Extended BDD-Based Cryptanalysis of Keystream Generators

Dirk Stegemann

Theoretical Computer Science, University of Mannheim, Germany

Abstract. The main application of stream ciphers is online-encryption of arbitrarily long data. Many practically used and intensively discussed stream ciphers consist of a small number of linear feedback shift registers (LFSRs) and a compression function that transforms the bitstreams produced by the LFSRs into the output keystream. In 2002, Krause proposed a Binary Decision Diagram (BDD) based attack on this type of ciphers, which ranges among the best generic short-keystream attacks on practically used ciphers such as the A5/1 generator used in GSM and the E_0 generator from the Bluetooth standard. In this paper we show how to extend the BDD-technique to nonlinear feedback shift registers (NFSRs), feedback shift registers with carry (FCSRs), and arbitrary compression functions. We apply our findings to the eSTREAM focus ciphers Trivium, Grain and F-FCSR. In the case of Grain, we obtain the first nontrivial cryptanalytic result besides generic time-memory-data tradeoffs.

Keywords: Stream cipher, cryptanalysis, BDD, Trivium, Grain, F-FCSR.

1 Introduction

The main purpose of LFSR-based keystream generators is online encryption of bitstreams $p \in \{0,1\}^*$ that have to be sent over an insecure channel, e.g., for encrypting speech data to be transmitted from and to a mobile phone over the air interface. In many stream ciphers, the output keystream $z \in \{0,1\}^*$ of the generator is bitwise XORed to the plaintext stream p in order to obtain the ciphertext stream $c \in \{0,1\}^*$, i.e., $c_i = p_i \oplus z_i$ for all i. Based on a secret key, which has to be exchanged between the sender and the authorized receiver prior to the transmission, and a public initialization vector (IV), the receiver can compute the keystream z in the same way as the sender computed it and decrypt the message using the above rule.

We consider the special type of FSR-based keystream generators that consist of an internal bitstream generator with a small number of Feedback Shift Registers (FSRs) and a compression function $\mathcal{C} : \{0,1\}^* \to \{0,1\}^*$. The secret key and the IV determine the initial state of the FSRs, which produce an internal bitstream $w \in \{0,1\}^*$ that is transformed into the output keystream z via

C. Adams, A. Miri, and M. Wiener (Eds.): SAC 2007, LNCS 4876, pp. 17–35, 2007.

$z = \mathcal{C}(w)$. Practical examples for this design include the E_0 generator used in Bluetooth [4], the A5/1 generator from the GSM standard for mobile telephones [5], and the self-shrinking generator [18].

In 2002, Krause proposed a Binary Decision Diagram (BDD) attack [14,15] on stream ciphers that are based on Linear Feedback Shift Registers (LFSRs). The BDD-attack is a generic attack in the sense that it does not depend on specific design properties of the respective cipher. It only relies on the assumptions that the generator's output behaves pseudorandomly and that the test whether a given internal bitstream w produces a sample keystream can be represented in a Free Binary Decision Diagram (FBDD) of size polynomial in the length of w. In addition, the attack reconstructs the secret key from the shortest information-theoretically possible prefix of the keystream (usually a small multiple of the keysize), whereas other generic attack techniques like algebraic attacks [1,6] and correlation attacks [10] in many cases require amounts of known keystream that are unlikely to be available in practice. In the case of E_0, the A5/1 generator and the self-shrinking generator, it has been shown in [16] that the performance of the attack in practice does not deviate significantly from the theoretical figures. The inherently high memory requirements of the attack can be reduced by divide-and-conquer strategies based on guessing bits in the initial state at the expense of slightly increased runtime [16,20].

In the ECRYPT stream cipher project eStream [8], a number of new ciphers have recently been proposed and analyzed. Many new designs partly replace LFSRs by other feedback shift registers such as nonlinear feedback shift registers (NFSRs) and feedback shift registers with carry (FCSRs) in order to prevent standard cryptanalysis techniques like algebraic attacks and correlation attacks. Moreover, combinations of different types of feedback shift registers permit alternative compression functions. We show that the BDD-based approach remains applicable in the presence of NFSRs and FCSRs combined with arbitrary output functions as long as not too many new internal bits are produced in each clock cycle of the cipher.

Three of the most promising hardware-oriented submissions to the eStream project are the ciphers Trivium [7], Grain [12], and the F-FCSR family [2]. All three ciphers are part of the focus group and are now being considered for the final portfolio that will be announced in the middle of 2008. We show that the BDD-attack is applicable to these ciphers and obtain the first exploitable cryptanalytic result on the current version of Grain besides generic time-memory-data-tradeoff attacks. Our results for the F-FCSR family emphasize already known but differently motivated security requirements for the choice of parameters.

This paper is organized as follows. We discuss some preliminaries on FSR-based keystream generators and BDDs in Sect. 2 and explain the extended BDD-attack in Sect. 3. Section 4 presents generic constructions for the BDDs that are used in the attack, and Sect. 5 applies our observations to the eStream focus ciphers Trivium, Grain and F-FCSR.

2 Preliminaries

2.1 FSR-Based Keystream Generators

A keystream generator consists of an n-bit internal state $s = (s_0, \ldots, s_{n-1})$, a state update function $F : \{0,1\}^n \to \{0,1\}^n$ and an output function $C : \{0,1\}^n \to \{0,1\}^*$. The starting state s^0 of the generator is derived from a secret key K and a public initialization vector IV. At each clock t, output bits are produced according to $C(s^t)$ from the current state s^t, and the internal state is updated to $s^{t+1} = F(s^t)$. Hence, the output of the generator is completely determined by the starting state s^0.

A widely adopted architecture especially for hardware oriented keystream generators is to represent the internal state in a number of Feedback Shift Registers (FSRs) $R^1, \ldots, R^{k-1}$ of lengths $n^{(0)}, \ldots, n^{(k-1)}$. The state s of the generator is the combined state of the FSRs, hence $n = \sum_{i=0}^{k-1} n^{(i)}$. Among the many types of FSRs discussed in the literature, Linear and Nonlinear Feedback Shift Registers as well as Shift Registers with Carry have proved to be particularly suitable building blocks for keystream generators.

Definition 1. *A* Feedback Shift Register (FSR) *consists of an n-bit register $a = (a_0, \ldots, a_{n-1})$ and a state update function $F : \{0,1\}^n \to \{0,1\}$. Starting from an initial configuration a^0, in each clock a_0 is produced as output and the register is updated according to $a := (a_1, \ldots, a_{n-2}, F(a_0, \ldots, a_{n-1}))$. Depending on whether F is a linear function, we call the register a* Linear Feedback Shift Register (LFSR) *or a* Nonlinear Feedback Shift Register (NFSR).

The FSR-construction is illustrated in Fig. 1.

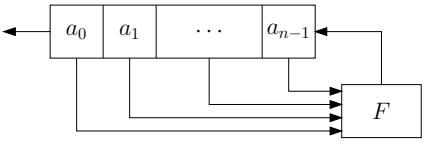

Fig. 1. Feedback Shift Register (FSR) of length n

The definition implies that the output bitstream $(w_t)_{t \geq 0}$ produced from an initial configuration $a^0 = (a_0^0, \ldots, a_{n-1}^0)$ can be expressed as

$$w_t = \begin{cases} a_t^0 & \text{for} \quad t \in \{0, \ldots, n-1\} \\ F(w_{t-n}, \ldots, w_{t-1}) & \text{for} \quad t \geq n \end{cases} .$$

As an alternative to LFSRs and NFSRs, Feedback Shift Registers with Carry were introduced and extensively analyzed in [13].

Definition 2. *A* Feedback Shift Register with Carry *in Fibonacci architecture (Fibonacci FCSR) consists of an n-bit feedback shift register $a = (a_0, a_1, \ldots,$*

a_{n-1}) *with feedback taps* $(c_1, \ldots, c_n)$ *and an additional* q*-bit memory* b *with* $q \leq \lfloor \log_2(n) \rfloor$.

Starting from an initial configuration (a^0, b^0)*, in each clock* a_0 *is produced as output, the sum* $\sigma := b + \sum_{i=1}^{n} a_{n-i} c_i$ *is computed over the integers, and the shift register and memory are updated according to* $a := (a_1, \ldots, a_{n-1}, \sigma \bmod 2)$ *and* $b := \sigma \operatorname{div} 2$.

A Fibonacci FCSR of length n is illustrated in Fig. 2.

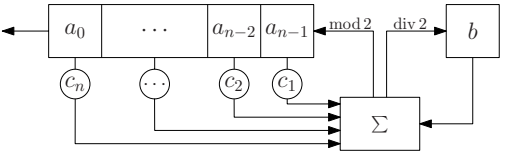

Fig. 2. FCSR of length n in Fibonacci architecture

We call an FCSR state (a, b) *periodic* if, left to run, the FCSR will eventually return to that same state. In case the Fibonacci FCSR is in a periodic state, the memory required to store the integer sum b can be further bounded as follows (cf. [11], Proposition 3.2).

Proposition 1. *If a Fibonacci FCSR is in a periodic state, then the value of the memory* b *is in the range* $0 \leq b < \mathrm{wt}(c)$*, where* $\mathrm{wt}(x)$ *for an* $x \in \{0,1\}^*$ *denotes the Hamming weight of* x.

Hence, if we know that the initial state (a^0, b^0) is periodic, we can limit the size of the memory to $q := \lfloor \log_2(\mathrm{wt}(c) - 1) \rfloor + 1$ bits.

Based on the initial configuration (a^0, b^0), we can describe the output bitstream $(w_t)_{t \geq 0}$ of a Fibonacci FCSR by

$$w_t = \begin{cases} a_t^0 & \text{for} \quad t \in \{0, \ldots, n-1\} \\ \sigma_t \bmod 2 & \text{for} \quad t \geq n \end{cases} ,$$

where $\sigma_t = b^{t-n} + \sum_{i=1}^{n} w_{t-i} c_i$ and $b^{t-n+1} = \sigma_t \operatorname{div} 2$ for $t \geq n$, which implies

$$\sigma_t = (\sigma_{t-1} \operatorname{div} 2) + \sum_{i=1}^{n} w_{t-i} c_i \text{ with } \sigma_{n-1} = 2b^0 \ . \tag{1}$$

Similarly to the Galois architecture of LFSRs, there exists a Galois architecture for FCSRs, which was first observed in [19] and further analyzed in [11].

Definition 3. *A* Feedback Shift Register with Carry *in Galois architecture (Galois FCSR) consists of a main register and a carry register. The main register has* n *register cells* $a_0, \ldots, a_{n-1}$ *and an associated multiplier vector* $(c_1, \ldots, c_n)$ *with* $c_n \neq 0$*. The cells of the carry register are denoted* $b_1, \ldots, b_{n-1}$*. The sign* $\sum$ *represents a full adder. At the* j*-th adder, the following input bits are received: 1)* a_j *from the preceding cell, 2)* $a_0 c_j$ *from the feedback line and 3)* b_j *from the*

memory cell. These are added to form a sum σ_j *(with* $1 \leq j \leq n-1$*). At the next clock cycle, this sum modulo* 2 *is passed on to the next cell in the register, and the higher order bit is used to replace the memory, i.e.,* $a'_{j-1} = \sigma_j \bmod 2$ *and* $b'_j = \sigma_j \operatorname{div} 2$*. At the beginning of the computation, the register is initialzied with a start configuration* (a^0, b^0)*.*

We note that the Galois architecture is generally more efficient than the Fibonacci architecture because the feedback computations can be performed in parallel and each addition involves at most 3 bits. An example Galois FCSR is illustrated in Fig. 3.

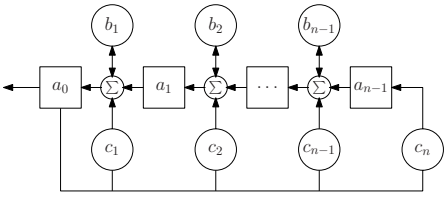

Fig. 3. FCSR of length n in Galois architecture

If memory bits are only present at those positions $i \in \{1, \ldots, n-1\}$ for which $c_i = 1$, which is equivalent to $b_i \leq c_i$ for all i, the Galois-FCSR can be mapped to a Fibonacci-FCSR (cf. Theorem 5.2 and Corollary 5.3 of [11]).

Theorem 1. *For a Galois FCSR with multiplier vector* $(c_1, \ldots, c_n)$ *and an initial loading* (a^0, b^0) *where* $b_i^0 \leq c_i$ *for all* $i \in \{1, \ldots, n-1\}$ *there exists a Fibonacci FCSR with feedback taps* $(c_1, \ldots, c_n)$ *and a starting state* $(\tilde{a}^0, \tilde{b}^0)$ *such that both FCSRs will produce the same output.*

In an FSR-based keystream generator, the FSRs may be interconnected in the sense that the update function F^i of R^i may also depend on the current content of the other registers, i.e., we have $F^i : \{0,1\}^{n_i} \to \{0,1\}$, $n_i \leq n$, for all $i \in \{0, \ldots, k-1\}$. The output function $C : \{0,1\}^n \to \{0,1\}^*$, which derives the output of each clock from the current state, usually depends on one or more state bits from each FSR.

Similarly to a single FSR, we can think of an FSR-based keystream generator with k registers as producing an internal bitstream $(w_t)_{t \geq 0}$, where

$$w_t := w_{s(t)}^{r(t)} \text{ with } r(t) = t \bmod k \text{ and } s(t) = s \operatorname{div} k \ ,$$

i.e., the t-th internal bit of the generator corresponds to the $s(t)$-th bit in the bitstream produced by $R^{r(t)}$. Again, the internal bitstream and hence the output of an FSR-based keystream generator are entirely determined by its starting state s^0, and the first m bits of the internal bitstream w can be computed as $(w_0, \ldots, w_{m-1}) = H_{\leq m}(s^0)$, where $H_{\leq m} : \{0,1\}^n \to \{0,1\}^m$. We denote the prefix of the keystream that is produced from an m-bit internal bitstream w by $C_m(w)$ with $C_m : \{0,1\}^m \to \{0,1\}^*$.

We call an integer i an *initial position* in an internal bitstream w, if w_i corresponds to a bit from the initial state of some FSR, and a *combined position* otherwise. Correspondingly, we denote by $\mathrm{IP}(i)$ the set of initial positions and by $\mathrm{CP}(i)$ the set of combined positions in $\{0,\ldots,i-1\}$. For an internal bistream w, let $\mathrm{IB}(w)$ denote the bits at the initial positions in w. Let $n_{\min}$ denote the maximum i for which all $i' \leq i$ are initial positions and $n_{\max}$ the minimum i for which all $i' > i$ are combined positions.

Definition 4. *We call an FSR-based keystream generator* regular*, if* $|C_m(w)| = \beta(m)$ *for all* $w \in \{0,1\}^m$*, i.e., an internal bitstream of length* m *always yields* $\beta(m)$ *keystream bits.*

Two important parameters of FSR-based keystream generators are the *best-case compression ratio* and the *information rate*, which we define as follows.

Definition 5. *If* γm *is the maximum number of keybits that the generator produces from internal bitstreams of length* m*, we call* $\gamma \in (0,1]$ *the* best-case compression ratio *of the generator. Moreover, for a randomly chosen and uniformly distributed internal bitstream* $W^{(m)} \in \{0,1\}^m$ *and a random keystream* Z*, we define as* information rate α *the average information that* Z *reveals about* $W^{(m)}$*, i.e.,* $\alpha := \frac{1}{m} I\left(W^{(m)}, Z\right) \in (0,1]$.[1]

Assumption 1 (Independence Assumption). *For all* $m \geq 1$*, a randomly chosen, uniformly distributed internal bitstream* $w^{(m)}$*, and all keystreams* $z \in \{0,1\}^*$*, we have* $Prob_w[C(w)$ *is prefix of* $z] = p_C(m)$*, i.e., the probability of* $C(w)$ *being a prefix of* z *is independent of* z*.*

As shown in [14], the computation of α can be simplified as follows if the generator fulfills the independence assumption.

Lemma 1. *If the Independence Assumption holds for a keystream generator, we have* $\alpha = -\frac{1}{m}\log_2(p_C(m))$.

Corollary 1. *The information rate* α *of a regular FSR-based keystream generator fulfilling the Independence Assumption is given by* $\alpha = \frac{\beta(m)}{m}$.

Proof. The Independence Assumption and Definition 4 imply that the $2^{\beta(m)}$ possible keystream blocks of length $\beta(m)$ that can be produced from the m-bit internal bitstream all have probability $p_C(m)$. Hence $p_C(m) = 2^{-\beta(m)}$ and therefore $\alpha = -\frac{1}{m}\log_2(2^{-\beta(m)}) = \frac{\beta(m)}{m}$. □

Finally, we assume the output keystream to behave pseudorandomly, which we formalize as follows.

Assumption 2 (Pseudorandomness Assumption). *For* $m \leq \lceil \alpha^{-1} n \rceil$*, let* $w^{(m)}$ *and* s^0 *denote randomly chosen, uniformly distributed elements of* $\{0,1\}^m$ *and* $\{0,1\}^{|\mathrm{IP}(m)|}$*, respectively. Then, it holds for all keystreams* z *that* $Prob_w[C(w)$ *is prefix of* $z] \approx Prob_{s^0}[C(H_{\leq m}(s^0))$ *is prefix of* $z]$.

[1] Recall that for two random variables A and B, the value $I(A,B) = H(A) - H(A|B)$ defines the information that B reveals about A.

We expect the Pseudorandomness Assumption to hold since a significant violation would imply the vulnerability of the generator to a correlation attack.

2.2 Binary Decision Diagrams (BDDs)

We briefly review the definitions of Binary Decision Diagrams and their most important algorithmic properties.

Definition 6. *A* Binary Decision Diagram *(BDD) over a set of variables* $X_n = \{x_1, \ldots, x_n\}$ *is a directed, acyclic graph* $G = (V, E)$ *with* $E \subseteq V \times V \times \{0, 1\}$. *Each inner node* v *has exactly two outgoing edges, a* 0*-edge* $(v, v_0, 0)$ *and a* 1*-edge* $(v, v_1, 1)$ *leading to the* 0*-successor* v_0 *and the* 1*-successor* v_1*, respectively. A BDD contains exactly two nodes with outdegree* 0*, the sinks* s_0 *and* s_1*. Each inner node* v *is assigned a label* $v.label \in X_n$*, whereas the two sinks are labeled* $s_0.label = 0$ *and* $s_1.label = 1$*. There is exacly one node with indegree* 0*, the root of the BDD. We define the size of a BDD to be the number of nodes in* G*, i.e.,* $|G| := |V|$*. Each node* $v \in V$ *represents a Boolean Function* $f_v \in B_n = \{f | f : \{0, 1\}^n \to \{0, 1\}\}$ *in the following manner: For an input* $a = (a_1, \ldots, a_n) \in \{0, 1\}^n$*, the computation of* $f_v(a)$ *starts in* v*. In a node with label* x_i*, the outgoing edge with label* a_i *is chosen, until one of the sinks is reached. The value* $f_v(a)$ *is then given by the label of this sink.*

Definition 7. *For a BDD* G *over* X_n*, let* $G^{-1}(1) \subseteq \{0, 1\}^n$ *denote the set of inputs accepted by* G*, i.e., all inputs* $a \in \{0, 1\}^n$ *such that* $f_{\text{root}}(a) = 1$.

Since general BDDs have many degrees of freedom for representing a particular Boolean function, many important operations and especially those that are needed in our context are NP-hard. We therefore concentrate on the more restricted model of Ordered Binary Decision Diagrams (OBDDs), which are defined as follows.

Definition 8. *A* variable ordering π *for a set of variables* $X_n = \{x_1, \ldots, x_n\}$ *is a permutation of the index set* $I = \{1, \ldots, n\}$*, where* $\pi(i)$ *denotes the position of* x_i *in the* π*-ordered variable list* $x_{\pi^{-1}(1)}, x_{\pi^{-1}(2)}, \ldots, x_{\pi^{-1}(n)}$.

Definition 9. *A* π-Ordered Binary Decision Diagram (π-OBDD) *with respect to a variable ordering* π *is a BDD in which the sequence of tests on a path from the root to a sink is restricted by* π*, i.e., if an edge leads from an* x_i*-node to an* x_j*-node, then* $\pi(i) < \pi(j)$*. We call a BDD* G *an OBDD, if there exists a variable ordering* π *such that* G *is a* π*-OBDD. We define the width of an OBDD* G *as* $w(G) := \max_i\{|\{v \in G | v.\text{label} = x_i\}|\}$.

Figure 4 shows a π-OBDD that computes the function $f(z_0, \ldots, z_3) = z_0 z_2 \vee z_0 \bar{z}_2 z_3 \vee \bar{z}_0 z_1 z_3$.

In contrast to BDDs, OBDDs allow for efficient implementations of the operations that we are interested in. Let π denote a variable ordering for $X_n = \{x_1, \ldots, x_n\}$ and let the π-OBDDs G_f, G_g and G_h represent Boolean functions $f, g, h : \{0, 1\}^n \to \{0, 1\}$. We have $|G_f| \leq m \cdot w(G_f)$, and there exists an algorithm MIN that computes in time $O(|G_f|)$ the uniquely determined minimal

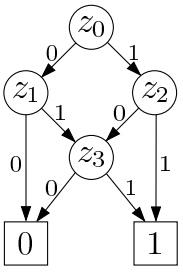

Fig. 4. A π-OBDD over $\{z_0, \ldots, z_3\}$ with $\pi(0) = 0$, $\pi(1) = 2$, $\pi(2) = 1$ and $\pi(3) = 3$

π-OBDD G with $w(G) \leq |G_f^{-1}(1)|$ that represents f. In time $O(|G_f| \cdot |G_g| \cdot |G_h|)$, we can compute a minimal G_0-OBDD G with $w(G) \leq w(G_f) \cdot w(G_g) \cdot w(G_h)$ that represents the function $f \wedge g \wedge h$. Additionally, it is possible to enumerate all elementes in $G_f^{-1}(1)$ in time $O\left(n \cdot |G_f^{-1}(1)|\right)$. We refer the reader to [21] for details on BDDs, OBDDs and the corresponding algorithms.

Note that we can straightforwardly use BDDs as a datastructure for subsets of $\{0,1\}^n$. In order to represent an $S \subseteq \{0,1\}^n$, we construct a BDD G_S that computes the characteristic function f_S of S given by $f_S(x) = 1$ if $x \in S$ and $f_S(x) = 0$ otherwise. Hence, G_S will accept exactly the elements of S. Moreover, we can compute a BDD representing the intersection $S \cap T$ of two sets S and T from their BDD-representations G_S and G_T by an AND-synthesis of G_S and G_T.

3 BDD-Based Initial State Recovery

The BDD-based attack on keystream generators, which was first introduced in [14], is a known-plaintext initial state recovery attack, i.e., the attacker tries to reconstruct the unknown initial state s^0 of the keystream generator from a few known plaintext bits $p_1, p_2, \ldots$ and their encryptions $c_1, c_2, \ldots$. Since a ciphertext bit c_i is computed from a plaintext bit p_i and a keystream bit z_i via $c_i = p_i \oplus z_i$, the keystream bit z_i can be reconstructed from (p_i, c_i) by computing $p_i \oplus c_i = z_i$.

We observe that for any internal bitstream $w \in \{0,1\}^m$ that yields a prefix of the observed keystream, the following two conditions must hold.

Condition 1. *w is an m-extension of the initial state bits in w, i.e., we have $H_{\leq m}(\mathrm{IB}(w)) = w$.*

Condition 2. *$C_m(w)$ is a prefix of the observed keystream z.*

We call any $w \in \{0,1\}^m$ that satisfies these conditions an *m-candidate*. Our strategy is now to start with $m = n_{\min}$ and to dynamically compute the m-candidates for $m > n_{\min}$, until only one m-candidate is left. The first bits of this m-candidate will contain the initial state s^0 that we are looking for. We can expect to be left with only one m-candidate for $m \geq \lceil \alpha^{-1} n \rceil$, which follows directly from the following Lemma (cf. [14] for a proof).

Lemma 2. *Under Assumption 2, it holds for all keystreams z and all $m \leq \lceil \alpha^{-1} n \rceil$ that $|\{s^0 \in \{0,1\}^n : C_m(H_{\leq m}(s^0)) \text{ is prefix of } y\}| \approx 2^{|\mathrm{IP}(m)|-\alpha m} \leq 2^{n-\alpha m}$. Hence, there exist approximately $2^{n-\alpha m}$ m-candidates.*

In order to compute and represent the intermediate m-candidates efficiently, we use the following BDD-based approach. For a given regular keystream generator we choose a suitable reading order π and represent bitstreams w fulfilling conditions 1 and 2 in the minimal π-OBDDs R_m and Q_m, respectively. Starting from $P_{n_{\min}} = Q_{n_{\min}}$, we compute for $n_{\min} < m \leq \lceil \alpha^{-1} n \rceil$ the π-OBDD $P_m = \mathrm{MIN}(P_{m-1} \wedge Q_m \wedge S_m)$, where the minimum π-OBDD S_m tests whether w_{m-1} is in the m-extension of $\mathrm{IB}(w)$. Note that we have $P_m = \mathrm{MIN}(Q_m \wedge R_m)$ with $R_m = \bigwedge_{i=1}^{m} S_i$ for all m, and P_m accepts exactly the m-candidates.

The cost of this strategy essentially depends on the sizes of the intermediate results P_m, which can be determined as follows.

Assumption 3 (BDD Assumption). *For all $m \geq n_{min}$, it holds that $w(R_m) \leq 2^{p \cdot |\mathrm{CP}(m)|}$ for an integer $p \geq 1$ and $w(S_m), w(Q_m) \in m^{O(1)}$.*

Lemma 3. *Let $\mathcal{K}$ denote a regular FSR-based keystream generator with k FSRs $R^0, \ldots, R^{k-1}$ of lengths $n^{(0)}, \ldots, n^{(k-1)}$. If $\mathcal{K}$ fulfills the BDD assumption and the Pseudorandomness Assumption, it holds for $n = \sum_{i=0}^{k-1} n^{(i)}$ and all $n_{\min} < m \leq \lceil \alpha^{-1} n \rceil$ that $w(P_m) \leq n^{O(1)} 2^{\frac{p(1-\alpha)}{p+\alpha} n}$.*

The proof of Lemma 3 is analogous to the LFSR-case presented in [14,15] and can be found in Appendix A.

From this bound on $w(P_m)$, we can straightforwardly derive the time, space and data requirements of the BDD-based attack.

Theorem 2. *Let $\mathcal{K}$ denote a regular FSR-based keystream generator with an unknown initial state $s^0 \in \{0,1\}^n$, information rate α and best-case compression ratio γ. If $\mathcal{K}$ fulfills the Independence Assumption, the Pseudorandomness Assumption and the BDD Assumption, an initial state $\tilde{s}^0$ that yields the same keystream as s^0 can be computed in time and with space $n^{O(1)} 2^{\frac{p(1-\alpha)}{p+\alpha} n}$ from the first $\lceil \gamma \alpha^{-1} n \rceil$ consecutive keystream bits of $\mathcal{K}$ under s^0.*

4 Generic BDD Constructions

4.1 Keystream Consistency Check Q_m

In most cases, the BDD Q_m that checks Condition 2 can be straightforwardly derived from the definition of the output function C. If the computation of an output bit z_t depends on $u(j) > 1$ bits from an FSR R^j, a fixed bit in the bitstream produced by R^j will generally appear and have to be read in the computation of up to $u(j)$ output bits. In this case, we compute an output bit z_t from a number of *new bits* which are being considered for the first time, and several *old bits* that were already involved in the computation of previous output bits. This would imply reading a fixed variable more than once on the same

path in Q_m, which is prohibited by the OBDD-definition. The less restrictive BDDs permit this construction, but can no longer guarantee the efficiency of the operations that our attack depends on. A similar problem has been considered in [14] in the context of the irregularly clocked A5/1 generator [5], which uses the bits of the internal bitstream both for computing output bits and as input for the clock control mechanism. A possible solution, which was also proposed in [14], is to increase the number of unknowns by working with $u(j)$ synchronized duplicates of the R^j-bitstream at the expense of a reduced information rate α.

We now consider the more general situation that the update function depends on the new bits and some function(s) $g_1, \ldots, g_r$ in the old bits. In this case, it suffices to introduce auxiliary variables for the values of these functions in order to ensure that z_t is computed only from new bits. This construction is illustrated in the following example.

Example 1. Consider the output function $z_t = C(w_{t+5}, w_{t+7}, w_{t+9})$, where C is defined by $C(x_1, x_2, x_3) = x_1 \oplus x_2 \oplus x_3$. Assuming canonical reading order, w_{t+9} would be the new bit and w_{t+5} and w_{t+7} the old bits. With the auxiliary variable $\tilde{w}_t := g_1(w_{t+5}, w_{t+7})$ and $g_1(x_1, x_2) := x_1 \oplus x_2$, we can express z_t as $z_t = \tilde{w}_t \oplus w_{t+9}$.

If we add for each auxiliary variable an FSR to the generator that outputs at clock t the corresponding value of g_j, we can compute z_t without considering the bits from the internal bitstream more than once. Obviously, the resulting equivalent generator is regular, but will have a lower information rate as before, since more bits of the internal bitstream have to be read in order to compute the same number of keystream bits.

4.2 FSR Consistency Check R_m

Recall that each bit w_t of an internal bitstream w is either an initial state bit of some FSR or a combination of other internal bits. In order to decide for a given internal bitstream whether it satisfies Condition 1, we need to check whether the update relations imposed on the bits at the combined positions are fulfilled. Hence, if a combined bit w_t is produced by an update relation $f^i(s_0, \ldots, s_{n-1})$, we need to check whether $f^i(w_{i_1}, \ldots, w_{i_p}) = w_t$, which is equivalent to testing whether

$$\tilde{f}^i(w_{i_1}, \ldots, w_{i_p}, w_t) := f^i(w_{i_1}, \ldots, w_{i_p}) \oplus w_t = 0 \ .$$

The OBDD S_m implements this test for a single combined bit w_{m-1} and represents the constant-one function if w_{m-1} is an initial bit. The OBDD $R_m = \bigwedge_{i=1}^{m} S_i$ performs the consistency tests for the whole internal bitstream.

We first consider the case of FSRs (without additional memory), for which we need the following definition.

Definition 10. *For a polynomial* $f : \{0,1\}^n \to \{0,1\}$ *with*

$$f(w_1, \ldots, w_n) = \bigoplus_{j \in M} m_j \ \textit{with}\ m_j = \bigwedge_{l \in M^j} w_l \ \textit{and}\ M^j \subseteq \{1, \ldots, n\} \ ,$$

and a reading order $\pi \in \sigma_n$*, we define the set of* active monomials *at clock* t *as*

$$\mathrm{AM}_\pi(f,t) := \{m_j : 0 < |\{\pi^{-1}(1),\ldots,\pi^{-1}(t)\} \cap M^j| < |M^j|\} \ .$$

Hence, $\mathrm{AM}(f,t)$ contains all monomials in f for which at least one, but not all factors are known after the first t inputs have been read.

Lemma 4. *For a polynomial $f : \{0,1\}^n \to \{0,1\}$ with $n > 1$ and a reading order π for the inputs, the set of inputs satisfying $f(w_1,\ldots,w_n) = 0$ can be represented in a π-OBDD of width $2^{\max_{1\le t\le n}\{|\mathrm{AM}_\pi(f,t)|\}+1}$.*

Proof. Let $p := \max_{1\le t\le n}\{|\mathrm{AM}_\pi(f,t)|\}$. In order to compute $f(w_1,\ldots,w_n)$, we may proceed in the following way. We define p auxiliary variables $b_1,\ldots,b_p$, which will store the intermediate values of partly evaluated monomials, and an additional variable b_0 for the sum of evaluated monomials. We initialize $b_0 := 0$, $b_t := 1$ for $t > 0$, and read the variables $w_1,\ldots,w_n$ in the order given by π. For each variable w_t, we update all auxiliary variables that are associated with monomials containing w_t. If a monomial becomes active by reading w_t, we allocate an auxiliary variable b_j and define $b_j := w_t$. If a monomial is entirely evaluated after reading w_t, we add its value to b_0 and free the associated auxiliary variable. Since there are at most p active monomials at any time, no more than $p+1$ auxiliary variables will be needed simultaneously.

From this strategy, we construct a π-OBDD G_f as follows. We define the vertex set

$$V(G_f) := \{(t, b_0,\ldots,b_p)\} \subseteq \{1,\ldots,n\} \times \{0,1\}^{p+1}$$

and the root of G_f as $(1,1,\ldots,1)$. For a monomial m_j, let $b_{\delta(j)}$ denote the auxiliary variable associated with m_j. For each $t \in \{1,\ldots,n-1\}$ and $i \in \{0,1\}$, we define the i-successor of $(\pi^{-1}(t), b_0,\ldots,b_p)$ as $(\pi^{-1}(t+1), b_0^i,\ldots,b_p^i)$. If m_j became inactive in $\pi^{-1}(t)$, i.e., m_j is active in $\pi^{-1}(t-1)$ but inactive in $\pi^{-1}(t)$, we set $b_0^i := b_0 \oplus b_{\delta(j)} \cdot i$ and reset $b_{\delta(j)}$ to 1. If m_j is active in t, we set $b^i_{\delta(j)} := b^i_{\delta(j)} \cdot i$. For all remaining auxiliary variables, we define $b^i_{\delta(j)} := b_{\delta(j)}$. If $t = n$, we compute $b_0^i,\ldots,b_p^i$ as before, and the i-successor of $(\pi^{-1}(t), b_0,\ldots,b_p)$ is defined to be the sink labeled with the value of $b_0^i \oplus 1$. This construction results in a π-OBDD that accepts only those inputs $(w_1,\ldots,w_n)$ that satisfy $f(w_1,\ldots,w_n) = 0$.

For all $i \in \{1,\ldots,n\}$, the OBDD contains at most 2^{p+1} vertices that are labeled with w_i, which implies the claim. □

From Lemma 4, we can directly derive an upper bound for the width of the π-OBDD S_m for an FSR.

Corollary 2. *For a given reading order π, an integer $m > 0$, an FSR R with update relation f, and $p := \max_{0\le t<m}\{|\mathrm{AM}_\pi(\tilde{f},t)|\} + 1$, we can construct a π-OBDD S_m of width at most 2^p that tests for an internal bitstream $w \in \{0,1\}^m$ if w fulfills the update relation imposed on w_{m-1}.*

Note that for $p = 1$, we obtain the LFSR-bound that was proved in [14].

We now turn to the case of Fibonacci FCSRs. Equation (1) implies that we need access to σ_{t-1} in order to check whether the update relation holds for w_t. Therefore, we work with a modified FCSR that outputs the sum σ_t instead of the

bit $w_t = \sigma_t \bmod 2$ in each clock. More precisely, the modified FCSR outputs for an initial memory state $(b_{q-1}^0, \ldots, b_0^0)$ with $b^0 = \sum_{i=0}^{q-1} b_i^0 2^i$ the values $a_t^0 =: \sigma_t^0$ for $t < n-1$, $(b_0^{q-1}, \ldots, b_0^0, a_{n-1}^0)$ for $t = n-1$, and $(\sigma_t^q, \sigma_t^{q-1}, \ldots, \sigma_t^0)$ for $t \geq n$ with $\sigma_t = \sum_{i=0}^{q} \sigma_t^i 2^i$ and $w_t = \sigma_t^0$.

Lemma 5. *For a Fibonacci FCSR R with q bits of memory, an integer $m > 0$, and π the canonical reading order, we can construct a π-OBDD S_m of width at most 2^{q+1} that tests for the internal bitstream $w \in \{0,1\}^m$ of the modified FCSR with $m = n - 1 + t(q+1)$ whether the last $q+1$ bits fulfill the update relation.*

Proof. In order to check whether $\sigma_t = (\sigma_{t-1} \operatorname{div} 2) + \sum_{i=1}^{n} w_{t-i} \cdot c_i$, we can equivalently test if

$$\sigma_t = \sum_{i=1}^{q} \sigma_{t-1}^i + \sum_{i=1}^{n} \sigma_{t-i}^0 \cdot c_i \ .$$

This test can be performed in a π-OBDD as follows. Define the vertex set V as

$$V := \{0, \ldots, m-1\} \times \{0,1\}^{q+1} \ ,$$

such that a vertex $v = (k, \sigma)$ with $\sigma = (\sigma^0, \ldots, \sigma^q)$ consists of a variable number k corresponding to some σ_l^j, and $q+1$ bits for storing the comparison value for $\sigma_t^0, \ldots, \sigma_t^q$. The root of the OBDD is $(0, 0, \ldots, 0)$. For each vertex $v \in V$ and each $i \in \{0,1\}$, the i-successor $v_i = (k+1, \tilde{\sigma})$, $\tilde{\sigma} = (\tilde{\sigma}^0, \ldots, \tilde{\sigma}^q)$, is defined as follows. If $l \in \{t-n, \ldots, t-1\}$ and $j = 0$, we compute $\tilde{\sigma} = \sigma + \sigma_{t-j}^0 \cdot c_j$. If $l = t-1$ and $j \in \{1, \ldots, q\}$, we define $\tilde{\sigma} = \sigma + \sigma_{t-1}^j 2^j$. If $l = t$ and $j \in \{0, \ldots, q\}$ and $\sigma_l^j \neq \sigma^j$, we define v_i to be the 0-sink. If $l = t$, $j = q$ and $\sigma_l^j = \sigma^j$, v_i is the 1-sink.

We can straightforwardly verify that this construction yields a π-OBDD of width at most 2^{q+1} which accepts only those inputs for which σ_t satisfies the update relations. □

In the case of Galois FCSRs with $b_i \leq c_i$ at all times, we denote by $a_i(t)$ and $b_i(t)$ the value of the register cells a_i and b_i at time t. The definition of Galois FCSRs implies $a_{n-1}(t) = a_0(t-1)$ and for $i \in \{n-2, \ldots, 0\}$ that $a_i(t) = a_{i+1}(t-1)$ if $c_i = 0$ and $a_i(t) = a_{i+1}(t-1) \oplus b_{i+1}(t-1) \oplus a_0(t-1)$ if $c_i = 1$. We therefore focus on the nontrivially computed bits and think of the main register as producing the bitstream

$$a_0(0), a_1(0), \ldots, a_{n-1}(0), \ldots, a_{i_1}(t), \ldots, a_{i_l}(t), \ldots \ ,$$

where $\{i_1, \ldots, i_l\} = \{1 \leq i < n | c_{i+1} = 1\}$, $l = \operatorname{wt}(c) - 1$, and $t > 0$. Similarly, we view the bitstream produced by the carry register as $b_{i_1}(t), \ldots, b_{i_l}(t), \ldots$ for $t \geq 0$.

Lemma 6. *For a Galois FCSR R with $b_i \leq c_i$ for all $i \in \{1, \ldots, n-1\}$, an integer $m > 0$, and π the canonical reading order, we can construct a π-OBDD S_m of width at most 2 that tests whether a bit in the bitstream produced by the main register fulfills the corresponding update relations. For a bit in the bitstream of the carry register, we can perform this consistency test in a π-OBDD of maximum width 8.*

Proof. According to Corollary 2, we can test the linear conditions on the $a_i(t)$ where $c_{i+1} = 1$ in a π-OBDD of width at most 2. Similarly, Corollary 2 yields a maximum width of $2^3 = 8$ in the case of the carry register since $b_{i_j}(t)$ can be computed as

$$b_{i_j}(t) = a_{i_j}(t-1)b_{i_j}(t-1) \oplus b_{i_j}(t-1)a_0(t-1) \oplus a_0(t-1)a_{i_j}(t-1) \ . \qquad \square$$

From the bounds on $w(S_m)$ for the different types of FSRs, we can now straightforwardly derive a bound for $w(R_m)$ for an FSR-based keystream generator. Let $\mathcal{K}$ denote an FSR-based keystream generator consisting of k FSRs $R^0, \ldots, R^{k-1}$ with π-OBDDs $S_m^0, \ldots, S_m^{k-1}$ with sizes at most $m^{O(1)}2^{p_i}$ for all $i \in \{0, \ldots, k-1\}$. Moreover, let s_i denote the fraction of combined bits that R^i contributes to the internal bitstream.

Corollary 3. *There exists a π-OBDD R_m of width at most $2^{|\mathrm{CP}(m)|\sum_{i=0}^{k-1} p_i s_i}$ that tests for a bitstream $w \in \{0,1\}^m$ whether it is an m-extension of the initial bits.*

Proof. The claim follows directly from $R_m = \bigwedge_{i=1}^{m} S_i$ and the OBDD-properties described in Sect. 2.2. $\square$

5 Applications

5.1 Trivium

Trivium [7] is a regular keystream generator consisting of three interconnected NFSRs R^0, R^1, R^2 of lengths $n^{(0)} = 93$, $n^{(1)} = 84$, and $n^{(2)} = 111$. The 288-bit initial state of the generator is derived from an 80 bit key and an 80 bit IV. The output function computes a keystream bit z_t by linearly combining six bits of the internal state, with each NFSR contributing two bits (cf. Appendix 6 for details). In order to mount the BDD-attack on Trivium, we write the output function as

$$z_t = g_1(s_1, s_{94}, s_{178}) \oplus s_{28} \oplus s_{109} \oplus s_{223}$$

and proceed as described in Sect. 4.1 by adding an LFSR R^3 which computes g_1 to the generator. For π equal to the canonical reading order, we have $p_i = \max_{1 \le t \le 288}\{|\mathrm{AM}_\pi(\tilde{f}^i, t)|\} + 1 = 2$ and $s_i = \frac{1}{4}$ for $i \in \{0, 1, 2\}$ as well as $p_3 = 1$ and $s_3 = \frac{1}{4}$, which implies $p = \sum_{i=0}^{3} p_i s_i = \frac{7}{4}$. Since the modified generator computes one keystream bit from four internal bits, we have $\beta(m) = \frac{1}{4}m$ and $\alpha = \gamma = \frac{1}{4}$. Based on Lemma 4, we can obviously construct a π-OBDD Q_m with $w(Q_m) \le 2$ that performs the consistency test for the observed keystream z such that the generator fulfills the BDD Assumption. Since each keystream bit involves 3 new bits, we can expect the Independence Assumption to hold. By plugging α, γ and p into the statement of Theorem 2, we obtain:

Theorem 3. *The secret initial state of the* Trivium *automaton can be recovered from the first n keystream bits in time and with space $n^{O(1)}2^{0.65625n} \approx 2^{189}$ for $n = 288$.*

Theorem 3 shows that the BDD-attack is applicable to Trivium, but its performance is not competitive with recently published attacks, which recover the initial state in around 2^{100} operations from $2^{61.5}$ keystream bits [17] or in around 2^{135} operations from $O(1)$ keystream bits [9].

5.2 Grain-128

The regularly clocked stream cipher Grain-128 was proposed in [12] and supports key size of 128 bits and IV size of 96 bits. The design is based on two interconnected shift registers, an LFSR R^0 and an NFSR R^1, both of lengths $n^{(0)} = n^{(1)} = 128$ and a nonlinear output function. The content of the LFSR is denoted by $s_t, s_{t+1}, \ldots, s_{t+127}$ and the content of the NFSR by $b_t, b_{t+1}, \ldots, b_{t+127}$. The corresponding update functions and the output function are given in Appendix C.

We add to the keystream generator an NFSR R^2 which computes the output bits z_t and have the generator output the values produced by R^2 in each clock. Hence, we can compute one keystream bit from 3 internal bits, which implies $\beta(m) = \frac{1}{3}m$ and $\alpha = \gamma = \frac{1}{3}$. For π equal to the canonical reading order, it is $p_0 = 1$, and we have $p_1 = \max_{0 \le i \le 117}\{|\mathrm{AM}_\pi(\tilde{f}^1, t+i)|\} + 1 = 4$, and $p_2 = \max_{0 \le i \le 95}\{|\mathrm{AM}_\pi(\tilde{f}^2, t+i)|\} + 1 = 4$. Hence, $p = \frac{1}{3} + \frac{4}{3} + \frac{4}{3} = 3$. Obviously, the consistency test for an observed keystream can be performed by a π-OBDD Q_m with $w(Q_m) \le 2^3 = 8$ according to Lemma 4. Therefore, the modified generator fulfills the BDD Assumption. Since new bits are utilized in the computation of each keybit, we can expect the Independence Assumption to hold. Hence, the application of Theorem 2 yields

Theorem 4. *The secret initial state of the Grain automaton can be recovered from the first n keystream bits in time and with space $n^{O(1)}2^{0.6n} \approx 2^{154}$ for $n = 256$.*

Theorem 4 is to the best of our knowledge the first exploitable cryptanalytic result besides generic time-memory-data-tradeoff attacks [3], which require time and keystream around 2^{128}.

5.3 The F-FCSR Stream Cipher Family

The F-FCSR stream cipher family in its current version is specified in [2]. It consists of the variants F-FCSR-H and F-FCSR-16.

F-FCSR-H has keylength 80 bits and consists of a single Galois FCSR M of length $n = 160$ and a feedback tap vector c of Hamming weight 83. Memory cells are only present at those 82 positions $i \in \{1, \ldots, n-1\}$, for which $c_i = 1$. At each clock, eight output bits b_j are created by taking the XOR-sum of up to 15 variables of the current internal state (cf. Appendix D for details).

In order to mount the BDD-attack, we split the FCSR into the main register R^0 and the carry register R^1. Since each output bit is computed as the sum of up to 15 internal bits, we are in a similar situation as described in Example 1 and

need additional LFSRs $R^2, \ldots, R^9$ to compute the keystream bits b_j, $0 \leq j < 8$. The modified output function simply returns these bits in each clock. With $l := \mathrm{wt}(c)-1$ we obtain eight output bits from $2l+8$ internal bits, hence $\beta(m) = \frac{8}{2l+8}m$ and $\alpha = \gamma = \frac{8}{2l+8} = \frac{2}{43}$. We have $p_0 = p_1 = l$, $p_i = 1$ for $i \in \{2, \ldots, 9\}$, $s_0 = s_1 = \frac{l}{2l+8}$, and $s_i = \frac{1}{2l+8}$ for $2 \leq i \leq 9$, which implies $p = \frac{2l^2+1}{2l+1}$. Obviously, the consistency test for the observed keystream z can be performed by an OBDD Q_m with $w(Q_m) \leq 2$. Hence, the modified F-FCSR-H fulfills the BDD Assumption. Since the computation of the keybits involves new internal bits in every clock, we can expect the Independence Assumption to hold. Note that we have l additional unknowns from the initial value of the carry register. Plugging the computed values into the statement of Theorem 3 implies the following theorem.

Theorem 5. *The secret initial state of the F-FCSR-H automaton can be recovered from the first $n+l$ keystream bits in time and with space $n^{O(1)}2^{0.9529(n+l)} \approx 2^{231}$ for $n = 160$ and $l = 82$.*

The F-FCSR-16 generator has the same structure as F-FCSR-H, but larger parameters. More precisely, we have keylength 128 bits, $n = 256$, and the feedback tap vector has Hamming weight 131 (i.e., $l = 130$), where memory cells are only present at nonzero tap positions as before. Since F-FCSR-16 produces 16 output bits per clock, we construct 16 additional LFSRs that produce these bits. Hence, we can compute 16 output bits from $2l + 16$ internal bits, which implies $\beta(m) = \frac{16}{2l+16}m$ and $\alpha = \gamma = \frac{16}{2l+16} = \frac{4}{69}$. Analogously to the case of F-FCSR-H, we obtain $p = \frac{2l^2+16}{2l+16}$. The modified generator satisfies the Independence Assumption and the BDD assumption as before, and we have $l = 130$ additional unknowns. We obtain by applying Theorem 3:

Theorem 6. *The secret initial state of the F-FCSR-16 automaton can be recovered from the first $n + l$ keystream bits in time and with space $n^{O(1)}2^{0.94(n+l)} \approx 2^{363}$ for $n = 256$ and $l = 130$.*

Our analysis supports the security requirement that the Hamming weight of c should not be too small, which was also motivated by completely different arguments in [2]. Although the BDD-attack is to the best of our knowledge the first nontrivial attack on the current version of the F-FCSR family, it is far less efficient than exhaustive key search.

6 Conclusion

In this paper, we have shown that the BDD-attack can be extended to keystream generators based on nonlinear feedback shift registers (NFSRs) and feedback shift registers with carry (FCSRs) as well as arbitrary output functions. We have applied our observations to the three eStream focus candidates Trivium, Grain and F-FCSR. In the case of Grain, we obtain the first exploitable cryptanalytic result besides generic time-memory-data tradeoffs. Our analysis of the F-FCSR family provides additional arguments for already proposed security requirements.

Acknowledgement. I would like to thank Willi Meier, Simon Künzli, and Matthias Krause for their valuable comments helping to improve this article.

References

1. Armknecht, F., Krause, M.: Algebraic attacks on combiners with memory. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 162–176. Springer, Heidelberg (2003)
2. Arnault, F., Berger, T.P., Lauradoux, C.: Update on F-FCSR stream cipher. eSTREAM, ECRYPT Stream Cipher Project, Report 2006/025 (2006), http://www.ecrypt.eu.org/stream
3. Biryukov, A., Shamir, A.: Cryptanalytic time/memory/data tradeoffs for stream ciphers. In: Okamoto, T. (ed.) ASIACRYPT 2000. LNCS, vol. 1976, pp. 1–13. Springer, Heidelberg (2000)
4. The Bluetooth SIG. Specification of the Bluetooth System (February 2001)
5. Briceno, M., Goldberg, I., Wagner, D.: A pedagogical implementation of A5/1 (May 1999), http://jya.com/a51-pi.htm
6. Courtois, N.: Fast algebraic attacks on stream ciphers with linear feedback. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 177–194. Springer, Heidelberg (2003)
7. de Cannière, C., Preneel, B.: Trivium specifications. eSTREAM, ECRYPT Stream Cipher Project (2005), http://www.ecrypt.eu.org/stream
8. eSTREAM, ECRYPT stream cipher project. http://www.ecrypt.eu.org/stream
9. eSTREAM Discussion Forum. A reformulation of Trivium. eSTREAM, ECRYPT Stream Cipher Project, Discussion Forum (2005), http://www.ecrypt.eu.org/stream/phorum/read.php?1,448
10. Golić, J.: Correlation properties of general binary combiners with memory. Journal of Cryptology 9(2), 111–126 (1996)
11. Goresky, M., Klapper, A.: Fibonacci and galois representations of feedback-with-carry shift registers. IEEE Transactions on Information Theory 48(11), 2826–2836 (2002)
12. Hell, M., Johansson, T., Maximov, A., Meier, W.: A stream cipher proposal: Grain-128. eSTREAM, ECRYPT Stream Cipher Project, Report 2005/010 (2005), http://www.ecrypt.eu.org/stream
13. Klapper, A., Goresky, M.: Feedback shift registers, 2-adic span, and combiners with memory. Journal of Cryptology 10, 111–147 (1997)
14. Krause, M.: BDD-based cryptanalysis of keystream generators. In: Knudsen, L.R. (ed.) EUROCRYPT 2002. LNCS, vol. 2332, pp. 222–237. Springer, Heidelberg (2002)
15. Krause, M.: OBDD-based cryptanalysis of oblivious keystream generators. Theor. Comp. Sys. 40(1), 101–121 (2007)
16. Krause, M., Stegemann, D.: Reducing the space complexity of BDD-based attacks on keystream generators. In: Robshaw, M. (ed.) FSE 2006. LNCS, vol. 4047, pp. 163–178. Springer, Heidelberg (2006)
17. Maximov, A., Biryukov, A.: Two trivial attacks on Trivium. eSTREAM, ECRYPT Stream Cipher Project, Repor 2007/006 (2007), http://www.ecrypt.eu.org/stream
18. Meier, W., Staffelbach, O.: The self-shrinking generator. In: De Santis, A. (ed.) EUROCRYPT 1994. LNCS, vol. 950, pp. 205–214. Springer, Heidelberg (1995)

19. Noras, J.: Fast pseudorandom sequence generators: Linear feedback shift registers, cellular automata, and carry feedback shift registers. Technical Report 94, Univ. Bradford Elec. Eng. Dept., Bradford, U.K (1997)
20. Shaked, Y., Wool, A.: Cryptanalysis of the bluetooth E_0 cipher using OBDDs. Technical report, Cryptology ePrint Archive, Report 2006/072 (2006)
21. Wegener, I.: Branching Programs and Binary Decision Diagrams: Theory and Applications. SIAM Monographs on Discrete Mathematics and Applications (2000)

A A Proof for Lemma 3

Proof. The definitions of Q_m and R_m imply that $P_m = Q_m \wedge R_m$ for $n_{\min} < m \leq \lceil \alpha^{-1} n \rceil$, and therefore $w(P_m) \leq w(Q_m) \cdot w(R_m)$. Under the BDD assumption we obtain

$$w(P_m) \leq w(Q_m) \cdot 2^{p \cdot |\mathrm{CP}(m)|} \ . \tag{2}$$

On the other hand, Lemma 2 implies that $w(P_m) \leq |P_m^{-1}(1)| \approx m \cdot 2^{n^* - \alpha m}$ for $n^* = |\mathrm{IP(m)}|$ and $n_{\min} < m \leq \lceil \alpha^{-1} n \rceil$, which means

$$w(P_m) \leq 2^{n^* - \alpha m} = 2^{(1-\alpha)n^* - \alpha \cdot |\mathrm{CP}(m)|} \ . \tag{3}$$

Combining eqns. (2) and (3), we obtain for $n_{\min} < m \leq \lceil \alpha^{-1} \rceil n$

$$\begin{aligned} w(P_m) &\leq w(Q_m) \min\{2^{p \cdot |\mathrm{CP}(m)|}, 2^{(1-\alpha)n^* - \alpha \cdot |\mathrm{CP}(m)|}\} \\ &= w(Q_m) \min\{2^{p \cdot r(n^*)}, 2^{(1-\alpha)n^* - \alpha r(n^*)}\} \text{ with } r(n^*) = m - |\mathrm{IP}(m)| \\ &\leq w(Q_m) \cdot 2^{p \cdot r^*(n^*)} \ , \end{aligned}$$

where $r^*(n^*)$ denotes the solution of $p \cdot r(n^*) = (1-\alpha)n^* - \alpha r(n^*)$. We obtain $r^*(n^*) = \frac{1-\alpha}{p+\alpha} n^*$ and hence $w(P_m) \leq w(Q_m) 2^{\frac{p(1-\alpha)}{p+\alpha} n^*}$. With $n_{\min} < m \leq \lceil \alpha^{-1} n \rceil$ and therefore $w(Q_m) \in m^{O(1)} \subseteq n^{O(1)}$ and $n^* = |\mathrm{IP}(m)| \leq n$, we get

$$w(P_m) \leq n^{O(1)} 2^{\frac{p(1-\alpha)}{p+\alpha} n} \text{ for all } n_{\min} < m \leq \lceil \alpha^{-1} n \rceil \ ,$$

which concludes the proof. □

B Trivium Algorithm

From a starting state $(s_1, \ldots, s_{288})$ the algorithm produces keystream bits z_t as follows.

for $t = 0$ to $N - 1$ **do**
  $t_1 \leftarrow s_1 \oplus s_{28}$
  $t_2 \leftarrow s_{94} \oplus s_{109}$
  $t_3 \leftarrow s_{178} \oplus s_{223}$
  $z_t \leftarrow t_1 \oplus t_2 \oplus t_3$
  $u_1 \leftarrow t_1 \oplus s_2 s_3 \oplus s_{100}$
  $u_2 \leftarrow t_2 \oplus s_{95} s_{96} \oplus s_{202}$

$u_3 \leftarrow t_3 \oplus s_{179}s_{180} \oplus s_{25}$
$(s_1, \ldots, s_{93}) \leftarrow (s_2, \ldots, s_{93}, u_3)$
$(s_{94}, \ldots, s_{177}) \leftarrow (s_{95}, \ldots, s_{177}, u_1)$
$(s_{178}, \ldots, s_{288}) \leftarrow (s_{179}, \ldots, s_{288}, u_2)$
end for

C Grain Algorithm

$$s_{t+128} = s_t \oplus s_{t+7} \oplus s_{t+38} \oplus s_{t+70} \oplus s_{t+81} \oplus s_{t+96}$$
$$b_{t+128} = s_t \oplus b_t \oplus b_{t+26} \oplus b_{t+56} \oplus b_{t+91} \oplus b_{t+96} \oplus b_{t+3}b_{t+67} \oplus b_{t+11}b_{t+13} \oplus b_{t+17}b_{t+18} \oplus b_{t+27}b_{t+59} \oplus b_{t+40}b_{t+48} \oplus b_{t+61}b_{t+65} \oplus b_{t+68}b_{t+84} \ .$$

In each clock, an output bit z_t is derived by

$$z_t = \left(\bigoplus_{j \in A} b_{t+j} \right) \oplus b_{t+12}s_{t+8} \oplus s_{t+13}s_{t+20} \oplus b_{t+95}s_{t+42} \oplus s_{t+60}s_{t+79} \oplus b_{t+12}b_{t+95}s_{t+95}$$

with $A = \{2, 15, 36, 45, 64, 73, 89\}$.

D F-FCSR-H Algorithm

At each clock, the generator uses the following static filter to extract a pseudo-random byte:

$$F = (\text{ae985dff } 26619\text{fc5 } 8623\text{dc8a af46d590 } 3\text{dd4254e})_{16}$$

The filter splits into 8 subfilters (subfilter j is obtained by selecting the bit j in each byte of F)

$$F_0 = (0011\ 0111\ 0100\ 1010\ 1010)_2 \ , \ F_4 = (0111\ 0010\ 0010\ 0011\ 1100)_2$$
$$F_1 = (1001\ 1010\ 1101\ 1100\ 0001)_2 \ , \ F_5 = (1001\ 1100\ 0100\ 1000\ 1010)_2$$
$$F_2 = (1011\ 1011\ 1010\ 1110\ 1111)_2 \ , \ F_6 = (0011\ 0101\ 0010\ 0110\ 0101)_2$$
$$F_3 = (1111\ 0010\ 0011\ 1000\ 1001)_2 \ , \ F_7 = (1101\ 0011\ 1011\ 1011\ 0100)_2$$

The bit b_i (with $0 \leq i \leq 7$) of each extracted byte is expressed by

$$b_i = \bigoplus_{j=0}^{19} f_i^{(j)} a_{8j+i} \quad \text{where } F_i = \sum_{j=0}^{19} f_i^{(j)} 2^j \ ,$$

and where the a_k are the bits contained in the main register.

The cipher is initialized with an 80-bit key and an IV of length $32 \leq v \leq 80$ according to the following procedure.

1. The main register a is initialized with the key and the IV:

$$a := K + 2^{80} \cdot \mathrm{IV} = (0^{80-v}||\mathrm{IV}||K)$$

2. All carry cells are initialized to 0.

$$C := 0 = (0^{82}) \ .$$

3. A loop is iterated 20 times. Each iteration of this loop consists in clocking the FCSR and then extracting a pseudorandom byte S_i, $0 \leq i \leq 19$, using the filter F.
4. The main register a is reinitialized with these bytes:

$$a := \sum_{i=0}^{19} S_i = (S_{19}|| \cdots ||S_1||S_0) \ .$$

5. The FCSR is clocked 162 times (output is discarded in this step).

After the setup phase, the output stream is produced by repeating the following two steps as many times as needed.

1. Clock the FCSR.
2. Extract one pseudorandom byte $(b_0, \ldots, b_7)$ according to the linear filter described above.