A Stream Cipher Proposal: Grain-128

Martin Hell, Thomas Johansson, Alexander Maximov
Department of Information Technology
Lund University, Sweden
E-mail: {martin,thomas,movax}@it.lth.se

Willi Meier
FH Aargau
CH-5210 Windisch, Switzerland
E-mail: meierw@fh-aargau.ch

***Abstract*— A new stream cipher, Grain-128, is proposed. The design is very small in hardware and it targets environments with very limited resources in gate count, power consumption, and chip area. Grain-128 supports key size of 128 bits and IV size of 96 bits. The design is very simple and based on two shift registers, one linear and one nonlinear, and an output function.**

I. Introduction

Symmetric cryptographic primitives for encryption are divided into block ciphers and stream ciphers. While design principles and security of block ciphers are quite well understood, stream cipher design still requires much research. Since block ciphers can be turned into stream ciphers, using OFB or CFB mode, there has been some debate whether stream ciphers are useful at all. The general opinion seems to be that pure stream ciphers are still interesting for two reasons. First, they can be designed to allow much faster keystream generation in software than a block cipher in stream cipher mode. Second, they can be designed to be smaller in hardware. Hence, for a stream cipher to be interesting it must either be very fast in software or very small in hardware. Otherwise, e.g., AES in OFB or CFB mode would be a more attractive option for a user. This opinion has been reflected by the eSTREAM project [1], which is an effort to identify new stream ciphers that might be interesting for widespread adoption. The eSTREAM project focuses on stream ciphers satisfying one or both of the above mentioned criteria.

The encryption operation in a binary additive stream cipher is simple. The keystream is binary xored with the plaintext to form the ciphertext. Similarly, decryption is done by xoring the same keystream with the ciphertext to obtain the original plaintext. The most important property of a stream cipher is the resistance against different attacks. It should not be possible to recover the initial state of the cipher and the generated keystream should be indistinguishable from a truly random sequence. There are several ways to construct a stream cipher. A common building block is a linear feedback shift register (LFSR). These have several statistical properties that coincide with the statistical properties of truly random strings. However, since the generation of new symbols is linear, some additional building block is needed in order to remove the linearity. Irregular clocking, nonlinear filters and decimation algorithms are commonly used to introduce nonlinearity in the keystream.

The purpose of this paper is to propose a 128-bit version of the stream cipher Grain. Grain is a binary additive stream cipher. The previous version, which is denoted Grain Version 1 [2], targets applications which have very limited hardware resources. Grain Version 1 supports a key size of 80 bits (as specified in eSTREAM), which is not feasible to exhaustively search with modern computers. Recent research in time-memory-data trade-off attacks suggests that it is possible to mount an attack with complexity $O(2^{K/2})$ where K is the size of the key. In this scenario the attacker has a collection of $2^{K/2}$ plaintexts encrypted under different keys and the aim of the attack is to find one of these keys. In this attack scenario 80 bit key size is not enough since an attack would have complexity $O(2^{40})$. Several researchers have recently expressed the opinion that 128 bit keys is a minimum in secure applications.

Grain-128 is designed to meet this requirement while preserving the advantages of Grain Version 1. It supports key size of 128 bits and IV size of 96 bits. The cipher is still very small and easy to implement in hardware. Furthermore, it is possible, and easy, to increase the speed at the expense of more hardware. This is a distinguishing feature of the Grain family of stream ciphers and is not explicitly found in many other ciphers. Grain-128 uses a linear feedback shift register to ensure good statistical properties and to guarantee a lower bound for the period of the keystream. To introduce nonlinearity, a nonlinear feedback shift register (NFSR) is used together with a nonlinear filter. The nonlinear filter takes input from both shift registers.

An unpublished version of Grain, referred to as Grain Version 0, was submitted to the eSTREAM project. This version was successfully cryptanalysed by independent researchers. All attacks were based on the same idea of making a linear approximation of the nonlinear circuit. The conclusion from this cryptanalysis is that the choice of one internal function was unfortunate. By a small tweak, just adding a few variables to the output function, the proposed attacks no longer worked. The attacks proposed have given further theoretical ground for the design principles of the Grain family. The functions in Grain-128 are constructed with this theory in mind. To the best of our knowledge, there is no 128 bit cipher offering the same security as Grain-128 and a smaller gate count in hardware.

The outline of the paper is as follows. In Section II we will present the design parameters of Grain-128 and in Section III the throughput and the possibility to increase the speed will be discussed. In Section IV we will look at the security of the cipher and based on this security evaluation the different design choices will be explained in Section V. Since Grain-

128 is designed to be suitable in hardware environments an estimation of the hardware complexity will be given in Section VI. Section VII will conclude the paper.

II. Design Details

This section specifies the details of the design. A theoretical motivation for the design choices will be given in Section V. An overview of the different blocks used in the cipher can be found in Fig. 1 and the specification will refer to this figure. The cipher consists of three main building blocks, namely an LFSR, an NFSR and an output function. The content of the LFSR is denoted by $s_i, s_{i+1}, \ldots, s_{i+127}$. Similarly, the content of NFSR is denoted by $b_i, b_{i+1}, \ldots, b_{i+127}$. The feedback polynomial of the LFSR, denoted $f(x)$, is a primitive polynomial of degree 128. It is defined as

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

To remove any possible ambiguity we also give the corresponding update function of the LFSR as

$$s_{i+128} = s_i + s_{i+7} + s_{i+38} + s_{i+70} + s_{i+81} + s_{i+96}.$$

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

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

Again, to remove any possible ambiguity we also write the corresponding update function of the NFSR. In the update function below, note that the bit s_i which is masked with the input to the NFSR is included, while omitted in the feedback polynomial.

$$\begin{aligned} b_{i+128} \quad = \quad & s_i + b_i + b_{i+26} + b_{i+56} + b_{i+91} + b_{i+96} + \\ & + b_{i+3}b_{i+67} + b_{i+11}b_{i+13} + b_{i+17}b_{i+18} + \\ & + b_{i+27}b_{i+59} + b_{i+40}b_{i+48} + b_{i+61}b_{i+65} + \\ & + b_{i+68}b_{i+84}. \end{aligned}$$

The 256 memory elements in the two shift registers represent the state of the cipher. From this state, 9 variables are taken as input to a Boolean function, $h(x)$. Two inputs to $h(x)$ are taken from the NFSR and seven are taken from the LFSR. This function is of degree 3 and very simple. It 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, x_1, x_2, x_3, x_4, x_5, x_6, x_7 and 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} respectively. The output function is defined as

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

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

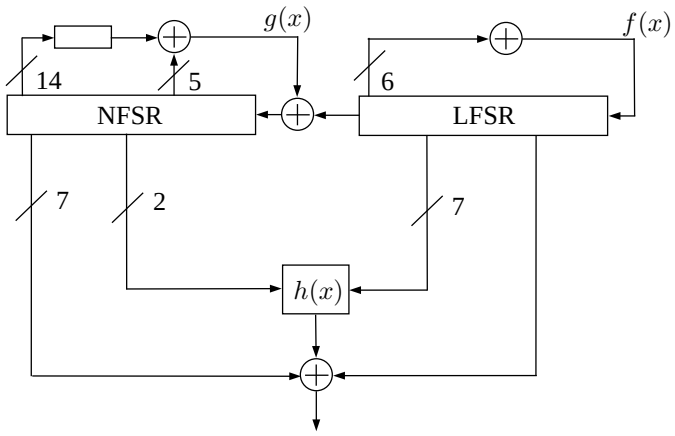

Fig. 1. An overview of the cipher.

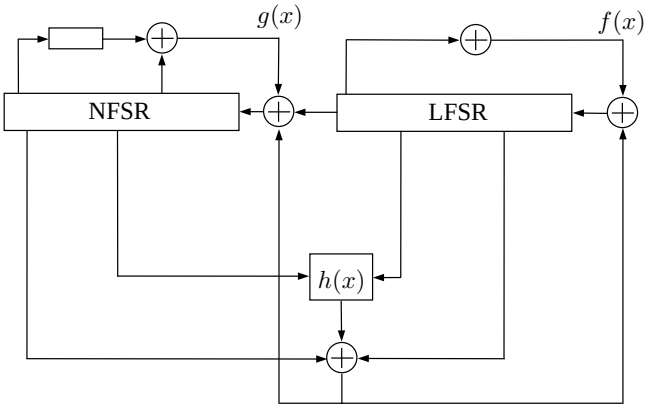

Fig. 2. The key initialization.

A. Key and IV Initialization

Before keystream is generated the cipher must be initialized with the key and the IV. Let the bits of the key, k, be denoted $k_i,\ 0 \leq i \leq 127$ and the bits of the IV be denoted $IV_i,\ 0 \leq i \leq 95$. Then the initialization of the key and IV is done as follows. The 128 NFSR elements are loaded with the key bits, $b_i = k_i,\ 0 \leq i \leq 127$, then the first 96 LFSR elements are loaded with the IV bits, $s_i = IV_i,\ 0 \leq i \leq 95$. The last 32 bits of the LFSR is filled with ones, $s_i = 1,\ 96 \leq i \leq 127$. After loading key and IV bits, the cipher is clocked 256 times without producing any keystream. Instead the output function is fed back and xored with the input, both to the LFSR and to the NFSR, see Fig. 2.

III. Throughput Rate

Both shift registers are regularly clocked so the cipher will output 1 bit/clock. Using regular clocking is an advantage compared to stream ciphers which uses irregular clocking or decimation of the output sequence, since no hardware

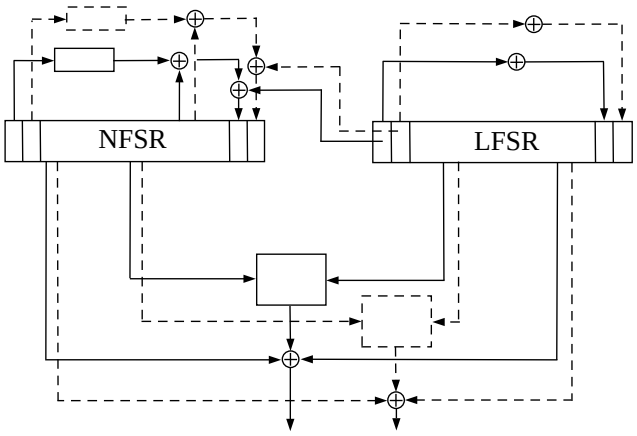

Fig. 3. The cipher when the speed is doubled.

consuming output buffer is needed. Regular clocking is also an advantage when considering side-channel attacks. It is possible to increase the speed of the cipher at the expense of more hardware. This is an important feature of the Grain family of stream ciphers compared to many other stream ciphers. Increasing the speed can very easily be done by just implementing the small feedback functions, $f(x)$ and $g(x)$, and the output function several times. In order to simplify this implementation, the last 31 bits of the shift registers, $s_i,\ 97 \leq i \leq 127$ and $b_i,\ 97 \leq i \leq 127$ are not used in the feedback functions or in the input to the output function. This allows the speed to be easily multiplied by up to 32 if a sufficient amount of hardware is available. For more discussion about the hardware implementation of Grain-128, we refer to section VI. An overview of the implementation when the speed is doubled can be seen in Fig. 3. Naturally, the shift registers also need to be implemented such that each bit is shifted t steps instead of one when the speed is increased by a factor t. By increasing the speed by a factor 32, the cipher will output 32 bits/clock. Since, in the key initialization, the cipher is clocked 256 times, the possibilities to increase the speed is limited to factors ≤ 32 that are divisible by 256. The number of clockings needed in the key initialization phase is then $256/t$. Since the output and feedback functions are small, it is quite feasible to increase the throughput in this way.

IV. Security Evaluation

The most important property of a stream cipher is its resistance to different cryptanalytic attacks. This section will give a small description of possible attacks. This results given in this section is the basis for the specific design choices behind the different functions and parameters used in Grain-128.

A. Linear Approximations

Linear sequential circuit approximations was first introduced by Golić, see [3]. It is shown that it is always possible to find a linear function of output bits that is unbalanced. For linear approximations, we study the structure of the Grain design in general. We consider an arbitrary choice of functions $g(\cdot)$, $h(\cdot)$ and $f(\cdot)$. The number of taps taken from the two registers in the function $h(\cdot)$ is also arbitrary. Here, the function $f(\cdot)$ is a primitive generating polynomial used for the LFSR. A Boolean nonlinear function $g(\cdot)$ is applied to generate a new state of the NFSR. Finally, the keystream is the output of another Boolean function $h(\cdot)$. Note that, to simplify notation, the function $h(\cdot)$ in this section also includes the linear terms added in the output function.

The results in this section was first given in [4] as follows. Let $A_g(\cdot)$ and $A_h(\cdot)$ be linear approximations for $g(\cdot)$ and $h(\cdot)$ with the biases ϵ_g and ϵ_h, respectively. I.e.,

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

Then, there exists a time invariant linear combination of the keystream bits and LFSR bits, such that this equation has the bias

$$\epsilon = 2^{(\eta(A_h)+\eta(A_g)-1)} \cdot \epsilon_g^{\eta(A_h)} \cdot \epsilon_h^{\eta(A_g)}, \tag{3}$$

where $\eta(a(\cdot))$ is the number of the NFSR state variables used in some function $a(\cdot)$. This bias can not immediately be used in cryptanalysis since also the LFSR has to be taken into account. However, as soon as the bias ϵ is large, a distinguishing or even a key-recovery attack can be mounted by e.g., finding a low weight parity check equation for the LFSR. When we talk about correlation attacks of different kinds, it has been shown in [4] that the strength of Grain is directly based on the difficulty of the general decoding problem (GDP), well-known as a hard problem. A set of time-memory trade-off solutions for the GDP is widely discussed in the literature, e.g., in [5]–[9].

Since for any function $a(\cdot)$ a biased linear approximation $A_a(\cdot)$ can always be found, it means that Grain will always produce biased samples from the keystream, according to (3). An important issue is to choose the functions $g(\cdot)$ and $h(\cdot)$ such that that bias is extremely small to prevent any possible attack faster than exhaustive search.

B. Algebraic Attacks

Algebraic attacks on stream ciphers have received much attention recently since they can be very efficient if the designer is not careful. A filter generator using only an LFSR and a nonlinear Boolean output function, $h(\cdot)$, could be very vulnerable to algebraic attacks, see [10]. In Grain-128, an NFSR is used to introduce nonlinearity together with the function $h(\cdot)$. Solving equations for the initial 256 bit state is not possible due to the nonlinear update of the NFSR. The algebraic degree of the output bit expressed in initial state bits will be large in general and also varying in time. This will defeat any algebraic attack on the cipher.

C. Time-Memory-Data Trade-off Attack

A generic time-memory-data trade-off attack on stream ciphers costs $O(2^{n/2})$, see [11], where n is the number of inner state variables in the stream cipher. In Grain-128, the two shift registers are of size 128 each so the total number of state variables is 256. Thus, the expected complexity of a time-memory-data trade-off attack is not lower than $O(2^{128})$.

D. Fault Attacks

Fault attacks are among the strongest attacks possible on any stream cipher. They were introduced in [12] and have shown to be efficient against many known stream cipher constructions. It is debatable to which extent these attacks are actually practical. One scenario in a fault attack is to allow the adversary to apply some bit flipping faults to one of the shift registers at his will. The attacker will, however, not have total control over the number of faults and the exact location of the faults. Neither will he have full knowledge of the number and position of the faults after the faults were introduced. An even stronger assumption is that the adversary is able to flip exactly one bit, but in a location which he can not control. He can also reset the cipher and introduce a new fault. We make the strongest assumption possible, namely that the adversary can introduce one single fault in a location of the LFSR that he can somehow determine. Note that this assumption may not be at all realistic. The aim is to look at the input-output properties for $h(\cdot)$, and to get information about the inputs from known input-output pairs. As long as the difference does not propagate to position b_{i+95} the difference that can be observed in the output is coming only from inputs of $h(\cdot)$ from the LFSR. If the attacker is able to reset the cipher many times, each time introducing a new fault in a known position that he can guess from the output difference, then we can not preclude that he will get information about a subset of state bits in the LFSR. Considering the more realistic assumption that the adversary is not able to control the number of faults that have been inserted then it seems more difficult to determine the induced difference from the output differences. It is also possible to introduce faults in the NFSR. These faults will never propagate to the LFSR, but the faults introduced here will propagate nonlinearly in the NFSR and their evolution will be harder to predict. Thus, introducing faults into the NFSR seems more difficult than into the LFSR.

V. Design Choices

In this section we give the details behind the choices for the parameters used in Grain-128. Section IV clearly shows that a proper choice of design parameters is important.

Size of the LFSR and the NFSR. The size of the key in Grain-128 is 128 bits. Because of the simple and generic time-memory-data trade-off attack, the internal state must be at least twice as large as the size of the key. Therefore, we choose the LFSR and the NFSR to be of size 128 bits.

Speed Acceleration. Although the binary hardware implementation of Grain is small and fast, its speed can still be increased significantly. The functions $f(\cdot)$, $g(\cdot)$, and $h(\cdot)$ can be implemented several times, so that several bits can be produced in parallel at the same time. In Grain-128 we explicitly allow up to 32 times speed acceleration. Many software oriented ciphers are word based with a word size of 32 bits. These ciphers output 32 bits in every clock or iteration. If needed, Grain-128 can also be implemented to output 32 bits/clock. For a simple implementation of this speed acceleration the functions $f(\cdot)$, $g(\cdot)$, and $h(\cdot)$ should not use variables taken from the first 31 taps of the LFSR and the NFSR. Obviously, speed acceleration is a trade-off between speed and hardware complexity. Speed can additionally be increased even more, by allowing the internal state to be increased proportionally. For more discussion on the throughput, see Section III.

Choice of $f(\cdot)$. This function is the generating polynomial for the LFSR, thus, it must be primitive. It has been shown (in e.g., [13]) that if the function $f(\cdot)$ is of low weight, there exist different correlation attacks. Therefore, the number of taps to be used for the generating function $f(\cdot)$ should be larger than five. A large number of taps is also undesirable due to the complexity of the hardware implementation.

Choice of $g(\cdot)$. This Boolean function is used for the NFSR, generating a nonlinear relation of the state of the register. The design of this function must be carefully chosen so that the attack given in Section IV-A will not be possible. Recall that the bias of the output will depend on the number of terms in the best linear approximation of $g(\cdot)$. It will also depend on the bias of this approximation. To increase the number of terms in the best linear approximation, the resiliency of the function must be high. On the other hand, to have as small bias as possible in the best approximation, the function should have high nonlinearity. It is well known that a bent function has the highest possible nonlinearity. However, bent functions can not be balanced. In order to have both high resiliency and nonlinearity, a highly resilient (linear) function is used together with a bent function. The bent function $b(\cdot)$ chosen is the function

$$b(x) = x_0x_1 + x_2x_3 + x_4x_5 + x_6x_7 + x_8x_9 + x_{10}x_{11} + x_{12}x_{13}.$$

This function has nonlinearity 8128. To increase the resiliency, 5 linear terms are added to the function. This will result in a balanced function with resiliency 4 and nonlinearity $2^5 \cdot 8128 = 260096$. This is an easy way to construct functions with high resiliency and nonlinearity. Another important advantage of this function is that it is very small and cheap to implement in hardware. The best linear approximation is any linear function using at least all the linear terms. There are 2^{14} such functions and they have bias $\epsilon_g = 2^{-8}$.

Choice of output function. The output function consists of the function $h(x)$ and terms added linearly from the two shift registers. This guarantees that the output will depend on the state of both registers. The function $h(x)$ takes input from both the LFSR and the NFSR. Similar to the function $g(\cdot)$, the bias of the output will depend on the number of terms in the best linear approximation of this function and also the bias of this approximation. Hence, this function has the same design

TABLE I

THE GATE COUNT USED FOR DIFFERENT FUNCTIONS.

Function	*Gate Count*
NAND2	1
NAND3	1.5
XOR2	2.5
D flip flop	8

TABLE II

THE ESTIMATED GATE COUNT IN AN ACTUAL IMPLEMENTATION.

Gate Count	*Speed Increase*					
Building Block	1x	2x	4x	8x	16x	32x
LFSR	1024	1024	1024	1024	1024	1024
NFSR	1024	1024	1024	1024	1024	1024
$f(\cdot)$	12.5	25	50	100	200	400
$g(\cdot)$	37	74	148	296	592	1184
Output func	35.5	71	142	284	568	1136
Total	2133	2218	2388	2728	3408	4768

criteria as $g(\cdot)$. The function $h(x)$ has nonlinearity 240 and since in total 8 variables are added linearly the output function has in total nonlinearity $2^8 \cdot 240 = 61440$. The function $h(x)$ is not balanced and the best linear approximations have bias $\epsilon_h = 2^{-5}$. There are in total 256 linear approximations with this bias.

VI. HARDWARE COMPLEXITY

The Grain family of stream ciphers is designed to be very small in hardware. In this section we give an estimate of the gate count resulting from a hardware implementation of the cipher. The gate count for a function depends on the complexity and functionality. The numbers are no natural constants and will depend on the implementation in an actual chip. Usually, the gate count is based on a 2 input nand gate which is defined to have gate count 1. Hence, the gate count can be seen as the equivalent number of nand gates in the implementation. Table I lists the equivalent gate count for the building blocks used in our estimation.

The total gate count for the different functions can be seen in Table II. This is just an estimate and the numbers are not exact, e.g., the multiplexers needed in order to switch between key/IV loading, initialization and keystream generation are not included in the count. Also, two extra xors are needed in key initialization mode. However, excluding these things results in insignificant deviations from the real values. The exact number of gates needed for each function will depend on the implementation anyway.

VII. CONCLUSION

A new stream cipher, Grain-128, has been presented. The design is a new member in the family of Grain stream ciphers. The size of the key is 128 bits and the size of the IV is 96 bits. The design parameters has been chosen based on theoretical arguments for linear approximations and other possible attacks. Grain-128 is very well suited for hardware environments where low gate count, low power consumption and small chip area are important requirements. One can very easily increase the speed of the cipher at the expense of extra hardware. To our knowledge, there is no 128 bit cipher offering the same security as Grain-128 and a smaller gate count in hardware.

ACKNOWLEDGMENT

The fourth author is supported by Hasler Foundation `www.haslerfoundation.ch` under project number 2005.

REFERENCES

[1] ECRYPT, "eSTREAM: ECRYPT Stream Cipher Project, IST-2002-507932," Available at http://www.ecrypt.eu.org/stream/.

[2] M. Hell, T. Johansson, and W. Meier, "Grain - a stream cipher for constrained environments." *International Journal of Wireless and Mobile Computing, Special Issue on Security of Computer Network and Mobile Systems.*, 2006.

[3] J. Golić, "Intrinsic statistical weakness of keystream generators," in *Advances in Cryptology—ASIACRYPT'94*, 1994, pp. 91–103.

[4] A. Maximov, "Cryptanalysis of the "Grain" family of stream ciphers," in *ACM Symposium on InformAtion, Computer and Communications Security (ASIACCS'06)*, 2006, pp. 283–288.

[5] T. Johansson and F. Jönsson, "Fast correlation attacks based on turbo code techniques," in *Advances in Cryptology—CRYPTO'99*, ser. Lecture Notes in Computer Science, M. Wiener, Ed., vol. 1666. Springer-Verlag, 1999, pp. 181–197.

[6] ——, "Fast correlation attacks through reconstruction of linear polynomials," in *Advances in Cryptology—CRYPTO 2000*, ser. Lecture Notes in Computer Science, M. Bellare, Ed., vol. 1880. Springer-Verlag, 2000, pp. 300–315.

[7] V. Chepyzhov, T. Johansson, and B. Smeets, "A simple algorithm for fast correlation attacks on stream ciphers," in *Fast Software Encryption 2000*, ser. Lecture Notes in Computer Science, B. Schneier, Ed., vol. 1978. Springer-Verlag, 2000, pp. 181–195.

[8] M. J. Mihaljević, M. Fossorier, and H. Imai, "Fast correlation attack algorithm with list decoding and an application," *Lecture Notes in Computer Science*, vol. 2355, pp. 196–210, 2002.

[9] P. Chose, A. Joux, and M. Mitton, "Fast correlation attacks: An algorithmic point of view," *Lecture Notes in Computer Science*, vol. 2332, pp. 209–221, 2002.

[10] N. Courtois and W. Meier, "Algebraic attacks on stream ciphers with linear feedback," in *Advances in Cryptology—EUROCRYPT 2003*, ser. Lecture Notes in Computer Science, E. Biham, Ed., vol. 2656. Springer-Verlag, 2003, pp. 345–359.

[11] A. Biryukov and A. Shamir, "Cryptanalytic time/memory/data tradeoffs for stream ciphers," in *Advances in Cryptology—ASIACRYPT 2000*, ser. Lecture Notes in Computer Science, T. Okamoto, Ed., vol. 1976. Springer-Verlag, 2000, pp. 1–13.

[12] J. Hoch and A. Shamir, "Fault analysis of stream ciphers." in *CHES 2004*, ser. Lecture Notes in Computer Science, vol. 3156. Springer-Verlag, 2004, pp. 240–253.

[13] A. Canteaut and M. Trabbia, "Improved fast correlation attacks using parity-check equations of weight 4 and 5," in *Advances in Cryptology—EUROCRYPT 2000*, ser. Lecture Notes in Computer Science, B. Preneel, Ed., vol. 1807. Springer-Verlag, 2000, pp. 573–588.

APPENDIX

Grain is a bit oriented design and for simplicity of reading the test vectors below are translated to hexadecimal strings. Hence, the most significant bit of the first hex value represents index 0.

```
Key:       00000000000000000000000000000000
IV:        000000000000000000000000
Keystream: 0fd9deefeb6fad437bf43fce35849cfe
```

```
Key:       0123456789abcdef123456789abcdef0
IV :       0123456789abcdef12345678
Keystream: db032aff3788498b57cb894fffb6bb96
```

The reference implementation uses bytes and in this case the least significant bit of the first byte will be treated as index 0. The test vectors are then given as:

```
Key:       00000000000000000000000000000000
IV:        000000000000000000000000
Keystream: f09b7bf7d7f6b5c2de2ffc73ac21397f
```

```
Key:       0123456789abcdef123456789abcdef0
IV :       0123456789abcdef12345678
Keystream: afb5babfa8de896b4b9c6acaf7c4fbfd
```