Analysis of Grain's Initialization Algorithm*

Christophe De Cannière[1,2], Özgül Küçük[1], and Bart Preneel[1]

[1] Katholieke Universiteit Leuven, Dept. ESAT/SCD-COSIC, and IBBT
Kasteelpark Arenberg 10, B–3001 Heverlee, Belgium
{christophe.decanniere,ozgul.kucuk}@esat.kuleuven.be
[2] Département d'Informatique École Normale Supérieure,
45, rue d'Ulm, F-75230 Paris cedex 05

Abstract. In this paper, we analyze the initialization algorithm of Grain, one of the eSTREAM candidates which made it to the third phase of the project. We point out the existence of a sliding property in the initialization algorithm of the Grain family, and show that it can be used to reduce by half the cost of exhaustive key search (currently the most efficient attack on both Grain v1 and Grain-128). In the second part of the paper, we analyze the differential properties of the initialization, and mount several attacks, including a differential attack on Grain v1 which recovers one out of 2^9 keys using two related keys and 2^{55} chosen IV pairs.

1 Introduction

Symmetric encryption algorithms are traditionally categorized into two types of schemes: block ciphers and stream ciphers. Stream ciphers distinguish themselves from block ciphers by the fact that they process plaintext symbols (typically bits) as soon as they arrive by applying a very simple but ever changing invertible transformation. As opposed to block ciphers, stream ciphers do not derive their security from the complexity of the encryption transformation, but from the unpredictable way in which this transformation depends on the position in the plaintext stream.

The most common type of stream ciphers are binary additive stream ciphers. The encryption transformation in this type of ciphers just consists of an exclusive or (XOR) with an independent sequence of bits called key stream. The key stream bits are derived from a secret internal state which is initialized using a secret key, and is then continuously updated.

The security of a binary additive stream cipher depends directly on the unpredictability of its key stream. In particular, the same sequence of key stream

* The work described in this paper has been partly supported by the European Commission under contract IST-2002-507932 (ECRYPT), by the Fund for Scientific Research – Flanders (FWO), the Chaire France Telecom pour la sécurité des réseaux de télécommunications, and the IAP Programme P6/26 BCRYPT of the Belgian State (Belgian Science Policy).

bits should never be reused to encrypt different plaintexts, and hence, a stream cipher should never be reinitialized with the same secret key. However, in order to avoid having to perform an expensive key agreement protocol for every single message, all modern stream ciphers accept during their initialization phase an additional parameter, typically called initialization vector (IV), which allows to generate different key streams from the same secret key.

Although the possibility to reuse the same key for several messages is an indispensable feature in many practical applications, the introduction of initialization vectors in stream ciphers also opens new opportunities for the adversary. Several recent stream cipher proposals [1–4] have succumbed to attacks exploiting relations between key stream bits generated from the same key but different (known or chosen) IVs. This clearly demonstrates the importance of a carefully designed initialization algorithm.

In this paper, we analyze the initialization algorithm of Grain, a family of hardware-oriented stream ciphers submitted to the eSTREAM Stream Cipher Competition. We will first show that a sliding property of the initialization algorithm, which was already noted in [5] but never formally published, not only results in a very efficient related-key attack, but can also be used more generally to reduce the cost of exhaustive key search. We will then study the differential properties of the initialization, and develop a differential attack on Grain v1 which recovers one out of 2^9 keys, and requires two related keys and 2^{55} chosen IV pairs. We will show that similar attacks apply to Grain-128, and that the requirement for related keys can be dropped if we consider reduced-round variants.

We finally note that we do not consider any of the attacks presented in this paper to be a serious threat in practice. However, they certainly expose some non-ideal behavior of the Grain initialization algorithm.

2 Description of Grain

Grain is a family of stream ciphers, proposed by Hell, Johansson, and Meier in 2005 [6], which was designed to be particularly efficient and compact in hardware. Its two members, Grain v1 and Grain-128, accept 80-bit and 128-bit keys respectively. The original version of the cipher, later referred to as Grain v0, was submitted to the eSTREAM project, but contained a serious flaw, as was demonstrated by several researchers [7, 8]. As a response, the initial submission was tweaked and extended to a family of ciphers.

In the next two sections we first describe the building blocks common to all members of the Grain family. Afterwards, we will show how these blocks are instantiated for the specific ciphers Grain v1 and Grain-128

2.1 Keystream Generation

All Grain members consist of three building blocks: an n-bit nonlinear feedback shift register (NFSR), an n-bit linear feedback shift register (LFSR), and a nonlinear filtering function. If we denote the content of the NFSR and the LFSR

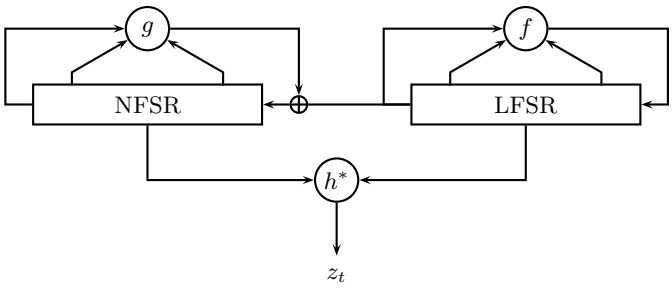


Fig. 1. Grain during the keystream generation phase

at any time t by $B_t = (b_t, b_{t+1}, \ldots, b_{t+n})$ and $S_t = (s_t, s_{t+1}, \ldots, s_{t+n})$, then the keystream generation process is defined as

$$\begin{aligned} s_{t+n} &= f(S_t)\,, \\ b_{t+n} &= g(B_t) + s_t\,, \\ z_t &= h^*(B_t, S_t)\,, \end{aligned}$$

where g and f are the update functions of the NFSR and LFSR respectively, and h^* is the filtering function (see Fig. 1).

2.2 Key and IV Initialization

The initial state of the shift registers is derived from the key and the IV by running an initialization process, which uses the same building blocks as for key stream generation, and will be the main subject of this paper. First, the key and the IV are loaded into the NFSR and LFSR respectively, and the remaining last bits of the LFSR are filled with ones. The cipher is then clocked for as many times as there are state bits. This is done in the same way as before, except that the output of the filtering function is fed back to the shift registers, as shown in Fig. 2 and in the equations below.

$$\begin{aligned} r_t &= h^*(B_t, S_t) + s_t\,, \\ s_{t+n} &= f(r_t, s_{t+1}, \ldots, s_{t+n-1})\,, \\ b_{t+n} &= g(B_t) + r_t\,. \end{aligned}$$

2.3 Grain v1

Grain v1 is an 80-bit stream cipher which accepts 64-bit IVs. The NFSR and the LFSR are both 80 bits long, and therefore, as explained above, the initialization

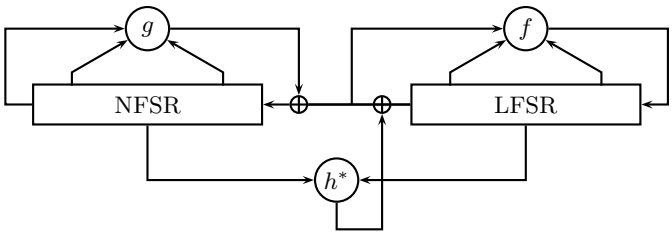


Fig. 2. Grain during the initialization phase

takes 160 cycles. The different functions are instantiated as follows:

$$\begin{aligned} f(S_t) &= s_t + s_{t+13} + s_{t+23} + s_{t+38} + s_{t+51} + s_{t+62}\,, \\ g(B_t) &= b_t + b_{t+14} + b_{t+62} \\ &\quad + g'(b_{t+9}, b_{t+15}, b_{t+21}, b_{t+28}, b_{t+33}, b_{t+37}, b_{t+45}, b_{t+52}, b_{t+60}, b_{t+63})\,, \\ h^*(B_t, S_t) &= \sum_{i \in \mathcal{A}} b_{t+i} + h(s_{t+3}, s_{t+25}, s_{t+46}, s_{t+64}, b_{t+63})\,, \end{aligned}$$

with $\mathcal{A} = \{1, 2, 4, 10, 31, 43, 56\}$, g' a function of degree 6, and h a function of degree 3. The exact definitions of these functions can be found in [6].

2.4 Grain-128

Grain-128 is the 128-bit member of the Grain family. The IV size is increased to 96 bits, and the shift registers are now both 128 bits long. The initialization takes 256 cycles, and the functions are defined as follows:

$$\begin{aligned} f(S_t) &= s_t + s_{t+7} + s_{t+38} + s_{t+70} + s_{t+81} + s_{t+96}\,, \\ g(B_t) &= b_t + b_{t+26} + b_{t+56} + b_{t+91} + b_{t+96} \\ &\quad + b_{t+3}b_{t+67} + b_{t+11}b_{t+13} + b_{t+17}b_{t+18} \\ &\quad + b_{t+27}b_{t+59} + b_{t+40}b_{t+48} + b_{t+61}b_{t+65} + b_{t+68}b_{t+84}, \\ h^*(B_t, S_t) &= \sum_{i \in \mathcal{A}} b_{t+i} + h(s_{t+8}, s_{t+13}, s_{t+20}, s_{t+42}, s_{t+60}, s_{t+79}, s_{t+95}, b_{t+12}, b_{t+95})\,. \end{aligned}$$

In the equations above, $\mathcal{A} = \{2, 15, 36, 45, 64, 73, 89\}$, and h is a very sparse function of degree 3. Again, we refer to the specifications [9] for the exact definition.

3 Slide Attacks

In this section we discuss a first class of attacks on Grain's initialization phase, which are based on a particular sliding property of the algorithm. Slide attacks

have been introduced by Biryukov and Wagner [10] in 1999, and have since then mainly been used to attack block ciphers. A rather unique property of this cryptanalysis technique is that its complexity is not affected by the number of rounds, as long as they are all (close to) identical. This will also be the case in the attacks presented below: the attacks apply regardless of how many initialization steps are performed.

Note that although we will illustrate the attacks using Grain v1, the discussion in the next sections applies to Grain-128 just as well.

3.1 Related (K, IV) Pairs

The sliding property exploited in the next sections is a consequence of the similarity of the operations performed in Grain at any time t, both during initialization and key generation, as well as of the particular way in which the key and IV bits are loaded. More specifically, let us consider a secret key $K = (k_0, \ldots, k_{79})$, used in combination with an initialization vector $\mathrm{IV} = (v_0, \ldots, v_{63})$. During the first 161 cycles (160 initialization steps and 1 key generation step), the registers will contain the following values:

$$\text{init. phase}\left\{\begin{array}{ll} B_0 = (k_0, \ldots\ldots, k_{78}, k_{79}) & S_0 = (v_0, \ldots, v_{62}, v_{63}, 1, \ldots, 1, \quad 1) \\ B_1 = (k_1, \ldots\ldots, k_{79}, b_{80}) & S_1 = (v_1, \ldots, v_{63}, \quad 1, 1, \ldots, 1, s_{80}) \\ \quad\vdots & \quad\vdots \\ B_{160} = (b_{160}, \ldots, b_{238}, b_{239}) & S_{160} = (s_{160}, \ldots\ldots\ldots\ldots, s_{238}, s_{239}) \\ B_{161} = (b_{161}, \ldots, b_{239}, b_{240}) & S_{161} = (s_{161}, \ldots\ldots\ldots\ldots, s_{239}, s_{240}) \end{array}\right.$$

Let us now assume that $s_{80} = 1$. Note that if this is not the case, it suffices to flip v_{13} for the assumption to hold. We then consider a second key $K^* = (k_1, \ldots, k_{79}, b_{80})$ together with the initialization vector $\mathrm{IV}^* = (v_1, \ldots, v_{63}, 1)$. After loading this pair into the registers, we obtain:

$$B_0^* = (k_1, \ldots\ldots, k_{79}, b_{80}) \qquad S_0^* = (v_1, \ldots, v_{63}, 1, 1, \ldots, 1, 1)$$

This, however, is identical to the content of B_1, and since the operations during the initialization are identical as well, the equality $B_t^* = B_{t+1}$ is preserved until step 159, as shown below.

$$\text{init. phase}\left\{\begin{array}{ll} B_0^* = (k_1, \ldots\ldots, k_{79}, b_{80}) & S_0^* = (v_1, \ldots, v_{63}, 1, 1, \ldots, 1, 1) \\ \quad\vdots & \quad\vdots \\ B_{159}^* = (b_{160}, \ldots, b_{238}, b_{239}) & S_{159}^* = (s_{160}, \ldots\ldots\ldots, s_{238}, s_{239}) \\ B_{160}^* = (b_{161}, \ldots, b_{239}, b_{239}^*) & S_{160}^* = (s_{161}, \ldots\ldots\ldots, s_{239}, s_{239}^*) \\ B_{161}^* = (b_{162}, \ldots, b_{239}^*, b_{240}^*) & S_{161}^* = (s_{162}, \ldots\ldots\ldots, s_{239}^*, s_{240}^*) \end{array}\right.$$

In step 160, b^*_{239} and s^*_{239} are not necessarily equal to b_{240} and s_{240}, since the former are computed in initialization mode, whereas the latter are computed in key stream generation mode. Nevertheless, and owing to the tap positions of Grain v1, the equality will still be detectable in the first 15 keystream bits.

Moreover, if $h^*(B^*_{159}, S^*_{159}) = h^*(B_{160}, S_{160}) = 0$ (this happens with probability 1/2), then both modes of Grain are equivalent, and hence the equality is preserved in the last step as well. After this point, (B^*_t, S^*_t) and (B_{t+1}, S_{t+1}) are both updated in key stream generation mode; their values will therefore stay equal till the end, leading to identical but shifted key streams.

With an appropriate choice of IVs, similar sliding behaviors can also be observed by sliding the keys over more bit positions. In general, we have the following property for $1 \leq n \leq 16$:

Property 1. For a fraction $2^{-2 \cdot n}$ of pairs (K, IV), there exists a related pair (K^*, IV^*) which produces an identical but n-bit shifted key stream.

Note that the existence of different (K, IV) pairs which produce identical but shifted key streams is in itself not so uncommon in stream ciphers. When a stream cipher is iterated, its internal state typically follows a huge predefined cycle, and the role of the initialization algorithm is to assign a different starting position for each (K, IV) pair. Obviously, if the total length of the cycle(s) is smaller than the number of possible (K, IV) pairs multiplied by the maximum allowed key stream length, then some overlap between the key stream sequences generated by different (K, IV) pairs is unavoidable. This is the case in many stream ciphers, including Grain. However, what is shown by the property above, is that the initialization algorithm of Grain has the particularity that it tends to cluster different starting positions together, instead of distributing them evenly over the cycle(s).

3.2 A Related-Key Slide Attack

A first straightforward application of the property described in the previous section is a related-key attack. Suppose that the attacker somehow suspects that two (K, IV) pairs are related in the way explained earlier. In that case, he knows that the corresponding key stream sequences will be shifted over one bit with probability 1/4, and if that happens, he can conclude that $s_{80} = 1$. This allows him to derive a simple equation in the secret key bits. Note that if the (K, IV) pairs are shifted over $n > 1$ positions, then with probability $2^{-2 \cdot n}$ the attacker will be able to obtain n equations.

As is the case for all related key attacks, the simple attack just described is admittedly based on a rather strong supposition. In principle, however, one could imagine practical situations where different session keys are derived from a single master key in a funny way, making this sort of related keys more likely to occur, or where the attacker has some means to transform the keys before they are used (for example by causing synchronization errors).

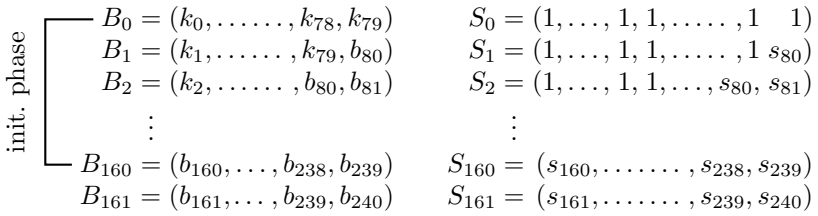
3.3 Speeding up Exhaustive Key Search

A second application, which is definitely of more practical relevance, is to use the sliding property of the initialization algorithm to speed up exhaustive key search by a factor two.

The straightforward way to run an exhaustive search on Grain v1 is described by the pseudo-code below:

for $K = 0$ to $2^{80} - 1$ **do**
  perform 160 initialization steps;
  generate first few key stream bits $(z_0, \ldots, z_t)$;
  check if $(z_0, \ldots, z_t)$ matches given key stream;
end for

Let us now analyze the special case where the given key stream sequence was generated using an IV which equals $I = (1, \ldots, 1)$. In this case, which can easily be enforced if we assume a chosen-IV scenario, the algorithm above can be improved by exploiting the sliding property. In order to see this, it suffices to analyze the contents of the registers during the initialization:



The improvement is based on the observation that if $s_{80} = 1$, we can check two keys without having to recalculate the initialization. If $s_{81} = 1$ as well, then we can simultaneously verify three keys, and so on. In order to use this property to cover the key space in an efficient way, we need to change the order in which the keys are searched, though. This is done in the pseudo-code below:

$K = 0$;
repeat
  perform 160 initialization steps;
  generate first 16 key stream bits $(z_0, \ldots, z_{15})$;
  for $t = 0$ to [largest $n < 16$ for which $S_n = I$] **do**
    check if $(z_t, \ldots, z_{15})$ matches given key stream;
    $K = B_{t+1}$;
  end for
until $K = 0$

Since K is updated in an invertible way, we know that the code will eventually reach $K = 0$ again. At this point, the code will only have checked a cycle of keys with an expected length of 2^{79}. This, however, is done by performing only 2^{78} initializations, making it twice as fast as the standard exhaustive search

algorithm. If we are unlucky, and the secret key is not found, then the algorithm can simply be repeated with a different starting key in order to cover a different cycle.

Finally note that the algorithm retains a number of useful properties of the regular exhaustive search algorithm: it can be used to attack several keys simultaneously, and it can easily be parallelized (using distinguished points to limit overlap). One limitation however, is that it can only be applied if the attacker can get hold of a keystream sequence corresponding to $\mathrm{IV} = (1, \ldots, 1)$, or of a set of keystream sequences corresponding to a number of related IVs. Depending on how the IVs are picked, the latter might be easier to obtain.

3.4 Avoiding the Sliding Property

As discussed earlier, the existence of related (K, IV) pairs in Grain cannot be avoided without increasing the state size. However, in order to avoid the particular sliding behavior of the initialization algorithm, one could try to act on the two factors that lead to this property: the similarity of the computations performed at different times t, and the self-similarity of the constant loaded into the last bits of the LFSR. The similarity of computations could be destroyed by involving a counter in each step. This would effectively increase the size of the state, but one could argue that this counter needs to be stored anyway to decide when the initialization algorithm finishes. An easier modification, however, would be to eliminate the self-similarity of the initialization constant. If the last 16 bits of the LFSR would for example have been initialized with $(0, \ldots, 0, 1)$, then this would already have significantly reduced the probability of the sliding property.

4 Differential Attacks

In this second part, we will analyze the differential properties of the initialization algorithm. We will first show how to generate (truncated) differential characteristics with useful properties, and will then discuss some additional techniques to efficiently use these characteristics in a key recovery attack.

4.1 Truncated Differential Characteristics

The main idea of differential cryptanalysis [11] is to apply differences at the input of a cryptographic function, and to search for non-random properties in the distribution of the corresponding differences at the output. An important tool to find such non-random properties are differential characteristics, which describe possible ways in which differences propagate throughout the internal structure of a function. If the probability that such a characteristic is followed from input to output is sufficiently high, then it will be possible to detect this in the distribution of the output differences.

In the case of stream ciphers, the inputs and outputs that are assumed to be accessible to the adversary are the IV and the keystream sequence respectively. In the attacks described below, we will only consider the difference in a single keystream bit, and ignore all other outputs. In the case of block ciphers, this technique is referred to as truncated differential cryptanalysis.

Finding high probability differential characteristics often boils down to finding sparse characteristics. In the case of Grain such characteristics are relatively easy to find by starting from a difference in a single bit in the state at some step t, and analyzing how this difference propagates, both backwards and forwards. Each time a difference enters the non-linear functions g' or h, we need to make a choice, since there will typically be several possible output differences. A good approach consists in choosing the difference which introduces as few additional differences in the next step as possible, in particular in the NFSR, since bits in this register are more often used as inputs to the non-linear functions than bits in the LFSR. An example of a characteristic found this way is depicted in Fig. 3.

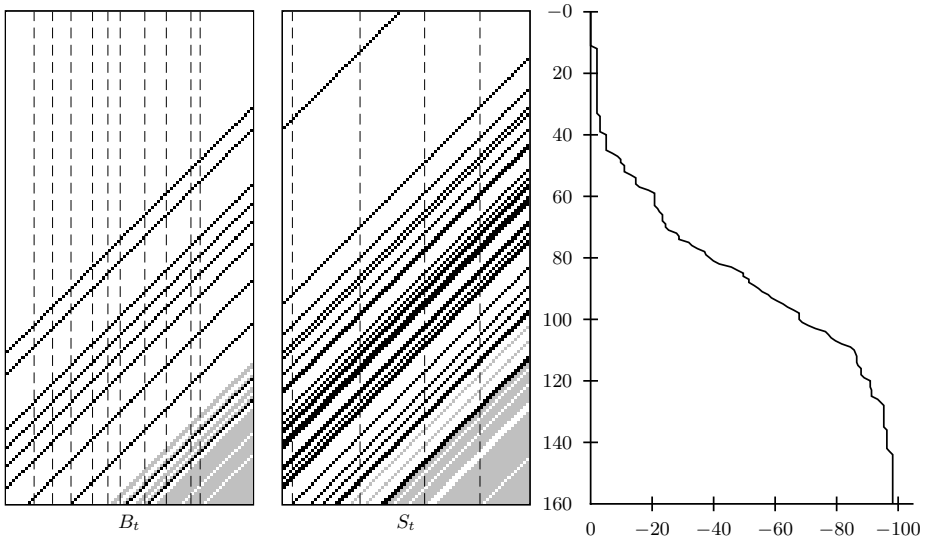


Fig. 3. A truncated differential path in Grain v1 and its probability on a $\log_2$ scale. Differences are denoted by black pixels; bits which do not affect the output are gray. The bit positions which affect either g' or h are marked with dashed lines

The probability of the truncated characteristic in Fig. 3 is almost 2^{-100}. In order to detect a significant bias in the difference of the single output bit which is predicted by this characteristic, we would in principle need to analyze this difference for at least $2^{2 \cdot 100}$ different IV pairs, which can obviously never be achieved with a 64-bit IV. However, if we allow the attacker to apply changes to the key as well (assuming a related-key scenario), then this hypothetical number can be significantly reduced, as shown in Fig. 4. In this case, the number of

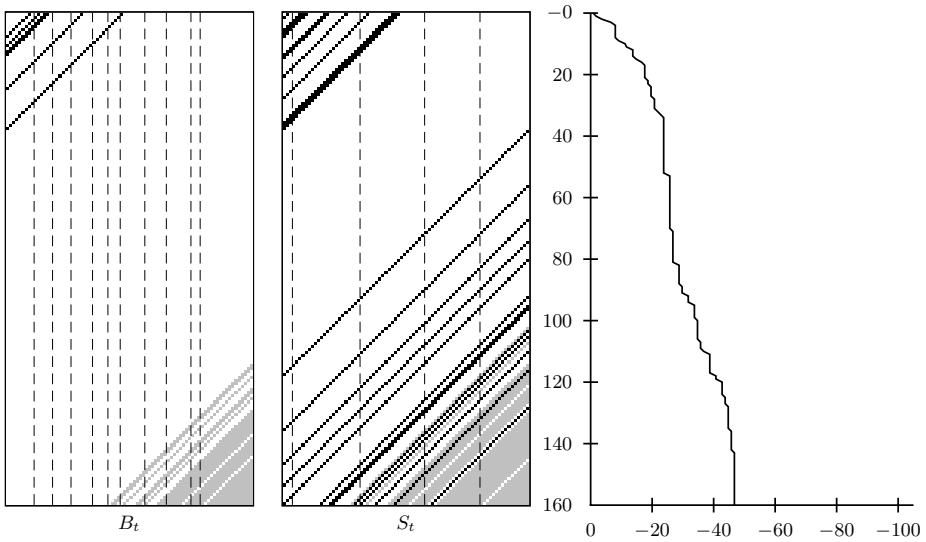


Fig. 4. A related-key truncated differential path in Grain v1 and its probability

required IV pairs is about $2^{2 \cdot 47}$. This is still considerably higher than 2^{63}, the total number of possible IV pairs, but in the next section we will introduce a technique to further reduce this requirement.

4.2 Partitioning the Key and IV Space

In order to the reduce the number of IV pairs needed to detect the bias in the difference of the output bit, we will exploit the fact that the propagation of differences in the first few steps only depends on a rather limited number of key and IV bits (or combinations of them). Hence, if we would guess these key bits, we would be able to get rid of the probabilistic behavior of the first part of the initialization.

Instead of trying to determine exactly which combination of key or IV bits affect the propagation of the differences up to a given step t (which would in fact be relatively easy to do in the case of Grain-128), we will use an alternative technique which allows us to consider the internal operations of the cipher as a black box. To this end, we introduce the function $F_t(K, \mathrm{IV})$ which returns 1 or 0 depending on whether or not the characteristic is followed in the first t steps when the algorithm is initialized with values (K, IV) and (K', IV') satisfying the input difference. The idea now is to partition the key and IV spaces into classes $\{\mathcal{K}_1, \mathcal{K}_2, \ldots\}$ and $\{\mathcal{IV}_1, \mathcal{IV}_2, \ldots\}$ according to the following equivalence relation:

Definition 1. *Two keys K_1 and K_2 are t-equivalent if $F_t(K_1, \mathrm{IV}) = F_t(K_2, \mathrm{IV})$ for all IVs. Similarly, two initialization vectors IV_1 and IV_2 are t-equivalent if $F_t(K, \mathrm{IV}_1) = F_t(K, \mathrm{IV}_2)$ for all keys K.*

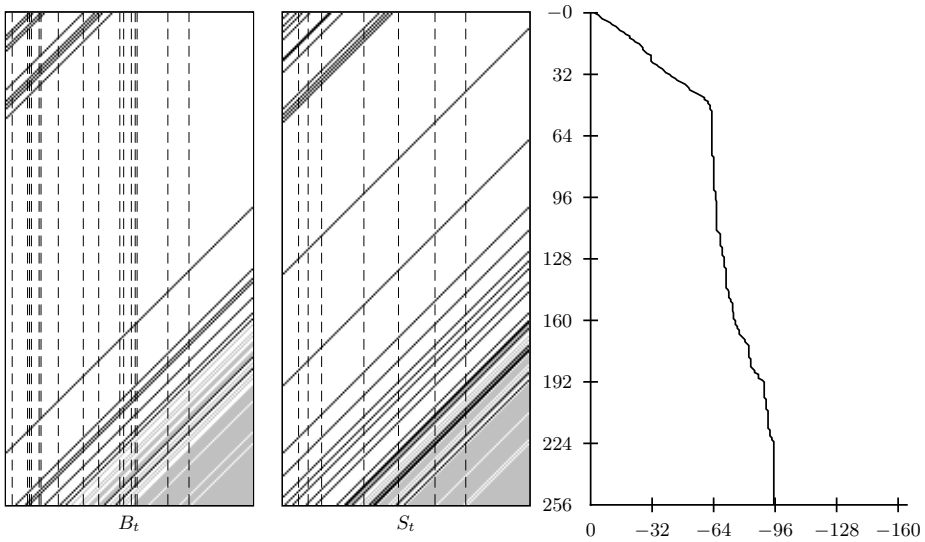


Fig. 5. A related-key truncated differential path in Grain-128 and its probability

In order to check for t-equivalence in practice, we can write F_t as a product $f_1 \cdot f_2 \cdots f_t$, where f_i indicates whether the desired difference at the input of round i propagates to the desired difference at the output of the round. If we observe that $f_i(K_1, \mathrm{IV}) = f_i(K_2, \mathrm{IV})$ for all i and for a sufficient number of random IVs, then we conclude that K_1 and K_2 are most likely t-equivalent.

Before proceeding with the description of the proposed attack, we introduce some additional notation:

p_1:	the total probability of the characteristic in the first t steps.
p_2:	the probability of the characteristic in the remaining steps.
p_K:	the fraction of keys for which $F_t(K, \cdot) \neq 0$ (weak keys).
p_{IV}:	the fraction of IVs for which $F_t(\cdot, \mathrm{IV}) \neq 0$ (weak IVs).
n_K/n_{IV}:	the number of key/IV bits.
$N_{\mathcal{K}}/N_{\mathcal{IV}}$:	the number of weak equivalence classes.

The attack itself consists of two phases:

1. Initialize the stream cipher with a pair of unknown but related keys (K, K') using N different pairs of related weak IVs $(\mathrm{IV}_i, \mathrm{IV'}_i)$. For each IV_i, compute in which class $\mathcal{IV}_i$ it resides, and depending on the difference in the keystream bit, increment the counter $c^0_{\mathcal{IV}_i}$ or $c^1_{\mathcal{IV}_i}$.
2. For all $N_{\mathcal{K}}$ weak key classes $\mathcal{K}_i$, compute the counters $c^0_{\mathcal{K}_i}$ and $c^1_{\mathcal{K}_i}$, with

$$c^0_{\mathcal{K}_i} = \sum_{F_t(\mathcal{K}_i, \mathcal{IV}_j)=1} c^0_{\mathcal{IV}_j} \quad \text{and} \quad c^1_{\mathcal{K}_i} = \sum_{F_t(\mathcal{K}_i, \mathcal{IV}_j)=1} c^1_{\mathcal{IV}_j}\,.$$

Cipher	Grain v1	Grain v1	Grain-128	Grain-128	Grain-128
Rounds	160	112	256	224	192
Related keys	yes	no	yes	no	no
# Weak keys	2^{71}	2^{80}	2^{87}	2^{126}	2^{126}
# Weak IVs	2^{57}	2^{63}	2^{84}	2^{93}	2^{93}
# Chosen IV pairs	2^{55}	(2^{72})	2^{73}	(2^{96})	2^{35}
t	33	28	75	78	76
p_1	2^{-23}	2^{-3}	2^{-64}	2^{-6}	2^{-6}
p_2	2^{-24}	2^{-35}	2^{-31}	2^{-47}	2^{-17}
$N_{\mathcal{K}}$	2^{22}	8	2^{27}	72	72
$N_{\mathcal{IV}}$	2^{21}	8	2^{32}	64	64

Table 1. Summary of the attacks

If N is sufficiently large, and assuming that the unknown key K was indeed weak, we expect the counters above to be biased for the correct key class. In order to get a rough estimate of the minimal value of N, we note that $F_t(\mathcal{K}_i, \mathcal{IV}_j)$ equals 1 with probability $p_1 \cdot p_K^{-1} \cdot p_{\mathrm{IV}}^{-1}$, and hence the expected value of $c^0_{\mathcal{K}_i} + c^1_{\mathcal{K}_i}$ (i.e., the number of IV pairs satisfying the characteristic up to step t, assuming a weak key $K \in \mathcal{K}_i$), is $N \cdot p_1 \cdot p_K^{-1} \cdot p_{\mathrm{IV}}^{-1}$. In order to be able to detect a bias between $c^0_{\mathcal{K}_i}$ and $c^1_{\mathcal{K}_i}$, we need this number to be at least p_2^{-2}, resulting in the following bound:

$$N > \frac{p_K \cdot p_{\mathrm{IV}}}{p_1 \cdot p_2^2} .$$

When we apply this idea to Grain v1, we obtain an attack which can successfully recover one key out of 2^9 and requires at least 2^{55} chosen IVs (see Table 1). In the case of Grain-128, we can recover one key out of 2^{41} using 2^{73} chosen IVs.

4.3 Attacks on Reduced Versions

In Sect. 4.1, we were forced to introduce related keys in order to increase the probability of the differential characteristics in Grain v1 and Grain-128. This is not necessary anymore if we consider reduced-round variants of the ciphers, though. Table 1 summarizes the complexity of a number of regular (not related-key) attacks for different reduced versions.

5 Conclusions

In this paper, we have analyzed the initialization algorithm of Grain taking two very different approaches. First we have studied a sliding property in the initialization algorithm, and shown that it can be used to reduce by half the cost of exhaustive key search. While this might not be significant for Grain-128, it could have some impact on Grain v1, given its relatively short 80-bit key. Moreover, we have shown that this attack could be avoided by making a minor change in the constant used during the initialization.

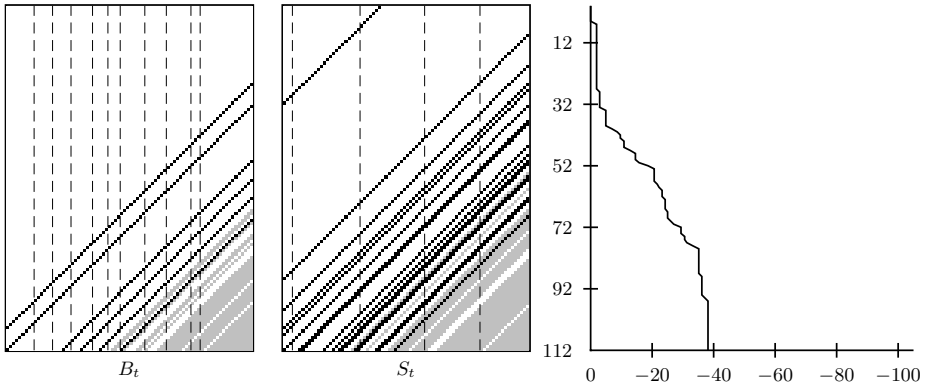


Fig. 6. A truncated differential in Grain v1 reduced to 112 rounds

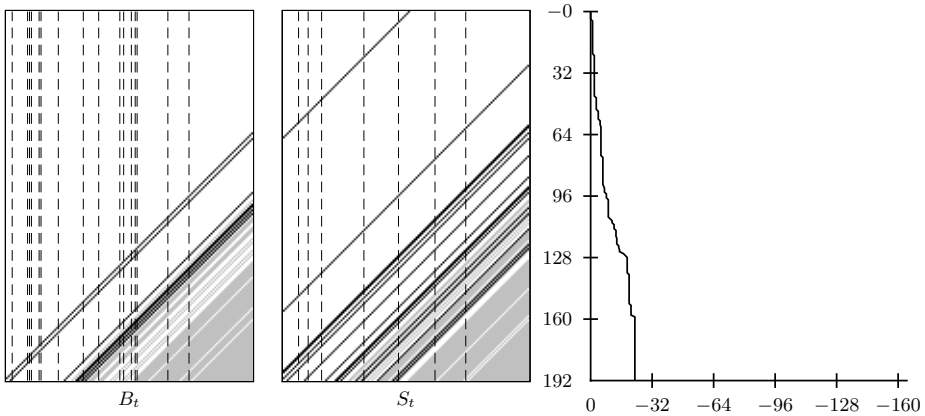


Fig. 7. A truncated differential in Grain-128 reduced to 192 rounds

In the second part of the paper, we have analyzed the differential properties of the initialization. We have constructed truncated differential characteristics for Grain v1, and have shown that by considering a specific partitioning of the key and the IV space, these characteristics can be used to mount a differential attack requiring two related keys and 2^{55} chosen IV pairs, which recovers one key out of 2^9. A similar attack also applies to Grain-128. As is the case for all related-key attacks, the practical impact of this result is debatable, but regardless of this, it can certainly be considered as a non-ideal behavior of the initialization algorithm.

References

1. Wu, H., Preneel, B.: Differential-linear attacks against the stream cipher Phelix. In Biryukov, A., ed.: Fast Software Encryption, FSE 2007. to appear in *Lecture Notes in Computer Science*, Springer-Verlag (2007)

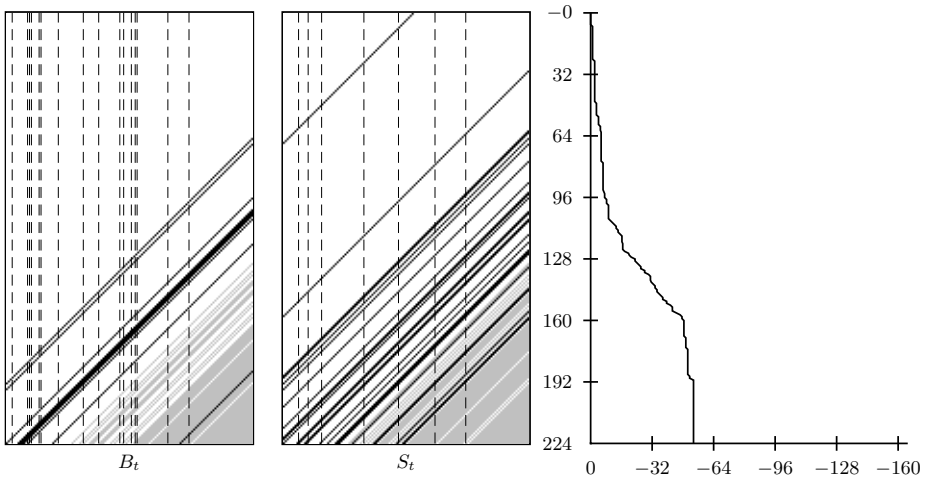


Fig. 8. A truncated differential in Grain-128 reduced to 224 rounds

2. Wu, H., Preneel, B.: Differential cryptanalysis of the stream ciphers Py, Py6 and Pypy. In Naor, M., ed.: Advances in Cryptology – EUROCRYPT 2007. Volume 4515 of Lecture Notes in Computer Science., Springer-Verlag (2007) 276–290
3. Joux, A., Reinhard, J.R.: Overtaking Vest. In Biryukov, A., ed.: Fast Software Encryption, FSE 2007. to appear in *Lecture Notes in Computer Science*, Springer-Verlag (2007) 58–72
4. Wu, H., Preneel, B.: Resynchronization attacks on WG and LEX. In Robshaw, M.J.B., ed.: Fast Software Encryption, FSE 2006. Volume 4047 of Lecture Notes in Computer Science., Springer-Verlag (2006) 422–432
5. Küçük, Ö.: 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`.
6. Hell, M., Johansson, T., Meier, W.: Grain – A Stream Cipher for Constrained Environments. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/010 (2005) `http://www.ecrypt.eu.org/stream`.
7. Berbain, C., Gilbert, H., Maximov, A.: Cryptanalysis of Grain. In Robshaw, M.J.B., ed.: Fast Software Encryption, FSE 2006. Volume 4047 of Lecture Notes in Computer Science., Springer-Verlag (2006) 15–29
8. S.Khazaei, M.H., M.Kiaei: Distinguishing attack on Grain. eSTREAM, ECRYPT Stream Cipher Project, Report 2006/071 (2005) `http://www.ecrypt.eu.org/stream`.
9. Hell, M., Johansson, T., Meier, W.: A Stream Cipher Proposal: Grain-128. eSTREAM, ECRYPT Stream Cipher Project (2006) `http://www.ecrypt.eu.org/stream`.
10. Biryukov, A., Wagner, D.: Slide attacks. In Knudsen, L.R., ed.: Fast Software Encryption, FSE'99. Volume 1636 of Lecture Notes in Computer Science., Springer-Verlag (1999) 245–259
11. Biham, E., Shamir, A.: Differential Cryptanalysis of the Data Encryption Standard. Springer-Verlag (1993)