the backend web services, are sensors and actuators. Sensors are used to collect data, for example of the environment (temperature, humidity, traffic etc.) or the personal fitness and health (i.e. pedometer, sleep quality tracker). On the other side there are actuators, which perform an action if they are triggered. Common examples for actuators are (wireless) lights and thermostats. Moreover, there exist devices which act as both, sensor and actuator. For example, wall sockets which sense the energy consumption of the connected device and can also be remotely controlled.

Sensors can collect massive amount of information while they are powered-on. Because the storage space and computing power of sensors are limited, they sent the data to servers for capturing and further processing. As the collected information may contain sensitive data, it is essential to protect the confidentiality with appropriate measures.

1.1 Stream Ciphers

Encryption is used to meet the requirements on confidentiality and privacy of data transmission. The wide-ranging field of cryptographic algorithms for encryption can be divided into symmetric key-based methods, where sender and receiver use the same cryptographic key, and asymmetric key-based methods, where sender and receiver require distinct keys. Moreover, algorithms can be differentiated in block ciphers and stream ciphers, whereas the former operates on a fixed-length of bits, the latter encrypts bits individually.

Although the need of stream ciphers has been questioned, because block ciphers can easily be turned into stream ciphers by using different operation modes (OFB, CFB, CTR and OCB) [83], they have their right to exist. Major advantages are their high throughput and their low complexity, leading to low hardware implementation costs. Besides of that, they do not need much memory and are often used if the buffer size is limited. Therefore, they are ideally suited for the use in IoT devices.

There are two types of stream ciphers, distinguishable by the manner of their key stream derivation. The so-called synchronous stream ciphers generate a key stream only depending on the input key, thus being independent of the plain text and the cipher text. While self-synchronizing stream ciphers, also termed as asynchronous stream ciphers, generate the key stream based on the input key and a fixed number of previously produced cipher text digits. [87]

Design and Functional Principle As mentioned before, stream ciphers act on a per bit basis, similar to a one-time pad (OTP). A crucial drawback of an OTP is the need of a random key with length as long as the given plain text. So for example a 500 MByte file needs a key with length at least of 4 GBit. Therefore this requirement is in a practical point of view hard to fulfill and rises issues in key distribution and management. These drawbacks have been considered by the design of stream ciphers, though the basic design principle is similar. [87]

Stream ciphers basically use two input data streams: An input data stream which contains the text to be encrypted and a key data stream for encryption/decryption of the input data. The key data stream is generated by a function taking a seed as input. This seed is referred to as encryption key. Instead of providing an encryption key with length at least as the plain text, as done by OTP, a smaller secret key is sufficient. This key is then used to produce a pseudorandom key stream. Considerable, the key stream must be deterministic, in the sense of the same encryption key always produces the same key stream. [87,98]

Encryption and Decryption The encryption and decryption function of stream ciphers are based on simple addition and modulo 2, which is basically the XOR operation. In [98] the operations are described as follows:

Let x_i be the i-th bit of the plain text and s_i be the i-th bit of the key stream. Then the encrypted output bit y_i is defined as:

$$y_i = (x_i + s_i \mod 2) \equiv x_i \oplus s_i$$

The decryption works similar. Let y_i be the i-th bit of the cipher text and s_i be the i-th bit of the key stream. Then the i-th bit of plain text x_i is given by:

$$x_i = (y_i + s_i \mod 2) \equiv y_i \oplus s_i$$

Attacks Stream ciphers can be exploited by different type of attacks, all targeting to discover the key used for encryption and decryption. In the following we present a brief overview of the most common attacks against stream ciphers, based on the survey [12] by Banegas.

Exhaustive Search Attack This attacks searches exhaustively through all possible states, seeking a match between the resulting and the observed key stream. In literature it is often referred to as *brute force attack*. Babbage [6] suggested in 1995 two attacks improving the exhaustive search on stream ciphers.

Algebraic Attack The algebraic attack can be applied to stream ciphers based on linear feedback shift registers (LFSR). The goal of this attack is to exploit the linearity to find the secret key. The attack consists of two phases, in the first phase a system of equations is build of the secret key bits and the output bits. This system can be solved in step two, if it consists of enough low degree equations and enough known key stream bits. As an result of the attack, the secret key is recovered.

Correlation Attack The correlation attack was first proposed in 1985 by Siegenthaler [107]. It uses the existence of statistical dependencies between the key stream and the output of a single constituent LFSR to reveal the secret key. This is possible if the encryption output is statistically biased by specific internal state bits which are used as input [109].

Fault Attack The fault attack can be used for exploiting systems not being directly vulnerable. It assumes that the attacker has physical access to the device and can perform bit flipping faults on the memory or on internal registers. The goal of the technique is to stress the device with external means and produce an error leading to a security failure. This method has first been applied on stream ciphers by Hoch and Shamir [68].

Distinguishing Attack The distinguishing attack relates to the distinction between the output of a specific cipher and truly random data. This distinguishing attack reveals information to be used to draw conclusions of the secret key. If an adversary cannot make this distinction, the cipher is considered as secure. Rose and Hawkes [104] analyzed the applicability of distinguishing attacks against stream ciphers. They demonstrated that even if a distinguishing attack can be performed successful, the resulting security impact for practice may not be relevant if the requirements of the attack are too high.

Chosen-IV Attack One common example for the chosen-IV attack was shown on the stream cipher Turing by Joux and Muller [73]. In general, the attack exploits weaknesses in the key scheduling algorithm of the cipher by using statistical techniques. This is achieved by generating different initialization vectors without changing the secret key. The goal of the attack is to extract information from memory about the LFSR's initial state. The extracted information can be used to create a system of equations which can then be solved to recover the key.

Slide Attack The slide attack was first proposed in [28] for block ciphers and later adapted to stream ciphers, like Trivium [101]. It exploits the key scheduling of the cipher. A slide attack generally requires two instances of an encryption process and a single function F describing the operation of the cipher for each round. According to [28], the main idea is to slide one copy of the encryption process against the other one, such that the processes are one round out of phase. For this, $2^{n/2}$ pairs of (P, C) are collected, where P denotes the plain text and C the related cipher text. If a so-called slid pair $(P_0, C_0), (P_1, C_1)$ with the property $P_0 = F(P_1)$ and $C_0 = F(C_1)$ is found, the cipher is vulnerable to a known-plaintext attack and therefore broken.

Cube Attack The cube attack counts to the newer attacks, first introduced by Dinur and Shamir [42] in 2009, using the example of Trivium. Later, the authors adressed the applicability on stream ciphers [45]. Generally, a cube attack can be applied to any block cipher, stream cipher, or MAC, without requiring details of the internal structure. For the attack the cryptosystem scheme is considered as *tweakable polynomials* over $GF(2)$, consisting of a secret part (e.g, secret key bits) and a public part (e.g., plain text bits, IV bits). Therefore each output bit must be representable by an unknown polynomial of relatively low degree with these two parts. The authors identified relations of the polynomial equations of the cryptographic schemes, which are typically derived from a single *master polynomial.* The idea is to exploit the relation between the polynomial equations by modifying the values of the tweakable public bits, resulting in derived polynomial equations. These can be used to eliminate the nonlinear terms. The remaining linear equations can then easily be solved to reveal the secret key.

Time-Memory Trade-off Attack The Time-Memory Trade-off (TMD) Attack is a cryptanalytic technique proposed by Hellmann [66] in 1980. Initially, the attack was introduced for block ciphers. Later, the applicability of the TMD attack on stream ciphers was shown by Biryukov and Adi [26]. Considered aspects within the attack are the possible keys N, the time T and the memory M. These are related by the tradeoff curve $TM^2 = N^2$ for $1 \leq T \leq N$. The attack consists of two phases [26]: The first phase, the preprocessing phase, explores the cryptosystem and summarizes the findings in large tables. This phase collects information not related to any specific key and can take, depending on the complexity, very long time. In the realtime phase the attacker uses the previously computed tables to find the key to given data produced from an unknown key.

Guess and Determine Attack This attack was first used by Pasalic [99] on LFSR-based stream ciphers. During the attack the adversary first guesses a set of state elements, then the remaining state elements and the running key sequence must be determined. The comparison of the resulting key sequence and the observed key sequence reveals if the guess was correct. If it was correct, the system is broken, otherwise the procedure must be repeated with other values. The attack was improved in [2] by using a heuristic instead of random guessing.

1.2 Established Stream Ciphers

In this section we provide a concise overview of well established stream ciphers used in practical applications.

As one of the first widely used stream ciphers **RC4**[4] can be considered. It was invented in 1987 by Ronald L. Rivest and is based on a secret internal state to generate the key stream. It basically consists of a substitution box combined with random permutations [80]. The cipher was used in many protocols, like SSL, SSHv1, RDP and WEP. Initially the algorithm was kept secret, but an anonymous user released its source code in 1994. As as consequence, different weaknesses of the algorithm were found, for that reason the cipher can be seen as broken [3]. Moreover, since February 2015 the use of RC4 within TLS is prohibited by RFC 7465 [100].

Another stream cipher whose code was initially kept secret is **A5/1**. It was developed in 1987 for data encryption between mobile device and base station within GSM[5]. It uses three linear feedback shift registers (LFSR) with irregular clocking. The algorithm became public through leaks and reverse engineering. An attack proposed 2000 by Biryukov et al. [27] uses a time-memory tradeoff to achieve real time cryptanalysis.

The stream cipher **E0** was introduced in 1999 within the specification of the Bluetooth system v1.0 B [30] and is based on four linear feedback shift registers of differing lengths. Improved attacks from 2005 require only 2^{38} computations by using the first 24-bits of $2^{23.8}$ frames [67].

Crypto1 is a proprietary stream cipher created by NXP Semiconductors specially for RFID tags, such as the Mifare RFID tag used by different electronic ticketing systems for public transport. It is based on a 48-bit LFSR. In 2008 an attack [81] was proposed using a weakness in the pseudo-random generator, thus allowing to recover the generated key stream.

As mentioned in Section 1.1, block ciphers can easily be turned into stream ciphers, therefore stream ciphers were not of large interest in the past. However, as stream ciphers also have advantages compared to block ciphers, like the high throughput and low complexity, a project called eSTREAM was initiated to promote the design of new stream ciphers for widespread adoption.

1.3 The eSTREAM Project

In 2004, the ECRYPT stream cipher project (eSTREAM) was initiated by a consortium of European research organizations. This project is funded by the European Union and targets the identification of "new stream ciphers suitable for widespread adoption" with focus on efficiency and compactness [46].

The project differentiated between two types of ciphers, software-oriented ciphers (profile 1) and hardware-oriented ciphers (profile 2). The requirements for each type were described in [7] as follows:

[4] Ron's Code 4

[5] Global System for Communications

- **Software-oriented** ciphers should be optimized for implementation in software and significantly outperform AES in a suitable stream cipher mode. Moreover, they should have a security level of at least 128-bit and support IVs of 64-bit and 128-bit. The main focus of this profile is on a high raw encryption speed for large amounts of data after a single initialization.
- **Hardware-oriented** ciphers should be optimized for implementation in highly constrained hardware environments. Concretely, they should significantly outperform AES in a restricted environment in at least one major regard. Besides of that, they should have a security level of 80-bit and support IVs of 32-bit and 64-bit.

The eSTREAM project was organized in 3 phases, each one resulting in a list of ciphers (portfolio) which passed the performance and security evaluations. The first phase, initiated in November 2014, started with a call for submissions and attracted about 34 stream ciphers from cryptographers all around the world, and hundreds of security and performance evaluations [21].

eSTREAM Finalists The final portfolio consists of four ciphers for profile 1 and four ciphers for profile 2. Finalists of profile 1 of the eSTREAM project are HC-128 [112], Rabbit [31], Salsa20/12 [24] and SOSEMANUK [19].

HC-128 uses a secret state, built of two tables, which are updated using a non-linear feedback function. Each step generates a 32-bit output word and updates one register of one of the tables. **Rabbit** is a synchronous stream cipher built of eight 32-bit registers and eight 32-bit counters. Due to the simple arithmetics and operations used, this cipher is among the most efficient ones evaluated within eSTREAM. The **Salsa20/12** cipher is, like Rabbit, built of simple operations: XOR, addition and bit rotation on 32-bit words. There exist variants with less or more rounds, like Salsa20/8 with 8 rounds. As evaluated within eSTREAM, Salsa20/12 has the best trade-off between security and performance. **SOSEMANUK** follows similar design principles as SNOW 2.0 and SERPENT. Its key length is variable between 128 and 256 bits. The cipher consists of a LFSR and a finite state machine, both operating on 32-bit words.

Our considered algorithms were all proposals of eSTREAM profile 2 (HW): **MICKEY**, an earlier specification of the current MICKEY 2.0, was one of the finalist of phase 3. Whereas the predecessor of **WG-8** was archived in phase 2 because of its vulnerability on the initialization which allowed a chosen IV attack, as demonstrated by Wu and Preneel [113,89]. **Grain v1** and **Trivium** are both finalists of phase 3.

We chose these algorithms as they have passed the eSTREAM project successfully and further, they showed promising properties. The authors of MICKEY 2.0 remark its low required area and low energy consumption, making it an ideal candidate for IoT applications. Whereas WG-8 is especially interesting due to its security properties, it is considered as secure against common stream cipher attacks. Moreover, measurements showed a high throughput while preserving a low energy consumption. Grain was chosen because it can be accelerated by additional hardware, making it an ideal candidate for applications where performance is more important than area consumption. Also the easy implementation of Grain is remarkable. Lastly, we chose Trivium due to its simple and efficient structure, and also the fact that it can be easily parallelized.

1.4 Related Work

The area of stream ciphers is heavily researched, with the result that there is a lot of theoretical knowledge on stream ciphers and many designs have been proposed and analyzed. In the past, many algorithms were proprietary and thus not fully-specified in open literature [87]. But in recent years more and more "open" ciphers have been proposed, as seen for example within the eSTREAM project. This fact has driven the research and allowed more extensive evaluations.

Within the eSTREAM project several papers with comparisons and evaluations have been published. However, these publications primarily focus on only one aspect, like performance or security. For example, the paper by Bernstein [23] compares the software performance of several phase-3 eSTREAM ciphers on different platforms. A comparison of Good and Benaissa [61] shows hardware implementations of phase-3 ciphers and their performance. Notably are Trivium and Grain, as they show a good ratio between power consumption, area consumption, and performance.

The work by Feldhofer [53] focuses on low-power implementations of Trivium and Grain. It compares both algorithms regarding its required chip area, the power consumption, and the required number of clock cycles for encrypting a fixed amount of data. The results show that Trivium with 3,090 gate equivalents requires less area than Grain with 3,360 gate equivalents. On contrary, Grain requires 104 clock cycles for encrypting 128 bits of data, whereas Trivium needs 176 cycles. Furthermore, Trivium requires a long initialization phase, about 1,603 cycles. The results show that the energy consumption of Grain (0.80 μA@100kHz) and Trivium (0.68 μA@100kHz) is similar.

Although, for the best of our knowledge there are no extensive comparisons of lightweight stream ciphers available, several surveys on lightweight block ciphers have been published. For example, Eisenbarth et al. [47] compares recent evaluation results of block ciphers, including PRESENT, DES, DESXL, HIGHT, CLEFIA, and mCrypton. The survey considers software and hardware implementations. The survey by Jana, Bhaumik and Maiti [72] compares DESL, DESXL, HIGHT, PRESENT, KTANTAN, KATAN, KLEIN, LED, PRINCE, TWINE, TEA and XTEA, regarding the structure of the ciphers, the area consumption and the best known attack.

As one of the first widely used stream ciphers **RC4**[6] can be considered. It was invented in 1987 by Ronald L. Rivest and is based on a secret internal state

[6] Ron's Code 4

to generate the key stream. It basically consists of a substitution box combined with random permutations [80]. The cipher was used in many protocols, like SSL, SSHv1, RDP and WEP. Initially the algorithm was kept secret, but an anonymous user released its source code in 1994. As as consequence, different weaknesses of the algorithm were found, for that reason the cipher can be seen as broken [3]. Moreover, since February 2015 the use of RC4 within TLS is prohibited by RFC 7465 [100].

Another stream cipher whose code was initially kept secret is **A5/1**. It was developed in 1987 for data encryption between mobile device and base station within GSM [7]. It uses three linear feedback shift registers (LFSR) with irregular clocking. The algorithm became public through leaks and reverse engineering. An attack proposed 2000 by Biryukov et al. [27] uses a time-memory tradeoff to achieve real time cryptanalysis.

The stream cipher **E0** was introduced in 1999 within the specification of the Bluetooth system v1.0 B [30] and is based on four linear feedback shift registers of differing lengths. Improved attacks from 2005 require only 2^{38} computations by using the first 24-bits of $2^{23.8}$ frames [67].

Crypto1 is a proprietary stream cipher created by NXP Semiconductors specially for RFID tags, such as the Mifare RFID tag used by different electronic ticketing systems for public transport. It is based on a 48-bit LFSR. In 2008 an attack [81] was proposed using a weakness in the pseudo-random generator, thus allowing to recover the generated key stream.

Another stream cipher, proposed in 2001 by Johansson and Ekdahl, is called **SNOW** [48]. It is word-oriented, consisting of a LFSR which feeds a finite state machine. The cipher is significantly faster than AES and was used as reference cipher for performance evaluation within the eSTREAM project. Because SNOW has shown weaknesses an improved version called SNOW 2.0 [49] has been proposed. Security evaluations show that SNOW 2.0 is susceptible to distinguishing attacks [96].

An example for a stream cipher designed for efficient hardware implementation is **MUGI** [111]. It was proposed in 2002 and is based on the key stream generator PANAMA [35]. It uses a substitution S-box and linear transformations, both also used in AES. Moreover it consists of a nonlinearly updated component and a linearly updated component, called buffer. An analysis from 2004 showed design weaknesses regarding the buffer, which allows to recover the 128-bit secret key from the reconstructed internal state of MUGI [57].

2 MICKEY

In this section we provide a short overview about the structure and function principle of the different MICKEY specifications.

The stream cipher family MICKEY [11] which is an abbreviation for **M**utual **I**rregular **C**locking **Key**stream generator, has been developed by Babbage and

[7] Global System for Communications

Dodd in 2005 as a candidate for the eSTREAM project. The ciphers are designed with focus on resource-constrained hardware platforms. The implementation in hardware has low complexity, while at the same time providing a high level of security.

All ciphers of the MICKEY family are based on two registers: A linear feedback shift register (LFSR) and a non-linear feedback shift register (NLFSR). The main difference between both is the update function. Whereas a LFSR has a linear state update function, the NLFSR uses a non-linear update function. The main idea was to combine the advantages of both, to achieve a higher security level. Notable as well is the use of irregular clocking of the shift registers, which aims to increase security while keeping the design simple.

Initially MICKEY 1.0 [9] was submitted to the eSTREAM project. A security evaluation by Hong and Kim [69] showed several weaknesses in the design. For that reason some changes were made, resulting in MICKEY 2.0 [10].

We present the cipher MICKEY 2.0 and the variant with longer key size, MICKEY-128 2.0. Next, we point out differences to the original specification MICKEY 1.0 respectively MICKEY-128 1.0 and show different hardware implementations and their complexity. Lastly, we provide an overview about known attacks against MICKEY 2.0 and MICKEY-128 2.0.

2.1 MICKEY 2.0

The cipher MICKEY 2.0 takes as input an 80-bit secret key $K = k_0, ..., k_{79}$ and an initialization variable IV with variable length from 0 to 80 bits. We denote its bits as $IV = iv_0, ..., iv_{IV.length-1}$.

Registers We define $n = 100$, derived by $1.25 * 80 = 100$ for MICKEY 2.0 with a 80-bit secret key. The basic components for the key stream generation are two registers R and S. Each of them is n stages long, whereas each stage contains exactly one bit. As for MICKEY 2.0 $n = 100$ holds, we denote $R = r_0, ..., r_{99}$ and $S = s_0, ..., s_{99}$ for the bits of the registers.

The linear register R is responsible for good local statistical properties, hardening the cipher against statistical attacks. It does ensure that the state of the generator does not repeat within a single key stream sequence [11]. The register S is the non-linear register. The authors introduce non-linearity to protect against attacks exploiting the linearity, like distinguishing attacks or algebraic attacks. Summarized, the advantages of both types of registers are used complementary, to harden the cipher against the different types of attacks.

Clocking Modes In this section we describe the clocking modes of the registers. In order to avoid confusions, *clocking* refers following to the switching of the two actions and not to the time when an action is performed.

Remarkable is the variable clocking used by the cipher. It does not only protect against many types of attacks, it also ensures that the key stream generator state does not repeat while producing a single key stream sequence. This implies that also the key stream does not repeat within a single key stream sequence, making it more difficult to reconstruct the internal state or draw conclusions of the generated output. If this property would not be given, a class of weak keys could lead to a break of the cipher. Furthermore, it prevents S from becoming stuck in a cycle of a few continuous repeating states because the feedback of S depends on the output of register R.

The cipher is called *mutual* irregularly clocked, meaning the clocking of each of the registers depends on the other. The action in each clock cycle is controlled by so-called control bits, one for each register, named C_R respectively C_S. The bit of each control bit is derived by bits of both register, which explains the mutuality of the clocking. This way of irregular clocking protects the cipher against attacks like “guess and determine”. This variable clocking architecture is illustrated in Figure 1 and explained in the following.

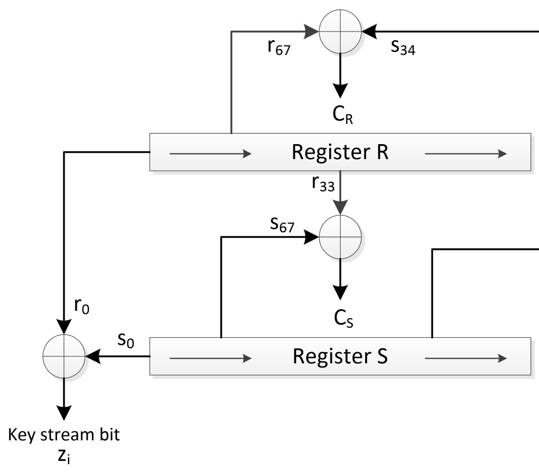

Fig. 1. Dependancy between the two registers R and S (based on [10])

Clocking of Register R In the following we describe the clocking modes of register R, controlled by the control bit C_R. The feedback tap positions of R, which affect the next state, are denoted with $RTAPS$. For a detailed definition of these bit sequences we refer to [10].

The behavior of register R can be described as follow: If the control bit $C_R = 0$, then the clocking is according to a standard linear shift register (LFSR). The feedback is based on the Galois-style feedback with characteristic polynomial $CP_R(x) = x^n + \sum_{i \in RTAPS} x^i$. Otherwise, if $C_R = 1$, then additionally to shifting each bit in the register to the right, each bit r_i is XORed with its predecessor bit r_{i-1} and fed back into the current stage.

The clocking behavior of R is described following in Listing 1. The state of the register R before the clocking operation is denoted by $r_0, ..., r_{99}$, and after clocking by $r'_0, ..., r'_{99}$.

Listing 1 Clocking of register R

```
1: procedure CLOCK_R($R, I_R, C_R$)    ▷ register R, input bit $I_R$, control bit $C_R$
2:     FEEDBACK_BIT = $r_{99} \oplus I_R$
3:     for $1 \leq i \leq 99$ do    ▷ bits are shifted to the right (common LFSR operation)
4:         $r'_i = r_{i-1}$
5:         $r'_0 = 0$
6:     end for
7:     for $0 \leq i \leq 99$ do
8:         if $i \in RTAPS$ then
9:             $r'_i = r'_i \oplus$ FEEDBACK_BIT ▷ tap bits are XORed with the feedback bit
10:        end if
11:    end for
12:    if $C_R = 1$ then
13:        for $0 \leq i \leq 99$ do
14:            $r'_i = r'_i \oplus r_i$    ▷ bits are XORed and fed back into current stage
15:        end for
16:    end if
17: end procedure
```

Clocking of Register S Completing the description of the clocking modes, we present the clocking behavior of the register S, which is controlled by the control bit C_S. It uses four sequences denoted by $(COMP0_i)_{i=1}^{98}$, $(COMP1_i)_{i=1}^{98}$, $(FB0_i)_{i=0}^{99}$ and $(FB1_i)_{i=0}^{99}$. For details of their definition we refer to [10].

In Listing 2 we denote $s_0, ..., s_{99}$ as the state of register S before clocking and $s'_0, ..., s'_{99}$ after clocking. The algorithm is simplified with aid of intermediate variables $\hat{s}_0, ..., \hat{s}_{99}$.

Overall Clocking Finally, we present the clocking of the overall generator in Listing 3. It has as input, besides of the registers R, S and the input bit I, a mixing bit M, which indicates whether the input bit R is solely based on the input bit I or generated by XORing with the bit s_{50} of register S. During the key loading and initialization phase the mixing bit is enabled, while during the key stream generation phase the mixing bit is disabled.

Key Loading and Initialization We have seen the clocking of the registers and the overall clocking, next we consider the loading and initialization of the keys. In this phase the key and the initialization variable is loaded into the registers. The mechanism used to load the key uses register S, thus it is a non-linear key loading mechanism. This non-linearity protects against resynchronization attacks.

Listing 2 Clocking of register S

```
procedure CLOCK_S($S, I_S, C_S$)    ▷ register S, input bit $I_S$, control bit $C_S$
    FEEDBACK_BIT = $s_{99} \oplus I_S$
    for $1 \leq i \leq 98$ do
        $\hat{s}_i = s_{i-1} \oplus ((s_i \oplus COMP0_i).(s_{i+1} \oplus COMP1_i))$
        $\hat{s}_0 = 0$
        $\hat{s}_{99} = s_{98}$
    end for
    if $C_S = 0$ then    ▷ $C_S = 0$
        for $0 \leq i \leq 99$ do
            $s'_i = \hat{s}_i \oplus (FB0_i.$FEEDBACK_BIT$)$
        end for
    else    ▷ $C_S = 1$
        for $0 \leq i \leq 99$ do
            $s'_i = \hat{s}_i \oplus (FB1_i.$FEEDBACK_BIT$)$
        end for
    end if
end procedure
```

Listing 3 Overall clocking

```
procedure CLOCK_KG($R, S, M, I$)    ▷ registers $R$, $S$, mixing bit $M$, input bit $I$
    $C_R = s_{34} \oplus r_{67}$    ▷ define control bit for R
    $C_S = s_{67} \oplus r_{33}$    ▷ define control bit for S
    if $M$ = TRUE then    ▷ mixing bit is TRUE
        $I_R = I \oplus s_{50}$
    else    ▷ mixing bit is FALSE
        $I_R = I$
    end if
    $I_S = I$
    CLOCK_R($R, I_R, C_R$)    ▷ clock register R
    CLOCK_S($S, I_S, C_S$)    ▷ clock register S
end procedure
```

At first, both registers are initialized with all zeros. Following the initialization variable IV is loaded, then the key K and lastly the preclocking is initiiated. This process is described in Listing 4.

Generation of the key stream As most stream ciphers, the cipher text is produced by bitwise XOR with the generated key stream bits. We denote the output as $z_0, , ..., z_j$. The process of key stream generation is illustrated in Listing 5.

Usage restrictions There are some usage restrictions of MICKEY, required to ensure the intended security of the cipher. The maximum length of a key stream sequence with a single pair (K, IV) is restricted to $2^{|K|/2}$ bits, thus for MICKEY 2.0 it is 2^{40} bits. It is allowed to generate 2^{40} such sequences from the same K

Listing 4 Key loading and initialization

```
for $0 \leq i \leq 99$ do                                  ▷ initialize $R$ and $S$ with all zeros
    $r_i = 0$
    $s_i = 0$
end for
for $0 \leq i \leq IV.\text{LENGTH} - 1$ do                ▷ load in $IV$
    $\text{CLOCK\_KG}(R, S, \text{MIXING} = \textsc{True}, I = iv_i)$
end for
for $0 \leq i \leq 79$ do                                  ▷ load in $K$
    $\text{CLOCK\_KG}(R, S, \text{MIXING} = \textsc{True}, I = k_i)$
end for
for $0 \leq i \leq 99$ do                                  ▷ preclock
    $\text{CLOCK\_KG}(R, S, \text{MIXING} = \textsc{True}, I = 0)$
end for
```

Listing 5 Key stream generation

```
for $0 \leq i \leq j$ do
    $z_i = r_0 \oplus s_0$
    $\text{CLOCK\_KG}(R, S, \text{MIXING} = \textsc{False}, I = 0)$
end for
```

with differing IVs. But it is not acceptable to use two IVs of different lengths with the same K or to reuse any IV with the same key K.

2.2 MICKEY-128 2.0

This variant of MICKEY uses a 128-bit key instead of the 80-bit key used for MICKEY 2.0. Thus, the register size for R and S is adapted to $128 * 1.25 = 160$ stages. As a consequence, as well the feedback tap positions and the four sequences used for register S, are scaled accordingly. The definition of these parameters can be found in the MICKEY-128 2.0 specification [8].

The cipher itself, including the clocking of the registers R and S, the clocking of the overall generator, the key initialization and the generation of the key stream, are basically the same as for MICKEY 2.0.

2.3 Earlier specifications: MICKEY 1.0 & MICKEY-128 1.0

The only difference to earlier specifications of MICKEY, named MICKEY 1.0 [9] and MICKEY-128 1.0 [11], is the smaller register size of 80-bit respectively 128-bit. This change in version 2.0 was performed due to the results of a security evaluation by Hong and Kim [69]. It was shown that MICKEY 1.0 has weaknesses regarding three aspects:

The cipher is susceptible to a Time-Memory-Data (TMD) tradeoff attack with online time, data and memory complexity, all less than the key size of 2^{80}. Although the one-off precomputation time complexity was always greater than 2^{80} and this attack can therefore not definitely be counted as successful, the authors acknowledged the found weakness.

Moreover, the analysis showed an entropy loss in the state update function as it is iterated. This can result in state collisions, caused by the convergence of distinct key stream sequences. The reduction of entropy is in general enabled by the variable clocking used in MICKEY, which reduces the entropy when the generator is clocked. For example, the production of a 2^{40}-bit sequence decreases the entropy to nearly 2^{120} from initially 2^{160}. If we consider the key length of 80-bit, then 120 is less than twice the key size. As a consequence, an amount of data less than 80-bit (key size) will lead to collisions.

Lastly, a small class of weak keys have been found. In the specification of MICKEY 1.0 the existence of such a class was stated, where the register R is in an all zeroes state and remains permanently in it. But because the presence of the state was estimated to be roughly 2^{-80}, which is less than the probability for any guessed secret to be correct, the authors did not try to prevent it. Hong and Kim showed that also for register S exists a fixed S-state. In this state the register S is all zero and the produced key stream is equal to the produced output of register R. Hence it was proved that there are more weak key classes and the probability of encountering one strictly increases to 2^{-79}, which is greater than estimated before.

2.4 Hardware Implementations and Complexity

The stream cipher MICKEY is optimized for hardware-constrained environments, this includes, for example, the aspects area, power consumption and memory. The latter depends on the considered MICKEY variant, i.e. MICKEY 2.0 requires 200 bits (100 bit for each register) and MICKEY-128 2.0 requires 320 bits (160 bit for each register) of memory in total. Within the eSTREAM project several proposals for hardware implementations were analyzed. Following we present a short overview.

In [79] the performance of an FPGA-based MICKEY 2.0 implementation was analyzed and compared to five other representative stream ciphers. The results in Table 1 confirm the low requirement regarding area consumption, which is the best of all competitors on this specific FPGA platform. Although the throughput of 250 Mbps is lower than of ZUC (7111), Trivium (326) and Snow3g (3328), it has a very good ratio of 2.551 Mbps/slice. This is besides of ZUC with 12.367 Mbps/slice the best value among all other candidates. Concluded, MICKEY 2.0 has low complexity in hardware while still providing a good throughput.

For MICKEY-128 2.0 an FPGA-based hardware implementation was evaluated in [71]. The implementation on a Xilinx Spartan 3, optimized for minimum area, required 176 slices. This is remarkably more than for other eSTREAM finalists Trivium (50), Grain 128 (50) and Grain v1 (44). Considering the throughput to area ratio of 1.27 Mbps/slice, the implementation has the lowest value compared to the others, Trivium (4.80 Mbps/slice), Grain 128 (3.92 Mbps/slice) and Grain v1 (4.45 Mbps/slice).

Summarized, the variant MICKEY-128 2.0 provides more security than MICKEY 2.0, but in return needs more area and provides a lower throughput.

Cipher	Device	Area [#slices]	Frequency [MHz]	Throughput [Mbps]	T/A [Mbps/Slice]
ZUC	XC5VLX110T	575	222.4	7111	12.367
MICKEY 2.0	**XC3S700A-4FG484**	**98**	**250**	**250**	**2.551**
Trivium	XC3S700A-4FG484	149	326	326	2.188
E0	XC3S700A-4FG484	140	187	187	1.335
Snow3g	XC3S700A-4FG484	3,559	104	3328	0.935
Grain V1	XC3S700A-4FG484	318	177	177	0.557

Table 1. Performance comparison of six representative stream ciphers (source: [79]).

These drawbacks from MICKEY-128 2.0 can be reduced by using parallelization. Although not explicitly designed for, the authors of MICKEY propose that instead of producing one bit per cycle, several bits per cycle can be generated. In [108] a two-folded parallelization was realized based on a FPGA. Instead of the suggested lookup tables, they used a novel mathematical interpretation of the algorithm which enabled the calculation of critical look-ahead bits. This technique was proved to be more efficient than the use of lookup tables. The results visualized in Table 2, show a throughput of 560 Mbps for an area of 392 slices, resulting in 1.43 Mbps/slice. The authors noted that further improvements, like pipelining and 4- or 8-bit key stream generation, may be realized in future work.

Cipher	Frequency [MHz]	Area [#slices]	Throughput [Mbps]	T/A [Mbps/#slice]
Trivium (x64)	211	344	13504	39.26
Grain 128 (x32)	133	534	4256	7.97
Grain v1 (x16)	130	348	2080	5.98
Trivium	240	50	240	4.80
Grain v1	196	44	196	4.45
Grain 128	196	50	196	3.92
MICKEY-128 2.0 (x2)	**(280)**	**392**	**560**	**1.43**
MICKEY-128 2.0	**223**	**176**	**223**	**1.27**

Table 2. Performance comparison of MICKEY-128 2.0 (source: [71,108]).

All ciphers presented in Table 2 were implemented on a Xilinx Spartan-3 FPGA XC3S50-5PQ208, except Trivium (x64) which was implemented on a Xilinx Spartan-3 FPGA XC3S400-5FG320 and MICKEY-128 2.0 (x2) which was implemented on a Xilinx Virtex-II Pro FPGA XC2VP30.

2.5 Security Evaluation

During the eSTREAM project the security of the cipher MICKEY has been evaluated extensively. The results of various analysis showed weaknesses which can be remediated by increasing the register size, see Section 2.3 for more details.

In the following we present some known attacks targeting MICKEY 2.0 and MICKEY-128 2.0 with focus on side-channels, like differential fault attacks and template attacks. As described in the specification of MICKEY 2.0, the authors have taken some measures to make the cipher secure against resynchronization attacks, algebraic attacks and attacks like “guess and determine” or “divide and conquer”. The last review of the eSTREAM finalists portfolio was in 2012, stating that there are no known cryptanalytic advances for MICKEY 2.0 or MICKEY-128 2.0 so far [34]. To the best of the authors knowledge, since then no new attacks - besides side-channels attacks - have been successfully accomplished.

Correlation Power Analysis Attack (CPA) An attack proposed by Liu et al. [74] is based on power traces gathered from the output gates and the internal nodes (of the gates) of a MICKEY 2.0 implementation with cascaded CMOS inverters. It uses a *Hamming-Distance* model to map the transitions of the cells' outputs (CMOS circuit) to the values of power consumption. This allows to determine a relation between power consumption and data handled by the cipher. With aid of the model, Liu et al. showed an attack against MICKEY 2.0, that just requires a few (about ten) power traces during initialization. As an result it was possible to reveal the secret key. This attack is enabled by the high correlation between power consumption and bit change during initialization. As a countermeasure an exchange of the steps “loading in K” and “loading in IV”, see Listing 4, was proposed.

Template Attack In general, CPA attacks require a high number of different power traces gathered during the key loading of the cipher. Therefore it is difficult for adversaries to perform these attacks under real circumstances. As an alternative, Chakraborty and Mukhopadhyay [33] proposed a template attack which uses IVs generated by a particle swarm optimization computational method. This drastically reduces the number of required power traces to break the cipher. Observation showed that about 500 traces were sufficient compared to about 3000 traces if the IVs were randomly chosen. As an result, this method not only requires less power traces, but also a much lesser number of IVs.

Differential Fault Attack Another attack based on side-channels uses random injections of faults, mostly introduced by laser shots or clock glitches, which causes a change of bits in the internal state. Using the output of the device, an adversary can deduce information of the internal state or even the secret key.

Banik and Maitra [13] proposed an attack, which needs about $2^{16.7}$ fault injections and $2^{32.5}$ computations on average, to completely recover the internal state. Important is, that the faults must be injected before entering the pseudo random bitstream generation (PRGA) phase, this is right after loading the IV/key and execution of pre-clocking. As an assumption of the attack, the adversary needs to know the length of the IV, because it is variable. Banik and Maitra suggest a straightforward way to determine the length by injecting faults during the PRGA clock round.

Later, an improvement of the attack was presented by Banik et al. [16]. In this attack it is assumed that injected faults affect two or three neighboring bits which reduces the required faults to $\approx 2^{18.4}$ on average. Moreover, the use of a SAT solver to further reduce the required faults was analyzed. Assuming the attacker has derived a bits of the state R_0 (register R) by applying faults, it is possible to find the remaining bits of R_0 and S_0 by using equations over the fault-free key-stream bits. The use of a SAT solver to find solutions to these non-linear equations reduced the required number of faults to $\approx 2^{16.06}$.

MICKEY-128 2.0 also was shown to be susceptible to the same type of attack. The attack by Karmakar and Chowdhury [75] on MICKEY-128 2.0 requires 480 faults and about 480 faulty/faulty-free key stream pairs to break the cipher. The used methodology at first determines the position of the fault by using single bit and related single bit faults. After that, linear equations are obtained involving internal state bits of different iterations. By determining the state bits for a specific base point, Karmakar and Chowdhury were able to predict the key stream of any later iteration. As a result, the linear equations could be solved and the internal state could be reconstructed.

Conclusion The presented attacks show that side-channels of MICKEY 2.0 and MICKEY-128 2.0 can be used to successfully break the cipher. Karmakar and Chuowdhury state "the attack is mountable in few hours [...]" [75], highlighting the practical feasibility of the shown attacks. As well all other presented attacks have been practically proven. For that reason, the use of MICKEY 2.0 or MICKEY-128 2.0 in environment where physical protection of side-channels can not be guaranteed, should be weighted up carefully.

3 WG-8

This section provides an overview of the stream cipher WG-8. We introduce its structure, implementation and performance as well as security aspects.

WG-8 [52] was presented in 2013 by Fan, Mandal and Gong. It is a lightweight variant of the WG stream cipher family [94], specifically designed for resource constrained devices. Like all WG ciphers, it makes use of Welch-Gong transformations [95] which generate bit sequences with provable randomness properties. These properties are very interesting in cryptographic contexts [58], [59]. They can be used to prove the security of WG ciphers against many common attacks on stream ciphers.

The initial WG cipher was submitted to the eSTREAM project but was eliminated at the end of phase 2, partly due to security concerns but mostly because it was too complex to be efficiently implemented in hardware. However, the latter issue was addressed in the design of WG-8 which has a hardware complexity similar to the hardware-oriented eSTREAM finalists.

In this section, we at first explain the structure of WG-8 in detail. Then we give a brief overview of the WG cipher family and its history. Next we have a look at how WG-8 can be implemented in software and hardware as well as the performance of such implementations. Finally, we address security aspects of WG-8.

3.1 Structure

The main component of the WG-8 cipher is a 160-bit LFSR. It operates byte wise and generates one byte per cycle. Its initial state is denoted as $S_0, ..., S_{19} \in \mathbb{F}_{2^8}$ with $S_i = (S_{i,7}, ...S_{i,0})$ for $0 \leq i \leq 19$. $\mathbb{F}_{2^8}$ is the finite field defined by the primitive polynomial $p(x) = x^8 + x^4 + x^3 + x^2 + x + 1$. In the following, $\oplus$ denotes addition in $\mathbb{F}_{2^8}$ (which is the same as XOR) and $\otimes$ denotes multiplication in $\mathbb{F}_{2^8}$.

The cipher uses an 80-bit key $(K_{79}, ...K_0)$ and an 80-bit initialization vector (IV) $(IV_{79}, ..., IV_0)$ which are loaded into the LFSR according to these equations (for $0 \leq i \leq 9$):

$$S_{2i} = (K_{8i+3}, ..., K_{8i}, IV_{8i+3}, ..., IV_{8i})$$

$$S_{2i+1} = (K_{8i+7}, ..., K_{8i+4}, IV_{8i+7}, ..., IV_{8i+4})$$

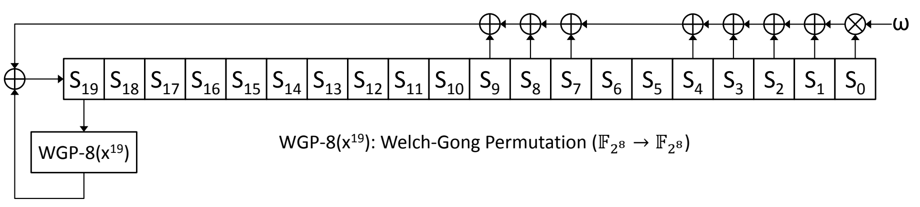
ω
S19 S18 S17 S16 S15 S14 S13 S12 S11 S10 S9 S8 S7 S6 S5 S4 S3 S2 S1 S0
WGP-8(x19)
WGP-8(x19): Welch-Gong Permutation (𝔽28 → 𝔽28)

Fig. 2. Initialization Phase of WG-8 (based on [52])

Initialization Phase Before the keystream is generated, the LFSR spends 40 cycles in the Initialization Phase which is shown in Figure 2.

In this phase, a Welch-Gong permutation (WGP-8) of the most recent byte (initially S_{19}) is added to the regular feedback of the LFSR (which makes the feedback non-linear during the Initialization Phase). The WG permutation ($\mathbb{F}_{2^8} \mapsto \mathbb{F}_{2^8}$) is defined as:

$$\text{WGP-8}(x^d) = q(x^d + 1) + 1$$

where q is a permutation polynomial defined as:

$$q(x) = x + x^{2^3+1} + x^{2^6+2^3+1} + x^{2^6-2^3+1} + x^{2^6+2^3-1}$$

d is called decimation and must be coprime to $2^8 - 1$. Unfortunately the purpose of the decimation is not made clear in [52]. However, in [1], the cipher WG-5 is discussed and it is said that the decimation value influences linear complexity as well as non-linearity. If carefully chosen, it can increase the resistance of a WG cipher against algebraic attacks. For the WG-8 cipher, d is set to 19.

The linear part of the feedback is calculated by first multiplying the oldest byte (initially S_0) with a constant ω, which must be a primitive element (also known as generator) of $\mathbb{F}_{2^8}$ with $p(\omega) = 0$. Unfortunately [52] does not offer any example values for ω. More bytes from the LFSR are then added to the product. The entire feedback is calculated as:

$$\begin{aligned} S_{k+20} = (\omega \otimes S_k) \oplus S_{k+1} \oplus S_{k+2} \oplus S_{k+3} \oplus S_{k+4} \oplus S_{k+7} \\ \oplus S_{k+8} \oplus S_{k+9} \oplus \text{WGP-8}(S_{k+19}^{19}), \quad 0 \leq k \leq 39 \end{aligned}$$

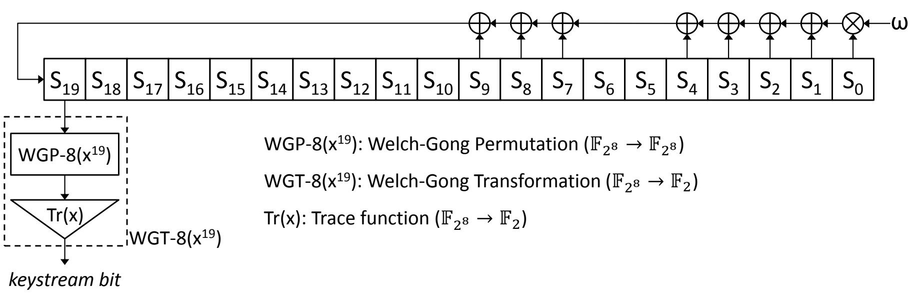
ω
S19 S18 S17 S16 S15 S14 S13 S12 S11 S10 S9 S8 S7 S6 S5 S4 S3 S2 S1 S0
WGP-8(x19)
Tr(x)
WGT-8(x19)
keystream bit
WGP-8(x19): Welch-Gong Permutation (𝔽28 → 𝔽28)
WGT-8(x19): Welch-Gong Transformation (𝔽28 → 𝔽2)
Tr(x): Trace function (𝔽28 → 𝔽2)

Fig. 3. Running Phase of WG-8 (based on [52])

Running Phase After the Initialization Phase the Running Phase, as shown in Figure 3, is entered. In this phase one keystream bit is generated per cycle.

The result of the WG permutation is no longer added to the feedback, which is thus defined as:

$$S_{k+20} = (\omega \otimes S_k) \oplus S_{k+1} \oplus S_{k+2} \oplus S_{k+3} \oplus S_{k+4} \oplus S_{k+7} \oplus S_{k+8} \oplus S_{k+9}, \quad k \geq 40$$

Instead, the trace function is applied to the permutation output. The trace function is defined as $\text{Tr}(x) = x + x^2 + x^{2^2} + ... + x^{2^7}$ and maps from $\mathbb{F}_{2^8}$ to $\mathbb{F}_2$. The combination of WG permutation and trace function is called WG transformation and is defined as:

$$\text{WGT-8}(x^d) = \text{Tr}(\text{WGP-8}(x^d))$$

where $d = 19$ is used once again for WG-8. The result of the WG transformation is the next keystream bit.

3.2 The WG stream cipher family

There are several versions of WG ciphers for use in different settings. They all have very similar structure and mainly vary in the number and size of elements in the LFSR.

WG version 1 The first version of the WG cipher was published by Nawaz and Gong in 2005 [93]. It was submitted to the eSTREAM project as a hardware-oriented stream cipher. It uses an LFSR with 11 elements from $\mathbb{F}_{2^{29}}$. The structure is similar to later versions, except that the WG transformation is applied to the element at the other end of the register (initially S_0). Even though the size of the LFSR is fixed, the specification allows different key sizes (80, 69, 112 and 128 bits). The IVs can have the same size as the keys, but shorter IVs of 64 or 32 bits are also possible. The only operational difference between these sizes is how keys and IVs are loaded into the LFSR initially.

This version of the cipher was broken. After only two months, Wu and Preneel presented a chosen IV attack on the cipher that could recover the key for most key/IV sizes [113]. Their attack was formally published in [114]. As an initial reaction to this attack, the creators of the WG cipher suggested doubling or even quadrupling the number of cycles spent in the Initialization Phase [92].

Final version and WG-128 The final version of the WG cipher (now officially called a family of stream ciphers) was published in 2008 [94]. It moved the WG transformation to the other end of the LFSR, just like in WG-8. This makes the cipher secure against the aforementioned chosen IV attack without introducing the overhead of additional initialization cycles.

The final specification is also more generic. It uses variables instead of fixed values for the number of elements in the LFSR as well as for their size. WG-128 was introduced as a concrete example. It is basically the original WG cipher, with key and IV sizes of 128 bits.

The WG ciphers were eventually eliminated from the eSTREAM selection process at the end of phase 2. According to [22] the reasons were twofold: Firstly, there were security concerns. In 2007 an attack on a general filter generator over $\mathbb{F}_{2^m}$ was presented by Rønjom and Helleseth [103]. It can be used to recover the internal state of the WG cipher. However, the cipher specification allows for no more than 2^{45} keystream bits to be generated with the same key/IV pair which is less than the $2^{45.0415}$ keystream bits that are required for the attack. While this technically makes the WG ciphers secure against this attack, it left the eSTREAM committee with a bad feeling as the cipher could be broken after only a slight relaxation of the its restrictions.

Secondly, the committee had doubts that the cipher could be implemented in hardware as efficiently as the other candidates. This is mainly because WG-128, as well as the original version of WG, contain many calculations in $\mathbb{F}_{2^{29}}$ which require a lot of area when implemented in hardware. All newer members of the WG family use much smaller finite fields to address this concern.

WG-16 In 2013 Fan and Gong presented the stream cipher WG-16 [51]. It is designed for the use in 4G-LTE networks. The authors also specify confidentiality and integrity algorithms that make use of their new cipher. To motivate their work, they claim that the ciphers in the current 4G-LTE standard are difficult to analyze and that the current integrity algorithms are vulnerable. WG-16 uses 128-bit keys and IVs as well as an LFSR that contains 32 elements from $\mathbb{F}_{2^{16}}$.

WG-7 The cipher WG-7 is the predecessor to WG-8, also targeting resource constrained devices. It was published by Luo et al. in 2010. It uses 80-bit keys and 81-bit IVs. The LFSR contains 23 elements from $\mathbb{F}_{2^7}$. In 2012 the cipher was broken by Orumiehchiha et al. [97]. We will look into the details of their attack in section 3.4 as it is related to the security of WG-8.

WG-5 In [1] Aasgaard, Gong and Mota discuss Hardware implementations and security issues of the stream cipher WG-5 which was originally proposed by Mota in 2012 in his master thesis [91]. The cipher is aimed at passive RFID tags and offers only a low security level. It is resistant against attacks only if the data encrypted with one key/IV pair does not exceed 256 kilobytes, which is an acceptable constraint for passive RFID tags.

3.3 Implementation

Software In the original work about WG-8 [52] the authors only address software implementations. They explore different implementations of the WG transformation, which is the most calculation intensive part of the cipher. They come to the conclusion that lookup tables are the best alternative in terms of speed, memory usage and power consumption. To implement the WG transformation, a lookup table with 256 bits is required. For the WG permutation, which is used in the initialization phase, an additional lookup table with 256 bytes is needed.

WG-8 was implemented on two microcontrollers, namely the 8-bit ATmega128L from Atmel and the 16-bit MSP430F1611 from Texas Instruments, both clocked at 8 MHz. For each controller, the authors compare the performance of WG-8 with implementations of other stream and block ciphers that exist for that controller. Since this paper is about hardware-oriented stream ciphers, we don't want to focus too much on software implementations. We only show some results for the ATmega, for which (in our opinion) more interesting ciphers were used for the comparison. Table 3 shows the performance of the ciphers. WG-8 is faster than most ciphers, has low energy consumption and average memory requirements. Notably, the block cipher AES clearly outperforms WG-8 on this device.

Hardware In [115] Yang et al. explore possible hardware implementations of WG-8. Their work focuses mainly on different implementations of the WG permutation and transformation. After trying three different types of tower field

cipher	optimization goal / method	memory usage [byte]		throughput	Energy/bit
		Flash	SRAM	[Kbits/sec]	[nJ]
AES	RAM	1,912	176	475.6	179
	Speed	1,912	256	513.8	165
PRESENT-80	Size	1,474	32	0.99	85,819
	Speed	2,398	528	66.7	1,274
Hummingbird-2	RAM	3,600	114	171.8	495
	Speed	3,200	1,500	258.6	329
Grain	Speed	778	20	12.9	6,556
Trivium	Speed	424	36	12.0	7,066
Salsa20	Speed	3,842	258	83.7	101,564
WG-8	**lookup table**	**1,984**	**20**	**185.5**	**458**

Table 3. Performance of WG-8 and other ciphers on ATmega128L microcontroller.

arithmetic, they come to the conclusion that, just as in software, lookup tables are the best solution. They require less space and energy and are faster than any other approach. They note however, that for ciphers like WG-16, lookup tables become very large and tower field arithmetic provides a smaller alternative.

Their paper also looks at potential parallelization of WG-8. They were able to implement a rather efficient version that generates 11 bits per clock cycle but only requires about three times the area compared to the non-parallelized version. The number 11 comes from the elements inside the LFSR that are used to calculate the feedback. The next 11 bytes can be calculated with the current values in the LFSR. However, the 12th byte can only be calculated after the first new byte has already been generated. After calculating the next 11 bytes, the corresponding keystream bits can be calculated by using 11 copies of the same 256-bit lookup table. As these lookup tables are very small, the overhead is minimal. In the initialization phase, a 256 byte lookup table is required as the WG permutation generates a byte and not a single bit. As this lookup table is much larger, the authors recommend not parallelizing the initialization phase.

The authors implemented WG-8 on a Spartan-3 XC3S1000 FPGA and as an application specific integrated circuit (ASIC) with 65nm CMOS technology. They also compare their results to the stream ciphers Grain (v1) and Trivium which have highly efficient hardware implementations. For the FPGA, they use [71] as source for Grain's and Trivium's performance data. For the ASIC however, they wrote their own implementations of Grain and Trivium, including versions that generate 11 keystream bits per cycle. It is not known if and how those implementations were optimized which should be kept in mind when comparing the results. Table 4 shows the performance on the FPGA and table 5 shows the ASIC performance. The tables contain only the lookup table implementation of WG-8 as it has the best performance in every way. For the ASIC implementations, the area is given in GE, i.e. gate equivalence, which is the number of NAND gates that would take the same area. The estimated area of a NAND gate is $2.08um^2$.

Grain and Trivium outperform WG-8 on the FPGA. Their single-bit implementations are much smaller, while offering even slightly more throughput than

cipher	data rate [bits / cycle]	area [slices]	throughput [Mbps]	throughput / area
WG-8	1	137	190	1.39
	11	398	2112	5.31
Grain	1	44	196	4.45
	16	348	2080	5.98
Trivium	1	50	240	4.80
	64	344	13504	39.26

Table 4. Performance of WG-8 and other ciphers on Spartan-3 XC3S1000 FPGA.

cipher	data rate [bits / cycle]	area [GE]	power [mW]	throughput [Mbps]	throughput / area	throughput / power
WG-8	1	1786	0.983	500	0.28	508.6
	11	3942	1.344	6710	1.70	4992.6
Grain	1	1126	2.04	1020	0.91	500
	11	1126	2.25	12078	10.73	5368
Trivium	1	1986	3.88	962	0.48	247.9
	11	2028	4	10890	5.37	2722.5

Table 5. Performance of WG-8 and other ciphers as 65nm CMOS ASIC.

WG-8. For the paralleled case, WG-8 actually offers more throughput than Grain (though much less than Trivium) but it also requires more space, which leads to slightly less throughput per area.

For the ASIC implementations however, there is no clear winner. The single-bit version of WG-8 needs more area than Grain, but less than Trivium. WG-8 needs about 2 times less power than Grain and about 3 to 4 times less power than Trivium. However its throughput is only about half the throughput of the other ciphers. Overall WG-8 can compete with established hardware oriented ciphers which the original WG cipher or WG-128 cannot. [50] explores hardware implementations of WG-128 using the same ASIC technology. For single-bit WG-128 the implementation requires more than 10 times the area and more than 4 times the power of the single-bit WG-8 implementation while only achieving slightly less than half of its throughput.

3.4 Security

Due to the WG transformation, the keystream generated by WG-8 possesses a number of randomness properties that the creators of WG-8 consider to be desirable.

- The keystream has a well-known period of $2^{160} - 1$.
- The keystream is balanced, i.e. over one period the number of 0's is only one less than the number of 1's.
- The keystream possesses ideal two-level autocorrelation. This means that the correlation between one full keystream period and a rotated version of that

period is always -1. The correlation between two bit sequences is defined as the difference of the number of equal bits and the number of unequal bits.
- The keystream has ideal t-tuple distribution ($1 \leq t \leq 20$). This means that all possible t-tuples appear equally often in one period.
- The keystream has a linear span, also known as linear complexity, of exactly $2^{33.32}$. The linear span is the minimum size of any LFSR that generates a bit sequence identical to the keystream.

The creators of WG-8 use these properties to argue about common attacks on stream ciphers and their effectiveness against WG-8. They show that most attacks either require an unrealistic amount of time or keystream bits. It is claimed that WG-8 is secure against the following attacks:

- Algebraic Attack
- Correlation Attack
- Cube Attack
- Distinguishing Attack
- Time-Memory Trade-off Attack

Key Recovery Attack in the related key setting In [40] Ding et al. present a key recovery attack on WG-8 which can fully restore the key within minutes on an average PC. However, the attack uses some powerful assumptions which we consider to be unrealistic. Nevertheless, the mere fact that the attack exists, shows that WG-8 is by no means perfectly secure and should be further analyzed.

The attack assumes that the cipher is used with several keys, which are all bit-wise rotated versions of each other. The rotations must be done by multiple of 4 bits. In this case, if the IVs used with those keys are also rotated versions of each other, there is a chance that the generated keystreams will be shifted versions of each other. The authors describe equations that depend on some key and IV bits and are fulfilled if and only if shifted keystreams occur. The attack consists of trying different IVs until shifted keystreams are detected, which means the equations are satisfied. Then the equations can be solved for the key bits. The more rotated versions of a key are used, the more key bits can be restored this way. To be able to detect shifted keystreams the attacker must have access to sufficiently many keystream bits.

Attacks on WG-7 and their implications for security of WG-8 In [97] two attacks on the stream cipher WG-7 are presented. First is a distinguishing attack that can almost certainly distinguish a keystream from a random bit sequence when $2^{13.5}$ keystream bits are known. Second is an algebraic attack which can restore the key as well as the internal state of the LFSR with $2^{19.38}$ keystream bits and a time complexity of $2^{26.73}$. The creators of WG-8 reason, that these attacks were possible because the LFSR feedback in WG-7 is calculated from only two LFSR elements. Since WG-8 uses eight elements to calculate the feedback, it is secure against these attacks.

Nevertheless, these attacks reveal two potential problems of WG ciphers in general. First of all, the randomness properties of the keystream might not be as advantageous as assumed. Such a keystream might be distinguishable from a truly random bit sequence because it is, in a way, too perfect. A truly random sequence will probably not possess all these properties.

More importantly, the algebraic attack was more efficient than the cipher creators expected. Their claim, that WG-7 was secure against such attacks, was simply invalid. This puts all other security claims of the WG ciphers in question as well. While there is no realistic attack on WG-8 yet, the cipher definitely should be analyzed more thoroughly before it is used in productive environments.

4 Grain

Grain is a family of stream ciphers, which target restricted hardware environments where gate count, power consumption and the memory is limited. The first version has been developed by Hell et al. in 2005[65]. An important feature of all Grain versions is that the speed of the algorithm can be increased by the expense of more hardware, i.e., the user can decide the speed of the cipher depending on the amount of available hardware. All versions are based on three building blocks: a linear feedback shift register (LFSR), a nonlinear feedback shift register (NLFSR), and a boolean filter function.

The European Network of Excellence in Cryptology (ECRYPT) initiated a request for stream cipher proposals, which was called the eStream project (http://www.ecrypt.eu.org/stream/). From initially 34 submitted candidates, seven final ciphers were eventually turned into a recommended portfolio in September 2008. The initial version Grain v0 was submitted to the contest as first Grain version, but several researchers independently discovered a weakness in the choice of the output function. As a consequence, the output function as well as the update function for the NLFSR of the cipher have been changed. The designers submitted the new version called Grain v1[65], as well as a variant version Grain-128[64] to the second evaluation phase. Grain v1 eventually became one of the seven final ciphers of the eSTREAM portfolio for the hardware-oriented profile 2.

In 2011 Martin gren, Martin Hell, Thomas Johansson, and Willi Meier published Grain-128a[63], a new version based on Grain-128. This new version supports optional message authentication with variable tag sizes and was furthermore strengthened against all known attacks on previous versions by using a slightly different non-linear update function. Currently, Grain-128a is recommended by the authors for 128 bit security, and Grain v1 for 80 bit security.

4.1 Specifications

Since the Grain v0 design was susceptible to serious attacks and has been replaced by Grain v1 and Grain-128 shortly after its publication, the specification details are not covered in this section.

Grain v1 Grain v1 has a secret key size of 80 bit, and an initialization vector IV of 64 bit. Figure 4 shows an overview of the building blocks of Grain. The content of the LFSR is denoted by $s_i, s_{i+1}, \dots, s_{i+79}$ and the content of the NLFSR is denoted by $b_i, b_{i+1}, \dots, b_{i+79}$. In order to update the shift register different tap positions are used, which can be expressed as a polynomial mod 2 in finite field arithmetic, called the feedback polynomial, or as an update function which xores the single tap positions to determine, how to update the LFSR[105]. The feedback polynomial $f(x)$ of the LFSR is of degree 80 and deliberately chosen to be a primitive polynomial to guarantee a period of at least $2^{80} - 1$. It is defined as

$$f(x) = 1 + x^{18} + x^{29} + x^{42} + x^{57} + x^{67} + x^{80}$$

which corresponds to the following tap positions and thus creates the following update function:

$$s_{i+80} = s_{i+62} + s_{i+51} + s_{i+38} + s_{i+23} + s_{i+13} + s_i.$$

The feedback polynomial $g(x)$ of the NLFSR is defined as

$$\begin{aligned} g(x) =& 1 + x^{18} + x^{20} + x^{28} + x^{35} + x^{43} + x^{47} + x^{52} + x^{59} + x^{66} + x^{71} + x^{80} \\ &+ x^{17}x^{20} + x^{43}x^{47} + x^{65}x^{71} + x^{20}x^{28}x^{35} + x^{47}x^{52}x^{59} + x^{17}x^{35}x^{52}x^{71} \\ &+ x^{20}x^{28}x^{43}x^{47} + x^{17}x^{20}x^{59}x^{65} + x^{17}x^{20}x^{28}x^{35}x^{43} + x^{47}x^{52}x^{59}x^{65}x^{71} \\ &+ x^{28}x^{35}x^{43}x^{47}x^{52}x^{59} \end{aligned}$$

which results in the following update function for the NLFSR:

$$\begin{aligned} b_{i+80} =& s_i + b_{i+62} + b_{i+60} + b_{i+52} + b_{i+45} + b_{i+37} + b_{i+33} + b_{i+28} + b_{i+21} \\ &+ b_{i+14} + b_{i+9} + b_i + b_{i+63}b_{i+60} + b_{i+37}b_{i+33} + b_{i+15}b_{i+9} \\ &+ b_{i+60}b_{i+52}b_{i+45} + b_{i+33}b_{i+28}b_{i+21} + b_{i+63}b_{i+45}b_{i+28}b_{i+9} \\ &+ b_{i+60}b_{i+52}b_{i+37}b_{i+33} + b_{i+63}b_{i+60}b_{i+21}b_{i+15} \\ &+ b_{i+63}b_{i+60}b_{i+52}b_{i+45}b_{i+37} + b_{i+33}b_{i+28}b_{i+21}b_{i+15}b_{i+9} \\ &+ b_{i+52}b_{i+45}b_{i+37}b_{i+33}b_{i+28}b_{i+21}. \end{aligned}$$

The NLFSR is chosen to be 2-resilient[70] in order to prevent correlation attacks and information leakage. These two registers describe the current state of the cipher. The input of the NLFSR is masked with the output of the LFSR, in order to ensure that the NLFSR is balanced, i.e., the number of 1's and 0' are nearly equal. The balanced boolean function $h(x)$ takes four bits from the LFSR and one bit from the NLFSR as input and is defined as:

$$\begin{aligned} h(x) = x_1 + x_4 + x_0x_3 + x_2x_3 + x_3x_4 + x_0x_1x_2 + x_0x_2x_3 + x_0x_2x_4 + x_1x_2x_4 \\ + x_2x_3x_4. \end{aligned}$$

The output bit is now generated as follows:

$$z_i = \sum_{k \in \mathcal{A}} b_{i+k} + h(s_{i+3}, s_{i+25}, s_{i+46}, s_{i+64}, b_{i+63})$$

where $\mathcal{A} = \{1, 2, 4, 10, 31, 43, 56\}$

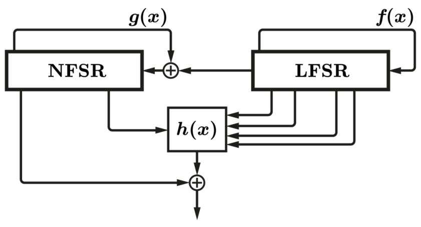

Fig. 4. Grain cipher (by Yossiea, free use, `https://he.wikipedia.org/w/index.php?curid=1217517`)

Key Initialization The Grain algorithm consists of two phases: a key initialization phase, which initializes the state of the cipher before it can be used to generate a keystream in the second phase. The key initialization is the first phase and takes the key k, whose bits are denoted by k_i, where $0 \leq i \leq 79$, and loads the NLFSR with the corresponding bits of the key, $b_i = k_i$. After that it loads the first 64 bit of the LFSR with the initialization vector, denoted by IV_i, where $0 \leq i \leq 79$, so that $s_i = IV_i$ for $0 \leq i \leq 63$. The remaining 16 bits of the LFSR are set to one, $s_i = 1$ for $64 \leq i \leq 79$. Now the algorithm is clocked 160 times without producing any output bits, but rather feeding the resulting bit of the output function z_i back and xoring it with the input of the NLFSR, as well as the LFSR. This process is depicted in Figure 5. The reason for this key initialization phase is to scramble the contents of the shift registers before the key stream is generated.

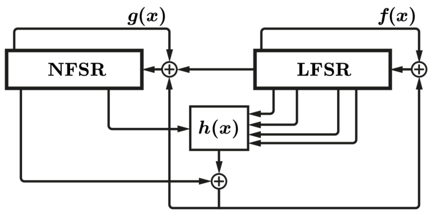

Fig. 5. Grain key initialization (by Yossiea, free use, `https://he.wikipedia.org/w/index.php?curid=1217518`)

Grain-128 Grain-128 is based on the Grain v1 cipher, but uses 128 bit for each of the feedback shift registers, representing an internal state of 256 bit. It supports a key size of 128 bit, the IV consists of 96 bit. Similar to Grain v1 the feedback polynomial of the LFSR is denoted by $f(x)$, and is of degree 128. It is defined as

$$f(x) = 1 + x^{32} + x^{47} + x^{58} + x^{90} + x^{121} + x^{128}.$$

The nonlinear feedback polynomial of the NLFSR is denoted by $g(x)$, and is the sum of one linear and one bent function. It is defined as

$$\begin{aligned} g(x) = {} & 1 + x^{32} + x^{37} + x^{72} + x^{102} + x^{128} + x^{44}x^{60} + x^{61}x^{125} + x^{63}x^{67} \\ & + x^{69}x^{101} + x^{80}x^{88} + x^{110}x^{111} + x^{115}x^{117}. \end{aligned}$$

Grain-128 uses nine variables from the shift registers as input to the boolean function $h(x)$, two from the NLFSR and seven from the LFSR. This function is defined as

$$h(x) = x_0x_1 + x_2x_3 + x_4x_5 + x_6x_7 + x_0x_4x_8$$

where the variables $x_0, \ldots, x_8$ correspond to the tap positions b_{i+12}, s_{i+8}, s_{i+13}, s_{i+20}, b_{i+95}, s_{i+42}, s_{i+60}, s_{i+79} and s_{i+95}. The output function of Grain-128 is defined as

$$z_i = \sum_{j \in \mathcal{A}} b_{i+j} + h(x) + s_{i+93}$$

where $\mathcal{A} = \{2, 15, 36, 45, 64, 73, 89\}$.

Key Initialization The key initialization is similar to the initialization phase for Grain v1. The NLFSR is loaded with the 128 bit key, the LFSR is loaded with the 96 bit IV and the remaining bits are filled with ones. After that the cipher is clocked 256 times, and the output is again fed back into the shift registers.

Grain-128a Grain-128a supports optional message authentication and was furthermore strengthened in regards to all known attacks against previous versions. It supports two different operation modes, one with enabled authentication, and one with disabled authentication. The authentication supports variable tag sizes w up to 32 bits, and when enabled the cipher creates a different key stream in comparison to $w = 0$, i.e., when the authentication is disabled. When $IV_0 = 1$, the authentication is mandatory, otherwise when $IV_0 = 0$ the message authentication is forbidden. Since Grain-128 and Grain-128a are so similar, the following explanation only highlights the differences between these two.

The update function for the NLFSR has been changed, the complexity has been increased:

$$\begin{aligned} g(x) = {} & 1 + x^{32} + x^{37} + x^{72} + x^{102} + x^{128} + x^{44}x^{60} + x^{61}x^{125} + x^{63}x^{67} \\ & + x^{69}x^{101} + x^{80}x^{88} + x^{110}x^{111} + x^{115}x^{117} + x^{46}x^{50}x^{58} \\ & + x^{103}x^{104}x^{106} + x^{33}x^{35}x^{36}x^{40}. \end{aligned}$$

In addition to the update function of the NSFR, the initialization phase has been altered. Grain-128a loads the NLFSR with the 128 bit key, and the LFSR with the 96 bit IV. The remaining bits of the LFSR are filled with ones, but for the last position, which is set to zero $s_{127} = 0$ to avoid a possible attack because of similar IVs (see paragraph 4.3). After that the cipher is clocked 256 times, and the output is again fed back into the shift registers, exactly like Grain-128 or Grain v1.

In the course of the introduction of message authentication the output function also had to be changed. It is different for each mode of operation. The authors therefore define a pre-output function y_i, which is

$$y_i = \sum_{j \in \mathcal{A}} b_{i+j} + h(x) + s_{i+93}$$

where $\mathcal{A} = \{2, 15, 36, 45, 64, 73, 89\}$. When the message authentication is disabled, the output bit is defined as

$$z_i = y_i.$$

That means, that all pre-output bits are directly used as keystream. Given the case, that message authentication has been enabled, the output bit is defined as

$$z_i = y_{64+2i},$$

which means that after skipping 64 bits every second bit is used as output bit.

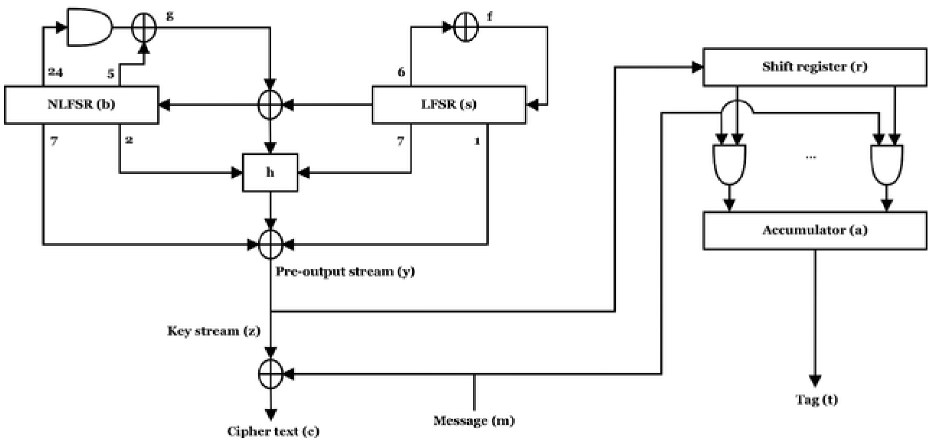

Fig. 6. Grain 128a with optional authentication (CC BY 3.0, `https://en.wikipedia.org/w/index.php?curid=39315190`)

Authentication In order to use the authentication two registers were introduced (cf. Figure 6), called shift register and accumulator. The initially skipped 64 output bits are used to initialize these two registers. The content of the accumulator at time i is denoted by $a_i^0, \ldots, a_i^{31}$ and is initialized with the first 32 bits from the initially skipped 64 bits. The content of the shift register is denoted by $r_i, \ldots, r_{i+31}$ and initialized with the second 32 bits. In order to generate a tag for the message $m_0, \ldots, m_{L-1}$ of the length L, at first the message is appended by a one, $m_L = 1$, to create different tags for two messages m and $m||0$. During the generation of the keystream the shift register is updated as $r_{i+32} = y_{2w+2i+1}$, and the accumulator is updated as $a^j_{i+1} = a^j_i + m_i r_{i+j}$ for $0 \leq j \leq 31$ and $0 \leq i \leq L$. The content of the final accumulator, $a^0_{L+1}, \ldots, a^31_{L+1}$, is called the tag and can be used for authentication. For tags of smaller size, $w < 32$, only the last w bits from the accumulator are used. The authentication follows the *Encrypt and MAC*[18] approach.

4.2 Hardware Implementation and Complexity

The design of the different Grain ciphers is deliberately chosen to be extremely simple to implement in hardware. Grain v1 only requires a memory of 160 bit, Grain-128 and Grain-128a of 256 bit, and the authors tried to simplify the used functions in order to save gates but still provide high security. The shift registers are regularly clocked which results in an output of 1 bit per single clock for the different ciphers. When using Grain-128a with message authentication every second bit is used to generate the tag, therefore the throughput rate is reduced by half. An important feature of all Grain implementations is that the speed of the algorithm can be increased by the expense of more hardware. This is accomplished by implementing $f(x)$ and $g(x)$, and the output function several times. In order to enable this functionality the last bits of the shift registers are not used in the feedback functions, but can be fed as input to multiple instances of parallel implemented feedback functions. For Grain v1 the last 15 bits are ignored, for Grain-128 and Grain-128a the last 31 bits are ignored. If the speed is increased by the factor t, one has to keep in mind that the shift registers need to be implemented so that each bit is shifted t steps instead of one. Using this implementation it is possible to increase the speed up to 16 times for Grain v1, i.e., output 16 bit per clock, and up to 32 times for Grain-128 and Grain-128a.

The amount of required gates for Grain is comparable to other hardware implementations of stream ciphers, e.g., E0 or A5/1[65], and with 1294 gates for the internal state, Grain v1 is one with the lowest amount of required gates from the complete eSTREAM portfolio[60]. Regarding the complete algorithm, Grain v1 requires 1450 gate equivalents, Grain-128 requires 2133 gate equivalents, Grain-128a without authentication requires 2145 gate equivalents, and Grain-128 with authentication requires 2769 gate equivalents.

Energy Consumption The energy consumption for Grain ciphers is also very low, at a clocking speed of 100 kHz, Grain v1 only requires 3.3 μW, and 4.3 μW for Grain-128. To the best of the authors knowledge, no energy consumption measurements have been conducted for Grain-128a.

4.3 Security

All the Grain algorithms have been cryptoanalyzed frequently during the last ten years, and a lot of papers on the topic have been published. Among the attacks the analysts identified are side channel attacks, e.g., different fault attacks, as well as some algebraic attacks, e.g., cube attack.

In 2005 Khazaei et al. published a distinguishing attack[78] on Grain v0. Using this attack it was possible to distinguish the Grain keystream output from a completely random data stream with a complexity of $O(2^{61})$. Furthermore, Berbain et al. showed in 2006[20] a recovery attack against Grain v0 by exploiting linear approximations of the feedback and filter function to derive the LFSR bits and recover the initial state of the NLFSR and get knowledge of the key. This attack requires 2^{43} computations and 2^{38} keystream bits to determine the 80 bit key. These attacks lead the authors of the initial Grain version to release Grain v1 and Grain-128 with changed output and feedback functions.

Slide resynchronization attack In 2006[82] zgl Kk presented an attack on the initialization of the of Grain v1, where they tried to find related keys and IVs. Although it did not result in an efficient key recovery attack, it showed a weakness in the initialization phase. De Cannire, Kk, and Preneel extended this research and published their results in 2008[38], where they pointed out that the existence of the sliding property in the initialization part of the Grain ciphers can be used to reduce the cost of an exhaustive key search by half. Lee, Yuseop, et al. presented 2008[84] a related-key chosen IV attack based on the slide resynchronization attack against Grain-v1 and Grain-128. This was the first recovery attack and was able to recover the secret key with $2^{22.59}$ chosen IVs, $2^{26.29}$ bit keystream sequences and $2^{22.90}$ computational complexity for Grain v1. To recover the secret key of Grain-128, the attack required $2^{26.59}$ chosen IVs, $2^{31.39}$ bit keystream sequences, and $2^{27.01}$ computational complexity. In 2013 Ding et al.[39] and Banik et al.[17] independently used the related-key chosen IV attack on Grain-128a and showed that the attack is better than an exhaustive key search.

Dynamic Cube Attack The cube attack is an efficient mathematical known plaintext attack, very similar to the distinguishing attack. It is able to fully recover the key and was initially introduced by Dinur and Shamir[43] in 2009, and soon efficiently implemented on a FPGA by Aumasson et al. in 2009[5]. An extrapolation of their results suggested, that it is possible to conduct a distinguishing attack on Grain-128 in time 2^{83}, which is below the 2^{128} complexity for an exhaustive search. Dinur, Itai and Shamir presented a new variant of the cube attack on Grain-128 in 2011[44], which they called the dynamic cube attack. This attack recovers the secret key by exploiting distinguishers obtained from cube testers. They were able to recover the full key of Grain-128, but only when it belongs to a subset of 2^{-10} of the possible keys. Nonetheless this attack is faster than an exhaustive search over the 2^{118} possible keys by a factor of about 2^{15}. This method has also been implemented against Grain v1 in 2013 by Rahimi and Barmshory[102]. They were able to to recover the full key in feasible time complexity with decreased initialization rounds. This attack is faster than exhaustive search by a factor of 2^{32}.

Fault Attacks A fault attack is a side channel attack, where the adversary induces faults in states or operations of the cipher and then analyzes the faulty ciphertext in comparison to a ciphertext without faults to break the system. The first fault attack has been presented by Berzati et al. in 2009[25] against Grain-128, Banik et al.[15] extended this work to other cipher, and showed 2012[14] a differential fault attack on Grain-128a using MACs. Most of the previous work targets the LFSR, but recently Karmakar and Chowdhury showed[76], that both LFSR and NLFSR can be attacked by an adversary and concluded, that both registers have to be protected.

Summary Although this section makes no claim of completeness, we showed attacks on the Grain family of stream ciphers. More information regarding Grain security can be found in these papers[29] [88] [116] [4] [117] [77] [36] [110] [41]. The mathematical attack based on the dynamic cube attack showed by Aumasson et al.[5] and the fault attacks are the only known weaknesses of Grain. Nontheless Grain-128a is the most secure member of the family, since Michael Lehmann and Willi Meier suggested[85] that it seems to be immune against dynamic cube and differential attacks.

5 Trivium

5.1 Specifications

In 2005 Christophe De Cannière and Bart Preneel developed a synchronous stream cipher called Trivium [37,32]. It became part of the eSTREAM portfolio and was designed for resource constrained environments with high performance requirements. Beside efficiency concerning speed and area a plain and simple architecture was also an important aspect in the design process. Just like Grain, the cipher consists of two phases which are explained in more detail in subsequent sections. The first phase deals with the initialization of the algorithm where the secret state of the algorithm is determined using a secret key and an initialization vector (IV). In the second phase the keystream generation process is executed by the algorithm using the data gathered in the first phase. The whole process produces keystreams with a maximum length of 2^{64} bits.

Setup phase In this phase the secret state (S) of the cipher is defined and stored using three Non-Linear Feedback Shift Register (NLFSR) of different capacities (93, 84, 111). The 93 bit NLFSR holds the secret key (SK) of the cipher. The remaining 13 bits of the register are initialized with zero and concatenated with the secret key bits:

$$(S_1, S_2, \ldots, S_{93}) = (SK_1, SK_2, \ldots, SK_{80}, 0, ..., 0)$$

As second component of the secret state the 80-bit initialization vector (IV) is used. The latter is stored in the 84 bit NLFSR representing the secret state positions from S_{94} to S_{173}. The four remaining bits are set to zero:

$$(S_{94}, S_{95}, ..., S_{177}) = (IV_1, IV_2, ..., IV_{80}, 0, ..., 0)$$

The lower section of the secret state i.e. 111 bit NLFSR is initialized with zero while the last three bits ($S_{286}, S_{287}, S_{288}$) are set to one:

$$(S_{178}, S_{179}, ..., S_{286}, S_{287}, S_{288}) = (0, 0, ..., 1, 1, 1)$$

Afterwards the secret state is further processed by executing several operations like AND, XOR and bit shifts. The following steps are repeated 1152 times in order to produce a secret state, which is required later to generate the keystream. In the first step certain state bits are associated in a predefined manner where $\oplus$ denotes the XOR operation and $\wedge$ denotes the AND operation. The results are stored in particular variables T_1, T_2 and T_3. Then the whole state is updated in each iteration.

```
1: **for** $i = 1$ to 1152 **do**
2:     $T_1 = S_{66} \oplus S_{91} \wedge S_{92} \oplus S_{93} \oplus S_{171}$
3:     $T_2 = S_{162} \oplus S_{175} \wedge S_{176} \oplus S_{177} \oplus S_{264}$
4:     $T_3 = S_{243} \oplus S_{286} \wedge S_{287} \oplus S_{288} \oplus S_{69}$
5:
6:     $(S_1, S_2, \ldots, S_{93}) = (T_3, S_1, \ldots, S_{92})$
7:     $(S_{94}, S_{95}, \ldots, S_{177}) = (T_1, S_{94}, \ldots, S_{176})$
8:     $(S_{178}, S_{179}, \ldots, S_{288}) = (T_2, S_{178}, \ldots, S_{287})$
9: **end for**
```

Key stream generation phase The key stream generation process is similar to the cipher initialization process described in the setup phase. Based on the current state (S) a keystream of length N (where $N \leq 2^{64}$) is generated by repeating the subsequent instructions one after the other up to N times. The first instructions consist of three XOR operations which are performed on three pairs of specific state bits, respectively. Each result is assigned to a particular variable T_1, T_2 and T_3. Afterwards one bit of keystream (denoted by Z_i) is generated for each iteration i as a result of a linear combination of T_1, T_2 and T_3.

After a keystream Z is generated, it can be used to perform encryption and decryption operations. The structure of Trivium is shown in Figure 7.

```
1: for $i = 1$ to $N$ do
2:     $T_1 = S_{66} \oplus S_{93}$
3:     $T_2 = S_{162} \oplus S_{177}$
4:     $T_3 = S_{243} \oplus S_{288}$
5:
6:     $Z_i = T_1 \oplus T_2 \oplus T_3$
7:
8:     $T_1 = T_1 \oplus S_{91} \wedge S_{92} \oplus S_{171}$
9:     $T_2 = T_2 \oplus S_{175} \wedge S_{176} \oplus S_{264}$
10:    $T_3 = T_3 \oplus S_{286} \wedge S_{287} \oplus S_{69}$
11:
12:    $(S_1, S_2, \ldots, S_{93}) = (T_3, S_1, \ldots, S_{92})$
13:    $(S_{94}, S_{95}, \ldots, S_{177}) = (T_1, S_{94}, \ldots, S_{176})$
14:    $(S_{178}, S_{179}, \ldots, S_{288}) = (T_2, S_{178}, \ldots, S_{287})$
15: end for
```

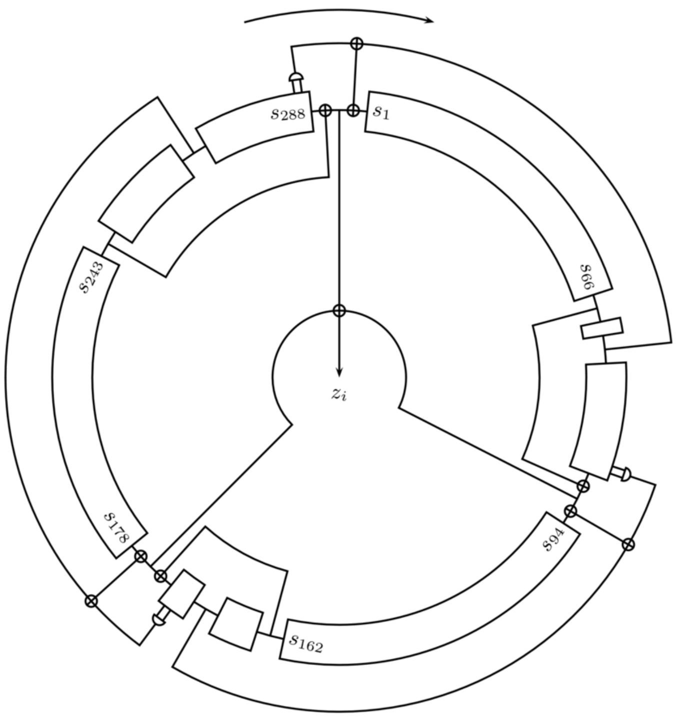

Fig. 7. Trivium structure (copyrighted free use, `https://commons.wikimedia.org/w/index.php?curid=598665`)

5.2 Hardware Implementation

Even though Trivium is primary considered as hardware oriented cipher it can be implemented in software as well. Its default setting (containing three AND gates, several XOR gates and three NLFSR) is able to produce one bit of keystream per cycle. In [90] a low power implementation of Trivium based on logic parallelization is suggested. First, the authors split each NLFSR up in two single registers with half size, respectively. One group of registers stores data based on a transition from zero to one in clock signal. The second group reacts to a transition from one to zero for storing data. This step divides the primary frequency used for one shift register by half. Afterwards the authors used additional logic (multiplexers) in combination with clock signal for selecting the right bits out of the different registers. They also considered loading the secret key and the initialization vector in parallel. For verifing results they implemented both standard and low power version of Trivium in Very High Speed Integrated Circuit Hardware Description Language (VHDL) and used three different kind of Complementary metal-oxide-semiconductor technologies (CMOS) for comparison. Martin Feldhofer [54] describes an optimized low power implementation of Trivium for usage in Radio-frequency identification (RFID) tags. In order to keep the clock frequency and threfore the (dynamic) power consumption as low as possible he divided all Flip-flops (using to store state bits) in 19×16 bit registers to reduce the number of elements beeing active at the same time.

5.3 Energy Consumption

Running standard Trivium with 100 kHz clock frequency it consumes about 5.6 μW energy [60]. Other energy consumption profiles for the different Trivium configurations can be seen in Table 6.

5.4 Security

Trivium is still an object of various security analysis. Despite its simplistic architecture it hardly discloses information about internal components like its secret state. To reveal as much information as possible a wide range of approaches and methods have been established so far. One obvious and inefficent approach is exhaustive key search. A more sophisticated way is found in [86] where both authors explain how to recover the secret key assuming some parts (about $2^{61.5}$ bits) of the keystream are known before. They propose increasing the size of the internal state rather than increasing the size of the secret key to provide an effective protection against their recovery attack. Another kind of key recovery attack is described by Fischer et al. [56]. In this paper the authors perform a key recovery attack on a reduced version of Trivium with a lower initialization complexity (using a total of 672 rounds). However there is no advantage over exhaustive search when using this key recovery attack on Trivium with 1152 initialization steps. A better performance is achieved when using the concept of cube attacks as described in [43]. Therein the concept is appropriated on reduced versions of Trivium, consisting 672, 735 and 767 initialization rounds, respectively. The former has an overall complexity of up to 2^{19} bits which is much better compared to the performance achieved by Fischer et al. The latter have complexities of 2^{30} and accordingly of 2^{45} bits.

6 Comparison

Table 6 contains the performance results for ASIC implementations of the discussed ciphers as well as AES for comparison. The WG-8 results are from [115]. The other results are from [62] which is linked on the official eSTREAM website. While we took the AES results also from [62], they are originally from [106] and [55]. Most results are for a 130nm CMOS process but the WG-8 implementation uses 65nm while the AES implementations use 110nm and 350nm. The different processes make reliable comparisons in power consumption impossible and may also affect the maximum frequency and thus the throughput. The area however, can be compared relatively well as it is given in GE (gate equivalence).

cipher	data rate [bits / cycle]	area [GE]	power [mW]	throughput [Mbps]	throughput / area	throughput / power	CMOS process
MICKEY 2.0	1	3188	8.701	454.5	0.14	52.2	130nm
MICKEY-128	1	5039	12.512	413.2	0.08	33.0	130nm
WG-8	1	1786	0.983	500	0.28	508.6	65nm
	11	3942	1.344	6710	1.70	4992.6	65nm
Grain v1	1	1294	7.772	724.6	0.56	93.2	130nm
	16	3239	11.929	9876.5	3.05	827.9	130nm
Grain-128	1	1857	15.283	925.9	0.50	60.6	130nm
	32	4617	15.093	14479.6	3.14	959.4	130nm
Trivium	1	2580	5.618	327.9	0.13	58.4	130nm
	64	4921	12.677	22299.6	4.53	1759.1	130nm
AES[106]	2.37	5398	-	311	0.06	-	110nm
AES[55]	0.124	3400	-	10	0.003	-	350nm

Table 6. Performance of ASIC implementations of different ciphers. ([62], [115])

Area, Power and Throughput of single-bit versions The fastest cipher is Grain-128 and the smallest is Grain v1. Grain v1 also has the highest througput/area and throughput/power values. For throughput/area, WG-8 is average while the MICKEY ciphers and Trivium make up the lower end. The power consumption of Grain v1, Trivium and MICKEY 2.0 is rather similar. The 128-bit ciphers Grain-128 and MICKEY-128 both require significantly more power than their 80-bit counterparts. In [115] the authors claim that WG-8 is very power efficient, requiring less than half the power of Grain v1 or Trivium. However, we have no means to verify that claim, as the power values cannot be compared due to the different architectures.

All ciphers are faster and smaller than AES which was a requirement for hardware-oriented eSTREAM finalists.

Parallelization The best cipher for parallization is definitely Trivium. Its throughput can be increased by a factor of 68 while only doubling the required area. WG-8, Grain v1 and Grain-128 can also be parallelized fairly well. While increasing the area by a factor of 2-3, the throughput can be increased by a factor of 13-16. Only MICKEY 2.0 cannot be efficiently parallelized. [108] shows an implementation that generates 2 bits per cycle but it requires much more space and even has lower throughput/area than the single-bit implementation.

In contrast to real stream ciphers, AES produces bursts of keystream bits every few cycles. The implementation from [106] shows an average of 2.37 bits per cycle, which is significantly less than what most stream ciphers can achieve. However, it is unknown to us, by how much this rate can be increased and at what cost.

Simplicity In [62] the numbers of code lines for each cipher are given. These numbers are used to rank the ciphers by simplicity. A cipher is considered to be simpler (i.e. easier to implement) if its implementation requires less lines of code. The fewest lines of code are needed for MICKEY-128 (127), followed by Grain128 (138), MICKEY 2.0 (149), Grain v1 (158) and Trivium (159). For a given key size, the MICKEY ciphers need less code than their competitors. In other words, the MICKEY ciphers are the easiest to implement. We have no information about the number of code lines required to implement WG-8 or AES.

Security All eSTREAM finalists were analyzed thoroughly during and after the election process. A number of serious weaknesses were found in Grain v1 which should no longer be used if possible. Both Grain-128 and MICKEY 2.0 are vulnerable only to side-channel attacks. Nevertheless, some of Grain's weaknesses also apply to Grain-128 and it is recommended to use Grain-128a instead. The most secure of the eSTREAM finalists appears to be Trivium as all known attacks are only a threat to implementations with a reduced number of initialization cycles.

For WG-8 the only existing attack can recover the entire key but only works in a highly unlikely scenario. However, the mere existence of this attack as well as attacks on its predecessor WG-7 cast doubts on WG-8's security. While the cipher is interesting for its provable cryptographic properties, more research is required before the cipher can be recommended for use in productive environments.

It should be noted that AES is older and used in more scenarios than any of the ciphers presented in this paper. As a consequence its security has been analyzed more thoroughly. This should be considered in applications where security is much more important than other aspects.

Summary Due to security concerns, both Grain v1 and WG-8 should not be used in practice even though Grain v1 has excellent performance and WG-8's performance is solid average. Of the remaining ciphers, Grain-128 has the best scores for throughput/area and throughput/power for single-bit implementations and is thus a very good choice if only one bit is required in each cycle. If a parallel implementation is needed, Trivium is clearly the best choice. It also appears to be the most secure cipher. MICKEY 2.0 is overall inferior to Grain and Trivium, especially because it cannot be efficiently parallelized. Though it has the advantage of a simple implementation, we expect it to be less used in practice.

All presented ciphers outperform AES in area and speed, as required by the eSTREAM project. However, their security is not as thoroughly analyzed as that of AES.

7 Conclusion

In this paper we introduced four hardware-oriented stream ciphers, namely the eSTREAM finalists Grain, Trivium and MICKEY 2.0 as well as WG-8, a recent member of the WG stream cipher family. We discussed their specifications, different versions, hardware implementations and security aspects. After comparing the ciphers with each other, we came to the conclusion that most of them are suitable for resource constrained devices. The only ciphers we cannot recommend are Grain v1, because of its well-known security issues and WG-8 because it is quite new and as we have shown, there are reasons to doubt its security. Nevertheless, we think WG-8 should be analyzed further as the cipher has some interesting cryptographic properties and decent performance.

For most applications, Grain-128 provides the best overall performance. For high throughput demands, Trivium is the better choice because it can be highly efficiently parallelized. MICKEY 2.0 has the simplest implementation but its performance is inferior to the other ciphers.

We believe that the search for fast, small and secure stream ciphers, that was started by eSTREAM, is not finished yet. New and interesting ciphers, like WG-8, show that there is still a lot of research potential in the field.

References

1. Aagaard, M.D., Gong, G., Mota, R.K.: Hardware implementations of the WG-5 cipher for passive RFID tags. In: 2013 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST). pp. 29–34
2. Ahmadi, H., Eghlidos, T.: Heuristic guess-and-determine attacks on stream ciphers. IET Information Security 3(2), 66–73 (2009)
3. AlFardan, N.J., Bernstein, D.J., Paterson, K.G., Poettering, B., Schuldt, J.C.N.: On the Security of RC4 in TLS. In: Proceedings of the 22nd USENIX Conference on Security. pp. 305–320. SEC'13, USENIX Association (2013)

4. ALMashrafi, M., Bartlett, H., Simpson, L., Dawson, E., Wong, K.K.H.: Analysis of indirect message injection for mac generation using stream ciphers. In: Information security and privacy. pp. 138–151. Springer (2012)
5. Aumasson, J.P., Dinur, I., Henzen, L., Meier, W., Shamir, A.: Efficient fpga implementations of high-dimensional cube testers on the stream cipher grain-128. SHARCS09 Special-purpose Hardware for Attacking Cryptographic Systems p. 147 (2009)
6. Babbage, S.H.: Improved "exhaustive search" attacks on stream ciphers. In: Security and Detection, 1995., European Convention on. pp. 161–166 (1995)
7. Babbage, S., de Cannière, C., Canteaut, A., Cid, C., Gilbert, H., Johansson, T., Parker, M., Preneel, B., Rijmen, V., Robshaw, M.: The eSTREAM Portfolio (2008)
8. Babbage, S., Dodd, M.: The stream cipher MICKEY-128 2.0 (17082006)
9. Babbage, S., Dodd, M.: The stream cipher MICKEY (version 1): Algorithm specification issue 1.0 (2005)
10. Babbage, S., Dodd, M.: The stream cipher MICKEY 2.0 (2006)
11. Babbage, S., Dodd, M.: The MICKEY Stream Ciphers. In: Robshaw, M., Billet, O. (eds.) New Stream Cipher Designs, Lecture Notes in Computer Science, vol. 4986, pp. 1–16. Springer Berlin Heidelberg (Heidelberg : Springer Berlin Heidelberg, 2008)
12. Banegas, G.: Attacks in Stream Ciphers: A Survey. IACR Cryptology ePrint Archive 2014, 677 (2014)
13. Banik, S., Maitra, S.: A Differential Fault Attack on MICKEY 2.0. In: Bertoni, G., Coron, J.S. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2013, Lecture Notes in Computer Science, vol. 8086, pp. 215–232. Imprint: Springer (Heidelberg : Imprint: Springer, 2013)
14. Banik, S., Maitra, S., Sarkar, S.: A differential fault attack on grain-128a using macs. In: Security, Privacy, and Applied Cryptography Engineering, pp. 111–125. Springer (2012)
15. Banik, S., Maitra, S., Sarkar, S.: A differential fault attack on the grain family of stream ciphers. In: Cryptographic Hardware and Embedded Systems–CHES 2012, pp. 122–139. Springer (2012)
16. Banik, S., Maitra, S., Sarkar, S.: Improved differential fault attack on MICKEY 2.0. Journal of Cryptographic Engineering 5(1), 13–29 (2015)
17. Banik, S., Maitra, S., Sarkar, S., Sönmez, T.M.: A chosen iv related key attack on grain-128a. In: Information Security and Privacy. pp. 13–26. Springer (2013)
18. Bellare, M., Namprempre, C.: Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In: Advances in CryptologyASIACRYPT 2000, pp. 531–545. Springer (2000)
19. Berbain, C., Billet, O., Canteaut, A., Courtois, N., Gilbert, H., Goubin, L., Gouget, A., Granboulan, L., Lauradoux, C., Minier, M., Pornin, T., Sibert, H.: Sosemanuk, a Fast Software-Oriented Stream Cipher. In: Robshaw, M., Billet, O. (eds.) New Stream Cipher Designs, Lecture Notes in Computer Science, vol. 4986, pp. 98–118. Springer Berlin Heidelberg (Heidelberg : Springer Berlin Heidelberg, 2008)
20. Berbain, C., Gilbert, H., Maximov, A.: Cryptanalysis of grain. In: Fast Software Encryption. pp. 15–29. Springer (2006)
21. Bernstein, D.J.: `https://competitions.cr.yp.to/`
22. Bernstein, D.J.: Which estream ciphers have been broken? Ecrypt report 10, 2008 (2008)
23. Bernstein, D.J.: Which phase-3 eSTREAM ciphers provide the best software speeds? (2008)

24. Bernstein, D.J.: The Salsa20 Family of Stream Ciphers. In: Robshaw, M., Billet, O. (eds.) New Stream Cipher Designs, Lecture Notes in Computer Science, vol. 4986, pp. 84–97. Springer Berlin Heidelberg (Heidelberg : Springer Berlin Heidelberg, 2008)
25. Berzati, A., Canovas, C., Castagnos, G., Debraize, B., Goubin, L., Gouget, A., Paillier, P., Salgado, S.: Fault analysis of grain-128. In: Hardware-Oriented Security and Trust, 2009. HOST'09. IEEE International Workshop on. pp. 7–14. IEEE (2009)
26. Biryukov, A., Shamir, A.: Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers. In: Okamoto, T. (ed.) Advances in Cryptology ASIACRYPT 2000 [Elektronische Ressource], Lecture Notes in Computer Science, vol. 1976, pp. 1–13. Springer-Verlag Berlin Heidelberg (Heidelberg : Springer-Verlag Berlin Heidelberg, 2000)
27. Biryukov, A., Shamir, A., Wagner, D.: Real Time Cryptanalysis of A5/1 on a PC. In: Goos, G., Hartmanis, J., Leeuwen, J., Schneier, B. (eds.) Fast Software Encryption, Lecture Notes in Computer Science, vol. 1978, pp. 1–18. Springer-Verlag Berlin Heidelberg (Heidelberg : Springer-Verlag Berlin Heidelberg, 2001)
28. Biryukov, A., Wagner, D.: Slide attacks. In: International Workshop on Fast Software Encryption. pp. 245–259 (1999)
29. Bjørstad, T.: Cryptanalysis of grain using time/memory/data tradeoffs (2013)
30. Bluetooth Special Interest Group: Specification of the Bluetooth System, Volume 1, Version 1.0 B (1999)
31. Boesgaard, M., Vesterager, M., Pedersen, T., Christiansen, J., Scavenius, O.: Rabbit: A New High-Performance Stream Cipher. In: Johansson, T. (ed.) Fast Software Encryption, Lecture Notes in Computer Science, vol. 2887, pp. 307–329. Springer-Verlag Berlin Heidelberg (Heidelberg : Springer-Verlag Berlin Heidelberg, 2003)
32. Canniere, C.D., Preneel, B.: Trivium specifications. citeseer. ist. psu. edu/734144. html (2005)
33. Chakraborty, A., Mukhopadhyay, D.: A Practical Template Attack on MICKEY-128 2.0 Using PSO Generated IVs and LS-SVM
34. Cid, C., Robshaw, M.J.B.: The eSTREAM Portfolio in 2012: ECRYPT II report D.SYM.10. ECRYPT II (2012)
35. Daemen, J., Clapp, C.: Fast Hashing and Stream Encryption with Panama. In: Vaudenay, S. (ed.) Fast Software Encryption, Lecture Notes in Computer Science, vol. 1372, pp. 60–74. Springer-Verlag Berlin Heidelberg (Heidelberg : Springer-Verlag Berlin Heidelberg, 1998)
36. Datta, P., Roy, D., Mukhopadhyay, S.: A probabilistic algebraic attack on the grain family of stream ciphers. In: Network and System Security, pp. 558–565. Springer (2014)
37. De Cannière, C.: Trivium: A stream cipher construction inspired by block cipher design principles. In: Information Security, pp. 171–186. Springer (2006)
38. De Cannière, C., Kücük, Ö., Preneel, B.: Analysis of grains initialization algorithm. In: Progress in Cryptology–AFRICACRYPT 2008, pp. 276–289. Springer (2008)
39. Ding, L., Guan, J.: Related key chosen iv attack on grain-128a stream cipher. Information Forensics and Security, IEEE Transactions on 8(5), 803–809 (2013)
40. Ding, L., Jin, C., Guan, J., Wang, Q.: Cryptanalysis of Lightweight WG-8 Stream Cipher. IEEE Transactions on Information Forensics and Security 9(4), 645–652 (2014)

41. Dinur, I., Güneysu, T., Paar, C., Shamir, A., Zimmermann, R.: An experimentally verified attack on full grain-128 using dedicated reconfigurable hardware. In: Advances in Cryptology–ASIACRYPT 2011, pp. 327–343. Springer (2011)
42. Dinur, I., Shamir, A.: Cube Attacks on Tweakable Black Box Polynomials. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 278–299. Springer Berlin Heidelberg (2009), `http://link.springer.com/content/pdf/10.1007%2F978-3-642-01001-9_16.pdf`
43. Dinur, I., Shamir, A.: Cube attacks on tweakable black box polynomials. In: Advances in Cryptology-EUROCRYPT 2009, pp. 278–299. Springer (2009)
44. Dinur, I., Shamir, A.: Breaking grain-128 with dynamic cube attacks. In: Fast Software Encryption. pp. 167–187. Springer (2011)
45. Dinur, I., Shamir, A.: Applying cube attacks to stream ciphers in realistic scenarios. Cryptography and Communications 4(3-4), 217–232 (2012)
46. ECRYPT Stream Cipher Project: Call for Stream Cipher Primitives: Version 1.3, 12.04.2015, `http://www.ecrypt.eu.org/stream/call`
47. Eisenbarth, T., Kumar, S., Paar, C., Poschmann, A., Uhsadel, L.: A Survey of Lightweight-Cryptography Implementations. IEEE Design & Test of Computers 24(6), 522–533 (2007)
48. Ekdahl, P., Johansson, T.: SNOW - a new stream cipher. In: Proceedings of First Open NESSIE Workshop, KU-Leuven. pp. 167–168 (2000)
49. Ekdahl, P., Johansson, T.: A New Version of the Stream Cipher SNOW. In: Nyberg, K., Heys, H. (eds.) Selected Areas in Cryptography, Lecture Notes in Computer Science, vol. 2595, pp. 47–61. Springer-Verlag Berlin Heidelberg (Heidelberg : Springer-Verlag Berlin Heidelberg, 2003)
50. El-Razouk, H., Reyhani-Masoleh, A., Gong, G.: New Implementations of the WG Stream Cipher. IEEE Transactions on Very Large Scale Integration (VLSI) Systems 22(9), 1865–1878 (2014)
51. Fan, X., Gong, G.: Specification of the stream cipher wg-16 based confidentiality and integrity algorithms. University of Waterloo, Waterloo, ON, Canada, Tech. Rep. CACR 6, 2013 (2013)
52. Fan, X., Mandal, K., Gong, G.: WG-8: A Lightweight Stream Cipher for Resource-Constrained Smart Devices. In: Singh, K., Awasthi, A.K. (eds.) Quality, Reliability, Security and Robustness in Heterogeneous Networks, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol. 115, pp. 617–632. Springer Berlin Heidelberg (2013)
53. Feldhofer, M.: Comparison of low-power implementations of Trivium and Grain. In: Workshop on The State of the Art of Stream Ciphers (SASC2007) pages. pp. 236–246 (2007)
54. Feldhofer, M.: Comparison of low-power implementations of trivium and grain. In: The State of the Art of Stream Ciphers, Workshop Record. pp. 236–246 (2007)
55. Feldhofer, M., Wolkerstorfer, J., Rijmen, V.: Aes implementation on a grain of sand. IEE Proceedings-Information Security 152(1), 13–20 (2005)
56. Fischer, S., Khazaei, S., Meier, W.: Chosen iv statistical analysis for key recovery attacks on stream ciphers. In: Progress in Cryptology–AFRICACRYPT 2008, pp. 236–245. Springer (2008)
57. Golić, J.D.: A Weakness of the Linear Part of Stream Cipher MUGI. In: Roy, B., Meier, W. (eds.) Fast Software Encryption, Lecture Notes in Computer Science, vol. 3017, pp. 178–192. Springer-Verlag Berlin Heidelberg (Heidelberg : Springer-Verlag Berlin Heidelberg, 2004)
58. Gong, G., Youssef, A.M.: On welch-gong transformation sequence generators. In: Selected Areas in Cryptography. pp. 217–232. Springer (2000)

59. Gong, G., Youssef, A.M.: Cryptographic properties of the welch-gong transformation sequence generators. Information Theory, IEEE Transactions on 48(11), 2837–2846 (2002)
60. Good, T., Benaissa, M.: Hardware results for selected stream cipher candidates. State of the Art of Stream Ciphers pp. 191–204 (2007)
61. Good, T., Benaissa, M.: Hardware performance of eStream phase-III stream cipher candidates. In: Proc. of Workshop on the State of the Art of Stream Ciphers (SACS'08) (2008)
62. Good, T., Benaissa, M.: Hardware performance of estream phase-iii stream cipher candidates. In: Proc. of Workshop on the State of the Art of Stream Ciphers (SACS08) (2008)
63. gren, M., Hell, M., Johansson, T., Meier, W.: Grain-128a: a new version of grain-128 with optional authentication. International Journal of Wireless and Mobile Computing 5(1), 48–59 (2011)
64. Hell, M., Johansson, T., Maximov, A., Meier, W.: A stream cipher proposal: Grain-128. In: IEEE International Symposium on Information Theory (ISIT 2006). Citeseer (2006)
65. Hell, M., Johansson, T., Meier, W.: Grain: a stream cipher for constrained environments. International Journal of Wireless and Mobile Computing 2(1), 86–93 (2007)
66. Hellman, M.: A cryptanalytic time-memory trade-off. IEEE Transactions on Information Theory 26(4), 401–406 (1980)
67. Hermelin, M., Nyberg, K.: Correlation Properties of the Bluetooth Combiner. In: Song, J. (ed.) Information Security and Cryptology - ICISC99, Lecture Notes in Computer Science, vol. 1787, pp. 17–29. Springer-Verlag Berlin Heidelberg (Heidelberg : Springer-Verlag Berlin Heidelberg, 2000)
68. Hoch, J.J., Shamir, A.: Fault analysis of stream ciphers. In: International Workshop on Cryptographic Hardware and Embedded Systems. pp. 240–253 (2004)
69. Hong, J., Kim, W.H.: TMD-Tradeoff and State Entropy Loss Considerations of Streamcipher MICKEY. In: Maitra, S., Venkatesan, R., Veni Madhavan, C.E. (eds.) Progress in Cryptology - INDOCRYPT 2005, Lecture Notes in Computer Science, vol. 3797, pp. 169–182. Springer Berlin Heidelberg (Heidelberg : Springer Berlin Heidelberg, 2005)
70. Hu, Y., Xiao, G.: Resilient functions over finite fields. Information Theory, IEEE Transactions on 49(8), 2040–2046 (2003)
71. Hwang, D., Chaney, M., Karanam, S., Ton, N., Gaj, K.: Comparison of FPGA-targeted hardware implementations of eSTREAM stream cipher candidates. The State of the Art of Stream Ciphers pp. 151–162 (2008)
72. Jana, S., Bhaumik, J., Maiti, M.K.: Survey on lightweight block cipher. International Journal of Soft Computing and Engineering 3, 183–187 (2013)
73. Joux, A., Muller, F.: A chosen iv attack against turing. In: International Workshop on Selected Areas in Cryptography. pp. 194–207 (2003)
74. Junrong, L., Dawu, G., Zheng, G.: Correlation Power Analysis Against Stream Cipher MICKEY v2. In: Computational Intelligence and Security (CIS), 2010 International Conference on. pp. 320–324 (2010)
75. Karmakar, S., Chowdhury, D.R.: Differential Fault Analysis of MICKEY-128 2.0. In: Fault Diagnosis and Tolerance in Cryptography (FDTC), 2013 Workshop on. pp. 52–59 (2013)
76. Karmakar, S., Chowdhury, D.R.: Fault analysis of grain family of stream ciphers. IACR Cryptology ePrint Archive 2014, 261 (2014)

77. Karmakar, S., Chowdhury, D.R.: A generic scan attack on hardware based estream winners. IACR Cryptology ePrint Archive 2014, 263 (2014)
78. Khazaei, S., Hassanzadeh, M., Kiaei, M.: Distinguishing attack on grain (2005)
79. Kitsos, P., Sklavos, N., Provelengios, G., Skodras, A.N.: FPGA-based performance analysis of stream ciphers ZUC, Snow3g, Grain V1, Mickey V2, Trivium and E0. Microprocessors and microsystems 37(2), 235–245 (2013)
80. Klein, A.: Attacks on the RC4 stream cipher. Designs, Codes and Cryptography 48(3), 269–286 (2008)
81. de Koning Gans, G., Hoepman, J.H., Garcia, F.D.: A Practical Attack on the MIFARE Classic. In: Grimaud, G., Kanade, T., Kittler, J., Kleinberg, J.M., Mattern, F., Mitchell, J.C., Naor, M., Nierstrasz, O., Pandu Rangan, C., Steffen, B., Sudan, M., Terzopoulos, D., Tygar, D., Vardi, M.Y., Weikum, G., Standaert, F.X. (eds.) Smart Card Research and Advanced Applications, Lecture Notes in Computer Science, vol. 5189, pp. 267–282. Springer Berlin Heidelberg (Heidelberg : Springer Berlin Heidelberg, 2008)
82. Küçük, Ö.: Slide resynchronization attack on the initialization of grain 1.0. eSTREAM, ECRYPT Stream Cipher Project, Report 44, 2006 (2006)
83. Lee, J., Kapitanova, K., Son, S.H.: The price of security in wireless sensor networks. Computer Networks 54(17), 2967–2978 (2010)
84. Lee, Y., Jeong, K., Sung, J., Hong, S.: Related-key chosen iv attacks on grain-v1 and grain-128. In: Information Security and Privacy. pp. 321–335. Springer (2008)
85. Lehmann, M., Meier, W.: Conditional differential cryptanalysis of grain-128a. In: Cryptology and Network Security, pp. 1–11. Springer (2012)
86. Maximov, A., Biryukov, A.: Two trivial attacks on trivium. In: Selected Areas in Cryptography. pp. 36–55. Springer (2007)
87. Menezes, A.J., van Oorschot, P.C., Vanstone, S.A.: Handbook of applied cryptography Alfred J. Menezes ; Paul C. van Oorschot ; Scott A. Vanstone. The CRC Press series on discrete mathematics and its applications, CRC Press, Boca Raton, Florida, USA (1997)
88. Mihaljevic, M.J., Gangopadhyay, S., Paul, G., Imai, H.: Internal state recovery of grain-v1 employing normality order of the filter function. Information Security, IET 6(2), 55–64 (2012)
89. Mirzaei, A., Dakhilalian, M., Modarres-Hashemi, M.: An Improved Attack on WG Stream Cipher. IJCSNS International Journal of Computer Science and Network Security 10(4), 45–52 (2010)
90. Mora-Gutiérrez, J., Jiménez-Fernández, C., Valencia-Barrero, M.: Low power implementation of trivium stream cipher. In: Integrated Circuit and System Design. Power and Timing Modeling, Optimization and Simulation, pp. 113–120. Springer (2013)
91. Mota, R.K.: Role of cryptographic welch-gong (wg-5) stream cipher in rfid security (2012)
92. Nawaz, Y., Gong, G.: Preventing chosen iv attack on wg cipher by increasing the length of key/iv setup. ECRYPT Stream Cipher Project Report 2005 47 (2005)
93. Nawaz, Y., Gong, G.: The WG Stream Cipher. In: Proceedings of the ECRYPT State of the Art of Stream Ciphers. vol. 033 (2005)
94. Nawaz, Y., Gong, G.: WG: A family of stream ciphers with designed randomness properties. Information Sciences 178(7), 1903–1916 (2008)
95. No, J.S., Golomb, S.W., Gong, G., Lee, H.K., Gaal, P.: Binary pseudorandom sequences of period 2 n-1 with ideal autocorrelation. Information Theory, IEEE Transactions on 44(2), 814–817 (1998)

96. Nyberg, K., Wallén, J.: Improved Linear Distinguishers for SNOW 2.0. In: Robshaw, M. (ed.) Fast Software Encryption, Lecture Notes in Computer Science, vol. 4047, pp. 144–162. Springer Berlin Heidelberg (Heidelberg : Springer Berlin Heidelberg, 2006)
97. Orumiehchiha, M.A., Pieprzyk, J., Steinfeld, R.: Cryptanalysis of WG-7: A lightweight stream cipher. Cryptography and Communications 4(3-4), 277–285 (2012)
98. Paar, C., Pelzl, J.: Understanding Cryptography. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)
99. Pasalic, E.: On guess and determine cryptanalysis of LFSR-based stream ciphers. IEEE Transactions on Information Theory 55(7), 3398–3406 (2009)
100. Popov, A.: Prohibiting RC4 Cipher Suites (2015), `http://www.ietf.org/rfc/rfc7465.txt`
101. Priemuth-Schmid, D., Biryukov, A.: Slid pairs in Salsa20 and Trivium. In: International Conference on Cryptology in India. pp. 1–14 (2008)
102. Rahimi, M., Barmshory, M., Mansouri, M.H., Aref, M.R.: Dynamic cube attack on grain-v1. IACR Cryptology ePrint Archive 2013, 268 (2013)
103. Rønjom, S., Helleseth, T.: Attacking the filter generator over gf (2 m). In: Arithmetic of Finite Fields, pp. 264–275. Springer (2007)
104. Rose, G.G., Hawkes, P.: On the Applicability of Distinguishing Attacks Against Stream Ciphers. IACR Cryptology ePrint Archive 2002, 142 (2002)
105. Saluja, K.K.: Linear feedback shift registers theory and applications. Department of Electrical and Computer Engineering, University of Wisconsin-Madison pp. 4–14 (1987)
106. Satoh, A., Morioka, S., Takano, K., Munetoh, S.: A compact rijndael hardware architecture with s-box optimization. In: Advances in CryptologyASIACRYPT 2001, pp. 239–254. Springer (2001)
107. Siegenthaler: Decrypting a Class of Stream Ciphers Using Ciphertext Only. IEEE Transactions on Computers C-34(1), 81–85 (1985)
108. Stefan, D., Mitchell, C.: On the Parallelization of the MICKEY-128 2.0 Stream Cipher. In: Proceedings of the ECRYPT State of the Art of Stream Ciphers. vol. 017 (2008)
109. Verdult, R.: Introduction to Cryptanalysis: Attacking Stream Ciphers
110. Wang, Y., Ding, L., Han, W., Wang, X.: The improved cube attack on grain-v1. IACR Cryptology ePrint Archive 2013, 417 (2013)
111. Watanabe, D., Furuya, S., Yoshida, H., Takaragi, K., Preneel, B.: A New Keystream Generator MUGI. In: Daemen, J., Rijmen, V. (eds.) Fast Software Encryption, Lecture Notes in Computer Science, vol. 2365, pp. 179–194. Springer-Verlag Berlin Heidelberg (Heidelberg : Springer-Verlag Berlin Heidelberg, 2002)
112. Wu, H.: The Stream Cipher HC-128. In: Robshaw, M., Billet, O. (eds.) New Stream Cipher Designs, Lecture Notes in Computer Science, vol. 4986, pp. 39–47. Springer Berlin Heidelberg (Heidelberg : Springer Berlin Heidelberg, 2008)
113. Wu, H., Preneel, B.: Chosen IV attack on stream cipher WG. ECRYPT Stream Cipher Project Report 45 (2005)
114. Wu, H., Preneel, B.: Resynchronization attacks on wg and lex. In: Fast Software Encryption. pp. 422–432. Springer (2006)
115. Yang, G., Fan, X., Aagaard, M., Gong, G.: Design space exploration of the lightweight stream cipher WG-8 for FPGAs and ASICs. In: Proceedings of the 8th Workshop on Embedded Systems Security, pp. 1–10. ACM (2013)
116. Zhang, B., Li, Z., Feng, D., Lin, D.: Near collision attack on the grain v1 stream cipher. In: Fast Software Encryption. pp. 518–538. Springer (2014)

117. Zhang, H., Wang, X.: Cryptanalysis of stream cipher grain family. IACR Cryptology ePrint Archive 2009, 109 (2009)