Some Results on Related Key-IV Pairs of Espresso

George Teşeleanu[1,2]

[1] Advanced Technologies Institute
10 Dinu Vintilă, Bucharest, Romania
tgeorge@dcti.ro
[2] Simion Stoilow Institute of Mathematics of the Romanian Academy
21 Calea Grivitei, Bucharest, Romania

Abstract. In this paper, we analyze the Espresso cipher from a related key chosen IV perspective. More precisely, we explain how one can obtain Key-IV pairs such that Espresso's keystreams either have certain identical bits or are shifted versions of each other. For the first case, we show how to obtain such pairs after 2^{32} iterations, while for the second case, we present an algorithm that produces such pairs in 2^{28} iterations. Moreover, we show that by making a minor change in the padding used during the initialization phase, it can lead to a more secure version of the cipher. Specifically, changing the padding increases the complexity of our second attack from 2^{28} to 2^{34}. Finally, we show how related IVs can accelerate brute force attacks, resulting in a faster key recovery. Although our work does not have any immediate implications for breaking the Espresso cipher, these observations are relevant in the related-key chosen IV scenario.

1 Introduction

With the growth of Internet of Things (IoT) applications, lightweight ciphers are becoming highly demanded in the IoT industry. Lightweight ciphers are required to offer users a high level of assurance, while running in resource-constrained devices. Additionally, with the rise of 5G networks, traffic volume is estimated to increase by 1000 times [11]. Hence, besides being implemented in IoT devices that usually have limited computing power and strict power constraints, lightweight ciphers should also offer low propagation delays in implementation.

Since previously cipher designs focused either on hardware size or speed, a new class of lightweight ciphers had to be introduced. Such a class was introduced in [8] and was designed to be a trade-off between hardware size and speed for a given security level. The basic idea of this new design is to combine the short propagation delays of the Galois Non-Linear Feedback Shift Registers (NFSRs) with the advantage of Fibonacci NFSRs, which are more easily analyzed from a security point of view. More precisely, the authors of [8] employ a NFSR in Gallois configuration and carry out their security analysis on a transformed NFSR which resembles a Fibonacci NFSR. They also provide a concrete construction, called Espresso, that is a representative of their design.

The only independent security analyses that we are aware of can be found in [12,13]. In [12], the authors propose a related key chosen IV attack on a variant of Espresso, denoted Espresso-a. Similar to [8], they transform the Galois NFSR to a Fibonacci one, however the output function is the same as that of Espresso. The authors of [13], state that the transformed NFSR studied in [8,12] are not equivalent to the original Galois NFSR, unless the output function is changed accordingly. Hence, the security analyses are not conducted on the actual cipher. To support their claim, the authors introduce a novel transformation that converts Espresso-like ciphers into LFSR filter generators. Then they provide several algebraic and fast correlation attacks that can be applied to the resulting filter generators. In light of their results, they also urge researchers to reassess Espresso's resistance against chosen IV attacks, differential attacks and weak key attacks.

Compared to previous approaches, instead of studying the equivalent Fibonacci NFSR, we propose three related key chosen IV attacks by working directly with the Galois NFSR. We will first study the differential properties of the initialization algorithm and we will show how to construct related Key-IV pairs that produce identical bits on certain positions. Our methods are influenced by the differential attacks, previously published in [4,10], designed against the Grain family. Secondly, we show a sliding property of the initialization algorithm that allows an attacker to construct related Key-IV pairs that generate shifted keystreams. Again, we were influenced by the sliding attacks devised against the Grain family (presented in [4–6,9,10]). To increase the complexity of our proposed slide attacks, we suggest a slight change to Espresso's padding. Thirdly, we propose a guess and determine attack that takes as input two or four related IV's and outputs the secret key. A similar approach[3] can be found for Grain-128a in [7] and Espresso-a in [12]. 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 Espresso initialization algorithm.

Structure of the Paper. We introduce notations and preliminaries in Section 2. In Section 3 we present differential attacks, in Section 4 we propose several constructions for generating related Key-IV pairs and in Section 5 we suggest several key recovery algorithms. We conclude in Section 6.

2 Preliminaries

Notations. Throughout the paper, the notation $\|$ denotes string concatenation, $\oplus$ denotes bitwise XOR and $|$ denotes bitwise OR. The $x \ggg i$ operator causes the bits in x to be rotated to the right by i positions. The subset $\{0, \ldots, s\} \in \mathbb{N}$ is denoted by $[0, s]$. The action of selecting a random element x from a sample space X is represented by $x \in_R X$. Hexadecimal strings are marked by the prefix `0x`. We define $MID_{[\ell_1,\ell_2]}(Q) = q_{\ell_1}\| \ldots \|q_{\ell_2}$ and $LSB_{\ell_1}(Q) = MID_{0,\ell_1}(Q)$, where $Q = q_0\| \ldots \|q_{\ell_1}\| \ldots \|q_{\ell_2}\| \ldots \|q_\ell$.

2.1 Description of Espresso

We further provide the specifications of Espresso as presented in [8]. One of the main building blocks of Espresso is a 256-bit NFSR in the Galois configuration. Let $X_i = [x_i, x_{i+1}, \ldots, x_{i+255}]$ denote the state of the NFSR at time i and let $g_j(X_i)$, where $j \in [0, 255]$, be the feedback functions of the NFSR. The nonlinear feedback functions are defined as follows

$$\begin{aligned}
g_{255}(X_i) &= x_i \oplus x_{i+41}x_{i+70}\\
g_{251}(X_i) &= x_{i+252} \oplus x_{i+42}x_{i+83} \oplus x_{i+8}\\
g_{247}(X_i) &= x_{i+248} \oplus x_{i+44}x_{i+102} \oplus x_{i+40}\\
g_{243}(X_i) &= x_{i+244} \oplus x_{i+43}x_{i+118} \oplus x_{i+103}\\
g_{239}(X_i) &= x_{i+240} \oplus x_{i+46}x_{i+141} \oplus x_{i+117}\\
g_{235}(X_i) &= x_{i+236} \oplus x_{i+67}x_{i+90}x_{i+110}x_{i+137}\\
g_{231}(X_i) &= x_{i+232} \oplus x_{i+50}x_{i+159} \oplus x_{i+189}\\
g_{217}(X_i) &= x_{i+218} \oplus x_{i+3}x_{i+32}\\
g_{213}(X_i) &= x_{i+214} \oplus x_{i+4}x_{i+45}\\
g_{209}(X_i) &= x_{i+210} \oplus x_{i+6}x_{i+64}\\
g_{205}(X_i) &= x_{i+206} \oplus x_{i+5}x_{i+80}\\
g_{201}(X_i) &= x_{i+202} \oplus x_{i+8}x_{i+103}\\
g_{197}(X_i) &= x_{i+198} \oplus x_{i+29}x_{i+52}x_{i+72}x_{i+99}\\
g_{193}(X_i) &= x_{i+194} \oplus x_{i+12}x_{i+121}
\end{aligned}$$

The remaining feedback functions are of type $g_j(X_i) = x_{i+j+1}$.

[3] both using only two related IV's

Another building block of the Espresso cipher is a non-linear output function $z(X_i)$ given by

$$\begin{aligned} z(X_i) =& x_{i+80} \oplus x_{i+99} \oplus x_{i+137} \oplus x_{i+227} \oplus x_{i+222} \oplus x_{i+187} \oplus x_{i+243}x_{i+217} \oplus x_{i+247}x_{i+231} \\ & \oplus x_{i+213}x_{i+235} \oplus x_{i+255}x_{i+251} \oplus x_{i+181}x_{i+239} \oplus x_{i+174}x_{i+44} \oplus x_{i+164}x_{i+29} \\ & \oplus x_{i+255}x_{i+247}x_{i+243}x_{i+213}x_{i+181}x_{i+174} \end{aligned}$$

We further describe the main algorithms used by the Espresso cipher in the initialization and keystream generation phases.

Key Loading Algorithm (KLA). Espresso uses a 128-bit key K, a 96-bit initialization vector IV and a fixed 32-bit padding $P = \texttt{0xfffffffe}$. The key is loaded in the NFSR as follows: $X_0 = K \| IV \| P$.

Key Scheduling Algorithm (KSA). After running KLA, the output[4] $z_i = z(X_i)$ is XOR-ed to $g_{255}(X_i)$ and $g_{217}(X_i)$ update functions, *i.e.*, during one clock the update functions are updated as $g_{255}(X_i) = x_i \oplus x_{i+41}x_{i+70} \oplus z_i$ and $g_{217}(X_i) = x_{i+218} \oplus x_{i+3}x_{i+32} \oplus z_i$.

Pipeline Key Scheduling Algorithm (PKSA). Due to the pipelining of the output function some extra clocks are needed before producing the keystream. Hence, the PKSA algorithm instead of outputting[4] z_i simply ignores it. Note that after each generated bit the NFSR's internal state is updated using the KSA routine with $g_{255}(X_i) = x_i \oplus x_{i+41}x_{i+70}$ and $g_{217}(X_i) = x_{i+218} \oplus x_{i+3}x_{i+32}$.

Pseudorandom Keystream Generation Algorithm (PRGA). After performing the KSA routine for 256 clocks and the PKSA routine for 3 clocks, bit z_i is used as the output keystream bit. After each generated bit the NFSR's internal state is updated as in the PKSA routine.

2.2 Security Model

In this paper, we will work in the *Related Key Chosen IV* security model. In this model, according to [5, Section 2.1], the adversary $\mathcal{A}$ is given access to an encryption oracle $\mathcal{O}$ that has access to the key K. Therefore, $\mathcal{A}$ can query $\mathcal{O}$ and thus obtain valid ciphertexts.

More precisely, for each query i, the adversary first chooses the oracle's parameters: an initialization vector IV_i, a function $\mathcal{F}_i : \{0,1\}^n \to \{0,1\}^n$ and a message m_i. Then $\mathcal{O}$ encrypts m_i using the Key-IV pair $(\mathcal{F}_i(K), IV_i)$. After repeating this process several times, the adversary's task is to distinguish the keystream output from a random stream or to compute the secret key efficiently.

3 Related Key-IV Pairs

Our first goal is to construct a family of related Key-IV functions such that the adversary can distinguish the resulting keystreams from random ones with high probability. An important step to construct such pairs is the observation that the KSA and PKSA routines are invertible. More precisely, if a state X_i is obtained by applying either KSA or PKSA to X_{i-1}, we can recover X_{i-1} from X_i by rolling back one clock. We further refer to the transition functions from X_i to X_{i-1} as KSA^{-1} and PKSA^{-1}. The exact details of KSA^{-1} and PKSA^{-1} are given in Algorithms 1 and 2.

We further denote by KSA_{256} and KSA^{-1}_{256} the KSA and KSA^{-1} routines performed for 256 clocks. Similarly, we define PKSA_3 and PKSA^{-1}_3. We also define $\text{KLA}^{-1}(X) = (LSB_{127}(X), MID_{[128,223]}(X))$ and $\Delta(X) = X \oplus \delta$, where $\delta \in \{0,1\}^{256}$. Using these routines we can obtain a pair of related Key-IVs (K, IV) and $(K, IV)_\Delta$ such that they produce almost similar initial keystreams. A high level description of the construction is provided in Figure 1.

We further present an algorithm that checks which keystream positions produced by the states X_0 and $X_{0,\Delta}$ are identical. Before stating our result, we first introduce a small modification to the keystream generation algorithm. Note that this modification is only used as part of Algorithm 3 and is needed to aid us find identical positions. We also make an assumption about Espresso's keystream bits.

[4] during one clock

Algorithm 1: PKSA^{-1} routine for Espresso

Input: State $X_i = (x_0, \ldots, x_{255})$
Output: The preceding state $X_{i-1} = (y_0, \ldots, y_{255})$

for $t = 0$ to 254 **do**
  $y_{t+1} = x_t$
$y_0 = x_{255} \oplus x_{40}x_{69}$
$y_{252} = x_{251} \oplus x_{41}x_{82} \oplus x_7$
$y_{248} = x_{247} \oplus x_{43}x_{101} \oplus x_{39}$
$y_{244} = x_{243} \oplus x_{42}x_{117} \oplus x_{102}$
$y_{240} = x_{239} \oplus x_{45}x_{140} \oplus x_{116}$
$y_{236} = x_{235} \oplus x_{66}x_{89}x_{109}x_{136}$
$y_{232} = x_{231} \oplus x_{49}x_{158} \oplus x_{188}$
$y_{218} = x_{217} \oplus x_2x_{31}$
$y_{214} = x_{213} \oplus x_3x_{44}$
$y_{210} = x_{209} \oplus x_5x_{63}$
$y_{206} = x_{205} \oplus x_4x_{79}$
$y_{202} = x_{201} \oplus x_7x_{102}$
$y_{198} = x_{197} \oplus x_{28}x_{51}x_{71}x_{98}$
$y_{194} = x_{193} \oplus x_{11}x_{120}$

Algorithm 2: KSA^{-1} routine for Espresso

Input: State $X_i = (x_0, \ldots, x_{255})$
Output: The preceding state $X_{i-1} = (y_0, \ldots, y_{255})$

$X_{i-1} = \mathrm{PKSA}^{-1}(X_i)$
$z = y_{80} \oplus y_{99} \oplus y_{137} \oplus y_{227} \oplus y_{222} \oplus y_{187} \oplus y_{243}y_{217} \oplus y_{247}y_{231} \oplus y_{213}y_{235} \oplus y_{255}y_{251} \oplus y_{181}y_{239} \oplus y_{174}y_{44} \oplus y_{164}y_{29} \oplus y_{255}y_{247}y_{243}y_{213}y_{181}y_{174}$
$y_0 = y_0 \oplus z$
$y_{218} = y_{218} \oplus z$

$$\begin{tikzcd}
(K, IV) \arrow[r, "\mathrm{KLA}"] \arrow[d, dashed, "\mathcal{F}"] & X_0^k \arrow[r, "\mathrm{KSA}_{256}"] & X_0^p \arrow[r, "\mathrm{PKSA}_3"] & X_0 \arrow[d, "\Delta"] \\
(K, IV)_\Delta & X_{0,\Delta}^k \arrow[l, "\mathrm{KLA}^{-1}"'] & X_{0,\Delta}^p \arrow[l, "\mathrm{KSA}_{256}^{-1}"'] & X_{0,\Delta} \arrow[l, "\mathrm{PKSA}_3^{-1}"']
\end{tikzcd}$$

Fig. 1: Construction of the Related Key-IV function

Modified Pseudorandom Keystream Generation Algorithm (PRGA′). To obtain our modified PRGA we replace $\oplus$ (XOR) and $\cdot$ (AND) operations in the original PRGA with | (OR) operations.

Assumption. Based on the experimental results we obtained, we further assume that the output of PRGA[5] is independently and uniformly distributed. To obtain these results 100 keystream were statistically tested using the NIST Test Suites [1, 2]. During our experiments we used the default pseudorandom numbers generator implemented in the GMP library [3] to randomly generate 100 Key-IV pairs.

Theorem 1. *Let $\delta \in \{0,1\}^{256}$, q_1 the number of desired identical positions in the keystream and q_2 the maximum number of search trials. Then, Algorithm 3 finds at most q_1 identical positions in a maximum of q_2 trials.*

[5] implicitly PKSA and PKSA^{-1}

Proof. Let ω be the Hamming weight of δ. We note that in Algorithm 3 the bits $b_{i_1}, \ldots, b_{i_\omega}$ on position $i_1, \ldots, i_\omega$ are set. For $j \in [1, \omega]$, if bit b_{i_j} is taken into consideration while computing the output bit of PRGA then the output of PRGA$'$ is also set due to the replacement of the original operations $\oplus$ and $\cdot$ with $|$ operations. The same argument is valid if a bit of Espresso's internal state is influenced by b_{i_j}. $\square$

Remark 1. Note that if we run Algorithm 3 we do not obtain all the identical positions. This is due to the fact that Algorithm 3 is prone to producing internal collisions, and thus eliminate certain positions that are identical in both keystreams. Although we do not find all the positions, our algorithm has the advantage of finding identical keystream positions automatically.

Algorithm 3: Search for identical keystream positions

Input: Integers $\delta \in \{0,1\}^{256}$ and $q_1, q_2 > 0$
Output: Keystream positions φ

Set $s \leftarrow 0$ and $\varphi \leftarrow \varnothing$
Let $X_0 \in \{0,1\}^{256}$ be the zero state $(0, \ldots, 0)$
Construct $X_{0,\Delta} = X_0 \oplus \delta$
while $|\varphi| \leq q_1$ and $s < q_2$ **do**
  Set $b \leftarrow \mathrm{PRGA}'(X_{0,\Delta})$ and update state $X_{0,\Delta}$ with the current state
  if $b = 0$ **then**
    Update $\varphi \leftarrow \varphi \cup \{s\}$
  Set $s \leftarrow s + 1$
return φ

Based on Algorithm 3, in Table 1 we present some examples. More precisely, two initial states X_0 and $X_{0,\Delta}$ which differ only in the position presented in Table 1, Column 1, produce identical output bits in the positions found in Table 1, Column 3, among the initial 160 key stream bits obtained during the PRGA.

Flipped Bit Position	Number of Identical Keystream Bits	Positions of Identical Keystream Bits
31	25	0-15, 19, 22, 23, 27, 34, 42, 55, 58, 71
47	10	0, 1, 25, 36, 39, 43, 47, 51, 66, 82
71	21	0, 1, 3, 4, 7, 8, 11, 12, 15-17, 19, 20, 21, 24, 25, 49, 60, 67, 71, 75
95	32	0-5, 7-9, 11, 12, 16, 18, 20, 22, 23, 27, 31, 32, 35, 36, 39, 41, 43-45, 48, 49, 73, 91, 95, 99
119	22	0, 1, 4, 5, 8, 9, 12, 13, 16, 19, 24, 27, 35, 36, 40, 42, 46, 51, 56, 59, 65, 67
143	32	0-2, 4, 5, 8-10, 12-14, 16-19, 21-24, 33, 36, 40, 43, 48, 51, 59, 60, 64, 66, 70, 83, 91
167	51	0-2, 4-8, 10-12, 14-17, 19-22, 24-26, 28, 29, 32-34, 36-38, 40-43, 45-48, 57, 60, 64, 67, 72, 75, 83, 84, 88, 90, 94, 107, 115
191	58	0-2, 5, 6, 8, 9, 11, 13-16, 18-20, 22-26, 28-32, 34-36, 38-41, 43-45, 48, 49, 52, 56-58, 61, 62, 65-67, 69, 71, 72, 81, 84, 88, 91, 99, 108, 112, 114, 131
215	81	0, 1, 3-26, 29, 30, 32, 33, 35, 37-40, 42-44, 46-50, 52-56, 58-60, 62-65, 67-69, 72, 73, 76, 80-82, 85, 86, 89-91, 93, 95, 96, 105, 108, 112, 115, 123, 132, 136, 138, 155
239	96	1-3, 5-7, 9-11, 13-16, 18-21, 23-25, 27-50, 53, 54, 56, 57, 59, 61-64, 66-68, 70-74, 76-80, 82-84, 86-89, 91-93, 96, 97, 100, 104-106, 109, 110, 113-115, 117, 119, 120, 129, 132, 136, 139, 147, 156

Table 1: Propagation of a Single Bit Differential

3.1 Multiple Key-IV Trials with a Fixed Differential

We further consider that the adversary is allowed to produce any related Key-IV pairs for a given fixed differential. In this case, the while loop of our proposed algorithm (Algorithm 4) has to run an expected 2^{32} times with different randomly chosen (K, IV) pairs, until $X_{0,\Delta}$ has the correct padding. Once this happens, we output a related Key-IV pair (K, IV) and (K', IV'). In Table 2 we provide one such an example.

Algorithm 4: Search for Key-IV pairs that produce almost similar initial keystreams for a given δ

Input: An integer $\delta \in \{0,1\}^{256}$
Output: Key-IV pairs (K, IV) and (K', IV')

Set $s \leftarrow 0$
while $s = 0$ **do**
  Choose $K \in_R \{0,1\}^{128}$ and $IV \in_R \{0,1\}^{96}$
  Run $\mathrm{KSA}_{256}(K \| IV)$ and $\mathrm{PKSA}_3(K \| IV)$ routines to obtain an initial state $X_0 \in \{0,1\}^{256}$
  Compute the state $X_{0,\Delta} = X_0 \oplus \delta$
  Run $\mathrm{PKSA}_3^{-1}(X_{0,\Delta})$ and $\mathrm{KSA}_{256}^{-1}(X_{0,\Delta})$ routines to produce state $X_{0,\Delta}^k = K' \| IV' \| P'$
  if $P' =$ `0xfffffffe` **then**
    Set $s \leftarrow 1$
    return (K, IV) and (K', IV')

Key	IV	State
`0xd17117b8c5f9042` `43a69b7db0a535d2b`	`0x96a2736a408` `208e40e4ce2e9`	`0x7a53d74a086602e4943e052d9fc6865` `b37d9c35fb68b0cf78e8b5bcba7f0a273`
`0xcee2d9eee6c6da3` `625309eb7737e3f4d`	`0x52385c5ecfd` `2fa898bf48b67`	`0x7a53d74a086602e4943e052d9fc6865` `b37d9c35fb68b0cf78e8b5bcba7f1a273`

Table 2: Key-IV pairs which differ only in the 239^{th} position

3.2 Single Key-IV Trials with Multiple Differentials

In practice, the attacker has access to a single Key-IV pair and he has to produce a second Key-IV pair related to the one given. In this case, the attacker has to try around 2^{32} different values for δ, until Algorithm 5 outputs a pair.

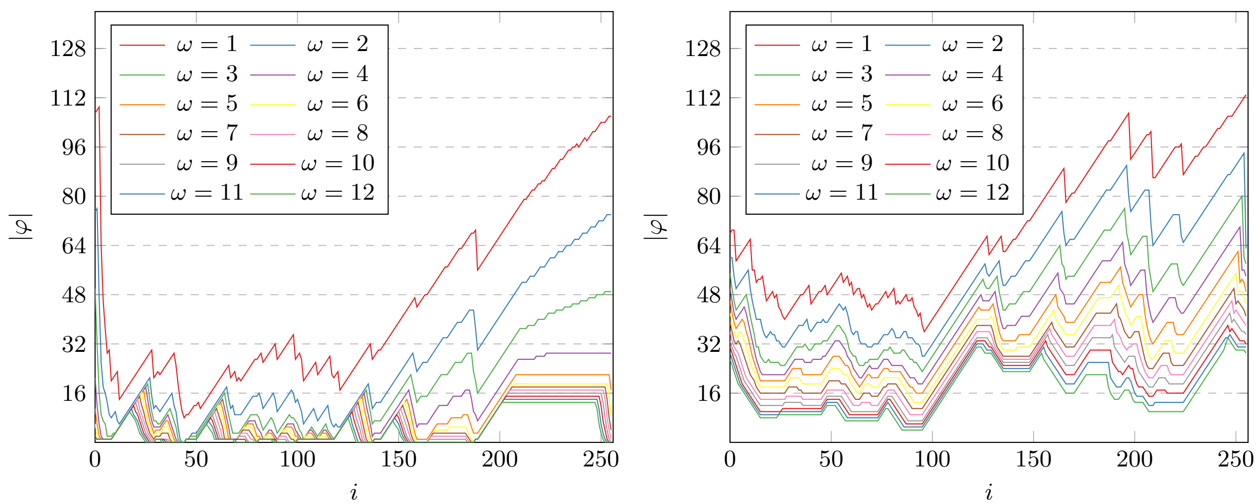
In Figure 2a we can see how cardinality of φ fluctuates depending on the iteration step i and the Hamming weight ω of δ. In [10], the authors introduce an algorithm that computes Key-IV pairs that produce similar initial Grain-128a keystreams for δ's of the form $0 \ldots 010 \ldots 0$. Our proposal (Algorithm 5) can be easily adapted to Grain-128a, and thus for comparison we also provide in Figure 2b the evolution of $|\varphi|$ in the case of Grain-128a.

For a given Δ, let X_1 be a random state such that $X_1 \neq X_{0,\Delta}$. Note that in Algorithm 5 parameter ℓ controls the probability of obtaining identical keystream bits for states X_0 and X_1 on the positions included in φ. More precisely, the probability of obtaining a collision for X_0 and X_1 is $1/2^{\ell}$. In Table 3 we can see the number of δ's such that $|\varphi| \geq 16$. Hence, for $\ell = 16$ in Algorithm 5 it is sufficient to run the while loop until $j \neq 5$ since $239 \cdot 137 \cdot 110 \cdot 69 \cdot 18 \geq 2^{32}$. In the case of Grain-128a it is sufficient to run the while loop until $j \neq 4$ since $256^4 \geq 2^{32}$.

Algorithm 5: Search for a Key-IV pair that produces an almost similar initial keystream with a given Key-IV pair (K, IV)

Input: A Key-IV pair (K, IV) and an integer $\ell > 0$
Output: A related Key-IV pair (K', IV')

Run $\mathrm{KSA}_{256}(K\|IV)$ and $\mathrm{PKSA}_3(K\|IV)$ routines to obtain an initial state $X_0 \in \{0,1\}^{256}$
Set the integer $j \leftarrow 0$ and the state $\delta = 0$
while $j \neq 256$ **do**
  Set the bit $\delta_j = 1$ and compute $j \leftarrow j + 1$
  for $i \in [0, 255]$ **do**
    Compute $\varphi \leftarrow$ *Algorithm* 3$(\delta, 160, 160)$
    if $|\varphi| < \ell$ **then**
      Skip the next instructions and go to the next i
    Compute the state $X_{0,\Delta} = X_0 \oplus \delta$
    Run $\mathrm{PKSA}_3^{-1}(X_{0,\Delta})$ and $\mathrm{KSA}_{256}^{-1}(X_{0,\Delta})$ routines to produce state $X_{0,\Delta}^k = K'\|IV'\|P'$
    if $P' =$ `0xfffffffe` **then**
      Set $s \leftarrow 1$
      return (K', IV')
    Rotate to the right $\delta = \delta \ggg 1$



(a) Espresso (b) Grain-128a

Fig. 2: The evolution of $|\varphi|$

Cipher	ω											
	1	2	3	4	5	6	7	8	9	10	11	12
Espresso	239	137	110	69	56(18)	51	49	48	47	0	0	0
Grain-128a	256	256	256	256	256	247	233	185	164	158	133	121

Table 3: Number of valid possibilities for $\ell = 16$

4 Key-IV Pairs That Produce Shifted Keystreams

In this section, we will show how an attacker can obtain related Key-IV pairs that produce 4-bit shifted keystreams. Our algorithm's main idea is that we can obtain a valid padding after running KSA^{-1} for 4 clocks if we fix the last four bits of the IV. We also provide a slower algorithm that uses the KSA routine, which will be useful in the next section. Our results are presented in Theorem 2. To increase the complexity of these attacks and consequently increase the security of the Espresso cipher, we recommend using the padding `0x7fffffff` instead of `0xfffffffe`. To support our claim we adapted Theorem 2 to the `0x7fffffff` padding and we presented the attacks' complexity in Theorem 3. Note that in all the attacks the $PRNG$ routine is composed of PKSA and PRGA.

Theorem 2. *There are two attack strategies that an adversary can use to produce* 4*-bit shifted keystreams. He can use either the KSA algorithm (see Algorithm 6) or the KSA*$^{-1}$ *algorithm (see Algorithm 7). The algorithms' have an average running time of* 2^{32} *and* 2^{28} *iterations, respectively.*

Proof. In the first case, the attacker can use the algorithm described in Algorithm 6 to obtain 4-bit shifted keystreams. For simplicity, we present in Table 4 the evolution of bits 255 to 224 of state X_0 after each run of the KSA routine. We highlighted with red the positions that are updated after each run[6] and we denote by ? the bits that are unknown to the attacker. We can easily see that after 4 clocks the bits from 255 to 228 are unknown to the attacker and are randomly distributed[7]. Hence, we should obtain a correct padding after 2^{28} iterations.

To obtain a shifted keystream we need an extra restriction. More precisely, when we run the KSA routine for 256 clocks state X_0 evolves to state X_{256}, but state $X'_0 = X_4$ evolves to state $X'_{256} = X_{260}$. Hence, to obtain the shifted keystream we need $z_{257} = z_{258} = z_{259} = z_{260} = 0$. The probability of this happening is $1/2^4$. Therefore, the average running time of Algorithm 6 is $2^{28} \cdot 2^4 = 2^{32}$.

Algorithm 6: Constructing Key-IV pairs that generate 4-bit shifted keystream (forward construction)

Output: Key-IV pairs (K', IV') and (K, IV)

```
Set s ← 0
while s = 0 do
    Choose K ∈_R {0,1}^128 and IV ∈_R {0,1}^96
    Run KSA(K‖IV) routine for 4 clocks to obtain a state X'_0 = K'‖IV'‖P'
    if P' = 0xfffffffe then
        Run KSA(K'‖IV') and PRNG routine for 252 clocks and 4 clocks, respectively, to obtain bits
          z_257, z_258, z_259, z_260
        if z_257 = z_258 = z_259 = z_260 = 0 then
            Set s ← 1
            return (K, IV) and (K', IV')
```

A more efficient strategy is described in Algorithm 7. In this case, we set the last four bits of the initialization vector to 1. In Table 5 we can see the state evolution of bits 255 to 220 after running the KSA^{-1} routine. We separated the extra four bits of the IV by a straight line and we denoted by $\times$ the bits that are unknown to the attacker, but are irrelevant for our attack. In this case, the updated positions are 252, 248, 244, 240, 236, 232. We can easily see that after 4 clocks we have 24 unknown positions. Thus, the expected running time until we obtain a correct padding is 2^{24}.

As in the first case, we need some additional restrictions. We can see that after running the KSA routine for 256 clocks state X_0 evolves to state X_{256}, but state $X'_0 = X_{-4}$ evolves to state $X'_{256} = X_{252}$. Hence, to

[6] 255, 251, 247, 243, 239, 235, 231

[7] due to the key bits involved in their computation

0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
?	0	1	1	?	1	1	1	?	1	1	1	?	1	1	1	?	1	1	1	?	1	1	1	?	1	1	1	1	1	1	1
?	?	0	1	?	?	1	1	?	?	1	1	?	?	1	1	?	?	1	1	?	?	1	1	?	?	1	1	1	1	1	1
?	?	?	0	?	?	?	1	?	?	?	1	?	?	?	1	?	?	?	1	?	?	?	1	?	?	?	1	1	1	1	1
?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	1	1	1	1

Table 4: State evolution of bits 255 to 224 after applying the KSA routine (Algorithm 6)

obtain the shifted keystream we need $z_{253} = z_{254} = z_{255} = z_{256} = 0$. Therefore, the average running time of Algorithm 7 is $2^{24} \cdot 2^4 = 2^{28}$.

Algorithm 7: Constructing Key-IV pairs that generate 4-bit shifted keystream (backward construction)

Output: Key-IV pairs (K'', IV'') and (K, IV)

Set $s \leftarrow 0$
while $s = 0$ **do**
  Choose $K \in_R \{0,1\}^{128}$ and $V \in_R \{0,1\}^{92}$
  Set $IV \leftarrow V \| \texttt{0xf}$
  Run KSA$^{-1}(K \| IV)$ routine for 4 clocks to obtain a state $X_0'' = K'' \| IV'' \| P''$
  if $P'' = \texttt{0xfffffffe}$ **then**
    Run KSA$(K \| IV)$ and PRNG routine for 252 clocks and 4 clocks, respectively, to obtain bits $z_{253}, z_{254}, z_{255}, z_{256}$
    if $z_{253} = z_{254} = z_{255} = z_{256} = 0$ **then**
      Set $s \leftarrow 1$
      return (K, IV) and (K'', IV'')

0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	?	1	1	1	?	1	1	1	?	1	1	1	?	1	1	1	?	1	1	1	?	1	1	1	1	1	1	1	1	1	1	1	×
1	1	?	?	1	1	?	?	1	1	?	?	1	1	?	?	1	1	?	?	1	1	?	?	1	1	1	1	1	1	1	1	1	1	×	×
1	?	?	?	1	?	?	?	1	?	?	?	1	?	?	?	1	?	?	?	1	?	?	?	1	1	1	1	1	1	1	1	1	×	×	×
?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	1	1	1	1	1	1	1	1	×	×	×	×

Table 5: State evolution of bits 255 to 220 after applying the KSA^{-1} routine (Algorithm 7)

□

We further consider the padding `0x7fffffff` and we study its impact on the average time needed to obtain shifted keystreams. We can easily see that this small change increases the complexity of finding shifted keystreams. Hence, we suggest using this padding instead of the classical one. Note that due to the attacks presented in Section 3, it is sufficient to devise a padding scheme that induces an average running time greater than 2^{32}.

Theorem 3. *There are two attack strategies that an adversary can use to produce 8-bit shifted keystreams. He can use either the KSA algorithm (see Algorithm 8) or the KSA^{-1} algorithm (see Algorithm 9). The algorithms' have an average running time of 2^{40} or 2^{34} iterations, respectively.*

Proof (sketch). The proof is similar to the proof of Theorem 2 and thus we omit some details.

In the first case, the attacker can use the algorithm described in Algorithm 8 to obtain 8-bit shifted keystreams. The evolution of bits 255 to 224 of state X_0 is presented in Table 6. We can easily see that after 8 clocks the bits from 255 to 224 are unknown to the attacker and thus he will obtain a correct padding after 2^{32} iterations. Note that, when we run the KSA routine for 256 clocks state X_0 evolves to state X_{256}, but state $X'_0 = X_8$ evolves to state $X'_{256} = X_{264}$. Hence, to obtain the shifted keystream we need $z_{257} = \ldots = z_{264} = 0$. Therefore, the average running time of Algorithm 8 is $2^{32} \cdot 2^8 = 2^{40}$.

Algorithm 8: Constructing Key-IV pairs that generate 8-bit shifted keystream (forward construction)

Output: Key-IV pairs (K', IV') and (K, IV)

Set $s \leftarrow 0$
while $s = 0$ **do**
    Choose $K \in_R \{0,1\}^{128}$ and $IV \in_R \{0,1\}^{96}$
    Run KSA($K \| IV$) routine for 8 clocks to obtain a state $X'_0 = K' \| IV' \| P'$
    if $P' =$ `0x7fffffff` **then**
        Run KSA($K' \| IV'$) and PRNG routine for 248 clocks and 8 clocks, respectively, to obtain bits $z_{257}, \ldots, z_{264}$
        if $z_{257} = \ldots = z_{264} = 0$ **then**
            Set $s \leftarrow 1$
            return (K, IV) and (K', IV')

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0
?	1	1	1	?	1	1	1	?	1	1	1	?	1	1	1	?	1	1	1	?	1	1	1	?	1	1	1	1	1	1	1
?	?	1	1	?	?	1	1	?	?	1	1	?	?	1	1	?	?	1	1	?	?	1	1	?	?	1	1	1	1	1	1
?	?	?	1	?	?	?	1	?	?	?	1	?	?	?	1	?	?	?	1	?	?	?	1	?	?	?	1	1	1	1	1
?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	1	1	1	1
?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	1	1	1
?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	1	1
?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	1
?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?

Table 6: State evolution of bits 255 to 224 after applying the KSA routine (Algorithm 8)

The second strategy is described in Algorithm 9. In this case, we set the last six bits of the initialization vector to 1. In Table 7 we can see the state evolution of bits 255 to 218. Note that we also have position 218 updated. We can easily see that after 8 clocks we have 26 unknown positions. Thus, the expected running time until we obtain a correct padding is 2^{26}. Note that, after running the KSA routine for 256 clocks state X_0 evolves to state X_{256}, but state $X'_0 = X_{-8}$ evolves to state $X'_{256} = X_{248}$. Hence, to obtain the shifted keystream we need $z_{249} = \ldots = z_{256} = 0$. Therefore, the average running time of Algorithm 9 is $2^{26} \cdot 2^8 = 2^{34}$. □

In Table 8 we present a set of examples for Algorithms 6 to 9.

Algorithm 9: Constructing Key-IV pairs that generate 8-bit shifted keystream (backward construction)

Output: Key-IV pairs (K'', IV'') and (K, IV)

Set $s \leftarrow 0$
while $s = 0$ **do**
  Choose $K \in_R \{0,1\}^{128}$ and $V \in_R \{0,1\}^{90}$
  Set $IV \leftarrow V \| \texttt{0x3f}$
  Run $\text{KSA}^{-1}(K\|IV)$ routine for 8 clocks to obtain a state $X''_0 = K''\|IV''\|P''$
  if $P'' = \texttt{0x7fffffff}$ **then**
    Run $\text{KSA}(K\|IV)$ and PRNG routine for 248 clocks and 8 clocks, respectively, to obtain bits $z_{249}, \ldots, z_{256}$
    if $z_{249} = \ldots = z_{256} = 0$ **then**
      Set $s \leftarrow 1$
      return (K, IV) and (K'', IV'')

1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1
1	1	1	?	1	1	1	?	1	1	1	?	1	1	1	?	1	1	1	?	1	1	1	?	1	1	1	1	1	1	0	1	1	1	1	1	1	?
1	1	?	?	1	1	?	?	1	1	?	?	1	1	?	?	1	1	?	?	1	1	?	?	1	1	1	1	1	0	1	1	1	1	1	1	?	?
1	?	?	?	1	?	?	?	1	?	?	?	1	?	?	?	1	?	?	?	1	?	?	?	1	1	1	1	0	1	1	1	1	1	1	?	?	×
?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	1	1	1	0	1	1	1	1	1	1	?	?	×	×
?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	1	1	0	1	1	1	1	1	1	?	?	×	×	×
?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	1	0	1	1	1	1	1	1	?	?	×	×	×	×
?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	0	1	1	1	1	1	1	?	?	×	×	×	×	×
?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	?	1	1	1	1	1	1	?	?	×	×	×	×	×	×

Table 7: State evolution of bits 255 to 218 after applying the KSA^{-1} routine (Algorithm 9)

	Key	IV	Keystream
Algorithm 6	0x2a13a9539900630	0x2c112eb15ad	0x6757b665d8a3e72
	f7a721a25e2193026	d58ec3a99599a	bd2bdfdc326a93404
	0xa13a9539900630f	0xc112eb15add	0x757b665d8a3e72b
	7a721a25e21930262	58ec3ad959aef	d2bdfdc326a934043
Algorithm 7	0xb1d331f900270d5	0x7e8b7fd12fe	0x6172f847028df4f
	f6a43069b404888cf	bf7c2f17d86ff	eb0906ea001fc6d1f
	0xfb1d331f900270d	0xf7e8b7fd12f	0x56172f847028df4
	5f6a43069b404888c	ebf7c2b17684f	feb0906ea001fc6d1
Algorithm 8	0xb64e24eddec37cf	0xaee197ec26b	0x0261c57c8b0238e
	8a30970c2155d30cf	76484bceb639d	469f8e67299c3ed57
	0x4e24eddec37cf8a	0xe197ec26b76	0x61c57c8b0238e46
	30970c2155d30cfae	4849c49c2f33f	9f8e67299c3ed5742
Algorithm 9	0xd90e03c9fdcf7ce	0xb6a7a25b255	0xca75acab22d4c9e
	231f9ac4c322ad987	b956c9672467f	e1fb6c9045f1379e0
	0xd7d90e03c9fdcf7	0x87b6a7a25b2	0x05ca75acab22d4c
	ce231f9ac4c322ad9	55b954817ee18	9ee1fb6c9045f1379

Table 8: Key-IV pairs that produce shifted keystreams

5 Key Recovery Algorithms

According to the results presented in Section 4, we know that related IV's exist. Note that we also know the average running time τ needed to find such a pair (IV, IV') and the keystream shift σ that they produce. Since we do not have access to the secret key, a simple strategy to finding such a pair is to choose a random IV and use it to generate α bits that are stored in memory. Then clock the NFSR either forward or backwards, and then randomly generate[8] IV' until we obtain a keystream with the desired shift σ. Note that the probability of randomly obtaining the desired shift is $1/2^{\alpha-\sigma}$. Therefore, if we choose a large enough α the probability is small enough.[9]

We further assume that we are in possession of two related IV's and we want to recover the secret key. Using a related IV pair, we can use a guess and determine attack[10] to recover the secret key. We propose three key recovery attacks. The first one (forward construction) uses IV pairs generated using the KSA routine, while the second (backward construction) use IV-pairs created using the KSA^{-1} routine. The last attack (mixed construction) assumes that we are in possession of two IV-pairs and is a combination of the forward and backward constructions.

5.1 Forward Construction

Before presenting our attack, we want to see which NFSR positions are modified[11] by the KSA routine after each clock. These positions are presented in Table 9.

Clock	Cells
1	193, 197, 201, 205, 209, 213, 217, 231, 235, 239, 243, 247, 251, 255
2	192, 193, 196, 197, 200, 201, 204, 205, 208, 209, 212, 213, 216, 217, 230, 231, 234, 235, 238, 239, 242, 243, 246, 247, 250, 251, 254, 255
3	$191-193$, $195-197$, $199-201$, $203-205$, $207-209$, $211-213$, $215-217$, $229-231$, $233-235$, $237-239$, $241-243$, $245-247$, $249-251$, $253-255$
4	$190-217$, $228-255$
5	$189-217$, $227-255$
6	$188-217$, $226-255$
7	$187-217$, $225-255$
8	$186-217$, $224-255$

Table 9: Modified cells after running the KSA routine

We first start our study with the classical Espresso cipher. Hence, looking at the KLA and KSA routines, we can see that on clock $i+1$ K's bits used by the feedback functions and the output function are found on positions $0-(127-i)$, where $i \in [0,3]$. According to Table 9, none of K's bits are modified. Similarly, we can see that IV's bits used by the feedback functions are not modified. In the case of the output function, we can see that the only positions that are modified are 213 and 217 at clocks $2-4$. Luckily we can recover them from IV''s bits. Also, note that for $i = 2, 3$ the value found on position 222 is 1 (due to the shifting of the initial padding).

As stated in Table 9, some positions between 223 and 255 are modified. But we are working with two related IV's that produce 4-bit shifted keystreams. Hence, we know that after 4 clocks we end up with a valid padding. Hence, we know their values.

[8] an average of τ IV's are generated

[9] *e.g.* $\alpha = 100$ *since* $\sigma = 4$ *or* 8

[10] An attacker starts by brute-forcing parts of a cryptographic key and then uses various methods to determine the remaining unknown portions, often relying on prior knowledge or observations about the encryption process.

[11] and hence, unknown to an attacker

Rewriting the feedback functions we obtain

$$
\begin{aligned}
g_{255}(X_{i+1}) &= k_i \oplus k_{i+41}k_{i+70} \oplus z(X_{i+1}) \\
g_{251}(X_{i+1}) &= \begin{cases} 1 \oplus k_{i+42}k_{i+83} \oplus k_{i+8} & \text{if } i \neq 3 \\ k_{i+42}k_{i+83} \oplus k_{i+8} & \text{if } i = 3 \end{cases} \\
g_{247}(X_{i+1}) &= 1 \oplus k_{i+44}k_{i+102} \oplus k_{i+40} \\
g_{243}(X_{i+1}) &= 1 \oplus k_{i+43}k_{i+118} \oplus k_{i+103} \\
g_{239}(X_{i+1}) &= 1 \oplus k_{i+46}iv_{i+13} \oplus k_{i+117} \\
g_{235}(X_{i+1}) &= 1 \oplus k_{i+67}k_{i+90}k_{i+110}iv_{i+9} \\
g_{231}(X_{i+1}) &= 1 \oplus k_{i+50}iv_{i+31} \oplus iv_{i+61} \\
g_{217}(X_{i+1}) &= iv_{i+90} \oplus k_{i+3}k_{i+32} \oplus z(X_{i+1}) \\
g_{213}(X_{i+1}) &= iv_{i+86} \oplus k_{i+4}k_{i+45} \\
g_{209}(X_{i+1}) &= iv_{i+82} \oplus k_{i+6}k_{i+64} \\
g_{205}(X_{i+1}) &= iv_{i+78} \oplus k_{i+5}k_{i+80} \\
g_{201}(X_{i+1}) &= iv_{i+74} \oplus k_{i+8}k_{i+103} \\
g_{197}(X_{i+1}) &= iv_{i+70} \oplus k_{i+29}k_{i+52}k_{i+72}k_{i+99} \\
g_{193}(X_{i+1}) &= iv_{i+66} \oplus k_{i+12}k_{i+121}
\end{aligned}
$$

where

$$
\begin{aligned}
z'(X_{i+1}) &= k_{i+80} \oplus k_{i+99} \oplus iv_{i+9} \oplus iv_{i+59} \oplus iv_{i+53} \oplus iv_{i+46}k_{i+44} \oplus iv_{i+36}k_{i+29} \\
z(X_{i+1}) &= \begin{cases} z'(X_{i+1}) \oplus iv_{85}iv_{89} \oplus iv_{94+i} & \text{if } i = 0 \\ z'(X_{i+1}) \oplus iv'_{81+i}iv'_{85+i} \oplus 1 \oplus iv'_{81+i}iv_{53+i}iv_{46+i} \oplus iv_{94+i} & \text{if } i = 1 \\ z'(X_{i+1}) \oplus iv'_{81+i}iv'_{85+i} \oplus iv'_{81+i}iv_{53+i}iv_{46+i} & \text{if } i = 2 \\ z'(X_{i+1}) \oplus iv'_{81+i}iv'_{85+i} \oplus iv'_{81+i}iv_{53+i}iv_{46+i} & \text{if } i = 3 \end{cases}
\end{aligned}
$$

From Espresso's feedback functions we can see that the only functions containing retrievable key bits are $g_{255}, g_{251}, g_{247}, g_{243}, g_{239}$ and g_{217}. Note that all positions, except 217, can be recovered from the padding (see Table 4). In the case of g_{217}, the value can be recovered from IV''s bits. Therefore, we obtain Algorithm 10 for recovering some of K's bits. To ease understanding, in Algorithm 10 we marked at each step the recovered key bits $\%rec$ and the used key bits $\%use$.

We further develop a key recovery algorithm for our proposed version of Espresso. To simplify description, we first write some intermediary feedback function

$$
\begin{aligned}
g'_{251}(X_{i+1}) &= 1 \oplus k_{i+42}k_{i+83} \oplus k_{i+8} \\
g'_{247}(X_{i+1}) &= 1 \oplus k_{i+44}k_{i+102} \oplus k_{i+40} \\
g'_{243}(X_{i+1}) &= 1 \oplus k_{i+43}k_{i+118} \oplus k_{i+103} \\
g'_{239}(X_{i+1}) &= 1 \oplus k_{i+46}iv_{i+13} \oplus k_{i+117} \\
g'_{235}(X_{i+1}) &= 1 \oplus k_{i+67}k_{i+90}k_{i+110}iv_{i+9} \\
g'_{231}(X_{i+1}) &= 1 \oplus k_{i+50}iv_{i+31} \oplus iv_{i+61} \\
g'_{217}(X_{i+1}) &= k_{i+3}k_{i+32} \\
g'_{213}(X_{i+1}) &= iv_{i+86} \oplus k_{i+4}k_{i+45} \\
g'_{209}(X_{i+1}) &= iv_{i+82} \oplus k_{i+6}k_{i+64} \\
g'_{205}(X_{i+1}) &= iv_{i+78} \oplus k_{i+5}k_{i+80}
\end{aligned}
$$

Algorithm 10: Key bits recovery algorithm for the `0xfffffffe` padding (forward construction)

Input: Chosen IV's IV and IV' and key bits k_j, where
$j \in \{4-6, 29-35, 44-49, 70-72, 84-86, 99-102, 121\}$

Output: 24 key bits k_j, where $j \in \{0-3, 8-11, 40-43, 80-83, 103-106, 117-120\}$

for $i \in [0,3]$ **do**
  $k_{i+117} \leftarrow k_{i+46}iv_{i+13}$ $\%rec: 117-120\ use: 46-49$
$k_{105} \leftarrow k_{45}k_{120}$ $\%rec: 105\ use: 45, 120$
$k_{43} \leftarrow k_{47}k_{105}$ $\%rec: 43\ use: 47, 105$
for $i \in [0,3]$ **do**
  if $i \neq 2$ **then** $k_{i+103} \leftarrow k_{i+43}k_{i+118}$ $\%rec: 103, 104, 106\ use: 43, 44, 46, 118, 119, 121$
for $i \in [0,2]$ **do**
  $k_{i+40} \leftarrow k_{i+44}k_{i+102}$ $\%rec: 40-42\ use: 44-46, 102-104$
$o_3 \leftarrow k_{102} \oplus iv_{12} \oplus iv_{62} \oplus iv'_{84} \oplus iv'_{88} \oplus iv_{56} \oplus iv_{49}k_{47} \oplus iv_{39}k_{32} \oplus iv'_{i+81} \oplus iv'_{i+81}iv_{i+53}iv_{i+46}$ $\%use: 102, 47, 32$
$k_{83} \leftarrow o_3 \oplus iv'_{89} \oplus iv_{93} \oplus k_6k_{35}$ $\%rec: 83\ use: 6, 35$
$k_3 \leftarrow o_3 \oplus k_{83} \oplus k_{44}k_{73}$ $\%rec: 3\ use: 83, 44, 73$
for $i \in [0,2]$ **do**
  $o_i \leftarrow k_{i+99} \oplus iv_{i+9} \oplus iv_{i+59} \oplus iv'_{i+81} \oplus iv'_{i+85} \oplus iv_{i+53} \oplus iv_{i+46}k_{i+44} \oplus iv_{i+36}k_{i+29}$
   $\%use: 99-101, 44-46, 29-31$
  if $i \neq 0$ **then** $o_i \leftarrow o \oplus 1 \oplus iv'_{i+81} \oplus iv'_{i+81}iv_{i+53}iv_{i+46}$
  if $i = 2$ **then** $o_i \leftarrow o \oplus 1$
  else $o_i \leftarrow o \oplus iv_{i+94}$
  $k_{i+80} \leftarrow o_i \oplus iv'_{i+86} \oplus iv_{i+90} \oplus k_{i+3}k_{i+32}$ $\%rec: 80-82\ use: 3-5, 32-34$
for $i \in [0,2]$ **do**
  $k_{i+0} \leftarrow o_i \oplus k_{i+80} \oplus k_{i+41}k_{i+70}$ $\%rec: 0-2\ use: 80-82, 41-43, 70-72$
for $i \in [0,3]$ **do**
  $k_{i+8} \leftarrow k_{i+42}k_{i+83}$ $\%rec: 8-11\ use: 42-45, 83-86$
  if $i = 3$ **then** $k_{i+8} \leftarrow k_{i+8} \oplus 1$

$$g'_{201}(X_{i+1}) = iv_{i+74} \oplus k_{i+8}k_{i+103}$$
$$g'_{197}(X_{i+1}) = iv_{i+70} \oplus k_{i+29}k_{i+52}k_{i+72}k_{i+99}$$
$$g'_{193}(X_{i+1}) = \begin{cases} iv_{i+66} \oplus k_{i+12}k_{i+121} & \text{if } i \leq 6 \\ iv_{i+66} \oplus k_{i+12}iv_{i-6} & \text{otherwise} \end{cases}$$

Using arguments similar to the classical case, we obtain the following feedback functions

$$g_{255}(X_{i+1}) = k_i \oplus k_{i+41}k_{i+70} \oplus z(X_{i+1})$$
$$g_{251}(X_{i+1}) = \begin{cases} g'_{251}(X_{i+1}) & \text{if } i \leq 3 \\ g'_{251}(X_{i+1}) \oplus 1 \oplus k_{i-4} \oplus k_{i+37}k_{i+66} \oplus z(X_{i-3}) & \text{otherwise} \end{cases}$$
$$g_{247}(X_{i+1}) = \begin{cases} g'_{247}(X_{i+1}) & \text{if } i \leq 3 \\ g'_{247}(X_{i+1}) \oplus k_{i+38}k_{i+79} \oplus k_{i+4} & \text{otherwise} \end{cases}$$
$$g_{243}(X_{i+1}) = \begin{cases} g'_{243}(X_{i+1}) & \text{if } i \leq 3 \\ g'_{243}(X_{i+1}) \oplus k_{i+40}k_{i+98} \oplus k_{i+36} & \text{otherwise} \end{cases}$$
$$g_{239}(X_{i+1}) = \begin{cases} g'_{239}(X_{i+1}) & \text{if } i \leq 3 \\ g'_{239}(X_{i+1}) \oplus k_{i+39}k_{i+114} \oplus k_{i+99} & \text{otherwise} \end{cases}$$
$$g_{235}(X_{i+1}) = \begin{cases} g'_{235}(X_{i+1}) & \text{if } i \leq 3 \\ g'_{235}(X_{i+1}) \oplus k_{i+42}iv_{i+9} \oplus k_{i+113} & \text{otherwise} \end{cases}$$

$$g_{231}(X_{i+1}) = \begin{cases} g'_{231}(X_{i+1}) & \text{if } i \leq 3 \\ g'_{231}(X_{i+1}) \oplus k_{i+63}k_{i+86}k_{i+106}iv_{i+5} \oplus k_{i+7}k_{i+116} & \text{otherwise} \end{cases}$$

$$g_{217}(X_{i+1}) = \begin{cases} g'_{217}(X_{i+1}) \oplus iv_{i+90} \oplus z(X_{i+1}) & \text{if } i \leq 5 \\ g'_{217}(X_{i+1}) \oplus z(X_{i+1}) & \text{if } i = 6 \\ g'_{217}(X_{i+1}) \oplus 1 \oplus z(X_{i+1}) & \text{if } i = 7 \end{cases}$$

$$g_{213}(X_{i+1}) = \begin{cases} g'_{213}(X_{i+1}) & \text{if } i \leq 3 \\ g'_{213}(X_{i+1}) \oplus k_{i-1}k_{i+28} \oplus z(X_{i-3}) & \text{otherwise} \end{cases}$$

$$g_{209}(X_{i+1}) = \begin{cases} g'_{209}(X_{i+1}) & \text{if } i \leq 3 \\ g'_{209}(X_{i+1}) \oplus k_i k_{i+41} & \text{otherwise} \end{cases}$$

$$g_{205}(X_{i+1}) = \begin{cases} g'_{205}(X_{i+1}) & \text{if } i \leq 3 \\ g'_{205}(X_{i+1}) \oplus k_{i+2}k_{i+60} & \text{otherwise} \end{cases}$$

$$g_{201}(X_{i+1}) = \begin{cases} g'_{201}(X_{i+1}) & \text{if } i \leq 3 \\ g'_{201}(X_{i+1}) \oplus k_{i+1}k_{i+76} & \text{otherwise} \end{cases}$$

$$g_{197}(X_{i+1}) = \begin{cases} g'_{197}(X_{i+1}) & \text{if } i \leq 3 \\ g'_{197}(X_{i+1}) \oplus k_{i+4}k_{i+99} & \text{otherwise} \end{cases}$$

$$g_{193}(X_{i+1}) = \begin{cases} g'_{193}(X_{i+1}) & \text{if } i \leq 3 \\ g'_{193}(X_{i+1}) \oplus k_{i+25}k_{i+48}k_{i+68}k_{i+95} & \text{otherwise} \end{cases}$$

where

$$\begin{aligned} z'(X_{i+1}) = {} & k_{i+80} \oplus k_{i+99} \oplus iv_{i+9} \oplus 1 \oplus iv_{i+59} \oplus b_{243}b_{217} \oplus b_{247}b_{231} \oplus b_{213}b_{235} \oplus b_{255}b_{251} \\ & \oplus iv_{i+53}b_{239} \oplus iv_{i+46}k_{i+44} \oplus iv_{i+36}k_{i+29} \oplus b_{255}b_{247}b_{243}b_{213}iv_{i+53}iv_{i+46} \end{aligned}$$

$$z(X_{i+1}) = \begin{cases} z'(X_{i+1}) \oplus iv_{i+94} & \text{if } i < 2 \\ z'(X_{i+1}) \oplus 1 & \text{if } i = 3, 4 \\ z'(X_{i+1}) \oplus 1 \oplus iv_{i+56} \oplus k_{i+45}iv_{i+26} & \text{if } 4 < i < 7 \\ z'(X_{i+1}) \oplus 1 \oplus iv_{i+56} \oplus k_{i+45}iv_{i+26} \oplus k_{12}k_{121} & \text{if } i = 7 \end{cases}$$

and

$$b_{255} = \begin{cases} k_{i-1} \oplus k_{i+40}k_{i+69} \oplus z(X_i) & \text{if } 0 < i \leq 4 \\ 1 & \text{otherwise} \end{cases}$$

$$b_{251} = \begin{cases} 1 \oplus k_{i+41}k_{i+82} \oplus k_{i+7} & \text{if } 0 < i \leq 4 \\ 1 & \text{otherwise} \end{cases}$$

$$b_{247} = \begin{cases} 1 \oplus k_{i+43}k_{i+101} \oplus k_{i+39} & \text{if } 0 < i \leq 4 \\ 1 & \text{otherwise} \end{cases}$$

$$b_{243} = \begin{cases} 1 \oplus k_{i+42}k_{i+117} \oplus k_{i+102} & \text{if } 0 < i \leq 4 \\ 1 & \text{otherwise} \end{cases}$$

$$b_{239} = \begin{cases} 1 \oplus k_{i+45}iv_{i+12} \oplus k_{i+116} & \text{if } 0 < i \leq 4 \\ 1 & \text{otherwise} \end{cases}$$

$$b_{235} = \begin{cases} 1 \oplus k_{i+66}k_{i+89}k_{i+109}iv_{i+8} & \text{if } 0 < i \leq 4 \\ 1 & \text{otherwise} \end{cases}$$

$$b_{231} = \begin{cases} 1 \oplus iv_{i+60} \oplus k_{i+49}iv_{i+30} & \text{if } 0 < i \leq 4 \\ 1 & \text{otherwise} \end{cases}$$

$$b_{217} = \begin{cases} iv_{i+89} & \text{if } i = 0 \\ iv_{i+89} \oplus k_{i+2}k_{i+31} \oplus z(X_i) & \text{if } 0 < i \leq 4 \\ iv'_{i+81} & \text{otherwise} \end{cases}$$

$$b_{213} = \begin{cases} iv_{i+85} & \text{if } i = 0 \\ iv_{i+85} \oplus k_{i+3}k_{i+44} & \text{if } 0 < i \leq 4 \\ iv'_{i+77} & \text{otherwise} \end{cases}$$

We can see that the only feedback function that contain retrievable bits are g_{255}, g_{251}, g_{247}, g_{243}, g_{239}, g_{235} and g_{217}. When we tried to recover the key bits, we found some loops that prevented us from recovering two bits. More precisely, we found the following dependencies

$$k_{87} \hookleftarrow k_{12} \hookleftarrow k_{87} \quad \text{and} \quad k_8 \hookleftarrow k_{83} \hookleftarrow k_{85} \hookleftarrow k_8,$$

where $a \hookleftarrow b$ denotes "to compute the value of a we need to know b". Also, we could not find a method for recovering k_9 to k_{10}, since their dependencies interfered with k_{87} to k_{84}. Hence, we do not claim that this is the optimal solution for recovering the key bits. Our solution is presented in Algorithm 11.

5.2 Backward Construction

In this case, we want to see how the KSA^{-1} routine affects the NFSR positions after each clock. The results are presented in Table 10.

With respect to the classical Espresso, we can see that the KSA^{-1} routine on clock $i-1$ K's and IV's bits used by the feedback functions are unchanged, where $i \in \{0, -1, -2, -3\}$. Moreover, we can see that the first 4 bits of IV' coincide with the last 4 bits of K. The only problem that we encounter is on position 218. Here on the last clock the feedback function uses x_{-1}, but the value can be easily obtained from k_{40}, k_{69} and the output function.

In the case of the output function, the only problematic positions are 213 and 217 from the -4 clock. The two bits coincide with bits 210 and 214 from clock -1. Lastly, for positions 232 to 255 we know the exact values due to related Key-IV pairs used by the algorithm. Therefore, we obtain

$$g_0^{-1}(X_{i-1}) = \begin{cases} x_{i+40}x_{i+69} \oplus z^{-1}(X_{i-1}) & \text{if } i = 0 \\ 1 \oplus x_{i+40}x_{i+69} \oplus z^{-1}(X_{i-1}) & \text{if } i \neq 0 \end{cases}$$

$$g_{252}^{-1}(X_{i-1}) = 1 \oplus k_{i+41}k_{i+82} \oplus k_{i+7}$$

$$g_{248}^{-1}(X_{i-1}) = 1 \oplus k_{i+43}k_{i+101} \oplus k_{i+39}$$

$$g_{244}^{-1}(X_{i-1}) = 1 \oplus k_{i+42}k_{i+117} \oplus k_{i+102}$$

$$g_{240}^{-1}(X_{i-1}) = 1 \oplus k_{i+45}iv_{i+12} \oplus k_{i+116}$$

$$g_{236}^{-1}(X_{i-1}) = 1 \oplus k_{i+66}k_{i+89}k_{i+109}iv_{i+8}$$

$$g_{232}^{-1}(X_{i-1}) = 1 \oplus k_{i+49}iv_{i+30} \oplus iv_{i+60}$$

$$g_{218}^{-1}(X_{i-1}) = \begin{cases} iv_{i+89} \oplus k_{i+2}k_{i+31} \oplus z^{-1}(X_{i-1}) & \text{if } i \neq -3 \\ iv_{i+89} \oplus g_0^{-1}(X_{-1})k_{i+31} \oplus z^{-1}(X_{i-1}) & \text{if } i = -3 \end{cases}$$

$$g_{214}^{-1}(X_{i-1}) = iv_{i+85} \oplus k_{i+3}k_{i+44}$$

Algorithm 11: Key bits recovery algorithm for the `0xeffffffff` padding (forward construction)

Input: Chosen IV's IV and IV' and key bits k_j, where
$j \in \{0-3, 8-12, 29-53, 67-77, 88-102, 111-116, 121-125\}$

Output: 27 key bits k_j, where $j \in \{4-7, 13-15, 80-87, 103-110, 117-120\}$

Function *update_bits*():
- $b_{213} \leftarrow b_{213} \oplus k_{i+3} \oplus k_{i+44}$ $\%use: 4-7, 45-48$
- $b_{217} \leftarrow iv_{i+89} \oplus k_{i+2}k_{i+31} \oplus o_{i-1}$ $\%use: 3-6, 32-35$
- $b_{231} \leftarrow b_{231} \oplus k_{i+49}iv_{i+30} \oplus iv_{i+60}$ $\%use: 50-53$
- $b_{235} \leftarrow b_{235} \oplus k_{i+66}k_{i+89}k_{i+109}iv_{i+8}$ $\%use: 67-70, 90-93, 110-113$
- $b_{239} \leftarrow b_{239} \oplus k_{i+45}iv_{i+12} \oplus k_{i+116}$ $\%use: 46-49, 117-120$
- $b_{243} \leftarrow b_{243} \oplus k_{i+42}k_{i+117} \oplus k_{i+102}$ $\%use: 43-46, 118-121, 103-106$
- $b_{247} \leftarrow b_{247} \oplus k_{i+43}k_{i+101} \oplus k_{i+39}$ $\%use: 44-47, 102-105, 40-43$
- $b_{251} \leftarrow b_{251} \oplus k_{i+41}k_{i+82} \oplus k_{i+7}$ $\%use: 42-45, 83-86, 8-11$
- $b_{255} \leftarrow k_{i-1} \oplus k_{i+40}k_{i+69} \oplus o_{i-1}$ $\%use: 0-3, 41-44, 70-73$

Function *update_o*(i):
- $o \leftarrow k_{i+99} \oplus iv_{i+9} \oplus 1 \oplus iv_{i+59} \oplus b_{243}b_{217} \oplus b_{247}b_{231} \oplus b_{213}b_{235} \oplus b_{255}b_{251} \oplus iv_{i+53}b_{239} \oplus iv_{i+46}k_{i+44} \oplus iv_{i+36}k_{i+29} \oplus b_{255}b_{247}b_{243}b_{213}iv_{i+53}iv_{i+46}$ $\%use: 99-106, 44-51, 29-36$

Function *main*():
- **for** $i \in [4, 7]$ **do**
  - $k_{i+113} \leftarrow k_{i+67}k_{i+90}k_{i+110}iv_{i+9} \oplus k_{i+42}iv_{i+9}$ $\%rec: 117-120\ use: 71-74, 94-98, 114-118, 46-49$
- **for** $i \in [4, 7]$ **do**
  - $k_{i+99} \leftarrow k_{i+46}iv_{i+13} \oplus k_{i+117} \oplus k_{i+39}k_{i+114}$ $\%rec: 103-106\ use: 50-53, 121-124, 43-46, 118-121$
- **for** $i \in [4, 7]$ **do**
  - $k_{i+103} \leftarrow k_{i+43}k_{i+118} \oplus k_{i+40}k_{i+98} \oplus k_{i+36}$ $\%rec: 107-110\ use: 47-50, 122-125, 44-47, 102-105, 40-43$
- **for** $i \in [7, 4]$ **do**
  - $b_{213}, b_{217} \leftarrow iv_{i+85}, iv_{89}$
  - $b_{231}, b_{235}, b_{239}, b_{243}, b_{247}, b_{251}, b_{255} \leftarrow 1, 1, 1, 1, 1, 1, 1$
  - **if** $i = 4$ **then** *update_bits*()
  - **if** $i > 4$ **then** $b_{213}, b_{217} \leftarrow iv'_{i+77}, iv'_{i+81}$
  - *update_o*(i)
  - **if** $i > 4$ **then** $o \leftarrow o \oplus iv_{i+56} \oplus k_{i+45}iv_{i+26}$ $\%use: 52-50$
  - **if** $i = 7$ **then** $o \leftarrow o \oplus k_{12}k_{121}$ $\%use: 12, 121$
  - $k_{i+80} \leftarrow iv'_{i+82} \oplus k_{i+3}k_{i+32}$ $\%rec: 87-84\ use: 10-7, 39-36$
  - **switch** i **do**
    - **case** ≤ 5 **do** $k_{i+80} \leftarrow k_{i+80} \oplus iv_{i+90} \oplus 1 \oplus o$
    - **case** 5 **do** $k_{i+80} \leftarrow k_{i+80} \oplus 1 \oplus o$
    - **otherwise do** $k_{i+80} \leftarrow k_{i+80} \oplus o$
  - $k_i \leftarrow k_{i+41}k_{i+70} \oplus k_{i+80} \oplus o$ $\%rec: 7-4\ use: 48-45, 77-74, 87-84$
- **for** $i \in [3, 0]$ **do**
  - $b_{213}, b_{217} \leftarrow iv_{i+85}, iv_{89}$
  - $b_{231}, b_{235}, b_{239}, b_{243}, b_{247}, b_{251}, b_{255} \leftarrow 1, 1, 1, 1, 1, 1, 1$
  - **if** $1 \leq i$ **then** *update_bits*()
  - *update_o*(i)
  - **if** $i < 2$ **then** $o \leftarrow o_i \oplus iv_{i+94}$
  - **else if** $i = 3$ **then** $o \leftarrow o_i \oplus 1$
  - $k_{i+80} \leftarrow iv'_{i+82} \oplus iv_{i+90} \oplus k_{i+8}k_{i+49} \oplus k_{i+3}k_{i+32} \oplus o$ $\%rec: 83-80\ use: 11-8, 52-49, 6-3, 35-32$
  - **if** $i \neq 0$ **then** $k_{i+12} \leftarrow 1 \oplus k_{i+46}k_{i+87} \oplus k_i \oplus k_{i+41}k_{i+70} \oplus k_{i+80} \oplus o$ $\%rec: 15-13\ use: 49-47, 90-88, 3-1, 44-42, 73-71, 83-80$

Clock	Cells
-1	0, 194, 198, 202, 206, 210, 214, 218, 232, 236, 240, 244, 248, 252
-2	0, 1, 194, 195, 198, 199, 202, 203, 206, 207, 210, 211, 214, 215, 218, 219, 232, 233, 236, 237, 240, 241, 244, 245, 248, 249, 252, 253
-3	$0-2$, $194-196$, $198-200$, $202-204$, $206-208$, $210-212$, $214-216$, $218-220$, $232-234$, $236-238$, $240-242$, $244-246$, $248-250$, $252-254$
-4	$0-3$, $194-221$, $232-255$
-5	$0-4$, $194-222$, $232-255$
-6	$0-5$, $194-223$, $232-255$
-7	$0-6$, $194-224$, $232-255$
-8	$0-7$, $194-225$, $232-255$

Table 10: Modified cells after running the KSA^{-1} routine

$$
\begin{aligned}
g_{210}^{-1}(X_{i-1}) &= iv_{i+81} \oplus k_{i+5}k_{i+63} \\
g_{206}^{-1}(X_{i-1}) &= iv_{i+77} \oplus k_{i+4}k_{i+79} \\
g_{202}^{-1}(X_{i-1}) &= iv_{i+73} \oplus k_{i+7}k_{i+102} \\
g_{198}^{-1}(X_{i-1}) &= iv_{i+69} \oplus k_{i+28}k_{i+51}k_{i+71}k_{i+98} \\
g_{194}^{-1}(X_{i-1}) &= iv_{i+65} \oplus k_{i+11}k_{i+120}
\end{aligned}
$$

where

$$
z'^{-1}(X_{i-1}) = k_{i+79} \oplus k_{i+98} \oplus iv_{i+8} \oplus iv_{i+58} \oplus iv_{i+52} \oplus iv_{i+45}k_{i+43} \oplus iv_{i+35}k_{i+28}
$$

$$
z^{-1}(X_{i-1}) = \begin{cases} z'^{-1}(X_{i-1}) \oplus iv_{i+88} \oplus iv_{i+84} \oplus iv_{i+84}iv_{i+52}iv_{i+45} & \text{if } i = 0 \\ z'^{-1}(X_{i-1}) \oplus iv_{i+88} \oplus iv_{i+84} \oplus iv_{i+84}iv_{i+52}iv_{i+45} & \text{if } i = -1 \\ z'^{-1}(X_{i-1}) \oplus iv_{i+88} \oplus iv_{i+84} \oplus iv_{i+93} \oplus 1 \oplus iv_{i+84}iv_{i+52}iv_{i+45} & \text{if } i = -2 \\ z'^{-1}(X_{i-1}) \oplus iv_{81} \oplus k_5k_{63} \oplus iv_{85} \oplus k_3k_{44} \oplus iv_{i+93} & \text{if } i = -3 \end{cases}
$$

From Espresso's reverse feedback functions we can see that the only functions containing retrievable key bits are $g_{252}^{-1}, g_{248}^{-1}, g_{244}^{-1}, g_{240}^{-1}$ and g_{218}^{-1}. Note that all positions, except 217, can be recovered from the padding (see Table 5). In the case of g_{218}^{-1}, the value can be recovered from IV''s bits. Therefore, we obtain Algorithm 12 for recovering some of K's bits.

In the case of our proposed version of Espresso, we can carry out a similar analysis. Again, for simplicity, we will first write some intermediary feedback function

$$
\begin{aligned}
g_0'^{-1}(X_{i-1}) &= 1 \oplus x_{i+40}x_{i+69} \\
g_{252}'^{-1}(X_{i-1}) &= 1 \oplus k_{i+41}k_{i+82} \oplus k_{i+7} \\
g_{248}'^{-1}(X_{i-1}) &= 1 \oplus k_{i+43}k_{i+101} \oplus k_{i+39} \\
g_{244}'^{-1}(X_{i-1}) &= 1 \oplus k_{i+42}k_{i+117} \oplus k_{i+102} \\
g_{240}'^{-1}(X_{i-1}) &= 1 \oplus k_{i+45}iv_{i+12} \oplus k_{i+116} \\
g_{236}'^{-1}(X_{i-1}) &= 1 \oplus k_{i+66}k_{i+89}k_{i+109}iv_{i+8} \\
g_{218}'^{-1}(X_{i-1}) &= \begin{cases} iv_{i+89} \oplus k_{i+2}k_{i+31} & \text{if } i \geq -2 \\ iv_{i+89} \oplus g_0^{-1}(X_{i+4})k_{i+31} & \text{otherwise} \end{cases} \\
g_{214}'^{-1}(X_{i-1}) &= \begin{cases} iv_{i+85} \oplus k_{i+3}k_{i+44} & \text{if } i \geq -3 \\ iv_{i+85} \oplus g_0^{-1}(X_{i+5})k_{i+44} & \text{otherwise} \end{cases}
\end{aligned}
$$

Algorithm 12: Key bits recovery algorithm for the `0xfffffffe` padding (backward construction)

Input: Chosen IV's IV and IV' and key bits k_j, where $j \in \{0-3, 25-31, 40-45, 63, 69, 76-82, 117\}$

Output: 24 key bits k_j, where $j \in \{4-7, 36-39, 95-102, 113-116, 124-127\}$

for $i \in [0, -2]$ **do**
- $k_{i+116} \leftarrow k_{i+45} iv_{i+12}$ $\%rec : 116-114\ use : 45-43$
- $k_{i+102} \leftarrow k_{i+42} k_{i+117}$ $\%rec : 102-100\ use : 42-40, 117-115$

$k_{113} \leftarrow k_{42} iv_9$ $\%rec : 113\ use : 42$

$k_{99} \leftarrow k_{43} k_{101} k_{114}$ $\%rec : 99\ use : 43, 101, 114$

for $i \in [0, -3]$ **do**
- $k_{i+127} \leftarrow iv'_{i+3}$ $\%rec : 127-124$
- $k_{i+7} \leftarrow k_{i+41} k_{i+82}$ $\%rec : 7-4\ use : 41-38, 82-79$
- **if** $i = 0$ **then** $k_{i+7} \leftarrow k_{i+7} \oplus 1$
- $k_{i+39} \leftarrow k_{i+43} k_{i+101}$ $\%rec : 39-36\ use : 43-40, 101-98$
- $o \leftarrow k_{i+79} \oplus iv_{i+8} \oplus iv_{i+58} \oplus iv_{i+52} \oplus iv_{i+45} k_{i+43} \oplus iv_{i+35} k_{i+28}$ $\%use : 79-76, 43-40, 28-25$
- **if** $i = 3$ **then** $o \leftarrow o \oplus iv_{81} \oplus k_5 k_{63} \oplus iv_{85} \oplus k_3 k_{44} \oplus iv_{90}$ $\%use : 5, 63, 3, 44$
- **else** $o \leftarrow o \oplus iv_{i+88} \oplus iv_{i+84} \oplus 1 \oplus iv_{i+84} iv_{i+52} iv_{i+45}$
- **if** $i = 0$ *or* $i = 1$ **then** $o \leftarrow o \oplus 1$
- **else** $o \leftarrow o \oplus iv_{i+93}$
- $k_{i+98} \leftarrow o \oplus iv'_{i+93} \oplus iv_{i+89} \oplus k_{i+2} k_{i+31}$ $\%rec : 98-95\ use : 2-0, 31-28$
- **if** $t = 0$ **then** $k_{-1} \leftarrow o \oplus k_{98} \oplus k_{40} \oplus k_{69}$ $\%use : 98, 40, 69$

$$g'^{-1}_{210}(X_{i-1}) = \begin{cases} iv_{i+81} \oplus k_{i+5} k_{i+63} & \text{if } i \geq -5 \\ iv_{i+81} \oplus g_0^{-1}(X_{i+7}) k_{i+63} & \text{otherwise} \end{cases}$$

$$g'^{-1}_{206}(X_{i-1}) = \begin{cases} iv_{i+77} \oplus k_{i+4} k_{i+79} & \text{if } i \geq -4 \\ iv_{i+77} \oplus g_0^{-1}(X_{i+6}) k_{i+79} & \text{otherwise} \end{cases}$$

$$g'^{-1}_{202}(X_{i-1}) = iv_{i+73} \oplus k_{i+7} k_{i+102}$$

$$g'^{-1}_{198}(X_{i-1}) = iv_{i+69} \oplus k_{i+28} k_{i+51} k_{i+71} k_{i+98}$$

Using these intermediary functions we obtain the following the feedback functions

$$g_0^{-1}(X_{i-1}) = \begin{cases} g'^{-1}_0(X_{i-1}) \oplus z^{-1}(X_{i-1}) & \text{if } i \geq -4 \\ g'^{-1}_0(X_{i-1}) \oplus k_{i+45} k_{i+86} \oplus k_{i+11} \oplus z^{-1}(X_{i-1}) & \text{otherwise} \end{cases}$$

$$g_{252}^{-1}(X_{i-1}) = \begin{cases} g'^{-1}_{252}(X_{i-1}) & \text{if } i \geq -4 \\ g'^{-1}_{252}(X_{i-1}) \oplus k_{i+47} k_{i+105} \oplus k_{i+43} & \text{otherwise} \end{cases}$$

$$g_{248}^{-1}(X_{i-1}) = \begin{cases} g'^{-1}_{248}(X_{i-1}) & \text{if } i \geq -4 \\ g'^{-1}_{248}(X_{i-1}) \oplus k_{i+46} k_{i+121} \oplus k_{i+106} & \text{otherwise} \end{cases}$$

$$g_{244}^{-1}(X_{i-1}) = \begin{cases} g'^{-1}_{244}(X_{i-1}) & \text{if } i \geq -4 \\ g'^{-1}_{244}(X_{i-1}) \oplus k_{i+49} iv_{i+16} \oplus k_{i+120} & \text{otherwise} \end{cases}$$

$$g_{240}^{-1}(X_{i-1}) = \begin{cases} g'^{-1}_{240}(X_{i-1}) & \text{if } i \geq -4 \\ g'^{-1}_{240}(X_{i-1}) \oplus k_{i+70} k_{i+93} k_{i+113} iv_{i+12} & \text{otherwise} \end{cases}$$

$$g_{236}^{-1}(X_{i-1}) = \begin{cases} g'^{-1}_{236}(X_{i-1}) & \text{if } i \geq -4 \\ g'^{-1}_{236}(X_{i-1}) \oplus k_{i+53} iv_{i+34} \oplus iv_{i+64} & \text{otherwise} \end{cases}$$

$$g_{232}^{-1}(X_{i-1}) = \begin{cases} k_{i+49} iv_{i+30} \oplus iv_{i+60} & \text{if } t \neq -7 \\ 1 \oplus k_{i+49} iv_{i+30} \oplus iv_{i+60} & \text{if } t = -7 \end{cases}$$

$$g_{218}^{-1}(X_{i-1}) = \begin{cases} g_{218}'^{-1}(X_{i-1}) \oplus z^{-1}(X_{i-1}) & \text{if } i \geq -4 \\ g_{218}'^{-1}(X_{i-1}) \oplus k_{i+7}k_{i+48} \oplus z^{-1}(X_{i-1}) & \text{otherwise} \end{cases}$$

$$g_{214}^{-1}(X_{i-1}) = \begin{cases} g_{214}'^{-1}(X_{i-1}) & \text{if } i \geq -4 \\ g_{214}'^{-1}(X_{i-1}) \oplus k_{i+9}k_{i+67} & \text{otherwise} \end{cases}$$

$$g_{210}^{-1}(X_{i-1}) = \begin{cases} g_{210}'^{-1}(X_{i-1}) & \text{if } i \geq -4 \\ g_{210}'^{-1}(X_{i-1}) \oplus k_{i+8}k_{i+83} & \text{otherwise} \end{cases}$$

$$g_{206}^{-1}(X_{i-1}) = \begin{cases} g_{206}'^{-1}(X_{i-1}) & \text{if } i \geq -4 \\ g_{206}'^{-1}(X_{i-1}) \oplus k_{i+11}k_{i+106} & \text{otherwise} \end{cases}$$

$$g_{202}^{-1}(X_{i-1}) = \begin{cases} g_{202}'^{-1}(X_{i-1}) & \text{if } i \geq -4 \\ g_{202}'^{-1}(X_{i-1}) \oplus k_{i+32}k_{i+55}k_{i+75}k_{i+102} & \text{otherwise} \end{cases}$$

$$g_{198}^{-1}(X_{i-1}) = \begin{cases} g_{198}'^{-1}(X_{i-1}) & \text{if } i \geq -4 \\ g_{198}'^{-1}(X_{i-1}) \oplus k_{i+15}k_{i+124} & \text{otherwise} \end{cases}$$

$$g_{194}^{-1}(X_{i-1}) = iv_{i+65} \oplus k_{i+11}k_{i+120}$$

where

$$\begin{aligned} z'^{-1}(X_{i-1}) = {} & k_{i+79} \oplus k_{i+98} \oplus iv_{i+8} \oplus b_{221} \oplus iv_{i+58} \oplus b_{242}b_{216} \oplus b_{246} \oplus b_{212}b_{234} \oplus b_{254}b_{250} \\ & \oplus iv_{i+52}b_{238} \oplus iv_{i+45}k_{i+43} \oplus iv_{i+35}k_{i+28} \oplus b_{254}b_{246}b_{242}b_{212}iv_{i+52}iv_{i+45} \end{aligned}$$

$$z^{-1}(X_{i-1}) = \begin{cases} z'^{-1}(X_{i-1}) & \text{if } i = -2 \\ z'^{-1}(X_{i-1}) \oplus 1 \oplus b_{246} & \text{if } i = -6 \\ z'^{-1}(X_{i-1}) \oplus 1 & \text{otherwise} \end{cases}$$

and

$$b_{254} = \begin{cases} 1 \oplus k_{i+44}k_{i+85} \oplus k_{i+10} & \text{if } -3 \geq i \geq -6 \\ 1 & \text{otherwise} \end{cases}$$

$$b_{250} = \begin{cases} 1 \oplus k_{i+46}k_{i+104} \oplus k_{i+42} & \text{if } -3 \geq i \geq -6 \\ 1 & \text{otherwise} \end{cases}$$

$$b_{246} = \begin{cases} 1 \oplus k_{i+45}k_{i+120} \oplus k_{i+105} & \text{if } -3 \geq i \geq -6 \\ 1 & \text{otherwise} \end{cases}$$

$$b_{242} = \begin{cases} 1 \oplus k_{i+48}iv_{i+15} \oplus k_{i+119} & \text{if } -3 \geq i \geq -6 \\ 1 & \text{otherwise} \end{cases}$$

$$b_{238} = \begin{cases} 1 \oplus k_{i+69}k_{i+92}k_{i+112}iv_{i+11} & \text{if } -3 \geq i \geq -6 \\ 1 & \text{otherwise} \end{cases}$$

$$b_{234} = \begin{cases} 1 \oplus k_{i+52}iv_{i+33} \oplus iv_{i+63} & \text{if } -3 \geq i \geq -6 \\ 1 & \text{otherwise} \end{cases}$$

$$b_{221} = \begin{cases} 1 & \text{if } i \geq -4 \\ 0 & \text{if } i = -5 \\ iv'_{i+101} & \text{otherwise} \end{cases}$$

$$b_{216} = \begin{cases} iv_{i+88} & \text{if } 0 \geq i > -3 \\ iv_{i+88} \oplus k_{i+6}k_{i+47} & \text{if } -3 \geq i \geq -6 \\ iv'_{i+96} & \text{if } i = -7 \end{cases}$$

$$b_{212} = \begin{cases} iv_{i+84} & \text{if } 0 \geq i > -3 \\ iv_{i+84} \oplus k_{i+8}k_{i+66} & \text{if } -3 \geq i \geq -6 \\ iv'_{i+92} & \text{if } i = -7 \end{cases}$$

We can see that the only feedback function that contain retrievable bits are $g_{252}^{-1}, g_{248}^{-1}, g_{244}^{-1}, g_{240}^{-1}$ and g_{218}^{-1}. When we tried to recover the key bits, we found some loops that prevented us from recovering three bits. More precisely, we found the following dependencies

$$k_{97} \hookleftarrow k_1 \hookleftarrow k_{35} \hookleftarrow k_{97} \quad \text{and} \quad k_{96} \hookleftarrow k_0 \hookleftarrow k_{34} \hookleftarrow k_{96} \quad \text{and} \quad k_{95} \hookleftarrow k_{116} \hookleftarrow k_{113} \hookleftarrow k_{95}.$$

Moreover, we note that the first 4 bits of IV' coincide with the last 4 bits of K. Again, we do not claim that our solution is optimal. Our solution is presented in Algorithm 13.

5.3 Mixed Construction

Once we have constructed two pairs of related IV's using the KSA and the KSA^{-1} routines, we can simply apply both the forward and the backward construction. Note that there might be better approaches when combining the forward and backward type constructions (*i.e.* constructions that recover different bits compared to ours).

In the classical case, we can recover 41 key bits. More precisely, the mixed construction takes as input the two pairs and the key bits k_j, where $j \in \{3-6, 25-35, 44-49, 63, 69-72, 76-79, 84-86, 98-102, 121\}$. Then it runs the forward construction and then it runs the backward one. Finally, the algorithm outputs k_j, where $j \in \{0-3, 7-11, 36-43, 80-83, 95-98, 103-106, 113-120, 124-127\}$.

Regarding our proposal, the mixed construction can recover 39 key bits. More precisely, the mixed construction takes as input the two pairs and the key bits k_j, where $j \in \{4-12, 21-31, 36-53, 60-82, 86-90, 95-97, 99-103, 106-108, 117\}$. Then it runs the backward construction and then it runs the forward one. Finally, the algorithm outputs k_j, where $j \in \{0-3, 13-15, 32-35, 83-85, 91-94, 98, 104, 105, 109-116, 118-127\}$.

Remark 2. Note that we also studied the backward and forward combination for the classic case. However, this combination performed poorer than the one we presented. Thus, we omitted it. The same happened for the forward and backward combination for our proposed padding scheme.

5.4 Complexity

To summarise, we provide in Table 11 the complexities of the key recovery attacks. We can see that when we take the attacks separately, the original padding has a better security margin. However, in the mixed case our proposal performs better.

Construction	Padding	
	`0xfffffffe`	`0x7fffffff`
Forward	$2^{104} + 2^{32}$	$2^{101} + 2^{40}$
Backward	$2^{104} + 2^{28}$	$2^{99} + 2^{34}$
Mixed	$2^{87} + 2^{32} + 2^{28}$	$2^{89} + 2^{40} + 2^{34}$

Table 11: Attack Complexity

Algorithm 13: Key bits recovery algorithm for the `0xefffffff` padding (backward construction)

Input: Chosen IV's IV and IV' and key bits k_j, where
$j \in \{4-7, 21-28, 31, 36-49, 60-68, 72-82, 86-89, 95-97, 99-103, 106-108, 117\}$
Output: 29 key bits k_j, where $j \in \{0-3, 32-35, 91-94, 98, 109-116, 120-127\}$

Function *update_bits*()**:**
- $b_{212} \leftarrow b_{212} \oplus k_{i+8}k_{i+66}$ $\%use: 5-2, 63-60$
- $b_{216} \leftarrow b_{216} \oplus k_{i+6}k_{i+47}$ $\%use: 3-0, 44-41$
- $b_{234} \leftarrow b_{234} \oplus k_{i+52}iv_{i+33} \oplus iv_{i+63}$ $\%use: 49-46$
- $b_{238} \leftarrow b_{238} \oplus k_{i+69}k_{i+92}k_{i+112}iv_{i+11}$ $\%use: 66-63, 89-86, 109-106$
- $b_{242} \leftarrow b_{242} \oplus k_{i+48}iv_{i+15} \oplus k_{i+119}$ $\%use: 45-42, 116-113$
- $b_{246} \leftarrow b_{246} \oplus k_{i+45}k_{i+120} \oplus k_{i+105}$ $\%use: 42-39, 117-114, 102-99$
- $b_{250} \leftarrow b_{250} \oplus k_{i+46}k_{i+104} \oplus k_{i+42}$ $\%use: 43-40, 101-98, 39-36$
- $b_{254} \leftarrow b_{254} \oplus k_{i+44}k_{i+85} \oplus k_{i+10}$ $\%use: 41-38, 82-79, 7-4$

Function *update_o*(i)**:**
- $o \leftarrow k_{i+79} \oplus iv_{i+8} \oplus b_{221} \oplus iv_{i+58} \oplus b_{242}b_{216} \oplus b_{246} \oplus b_{212}b_{234} \oplus b_{254}b_{250} \oplus iv_{i+52}b_{238} \oplus iv_{i+45}k_{i+43} \oplus iv_{i+35}k_{i+28} \oplus b_{254}b_{246}b_{242}b_{212}iv_{i+52}iv_{i+45}$ $\%use: 79-72, 43-36, 28-21$

Function *main*()**:**
- **for** $i \in [0, -7]$ **do**
  - $k_{i+127} \leftarrow iv'_{i+7}$ $\%rec: 127-120$
- $k_{109} \leftarrow k_{38}iv_5 \oplus k_{63}k_{86}k_{106}iv_5$ $\%rec: 109$ $use: 38, 63, 86, 106$
- **for** $i \in [-4, -6]$ **do**
  - $k_{i+116} \leftarrow k_{i+45}iv_{i+12} \oplus k_{i+70}k_{i+93}k_{i+113}iv_{i+12}$ $\%rec: 112-110$ $use: 41-39, 66-64, 89-87, 109-107$
- $k_2 \leftarrow k_{36}k_{77} \oplus k_{42}k_{100} \oplus k_{38}$ $\%rec: 2$ $use: 36, 77, 42, 100, 38$
- $b_{212}, b_{216} \leftarrow iv'_{84}, iv'_{88}$
- $b_{221}, b_{234}, b_{238}, b_{242}, b_{246}, b_{250}, b_{254} \leftarrow 1, 1, 1, 1, 1, 1, 1$
- *update_o*(0)
- $k_{98} \leftarrow o \oplus 1 \oplus iv'_{97} \oplus iv_{89} \oplus k_2k_{31}$ $\%rec: 98$ $use: 2, 31$
- $k_{35} \leftarrow k_{39}k_{97} \oplus k_{42}k_{117} \oplus k_{103}$ $\%rec: 35$ $use: 39, 97, 42, 117, 102$
- $k_{113} \leftarrow k_{35}k_{110} \oplus k_{95} \oplus k_{42}iv_9$ $\%rec: 113$ $use: 35, 110, 95, 42$
- **for** $i \in [-4, -6]$ **do**
  - $k_{i+120} \leftarrow k_{i+42}k_{i+117} \oplus k_{i+102} \oplus k_{i+49}iv_{i+16}$ $\%rec: 116-114$ $use: 38-36, 113-111, 98-96, 45-43$
- **for** $i \in [-5, -6]$ **do**
  - $k_{i+39} \leftarrow k_{i+43}k_{i+101} \oplus k_{i+46}k_{i+121} \oplus k_{i+106}$ $\%rec: 34, 33$ $use: 38, 37, 96, 95, 41, 40, 116, 115, 101, 100$
- **for** $i \in [-4, -7]$ **do**
  - **if** $i \neq 5$ **then** $k_{i+7} \leftarrow k_{i+41}k_{i+82} \oplus k_{i+47}k_{i+105} \oplus k_{i+43}$ $\%rec: 3, 1, 0$ $use: 37, 35, 34, 78, 76, 75, 43, 41, 40, 101, 99, 98, 39, 37, 36$
- **for** $i \in [-1, -7]$ **do**
  - $b_{212}, b_{216} \leftarrow iv_{i+84}, iv_{i+88}$
  - $b_{234}, b_{238}, b_{242}, b_{246}, b_{250}, b_{254} \leftarrow 1, 1, 1, 1, 1, 1$
  - **if** $-3 \geq i \geq -6$ **then** *update_bits*()
  - **if** $i = -7$ **then** $b_{212}, b_{216} \leftarrow iv'_{i+92}, iv'_{i+96}$
  - **switch** i **do**
    - **case** ≤ 4 **do** $b_{221} = 1$
    - **case** 5 **do** $b_{221} = 0$
    - **otherwise do** $b_{221} = iv'_{i+101}$
  - *update_o*(i)
  - **if** $i \neq -2$ **then** $o \leftarrow o \oplus 1$
  - **if** $i = -6$ **then** $o \leftarrow o \oplus b_{246}$
  - **if** $i \leq -4$ **then** $k_{i+98} \leftarrow o \oplus iv'_{i+97} \oplus iv_{i+89} \oplus k_{i+2}k_{i+31} \oplus k_{i+7}k_{i+48}$ $\%rec: 94-91$ $use: 27-24, 3-0, 44-41$
  - **if** $i \geq -4$ **then** $k_{i-1} \leftarrow 1 \oplus k_{i+40}k_{i+69} \oplus k_{i+98} \oplus o$ $\%use: 39-36, 68-65, 97-94$
- $k_{32} \leftarrow k_{36}k_{94} \oplus k_{39}k_{114} \oplus k_{99}$ $\%rec: 32$ $use: 36, 94, 39, 114, 99$

6 Conclusions

In this paper, we have shown that given any Key-IV pair, one can easily construct another pair, with expected 2^{32} time complexity, that produces the same bits as the initial keystream on a significant amount of positions.

Furthermore, we have studied related Key-IV pairs that produce shifted keystreams. We have shown how one can obtain two related Key-IV pairs, in expected 2^{28} trials, such that the pairs generate 4-bit shifted keystreams. To increase the complexity of these attacks, we have proposed a new padding scheme and have proven that the complexity increases to 2^{34}.

Additionally, we managed to describe several attacks that recover some of the key bits and requires only two/four related IV's. Hence, we can decrease the complexity of conducting a brute force attack on the key to 2^{87} in the classical case and to 2^{89} for our proposal.

References

1. NIST SP 800-22: Download Documentation and Software. https://csrc.nist.gov/Projects/Random-Bit-Generation/Documentation-and-Software
2. NIST SP 800-90B: Entropy Assessment. https://github.com/usnistgov/SP800-90B_EntropyAssessment
3. The GNU Multiple Precision Arithmetic Library. https://gmplib.org/
4. Banik, S., Maitra, S., Sarkar, S.: Some Results on Related Key-IV Pairs of Grain. In: SPACE 2012. Lecture Notes in Computer Science, vol. 7644, pp. 94–110. Springer (2012)
5. Banik, S., Maitra, S., Sarkar, S., Meltem Sönmez, T.: A Chosen IV Related Key Attack on Grain-128a. In: ACISP 2013. Lecture Notes in Computer Science, vol. 7959, pp. 13–26. Springer (2013)
6. Cannière, C., Küçük, Ö., Preneel, B.: Analysis of Grain's Initialization Algorithm. In: AFRICACRYPT 2008. Lecture Notes in Computer Science, vol. 5023, pp. 276–289. Springer (2008)
7. Ding, L., Guan, J.: Related Key Chosen IV Attack on Grain-128a Stream Cipher. IEEE Trans. Inf. Forensics Secur. **8**(5), 803–809 (2013)
8. Dubrova, E., Hell, M.: Espresso: A Stream Cipher for 5G Wireless Communication Systems. Cryptography and Communications **9**(2), 273–289 (2017)
9. Küçük, Ö.: Slide Resynchronization Attack on the Initialization of Grain 1.0. Tech. rep. (2006)
10. Maimuţ, D., Teşeleanu, G.: New Configurations of Grain Ciphers: Security Against Slide Attacks. In: SECITC 2021. Lecture Notes in Computer Science, vol. 13195, pp. 260–285. Springer (2021)
11. Olsson, M., Cavdar, C., Frenger, P.K., Tombaz, S., Sabella, D., Jäntti, R.: 5GrEEn: Towards Green 5G mobile networks. In: WiMob 2013. pp. 212–216. IEEE Computer Society (2013)
12. Wang, M.X., Dai Lin, D.: Related Key Chosen IV Attack on Stream Cipher Espresso Variant. In: CSE 2017. vol. 1, pp. 580–587. IEEE Computer Society (2017)
13. Yao, G., Parampalli, U.: Generalized NLFSR Transformation Algorithms and Cryptanalysis of the Class of Espresso-like Stream Ciphers. CoRR **abs/1911.01002** (2019)