Stream Cipher Operation Modes with Improved Security against Generic Collision Attacks

Matthias Hamann and Matthias Krause

University of Mannheim, Germany
{hamann,krause}@uni-mannheim.de
http://th.informatik.uni-mannheim.de/

Abstract. Most stream ciphers used in practice are vulnerable against generic collision attacks, which allow to compute the secret initial state on the basis of $\mathcal{O}(2^{n/2})$ keystream bits in time and space $\mathcal{O}(2^{n/2})$, where n denotes the inner state length of the underlying keystream generator. This implies the well-known rule that for reaching n-bit security, the inner state length should be at least $2n$. Corresponding to this, the inner state length of recent proposals for practically used stream ciphers is quite large (e.g., $n = 288$ for Trivium and $n = 160$ for Grain v1).
In this paper, we suggest a simple stream cipher operation mode, respectively a simple way how to modify existing operation modes like that in the Bluetooth system, which provides provable security near $2^{2n/3}$ against generic collision attacks. Our suggestion refers to stream ciphers (like E_0 in Bluetooth) which generate keystreams that are partitioned into packets and where the initial states for each packet are computed from a packet-IV and the secret session key using a resynchronization algorithm.
Our security analysis is based on modeling the resynchronization algorithm in terms of the $FP(1)$-construction $E(x, k) = F(P(x \oplus k) \oplus k)$, where k denotes an n-bit secret key (corresponding to the symmetric session key), F denotes a publicly known n-bit function (corresponding to the output function of the underlying keystream generator), P denotes a publicly known n-bit permutation (corresponding to the iterated state update function of the generator), and the input x is a public initial value. Our security bounds follow from the results presented in [12], where a tight $\frac{2}{3}n$ security bound for the $FP(1)$-construction in the random oracle model was proved.

Keywords: Stream Cipher Operation Modes, Time-Memory-Data Tradeoff Attacks, Provable Security, Even-Mansour Constructions

1 Introduction

During the last decades, many stream ciphers for practical use have been suggested and many different techniques for cryptanalyzing them have been developed (correlation attacks, fast correlation attacks, guess-and-verify attacks, BDD-attacks, time-memory-data tradeoff attacks etc.). The typical aim of these attacks is to gain some nontrivial information about the inner state of the underlying keystream generator (KSG) from a given piece of keystream.

While most attacks try to exploit structural weaknesses specific to the respective ciphers (e.g., of their KSG state transition and output function), generic collision attacks represent an inherent weakness of almost all existing practical stream ciphers. More precisely, time-memory-data tradeoff attacks like those of *Babbage* [2] and *Biryukov* and *Shamir* [5] allow to compute the secret initial state on the basis of $\mathcal{O}(2^{n/2})$ keystream bits in time and space $\mathcal{O}(2^{n/2})$, where n denotes the inner state length of the used KSG. This yields the well-known rule that for achieving n-bit security by a stream cipher (in standard operation mode), one has to choose an inner state length of at least $2n$ for the underlying KSG, which has the implication that modern practical stream ciphers have comparatively large inner state lengths (e.g., 288 bit for Trivium [7] or 160 bit for Grain v1 [11]).

Based on the results in [12], our research shows a way to reduce the size of the inner state by using operation modes which yield provable security beyond the birthday bound against time-memory-data tradeoff attacks. The stream cipher operation modes presented in this paper refer to the *packet mode*, used, e.g., with the E_0 cipher in the Bluetooth system [15], where the keystream is generated packet-wise, i.e., as a sequence of packets of moderate packet length R (e.g., at most 2790 bits for the basic rate of Bluetooth, cf. [15]), and the initial state for each keystream packet i is computed from a secret session key k and a public initial value IV^i by using a state initialization algorithm. The packet mode can be seen as a compromise between the *one-stream mode* (used, e.g., by Trivium and Grain v1), which has minimal initialization effort per keystream bit but also minimal security w.r.t. generic time-memory-data tradeoff attacks, and block cipher-based modes (used, e.g., by the GSM standard A5/3), which have maximal security w.r.t. time-memory-data treadeoff attacks but also maximal initialization effort per keystream bit.

Our security results for packet modes are based on modeling the process of state initialization and keystream generation through FP-constructions, which were first introduced in [12]. This is done by identifying n with the inner state length of the KSG. The function P corresponds to stepping the KSG a certain number of times without producing output, an operation which is used in most state initialization algorithms of practical stream ciphers and can often be described by an efficiently invertible permutation. F corresponds to the function which assigns to each $x \in \{0,1\}^n$ the block $F(x)$ of the first n keystream bits generated on the initial inner state x. Standard security assumptions on keystream generators imply that F must be a cryptographically strong one-way function.

A first consequence of this approach is that the initialization and keystream generation algorithm used in the Bluetooth cipher E_0 corresponds to the $F(0)$-construction $F(x \oplus k)$ and allows to recover the secret session key on the basis of $2^{n/2}$ packets. However, a slight change in this algorithm, namely adding the session key twice in the initialization phase, raises the security w.r.t. to time-memory-data tradeoff attacks from $n/2$ to $2/3 \cdot n$, where n denotes the KSG's inner state length (which is 132 bit in the case of E_0).

A state initialization algorithm corresponding directly to the $FP(1)$-construction $F(P(x \oplus k) \oplus k)$ refers to a KSG with non-linear state update function and would execute the following three steps: Given a secret key $k \in \{0,1\}^n$ and an initial value $IV^i \in \{0,1\}^n$, load $IV^i \oplus k$ into the n inner state register cells, then iterate the KSG a certain number of times without producing output, then add k bitwise to the resulting inner state and start to produce the i-th keystream packet. The block of the first n keystream bits then corresponds to $E(IV^i, k) := F(P(IV^i \oplus k) \oplus k)$.

The paper is structured as follows. In section 2, we provide an overview of the basic properties of stream ciphers and introduce the terminology used subsequently. In particular, we describe the typical components of stream ciphers employed in practice and distinguish between two different operation modes. In section 3, we then go into details about the inner workings of some prominent stream ciphers, with a focus on their respective state initialization algorithms and how they are operated. Following this, in section 4, we introduce a way of modeling the state initialization and keystream generation of stream ciphers and exemplify this model using the ciphers described in section 3. The resulting insights will finally allow us in section 5 to compare the security (w.r.t. generic collision attacks) of existing approaches for operating stream ciphers to the security of our new design paradigm.

2 Operation Modes of Stream Ciphers

Stream ciphers are symmetric encryption algorithms intended for encrypting, in an online manner, plaintext bit streams X, which have to pass an insecure channel. The encryption is done by adding bitwise a keystream S which is generated in dependence of a secret session key k. The legal recipient, who also knows k, decrypts the encrypted bitstream $Y = X \oplus S$ by generating S and computing $X = Y \oplus S$.

Typically, a stream cipher is defined by the following three components:

– A session key generation protocol, which defines how Alice and Bob generate a common symmetric session key k.
– A keystream generator (KSG), which generates a keystream $S(q_1)$ in dependence of an initial state q_1.
– A state initialization algorithm *StateInit*, which defines how to compute the initial KSG-state $q_1 = \mathit{StateInit}(IV, k)$ for the KSG from a (public) initial value IV and the secret session key k.

In our context, we focus on the KSG and the state initialization algorithm. KSGs are clockwise working devices which can be formally specified by finite automata, defined by an inner state length n, the set of inner states $\{0,1\}^n$, a state update function $\delta : \{0,1\}^n \longrightarrow \{0,1\}^n$ and an output function $\mathit{out} : \{0,1\}^n \longrightarrow \{0,1\}^*$. Starting from an initial state q_1, in each clock cycle $i \geq 1$, the KSG produces a piece of keystream $z_i = \mathit{out}(q_i)$ (as a rule, one bit long) and changes the inner state according to $q_{i+1} = \delta(q_i)$. The output bitstream $S(q_1)$ is defined by concatenating all the outputs $z_1 z_2 z_3 \cdots$.

Typically, the KSG is not only used for the keystream generation but also for performing the state initialization. In section 3, the specifications of the *StateInit*-algorithms and the KSGs for various practically used stream ciphers like Trivium, Grain v1, the Bluetooth cipher E_0 etc. are listed.

Among stream cipher operation modes, we distinguish *one-stream modes* and *packet modes*. In a one-stream mode, the whole plaintext stream of the session is encrypted by a sufficiently long prefix of the keystream $S(q_1)$, whereas in a packet mode, the plaintext X is partitioned into packets $X = X^1 X^2 X^3 \cdots$ of a moderate maximum packet length R (e.g., at most 2790 bits for the basic rate of Bluetooth, cf. [15]), and separate pieces of keystream $S^i = S(q_1^i)$ are produced for each packet. Here, in the packet mode, the initial packet states q_1^i will be computed via

$$q_1^i = \mathit{StateInit}(IV^i, k),$$

in dependence of the secret session key k and a separate public initial value IV^i for each packet.

3 Some Stream Cipher Examples

This section provides an overview of some prominent stream ciphers, with a focus on their respective state initialization algorithms and operation modes. It includes E_0 (used in Bluetooth) and A5/1 (used in GSM) as well as the eSTREAM [14] hardware portfolio members Trivium and Grain v1. In section 4, we will see how the state initialization algorithms and the keystream generation of these stream ciphers can be modeled in relation to the FP-construction. Please observe in the following that E_0 in Bluetooth and $A5/1$ in GSM are both operated in packet modes in the sense that each packet in a session is encrypted under the same session key but using different, publicly known IVs as described in section 2.

3.1 E_0 (used in Bluetooth)

The classical way of ensuring data confidentiality for Bluetooth connections between a master device A and a slave device B utilizes stream cipher-based encryption on a per-packet basis. More precisely, when encryption is activated, the payload of each packet (at most 2790 bits for the so-called basic rate [15]) is encrypted under a separate payload key using the algorithm E_0, which produces a keystream of appropriate length that is XORed to the plaintext in a bitwise fashion. By *payload* *key*, we refer to the initial state of the KSG just before the first keystream bit for the packet is produced (i.e., in terms of this paper, to the output of the state initialization algorithm *StateInit*). It is derived on the basis of the current 128-bit encryption key K_c, the 48-bit Bluetooth address ADR of the master, and 26 bits $CL_0, \ldots, CL_{25}$ of the master's clock (to which both devices are synchronized) at the time of the first transmission slot of the packet.[1] Before we continue with the respective details, please note that the encryption key K_c corresponds to a session key, i.e., K_c is constant for a potentially very large number of successive packets. Consequently, the difference between the payload keys of different packets during such a session solely arises from the difference of the corresponding, publicly known clock values at the time of encryption.

In the following, we will describe how the individual payload key for a packet is generated by E_0 based on the inputs K_c, ADR, and $CL_0, \ldots, CL_{25}$.

The general structure of the encryption engine of E_0 is depicted in figure 1.

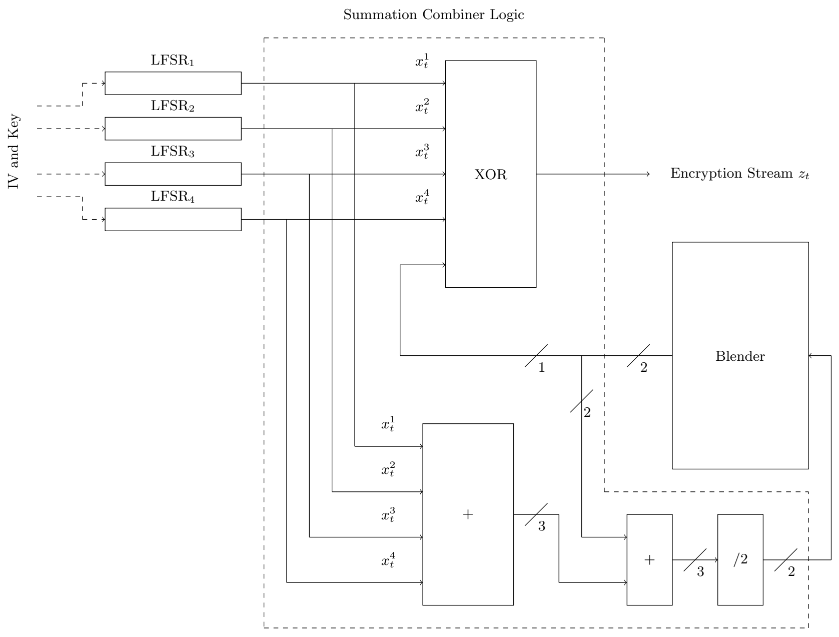

Fig. 1. E_0 encrpytion engine. $x_t^1, \ldots, x_t^4$ denote the output bits of the four LFSRs, respectively, in step t. (cf. [15], p. 1323)

The internal state of the four *linear feedback shift registers* (LFSRs) has a total size of 128 bits and the respective feedback polynomials are all primitive (cf. [15], p. 1322). Figure 2 shows the positions of the feedback taps. For further details like the exact definition of the *blender* finite state machine (4 bits), we refer the reader to the official Bluetooth specification ([15], p. 1322–1324). In our context, it is sufficient to understand that the inputs K_c, ADR, and $CL_0, \ldots, CL_{25}$ are split up and, together with the two 3-bit constants 111 and 001, arranged as depicted in figure 2 prior to payload key generation, which then works as follows (cf. [15], p. 1325–1326):

[1] Unlike the symmetric encryption key K_c, the values ADR and $CL_0, \ldots, CL_{25}$ are considered to be public. For more details on the generation of K_c please see appendix A.

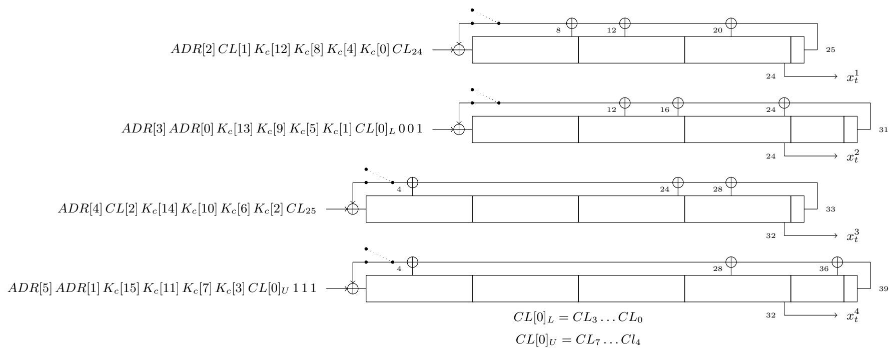
$ADR[2]\, CL[1]\, K_c[12]\, K_c[8]\, K_c[4]\, K_c[0]\, CL_{24}$
8
12
20
25
24
x_t^1
$ADR[3]\, ADR[0]\, K_c[13]\, K_c[9]\, K_c[5]\, K_c[1]\, CL[0]_L\, 0\,0\,1$
12
16
24
31
24
x_t^2
$ADR[4]\, CL[2]\, K_c[14]\, K_c[10]\, K_c[6]\, K_c[2]\, CL_{25}$
4
24
28
33
32
x_t^3
$ADR[5]\, ADR[1]\, K_c[15]\, K_c[11]\, K_c[7]\, K_c[3]\, CL[0]_U\, 1\,1\,1$
4
28
36
39
32
x_t^4
$CL[0]_L = CL_3 \ldots CL_0$
$CL[0]_U = CL_7 \ldots Cl_4$

Fig. 2. Initializing the E_0 LFSRs. $ADR[i]$, $K_c[i]$, and $CL[i]$ denote the bytes of ADR, K_c, and CL, respectively. (cf. [15], p. 1327)

1. Disable the feedback switches of the four LFSRs and set their contents to zero. (time $t = 0$)
2. Shift K_c, ADR, $CL_0, \ldots, CL_{25}$, and the bits 111 and 001 into the registers of the four LFSRs as depicted in figure 2 (e.g., at $t = 1$, CL_{24} is shifted into LFSR$_1$, 1 into LFSR$_2$, CL_{25} into LFSR$_3$ and 1 into LFSR$_4$). For each LFSR, once the first input bit has reached the rightmost cell, close the feedback switch.
3. When the switch of the last LFSR (i.e., LFSR$_4$) is closed ($t = 39$), reset the blender registers (4 bits) to zero.
4. Shift in the rest of the input bits while starting to produce output bits which are **not** used as keystream.
5. Once all all input bits have been shifted in (at this point, LFSR$_1$ has been clocked 30 times with closed switches, LFSR$_2$ 24 times, LFSR$_3$ 22 times, LFSR$_4$ 16 times), keep on clocking until 200 bits in total (including those in the previous step) have been produced. ($t = 239$)

The last 128 bits which were generated are then loaded in parallel (i.e., copied) into the registers of the four LFSRs as their new initial state at $t = 240$ (for details, again, see [15]). The values of the blender registers are kept. At this point, the assignments to the registers of the four LFSRs as well as to the registers of the blender represent what was previously referred to as the payload key (132 bits in total). Based on this initial state, the encryption engine is then further clocked and its output bits serve as keystream bits for the encryption of the current payload.

If another packet needs to be encrypted, the whole initialization process of the cipher is repeated with a new clock value as the input of the E_0 algorithm, leading to a different payload key.

3.2 A5/1 (used in GSM)

Though considered outdated for security reasons (see, e.g., [3] or the time-memory-data tradeoff attack using the FPGA cluster COPACOBANA in [10]), the stream cipher A5/1 is still widely used to encrypt GSM communication. Upon connection establishment, the mobile device is authenticated and a 64-bit symmetric session key K_c is generated using one of the COMP128 algorithms, which take a publicly known 128-bit challenge (i.e., a random number produced by the authentication center of the network) and a common secret 128-bit key (stored on the SIM card and known to the authentication center) as inputs. Based on this 64-bit session key K_c and an additional IV, data frames of 114 downlink bits and 114 uplink bits are protected using the A5/1 algorithm, whose basic structure is depicted in figure 3.

The LFSRs in A5/1 are either clocked all three in parallel or in so-called *majority mode*, where, at each clock cycle, the majority function over the three majority bits in figure 3 is computed and

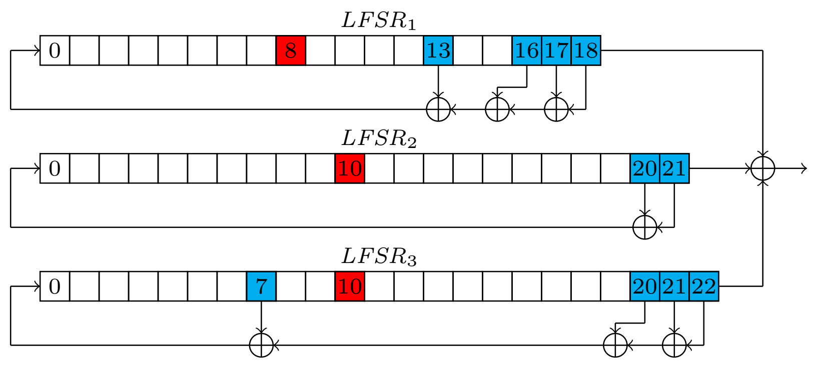

Fig. 3. Structure of A5/1, which consists of three LFSRs of total state length 64 bit. The blue register cells denote the feedback taps of the LFSRs, the red cells denote their majority bits.

each LFSR is clocked if and only if its majority bit equals the output of the majority function. Please note that the feedback polynomials of the three LFSRs are all primitive (cf. [4]). The corresponding feedback taps are depicted in figure 3.

Let $K_c := (K_{63}, \ldots, K_0)$ denote the secret session key and let $F := (F_{21}, \ldots, F_0)$ denote the 22-bit representation of the publicly known frame number. The 228 keystream bits, which will be XORed to the corresponding 114 downlink bits and 114 uplink bits, respectively, are generated as follows (cf. [4]).

1. Set all register cells of the three LFSRs to 0.
2. For $i = 0$ to 63 do
   2.1 $\text{LFSR}_1[0] = \text{LFSR}_1[0] + K_i$
   $\text{LFSR}_2[0] = \text{LFSR}_2[0] + K_i$
   $\text{LFSR}_3[0] = \text{LFSR}_3[0] + K_i$
   2.2 Clock all three LFSRs (i.e., no majority clocking).
3. For $i = 0$ to 21 do
   3.1 $\text{LFSR}_1[0] = \text{LFSR}_1[0] + F_i$
   $\text{LFSR}_2[0] = \text{LFSR}_2[0] + F_i$
   $\text{LFSR}_3[0] = \text{LFSR}_3[0] + F_i$
   3.2 Clock all three LFSRs (i.e., no majority clocking).
4. Clock the KSG 100 times using majority clocking and discard the output bits, which are each computed as the XOR of the three bits $\text{LFSR}_1[18]$, $\text{LFSR}_2[21]$, and $\text{LFSR}_3[22]$ as depicted in figure 3.
5. Generate 114 keystream bits for downlink encryption clocking the KSG 114 times (using majority clocking).
6. Generate 114 keystream bits for uplink encryption clocking the KSG 114 times (using majority clocking).

For the enrcyption of the next frame, the above process is repeated with the same session key K_c but a different frame number F.

Please note that the state of the LFSRs after step 4 (i.e., right before the first keystream bit is produced) corresponds to what was called the payload key in the case of Bluetooth (cf. section 3.1) or, in terms of the abstract model introduced in section 2, to the output of the *StateInit*-algorithm.

3.3 Trivium

Trivium [7] is a synchronous stream cipher and one of the three members of the eSTREAM hardware portfolio. It is designed to produce a keystream of up to 2^{64} bits based on an 80-bit symmetric key

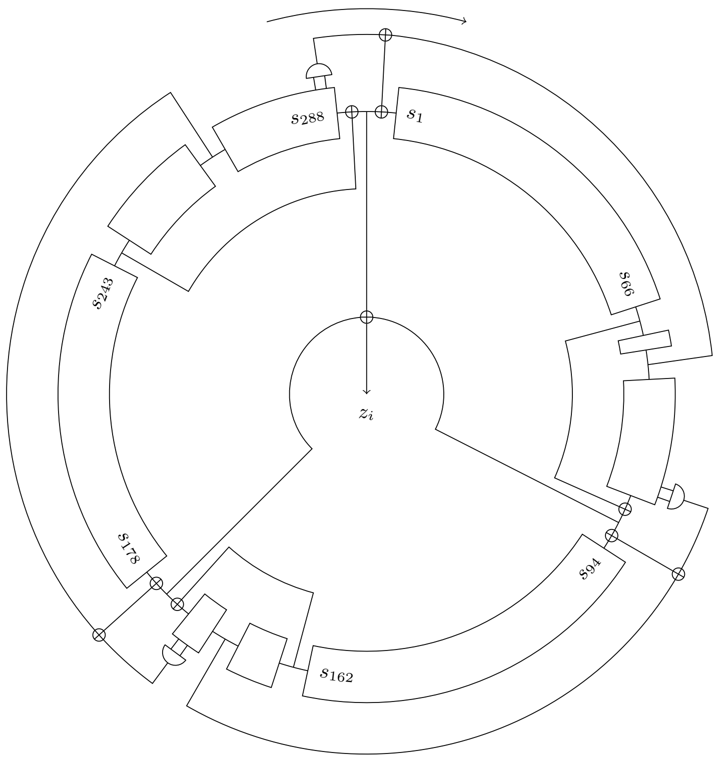

Fig. 4. Structure of Trivium (total state length 288 bit). z_i denotes the output bit in step i. (cf. [7])

and a 64-bit IV as inputs. Its inner structure consists of three shift registers of total length 288 bit, which are interwoven non-linearly as depicted in figure 4. Each keystream bit is computed as the XOR of six bits from the current internal state, two from each shift register.

Before the first keystream bit is output, the following initialization procedure is performed based on an 80-bit key $K := (K_{80}, \ldots, K_1)$ and an 80-bit initial value $IV := (IV_{80}, \ldots, IV_1)$ (cf. [7]):

```
1. $(s_1, s_2, \ldots, s_{93}) \leftarrow (K_{80}, \ldots, K_1, 0, \ldots, 0)$
   $(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (IV_{80}, \ldots, IV_1, 0, \ldots, 0)$
   $(s_{178}, s_{179}, \ldots, s_{288}) \leftarrow (0, \ldots, 0, 1, 1, 1)$
2. For $i = 1$ to $4 \cdot 288$ do
   2.1 $t_1 \leftarrow s_{66} + s_{91} \cdot s_{92} + s_{93} + s_{171}$
       $t_2 \leftarrow s_{162} + s_{175} \cdot s_{176} + s_{177} + s_{264}$
       $t_3 \leftarrow s_{243} + s_{286} \cdot s_{287} + s_{288} + s_{69}$
   2.2 $(s_1, s_2, \ldots, s_{93}) \leftarrow (t_3, s_1, \ldots, s_{92})$
       $(s_{94}, s_{95}, \ldots, s_{177}) \leftarrow (t_1, s_{94}, \ldots, s_{176})$
       $(s_{178}, s_{179}, \ldots, s_{288}) \leftarrow (t_2, s_{178}, \ldots, s_{287})$
```

In terms of our model (cf. section 2), the 288-bit state $s_1, \ldots, s_{288}$ of the three shift registers right after step 2 has been completed corresponds to the output of the *StateInit*-algorithm.

Based on this internal state, the KSG is then clocked using the same operations as in step 2 of the initialization procedure with the only difference that in each clock cycle, the keystream bit $s_{66} + s_{93} + s_{162} + s_{177} + s_{243} + s_{288}$ is output in order to encrypt one bit of plaintext via XORing.

Please observe that the state update of Trivium is reversible (see also [7]), as this will be of importance when we model the state initialization of Trivium in terms of our FP-construction in section 4.

3.4 Grain v1

In the following, we will describe the 80-bit version of the stream cipher Grain v1 [11], which, like Trivium, is a member of the eSTREAM hardware portfolio. It takes an 80-bit key and a 64-bit IV as inputs and is composed of two shift registers, one an 80-bit *non-linear feedback shift register* (NFSR) and one an 80-bit *linear feedback shift register* (LFSR), which are connected as depicted in figure 5.

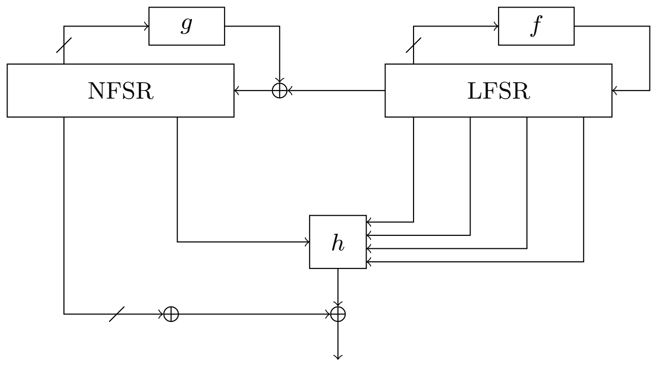

Fig. 5. Structure of Grain v1 (keystream generation phase). f denotes the linear feedback function of the LFSR, g denotes the non-linear feedback function of the NFSR. (cf. [11])

Let us denote the 80-bit state of the NFSR by $(n_{80}, \ldots, n_1)$ and the 80-bit state of the LFSR by $(l_{80}, \ldots, l_1)$. Then the state update of the NFSR is defined by $n'_i := n_{i-1}$, $i = \{80, \ldots, 2\}$, and

$$\begin{aligned}
n'_1 = {} & n_{59}n_{52}n_{47}n_{43}n_{35}n_{28} \\
& + n_{71}n_{65}n_{59}n_{52}n_{47} + n_{43}n_{35}n_{28}n_{20}n_{17} \\
& + n_{65}n_{59}n_{20}n_{17} + n_{47}n_{43}n_{28}n_{20} + n_{71}n_{52}n_{35}n_{17} \\
& + n_{59}n_{52}n_{47} + n_{35}n_{28}n_{20} \\
& + n_{71}n_{65} + n_{47}n_{43} + n_{20}n_{17} \\
& + n_{80} + n_{71} + n_{66} + n_{59} + n_{52} + n_{47} + n_{43} + n_{35} + n_{28} + n_{20} + n_{18} \\
& + l_{80},
\end{aligned}$$

where $(n'_{80}, \ldots, n'_1)$ denotes the new state of the NFSR.

Analogously, the state update of the LFSR, whose feedback polynomial is primitive, can be defined by $l'_i := l_{i-1}$, $i = \{80, \ldots, 2\}$, and

$$l'_1 := l_{80} + l_{67} + l_{57} + l_{42} + l_{29} + l_{18},$$

where $(l'_{80}, \ldots, l'_1)$ denotes the new state of the LFSR.

The result of the non-linear function h, which takes $l_{77}, l_{55}, l_{34}, l_{16}$, and n_{17} as inputs, is XORed with the NFSR bits $n_{79}, n_{78}, n_{76}, n_{70}, n_{49}, n_{37}, n_{24}$ to produce the output of the cipher. For the exact specification of h, we refer the reader to, e.g., [11]. In the context of this paper, it is sufficient to observe that neither l_{80} nor n_{80} are among the inputs of h, as this implies that the state update is not only reversible during the keystream generation phase[2] depicted in figure 5, but also during the initialization phase, which is depicted in figure 6 and will now be described in further detail.

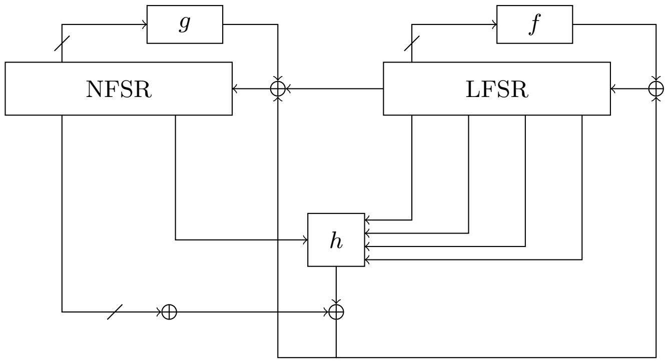

Fig. 6. Structure of Grain v1 (initialization phase). (cf. [11])

At the beginning of the initialization phase, the 80-bit key is parallel loaded (i.e., copied) to the 80-bit NFSR register and the 64-bit IV is parallel loaded to the first 64 register cells of the LFSR (for an exact specification of indices, again, see [11]). The 16 remaining LFSR register cells are filled with ones. After this, the cipher is clocked 160 times but instead of producing keystream, the output is XORed to the state update for the register cells n_1 and l_1, as depicted in figure 6.

Here, the state of the two shift registers (160 bit in total) after the initialization phase corresponds to the output of the *StateInit*-algorithm in our model (cf. section 2).

3.5 Block Cipher-based Constructions

Before we move on to section 4, where we will see how the state initialization algorithms and the keystream generation of the stream ciphers described in subsections 3.1–3.4 can be modeled in relation to the FP-construction, we will briefly discuss another way how to realize stream ciphers, which might seem rather different at first but, in fact, is also connected to our model.

Instead of designing a KSG from scratch, stream ciphers can also be built on the basis of block ciphers, e.g., by using them in counter mode as depicted in figure 7.

Notably, "one of the requirements imposed on all eSTREAM stream cipher submissions was that they should demonstrate the potential to be superior to the AES in at least one significant aspect" [13] and the respective testing framework included AES in counter mode.

Please observe that using a block cipher in counter mode to generate a keystream as shown in figure 7 can also be interpreted as a kind of packet mode, similar to what we described for E_0 and A5/1. In this case, however, the packet size is not determined by the underlying technology (i.e., the maximum payload size for Bluetooth packets or the fixed frame size in case of GSM) but by the block size of the used block cipher instead. Concretely, the concatenation $IV||Ctr_i$ used as the input for the block cipher in figure 7 can be interpreted as a publicly known initial value IV^i on the basis of which the keystream $Keystream_i$ (i.e., S^i in terms of section 2) for the packet i, whose size equals the block length of the underlying cipher, is generated.

[2] The LFSR state update is trivially reversible and the NFSR update function contains n_{80} only as a linear term.

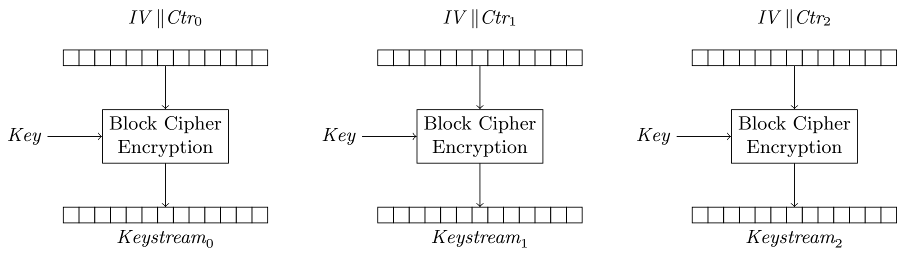

Fig. 7. Example of block cipher-based keystream generation using counter mode. $IV||Ctr_i$ denotes the concatenation of the initial value IV and the counter value Ctr_i in step i. $Keystream_i$ denotes the keystream fragment produced in step i.

4 Modeling the State Initialization and Keystream Generation of Stream Ciphers

In this section, we are now going to introduce a way of modeling the state initialization and keystream generation of stream ciphers and exemplify this model using the ciphers described in section 3. The resulting insights will then allow us in section 5 to compare the security of existing approaches for operating stream ciphers to the security of our new design paradigm.

During the last decades, many KSGs for practical use have been suggested and many different techniques for cryptanalyzing stream ciphers have been developed (correlation attacks, fast correlation attacks, guess-and-verify attacks, BDD-attacks, time-memory-data tradeoff attacks etc.). The typical aim of these attacks is to gain some nontrivial information about the underlying inner state of the KSG from a given piece of keystream. To achieve this goal, most attacks exploit structural weaknesses of the KSG state transition and output function.

In this work, we discuss the security of stream ciphers with respect to generic collision attacks. More precisely, we address time-memory-data tradeoff attacks, which were first described in papers of *Babbage* [2]: Let $q_1 \in \{0,1\}^n$ be the n-bit inner state of the keystream generator right before the first output bit is produced (i.e., q_1 is the result of the *StateInit*-procedure as described in section 2) and let $F : \{0,1\}^n \rightarrow \{0,1\}^n$ denote a public, efficiently computable function, where $F(q)$ is defined as the first n keystream bits generated on some inner state $q \in \{0,1\}^n$. Suppose further that Eve knows a prefix of length $D + n - 1$ of the full keystream $S(q_1)$. Then, with high probability, after sampling $2^n/D$ times a random state $q \in \{0,1\}^n$ and computing $F(q)$, Eve finds some inner state q^* such that $F(q^*)$ is a substring of $S(q_1)$, which allows to compute q_1 from q^* if the state transition is efficiently invertible (as, e.g., in the case of Trivium and Grain v1). This yields a time-memory-data tradeoff of $\mathcal{O}(2^{n/2})$. The vulnerability of stream ciphers against this simple type of attack, which even works if the KSG is ideally designed, is one main reason for the rule that an inner state length of $2n$ has to be invested for reaching n bit security. Many practical stream cipher designs like Trivium (288-bit internal state) or Grain v1 (160-bit internal state) were influenced by this principle, as we have already seen in section 3.

Note that the main security requirement for keystream generators is that it should be hard to distinguish a bitstream $S(q_1)$, generated by the KSG on a secret initial state $q_1 \in \{0,1\}^n$, from a truly random bitstream. Clearly, this implies that it should be impossible to compute q_1 from a prefix of length n of $S(q_1)$, i.e., F should be a cryptographically hard one-way function.

Having defined the initial state $q_1 \in \{0,1\}^n$ as the output of the *StateInit*-procedure in section 2, we saw in section 3 that a typical subroutine of *StateInit* is to clock the respective KSG a certain number of times without producing output. In practical stream ciphers, the state transition function δ is often bijective (see, e.g., the corresponding remarks for Trivium and Grain v1 in sections 3.3 and 3.4, respectively) and efficiently invertible in the sense that it is possible to efficiently compute preimages. In the following, we will identify this operation by an efficiently invertible, public function $P : \{0,1\}^n \longrightarrow \{0,1\}^n$, which, as in the case of Trivium and Grain v1, is often a permutation or, as in the case of the Bluetooth cipher E_0, very "close"[3] to being one.

We now present some examples how the state initialization algorithms and the keystream generation of the practical stream ciphers described in section 3 can be modeled by using functions F and P as describe above. Given some IV x and a secret session key k, $q_1(x,k)$ denotes the initial state produced by the *StateInit*-procedure of the respective stream cipher and $E(x,k)$ denotes the first n keystream bits generated (i.e., $E(x,k) = F(q_1(x,k))$.[4]

E_0 in Bluetooth (cf. sec. 3.1): The way x (80 bit) and k (128 bit) are loaded into the four LFSRs at the beginning of the initialization procedure is completely $GF(2)$-linear. Hence, the combined state of the four LFSRs right after x and k have been shifted in completely (i.e., at $t = 55$) can be described as $f(x) \oplus g(k)$, where $f : \{0,1\}^{80} \longrightarrow \{0,1\}^{128}$ and $g : \{0,1\}^{128} \longrightarrow \{0,1\}^{128}$ are $GF(2)$-linear functions. Note, however, that, at this point in time, 32 bits of the secret session key k have been shifted into $LFSR_1$ (and nowhere else), which is only 25 bits wide (see figure 2 in section 3.1). Consequently, a time-memory-data tradeoff attack recovering $f(x) \oplus g(k)$ (see section 5.2) will "only" reveal a linear combination with reduced information about k, which may induce further effort, e.g., by requiring to search for the right 4-bit state of the blender engine, when trying to decrypt other packets encrypted under the same session key. The 4-bit state $h(x,k)$ of the blender registers at $t = 55$ is determined by a non-linear function $h : \{0,1\}^{80} \times \{0,1\}^{128} \longrightarrow \{0,1\}^4$ depending on the 80-bit IV x and the 128-bit key k.

Based on $f(x)$, $g(k)$, and $h(x,k)$, we can now denote the full inner state of the Bluetooth cipher E_0 at $t = 55$ as $((f(x) \oplus g(k)) \,||\, h(x,k))$. As pointed out previously, the state transition of E_0 can be described by an efficiently invertible and "nearly" bijective function. Hence, we will describe the phase between $t = 56$ and $t = 111$, during which, as part of the state initialization, the cipher is clocked and its output is discarded, using an efficiently invertible and "nearly" bijective function $P : \{0,1\}^{132} \longrightarrow \{0,1\}^{132}$. A distinctive property of the E_0 cipher in Bluetooth is that, as the final step of the state initialization algorithm, 128 output bits are produced (between $t = 112$ and $t = 239$) which are not used as keystream (i.e., they are kept internal) but instead, at $t = 240$, are parallel loaded to the registers of the four LFSRs. We will describe this step between $t = 112$ and $t = 240$ using the function $F' : \{0,1\}^{132} \longrightarrow \{0,1\}^{132}$, where, given a 132-bit state $y \in \{0,1\}^{132}$ at $t = 111$, $F'(y)$ denotes the concatenation of the first 128 bits of $F(y)$ and the 4-bit state $h'(y)$ of the blender engine at $t = 239$.[5] Subsequently, after $t = 240$, the keystream for the corresponding Bluetooth packet (of length at most 2790 bits) is produced based on the initial state $q_1(x,k) = F'(P(((f(x) \oplus g(k)) \,||\, h(x,k))))$.

In summary, the way the Bluetooth cipher E_0 generates the first 132 keystream bits in dependence of the secret session key k and the initial value x can be modeled as

$$E(x,k) = F\left(F'\left(P\left(((f(x) \oplus g(k)) \,||\, h(x,k))\right)\right)\right).$$

A5/1 in GSM (cf. sec. 3.2): The way the secret session key k and the IV x are loaded to the LFSRs of the A5/1 engine is rather similar to the Bluetooth cipher E_0. A major difference, however, is that, during the first 64 clock cycles of the *StateInit*-procedure, each of the 64 key bits is introduced in a $GF(2)$-linear way to each of the three LFSRs of total size 64 bit. The way we

[3] In E_0, the state update of the four LFSRs (128 bit) is fully bijective as they are completely independent of each other and the blender engine. Only the state update of the blender registers (4 bit) possibly isn't.

[4] For the sake of simplicity, we denote any combination of publicly known nonces and constants used as inputs for the state initialization by a single IV x.

[5] The 4-bit state of the blender engine doesn't change during the parallel loading of the first 128 bits of $F(y)$ to the LFSRs at $t = 240$.

model A5/1 in terms of F and P is based on the following two "key ideas" from [6] by *Biryukov*, *Shamir*, and *Wagner*:

- "Key idea 3: A5/1 can be efficiently inverted",
- "Key idea 4: The key can be extracted from the initial state of any frame"[6].

As in the case of the Bluetooth cipher E_0, the way x (22 bit) and k (64 bit) are loaded into the three LFSRs at the beginning of the initialization procedure is completely $GF(2)$-linear. Hence, the combined state of the LFSRs right after x and k have been shifted in completely (i.e., at the end of step 3 of the algorithm in section 3.2) can be described as $f(x) \oplus g(k)$, where $f : \{0,1\}^{22} \longrightarrow \{0,1\}^{64}$ and $g : \{0,1\}^{64} \longrightarrow \{0,1\}^{64}$ are $GF(2)$-linear functions. Due to "key idea 4" of *Biryukov*, *Shamir*, and *Wagner*, the subsequent stepping of the KSG for 100 times using majority clocking (step 4 of the algorithm in section 3.2) can be described by an efficiently invertible function $P : \{0,1\}^{64} \longrightarrow \{0,1\}^{64}$. Finally, based on the initial state $q_1(x,k) = P(f(x) \oplus g(k))$ after step 4, the 228 keystream bits for the corresponding frame are then produced. Hence, the way A5/1 generates the first 64 keystream bits in dependence of the secret session key k and the initial value x can be modeled as

$$E(x,k) = F\left(P\left(f(x) \oplus g(k)\right)\right).$$

Trivium (cf. sec. 3.3): For Trivium, the way the IV x (208 bits including constants) and the secret key k (80 bits) are introduced to the state registers of total length 288 bits of the KSG is even simpler than in the case of E_0 or A5/1 as the respective values are simply parallel loaded (see step 1 of the state initialization algorithm in section 3.3). As pointed out previously, the subsequent clocking of the Trivium engine without producing output (step 2 of the state initialization algorithm) is easily reversible. Thus, we can describe this step (comprising $4 \cdot 288$ clock cycles) by an efficiently invertible permutation $P : \{0,1\}^{288} \longrightarrow \{0,1\}^{288}$. Based on the resulting initial state $q_1(x,k) = P(k||x)$, the first 288 keystream bits produced in dependence of the secret session key k and the initial value x can then be modeled as

$$E(x,k) = F\left(P\left(k||x\right)\right).$$

Grain v1 (cf. sec. 3.4): The state initialization of Grain v1 works similar to the one of Trivium. Once the secret key k (80 bits) and the IV x (80 bits including constants) have been parallel loaded to the registers of the NFSR and the LFSR, respectively, the cipher is clocked 160 times without producing keystream. Even though the output of the non-linear function h is fed back to both shift registers during this phase, we have seen in section 3.4 that the state update of Grain v1 is still straightforwardly reversible. In terms of our model, we can describe it using an efficiently invertible permutation $P : \{0,1\}^{160} \longrightarrow \{0,1\}^{160}$. Just like in the case of Trivium, the resulting initial state can be modeled as $q_1(x,k) = P(k||x)$. Consequently,

$$E(x,k) = F\left(P\left(k||x\right)\right)$$

describes how the first 160 keystream bits are produced in dependence of the secret session key k and the initial value x.

[6] Note that *Biryukov*, *Shamir*, and *Wagner* refer to the state of the LFSRs right after the frame counter has been introduced (i.e., after step 3 of the algorithm in section 3.2) as the "initial state" whereas we denote by *initial state* the output of the *StateInit*-procedure (i.e., the state at the end of step 4 of the algorithm in section 3.2). This, however, does not harm our argumentation as "key idea 4" can be combined with "key idea 3".

5 Analyzing and Improving the Security of Stream Ciphers against Time-Memory-Data Tradeoff Attacks

In section 4, we have seen how to model the state initialization and keystream generation of prominent practical stream ciphers using the functions F and P. In this section, we are now going to describe the general concept of *FP-constructions* along with sharp security bounds for two special instances, namingly, the $F(0)$-construction and the $FP(1)$-construction (subsection 5.1). Moreover, we will show how the stream ciphers modeled in section 4 fit into this concept and how this allows to straightforwardly derive information about their security against time-memory-data tradeoff attacks (subsection 5.2). Finally, we will introduce the $FP(1)$*-mode* as a new design principle for stream ciphers, which provably allows to raise the security against time-memory-data tradeoff attacks from $n/2$ to $2/3 \cdot n$.

5.1 FP-Constructions and their Security Bounds

The concept of FP-constructions was introduced in [12]. The underlying motivation was to extend the focus of key alternating ciphers (like Even-Mansour ciphers [8]) through studying similar generic constructions for pseudorandom functions (PRFs), where, besides public permutations, a publicly known one-way function $F : \{0,1\}^n \longrightarrow \{0,1\}^n$ is allowed as a possible additional component.

Let $x \in \{0,1\}^n$ denote some publicly known value and let $k \in \{0,1\}^n$ be the secret n-bit key. The simplest FP-construction is the $F(0)$-construction $F(x \oplus k)$, for which a sharp $n/2$-security bound in the so-called *random oracle model* was already shown in [9] (see also [12] for a complete proof). The main result in [12] was a sharp $2/3 \cdot n$-security bound for the $FP(1)$-construction $F(P(x \oplus k) \oplus k)$ depicted in figure 8, where P denotes a public permutation over $\{0,1\}^n$.

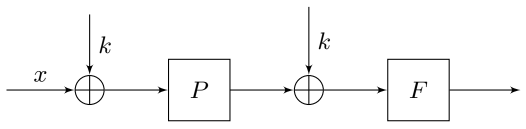

Fig. 8. The $FP(1)$-construction. (cf. [12])

In the random oracle model, the adversary tries to recover the secret key k on the basis of oracle queries to a P/P^{-1}-oracle, to an F-oracle and to an E-oracle, where queries to the E-oracle will be answered according to $F(P(x \oplus k) \oplus k)$. A sharp $2/3n$-security bound means that, on the one hand, there is a successful key recovery attack of running time $\mathcal{O}(2^{2/3 \cdot n})$ using $\mathcal{O}(2^{2/3 \cdot n})$ oracle queries, and that, on the other hand, for all $\alpha < 2/3$ any attack making only $\mathcal{O}(2^{\alpha \cdot n})$ oracle queries has an only exponentially small (in n) success probability to recover the secret key. Note that this adversary model corresponds exactly to the generic time-memory-data tradeoff attacks against stream ciphers in [2] and [5].

5.2 Stream Ciphers corresponding to the $F(0)$-Construction

We will now see that the practical stream cipher examples described in section 3 and modeled in terms of F and P in section 4 all provide only $n/2$-security against time-memory-data tradeoff attacks. Please note that none of the results in this subsection are surprising or new. Instead, they are meant to exemplify the way FP-constructions can serve to analyze the security of stream ciphers against generic collision attacks and, more importantly, also provide hints on how to modify the respective ciphers and operation modes in order to achieve $2/3 \cdot n$-security as will be described in subsection 5.3.

***E_0* in Bluetooth (cf. sec. 3.1 and sec. 4):** In section 4, we have seen that the way the Bluetooth cipher E_0 generates the first 132 keystream bits in dependence of the 128-bit secret session key k and the 80-bit initial value x can be modeled as

$$E(x,k) = F\left(F'\left(P\left(((f(x) \oplus g(k))\,||\,h(x,k))\right)\right)\right),$$

where f and g are $GF(2)$-linear functions and h is non-linear.

Note that, in our model for E_0, P was only "nearly" a bijection and, more importantly, the *StateInit*-procedure was not efficently invertible as it contained the one-way function F'. In consequence, a time-memory-data tradeoff attack (of complexity 2^{66}) recovering the initial state

$$q_1(x,k) = F'\left(P\left(((f(x) \oplus g(k))\,||\,h(x,k))\right)\right),$$

as described in section 4, would not yield the secret session key k but instead only allow the attacker to decrypt the single packet whose IV was x.

For a successful attack against all the other packets, one has to recover the secret session key k. In order to do so, we define the one-way function $\tilde{F} := F \circ F' \circ P$ for E_0 and consider a slightly different attack scenario, i.e., we study the complexity of a **session key recovery attack in dependence of the number of known (or partially known) keystream packets**. It is easy to see that, now, the 132-bit input $q_0 := (f(x) \oplus g(k))\,||\,h(x,k)$ of $\tilde{F}$ can be recovered using a time-memory-data tradeoff attack with time and memory complexity 2^{66} on the basis of 2^{66} keystream packets. Due to the non-linearly derived 4-bit suffix $h(x,k)$ of the intermediary state q_0 (at $t = 55$) and the fact that $g(k)$ will only reveal a linear combination in k (cf. section 4), the Bluetooth cipher E_0 doesn't perfectly (but very closely) map to the $F(0)$-construction. However, the design principle suggested by the $FP(1)$-mode in section 5.3 will still apply.

A5/1 in GSM (cf. sec. 3.2 and sec. 4): We have seen in section 4 that the way A5/1 generates the first 64 keystream bits in dependence of the secret session key k and the initial value x can be modeled as

$$E(x,k) = F\left(P\left(f(x) \oplus g(k)\right)\right).$$

Here, as in the case of E_0, P is efficiently invertible but not a bijection. However, in contrast to E_0, the *StateInit*-procedure of A5/1 does not contain any one-way components. Thus, recovering some state $\delta^j\left(q_1(x,k)\right)$, where δ denotes the state transition function and $q_1(x,k) = P(f(x) \oplus g(k))$ is the initial state of the corresponding packet, using a time-memory-data tradeoff attack with time and space complexity 2^{32} on the basis of about 2^{32} keystream bits, as described in section 4, will reveal $q_1(x,k)$ and, hence, $g(k)$ to an attacker.[7] This, in turn, will allow him to decrypt all packets which were encrypted under the session key k. (For improved time-memory-data tradeoff attacks on A5/1 see, e.g., [6].)

Trivium (cf. sec. 3.3 and sec. 4): Trivium, which is operated in a one-stream mode, maps perfectly to the FP-construction, i.e., the way the first 288 keystream bits are produced based on the 80-bit secret session key k and the 208-bit (including constants) initial value x can be described as

$$E(x,k) = F\left(P\left(k||x\right)\right),$$

where F denotes a public one-way function and P is a public, efficiently invertible permutation (see section 4). Using the conventional time-memory-data tradeoff attack with time and memory

[7] We suppose that the attacker sees at least 64 subsequent keystream bits per packet. Moreover, it is obvious that the attack on A5/1 would also work for $\tilde{F} := F \circ P$ when given 2^{32} keystream packets, i.e., in a similar $F(0)$-like scenario as described for the Bluetooth cipher E_0.

complexity 2^{144} on the basis of 2^{144} keystream bits, some internal state $\delta^j\left(q_1(x,k)\right)$ can be recovered with high probability and the corresponding initial state $q_1(x,k) = P\left(k||x\right)$ can be computed as the state transition function δ is efficiently invertible (see section 3.3). Clocking back the cipher even further, $k||x$ and, hence, the secret key k is revealed. Again, as for A5/1, Trivium could also be modeled in terms of the $F(0)$-construction by setting $\tilde{F} := F \circ P$ and supposing $\tilde{F}$ to be a one-way function. A corresponding time-memory-data tradeoff attack with time and space complexity 2^{144} would require the cipher to be operated in packet mode with 2^{144} packets available to the attacker.

Grain v1 (cf. sec. 3.4 and sec. 4): In section 4, we have seen that Grain v1 can be modeled in exactly the same way as Trivium. Consequently, the same arguments apply with respect to time-memory-data tradeoff attacks when the cipher is used in one-stream or in packet mode (attack complexity 2^{80}).

5.3 The $FP(1)$-Mode for Stream Cipher-based Encryption

A more promising approach is to use a state initialization algorithm which mimics the $FP(1)$-construction. Given a secret session key $k \in \{0,1\}^n$, we use the term $FP(1)$-*mode* to refer to the following generic encryption scheme, which operates on packets of polynomially bounded length R. For each packet, do:

– Fix a public initial value $x \in \{0,1\}^n$.
– Load $x \oplus k$ into the inner state registers.
– Run the KSG (which is supposed to have a bijective, efficiently invertible but nonlinear state update function) for a certain number of clock cycles without producing output. (We denote this step by $P : \{0,1\}^n \longrightarrow \{0,1\}^n$.)
– Add k to the resulting inner state $P(x \oplus k)$.
– Generate the keystream on the resulting inner state $P(x \oplus k) \oplus k$, which implies that the block of the first n keystream bits equals $F(P(x \oplus k) \oplus k)$.

For a convincing application of the previously described, sharp $2/3 \cdot n$ security bound for the $FP(1)$-construction to this kind of stream cipher operation mode, one has to strengthen the original lower bound argument by considering stronger adversaries. This is because, for efficiency reasons, the packet length R should be allowed to be larger than the inner state length n. In consequence, with each packet, the adversary gets not only $F(P(x \oplus k) \oplus k)$ (which corresponds to the first n bits of the keystream packet) but even all values $F(\delta^j(P(x \oplus k) \oplus k))$ for $j = 0, \cdots, R - n$, where δ denotes the KSG state update function and $F(\delta^j(P(x \oplus k) \oplus k))$ corresponds to the keystream bits $j+1, \cdots, j+n$ of the packet.

In [12], *Krause* shows that if the packet length R is polynomially bounded in n, the lower bound for the $FP(1)$-construction holds even in an extended random oracle model, where the adversary is allowed to ask an additional kind of oracle queries, which cover his potential knowledge of the complete packet keystream. Consequently, the $FP(1)$-mode is provably $2/3 \cdot n$-secure w.r.t. time-memory-data tradeoff attacks.

Recalling that the Bluetooth cipher E_0 also works in packet mode, naturally the question arises whether the ideas underlying the $FP(1)$-mode could be used to increase the security of E_0. And, in fact, adding the key in a bitwise fashion after the parallel loading of the value of F' (i.e., right after $t = 240$) to the registers of the four LFSRs would raise the security from $n/2$ to about $2/3 \cdot n$.

As a final remark, we would like to stress that XORing the same key k to the same registers at two different points in the course of an algorithm can be realized rather efficiently in hardware, as many components like multiplexors (in the case of serialized key addition) or an array of XOR gates (in the case of parallel key addition) can simply be reused. The remaining small area overhead induced by the second key addition is easily compensated for by the saving in registers (e.g., an internal state of length 120 bit would already provide 80-bit security against time-memory-data tradeoff attacks), which are a major cost driver when designing cryptographic protocols for ultra-constrained application-specific integrated circuits (ASICs) as described, e.g., in [1]. Consequently, our construction principle seems to be particularly interesting for lightweight applications.

6 Conclusion

In this work, we presented a simple stream cipher operation mode, respectively a simple way how to modify existing operation modes (like that of E_0 in the Bluetooth system), which provides provable security near $2^{2n/3}$ against generic collision attacks, where n denotes the size of the internal state of the underlying KSG. Our suggestion refers to stream ciphers with packet-wise keystream generation, where, for each packet of polynomially bounded length R, the initial state is computed from an individual packet-IV and the secret session key using a resynchronization algorithm.

For the security analysis, we made use of the concept of FP-constructions, which was introduced in [12]. Our results were twofold: On the one hand, we saw for several prominent stream cipher examples that each of them can be modeled in terms of the $F(0)$-construction and, hence, succumbs to time-memory-data tradeoff attacks of complexity $\mathcal{O}(2^{n/2})$. On the other hand, the tight $\frac{2}{3}n$-security bound for the $FP(1)$-construction $F(P(x \oplus k) \oplus k)$ in the (extended) random oracle model proved in [12] allowed us to straightforwardly conclude the $\frac{2}{3}n$-security of our $FP(1)$-mode against such generic collision attacks.

A natural topic for future research is the development of a concrete instantiation of a stream cipher using the $FP(1)$-mode. An interesting question closely related to this would be how to safely shrink the inner state of existing stream ciphers like Trivium or Grain v1 based on the $FP(1)$-mode without making them vulnerable to other types of attacks.

References

1. Frederik Armknecht, Matthias Hamann, and Vasily Mikhalev. Lightweight authentication protocols on ultra-constrained RFIDs - myths and facts. In Nitesh Saxena and Ahmad-Reza Sadeghi, editors, *Radio Frequency Identification: Security and Privacy Issues*, volume 8651 of *Lecture Notes in Computer Science*, pages 1–18. Springer International Publishing, 2014.
2. S.H. Babbage. Improved "exhaustive search" attacks on stream ciphers. In *Security and Detection, 1995., European Convention on*, pages 161–166, May 1995.
3. Elad Barkan, Eli Biham, and Nathan Keller. Instant ciphertext-only cryptanalysis of GSM encrypted communication. In Dan Boneh, editor, *Advances in Cryptology - CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 600–616. Springer Berlin Heidelberg, 2003.
4. Eli Biham and Orr Dunkelman. Cryptanalysis of the A5/1 GSM stream cipher. In Bimal Roy and Eiji Okamoto, editors, *Progress in Cryptology —INDOCRYPT 2000*, volume 1977 of *Lecture Notes in Computer Science*, pages 43–51. Springer Berlin Heidelberg, 2000.
5. Alex Biryukov and Adi Shamir. Cryptanalytic time/memory/data tradeoffs for stream ciphers. In Tatsuaki Okamoto, editor, *Advances in Cryptology — ASIACRYPT 2000*, volume 1976 of *Lecture Notes in Computer Science*, pages 1–13. Springer Berlin Heidelberg, 2000.
6. Alex Biryukov, Adi Shamir, and David Wagner. Real time cryptanalysis of A5/1 on a PC. In Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, and Bruce Schneier, editors, *Fast Software Encryption*, volume 1978 of *Lecture Notes in Computer Science*, pages 1–18. Springer Berlin Heidelberg, 2001.
7. Christophe De Cannière and Bart Preneel. Trivium. In Matthew Robshaw and Olivier Billet, editors, *New Stream Cipher Designs*, volume 4986 of *Lecture Notes in Computer Science*, pages 244–266. Springer Berlin Heidelberg, 2008.
8. Shimon Even and Yishay Mansour. A construction of a cipher from a single pseudorandom permutation. In Hideki Imai, Ronald L. Rivest, and Tsutomu Matsumoto, editors, *Advances in Cryptology — ASIACRYPT '91*, volume 739 of *Lecture Notes in Computer Science*, pages 210–224. Springer Berlin Heidelberg, 1993.
9. Peter Gazi and Stefano Tessaro. Secret-key cryptography from ideal primitives: A systematic overview. In *Information Theory Workshop (ITW), 2015 IEEE*, pages 1–5, April 2015.

10. Tim Güneysu, Timo Kasper, Martin Novotny, and Christof Paar. Cryptanalysis with COPACOBANA. *IEEE TRANSACTIONS ON COMPUTERS*, 57(11):1498–1513, 2008.
11. Martin Hell, Thomas Johansson, and Willi Meier. Grain: a stream cipher for constrained environments. *IJWMC*, 2(1):86–93, 2007.
12. Matthias Krause. Analyzing constructions for key-alternating pseudorandom functions with applications to stream cipher operation modes. Cryptology ePrint Archive, Report 2015/636, 2015. `http://eprint.iacr.org/`.
13. ECRYPT European Network of Excellence for Cryptology. eSTREAM optimized code howto, 2005. `http://www.ecrypt.eu.org/stream/perf/`.
14. ECRYPT European Network of Excellence for Cryptology. eSTREAM: the ECRYPT stream cipher project, 2008. `http://www.ecrypt.eu.org/stream/`.
15. Bluetooth SIG. Bluetooth core specification 4.2, 2014. `https://www.bluetooth.org/DocMan/handlers/DownloadDoc.ashx?doc_id=286439`.

Appendix

A Bluetooth: Generating the Encryption Key K_c

When two Bluetooth devices connect for the first time (pairing), an initialization key K_{init} is computed based on a common PIN, the Bluetooth address of one of the devices, and some 128-bit random number using the algorithm E_{22}. Subsequently, K_{init} is used together with two random 128-bit values and the 48-bit Bluetooth addresses of both devices to create the the so-called link key (or combination key) K_{ab} using the algorithm E_{21}. This link key is sometimes also referred to as the authentication key, as it is used together with some (changing) 128-bit random number during the subsequent authentications of the respective devices using the algorithm E_1. As a result of such an authentication during connection establishment, a 32-bit authentication token *SRES* and a 96-bit value called *ACO* (Authenticated Ciphering Offset) are produced. If authentication has been successful (i.e., the authentication tokens match), a new encryption key K_c will be generated each time encryption is enabled by the communicating parties.[8] The generation of K_c is performed using the algorithm E_3, which is based on a hash function with the following inputs: the link key K_{ab}, a 128-bit random number *EN_RAND*, and the 96-bit *ACO* generated during authentication. To account for legal export restrictions, the Bluetooth specification provides means of shortening the effective key length of K_c to less than 128 bits. The resulting keys are often referred to as K'_c. However, as the absolute key length will still be 128 bits and as we are aiming for the highest possible security here, we omit this shortening step for the sake of clarity and simply talk of the encryption key K_c when describing packet encryption using the algorithm E_0 in section 3.1.

[8] According to the official Bluetooth specification ([15], p. 1308), when using E_0 encryption, "the encryption keys shall be refreshed by the Link Manager at least once every 2^{28} Bluetooth Clocks (about 23.3 hours)".