A New Version of Grain-128 with Authentication

Ågren, Martin; Hell, Martin; Johansson, Thomas; Meier, Willi

2011

A New Version of Grain-128 with Authentication

Martin Ågren[1], Martin Hell[1], Thomas Johansson[1], and Willi Meier[2]

[1] Dept. of Electrical and Information Technology, Lund University,
P.O. Box 118, 221 00 Lund, Sweden
{martin.agren,martin,thomas}@eit.lth.se
[2] FHNW, CH-5210 Windisch, Switzerland
willi.meier@fhnw.ch

Abstract. A new version of the stream cipher Grain-128 is proposed. The new version, Grain-128a, is strengthened against all known attacks and observations on the original Grain-128, and has built-in support for authentication. The changes are modest, keeping the basic structure of Grain-128. This gives a high confidence in Grain-128a and allows for easy updating of existing implementations.

Keywords: Grain-128a, Stream Cipher, Cryptographic Primitive, Hardware Attractive, Lightweight, Message Authentication, MAC

1 Introduction

Many stream ciphers have been proposed over the years, and new designs are published as cryptanalysis enhances our understanding of how to design safer and more efficient primitives. While the NESSIE project failed to name a "winner" after evaluating several new designs around ten years ago, the eSTREAM project finally decided on two portfolios of promising candidates. One of these aimed at hardware attractive constructions, and consists of Grain [1], Trivium [2], and MICKEY [3].

Grain was designed by Hell, Johansson, and Meier and is notable for its extremely small hardware representation. During the initial phase of the eSTREAM project, the original version, Grain v0, was strengthened. The final version is known as Grain v1.

Like the other portfolio ciphers, Grain v1 is modern in the sense that it allows for public IVs, yet they only use 80-bit keys. Recognizing the emerging need for 128-bit keys, Hell, Johansson, Maximov, and Meier proposed Grain-128 [4] supporting 128-bit keys and 96-bit IVs. The design is akin to that of 80-bit Grain, but noticeably, the nonlinear parts of the cipher have smaller degrees than their counterparts in Grain v1.

We specify a new version of Grain-128, namely Grain-128a. The new stream cipher has native support for authentication, and is expected to be comparable to the old version in hardware performance.

The authentication supports variable tag-sizes w up to 32 bits, and varying w does not affect the keystream generated by Grain-128a.

Grain-128a uses slightly different non-linear functions in order to strengthen it against the known attacks and observations on Grain-128. Existing implementations of Grain-128 can be reused to a very large extent as the changes, summarized in Section 6, are modest. This also allows us to have a high confidence in Grain-128a, as the cryptanalysis carries over from Grain-128.

The details of the design are specified in Section 2. The throughput is discussed in Section 3, and a security analysis is performed in Section 4. The design choices are motivated theoretically in Section 5, and Section 6 details the differences to Grain-128. The hardware performance is discussed in Section 7, while Section 8 concludes the paper. The appendix contains several test vectors.

2 Design Details

Grain-128a consists of a mechanism that produces a pre-output stream, and a secondary, optional, mechanism that handles the authentication. Fig. 1 depicts an overview of the building blocks of the pre-output generator, while the authentication mechanism is sketched in Fig. 3. The pre-output generator in turn is constructed out of three main building blocks, namely an LFSR, an NFSR and a pre-output function. We denote by $s_i, s_{i+1}, \ldots, s_{i+127}$ the contents of the LFSR. Similarly, the content of the NFSR is denoted by $b_i, b_{i+1}, \ldots, b_{i+127}$. Together, the 256 memory elements in the two shift registers represent the state of the pre-output generator.

The primitive feedback polynomial of the LFSR, denoted $f(x)$, 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 defined as

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

To once more remove any possible ambiguity we also give the rule for updating the NFSR.

$$\begin{aligned} b_{i+128} = s_i &+ b_i + b_{i+26} + b_{i+56} + b_{i+91} + b_{i+96} \\ &+ b_{i+3}b_{i+67} + b_{i+11}b_{i+13} + b_{i+17}b_{i+18} \\ &+ b_{i+27}b_{i+59} + b_{i+40}b_{i+48} + b_{i+61}b_{i+65} \\ &+ b_{i+68}b_{i+84} + b_{i+88}b_{i+92}b_{i+93}b_{i+95} \\ &+ b_{i+22}b_{i+24}b_{i+25} + b_{i+70}b_{i+78}b_{i+82}. \end{aligned}$$

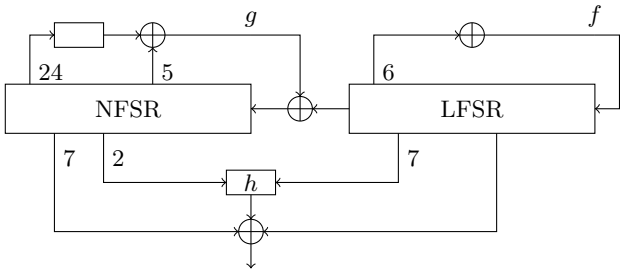

Fig. 1. An overview of the pre-output generator.

Note that the update rule contains the bit s_i which is output from the LFSR and masks the input to the NFSR, while it was left out in the feedback polynomial.

Nine state variables are taken as input to a Boolean function, $h(x)$: two bits come from the NFSR and seven from the LFSR. This function is defined as

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

where the variables $x_0, \ldots, x_8$ correspond to, respectively, the state variables 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+94}. The pre-output function is defined as

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

where $\mathcal{A} = \{2, 15, 36, 45, 64, 73, 89\}$. The output function is defined as

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

meaning that we pick every second bit as output of the cipher after skipping the first 64 bits. Those 64 initial bits and the other half will (may) be used for authentication, see Section 2.2.

2.1 Key and IV Initialization

Before keystream is generated the cipher must be initialized with the key and the IV. Denote the bits of the key as k_i, $0 \leq i \leq 127$ and the IV bits IV_i, $0 \leq i \leq 95$. Initialization of the key and IV is done as follows. The 128 NFSR elements are loaded with the key bits, $b_i = k_i$, $0 \leq i \leq 127$, and 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 are filled with ones and a zero, $s_i = 1$, $96 \leq i \leq 126$, $s_{127} = 0$. Then, the cipher is clocked 256 times without producing any keystream. Instead the pre-output function is fed back and xored with the input, both to the LFSR and to the NFSR, see Fig. 2.

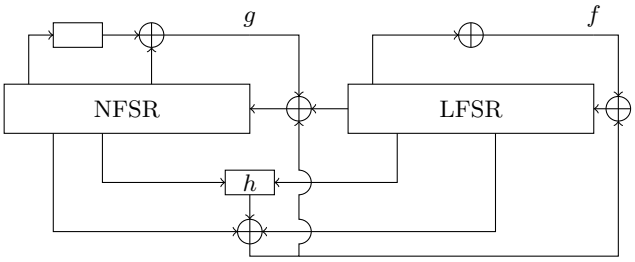

Fig. 2. The key initialization.

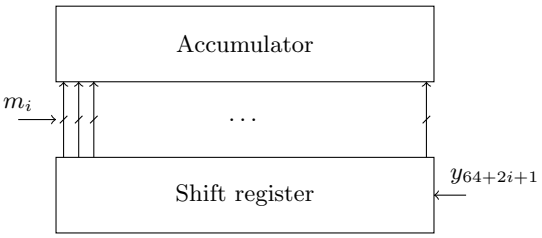

Fig. 3. An overview of the authentication as it is clocking in message and pre-output bits.

2.2 Authentication

Assume that we have a message of length L defined by the bits $m_0, \ldots, m_{L-1}$. Set $m_L = 1$. Note that $m_L = 1$ is the padding, which is crucial for the security of the authentication as it ensures that $\boldsymbol{m}$ and $\boldsymbol{m}||0$ have different tags.

In order to provide authentication, two registers of size 32 are used. They are called the accumulator and the shift register. The content of the accumulator at time i is denoted by $a_i^0, \ldots, a_i^{31}$. The content of the shift register is denoted by $r_i, \ldots, r_{i+31}$. The accumulator is initialized through $r_i = y_i,\ 0 \leq i \leq 31$, and the shift register is initialized through $a_0^j = y_{32+j},\ 0 \leq j \leq 31$. The shift register is updated as $r_{i+32} = y_{64+2i+1}$. The accumulator is updated as $a_{i+1}^j = a_i^j + m_i r_{i+j}$ for $0 \leq j \leq 31$ and $0 \leq i \leq L$.

The final content of the accumulator, $a_{L+1}^0, \ldots, a_{L+1}^{31}$, is denoted the *tag* and can be used for authentication. We write $t_i = a_{L+1}^i,\ 0 \leq i \leq 31$.

See Fig. 3 for a graphical representation of the authentication mechanism.

To guarantee an implementation-independent use of shorter tags, we define w-bit tags through $t_i^{(w)} = t_{32-w+i},\ 0 \leq i \leq w-1$. This amounts to using the right-most part of the tag in Fig. 3.

3 Throughput Rate

Both shift registers are regularly clocked so the cipher will output one bit every second clock. This regular clocking is an advantage, both in terms of performance

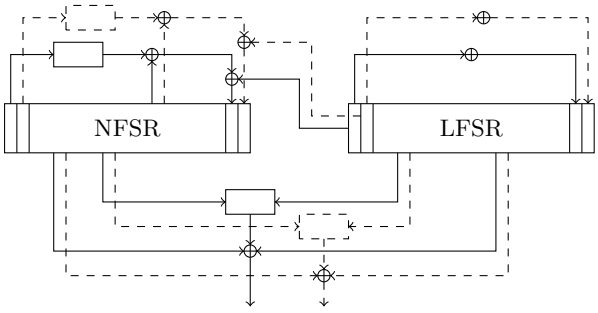

Fig. 4. The cipher when the speed is doubled.

and resistance to side-channel attacks, compared to using irregular clocking or decimation.

An important feature of the Grain family of stream ciphers is that the speed can be increased at the expense of more hardware. This requires the small feedback functions, $f(x)$ and $g(x)$, and the pre-output function to be implemented several times. To aid this, the last 31 bits of the shift registers, $s_i, b_i, 97 \leq i \leq 127$ are not used in the respective feedback function or in the input to the pre-output function. This allows the speed to be easily multiplied by up to 32 if a sufficient amount of hardware is available.

An overview of the implementation when the speed is doubled can be seen in Fig. 4. The shift registers also need to be implemented such that each bit is shifted t steps instead of just one when the speed is increased by a factor t. The possibilities to increase the speed is limited to powers of two as t needs to divide e.g., the initialization count, which is 256, and the authentication initialization, which is another 64 basic clockings. Since the pre-output and feedback functions are small, it is quite feasible to increase the throughput in this way. By increasing the speed by a factor 32, the cipher will output 16 bits/clock.

For more discussion about the hardware implementation of Grain-128a, we refer to Section 7.

4 Security Evaluation

Excellent hardware performance is of little use if the cipher is not secure. We outline several possible cryptanalytic attacks, and build upon these insights to decide on the different functions and parameters used in Grain-128a.

In the following, we will consider the pre-output stream y_i, as the keystream z_i is just as good from a security point of view (but half the length), and the authentication will rely on the security of the pre-output stream.

4.1 Linear Approximations

Golić [5] realized that in any stream cipher, one can always find some linear combination of the output bits that is unbalanced, meaning it is more often e.g.,

0 than 1. In this section, we consider the general Grain design, ignoring specifics such as the exact choices of f, g, and h. The function f is of course restricted to being a primitive polynomial, as it is the feedback function of the LFSR. Updating the NFSR is similarly made through g, and the output is created through h. To simplify notation, this section denotes by h the entire output function, i.e., it includes the bits added linearly in the output function.

Maximov studied this general structure in [6] and introduced A_g and A_h as linear approximations for g and h with biases ϵ_g and ϵ_h, respectively. That is,

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

Then, a time invariant linear combination of the keystream bits and LFSR bits exists, and the bias of this equation is

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

where $\eta(a)$ is the number of the NFSR state variables used in the function a. The LFSR taps have not been accounted for, and this bias can not be readily used in any attack. However, by summing shifted versions of this function, so that the LFSR contributions add up to zero, a practical attack can be mounted, at least if the bias ϵ of the new linear equation is large. Finding a low weight parity check equation [7–10] for the LFSR improves this ϵ at the expense of requiring longer keystream, and the pre-computation of finding such a parity check equation. Maximov also showed that the strength of Grain against correlation attacks is based on the difficulty of the general decoding problem (GDP), which is well-known to be a hard problem. Various time-memory trade-off approaches to the GDP have been discussed in e.g., [11–15].

As one can always find a biased linear approximation A_a for any function a, one can never eliminate the biased nature of Grain's output. It thus comes down to choosing particular functions g and h such that this bias is extremely small, so that the resulting attack will be a less promising choice than a simple brute force.

4.2 Algebraic Attacks

The individual bits in the pre-output stream can be expressed as functions of the initial state, i.e., the state bits just prior to pre-output generation begins. Thus, with access to a stream of such bits, an attacker can attempt to solve the corresponding system of equations. If Grain-128a did not contain the NFSR, i.e., it was a basic filter generator, such algebraic attacks could be very successful. However, Grain-128a *does* use an NFSR, which introduces much more nonlinearity, together with h, see e.g., [16]. Solving equations for the initial 256 bit state is not possible due to the nonlinear update of the NFSR and the NFSR state bits used nonlinearly in h [17].

4.3 Time-Memory-Data Trade-off Attack

A generic attack that can be applied to a large class of cryptographic primitives, and on stream ciphers in particular, is the time-memory-data trade-off attack. The cost is $O(2^{n/2})$ where n is the size of the state [18]. As the state in Grain-128a is of size 256, the expected complexity of such an attack is at least $O(2^{128})$, which exceeds that of brute force.

4.4 Fault Attacks

Fault attacks were introduced in [19] and have been efficient against many known stream cipher constructions. Whether they are practical is not so clear: one scenario in a fault attack is to allow the adversary to introduce some bit flipping faults to one of the shift registers. We note that faults in the NFSR should be harder to trace than faults in the LFSR, and the strongest assumption possible is therefore that the adversary can introduce a single fault in a location of the LFSR that he can somehow determine. When the fault propagates to position b_{i+95}, the difference has spread to the NFSR-related output, and is soon introducing nonlinearities. Until that point in time, the difference observed in the output is coming only from inputs of h from the LFSR. Allowing the adversary to reset Grain-128a many times, each time introducing a new fault, might enable him to acquire information about some subset of LFSR bits. Slightly more realistic assumptions on the ability to introduce a known number of faults makes it more difficult to deduce LFSR bits from the differences in output.

4.5 Side-Channel Attacks

Any attacker that can observe some signal that is emitted from the implementation of a cryptographic primitive — most often power consumption or some function thereof — and that is dependent on the inner calculations, may be able to deduce the numbers, bits, etc. used in these calculations and thus, e.g., the key or the message.

We note that the authentication mechanism performs work on two vastly different levels of power consumption. Viewing a power diagram of a naive implementation that processes one message bit every clocking, it should be easy to tell apart ones from zeros.

Just as with any other cryptographic primitive, care must be taken to protect an actual implementation of Grain-128a against side-channel attacks [20].

4.6 Weak Key-IV pairs

Zhang and Wang [21] have shown that there are 2^{96} weak key-IV pairs in Grain-128, each leading to an all-zero LFSR after the initialization phase. They have also demonstrated how to distinguish such keystream, and how to recover the initial state.

We note that the IV is normally assumed to be public, and that the probability of using a weak key-IV pair is 2^{-128}. Any attacker guessing this to happen and then launching a rather expensive attack, is much better off just guessing a key.

4.7 The Authentication

It has been shown that an attacker who replaces a message-tag pair $(\boldsymbol{m}, \boldsymbol{t})$ with a modified version $(\boldsymbol{m} + \boldsymbol{a}, \boldsymbol{t} + \boldsymbol{b})$ has a success probability bounded by $2^{-32} + 2\epsilon$, where ϵ measures the randomness in the keystream sequence used for authentication. Details are available in [22, 23]. From (1) in Section 4.1 and specific values of ϵ_g, ϵ_h given later, we know that $\epsilon \ll 2^{-32}$ in our case. We can therefore claim that the success probability of this substitution attack is bounded by approximately 2^{-32}, and that the best attack is to basically guess the tag for any message. The attack probability is similarly bounded by approximately 2^{-w} for w-bit tags.

From the work in [22,24], it is also clear that avoiding reuse of the key-IV pair is crucial to the security of the authentication, just as it is for the encryption. An attacker who is able to tweak a message-tag pair and have it accepted (this happens with probability 2^{-w}) will be able to perform subsequent forgeries with probability 1 if the key-IV pair is reused.

The authentication mechanism is very similar to that in the 3GPP algorithm 128-EIA3 [25], which uses the stream cipher ZUC [26]. However, in 128-EIA3 two entirely different instances of ZUC are used. The IVs are similar or even equal, but two different keys are utilized: one for encryption and one for authentication. As encryption and authentication are performed simultaneously, one needs to utilize two implementations of ZUC or an expensive buffering. We consider our approach superior from a hardware point of view as the authentication and encryption share the pre-output stream of a single instance of Grain-128a.

Note also that a draft version of 128-EIA3 was broken by Fuhr et al. [27]. This attack does not apply to Grain-128a as it uses the technique mentioned in [22,27] to avoid the exploited problem. Thus, Grain-128a extracts the "one time pad", used to finalize the MAC, from the beginning of the pre-output stream.

[27] also wonders whether the IV is any problem — it is not; if the constant key, variable IV used with the authentication mechanism in Grain-128a was a problem, there would exist a strong distinguisher on the pre-output stream (and very likely the keystream) when Grain-128a is used in the most natural of modes for stream ciphers: constant key, variable IV.

5 Design Choices

From the above, it is apparent that it is crucial to select design parameters with great care. This section gives the details regarding the choices for the parameters and functions used in Grain-128a.

5.1 Size of the LFSR and the NFSR

The size of the key in Grain-128a is 128 bits. Considering the simple and generic time-memory-data trade-off attack, the size of the internal state must be at least twice that of the key. Thus, we decide on an internal state consisting of 256 bits. Dividing these equally between the NFSR and the LFSR is an apparent choice.

5.2 Speed Acceleration

As outlined previously, Grain-128a can be made significantly faster by implementing the functions f, g, and h several times. For a simple implementation of this speed acceleration up to a factor 32, these functions should be chosen not to use variables taken from the 31 right-most taps of the registers, as seen in Fig. 1.

5.3 Choice of f

As f should be the generating polynomial for the LFSR, and we want the period to be maximal, we need f to be primitive. It is well-known that polynomials of low weight can be exploited in various correlation attacks [28]. This implies that we should use many taps of the LFSR, but on the other hand, it is undesirable to use a very large number of taps, due to the hardware cost.

5.4 Choice of g

The purpose of this function is to create nonlinear relations between state bits, and we need to avoid the attack described in Section 4.1. The best linear approximation of g is of considerable interest, and for it to contain many terms, we need the resiliency of the function g to be high. We also need a high nonlinearity in order to obtain a small bias. To construct g, we thus use two functions — one with high nonlinearity and a linear one with high resiliency. The function

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

collecting the nonlinear terms, has nonlinearity 8356352. In order to strengthen the resiliency, 5 linear terms are added to the function. As a result, g is balanced, has nonlinearity $2^5 \cdot 8356352 = 267403264$ and resiliency 4. The set of best linear approximations is the set of linear functions where at least all the linear terms of g are present. This set is of size 2^{14} and all the functions in it have bias $\epsilon_g = 63 \cdot 2^{-15} < 2^{-9}$.

5.5 Choice of pre-output function

In order to make it certain that both registers affect the pre-output in each time step, terms from both registers are added linearly to the function h, which

also uses bits from both registers. The nonlinearity of h is 240 and adding 8 variables linearly yields a total nonlinearity of $2^8 \cdot 240 = 61440$. The best linear approximation has bias $\epsilon_h = 2^{-5}$, and there are in total 2^8 linear approximations of h with that bias.

5.6 Choice of authentication mechanism

[22] has made a thorough comparison of several approaches to authentication. It is clear that there is a choice to make between 1) register count, 2) security (substitution attack success probability), and 3) need of randomness (using a lot of keystream vs processing an initial seed). Since Grain-128a aims to be cost-efficient in hardware and yet very secure, the third parameter, keystream consumption during authentication, has been allowed to become high. Indeed, more pre-output bits are used for authentication than for encryption. There is, however, a very natural explanation for this under the assumption that whoever is about to implement the authentication mechanism in Grain-128a has already implemented its encryption mechanism. As mentioned in Section 3, it is quite cheap to double the rate of Grain-128a. Thus, the "cost" of upgrading from a hypothetical Grain-128a without any authentication to also using authentication amounts to the authentication mechanism itself and some additional gates in order to double the rate. Note that we could have created two keystreams from the NFSR and LFSR — one for encryption and one for authentication. This would in a sense allow us to double the throughput, but could have disastrous drawbacks if we are not very careful, and we have decided to stick with the much safer approach.

5.7 Choice of support for variable tag lengths

We suggest 32 bits as an upper tag size, as any application using Grain-128a is supposedly operating under some resource constraints and using e.g., 64 bits seems superfluous. Also, support for 64 bit tags would mean more clockings before keystream generation begins, even in the absence of authentication, since the correct number of pre-output bits need to be calculated and discarded. Note that a different approach could have been taken to allowing variable tag sizes: First initialize the cipher, then, perhaps, initialize the authentication using the desirable tag size, then produce keystream and, perhaps, authenticate. This could be done without discarding any pre-output bits. Using a certain key and IV, different tag sizes would naturally lead to different keystreams, but more worryingly, there would appear a possibility of using different pre-output bits for different purposes in different settings. Consider a known plaintext on a version without authentication. This would give the attacker the entire pre-output stream. If the receiver could be tricked into using 32 bit tags, the attacker could not only spoof an encryption (which is of course trivial with known keystream), but also the corresponding authenticating tag, thus elevating the supposed security of the scheme while still breaking it. (An attacker able to shorten the tags is of course very powerful, but that increasing the tag-size could be a security problem is

not at all obvious.) Considering this, we have decided to pre-determine which pre-output bits are used for what purpose. This does mean that applications with no or smaller tags will see a small overhead, but the overall confidence in the algorithm will be greater.

5.8 Choice of authentication initialization

We load the accumulator with the first 32 pre-output bits, and the register with the next 32. An alternative would have been to alternately load one bit into each register, i.e., $r_i = y_{2i}$, $a_0^i = y_{2i+1}$ for $0 \leq i \leq 31$. This would have meant that no matter w, a chunk of pre-output bits would have been discarded, and another chunk (the "end" before keystream generation begins) used to initialize the authentication mechanism. This could be interpreted as a prolonged initialization of Grain-128a. Our specification instead uses two separate chunks to load the accumulator and the register, respectively. With $0 < w < 32$, this means that the discarded pre-output bits are found in two separate blocks. We note, however, that this allows the accumulator to be loaded through the accumulating mechanism: one can load the first chunk of pre-output bits into the register and then "accumulate" it onto a zeroed accumulator. Later, the register is loaded with the bits that it should contain when Grain-128a is ready to produce keystream and authenticate message bits. Cryptanalytically, we note that the alternative approach would have allowed an attacker to access the xor of the two supposedly "weakest" pre-output bits: $r_0 + a_0^0 = y_0 + y_1$. Instead, the attacker can only learn these bits masked with bits that are produced later, being even more initialized: $r_0 + a_0^0 = y_0 + y_{32}$. This is not to imply that we do not trust the pre-output bits to be properly initialized — we only note that some bits are even more initialized, and it seems favourable to mix less and more initialized ones.

6 Differences From Grain-128

A number of changes have been made compared to Grain-128. In this section, we list and motivate each of these differences.

6.1 The Function $g(x)$

We have added three monomials: two of degree three and one of degree four. This is in response to the papers by Aumasson et al. [29] and Stankovski [30]. Both papers try to find sets of IV bits, where the remaining key and IV bits are fixed. E.g., with a set of 40 IV bits, one requests the first bit of the 2^{40} keystreams corresponding to the 2^{40} initial values. The first bit in the keystream is a function of the key and IV bits, and by processing these 2^{40} "first bits", one might be able to find some information on the secret key, at least if the function describing this bit is not complicated enough. It is natural to study instead the bits that are discarded during the initialization, as it is supposedly easier to find

any information in them, and it should be possible to get an idea of whether the initialization is strong enough. More details are available in the papers.

Stankovski defines a nonrandomness threshold and claims that there is nonrandomness throughout the full 256 rounds of initialization of Grain-128. This implies that the key and IV material is not properly mixed before keystream generation starts, and highlights that the initialization used too few clockings and/or too little nonlinearity.

As a consequence of adding authentication, the number of clockings before the encryption keystream is created grows from 256 to 320. The cryptographic properties needed for the authenticating bitstream, on the other hand, are not at all as strong as those that we demand from a stream cipher. (If the authentication mechanism would allow leakage of the pre-output bits $y_0, y_1, \ldots, y_{63}$, it would still be possible to access this slightly less initialized keystream. However, as an effect of the message padding, an attacker can only get hold of the xor of two (or more) windows of authenticating keystream material.)

We tried Stankovski's algorithm on variants of Grain-128a, analyzing the initialization, where we used several different candidate polynomials $g_i(x)$. We finally settled on one that had very good behaviour, both in terms of passing the nonrandomness tests of [30], and in terms of hardware implementation. The results are shown in Fig. 5. While this does not *prove* that Grain-128a mixes key and IV variables enough, it shows that the new design is less susceptible to this problem.

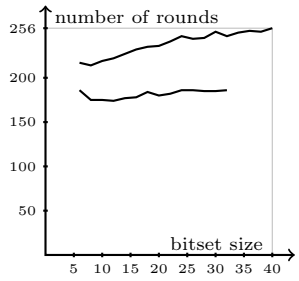

Fig. 5. The upper curve is Stankovski's result on Grain-128, where he starts from the optimal bitset of size 6, using only IV bits, and continuously add two bits according to his greedy algorithm in order to find good bitsets (cubes) where many initialization "output" bits xor to zero. The exact number of such bits is used to define the "nonrandomness." Finally he reaches a bitset of size 40 such that all initialization output bits xor to zero. The same strategy does not work as well on the initialization of Grain-128a. The curve starts lower and does not rise. We have launched an even more computationally demanding strategy of adding three bits rather than two in each step, but the curve resulting from that experiment shows the same non-growing tendency and has been excluded to avoid cluttering the figure.

6.2 IV Initialization

Setting $s_{127} = 0$ during IV initialization is a direct response to the observation in [31], where it was pointed out that using only ones to fill the IV register, there was a high probability that two very similar key-IV pairs would produce keystreams that were shifted variants of each other. As a direct consequence of this change, the previously known attacks [31–33] on Grain-128 are no longer applicable.

6.3 Authentication

We add authentication to Grain-128a.

6.4 Throughput Rate

The throughput rate is lower than in Grain-128, but it is quite easy to double it in response.

6.5 A Tap in the Pre-Output Function $h(x)$

Dinur and Shamir recently used techniques similar in spirit to Stankovski's in what they dub a dynamic cube attack [34]. For a fraction 2^{-10} of all keys, they are able to break the full key of Grain-128 by requesting, and storing, keystreams corresponding to 2^{59} chosen IVs. By nulling state bits, they are able to significantly simplify the equations that need to be solved in order to find the key bits.

This is partly due to the low degree of g, and partly to the choice of $x_4 = b_{i+95}$ and $x_8 = s_{i+95}$ in the pre-output function of Grain-128: these bits are multiplied together, but are very similar during the initialization phase when the suppressed pre-output bit is fed back to the registers. To mitigate this weakness, Grain-128a uses $x_4 = b_{i+95}$ and $x_8 = s_{i+94}$ in the pre-output function of Grain-128a.

Also note that Dinur and Shamir need access to the first bit generated after 256 initialization rounds. In Grain-128a, they will only be able to access the xor of this bit and at least one other, even more "initialized", pre-output bit. If they instead choose to use the first output bit of Grain-128a, it will have been produced after 320 rounds as opposed to 256.

7 Hardware Complexity

Grain-128a can be constructed using flip flops, xors, etc. and the gate counts required for these fundamental elements can be given as estimates at best. The exact cost of any implementation will depend on many parameters, such as the exact type of hardware used, the latest-and-greatest optimisations and tricks, and so on. Nonetheless, an estimate using some established measurements is highly useful in quickly assessing the feasibility of an algorithm.

Table 1. The gate count used for different functions.

Function	*Gate Count*
NAND2	1
NAND3	1.5
NAND4	2
XOR2	2.5
Flip flop	8

Table 2. The estimated gate count in an actual implementation. The total given for the w-bit MAC only relates to the authentication mechanism itself, not the pre-output generator needed to actually run it. The cost of the "accumulating logic" of the authentication mechanism is the same for speeds 1x and 2x — one implementation makes use of this logic every second clocking, and the other on each one.

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	12.5	25	50	100	200	400
g	49.5	99	198	396	792	1584
Pre-output function	35.5	71	142	284	568	1136
Accumulator	$8w$	$8w$	$8w$	$8w$	$8w$	$8w$
Register	$8w$	$8w$	$8w$	$8w$	$8w$	$8w$
Accumulating logic	$3.5w$	$3.5w$	$7w$	$14w$	$28w$	$56w$
Total (only enc.)	2145.5	2243	2438	2828	3608	5168
Total (only w-bit MAC)	$19.5w$	$19.5w$	$23w$	$30w$	$44w$	$72w$
Total (enc. + 32-bit MAC)	2769.5	2867	3174	3788	5016	7472

We use fundamental gate counts similar to those found in e.g., [4] where the nand gate with two inputs is defined to have unit gate count, and the other basic building elements are measured in equivalent nand gates. The list of the equivalent gate counts that have been used in deriving hardware numbers in this paper is found in Table 1.

Table 2 gives the gate counts for the larger building blocks of Grain-128a, as well as the total gate count for the entire Grain-128a. Basic combinatorics, e.g., the multiplexers needed to select between e.g., initialization of the pre-output generator, initialization of the authentication, and keystream generation, have not been included. The few extra xors needed during initialization have also been left out. As the gate counts are already estimates, these small numbers are not important.

7.1 Different Tag Sizes

It is possible to make the authentication mechanism consume less hardware resources, at the cost of increasing the success probability of the attack. The intuitive approach to producing a shorter tag is to simply chop the original one, discarding some bits. As Grain-128a aims for large flexibility and efficiency, the construction allows to not calculate these bits in the first place.

Note that care must be taken to discard the correct pre-output bits as to not affect the calculations of the remaining part of the authentication tag as well as the encryption keystream.

7.2 The Increase From Grain-128

Let us compare the hardware cost of an implementation without authentication, producing one bit per clock to that of Grain-128. This was the smallest possible Grain-128, and the increase in this cost should give us an idea of the cost of the extra flexibility and security added in Grain-128a.

Grain-128 required 2133 gate equivalents to implement the basic design, producing one bit of keystream per clocking. Compare this with the new design, looking at the version that produces one bit of keystream (two bits of pre-output) per clocking. According to Table 2, the number of gate equivalents is 2243. This is a mere five per cent increase. Note that while Grain-128 initialized in 256 clockings, Grain-128a (in 2x mode), generates keystream after only $(256 + 64)/2 = 160$ clockings.

8 Conclusion

A new stream cipher, Grain-128a, 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 have been chosen based on theoretical arguments for various possible attacks, and in light of known observations on older members of the family. With a low gate count, a low power consumption and a small chip area, Grain-128a is very well suited for hardware environments. The speed of the cipher can be increased very easily at the expense of extra hardware. Grain-128a is slightly more expensive in hardware than Grain-128, but offers better security and the possibility of adding authentication. To our knowledge, there is no 128 bit cipher offering the same security as Grain-128a and a smaller gate count in hardware.

Acknowledgment

The first author is supported by the Swedish Foundation for Strategic Research (SSF) through its Strategic Center for High Speed Wireless Communication at Lund.

The fourth author is supported in part by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center of the Swiss National Science Foundation under the grant number 5005-67322.

References

1. 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.*, vol. 2, no. 1, pp. 86–93, 2006.
2. C. De Cannière and B. Preneel, "Trivium," in *New Stream Cipher Designs*, ser. Lecture Notes in Computer Science, M. Robshaw and O. Billet, Eds., vol. 4986. Springer-Verlag, 2008, pp. 244–266.
3. S. Babbage and M. Dodd, "The MICKEY Stream Ciphers," in *New Stream Cipher Designs*, ser. Lecture Notes in Computer Science, M. Robshaw and O. Billet, Eds., vol. 4986. Springer-Verlag, 2008, pp. 191–209.
4. M. Hell, T. Johansson, A. Maximov, and W. Meier, "A Stream Cipher Proposal: Grain-128," in *International Symposium on Information Theory—ISIT 2006*. IEEE, 2006.
5. J. D. Golić, "Intrinsic statistical weakness of keystream generators," in *Advances in Cryptology—ASIACRYPT'94*, 1994, pp. 91–103.
6. 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.
7. W. Meier and O. Staffelbach, "Fast correlation attacks on certain stream ciphers," *Journal of Cryptology*, vol. 1, no. 3, pp. 159–176, 1989.
8. W. Penzhorn and G. Kühn, "Computation of low-weight parity checks for correlation attacks on stream ciphers," in *Cryptography and Coding - 5th IMA Conference*, ser. Lecture Notes in Computer Science, C. Boyd, Ed., vol. 1025. Springer-Verlag, 1995, pp. 74–83.
9. J. D. Golić, "Computation of low-weight parity-check polynomials," *Electronic Letters*, vol. 32, no. 21, pp. 1981–1982, October 1996.
10. D. Wagner, "A generalized birthday problem," in *Advances in Cryptology—CRYPTO 2002*, ser. Lecture Notes in Computer Science, M. Yung, Ed., vol. 2442. Springer-Verlag, 2002, pp. 288–303.
11. 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.
12. ——, "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.
13. 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.
14. 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.

15. 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.
16. 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.
17. C. Berbain, H. Gilbert, and A. Joux, "Algebraic and correlation attacks against linearly filtered non linear feedback shift registers," in *Selected Areas in Cryptography—SAC 2008*, ser. Lecture Notes in Computer Science, R. Avanzi, L. Keliher, and F. Sica, Eds., vol. 5381. Springer-Verlag, 2008, pp. 184–198.
18. 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.
19. 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.
20. W. Fischer, B. M. Gammel, O. Kniffler, and J. Velten, "Differential power analysis of stream ciphers," 2007, the State of the Art of Stream Ciphers, Workshop Record, SASC 2007, Bochum, Germany.
21. H. Zhang and X. Wang, "Cryptanalysis of stream cipher Grain family," Cryptology ePrint Archive, Report 2009/109, 2009, http://eprint.iacr.org/.
22. M. Ågren, M. Hell, and T. Johansson, "On hardware-oriented message authentication with applications towards RFID," in *Proceedings of the 2011 Workshop on Lightweight Security & Privacy: Devices, Protocols, and Applications (to appear)*, E. Savas, A. A. Selçuk, and U. Uludag, Eds., 2011.
23. H. Krawczyk, "New hash functions for message authentication," in *Advances in Cryptology—EUROCRYPT'95*. Springer-Verlag, 1995, pp. 301–310.
24. H. Handschuh and B. Preneel, "Key-recovery attacks on universal hash function based MAC algorithms," in *Advances in Cryptology—CRYPTO 2008*, ser. Lecture Notes in Computer Science, D. Wagner, Ed., vol. 5157. Springer-Verlag, 2008, pp. 144–161.
25. 3GPP, "Specification of the 3GPP confidentiality and integrity algorithms 128-EEA3 & 128-EIA3. Document 1: 128-EEA3 and 128-EIA3 specification," 3rd Generation Partnership Project (3GPP), TS, Jul. 2010. [Online]. Available: http://gsmworld.com/our-work/programmes-and-initiatives/fraud-and-security/gsm_security_algorithms.htm
26. ——, "Specification of the 3GPP confidentiality and integrity algorithms 128-EEA3 & 128-EIA3. Document 2: ZUC specification," 3rd Generation Partnership Project (3GPP), TS, Jul. 2010. [Online]. Available: http://gsmworld.com/our-work/programmes-and-initiatives/fraud-and-security/gsm_security_algorithms.htm
27. T. Fuhr, H. Gilbert, J.-R. Reinhard, and M. Videau, "A forgery attack on the candidate LTE integrity algorithm 128-EIA3 (updated version)," Cryptology ePrint Archive, Report 2010/618, 2010, http://eprint.iacr.org/.
28. 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.
29. J.-P. Aumasson, I. Dinur, L. Henzen, W. Meier, and A. Shamir, "Efficient FPGA implementations of high-dimensional cube testers on the stream cipher Grain-128,"

in *Workshop on Special Purpose Hardware for Attacking Cryptographic Systems (SHARCS'09)*, 2009.
30. P. Stankovski, "Greedy distinguishers and nonrandomness detectors," in *Progress in Cryptology—INDOCRYPT 2010*, ser. Lecture Notes in Computer Science, G. Gong and K. C. Gupta, Eds., vol. 6498. Springer-Verlag, 2010, pp. 210–226.
31. Ö. Küçük, "Slide resynchronization attack on the initialization of Grain 1.0," eSTREAM, ECRYPT Stream Cipher Project, Report 2006/044, 2006, http://www.ecrypt.eu.org/stream.
32. C. De Cannière, Ö. Küçük, and B. Preneel, "Analysis of Grain's initialization algorithm," in *Progress in Cryptology—AFRICACRYPT 2008*, ser. Lecture Notes in Computer Science, S. Vaudenay, Ed., vol. 5023. Springer-Verlag, 2008, pp. 276–289.
33. Y. Lee, K. Jeong, J. Sung, and S. Hong, "Related-key chosen IV attacks on Grain-v1 and Grain-128," in *13th Australasian Conference on Information Security and Privacy, ACISP 2008*, ser. Lecture Notes in Computer Science, Y. Mu, W. Susilo, and J. Seberry, Eds., vol. 5107. Springer-Verlag, 2008, pp. 321–335.
34. I. Dinur and A. Shamir, "Breaking Grain-128 with dynamic cube attacks," in *Fast Software Encryption 2011*, ser. To be published in Lecture Notes in Computer Science, A. Joux, Ed. Springer-Verlag, 2011.

Table 3. Test vectors for Grain-128a.

key	00000000000000000000000000000000	0123456789abcdef123456789abcdef0
iv	0000000000000000 00000000	0123456789abcdef 12345678
pre-output stream	c0207f221660650b6a952ae26586136fa0904140c8621cfe8660c0dec0969e9436f4ace92cf1ebb7	f88720c13f46e6a43c07eeed89161a4dd73bd6b8be8b6b116879714ebb630e0a4c12f0399412982c
accumulator	c0207f22	f88720c1
register	1660650b	3f46e6a4
keystream	787d4917c800a52f948b89b85cee6cfd	61fea132979efb706643f53321c681a6
macstream	8708b25b0498886e288e86666e292d97	63ab164bf5e46195 8dda5920a4c56442
tag($\boldsymbol{m}_0$)	d6401a29	c7c1c655
tag($\boldsymbol{m}_1$)	ece0b535	860aed89
tag($\boldsymbol{m}_2$)	fa80d03e	b94c0b2d
tag($\boldsymbol{m}_3$)	359f67d1	4934b8ad
tag($\boldsymbol{m}_4$)	4c8dcab1	3c3c9320

A Test Vectors

Reflecting the bit-wise nature of Grain-128a, the first bit emitted as keystream is the most significant one.

Among the test vectors are the authentications of five different messages. Message 0, $\boldsymbol{m}_0$, is the message of length 0. Messages 1 and 2 are both of length 1: $\boldsymbol{m}_1 = \boldsymbol{m}_2 + 1 = 0$. These three messages are supposedly helpful in verifying the initialization and basic functioning of the MAC algorithm.

Message 3 is of length 20 and its hexadecimal representation is $\boldsymbol{m}_3 = \texttt{12340}$. Message 4 is 41 bits long and can, using slightly abused notation, be represented as $\boldsymbol{m}_4 = \texttt{123456789e8}$. To avoid any confusion we also give the bit representation of $\boldsymbol{m}_4$: 00010010000110100010101100111100010011110 1.

The test vectors named "macstream" are the sequences shifted into the register, i.e., the pre-output bits $y_{65}, y_{67}, \ldots$.

For the purpose of testing shorter tags, we give the 16-bit tag for $\boldsymbol{m}_4$ authenticated using the all-zero key and all-zero IV as `cab1`.