# Grain-128AEAD, Round 3 Tweak and Motivation

Martin Hell[1], Thomas Johansson[1], Alexander Maximov[2], Willi Meier[3], and Hirotaka Yoshida[4]

[1]Lund University, Sweden
[2]Ericsson AB, Sweden
[3]FHNW, Switzerland
[4]AIST, Japan

**Abstract**

Weaknesses in the Grain-128AEAD key re-introduction, as part of the cipher initialization, are analyzed and discussed. We consider and analyze several possible alternatives for key re-introduction and identify weaknesses, or potential weaknesses, in them. Our results show that it seems favorable to separate the state initialization, the key re-introduction, and the $A/R$ register initialization into three separate phases. Based on this, we propose a new cipher initialization and update the cipher version to Grain-128AEADv2. It can be noted that previously reported and published analysis of the initialization remains valid also for this new version.

## 1 Introduction

Grain-128AEAD is a member of the Grain family of stream ciphers. Compared to previously defined variants, it modifies the cipher initialization such that the key is re-introduced at the end of the initialization. The purpose of this key re-introduction is to not allow the secret key to be immediately reconstructed in case the states of the LFSR and NFSR are known.

Even though any stream cipher would be considered broken if the state can be recovered in less than $2^K$ computations, where $K$ is the keysize, such additional precautions provide some practical security in certain cases since only the current instantiation is broken in case of a state recovery. For a lightweight cipher, it is important that this key re-introduction is very resource efficient.

Since a key-from-state recovery assumes an already broken cipher, it is not crucial that the key reconstruction requires $2^K$ computations, but a too efficient key reconstruction limits the value of this additional precaution.

In [CT21], Chang and Turan noted that with knowledge of the LFSR and NFSR states, a message tag, and the corresponding message, it is possible to reconstruct the secret key with complexity $2^{62}$. This complexity is probably

less than you would expect from a state-of-the-art cipher and it seems that the Grain-128AEAD key re-introduction does not provide much added security. Indeed, any state recovery attack would now only require an additional $2^{62}$ computational steps to reconstruct the key.

In this note, we first briefly outline and discuss the analysis by Chang and Turan. After analyzing the main issue with the key re-introduction, we present and discuss a few different main strategies for protecting against key reconstruction from a known state. In addition to the strategy from [CT21], we also analyze differential biases that could be used to reconstruct the key. Our analysis results in a proposed tweak to the Grain-128AEAD cipher initialization algorithm. We denote the new cipher Grain-128AEADv2 in order to distinguish the two initialization procedures.

## 2 Grain-128AEAD initialization

Similar to all previous versions, Grain-128AEAD uses three main functions together with an LFSR and an NFSR. If authentication is used, which is optional in Grain-128a and mandatory in Grain-128AEAD, ther are also two additional register for supporting this, denoted A and R. A schematic overview of the initializations is given in Figure 1. We will adopt the notation as used in [CT21] for the shift register bits, i.e., let $(B_t, S_t, A_t, R_t)$ be the full state of Grain-128AEAD in time $t$, where,

$$
\begin{aligned}
B_t &= (b_t, \ldots, b_{t+127}) \text{ denotes NFSR state at } t \geq 0,\\
S_t &= (s_t, \ldots, s_{t+127}) \text{ denotes LFSR state at } t \geq 0,\\
A_t &= (a_0^t, \ldots, a_{63}^t) \text{ denotes the Accumulator bits at } t \geq 384,\\
R_t &= (r_0^t, \ldots, r_{63}^t) \text{ denotes the Register bits at } t \geq 384.
\end{aligned}
$$

The functions for updating the LFSR and NFSR are given by

$$
\begin{aligned}
s_{t+128} &= s_t + s_{t+7} + s_{t+38} + s_{t+70} + s_{t+81} + s_{t+96}\\
&= s_t + f'(s_{t+7..t+96}), \qquad (1)\\
b_{t+128} &= s_t + b_t + b_{t+26} + b_{t+56} + b_{t+91} + b_{t+96} + b_{t+3}b_{t+67} + b_{t+11}b_{t+13}\\
&\quad + b_{t+17}b_{t+18} + b_{t+27}b_{t+59} + b_{t+40}b_{t+48} + b_{t+61}b_{t+65} + b_{t+68}b_{t+84}\\
&\quad + b_{t+22}b_{t+24}b_{t+25} + b_{t+70}b_{t+78}b_{t+82} + b_{t+88}b_{t+92}b_{t+93}b_{t+95}\\
&= s_t + b_t + g'(b_{t+3..t+96}), \qquad (2)
\end{aligned}
$$

where the functions $f'()$ and $g'()$ are introduced in order to simplify notation in our analysis in later sections. The output of Grain-128AEAD is given by

$$
\begin{aligned}
y_t &= s_{t+93} + b_{t+2} + b_{t+15} + b_{t+36} + b_{t+45} + b_{t+64} + b_{t+73} + b_{t+89} \qquad (3)\\
&\quad + h(b_{t+12}, s_{t+8}, s_{t+13}, s_{t+20}, b_{t+95}, s_{t+42}, s_{t+60}, s_{t+79}, s_{t+94})\\
&= b_{t+2} + h'(b_{t+12..t+95}, s_{t+8..t+94}), \qquad (4)
\end{aligned}
$$

where, again, $h'()$ is introduced for later convenience. The key and nonce

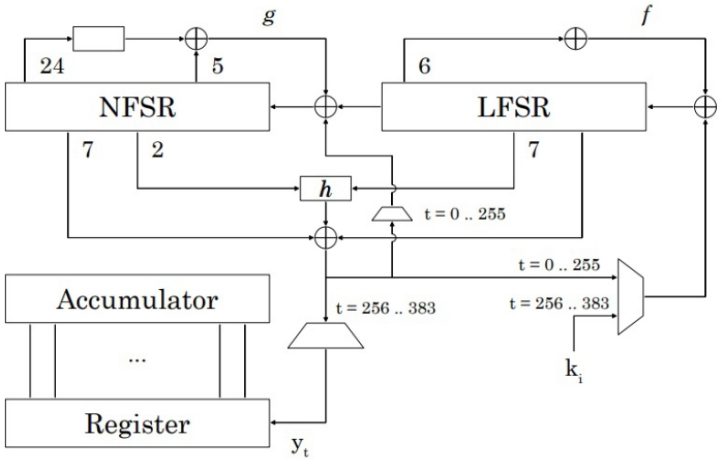


Figure 1: Overview of the initialisation of Grain-128AEAD

(IV) are 128 and 96 bits respectively and we denote them as $k_0, \ldots, k_{127}$ and $IV_0, \ldots, IV_{95}$. To initialize the cipher, let

$$B_0 = (k_0, \ldots, k_{127}), \tag{5}$$
$$S_0 = (IV_0, \ldots, IV_{95}, 1, 1, \ldots, 1, 0). \tag{6}$$

Then, for 256 clocks, the LFSR and NFSR are updated according to

$$s_{t+128} = s_t + f'(s_{t+7..t+96}) + y_t, \quad 0 \leq t \leq 255, \tag{7}$$
$$b_{t+128} = s_t + b_t + g'(b_{t+3..t+96}) + y_t, \quad 0 \leq t \leq 255. \tag{8}$$

Then, in the next 128 clocks, the key is re-introduced into the LFSR while the NFSR is updated as in regular keystream mode,

$$s_{t+128} = s_t + f'(s_{t+7..t+96}) + k_{t-256}, \quad 256 \leq t \leq 383, \tag{9}$$
$$b_{t+128} = s_t + b_t + g'(b_{t+3..t+96}), \quad 256 \leq t \leq 383. \tag{10}$$

In parallel to this key re-introduction, the $A$ and $R$ registers are initialized with the generated $y_t$. At the end of the initialization, we thus have

$$S_{384} = (s_{384}, \ldots, s_{511}),$$
$$B_{384} = (b_{384}, \ldots, s_{511}),$$
$$A_{384} = (y_{256}, \ldots, y_{319}),$$
$$R_{384} = (y_{320}, \ldots, y_{384}).$$

Note that this notation is slightly different from the design document, but consistent with [CT21]. Starting at $t = 384$, the generated $y_t$ is used for encryption and message authentication. The details here are left out as we will only be considering the cipher initialization.

## 3 Reconstructing the key

We outline the key reconstruction approach proposed by Chang and Turan in [CT21]. The LFSR/NFSR can always be clocked backward during the running phase ($t \geq 384$). Thus if the state is recovered at any time $t \geq 384$, it is straightforward to obtain the state $B_{384}$ and $S_{384}$. Thus, we can assume that the attacker has knowledge of $b_t, t \geq 384$ and $s_t, t \geq 384$. However, finding $B_{383}$ and $S_{383}$, which includes $b_{383}$ and $s_{383}$ requires knowing the key bit $k_{127}$, as

$$s_{511} = s_{383} + s_{390} + s_{421} + s_{453} + s_{464} + s_{479} + k_{127}. \tag{11}$$

Thus, we have two unknowns. Combining this with the update for $b_{511}$, we have

$$\begin{aligned} b_{511} =& s_{383} + b_{383} + b_{409} + b_{439} + b_{474} + b_{479} + b_{386}b_{450} + b_{394}b_{396} \\ &+ b_{400}b_{401} + b_{410}b_{442} + b_{423}b_{431} + b_{444}b_{448} + b_{451}b_{467} \\ &+ b_{405}b_{407}b_{408} + b_{453}b_{461}b_{465} + b_{471}b_{475}b_{476}b_{478}. \end{aligned} \tag{12}$$

Adding this equation gives another unknown, $b_{383}$. However, if we assume that also the register $R$ is known at time $t = 384$, i.e., $R_{384} = [y_{320}, \ldots y_{383}]$ is known, then we can add the expression

$$\begin{aligned} y_{381} =& h(b_{393}, s_{389}, s_{394}, s_{401}, b_{476}, s_{423}, s_{441}, s_{460}, s_{475}) + s_{474} + b_{383} \\ &+ b_{396} + b_{417} + b_{426} + b_{445} + b_{454} + b_{470}, \end{aligned} \tag{13}$$

which includes the unknown term $b_{383}$. Thus, we now have 3 equations and 3 unknown variables that are linearly added in this equation. All key bits can now be recovered by continuing to clock backwards. The pre-output bits in the accumulator

$$A_{384} = [y_{256}, y_{257}, \ldots, y_{319}]$$

can be computed as

$$A_{384} = T + \sum_{i=0}^{L-1} m_i \cdot R_{2i+384}, \tag{14}$$

where $m_i, m_{i+1}, \ldots, m_L$ is a known message with the corresponding authentication tag $T$. With both the state ($S_{384}$ and $B_{384}$) and the register contents known, we can easily reconstruct the secret key. Since the keystream does not depend on the registers $A$ and $R$, a state recovery attack is more likely to recover only the LFSR and NFSR states. Then, as $y_{382}$ and $y_{383}$ can be directly determined from $S_{384}$ and $B_{384}$, a key reconstruction requires $2^{62}$ computational steps, i.e., guessing the bits $y_{320}, \ldots, y_{381}$.

## 4 Basic attempts to make the key re-introduction stronger

In this section, we analyze alternative approaches for key re-introduction in parallel with the initialization of A/R registers while maintaining 384 clocks for

initialization. We take a conservative approach and assume that the content of all four registers $(B, S, A, R)$ are known to an adversary at time $t = 384$. As will be shown, virtually any solution in this model fails to protect the key if the state is recovered.

### 4.1 Group 1: push key bits into NFSR instead of LFSR

There are a number of initialization options that fall into the same category, where the derivation of unknown bits is immediately possible without any extra effort.

Let us first consider what happens if we XOR the key bits into the NFSR instead of the LFSR, i.e., the updates given by Eq. (9) and (10) are replaced by

$$s_{t+128} = s_t + f'(s_{t+7..t+96}), \tag{15}$$
$$b_{t+128} = s_t + b_t + g'(b_{t+3..t+96}) + k_{t-256}. \tag{16}$$

Considering the equations for computing the same bits as before ($y_{381}, s_{511}$ and $b_{511}$), we have

$$s_{511} = s_{383} + f'(s_{390..479}), \tag{17}$$
$$b_{511} = s_{383} + b_{383} + g'(b_{386..479}) + k_{127}, \tag{18}$$
$$y_{381} = b_{383} + h'(b_{393..476}, s_{389..475}). \tag{19}$$

As seen, we end up in the same situation with three equations and three unknowns that can easily be solved explicitly.

**Summary:** Any initialization option that leads to a linearly independent system of 3 equations on 3 unknowns is easily broken.

### 4.2 Group 2: push key bits into both NFSR and LFSR

Another approach could be to make the above vulnerable system of 3 equations linearly dependent. There are also several options in this category, but we give just one example where we XOR the key bits into *both* the LFSR and the NFSR, i.e., the update given by Eq. (10) is replaced by

$$b_{t+128} = s_t + b_t + g'(b_{t+3..t+96}) + k_{t-256}. \tag{20}$$

The corresponding expressions for $y_{381}, s_{511}$ and $b_{511}$ are then

$$s_{511} = s_{383} + k_{127} + f'(s_{390..479}), \tag{21}$$
$$b_{511} = s_{383} + b_{383} + k_{127} + g'(b_{386..479}), \tag{22}$$
$$y_{381} = b_{383} + h'(b_{393..476}, s_{389..475}), \tag{23}$$

which is a linearly dependent system of equations. At first glance, this seems to be a better key re-introduction, as it is not possible to determine both $s_{383}$ and $k_{127}$, but only their sum $s_{383} + k_{127}$. However, this does leak information,

and it is possible to guess the value of $s_{383}$ and verify this guess *in a time offset manner* as follows.

Let us first generalize Eq. (21-23),

$$s_{t+128} = (s_t + k_{t-256}) + f'(s_{t+7..t+96}), \tag{24}$$
$$b_{t+128} = (s_t + k_{t-256}) + b_t + g'(b_{t+3..t+96}), \tag{25}$$
$$y_{t-8} = b_{t-6} + b_{t+4}s_t + e'(b_{t+4..t+87}, s_{t+5..t+88}). \tag{26}$$

As for the key reconstruction algorithm, we will run a recursion starting from $t = 383$ and clocking the cipher backward. On each step of the recursion we assume that all values $s_{t+1..}, b_{t+1..}$ are known (or guessed), and we want to recover the three new bits of $s_t, b_t, k_{t-256}$.

From Eq. (24-25) we can derive $b_t$ and $(s_t + k_{t-256})$. Then, we can guess $s_t$ and from that derive $k_{t-256}$. This guess can then be verified just only 6 recursion steps later at time $(t-6)$ by using Eq. (26) since it involves the guessed bit $s_t$, and the newly derived bit $b_{t-6}$, while $y_{t-8}$ here serves as a known value taken from the $A/R$ states and is used as the verification value for the guessing paths along the recursion. Note that the derived $b_t$ is also correct only if all previous guesses of the involved bits were correct.

**Summary:** Re-introduction of the key bits such that the 3 equations become linearly dependent does not help, since the previous guesses may be verified at a later stage of a recursion backtracing algorithm.

### 4.3 Generic recursive backtracing attack

An even simpler and generic backtracing recursion that covers attacks on any tweak from both groups listed above, would be to just guess the $s_t$ in each step $t$, derive $b_t, k_{t-256}$, and simply compute the value of $y_t$ and verify it against the known correct value taken from the $A/R$ state. In this case, we do not even need to go deeper into the structure of the Boolean functions involved, and the recursion will automatically return one step backward once it detects that some previous guess was wrong. The complexity of the key reconstruction can be summarized in Theorem 1.

**Theorem 1** (Backtracing complexity)**.** *For all considered initializations from the groups 1 and 2, one can organize a recursion starting from $t = 383$ and going down to $t = 256$, where the expressions on $b_{t+128}, s_{t+128}$ involve 3 unknowns $s_t, b_t, k_{t-256}$. At every step of recursion, the attacker guesses the value of $s_t$ (or $b_t$) and directly derives the other two unknowns, then clocks the cipher backward by 1 step and decrements the time instance $t$ by 1.*

*In the above recursion, if the guessed value $s_t$ can be verified (against some other equation or a new constraint, e.g., y-values) only after the recursion depth $d$ (i.e., in time $t - d$), then the overall backtracing complexity will be $O(2^d)$.*

**Simulation results.** We implemented the above backtracing recursion algorithm and applied it on two different initialization options from the second group of initializations in subsection 4.2. We were able to reconstruct the whole

128-bit key quite efficiently within some milliseconds, and the time complexity matched well the results of Theorem 1.

**Summary:** The main problem with these approaches is that the key is re-introduced while initializing the registers $A$ and $R$. Thus, it is possible to use the values of $y$ in these registers for verification in the reconstruction algorithm.

### 4.4 The last option: parallel XOR of the whole key at the end of initialization

A straightforward approach, and the one that most closely mimics the FP(1) mode, is to simply XOR all key bits into one of the registers as a final step in the initialization. This is also a tweak that was suggested in [CT21].

First of all, this has a significant drawback of adding to the hardware footprint, since, for 128 register cells, we need to add one XOR gate and one multiplexer, i.e., at least 256 new gates. This makes the hardware footprint much larger and we would still prefer to explore options where the key is serially inserted into one or both registers.

Secondly, this approach seems also vulnerable to the generic backtracing recursion or similar, since $y$-values stored in $A/R$ can again be used for verification at some backward time instances of the recursion against the guessing paths. Moreover, the order of guessing in this scenario can be chosen freely by an attacker.

**Conclusions:** Any tweak where the $A/R$ bits are initialized in parallel or before the key re-introduction can be broken. From this, we conclude that the initialization of the registers $A$ and $R$ must be done *after* the key re-introduction. In the next section, we will consider new approaches for initialization that are more resistant to key reconstruction when the state is known.

## 5 Re-introducing the key before $A/R$ register initialization

Separating key re-introduction and $A/R$ initialization removes the possibility to verify guesses against the $A/R$ content. In this section, we will consider various options for a new tweak in initialization that meet these new requirements.

### 5.1 New generic initialization steps

Considering the above analysis, it becomes clear that the initialization should consist of 3 clearly separated phases. We believe these phases should be as follows:

- $C_n$ standard initialization clocks, like in Grain-128a, where the pre-output

$y_t$ is added to both LFSR and NFSR, i.e., for $t = [0, \ldots, C_n - 1]$:

$$s_{t+128} = s_t + f'(s_{t+7..t+96}) + y_t, \tag{27}$$
$$b_{t+128} = s_t + b_t + g'(b_{t+3..t+96}) + y_t; \tag{28}$$

- $C_k$ clocks where the key is re-introduced (to be defined later);
- $C_m$ clocks where the pre-output $y_t$ is used to initialize the registers $A/R$, while the LFSR and NFSR are updated in the standard keystream mode, i.e., for $t = [C_n + C_k, \ldots, C_n + C_k + C_m - 1]$:

$$s_{t+128} = s_t + f'(s_{t+7..t+96}), \tag{29}$$
$$b_{t+128} = s_t + b_t + g'(b_{t+3..t+96}), \tag{30}$$
$$A/R \leftarrow y_t. \tag{31}$$

In a straightforward approach, one could select $C_k = 128$ to re-introduce the key bits serially in 128 clocks into either (or both) the LFSR and/or NFSR. However, we can compress this stage to $C_k = 64$ clocks by splitting the key into two parts, adding each part to one of the registers,

$$s_{t+128} = s_t + f'(s_{t+7..t+96}) + y_t + k_{t-C_n+64}, \tag{32}$$
$$b_{t+128} = s_t + b_t + g'(b_{t+3..t+96}) + y_t + k_{t-C_n}, \tag{33}$$

for 64 time instances $t = [C_n..C_n{+}63]$. This solution is efficient in both hardware and software. We fix the two initialization parameters to $C_k = 64$ and $C_m = 128$, and it remains to decide $C_n$, i.e., the duration of the initial phase.

## 5.2 An attempt to keep 384 clocks in total, $C_n = 192$

To keep the original 384 initialization clocks, we explore the possibility to re-introduce the key bits in clocks [192..255]. Assume a state recovery attack, where the state $B_{384}, S_{384}$ after initialization is recovered. Since the key is introduced earlier, we can clock backward and recover $B_{256}, S_{256}$. The content of $A/R$ can be recovered immediately since they are initialized at $256 \leq t \leq 383$ using $y_{256}, \ldots, y_{383}$. However, now these bits cannot be used for verification of the guessing paths. The last key bits, $k_{63}$ and $k_{127}$ are introduced through

$$s_{383} = (s_{255} + f'(s_{262..351}) + y_{255} + k_{127}), \tag{34}$$
$$b_{383} = (s_{255} + b_{255} + g'(b_{258..351})) + y_{255} + k_{63}, \tag{35}$$

where $s_{255}, b_{255}, k_{63}, k_{127}$ are unknowns, and other terms can be derived; therefore, the pre-outputs $y_t$ in time $t \geq 256$ are now useless for key reconstruction (unlike the approaches in section 4).

The only remaining possibility for an attacker to verify the guesses, or to recover key bits, it to link somehow the known state $B_{256}, S_{256}$ to the initial state $B_0, S_0$, i.e., to the original $Key, IV$ values.

### 5.2.1 Differential analysis

We now show that if $C_n$ is too small, then the initialization as defined by Eq. (32) and (33) leaks key information through a differential attack. Consider the following sum of variables, denoted by $z$,

$$\begin{aligned} z_{t+128} &= b_{t+128} + s_{t+128} + g'(b_{t+3..t+96}) + f'(s_{t+7..t+96}) \\ &= (s_t + b_t + g'(b_{t+3..t+96}) + y_t + k_{t-192}) + f'(s_{t+7..t+96})) \\ &\quad + (s_t + f'(s_{t+7..t+96}) + y_t + k_{t-128}) \\ &\quad + g'(b_{t+3..t+96}) + f'(s_{t+7..t+96}) \\ &= b_t + k_{t-192} + k_{t-128}. \end{aligned} \tag{36}$$

Since we know $b_t$ and $s_t$ for $t \geq 256$, we can compute $z_{t+128}$ for $t \geq 253$. Thus, we can find a differential

$$\Delta z_{t+128} = \Delta(b_t + k_{t-256} + k_{t-192}) = \Delta b_t, \quad \text{for } t \geq 253. \tag{37}$$

### 5.2.2 A possible scenario for key bit reconstruction with conditional differentials

In a simple scenario we would like to recover some key bit $k_x$, based on the conditional differential distributions $D_0 = (\Delta b_t | k_x = 0)$ and $D_1 = (\Delta b_t | k_x = 1)$, for some $\Delta IV$. If these distributions have different bias, then by collecting many samples $\Delta b_t$, we can determine the key bit $k_x$. This way, we can recover one key bit. The differential should be introduced, ideally, by some $\Delta IV$ while keeping the same $Key$, and we would then collect $r$ pairs of the form $(Key, IV_i)$ and $(Key, IV_i + \Delta IV)$, for $i = 1..r$, some fixed $Key$ and random $IV_i$s. Then, for each pair $\Delta z_{t+128}$ (that is equal to $\Delta b_t$) is computed and the empirical distribution $D$ is constructed. Finally, we compute the distances from $D$ to both $D_0$ and $D_1$, and the shorter distance decides on the key bit value $k_x$.

Of course, the above procedure for recovering $k_x$ requires applying a state recovery attack on $2r$ keystreams, recovering $2r$ states of $(B_{256}, S_{256})$. These states are used for collecting the $z_{t+128}$ samples and the construction of the empirical distribution $D$. However, when, for example, the two key bits $k_{63}, k_{127}$ are recovered, it is possible to clock further backward, and all those recovered states can be used to collect differential samples to recover some other key bit.

One approach could be to collect many samples with a number of differentials $\Delta IV_0, \ldots, \Delta IV_{127}$ - one for each key bit, not necessarily to be used at the same time instance $t$. Then, depending on the time instance $t$ and the target key bit $k_x$, we could derive differential samples from one of the $i$th group of the recovered states. As soon as it becomes possible (i.e., when certain key bits are recovered), the attacker clocks all states backward by one or more clocks, which opens up for applying other $\Delta_i$s and thus to recover other key bits.

### 5.2.3 Differential probabilistic model

In order to study closer conditional and/or differential probabilities, we have adopted the following model for a binary signal $x$, where the signal $x$ is associ-

ated with two probabilities:

$$p_x = Pr\{x = 1\},$$
$$p_{\Delta x} = Pr\{x \oplus x' = 1\},$$

where $x'$ is the same signal but may have a different value, i.e., $\Delta x = x \oplus x'$.

For two *independent* signals $x$ and $y$ we derive expressions for the resulting probabilities of XOR and AND gates:

$$\begin{aligned} p_{x\oplus y} &= p_x + p_y - 2p_x p_y, \\ p_{\Delta(x\oplus y)} &= p_{\Delta x} + p_{\Delta y} - 2p_{\Delta x} p_{\Delta y}, \\ p_{x\&y} &= p_x p_y, \\ p_{\Delta(x\&y)} &= p_{\Delta x} p_{\Delta y}(2(1-p_x)(1-p_y) - 1) + p_x p_{\Delta y} + p_y p_{\Delta x}. \end{aligned}$$

By this, we can configure the initial state of Grain with these signals, where some of the signals will be random values, constants 0 or 1, or differential bits. Then we clock the cipher and derive probabilities for the resulting signals. In the end, we check if some bit of the state or its differential has a detectable bias and if yes then we can try to use it in an attack.

Note that this method is expected, in most cases, to give a *lower bound* for these biases since above we consider $x$ and $y$ as independent. In reality, many of those signals will be dependent. For example, $x = abc$ and $y = bcd$ are dependent as they share $b$ and $c$ signals. Also, if some term $a$ is added in a Boolean expression twice then it should be canceled out, while in this model it will be treated as two different independent signals, thus making the resulting biases smaller than in reality. Therefore, this method is suitable for first searching statistical anomalies, but the actual bias can be verified and refined through, e.g., real simulations.

The state of Grain is thus initialized as follows:

$$\begin{aligned} \text{Constant } 0 &\rightarrow p_0 = 0,\ p_{\Delta 0} = 0, \\ \text{Constant } 1 &\rightarrow p_1 = 1,\ p_{\Delta 1} = 0, \\ \text{Key bits} &\rightarrow p_k = 0.5,\ p_{\Delta k} = 0, \\ \text{Fixed random } IV \text{ bits} &\rightarrow p_{iv} = 0.5,\ p_{\Delta iv} = 0, \\ \text{Differential } \Delta IV \text{ bits} &\rightarrow p_{iv} = 0 \text{ or } 1,\ p_{\Delta iv} = 1. \end{aligned}$$

### 5.2.4 Examples of conditional differentials

We have not managed to find an exact path for the sketched conditional differential attack, but we have found at least some examples where some of the key bits can be recovered by looking into $\Delta z_{255+128}$.

The best approach found is to initialize the cipher with a difference in a single IV-bit. If $\Pr(\Delta z_{383}) = \Pr(\Delta b_{255}) = 1/2 + \varepsilon$ we can distinguish the cipher from random using about $\varepsilon^{-2}$ such samples. By utilizing the differential probabilistic

model, we find that such a difference can be observed with a differential in $IV_{56}$, whenever all other $Key$ and $IV$ bits are random:

$$\begin{aligned} \Delta IV_{56} | Key_{109} = 0 \rightarrow \ \varepsilon_{\Delta b_{255}} = 2^{-7.73}, \\ \Delta IV_{56} | Key_{109} = 1 \rightarrow \ \varepsilon_{\Delta b_{255}} = 0. \end{aligned} \tag{38}$$

The above biases were refined through real simulations by collecting $2^{24}$ samples, resulting in:

$$\begin{aligned} \Delta IV_{56} | Key_{109} = 0 \rightarrow \ \varepsilon_{\Delta b_{255}} = 2^{-4.23}, \\ \Delta IV_{56} | Key_{109} = 1 \rightarrow \ \varepsilon_{\Delta b_{255}} = 0. \end{aligned} \tag{39}$$

We do not know exactly how these distributions will behave for a concrete fixed key, but let us assume that for all or most keys the above is true. Then, recovering $k_{109}$ would proceed as follows. Collect differential samples $\Delta z_{383}$ for $\Delta IV_{56}$, i.e., the cipher is initialized with many random IV pairs where we flip only the bit $IV_{56}$. Then, based on the empirical distribution of $\Delta z_{383}$ we can determine the value of $k_{109}$. This will require around $2 \cdot 2^{8.46} = 2^{9.46}$ initializations, each followed by a state recovery attack. Recovering all key bits in this way requires finding biases similar to Eq. (39) (where we can also utilize any other $t \geq 253$). While this might not be feasible for all key bits, recovering some key bits in this way could be followed by an exhaustive search for the rest. We stress that this requires a state recovery for each initialization and is thus always more expensive than a state recovery attack. Still, the relatively large biases found here question the suitability of re-introducing the key in this way as early as $t = 192$ since a state recovery attack gives information about the state already at $t = 256$.

**Conclusion:** In the presented example we only show that such a conditional differential exists to some extent. This approach for key reconstruction can be investigated further, and we leave it for future research. What is clear is that for $C_n = 192$ there is at least *some* leakage of information about the key, although the exact attack scenario is not easy to find.

#### 5.2.5 Other differentials detected

The differential probabilistic model was also used to detect some other biases. For example, we were fixing the key to a random state and were running the model to see how far we can get a bias with that model. We found that in $\sim 9.4\%$ of random keys, where we also set $IV_{48..96} = 0$ while $IV_{0..47}$ are random, we get for $\Delta IV_{69}$ the differential $\Delta b_{288}$ to have the bias around $2^{-10.9..-13.9}$.

Through real simulations, we collected $2^{30}$ samples for each random key, and got the refined bias in the range $2^{-10..-12}$, but with a higher success rate of $\sim 12.5\%$ for 2600 random keys tested. For these simulations we used a PRNG with high entropy. Simulation results are given in Figure 2.

I.e., for about one key out of 8 random keys, we can distinguish $\Delta z_{288+128}$ from random by collecting around $2^{20..24}$ IV-differential samples. However, the result of the distinguisher may leak up to 5 bits of information about the key

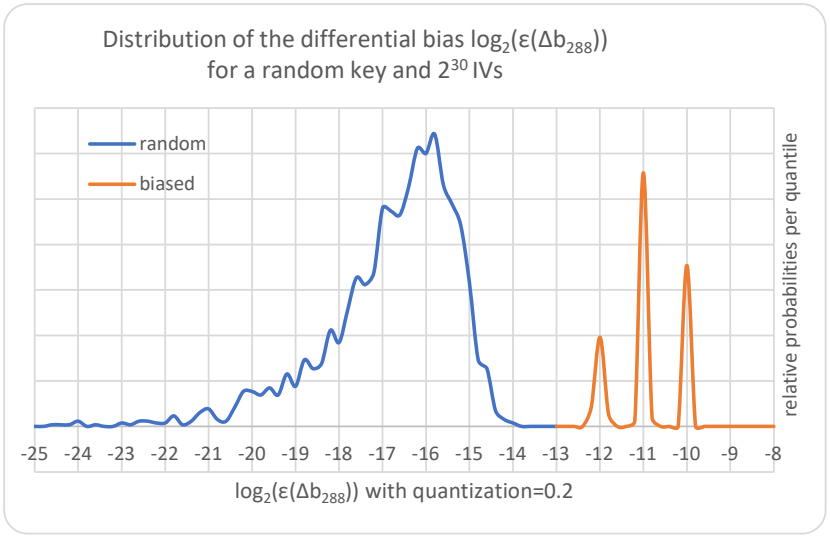


Figure 2: Refining simulation results for the bias of $\Delta b_{288}$.

in connection to 4 possible answers: random, or one of the 3 biased peaks. To be more precise, reverse-engineering of the random keys led to the four different answers resulting in the following key information:

$$\begin{aligned}
&\textbf{if } \Delta b_{288} \text{ is biased } &&\textbf{then } k_{73} = k_{122} = 0, k_{109} = 1,\\
&\textbf{if } \varepsilon(\Delta b_{288}) \approx 2^{-10} &&\textbf{then } k_{77} = k_{112} = 0,\\
&\textbf{if } \varepsilon(\Delta b_{288}) \approx 2^{-11} &&\textbf{then } k_{77} + k_{112} = 1,\\
&\textbf{if } \varepsilon(\Delta b_{288}) \approx 2^{-12} &&\textbf{then } k_{77} = k_{112} = 1.
\end{aligned}$$

## 5.3 What would be the minimum $C_n$?

### 5.3.1 From differential attacks perspectives

Recall the previously derived expression for $\Delta z_{t+128}$,

$$\Delta z_{t+128} = \Delta(b_{t+128} + s_{t+128} + g'(b_{t+3..t+96}) + f'(s_{t+7..t+96})) = \Delta b_t,$$

and the first (smallest) $t$ for which the above differential is available, right after the set of state recovery attacks, is $t = C_n + C_k - 3$ (see Eq. (36)), where $C_k = 64$ but $C_n$ is not yet known. Thus, if there is a weakness in $\Delta b_t$ then the lower bound for $C_n$ to mitigate such a weakness would be

$$C_n \geq t - 60.$$

In [MTQ17], the authors managed to recover 18 key bits after 169 clocks of initialization, but there they were looking at $y_t$ values instead of the state bits.

In a naïve approximation, this would translate to having some (differential) bias in at most $s_{169+94}, b_{169+95}$ state bits, where $169 + 95 = 264$ is the highest index of the state bits involved in $y_{169}$. Thinking pure theoretically one could, perhaps, collect some statistics on up to $\Delta b_{264}$. This leads to the first lower bound $C_n \geq 204$. Clearly, $C_n = 192$ looks too low – the case we first considered in subsection 5.2 in order to keep 384 clocks of initialization.

However, in our simulations, we were able to find highly biased differentials up to around $\Delta b_{288}$ (see subsubsection 5.2.5), which means that $C_n$ must be at least $C_n \geq 228$ in order to prevent these differentials as well.

Finally, we would like to note that the mentioned paper [MTQ17] also provides a distinguishing attack after 195 initializations rounds, by again looking into $y_t$. Note that in the case of a state recovery attack a distinguishing attack is not relevant at all – if the attacker can recover the state of a given keystream then, certainly, we have a distinguisher. What only matters for this type of analysis is the possibility to reconstruct at least some key bits given available expressions and values. Nevertheless, if we could anyways get some biased conditional distribution on $\Delta b_{195+95}$ then that assumption translates into the hardest lower bound $C_n \geq 230$.

Although we make here a very strong assumption that some key bits can still be recovered by looking at $\Delta b_{290}$, there is great uncertainty about how large the bias would be, and therefore how many samples one has to collect. The complexity of such an attack is then at least the number of samples needed multiplied by the complexity of a single state recovery attack. There is also uncertainty about how many and which key bits can be statistically detected, and whether this leads to any additional backward clocks of the collected states. The order of determining key bits is important for the backtracing ability.

**Summary:** Given the current state-of-the-art analysis of Grain, and making the most strict (and, perhaps, unrealistic) assumptions, we conclude that we have a lower bound $C_n \geq 230$. This means that we cannot really stay with 384 clocks for the initialization (starting to re-introduce the key at $t = 192$), but should increase it by at least 38 (plus some added margin), even if these strong attack models are hard to achieve.

#### 5.3.2 Combining state recovery, guess-and-determine, and distinguishing attacks

Let us again consider the distinguishing attack in [MTQ17] using $y$-terms after 195 initialization clocks. This means that there could also exist a distinguisher on $\Delta b_{195+95}$, since $y_{195}$ involves $b_{195+95}$. So, if $C_n = 229$ we can sample $\Delta z_{418} = \Delta b_{290}$ and distinguish that from random. Though a pure distinguishing attack itself is not interesting in key reconstruction, such a distinguisher can be used for verifying guessed key bits.

Assume that, based on the previous discussion, we adopt $C_n = 256$. Then, we can collect $r$ pairs of keystreams, each pair with one keystream generated with some random $IV$, and one using a differential $IV + \Delta IV$ for some fixed $\Delta IV$. Then we apply a state recovery attack on each of the $2r$ keystreams,

recovering $r$ pairs of states $(B_{320}^{(i)}, S_{320}^{(i)})$ and $(B_{320}^{\prime(i)}, S_{320}^{\prime(i)})$, for $i = 1, \ldots, r$.

Then, by guessing the first 54 key bits, we can reverse 27 initialization steps, recovering $(B_{293}^{(i)}, S_{293}^{(i)})$ and $(B_{293}^{\prime(i)}, S_{293}^{\prime(i)})$. Thus, we reach $C_n = 229$. At this point, for each pair of states in time $t = 293$, we compute $r$ samples $\Delta z_{290+128}^{(i)}$, i.e., $\Delta b_{290}^{(i)}$, see Eq. (37). With many states, we can use the empirical distribution for $\Delta b_{290}$ and distinguish it from random. If the 54 key bits were correctly guessed, we get a biased empirical distribution, otherwise, the guess was incorrect. Note that, for every key guess we can use the same set of the recovered states for sampling.

**Example.** Let us give an example of the complexity of this attack. Assume there is a $\Delta IV$ that makes $\Delta b_{290}$ being biased with $\varepsilon = 2^{-10}$. With $2^{54}$ distributions, to distinguish the correct key guess, we need about $(54 \cdot 2 \ln 2) \cdot 2^{20} = 2^{26.2}$ samples, i.e., $2^{27.2}$ keystreams are needed (see e.g., [HJB09] for a derivation of this expression). If the state recovery attack has complexity $2^e$ then the total attack complexity would be $O(2^e \cdot 2^{27.2} + 2^{54} \cdot 2^{27.2}) = O(2^{e+27.2} + 2^{81.2})$. Note also, if the bias of $\Delta b_{290}$ would be too small, the attacker can guess a few more key bits and reach a lower index of $\Delta b_t$ where the bias is larger. For example, guessing 2 more key bits makes it possible to backtrace one more step.

**Conclusion:** At this point we do not have further details, the sketched attack is purely theoretical at the moment, and we leave further investigations for future research. However, to mitigate such kinds of more advanced attacks, the lower bound on $C_n$ should be increased by 64, i.e., $C_n \geq 294$. In this case, all key bits need to be guessed in order to reach the time instance where one can collect biased samples and use them for verification. And, of course, future research might reveal biases in $\Delta b_t$ with $t > 290$. For these reasons, $C_n = 256$ also looks too low in the context of key reconstruction from known states.

# 6 Proposed key re-introduction

From the analysis in the previous sections, we can conclude:

1. The key should not be re-introduced while also initializing the $A$ and $R$ register, or in parallel after;

2. Introducing the key as early as $t = 256$ is questionable due to rather large biases found in some differentials.

Based on our analysis done in the previous sections, we propose to increase the initialization by 128 extra clocks and to adopt $C_n = 320, C_k = 64, C_m = 128$ in the generic description given in subsection 5.1. We believe this tweak makes the key re-introduction secure with high confidence, and denote the cipher using this new initialization Grain-128AEADv2. Finally, we would like to highlight a few positive points:

- Both Grain-128a and Grain-182AEAD have 256 clocks of initialization, and no attack was found on that so far. In the proposed tweak we do

$C_n = 320$ initialization clocks, in order to protect the key re-introduction phase in case the whole state is recovered from the keystream. After that we re-introduce the key, followed by the $A/R$ initialization. I.e., in the proposed design we have security not worse than in the previous instances of Grain, which in turn were analyzed for many years;

- We use only 64 clocks for the key re-introduction phase, which is a compromise between a parallel XOR of the key (that is more expensive in hardware) and the introduction of the key bits in 128 clocks (that is more expensive in time). Moreover, we believe that serialized key re-introduction is more secure than the parallel, since then an attacker has much less freedom to exploit available Boolean expressions;
- The attacks on the key re-introduction, similar to the one in [CT21] are no longer possible. The three initialization phases are now clearly separated;
- With the proposed tweak we prevent key reconstruction from known states also with more advanced and comprehensive types of attacks, though some of them currently are theoretical and speculative, and also under strong assumptions.

# 7 Conclusions

The weakness discovered in the key re-introduction of Grain-128AEAD shows that the key can be reconstructed with low complexity if the state is known. We analyze the key re-introduction further by considering several different possible approaches, and we also analyze these approaches, both in relation to a previously published reconstruction technique, but also by considering more sophisticated methods. As a result, we present an update to the cipher initialization that is both more secure, but also maintains the validity of previous analysis of the initialization algorithm. The new cipher version is denoted Grain-128AEADv2.

# References


[CT21] Donghoon Chang and Meltem Sonmez Turan. Recovering the key from the internal state of grain-128aead. Cryptology ePrint Archive, Report 2021/439, 2021. https://eprint.iacr.org/2021/439.

[HJB09] Martin Hell, Thomas Johansson, and Lennart Brynielsson. An overview of distinguishing attacks on stream ciphers. *Cryptography and Communications*, 1(1):71–94, 2009.

[MTQ17] Zhen Ma, Tian Tian, and Wen-Feng Qi. Conditional differential attacks on grain-128a stream cipher. *IET Information Security*, 11(3):139–145, 2017.