`Grain-128AEAD` - A lightweight AEAD stream cipher

Cover sheet

Martin Hell, Lund University, Sweden
Thomas Johansson, Lund University, Sweden
Willi Meier, FHNW, Switzerland
Jonathan Sönnerup, Lund University, Sweden
Hirotaka Yoshida, AIST, Japan

Corresponding submitter:
Hirotaka Yoshida
Cyber Physical Security Research Center (CPSEC),
National Institute of Advanced Industrial Science and Technology (AIST),
2-4-7 Aomi, Koto-ku, Tokyo, 135-0064, Japan
hirotaka.yoshida@aist.go.jp
phone: +81298613258

Backup point of contact:
Martin Hell
Dept. of Electrical and Information Technology
Box 118, Lund University, Sweden
martin.hell@eit.lth.se
phone: +46462224353

Contents

1 Introduction

This document specifies and presents Grain-128AEAD, an authenticated encryption algorithm with support for associated data. The specification is closely based on Grain-128a, introduced in 2011, which has, already for several years, been analyzed in the literature. To benefit from the maturity of the Grain family, our strategy in the design of Grain-128AEAD is to have the changes made to Grain-128a as small as possible. This allows us to argue for the security of the cipher, based on previous results on Grain-128a.

Grain-128a is in turn based on Grain v1 and Grain-128, which have both been extensively analyzed, providing much insight into the security of the design approach. All Grain stream ciphers also allow the throughput to be increased by adding additional copies of the Boolean functions involved.

1.1 NIST requirements

This section provides a mapping of the requirements given by NIST [44] to the respective sections in this document and supporting files.

1.1.1 Cover Sheet

The cover sheet with the name of the submission, name of the submitters, including contact information for the corresponding submitter and a backup point of contact is provided as the first page of this document.

1.1.2 Algorithm Specification and Supporting Documentation

The documentation requirements are provided in [44, Section 2.2].

- The complete written specification of the algorithm is given in Section 2.
- The design rationale and an explanation for the different design decisions are given in Section 3. This also includes specific constants that are used in the algorithm.
- The submission describes a single AEAD algorithm, denoted Grain-128AEAD that takes a 128-bit key and a 96-bit nonce. It does not implement hashing functionality.

- Grain-128AEAD has been designed with 128-bit security in mind. Thus, referring to the NIST requirements [44, Section 3.1], we expect that crypt-analytic attacks requires at least 2^{112} computations on a classical computer in a single key setting.
- Known cryptanalytic attacks, using attacks on Grain-128a as a reference point, on the algorithm are specified in Section 4.
- Advantages and limitations of Grain-128AEAD are given in Section 6.
- References given in Section 4 provide a list of published materials that analyze the security of the very similar Grain-128a.

1.1.3 Source Code and Test Vectors

These requirements are provided in [44, Section 2.3]. Source code of a reference implementation is provided separately from this document. Test vectors from the reference implementation are provided in Section 7.

1.1.4 AEAD Requirements

The AEAD requirements are provided in [44, Section 3.1].

- Grain-128AEAD takes a variable-length plaintext, variable-length associated data, a fixed-length nonce (IV) of size 96 bits, and a fixed-length key of size 128 bits. The output is a variable length ciphertext. The plaintext is recovered from a valid ciphertext. An invalid ciphertext does not return a plaintext.
- For a single key, the nonce must be unique. If the nonce is not unique, i.e., it is repeated for the same key, the algorithm leaks information about the two plaintext, and the MAC can be forged.
- The Grain-128AEAD is one algorithm with the only supported parameters are 128-bit key and 96-bit nonce.
- Grain-128AEAD is a bit oriented stream cipher and it thus also allows byte string inputs. The message padding of one '1' bit, can in an environment that only operates with bytes, be replaced by a '1' followed by seven '0's. This will not affect the MAC result.

- Grain-128AEAD has a keystream limitation of 2^{80} bits, i.e., a pre-output stream limitation of 2^{81} bits.

1.2 Acknowledgments

We wish to thank Alexander Maximov and Martin Ågren, who have been involved in designing previous variants in the Grain family of stream ciphers. Their work has been valuable to the understanding of the cipher and design choices made to Grain-128AEAD have used inspiration from their work.

2 Algorithm Specification

Grain-128AEAD consists of two main building blocks. The first is a pre-output generator, which is constructed using a Linear Feedback Shift Register (LFSR), a Non-linear Feedback Shift Register (NFSR) and a pre-output function, while the second is an authenticator generator consisting of a shift register and an accumulator. The design is very similar to Grain-128a, but has been modified to allow for larger authenticators and to support AEAD. Moreover, the modes of usage have been updated.

2.1 Building Blocks and Functions

The pre-output generator generates a stream of pseudo-random bits, which are used for encryption and the authentication tag. It is depicted in Fig. 1. The

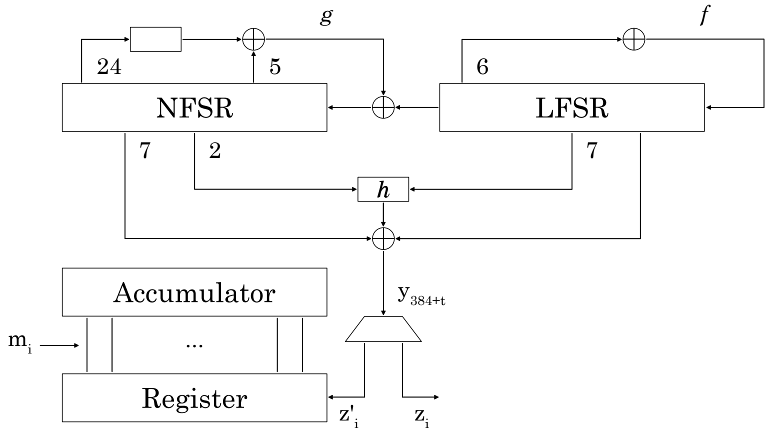

Figure 1: An overview of the building blocks in Grain-128AEAD.

content of the 128-bit LFSR is denoted $S_t = [s_0^t, s_1^t, \ldots, s_{127}^t]$ and the content of the 128-bit NFSR is similarly denoted $B_t = [b_0^t, b_1^t, \ldots, b_{127}^t]$. These two shift registers represent the 256-bit state of the pre-output generator.

The primitive feedback polynomial of the LFSR, defined over GF(2) and denoted $f(x)$, is defined as

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

The corresponding update function of the LFSR is given by

$$\begin{aligned} s_{127}^{t+1} &= s_0^t + s_7^t + s_{38}^t + s_{70}^t + s_{81}^t + s_{96}^t \\ &= \mathcal{L}(S_t). \end{aligned}$$

The nonlinear feedback polynomial of the NFSR, denoted $g(x)$ and also defined over GF(2), is defined as

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

and the corresponding update function is given by

$$\begin{aligned} b_{127}^{t+1} &= s_0^t + b_0^t + b_{26}^t + b_{56}^t + b_{91}^t + b_{96}^t + b_3^t b_{67}^t + b_{11}^t b_{13}^t \\ &+ b_{17}^t b_{18}^t + b_{27}^t b_{59}^t + b_{40}^t b_{48}^t + b_{61}^t b_{65}^t + b_{68}^t b_{84}^t \\ &+ b_{22}^t b_{24}^t b_{25}^t + b_{70}^t b_{78}^t b_{82}^t + b_{88}^t b_{92}^t b_{93}^t b_{95}^t \\ &= s_0^t + \mathcal{F}(B_t). \end{aligned}$$

Nine state variables are taken as input to a Boolean function $h(x)$. Two of these bits are taken from the NFSR and seven are taken from the LFSR. The 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, respectively, the state variables $b_{12}^t, s_8^t, s_{13}^t, s_{20}^t, b_{95}^t, s_{42}^t, s_{60}^t, s_{79}^t$ and s_{94}^t.

The output of the pre-output generator, is then given by the pre-output function

$$y_t = h(x) + s_{93}^t + \sum_{j \in \mathcal{A}} b_j^t,$$

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

The authenticator generator consists of a shift register, holding the most recent 64 odd bits from the pre-output, and an accumulator. Both are of size 64 bits. We denote the content of the accumulator at instance i as $A_i = [a_0^i, a_1^i, \ldots, a_{63}^i]$. Similarly, the content of the shift register is denoted $R_i = [r_0^i, r_1^i, \ldots, r_{63}^i]$.

2.2 Key and Nonce Initialization

Before the pre-output can be used as keystream and for authentication, the internal state of the pre-output generator and the authenticator generator registers are initialized with a key and nonce. Denote the key bits as k_i, $0 \leq i \leq 127$ and the nonce (IV) bits as IV_i, $0 \leq i \leq 95$. Then the state is initialized as follows. The 128 NFSR bits are loaded with the bits of the key $b_i^0 = k_i$, $0 \leq i \leq 127$ and the first 96 LFSR elements are loaded with the nonce bits, $s_i^0 = IV_i$, $0 \leq i \leq 95$. The last 32 bits of the LFSR are filled with 31 ones and a zero, $s_i^0 = 1, 96 \leq i \leq 126$, $s_{127}^0 = 0$. Then, the cipher is clocked 256 times, feeding back the pre-output function and XORing it with the input to both the LFSR and the NFSR, i.e.,

$$\begin{aligned} s_{127}^{t+1} &= \mathcal{L}(S_t) + y_t, \quad 0 \leq t \leq 255, \\ b_{127}^{t+1} &= s_0^t + \mathcal{F}(B_t) + y_t, \quad 0 \leq t \leq 255. \end{aligned}$$

Once the pre-output generator has been initialized, the authenticator generator is initialized by loading the register and the accumulator with the pre-output keystream as

$$\begin{aligned} a_j^0 &= y_{256+j}, \qquad 0 \leq j \leq 63, \\ r_j^0 &= y_{320+j}, \qquad 0 \leq j \leq 63. \end{aligned}$$

When the register and the accumulator are initialized, the key is simultaneously shifted into the LFSR,

$$s_{127}^{t+1} = \mathcal{L}(S_t) + k_{t-256}, \quad 256 \leq t \leq 383,$$

while the NFSR is updated as

$$b_{127}^{t+1} = s_0^t + \mathcal{F}(B_t), \quad 256 \leq t \leq 383.$$

Thus, when the cipher has been fully initialized the LFSR and the NFSR states are given by S_{384} and B_{384}, respectively, and the register and accumulator are given by R_0 and A_0, respectively. The initialization procedure is summarized in Fig 2.

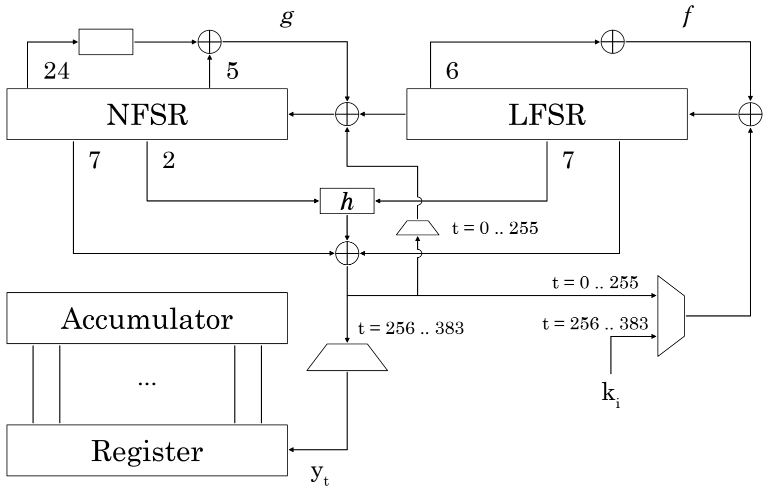

Figure 2: An overview of the initialization in Grain-128AEAD. Note that, in hardware, the accumulator initialization is realized by first loading 64 pre-output bits into the register, followed by moving them to the accumulator.

2.3 Operating Mode

For a message $\boldsymbol{m}$ of length L, denoted $m_0, m_1, \ldots, m_{L-1}$, set $m_L = 1$ as padding in order to ensure that $\boldsymbol{m}$ and $\boldsymbol{m}\|0$ have different tags.

After initializing the pre-output generator, the pre-output is used to generate keystream bits z_i for encryption and authentication bits z'_i to update the register in the accumulator generator. The keystream is generated as

$$z_i = y_{384+2i},$$

i.e., every even bit (counting from 0) from the pre-output generator is taken as a keystream bit. The authentication bits are generated as

$$z'_i = y_{384+2i+1},$$

i.e., every odd bit from the pre-output generator is taken as an authentication bit. The message is encrypted as

$$c_i = m_i \oplus z_i, \quad 0 \le i < L.$$

The accumulator is updated as

$$a_j^{i+1} = a_j^i + m_i r_j^i, \qquad 0 \leq j \leq 63, \quad 0 \leq i \leq L,$$

and the shift register is updated as

$$\begin{aligned} r_{63}^{i+1} &= z_i', \\ r_j^{i+1} &= r_{j+1}^i, \qquad 0 \leq j \leq 62. \end{aligned}$$

2.4 Keystream Limitation

Grain stream ciphers have been designed to allow for encrypting large chunks of data using the same key/nonce pair. Grain-128AEAD restricts the number of keystream bits for each key/nonce to 2^{80}. Thus, the number of pre-output bits that can be generated under one key/nonce pair is 2^{81}.

2.5 Authenticated Encryption with Associated Data

An AEAD scheme allows for data that is authenticated, but unencrypted. Grain-128AEAD achieves this simply by explicitly forcing y_{384+2i} to zero for bits that should not be encrypted, but should still be authenticated. This means that it is possible to control the associated data on bit level, and this data can appear anywhere in the message.

In more detail, we define an *AEAD mask*, denoted

$$d = d_0, d_1, \ldots, d_{L-1},$$

which specifies which bits should be encrypted. If the mask contains only ones (no associated data), the encryption is done as given above. But if the AEAD mask contains zeros, then the encryption is instead done as

$$c_i = m_i \oplus z_i \cdot d_i, \quad 0 \leq i < L,$$

and decryption in the obvious way. The AEAD mask should be predetermined and fixed for the protocol using Grain-128AEAD. The use of such a mask includes great flexibility, as it allows unencrypted data not only as the initial part of a transmitted packet of bits, but one can actually have unencrypted data in any positions of a packet. This provides flexibility in protocol design. The small cost to pay for this is the fact that the cipher is producing some pre-output bits that are not used (those corresponding to unencrypted bits). But since there is essentially no additional cost to handle the introduction of unencrypted data in the construction, we believe that this is an efficient solution.

In the case of variable length associated data, as in the specified NIST API, one has to be a bit careful. If we assume that the first X bits of the data should be unencrypted and the remaining $L - X$ bits encrypted, then $d_i = 0$ if $i < X$ and $d_i = 1$ otherwise, and the ciphertext is generated accordingly. If a ciphertext of the form (ad,ad length X, message, MAC) given by

$$((m_0, m_1, \dots, m_{x-1}), x, (m_x, m_{x+1}, \dots), t)$$

is replaced by a ciphertext

$$((m_0, m_1, \dots, m_{x-1}, m_x), x + 1, (m_{x+1}, m_{x+2}, \dots), t)$$

then this is also a valid ciphertext as the MAC is generated from the same binary string. When variable length associated data is used, we assume that attacks as the one exemplified above are handled by the protocol layer above when it is relevant.

2.6 Using Grain-128AEAD with NIST API

This section will describe how the core Grain-128AEAD algorithm described above is used in NISTs API. To clarify the use of notation, by m we mean the message sent to the API that is subject to encryption, and m' is used to denote the complete string that is used by the core Grain-128AEAD algorithm. Similarly, c is the ciphertext that includes the encrypted string and the authentication information, while c' denotes the bitstring that is decrypted by the core algorithm (thus also including the associated data). It can be noted that the NIST API is byte oriented. This does not pose any significant restrictions to Grain-128AEAD, but the padding bit is here given as 0x80 instead of '1'. Due to the design of the authenticator, these two paddings are equivalent.

2.6.1 Encryption and MAC Generation with NIST API

So for the specific case of the NIST software API, we propose to map the bytewise input (ad, ad length, message, message length) to a bitstring m' in the following

Algorithm 1: AEAD Encrypt with NIST API

```
   Input: $ad$, $adlen$, $m$, $mlen$, $k$, $nonce$
   Output: $c$
1  Initialize generator with $k$ and $nonce$
2  Construct $m' = (\text{Encode}(adlen)\|ad\|m\|\text{0x80})$
3  Let
4     $M$ = bit length of $\text{Encode}(adlen)\|ad$
5     $d_i = 0, \quad (0 \le i \le M-1)$
6     $d_i = 1, \quad (M \le i \le M+mlen-1)$
7  Encrypt using $c'_i = m'_i \oplus z_i d_i, \quad (0 \le i \le M+mlen-1)$
8  Authenticate using $z'_i$ and generate $A_{M+mlen+1}$
9  $c = (c'_M, c'_{M+1}, \ldots, c'_{M+mlen-1})\|A_{M+mlen+1}$
10 return 0
```

way. The bitstring is constructed as

$$m' = \text{Encode(ad length)}||\text{ad}||\text{m}||\text{0x80},$$

where Encode() $= y$ denotes a length encoding similar to the definite form used in DER encoding in, e.g., X.509. If the first byte in y starts with a 0, the remaining 7 bits is an encoding of the number of bytes in the associated data (up to 127 bytes). If the first byte in y starts with a 1, the remaining 7 bits are instead an encoding of the number of forthcoming bytes that are used to describe the length (in bytes) of the associated data. In y, this first byte is then followed by the bytes describing the length.

We then encrypt and authenticate m' in AEAD mode by setting the AEAD mask $d_i = 0$ for all bits $i < M$, where M denotes the bit length of the string Encode(ad length)||ad. For $i \ge M$, we set $d_i = 1$. A summary of the AEAD encryption algorithm, as used with the NIST API, is given by Algorithm 1.

2.6.2 Decryption and MAC Verification with NIST API

On the receiver side, the received MAC has to be verified and decryption has to be performed. The API takes as input the associated data and a ciphertext c which is the encrypted message concatenated with the MAC. Together with the encoding of the ad length and the padding, the bitstring c' is formed. The bitstring m' is determined by computing $m'_i = c'_i \oplus z_i \cdot d_i, \quad 0 \le i < L$. This also includes recomputing the MAC from m' as in the encryption. The receiver

Algorithm 2: AEAD Decrypt with NIST API

```
   Input: $ad$, $adlen$, $c$, $clen$, $k$, $nonce$
   Output: $m$
1  Initialize generator with $k$ and $nonce$
2  Construct $c' = (\text{Encode}(adlen) \| ad \| c_0, \ldots, c_{clen-65} \| \text{0x80})$
3  Let
4      $M$ = bit length of $\text{Encode}(adlen) \| ad$
5      $mlen = clen - 64$
6      $d_i = 0, \quad (0 \leq i \leq M-1)$
7      $d_i = 1, \quad (M \leq i \leq M + mlen - 1)$
8  **Decrypt** using $m'_i = c'_i \oplus z_i d_i, \quad (0 \leq i \leq M + mlen - 1)$
9  **Authenticate** using $z'_i$ and generate $A_{M+mlen+1}$
10 Set $m = m'_M, \ldots, m'_{M+mlen-1}$
11 **if** $(c_{clen-64}, \ldots, c_{clen-1}) == A_{M+mlen+1}$
12     return 0
13 **else**
14     return -1
```

side contains a comparison between the two 64 bit MAC values, resulting in a flag value with value 0 if the two MAC values are equal and -1 otherwise. This comparison is part of the core implementation. Further possible checks, like nonce reuse checks, depends on the application and is considered to take place outside the core implementation. A summary of the AEAD decryption and MAC verification is given in Algorithm 2. Again, note the 0x80 padding, which is due to the byte oriented nature of the NIST API.

3 Design Rationale

This section presents a short overview of the Grain stream ciphers and how the design has evolved through the different versions. It also enumerates and discusses the differences between Grain-128a and the proposed Grain-128AEAD.

3.1 Short History of the Grain Family of Stream Ciphers

The Grain family of stream ciphers are based on the idea behind the nonlinear filter generator. In a nonlinear filter, an LFSR is used to provide a sequence with large period, and a nonlinear function, taking parts of the LFSR sequence as input, is used to add nonlinearity to the keystream sequence. Much work has been put into analyzing the nonlinear filter generator and it is clear that it is very difficult to design a secure nonlinear filter generator with a reasonable hardware footprint [13]. In particular algebraic attacks have been shown to be very strong against this design, see e.g., [16, 42].

In order to better withstand algebraic attacks, and to make the relation between state/key and keystream more complex, Grain adds an NFSR to the nonlinear combiner. The initial submission to the ECRYPT eSTREAM project was analyzed in [35, 10], showing that the nonlinear functions required higher resiliency and nonlinearity. The modified design was subsequently published as Grain v1 [27] and was later selected for the final portfolio in eSTREAM. Grain v1 uses an 80-bit key, and a 128-bit key variant was proposed in [26]. Based on previous results on the Grain construction, Grain-128 was more aggressively designed, making the nonlinear NFSR feedback function of degree 2, but with high nonlinearity and resiliency. The relatively small functions compensated for the fact that the shift registers were increased to 128 bits each, which increased the hardware footprint. The low degree functions were exploited in [3, 46] in order to cryptanalyze a significant number of initializations rounds. These results suggested that the nonlinear functions needed a higher security margin. Grain-128a was proposed in [56], and in addition to increasing the degree of the nonlinear feedback function, an optional authentication mode was added. Work on Grain-128 were subsequently improved [19, 18, 20, 33], emphasizing the need for more complex Boolean functions, and Grain-128 is considered broken and should not be used. The design proposed in this paper, Grain-128AEAD, is closely based on Grain-128a, using the same feedback and output functions. However, slight modifications have been made in order to add security and make it resistant to the attack proposed in [49].

3.2 Differences Between Grain-128AEAD and Grain-128a

Grain-128AEAD takes Grain-128a as starting point, but introduces a number of slight modifications. The modifications are primarily motivated by the NIST Lightweight Cryptography Standardization Process, but inspiration also comes from recent results in [49, 24].

3.2.1 Larger MACs

The register and the authenticator has been increased to 64 bits (instead of 32 bits) in order to allow for authentication tags (MACs) of size 64 bits.

3.2.2 No Encryption-only Mode

Grain-128a allowed for an operation mode with only encryption, where the authentication was removed. This mode resulted in smaller hardware footprint since the two additional registers, and their associated logic, could be left out from an implementation. The encryption-only mode was also more efficient since the initialization process does not include initializing the register and the accumulator, and every pre-output bit was used as keystream. The proposed Grain-128AEAD is a pure authenticated encryption algorithm, and authentication of data is always supported. Thus, there is only one mode of operation.

3.2.3 Initialization Hardening

Based on the ideas in [24] and used in Lizard [25], Grain-128AEAD re-introduces the key into the internal state during the initialization clock cycles. More specifically, it is serially shifted into the LFSR in parallel to the initialization of the register and the accumulator. Several variants can be considered here, including where and when to add the key. The LFSR is chosen due to the fact that if the LFSR is recovered (e.g., in a fast correlation attack as in [49]), it is comparably easy to recover the NFSR state. Moreover, since the LFSR output is XORed with the NFSR input, the key bits will continue to affect also the NFSR during pre-output generation. As for when, we choose to re-introduce it during the last 128 clocks of the initialization. This provides maximum separation between its first introduction in the key loading part, where the key is loaded into the NFSR, and when it is re-introduced. Relations between keys are e.g., more difficult to exploit if the key is properly mixed into the state before the key is re-introduced.

By introducing the key as the last part of the initialization, we achieve the attractive effect that a state recovery attack does not immediately imply key recovery, as was the case for previous versions of Grain. While a state recovery would still render the cipher to be considered broken, the practical effect to deployed devices is highly limited. Recovering the state will only compromise the security of the current message, and not all messages using the same key.

3.2.4 Keystream Limitation

Grain stream ciphers have been designed to allow for encrypting large chunks of data using the same key/nonce pair. Previously, the Grain ciphers have not had any explicit limitation on the keystream length. However, to rule out attacks that use very large keystream sequences, Grain-128AEAD restricts the number of keystream bits for each key/nonce to 2^{80}. We believe that this is well above what will be needed in the foreseeable future. Restricting the number of keystream bits will also make attacks that use linear approximations more difficult, e.g., [49].

3.3 Design Choices for Individual Building Blocks

In this section we motivate design choices for the individual building blocks in Grain-128AEAD. We also specify any security relevant properties for the involved building blocks.

3.3.1 LFSR and NFSR

The key size of Grain-128a is 128 bits. Due to Time/memory/Data tradeoff attacks, the usual strategy is to have a state size that is at least twice the key size. Imposing restrictions on the keystream could make it possible to relax this requirement. However, a redesign of the shift registers would also require a redesign of the involved functions. Such a significant redesign would make it more difficult to re-use the built up knowledge from previous analysis of the Grain ciphers, in particular Grain-128AEAD.

3.3.2 Increasing the Throughput

The throughput of Grain-128AEAD, similar to the other Grain ciphers, can be easily increased by adding more copies of the Boolean functions f, g, and h. To simplify the implementation of the throughput increase up to a factor 32, the Boolean functions do not use the 31 right-most taps of the LFSR and NFSR. The effect of the additional hardware does not necessarily give a linear increase in throughput when clocking at maximum frequency, as can be seen in Section 5, but for moderate clock frequencies, the increase is linear.

3.3.3 Choice of f

The feedback polynomial of the LFSR has two main requirements. First, it must be a primitive polynomial, and second, it must have enough taps to resist correlation attacks, but few taps in order to minimize the hardware cost. To meet these requirements we choose a primitive polynomial of weight 7, i.e., the linear recurrence relation of the generated sequence consists of 7 bits.

3.3.4 Choice of g

The feedback function for the NFSR is used to create nonlinear relations between state bits. Its degree must be high enough to resist cube attacks and its nonlinearity and resiliency must be high enough so that linear approximations have small bias and also consists of many terms. Due to cube attacks on Grain-128 [18], it seems not enough to have a function of degree 2. Thus, the degree used in Grain-128AEAD (as in Grain-128a) is increased to 4. The function

$$\begin{aligned} b(x) &= x_0x_1 + x_2x_3 + x_4x_5 + x_6x_7 + x_8x_9 + x_{10}x_{11} + x_{12}x_{13} \\ &\quad + x_{14}x_{15}x_{16} + x_{17}x_{18}x_{19} + x_{20}x_{21}x_{22}x_{23}, \end{aligned}$$

has nonlinearity 8356352. The resiliency of the function is strengthened by adding 5 linear terms. As a result, the NFSR feedback function $g(x)$ is balanced, has nonlinearity $2^5 \cdot 8356352 = 267403264$, and resiliency 4. There are 2^{14} linear approximations of g with bias $\epsilon_g = 63 \cdot 2^{-15} < 2^{-9}$.

3.3.5 Choice of Pre-output Function

The pre-output function takes input from both the NFSR and LFSR. Similarly to g, the output function consists of one nonlinear (h) and one linear part. The nonlinear function h has nonlinearity 240, and adding 8 variables linearly gives a total nonlinearity of the pre-output function of $2^8 \cdot 240 = 61440$. There are in total 2^8 linear approximations with the highest bias $\epsilon_h = 2^{-5}$.

3.3.6 Authentication of Messages

The authentication mechanism in Grain-128AEAD is based on universal hash functions, introduced in [53]. The sender and the receiver agrees on a hash function from a family of hash functions and hashes the message with this hash function. Then, the hash is encrypted. The actual implementation of this authentication mechanism dates back to the work of Krawczyk in [37], and further discussed in [55]. For hashing, the message is multiplied by a Toeplitz matrix defined by an ϵ-biased sequence.

$$\begin{bmatrix} t_0 \\ t_1 \\ \vdots \\ t_{w-1} \end{bmatrix} = \begin{bmatrix} k_{-1} & k_0 & k_1 & \dots & k_{L-2} \\ k_{-2} & k_{-1} & k_0 & \dots & k_{L-3} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ k_{-w} & k_{-w+1} & k_{-w+2} & \dots & k_{L-1-w} \end{bmatrix} \begin{bmatrix} m_0 \\ m_1 \\ \vdots \\ m_{L-1} \end{bmatrix}$$

The Toeplitz matrix multiplication is implemented using the shift register R and the accumulator A as follows. Interpret the matrix columns as

$$R_0 = [k_{-w}, \dots, k_{-1}], R_1 = [k_{-w+1}, \dots, k_0], \dots, R_{L-1} = [k_{L-1-w}, \dots, k_{L-2}].$$

Then, all w entries in R_0 are multiplied by m_0, all w entries in R_1 are multiplied by m_1 etc. Moreover, R_i is just R_{i-1} shifted left by one step. Thus, we can implement this using a shift register with starting state $R_0 = [k_{-w}, \dots, k_{-1}]$ and an accumulator A as

$$A \leftarrow A \oplus R_i m_i,$$

i.e., if the message bit m_i is one, we update the accumulator by adding the shift register content, otherwise we do nothing. Then we shift in the next keystream bit into the register. A 64-bit keystream block, here seen as a "one-time pad", is added to the accumulator in order to encrypt the hash. Instead of taking this from the end of the keystream, this is extracted as part of the initialization phase of the cipher, i.e., when the accumulator is initialized with keystream bits.

It is well known, see [37, 55], that if the sequence defining the Toeplitz matrix is ϵ-biased, then the substitution probability for the MAC is $P_S \leq 2^{-w} + 2\epsilon$, where w is the size of the tag. This provides a provable bound for the security of the MAC, as well as a straight-forward hardware implementation of the authentication mechanism. In Grain-128AEAD, the sequence defining the Toeplitz matrix is taken from the pre-output sequence. Alternative approaches would be to take this sequence as part of the state, thereby removing the need for the extra shift register R. Yet another approach would be to compute an additional output function used for the authentication. This would double the speed as we would not have to take every second output bit for authentication and encryption respectively. However, using the pre-output allows us to relate the security of the MAC to the security of the stream cipher.

4 Security Analysis and Cryptanalytic Attacks

The security of the Grain family of stream ciphers has been investigated by a large number of third party analysts, publishing various analysis results on the different variants of Grain. Since its first introduction in 2005, much have been learned about the construction and the design approach. There have also been several published ciphers inspired by the design, e.g., Sprout [2] and its successor Plantlet [43]. Also Fruit [22] and Fruit-80 [1] are based on the same design idea. These ciphers have in common that they attempt to realize extremely resource constrained encryption. To minimize the hardware footprint, the key is assumed to be stored in non-volatile memory (NVM) on a device, and this memory is made part of the cryptographic algorithm. Since the key needs to be stored on a device anyway, using the key directly from NVM in the algorithm does not impose additional hardware to the construction. This is not the case for Grain, as we allow the key to be updated in the device, and the key storage is not a part of the cipher. Still, the fact that the above mentioned ciphers use the Grain design idea shows that the design seems to be very suitable for lightweight cryptography.

4.1 General Security Analysis

A main class of attacks on stream ciphers is the Time/Memory/Data tradeoff (TMD-TO) attack, an efficient method of finding either the key or the state of ciphers by balancing between time, memory and keystream data. This can sometimes be much more efficient and more practically applicable than a simple exhaustive key search attack. Some stream ciphers are vulnerable to TMD-TO attacks and their effective key lengths could then be reduced. This typically happens if the state size is too small. A famous practical TMD-TO attack on A5/1 was given in [12].

A TMD-TO attack consists of two parts. The first is a preprocessing phase, during which a table is constructed. The mapping of different keys or internal states to some keystream segment is computed and stored in the table. It is sorted on keystream segments and this process is assumed to use time complexity P and memory M. In the second (real-time) phase, the attacker has intercepted D keystream segments and search for a collision with the table with time complexity T. A collision will recover the corresponding input. By a trade-off between parameters P, D, M, and T, attackers can devise attacks according to available time, memory and data. Examples of tradeoffs are Babbage-Golic (BG) [4, 23] and Biryukov-Shamir (BS) [11] with curves $TM = N$, $P = M$ with $T \leq D$; and $MT^2D^2 = N^2$, $P = N/D$ with $T \geq D^2$, where N is the input space, respectively.

For Grain-128AEAD, attackers have no direct way to reconstruct the internal state, since the cipher has an internal state of size 256 bits (128-bit LFSR + 128-bit NFSR), i.e. $N = 2^{256}$. The best attack complexity achieved under BG tradeoff is with $T = M = D = N^{1/2} = 2^{128}$, which is not favourable compared to exhaustive key search. Also the BS tradeoff does not give complexity parameters of particular interest. Some improvements to TMD-TO attacks can be achieved through so-called BSW sampling [12] and the performance of such an approach is characterized by the sampling resistance of the stream cipher. Various generalizations of the concept of sampling resistance can be considered, e.g. [32], but it seems unlikely that this will lead to an attack with better performance than a standard Hellman-type time-memory tradeoff attack on the keyspace, a generic attack applicable to any cipher. Also, our limit on the length of keystreams affects such attacks.

Another class of general attacks are algebraic attacks, where the attacker derives a system of nonlinear equations in unknown key bits or unknown state bits and then solves the system. In general, solving a system of nonlinear equations is not known to be solvable in polynomial time, but there might be special cases that can be solved efficiently [15]. Due to the NFSR, the degree of the equations will gradually increase and it does not look promising to try to derive a system of nonlinear equations due to this property as well as the algebraic degree of the h function.

A general cryptanalytic technique is a guess-and-determine attack, where one guesses parts of the state and then from the keystream tries to determine other parts of the state. The goal is to guess as few positions as possible and determine as many as possible from equations involving the keystream. Again, since the dependence between a keystream symbol and the state includes many different positions in the state and some of them in nonlinear expressions, one has to guess a large portion of state variables in order to use an equation to determine a single state variable.

Being a binary additive stream cipher, Grain-128AEAD does not allow reuse of a key/nonce pair since this will leak information about the corresponding plaintexts. Moreover, since Grain-128AEAD closely resembles Grain-128a, a key/nonce pair used in one cipher may also not be reused in the other. Such cross-cipher key/nonce reuse in a related cipher model is outside the security model of Grain-128AEAD.

In the subsequent subsections, we now describe the attacks that we consider as the main threat against lightweight stream ciphers in general and Grain-128AEAD in particular.

4.2 Linear Approximations

It is always possible to find a linear combination or the output bits that is unbalanced. In [41], it was shown that by using linear approximations of the functions g and h in Grain, an expression for the bias of a linear combination of output bits can be given. For sake of simplicity, we here include all bits in the output function when we refer to the function h. Let ϵ_g and ϵ_h be the bias of the two nonlinear functions, and let A_g and A_h be linear approximations of the two functions, i.e.,

$$\Pr\{A_g = g\} = 1/2 + \epsilon_g,$$
$$\Pr\{A_h = h\} = 1/2 + \epsilon_h.$$

Then, a time invariant linear combination of keystream bits and LFSR bits can be found that, using the piling-up lemma [40], has the bias

$$\epsilon = 2^{(\eta(A_h)+\eta(A_g)-1)} \cdot \epsilon_g^{\eta(A_h)} \cdot \epsilon_h^{\eta(A_g)},$$

where $\eta(A_g)$ and $\eta(A_h)$ denotes the number of NFSR state variables that are used in the approximations A_g and A_h respectively. In order to get an expression involving only keystream bits, shifted versions of this function (according to the LFSR recurrence relation) can be added. This will again lower the bias according to the piling-up lemma. Finding and using a low-weight multiple of the LFSR polynomial can be used to improve the bias (see e.g., [50]).

Since it is always possible to find a biased linear approximation of the Boolean functions g and h, biased relations in keystream bits will always exist. The functions in Grain-128AEAD have thus been designed to make this bias low. For the function g, we have $\epsilon_g < 2^{-9}$ and $\eta(A_g) = 5$, and for the function h (including the linearly added bits), we have $\epsilon_h < 2^{-5}$ and $\eta(A_h) = 7$. This will give $\epsilon < 2^{-77}$ for this linear approximation (which also includes LFSR bits).

4.3 Correlation Attacks

Grain-128a was designed to resist conventional (fast) correlation attacks that exploit correlations between the state of the LFSR and the corresponding key stream. There has been devised a fast correlation attack on small state Grain-like stream ciphers in [54]. Due to a much bigger state, this attack does not apply to Grain-128a. On the other hand, a recent paper [49] reveals that there are multiple linear approximations in Grain-128a that together with a viewpoint based on a finite field allow a fast correlation attack on the raw encryption mode of Grain-128a (and on the other members of the Grain family), where every keystream bit is assumed to be accessible by an opponent. This attack recovers the state of Grain-128a with data and time complexity of about 2^{114}. The data needs to come from the same secret key and the same nonce.

It should be noted that this fast correlation attack does not apply to Grain-128a in authentication mode, as then only every second key stream bit may be accessible to an opponent. In addition, Grain-128AEAD limits the keystream length to 2^{80} bits for a same secret key and nonce. Thus, these fast correlation attacks do not apply to Grain-128AEAD.

4.4 Chosen IV Attacks

A variety of chosen IV attacks on Grain have been proposed, in both fixed key scenario as well as in the related key setting, and either for distinguishing purpose or for key recovery. In a fixed key scenario, chosen IV attacks have been devised on reduced-round versions using conditional differentials and using cube attacks, or combinations of both [36, 38, 21, 39]. On Grain-128, a dynamic cube attack has been developed that succeeds in finding the secret key for the full 256-round initialization for a fraction of keys, [18]. Dynamic cube attacks have not been successful on Grain-128a thus far. Most of these results are experimental in nature, and do work only if the computational effort is practically feasible.

More recently, division property has been developed to improve cube attacks. Division property is an iterated technique for integral distinguishers introduced by Todo, in [48] and was applied initially to block ciphers. It turned out that it also applies to the initialization of stream ciphers, not only for distinguishers but also for key recovery. As opposed to conventional cube attacks, it can provide theoretical results. The latest result on Grain-128a in this direction is a key recovery on 184 initialization rounds, [51]. The data complexity is 2^{95}, and the computational complexity corresponds to about 2^{110} operations.

An attack that reaches the largest number of initialization rounds of Grain-128a in a fixed key scenario thus far is a conditional differential distinguishing attack and reaches 195 initialization rounds, but it works only for a fraction of all keys, [39].

As a result, there exist no chosen IV attacks on full round initialization of Grain-128a in a single key scenario. The strengthened initialization procedure of Grain-128AEAD is expected to prevent such attacks even further.

The relevance of related key cryptanalysis of stream ciphers has been a subject of debate. A related key attack on Grain-128a in [17] recovers the secret key with a computational complexity 2^{96}, requiring 2^{96} chosen IVs and about 2^{104} keystream bits. It requires only 2 related keys. Another related key attack in [8] recovers the secret key using 2^{64} chosen IVs and 2^{32} related keys, where these figures need to be multiplied by some factor (about 2^{8}).

Due to the modified initialization procedure, related key attacks on Grain-128AEAD are generally expected to become harder than those known on Grain-128a. This refers in particular to attacks in the spirit of [17] and [8].

4.5 Fault Attacks

In the scenario of fault attacks on stream ciphers, the attacker is allowed to inject faults into the internal state, which means either flipping a binary value in memory or assigning a value to zero. By analyzing the difference in keystreams for the faulty and the fault-free case, one attempts to deduce the complete or some partial information about the internal state or the secret key. Fault attacks on stream ciphers have recently received some attention, starting with the work of Hoch and Shamir [28]. The most common methods of injecting faults is by using laser or through clock glitches. Fault attacks usually rely on assumptions that is beyond the model of cryptanalysis and for this reason one can often find rather efficient fault attacks on most ciphers. In some scenarios they are, however, not unrealistic and the exact complexity and the related requirements are of interest to study.

Fault attacks on the Grain family of stream ciphers were studied in [14] and [34]. More recently, there was a number of papers providing improved attacks, [6, 45, 5, 7]. In [45] the model is the most realistic one as it considers that the cipher has to be re-initialized only a few times and faults are injected to any random location and at any random clock cycle. No further assumptions are needed over location and timing for injections. In the attack one constructs algebraic equations based on the description of the cipher by introducing new variables so that the degrees of the equations do not increase. Following algebraic cryptanalysis, such equations based on both fault-free and faulty key-stream bits are collected. Then a solving phase using the SAT Solver recovers the state of any Grain member in minutes, For Grain v1, Grain-128 and Grain-128a, it uses only 10, 4 and 10 injected faults, respectively.

We stress that we are not claiming resistance against fault attacks for Grain-128AEAD. Rather, when fault attacks is a realistic threat, one has to implement protection mechanisms against fault injection.

4.6 Security of the Authentication

As noted in Section 3.3.6, if the sequence defining the Toeplitz matrix is ϵ-biased, then the substitution probability for the MAC is $P_S \leq 2^{-w} + 2\epsilon$, where w is the size of the tag. Since the pre-output sequence is used both for encryption and authentication, a substitution attack on the MAC is related to a finding linear relations in the pre-output sequence. From Section 4.2, we know that a linear relation involving pre-output and LFSR bits can be found that has bias around 2^{-77}. This provides a comfortable security margin against substitution attacks on the authentication. Given this, we claim that the success probability of a substitution attack is close to 2^{-64}, i.e., guessing the tag.

5 Hardware Implementation

Lightweight ciphers are important in constrained devices. A minimal design is desirable, e.g., minimum area and very low power consumption since they often must operate for an extended period of time, without a battery change. In some cases, devices run without its own power supply, something that is often the case with RFID tags.

Table 1: The gate count for different functions.

Function	**Gate Count**
NAND2	1.0
NAND3	1.5
NAND4	2.0
XOR2	2.5
XOR3	6.5
Flip flop	8.0

Grain-128AEAD can be constructed using primitive hardware building blocks, such as NAND gates, XOR gates and flip flops. In order to get an idea of the hardware footprint related to an implementation of the cipher, we implement the stream cipher using 65 nm library from ST Microelectronics, `stm065v536`. For synthesis and power simulation, the Synopsys Design Compiler 2013.12 is used. It can be noted that the result is highly dependent on what kind of gates are available and how the tool utilizes the standard cells. We define a 2-input NAND gate to have a gate count of 1 and other gate counts are given in relation to this NAND gate. An excerpt from the standard-cell library documentation is given in Table 1.

Table 2: Gate count for the different building blocks, for different levels of parallelization, s.

Building Block	**Gate Count**		
	$s = 1$	$s = 2$	$s = 32$
LFSR	1024	1024	1024
NFSR	1024	1024	1024
f	19	38	608
g	62.5	125	2000
h	41.5	83	1328
Control logic	219.5	475.5	942.5
Accumulator	512	512	512
Register	512	512	512
Accumulator logic	224	224	4160
Total	3638.5	4017.5	12110.5

We synthesize the design and extract the gate count for each building block. A summary of the gate count for each building block, and for different parallelization levels, is given in Table 2. The control logic and accumulator logic is extra circuitry and state machines for controlling the stream cipher, i.e., loading key and nonce, multiplexing data, etc.

The gate count remains constant during synthesis, but the physical area, power and speed changes based on the optimization techniques employed. First, we synthesize the design at clock frequency 100 kHz. The design is synthesized for three levels of parallelization; 1, 2, and 32 times. The result is given in Table 3.

We also synthesize for the maximum possible speed, to achieve maximum throughput, without constraints on area. The results are given in Table 4.

Table 3: Implementation results running at 100 kHz, for different levels of parallelization.

Parallelization	**Area**	**Power**	**Throughput**
1	4934 μm^2	313 nW	50 kbit/s
2	5336 μm^2	368 nW	100 kbit/s
32	16853 μm^2	574 nW	1600 kbit/s

Table 4: Implementation results running at maximum possible speed, for different levels of parallelization.

Parallelization	**Speed**	**Area**	**Power**	**Throughput**
1	1.12 GHz	5258 μm^2	3.6 mW	560 Mbit/s
2	1.18 GHz	5629 μm^2	4.3 mW	1.18 Gbit/s
32	662 MHz	17632 μm^2	4.0 mW	10.59 Gbit/s

Note that we have not used any particular optimization techniques in the implementation and we expect that these figures can be further improved.

6 Advantages and Limitations

6.1 Suitability of Grain-128AEAD in IoT/Embedded Systems

Grain-128AEAD can be very suitable in Internet of things (IoT) and embedded systems. Strong advantages of Grain-128AEAD and its precedent versions can be seen in its industrial relevance.

Since around 2004, RFID systems which typically consist of tags, tag readers, and servers, have become popular in various industry domains. Through the eSTREAM competition process, the security and the performance of the Grain family has been widely recognized. This has led to the development of the standard ISO/IEC 29167-13:2015 [30] specifying Grain-128a for RFID systems.

For the coming years, one of the most important embedded systems is automotive system. Modern vehicles provide services based on subsystems. One example is the automotive Passive Keyless Entry and Start (PKES) systems where the key fob communicate with one of the Electronic Control Unit (ECU)s when the driver approaches his vehicle.

Since 2010, various kind of attacks [47] have been mounted on PKES employed in real-world vehicles. To ensure the security against these attacks, we need cryptography. In case that the developers implement cryptography for vehicles of new or future models, it is reasonably expected that a cryptographic primitive is implemented in hardware on the key fob (RFID tag) and is implemented in embedded software on an in-vehicle ECU, which means the cryptographic primitive has to meet *both* hardware requirement and embedded software requirement.

We expect that Grain-128AEAD can be a very good option in this respect, given its competitive performance in hardware, reported in this document, as well as remarkable performance in embedded software. In fact, the passive IT70 RFID tag [29] that Honeywell has designed for automotive applications implements the ISO/IEC 29167-13:2015 standard. On the other hand, it was reported at the FSE 2018 rump session [52] that Grain-128a requires 164 byte of RAM and takes 385 μs for 16-byte input data on ARM Cortex-M3, which is significantly better than AES and faster than the lightweight SKINNY-128-128 block cipher.

Another advantage of Grain-128AEAD for IoT applications can be seen in its use of the *mask*. In 2017, the IoT security standard ITU-T X.1362 [31] specifying the Encryption with Associated Mask Data (EAMD) protocols for IoT has been published. It aims to meet severe realtime requirement in IoT/embedded systems by using the similar idea of mask to the mask used in Grain-128AEAD. The advantage of Grain-128AEAD over EAMD can be that Grain-128AEAD has less overhead than EAMD because Grain-128AEAD incorporate a mask at the primitive level while EAMD use the mask at the protocol level.

6.2 Other Aspects

- Grain-128AEAD is closely based on Grain-128a, which has been extensively analyzed by third parties. Also its predecessors, Grain v1 and Grain-128 have been thoroughly analyzed since the first introduction of Grain in 2005. This provides confidence in the design and can be seen as an advantage.

- Distinguishing attacks are not explicitly mentioned in the submission requirements from NIST. We note that many block cipher modes of operation allows for distinguishing attacks using roughly $2^{b/2}$ keystream blocks, where b is the block size in bits. Grain has been designed to resist distinguishing attacks that use much longer keystream sequences than this fundamental bound. Thus, if resistance against distinguishing attacks is to be considered a security property, Grain-128AEAD provides a security advantage over block ciphers.

- In [9], it was shown that lightweight stream ciphers are typically more suitable than lightweight block ciphers for energy optimization when encrypting longer messages, in particular when the speed can be increased at the expense of moderate extra hardware. Thus, in these cases, Grain-128AEAD can provide authenticated encryption with low energy consumption.
- Compared to e.g., Sprout [2], Plantlet [43], Fruit [22] and Fruit-80 [1], Grain-128AEAD does not store the key in non-volatile memory. This has the limitation that the internal state size is larger than for those ciphers, but at the same time it has the advantage that the key can be updated in the device, making it usable for a wider range of use cases.
- Grain-128AEAD, as well as the other ciphers in the Grain family, are designed to explicitly and easily increase the throughput at the expense of additional hardware. This provides a wide range of use cases, from situations where hardware footprint is of highest concern, to situations where throughput is more important.

7 Test Vectors

In this section, we give test vectors for two different combinations of key, nonce, and message. The Grain family of stream ciphers are bit-oriented and in a byte-oriented implementation, as required in the NIST defined API, a decision regarding how to interpret the order of bits needs to be made. We stress that this decision is up to the protocol/application designer since the most suitable interpretation can be situation specific. The test vectors below, as well as the reference implementation, assume that the least significant bit (LSB) of each byte is read first. Thus, the hexadecimal representation of the test vectors below are bytewise reversed. The key

```
0x01234FFFFFFFFFFFFFFFFFFFFFFFFFFF
```

corresponds to

$$(k_0, k_1, ..., k_{127}) =$$
$$(1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 1..., 1).$$

The message, the associated data and the registers are interpreted similarly. To simplify debugging, the test vectors are given in three stages:

- Directly after loading the key and nonce
- After the initialization of Grain-128AEAD
- The final result with the tag and ciphertext

The first test vector is given below. It uses a zero key/nonce pair and an empty message.

```
Key:    0x00000000000000000000000000000000
Nonce:  0x000000000000000000000000
AD:
PT:

After Loading
NFSR:   0x00000000000000000000000000000000
LFSR:   0x000000000000000000000000ffffff7f
ACC:    0x0000000000000000
REG:    0x0000000000000000

After Initialization
NFSR:   0x5499d7ab3dd190299b746e868fd3409e
LFSR:   0xcf19b05a3892a28ad3f6b1bfb4203bbb
ACC:    0x0304fe446806a6d0
REG:    0x56a95447a661c8f6

Ciphertext and Tag
CT:
Tag:    0xaa50b9e209ce50f0
```

The second test vector is given below.

```
Key:    0x000102030405060708090a0b0c0d0e0f
Nonce:  0x000102030405060708090a0b
AD:     0x0001020304050607
PT:     0x0001020304050607
```

```
After Loading
NFSR:   0x000102030405060708090a0b0c0d0e0f
LFSR:   0x000102030405060708090a0bffffff7f
ACC:    0x0000000000000000
REG:    0x0000000000000000

After Initialization
NFSR:   0x79b1c18f580fe3ae76434f3c3c6ab613
LFSR:   0x5786a81c2bd080517c27132a29ab3e4b
ACC:    0x6e73d6fb62c21220
REG:    0x3b960a19fbd66db2

Ciphertext and Tag
CT:     0x72111052D73C410E
Tag:    0x8D98EA68D9A2C044
```

References

[1] Vahid Amin Ghafari and Honggang Hu. Fruit-80: A secure ultra-lightweight stream cipher for constrained environments. *Entropy*, 20(3):180, 2018.

[2] Frederik Armknecht and Vasily Mikhalev. On lightweight stream ciphers with shorter internal states. In Gregor Leander, editor, *Fast Software Encryption*, pages 451–470. Springer Berlin Heidelberg, 2015.

[3] Jean-Philippe Aumasson, Itai Dinur, Luca Henzen, Willi Meier, and Adi Shamir. Efficient FPGA implementations of high-dimensional cube testers on the stream cipher Grain-128. *SHARCS'09 Special-purpose Hardware for Attacking Cryptographic Systems*, page 147, 2009.

[4] S.H. Babbage. Improved "exhaustive search" attacks on stream ciphers. *IET Conference Proceedings*, pages 161–166(5), January 1995.

[5] Subhadeep Banik, Subhamoy Maitra, and Santanu Sarkar. A differential fault attack on Grain-128a using MACs. In Andrey Bogdanov and Somitra Sanadhya, editors, *Security, Privacy, and Applied Cryptography Engineering*, pages 111–125. Springer Berlin Heidelberg, 2012.

[6] Subhadeep Banik, Subhamoy Maitra, and Santanu Sarkar. A differential fault attack on the Grain family of stream ciphers. In Emmanuel Prouff and Patrick Schaumont, editors, *Cryptographic Hardware and Embedded Systems – CHES 2012*, pages 122–139. Springer Berlin Heidelberg, 2012.

[7] Subhadeep Banik, Subhamoy Maitra, and Santanu Sarkar. A differential fault attack on the Grain family under reasonable assumptions. In Steven Galbraith and Mridul Nandi, editors, *Progress in Cryptology - INDOCRYPT 2012*, pages 191–208. Springer Berlin Heidelberg, 2012.

[8] Subhadeep Banik, Subhamoy Maitra, Santanu Sarkar, and Turan Meltem Sönmez. A chosen IV related key attack on Grain-128a. In Colin Boyd and Leonie Simpson, editors, *Information Security and Privacy*, pages 13–26. Springer Berlin Heidelberg, 2013.

[9] Subhadeep Banik, Vasily Mikhalev, Frederik Armknecht, Takanori Isobe, Willi Meier, Andrey Bogdanov, Yuhei Watanabe, and Francesco Regazzoni. Towards low energy stream ciphers. *IACR Transactions on Symmetric Cryptology*, 2018(2):1–19, Jun. 2018.

[10] Côme Berbain, Henri Gilbert, and Alexander Maximov. Cryptanalysis of Grain. In Matthew Robshaw, editor, *Fast Software Encryption*, pages 15–29. Springer Berlin Heidelberg, 2006.

[11] Alex Biryukov and Adi Shamir. Cryptanalytic time/memory/data tradeoffs for stream ciphers. In Tatsuaki Okamoto, editor, *Advances in Cryptology — ASIACRYPT 2000*, pages 1–13. Springer Berlin Heidelberg, 2000.

[12] Alex Biryukov, Adi Shamir, and David Wagner. Real time cryptanalysis of A5/1 on a PC. In Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, and Bruce Schneier, editors, *Fast Software Encryption*, pages 1–18. Springer Berlin Heidelberg, 2001.

[13] An Braeken and Joseph Lano. On the (im)possibility of practical and secure nonlinear filters and combiners. In Bart Preneel and Stafford Tavares, editors, *Selected Areas in Cryptography*, pages 159–174. Springer Berlin Heidelberg, 2006.

[14] G. Castagnos, A. Gouget, P. Paillier, A. Berzati, C. Canovas, L. Goubin, S. Salgado, and B. Debraize. Fault analysis of GRAIN-128. In *Hardware-*

Oriented Security and Trust, IEEE International Workshop on(HST), pages 7–14, 2009.

[15] Nicolas Courtois, Alexander Klimov, Jacques Patarin, and Adi Shamir. Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In Bart Preneel, editor, *Advances in Cryptology — EUROCRYPT 2000*, pages 392–407, 2000.

[16] Nicolas T Courtois. Fast algebraic attacks on stream ciphers with linear feedback. In *Annual International Cryptology Conference, CRYPTO 2003*, pages 176–194. Springer, 2003.

[17] Lin Ding and Jie Guan. Related key chosen IV attack on Grain-128a stream cipher. *IEEE Transactions on Information Forensics and Security*, 8(5):803–809, 2013.

[18] Itai Dinur, Tim Güneysu, Christof Paar, Adi Shamir, and Ralf Zimmermann. An experimentally verified attack on full Grain-128 using dedicated reconfigurable hardware. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, pages 327–343. Springer Berlin Heidelberg, 2011.

[19] Itai Dinur and Adi Shamir. Breaking Grain-128 with dynamic cube attacks. In Antoine Joux, editor, *Fast Software Encryption*, pages 167–187. Springer Berlin Heidelberg, 2011.

[20] Ximing Fu, Xiaoyun Wang, Jiazhe Chen, and Marc Stevens. Determining the nonexistent terms of non-linear multivariate polynomials: How to break Grain-128 more efficiently. *IACR Cryptology ePrint Archive*, 2017:412, 2017.

[21] Vahid Amin Ghafari and Honggang Hu. A new chosen IV statistical attack on Grain-128a cipher. In *Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), 2017 International Conference on*, pages 58–62. IEEE, 2017.

[22] Vahid Amin Ghafari, Honggang Hu, and Chengxin Xie. Fruit: ultra-lightweight stream cipher with shorter internal state. *eSTREAM, ECRYPT Stream Cipher Project*, 2016.

[23] Jovan Dj. Golić. Cryptanalysis of alleged a5 stream cipher. In Walter Fumy, editor, *Advances in Cryptology — EUROCRYPT '97*, pages 239–255. Springer Berlin Heidelberg, 1997.

[24] Matthias Hamann and Matthias Krause. On stream ciphers with provable beyond-the-birthday-bound security against time-memory-data tradeoff attacks. *Cryptography and Communications*, 10(5):959–1012, 2018.

[25] Matthias Hamann, Matthias Krause, and Willi Meier. Lizard–a lightweight stream cipher for power-constrained devices. *IACR Transactions on Symmetric Cryptology*, 2017(1):45–79, 2017.

[26] Martin Hell, Thomas Johansson, Alexander Maximov, and Willi Meier. A stream cipher proposal: Grain-128. In *Information Theory, 2006 IEEE International Symposium on*, pages 1614–1618. IEEE, 2006.

[27] Martin Hell, Thomas Johansson, and Willi Meier. Grain: a stream cipher for constrained environments. *International Journal of Wireless and Mobile Computing*, 2(1):86–93, 2007.

[28] Jonathan J. Hoch and Adi Shamir. Fault analysis of stream ciphers. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 2004*, pages 240–253. Springer Berlin Heidelberg, 2004.

[29] Honeywell. IT70 secure passive rfid tag. *Technical Specifications*, 2017.

[30] ISO/IEC 29167-13:2015 information technology — automatic identification and data capture techniques — part 13: Crypto suite Grain-128A security services for air interface communications, 2015.

[31] ITU-T X.1362 simple encryption procedure for internet of things (IoT) environments, 2017. https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-X.1362-201703-I!!PDF-E&type=items.

[32] Lin Jiao, Bin Zhang, and Mingsheng Wang. Two generic methods of analyzing stream ciphers. In Javier Lopez and Chris J. Mitchell, editors, *Information Security*, pages 379–396. Springer International Publishing, 2015.

[33] Linus Karlsson, Martin Hell, and Paul Stankovski. Not so greedy: Enhanced subset exploration for nonrandomness detectors. In Paolo Mori, Steven Furnell, and Olivier Camp, editors, *Information Systems Security and Privacy*, pages 273–294. Springer International Publishing, 2018.

[34] Sandip Karmakar and Dipanwita Roy Chowdhury. Fault analysis of Grain-128 by targeting NFSR. In Abderrahmane Nitaj and David Pointcheval, editors, *Progress in Cryptology – AFRICACRYPT 2011*, pages 298–315. Springer Berlin Heidelberg, 2011.

[35] Shahram Khazaei, Mahdi M Hasanzadeh, and Mohammad S Kiaei. Linear sequential circuit approximation of Grain and Trivium stream ciphers. *IACR Cryptology ePrint Archive*, 2006:141, 2006.

[36] Simon Knellwolf, Willi Meier, and María Naya-Plasencia. Conditional differential cryptanalysis of NLFSR-based cryptosystems. In Masayuki Abe, editor, *Advances in Cryptology - ASIACRYPT 2010*, pages 130–145. Springer Berlin Heidelberg, 2010.

[37] Hugo Krawczyk. New hash functions for message authentication. In Louis C. Guillou and Jean-Jacques Quisquater, editors, *Advances in Cryptology — EUROCRYPT '95*, pages 301–310. Springer Berlin Heidelberg, 1995.

[38] Michael Lehmann and Willi Meier. Conditional differential cryptanalysis of Grain-128a. In Josef Pieprzyk, Ahmad-Reza Sadeghi, and Mark Manulis, editors, *Cryptology and Network Security*, pages 1–11. Springer Berlin Heidelberg, 2012.

[39] Zhen Ma, Tian Tian, and Wen-Feng Qi. Conditional differential attacks on Grain-128a stream cipher. *IET Information Security*, 11(3):139–145, 2016.

[40] Mitsuru Matsui. Linear cryptanalysis method for des cipher. In Tor Helleseth, editor, *Advances in Cryptology — EUROCRYPT '93*, pages 386–397. Springer Berlin Heidelberg, 1994.

[41] Alexander Maximov. Cryptanalysis of the "grain" family of stream ciphers. In *Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security*, ASIACCS '06, pages 283–288. ACM, 2006.

[42] Willi Meier, Enes Pasalic, and Claude Carlet. Algebraic attacks and decomposition of boolean functions. In *International Conference on the Theory*

and Applications of Cryptographic Techniques, EUROCRYPT 2004, pages 474–491. Springer, 2004.

[43] Vasily Mikhalev, Frederik Armknecht, and Christian Müller. On ciphers that continuously access the non-volatile key. *IACR Transactions on Symmetric Cryptology*, pages 52–79, 2016.

[44] NIST. Submission requirements and evaluation criteria for the lightweight cryptography standardization process, 2018. `https://csrc.nist.gov/projects/lightweight-cryptography`.

[45] Santanu Sarkar, Subhadeep Banik, and Subhamoy Maitra. Differential fault attack against Grain family with very few faults and minimal assumptions. *IEEE Transactions on Computers*, 64(6):1647–1657, 2015.

[46] Paul Stankovski. Greedy distinguishers and nonrandomness detectors. In Guang Gong and Kishan Chand Gupta, editors, *Progress in Cryptology - INDOCRYPT 2010*, pages 210–226. Springer Berlin Heidelberg, 2010.

[47] Stefan Tillich and Marcin Wójcik. Security analysis of an open car immobilizer protocol stack. In *the industry track of the 10th International Conference on Applied Cryptography and Network Security (ACNS'12)*, 2012.

[48] Yosuke Todo. Structural evaluation by generalized integral property. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015*, pages 287–314, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.

[49] Yosuke Todo, Takanori Isobe, Willi Meier, Kazumaro Aoki, and Bin Zhang. Fast correlation attack revisited. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018*, pages 129–159. Springer International Publishing, 2018.

[50] David Wagner. A generalized birthday problem. In Moti Yung, editor, *Advances in Cryptology — CRYPTO 2002*, pages 288–304. Springer Berlin Heidelberg, 2002.

[51] Qingju Wang, Yonglin Hao, Yosuke Todo, Chaoyun Li, Takanori Isobe, and Willi Meier. Improved division property based cube attacks exploiting algebraic properties of superpoly. In Hovav Shacham and Alexandra Boldyreva,

editors, *Advances in Cryptology – CRYPTO 2018*, pages 275–305. Springer International Publishing, 2018.

[52] Y. Watanabe, H. Yamamoto, and H. Yoshida. Extending FELICS for Automotive PKES Systems. In *2019 FSE Rump session proceedings*, pages 152–158, Mar 2018. https://fse.iacr.org/2018/files/proceedings_rumpsession.pdf.

[53] Mark N. Wegman and J.Lawrence Carter. New hash functions and their use in authentication and set equality. *Journal of Computer and System Sciences*, 22(3):265 – 279, 1981.

[54] Bin Zhang, Xinxin Gong, and Willi Meier. Fast correlation attacks on Grain-like small state stream ciphers. *IACR Trans. Symmetric Cryptol.*, 2017(4):58–81, 2017.

[55] M. Ågren, M. Hell, and T. Johansson. On hardware-oriented message authentication with applications towards rfid. In *2011 Workshop on Lightweight Security Privacy: Devices, Protocols, and Applications*, pages 26–33, March 2011.

[56] Martin Ågren, Martin Hell, Thomas Johansson, and Willi Meier. Grain-128a: a new version of Grain-128 with optional authentication. *International Journal of Wireless and Mobile Computing*, 5(1):48–59, 2011.