Stream ciphers in the IoT: Grain and Trivium

Lennart Diedrich[1], Lulzim Murati[1], and Alexander Wiesmaier[1,2]

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

Abstract. The Internet of Things has become very popular during the last years, and with it the widespread use of pervasive devices. These devices are often extremely resource constrained, and most of them require high performance in terms of security, latency and speed. Hardware based encryption ciphers are very suitable for these kind of resource constrained devices in order to provide the required confidentiality. Stream ciphers often use shift registers, which are easy to implement in hardware, and thus should be considered when looking for suitable algorithms. Grain and Trivium are two representative ciphers for this kind of data encryption, and hence well suited for the application in the Internet of Things. In this paper, we describe these two algorithms, show their features and possible drawbacks, and finally present a brief comparison.

Keywords: Grain, Trivium, comparison, lightweight cryptography, IoT

1 Introduction

Over the last years we have seen an extreme growth of data transfer, but also for the data itself. The internet as a communication technology has become very important, and is an integral part of our everyday live. New technologies simplified our access to the internet, it can be accessed nearly everywhere wirelessly. Furthermore, with the advent of the Internet of Things, where smart everyday things get connected and communicate with each other, confidentiality has become very important. But this wireless data communication is extreme susceptible to be compromised by malicious adversaries, and therefore this data communication has to be secured appropriately.

In order to protect privacy and confidentiality of data, cryptography is used. Modern cryptography can be classified into two major categories: symmetric key based methods, where the same cryptographic key is used for both encryption of plaintext and decryption of ciphertext, and asymmetric key based methods, where communicating parties do not share the same cryptographic key. Stream ciphers are member of the former group, and have recently received discussions about their legitimacy, since block ciphers can be easily turned into stream ciphers by using the OFB, CFB, OCB, or CTR encryption mode[39]. Furthermore block ciphers are well studied and received a lot of cryptoanalysis. Block ciphers, in contrast to stream ciphers, operate on a fixed size of data blocks, whereas stream ciphers operate on a per bit basis. Nonetheless, stream ciphers also have their advantages. Due to their low complexity they can be easily implemented in hardware at low cost. Additionally they proved to work very fast on bit streams, significantly faster than block ciphers. That is why they are the first choice for applications with a high throughput, but with low hardware and memory conditions.

Stream ciphers work similar to a one time pad (OTP) and encrypt each bit individually. The algorithm can be understood as a pseudorandom digit generator, i.e., a binary keystream generator. Real random number generators are very important in cryptography, but also hard to implement since they require an unpredictable and random source. This random source can be, e.g., thermal noise inside electrical conductors, or the timing and movement of a read/write head of a harddisk. The keystream required for stream ciphers needs to be a deterministic keystream, that means that given the same seed it always creates the same output. This generated keystream is then xored with the plaintext to generate the ciphertext. The decryption process is similarly done by xoring the identical keystream with the previously generated ciphertext to retrieve the plaintext.

1.1 Related Work

The topic of stream ciphers is a heavily researched area and many algorithms have been proposed over the years. In 1987 Ronald L. Rivest developed RC4 (Ron's Code 4), one of the most used stream ciphers, which was kept secret until an anonymous release of the sourcecode in 1994. RC4 is used in many software products, e.g., SSL, SSHv1 or the WEP encryption for wireless networks, and is based on a substitution box in combination with random permutations[37]. Different attacks showed the weaknesses of the algorithm and therefore it can be seen as broken[1] and should not be used anymore.

The A5/1 stream cipher was also developed in 1987 and is used in the GSM system to encrypt the communication between the mobile device and the base station. It is based on three linear feedback shift registers (LFSR). Similar to RC4 this algorithm was also kept secret but eventually became pubic through reverse engineering[17]. A number of attacks have been presented and the algorithm can now be cryptanalysed in real time using a time-memory tradeoff attack published in 2001[14]. Another notable stream cipher is E0[16], which is used in Bluetooth protocol to protect the communication. E0 must be considered as insecure[42].

But there are also some recently developed stream ciphers, such as HC-128[49], Salsa20[11], or SOSEMANUK[9]. These three algorithms aim to be easy to implement in software and also intent to provide decent security as well as encryption speed. HC-128 consists of two secret tables, which are updated using a non-linear feedback function. At each step, a non-linear output Filtering function generates a 32-bit output word, which results in 3.05 cycles/byte on Pentium M processor. Salsa20 was developed by Daniel J. Bernstein and aims to be significantly faster than traditional block ciphers. The Salsa20 encryption function is a chain of XOR, addition, and rotate operations on 32-bit words. The algorithms rounds can be reduced for applications where speed is more important than confidence. SOSEMANUKs structure is influenced by the stream cipher SNOW[26] and the block cipher SERPENT[13]. It has a variable length key between 128 and 256 bits and a initialization vecor of 128 bits and consists of a 320-bit LFSR, and a Finite State Machine (FSM) including two 32-bit registers.

MICKEY[4] on the other hand is similar to Trivium and Grain a cipher designed to be used on hardware with limited resources. The name is an abbreviation of "Mutual Irregular Clocking keystream generator", and it uses irregular clocked shift registers to generate a keystream.

2 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[32]. 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[32], as well as a variant version Grain-128[31] 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[30], 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.

2.1 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 1 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[47]. 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[33] 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 $\mathcal{A} = \{1, 2, 4, 10, 31, 43, 56\}$

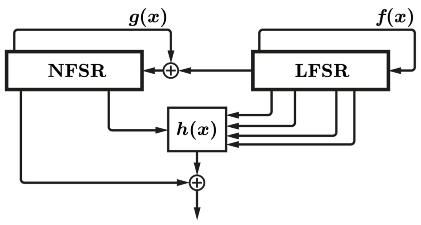

Fig. 1. Grain cipher (by Yossiea, free use, `https://he.wikipedia.org/w/index.php?curid=1217517`)

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 2. The reason for this key initialization phase is to scramble the contents of the shift registers before the key stream is generated.

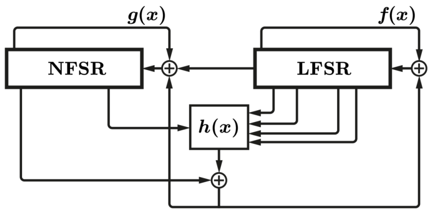

Fig. 2. Grain key initialization (by Yossiea, free use, `https://he.wikipedia.org/w/index.php?curid=1217518`)

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 2.3). 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.

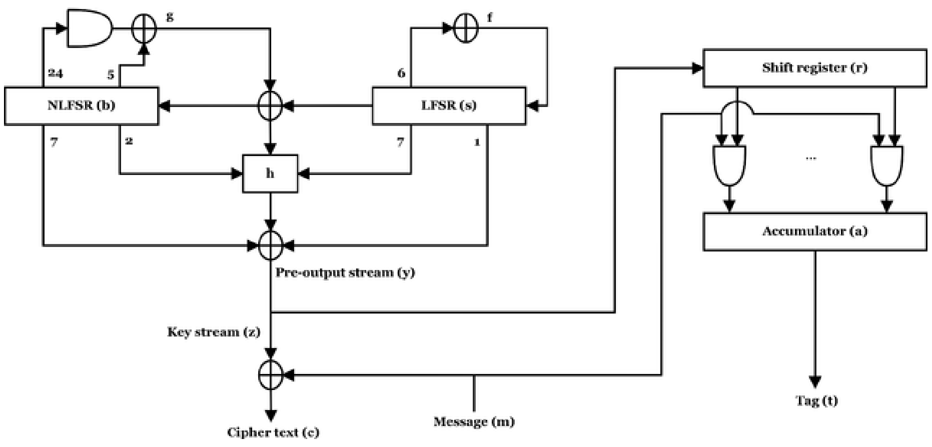

Fig. 3. Grain 128a with optional authentication (CC BY 3.0, `https://en.wikipedia.org/w/index.php?curid=39315190`)

Authentication In order to use the authentication two registers were introduced (cf. Figure 3), 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*[8] approach.

2.2 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[32], and with 1294 gates for the internal state, Grain v1 is one with the lowest amount of required gates from the complete eSTREAM portfolio[29]. 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 (see Table 1).

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.

2.3 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[36] 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[10] 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[38] 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[21], 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[40] 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.[22] and Banik et al.[7] 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[24] in 2009, and soon efficiently implemented on a FPGA by Aumasson et al. in 2009[3]. 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[25], 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[46]. 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[12] against Grain-128, Banik et al.[6] extended this work to other cipher, and showed 2012[5] a differential fault attack on Grain-128a using MACs. Most of the previous work targets the LFSR, but recently Karmakar and Chowdhury showed[34], 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[15] [44] [50] [2] [51] [35] [19] [48] [23]. The mathematical attack based on the dynamic cube attack showed by Aumasson et al.[3] 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[41] that it seems to be immune against dynamic cube and differential attacks.

3 Trivium

3.1 Specifications

In 2005 Christophe De Cannière and Bart Preneel developed a synchronous stream cipher called Trivium [20,?]. 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:

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

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:

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

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:

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

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.

Key stream generation phase The key stream generation process is similar to the cipher initialization process described in the setup phase. Based on the

```
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**
```

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.

```
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**
```

After a keystream Z is generated, it can be used to perform encryption and decryption operations. The structure of Trivium is shown in Figure 4.

3.2 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 [45] 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

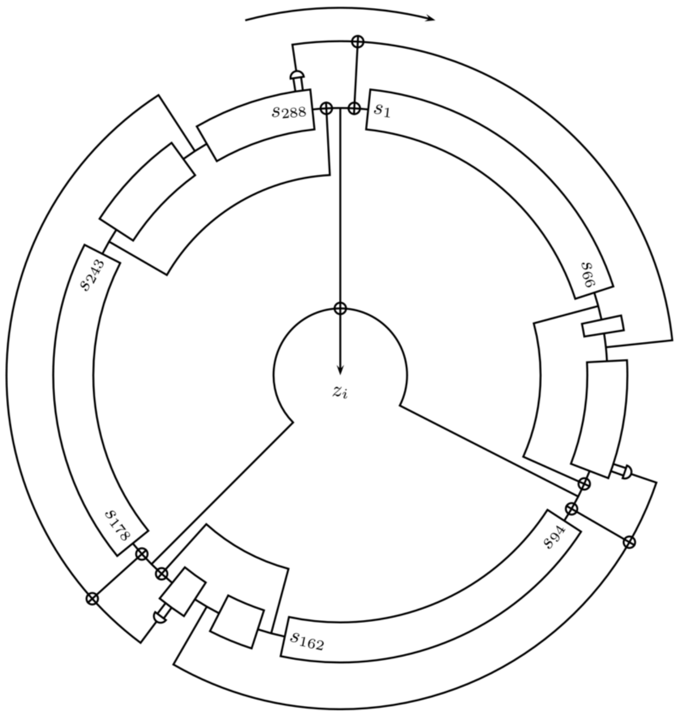

Fig. 4. Trivium structure (copyrighted free use, `https://commons.wikimedia.org/w/index.php?curid=598665`)

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 [27] 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.

3.3 Energy Consumption

Running standard Trivium with 100 kHz clock frequency it consumes about 5.6 μW energy [29]. Other energy consumption profiles for the different Trivium configurations can be seen in Table 1.

3.4 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 [43] 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. [28]. 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 [24]. 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.

4 Comparison

When it comes to environments with high demands concerning speed, security and power efficiency one might think about stream ciphers like Trivium or Grain. Both ciphers are optimized in providing value for similar use cases. They are clear in size and work properly on hardware with a low clock frequency. Assuming Grain v1 the size of the secret key is identical to that from standard Trivium, namely 80 bits. In both ciphers crucial operations are performed in a non linear manner preventing the disclosure of sensitive data to adversaries. Therefore NLFSR components are used for both stream ciphers. Although primary hardware oriented both can be implemented in software as well.

Nonetheless there are some major distinctions between Grain and Trivium. The former has no restrictions concerning the output of the keystream generation process. More precisely, the lengths of the produced keystreams are arbitrary compared to the keystreams generated by Trivium. The latter produces keystreams at most 2^{64} bits in size. In addition, there are differences concerning the number of initialization rounds and the energy consumptions between both algorithms. Table 1 provides a summary of both algorithms regarding *Gate Count*, *Energy Consumption* (μW) and *Throughput* (*Mbps*) based on $100\,kHz$ clock frequency. In terms of Trivium, the data for calculating the *Gate Count* is derived from [18]. To provide a better comparison to Grain a reduced number of NAND gates per Flip-flop (8 rather than 12) is assumed, the calculation rule and the number of gates remain the same [18] though. The results (*Gate Count*) and other information [29] are reflected in Table 1. The parameter t indicates the level of parallelization.

Table 1. Comparison of Grain and Trivium at $100kHz$

Cipher	Gate Count	Energy (μW)	Throughput (Mbps)
Grain v0	1435	na	na
Grain v1, t=0	1450	3.3	0.1
Grain v1, t=2	1637	na	0.2
Grain v1, t=4	2010	4.5	0.4
Grain v1, t=8	2756	6.1	0.8
Grain v1, t=16	4248	9.3	1.6
Grain 128, t=0	2133	4.3	0.1
Grain 128, t=2	2218	na	0.2
Grain 128, t=4	2388	5.6	0.4
Grain 128, t=8	2728	6.9	0.8
Grain 128, t=16	3408	9.3	1.6
Grain 128, t=32	4768	14.8	3.2
Grain 128a w/o ; with MAC, t=0	2145.5 ; 2769.5	na	0.1 ; 0.05
Grain 128a w/o ; with MAC, t=2	2243 ; 2867	na	0.2 ; 0.1
Grain 128a w/o ; with MAC, t=4	2438 ; 3174	na	0.4 ; 0.2
Grain 128a w/o ; with MAC, t=8	2828 ; 3788	na	0.8 ; 0.4
Grain 128a w/o ; with MAC, t=16	3608 ; 5016	na	1.6 ; 0.8
Grain 128a w/o ; with MAC, t=32	5168 ; 7472	na	3.2 ; 1.6
Trivium v1, t=0	2336	5.6	0.1
Trivium v1, t=4	na	5.9	0.4
Trivium v1, t=8	2560	6.4	0.8
Trivium v1, t=16	2816	6.4	1.6
Trivium v1, t=32	3328	10.3	3.2
Trivium v1, t=64	4352	14.3	6.4

5 Conclusion

In this paper Trivium and all existing modifications of Grain have been introduced. Both ciphers have been described in terms of their specifications, their hardware implementations and their security. Finally a comparison between both ciphers has been drawn. The results show that each cipher can be recommended for the application on resource constrained devices.

References

1. AlFardan, N.J., Bernstein, D.J., Paterson, K.G., Poettering, B., Schuldt, J.C.: On the security of rc4 in tls. In: USENIX Security. pp. 305–320 (2013)
2. 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)
3. 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)
4. Babbage, S., Dodd, M.: The mickey stream ciphers. In: New Stream Cipher Designs, pp. 191–209. Springer (2008)

5. 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)
6. 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)
7. 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)
8. Bellare, M., Namprempre, C.: Authenticated encryption: Relations among notions and analysis of the generic composition paradigm. In: Advances in CryptologyASI-ACRYPT 2000, pp. 531–545. Springer (2000)
9. Berbain, C., Billet, O., Canteaut, A., Courtois, N., Gilbert, H., Goubin, L., Gouget, A., Granboulan, L., Lauradoux, C., Minier, M., et al.: Sosemanuk, a fast software-oriented stream cipher. In: New Stream Cipher Designs, pp. 98–118. Springer (2008)
10. Berbain, C., Gilbert, H., Maximov, A.: Cryptanalysis of grain. In: Fast Software Encryption. pp. 15–29. Springer (2006)
11. Bernstein, D.J.: The salsa20 family of stream ciphers. In: New stream cipher designs, pp. 84–97. Springer (2008)
12. 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)
13. Biham, E., Anderson, R., Knudsen, L.: Serpent: A new block cipher proposal. In: Fast Software Encryption. pp. 222–238. Springer (1998)
14. Biryukov, A., Shamir, A., Wagner, D.: Real time cryptanalysis of a5/1 on a pc. In: Fast Software Encryption. pp. 1–18. Springer (2001)
15. Bjørstad, T.: Cryptanalysis of grain using time/memory/data tradeoffs (2013)
16. Bluetooth, S.: Bluetooth core specification. Part B: Baseband Specification p. 47 (2003)
17. Briceno, M., Goldberg, I., Wagner, D.: A pedagogical implementation of a5/1 (1999)
18. Canniere, C.D., Preneel, B.: Trivium specifications. citeseer. ist. psu. edu/734144. html (2005)
19. 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)
20. De Cannière, C.: Trivium: A stream cipher construction inspired by block cipher design principles. In: Information Security, pp. 171–186. Springer (2006)
21. De Cannière, C., Kücük, Ö., Preneel, B.: Analysis of grains initialization algorithm. In: Progress in Cryptology–AFRICACRYPT 2008, pp. 276–289. Springer (2008)
22. 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)
23. 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)
24. Dinur, I., Shamir, A.: Cube attacks on tweakable black box polynomials. In: Advances in Cryptology-EUROCRYPT 2009, pp. 278–299. Springer (2009)
25. Dinur, I., Shamir, A.: Breaking grain-128 with dynamic cube attacks. In: Fast Software Encryption. pp. 167–187. Springer (2011)
26. Ekdahl, P., Johansson, T.: Snow-a new stream cipher. In: Proceedings of First Open NESSIE Workshop, KU-Leuven. pp. 167–168 (2000)

27. 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)
28. 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)
29. Good, T., Benaissa, M.: Hardware results for selected stream cipher candidates. State of the Art of Stream Ciphers pp. 191–204 (2007)
30. 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)
31. 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)
32. 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)
33. Hu, Y., Xiao, G.: Resilient functions over finite fields. Information Theory, IEEE Transactions on 49(8), 2040–2046 (2003)
34. Karmakar, S., Chowdhury, D.R.: Fault analysis of grain family of stream ciphers. IACR Cryptology ePrint Archive 2014, 261 (2014)
35. Karmakar, S., Chowdhury, D.R.: A generic scan attack on hardware based estream winners. IACR Cryptology ePrint Archive 2014, 263 (2014)
36. Khazaei, S., Hassanzadeh, M., Kiaei, M.: Distinguishing attack on grain (2005)
37. Klein, A.: Attacks on the rc4 stream cipher. Designs, Codes and Cryptography 48(3), 269–286 (2008)
38. Küçük, Ö.: Slide resynchronization attack on the initialization of grain 1.0. eSTREAM, ECRYPT Stream Cipher Project, Report 44, 2006 (2006)
39. Lee, J., Kapitanova, K., Son, S.H.: The price of security in wireless sensor networks. Computer Networks 54(17), 2967–2978 (2010)
40. 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)
41. Lehmann, M., Meier, W.: Conditional differential cryptanalysis of grain-128a. In: Cryptology and Network Security, pp. 1–11. Springer (2012)
42. Lu, Y., Vaudenay, S.: Faster correlation attack on bluetooth keystream generator e0. In: Advances in Cryptology–CRYPTO 2004. pp. 407–425. Springer (2004)
43. Maximov, A., Biryukov, A.: Two trivial attacks on trivium. In: Selected Areas in Cryptography. pp. 36–55. Springer (2007)
44. 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)
45. 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)
46. Rahimi, M., Barmshory, M., Mansouri, M.H., Aref, M.R.: Dynamic cube attack on grain-v1. IACR Cryptology ePrint Archive 2013, 268 (2013)
47. Saluja, K.K.: Linear feedback shift registers theory and applications. Department of Electrical and Computer Engineering, University of Wisconsin-Madison pp. 4–14 (1987)
48. Wang, Y., Ding, L., Han, W., Wang, X.: The improved cube attack on grain-v1. IACR Cryptology ePrint Archive 2013, 417 (2013)

49. Wu, H.: The stream cipher hc-128. In: New Stream Cipher Designs, pp. 39–47. Springer (2008)
50. 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)
51. Zhang, H., Wang, X.: Cryptanalysis of stream cipher grain family. IACR Cryptology ePrint Archive 2009, 109 (2009)