Electronic Colloquium on Computational Complexity, Report No. 78 (2001)

BDD-based Cryptanalysis of Keystream Generators

Matthias Krause

Theoretische Informatik, Universität Mannheim, 68131 Mannheim, Germany
e-mail: krause@informatik.uni-mannheim.de

Abstract. Many of the keystream generators which are used in practice are LFSR-based in the sense that they produce the keystream according to a rule $y = C(L(x))$, where $L(x)$ denotes an internal linear bitstream, produced by a small number of parallel linear feedback shift registers (LFSRs), and C denotes some nonlinear compression function. We present an $n^{O(1)}2^{(1-\alpha)/(1+\alpha)n}$ time bounded attack, the FBDD-attack, against LFSR-based generators, which computes the secret initial state $x \in \{0,1\}^n$ from cn consecutive keystream bits, where α denotes the rate of information, which C reveals about the internal bitstream, and c denotes some small constant. The algorithm uses Free Binary Decision Diagrams (FBDDs), a data structure for minimizing and manipulating Boolean functions. The FBDD-attack yields better bounds on the effective key length for several keystream generators of practical use, so a $0.656n$ bound for the self-shrinking generator, a $0.6403n$ bound for the A5/1 generator, used in the GSM standard, a $0.6n$ bound for the E_0 encryption standard in the one level mode, and a $0.8823n$ bound for the two-level E_0 generator used in the Bluetooth wireless LAN system.

1 Introduction

A keystream generator is a finite automaton, E, determined by a set Q of inner states, a state transition function $\delta_E : Q \longrightarrow Q$, and an output function $a_E : Q \longrightarrow \{0,1\}$. The usual case is that $Q = \{0,1\}^n$ for some integer $n \geq 1$, n is called the keylength of E. Starting from an initial state $x^0 \in Q$, in each time unit i, E outputs a key bit $y_i = a_E(x^i)$ and changes the inner state according to $x^{i+1} = \delta_E(x^i)$. For each initial state $x \in \{0,1\}^n$ we denote by

$$y = E(x)$$

the keystream $y = y_0, y_1, \ldots$ produced by E when starting on x.

Keystream generators are designed for fast online encryption of bitstreams which have to pass an insecure channel. A standard application is to ensure the over-the-air privacy of communicating via mobile cellular telephones. A plaintext bit stream $p_0, p_1, p_2, \ldots$ is encrypted into a ciphertext bitstream $e_0, e_1, e_2, \ldots$ via the rule

$$e_i = p_i \oplus y_i,$$

where $y = E(x)$. The legal receiver knows x and can decrypt the bitstream using the same rule. The only secret information is the initial state x, which is exchanged before starting the transmission using a suitable key-exchange protocol. It is usual to make the pessimistic assumption that an attacker knows not only the encrypted bitstream, but even some short piece of the plaintext, and, therefore, can easily compute some piece of the keystream. Consequently, the security of keystream generators has to be based on the assumption that there is no feasible way to compute the secret initial state x from $y = E(x)$. Observe that the trivial exhaustive search attack needs time $n^{O(1)}2^n$.

ISSN 1433-8092

In this paper we suggest a new type of attack against keystream generators, which we will call FBDD-attack, and show that LFSR-based keystream generators are vulnerable against FBDD-attacks. We will call a generator to be LFSR-based if it consists of two components, a linear bitstream generator L which generates for each initial state $x \in \{0,1\}^n$ a linear bitstream $L(x)$ by one or more parallel LFSRs, and a compressor C which transforms the internal bitstream into the output keystream $y = C(L(x))$. Due to the ease of implementing LFSRs in hardware, and due to the nice pseudorandomness properties of bitstreams generated by maximal length LFSRs, many keystream generators occuring in practice are LFSR-based.

FBDD is the abbreviation for *free binary decision diagrams*, a data structure for representing and manipulating Boolean function, which were introduced by *Gergov* and *Meinel* in [8] and *Sieling* and *Wegener* in [14]. Due to specific algorithmic properties, FBDDs and in particular ordered BDDs (OBDDs), a special kind of FBDDs, became a very usefull tool in the area of automatic hardware design and verification (see also the paper of *Bryant* [4] who initiated the study of graph-based data structures for Boolean function manipulation). The important properties of FBDDs are that they can be efficiently minimized, that they allow an efficient enumeration of all satisfying assignments, and that the minimal FBDD-size of a Boolean function is not much larger than the number of satisfying assignments. We show that these properties can also be successfully used for cryptanalysis. The problem of finding a secret key x fulfilling $y = E(x)$ for a given encryption algorithm E and a given ciphertext y can be reduced to finding the minimal FBDD P for the decision if x fulfils $y = E(x)$. If the length of y is close to the unicity distance of E then P is small, and x can be efficiently computed from P. The main result of this paper is that the special structure of LFSR-based keystream generators implies a nontrivial dynamic algorithm for computing this FBDD P.

In particular, the weakness of LFSR-based keystream generators is that the compressor C has to produce the keystream in an online manner. For getting a high bitrate, C should use only a small memory, and should consume only a few new internal bits for poducing the next output bit. These requirements imply that the decision if an internal bitstream z generates a prefix of a given keystream y via C can be computed by small FBDDs. This allows to compute dynamically a sequence of FBDDs P_m, $m \geq n$, which test a given initial state $x \in \{0,1\}^n$ whether $C(L_{\leq m}(x))$ is prefix of y, where $L_{\leq m}(x)$ denotes the first m bits of the internal linear bitstream generated via L on x. On average, the solution becomes unique for $m \geq \lceil \alpha^{-1} n \rceil$, where α denotes the rate of information which C reveals about the internal bitstream. The FBDDs P_m are small at the beginning and again small if m approaches $\lceil \alpha^{-1} n \rceil$, and we will show that all intermediate FBDDs have a size of at most $n^{O(1)} 2^{(1-\alpha)/(1+\alpha)n}$. For all m the FBDD P_m has to read the first $\lceil \gamma m \rceil$ bits of the keystream, where γ denotes the best case compression ratio of C. Thus, our algorithm computes the secret initial state x from the first $\lceil \gamma \alpha^{-1} n \rceil$ bits of $y = E(x)$. Observe that $\gamma = \alpha$ if C consumes always the same number of internal bits for producing one output bit, and $\alpha < \gamma$ if not. It holds $\gamma \alpha^{-1} \leq 2.5$ in all our examples. Note that for gaining a high bit-rate, α and γ should be as large as possible. Our results say that a higher bit-rate has to be paid by a loss of security.

One advantage of the FBDD-attack is that it is a *short-keystream* attack, i.e., the number of keybits needed for computing the secret initial state $x \in \{0,1\}^n$ is at most cn for some small constant $c \geq 1$. We apply the FBDD-attack to some of the keystream generators which are most intensively discussed in the current literature, the A5/1 generator which is used in the GSM standard, the self-shrinking generator, and the E_0-encryption standard, which is included in the Bluetooth wireless LAN system. For all theses ciphers, the FBDD-attack has a better time behaviour than *all* short-keystream attacks known before. In some cases there have been obtained *long-keystream* attacks which have a better time behaviour. They use a time-memeory tradeoff technique suggested by *Golić* [7] and are based on the assumption that a long piece of keystream of length 2^{dn}, $d < 1$ some constant, is available. We give an overview on previous results and the relations to our results in section 7.

The paper is organized as follows. In section 2 we give some formal definitions concerning LFSR-based keystream generators and present the keystream generators which we want to cryptanalyze. In section 3 FBDDs are introduced. In section 4 we derive FBDD-relevant properties of LFSR-based keystream generators. In section 5 we derive the relevant parameters of LFSR-based keystream generators and formulate the main result. Our cryptanalysis algorithm is presented in section 6 and is applied to particular generators in section 7. Due to space restrictions we had to put some of the calculations and FBDD-constructions into an appendix section.

At the first glance it may seem contradictory that we consider practical ciphers like the A5/1 generator with variable keylength n. But observe that definitions of LFSR-based keystream generators, even if they originally were designed for fixed keylength, can be generalized to variable keylength in a very natural way, simply by considering the internal LFSRs to have variable length. Considering variable keylength n allows to evaluate the security of ciphers in terms of how many polynomial time operations are necessary for breaking the cipher. This 'rough' way of security evaluation is sufficient in our context, since the aim of this paper is to present only the general algorithmic idea of the FBDD-attack, to give a rather rough estimation of the time behaviour, and to show the inherent weakness of LFSR-based generators. For practical implementations of FBDD attacks against real-life generators much more effort has to be invested for making the particular polynomial time operations as efficient as possible, see the discussion in section 8.

2 LFSR-based Keystream Generators

Let us call a keystream generator to be LFSR-based if the generation rule $y = E(x)$ can be written as

$$y = C(L(x)),$$

where L denotes a linear bitstream generator consisting of one or more LFSRs, and $C : \{0,1\}^* \longrightarrow \{0,1\}^*$ denotes a nonlinear compression function, which transforms the internal linear bitstream $L(x)$ into the nonlinear (compressed) output keystream $y = C(L(x))$. [1] Formally, an n-LFSR is a device which produces a bitstream

$$L(x) = L_0(x), L_1(x), \ldots, L_m(x), \ldots$$

on the basis of a public string $c = (c_1, \ldots, c_n) \in \{0,1\}^n$, the generator polynomial, and a secret initial state $x = (x_0, \ldots, x_{n-1}) \in \{0,1\}^n$, according to the relation $L_i(x) = x_i$ for $0 \le i \le n-1$ and

$$L_m(x) = c_1 L_{m-1}(x) \oplus c_2 L_{m-2}(x) \oplus \ldots \oplus c_n L_{m-n}(x) \qquad (1)$$

for $m \ge n$. Observe that for all $m \ge 1$, $L_m(x)$ is a GF(2)-linear Boolean function in $x_0, \ldots, x_{n-1}$ which can be easily determined via iteratively applying (1).

A linear bitstream generator L of keylength n is defined to be an algorithm which, for some $k \ge 1$, generates a linear bitstream $L(x)$

$$L(x) = L_0(x), L_1(x), L_2(x), \ldots$$

[1] C compresses the internal bit-stream in an online manner, i.e., $C(z')$ is prefix of $C(z)$ if z' is prefix of z, for all $z, z' \in \{0,1\}^*$. This justifies to write $y = C(L(x))$ despite of the fact that $L(x)$ is assumed to be infinitely long.

by k parallel LFSRs $L^0, \dots, L^{k-1}$ of keylengths $n_0, \dots, n_{k-1}$, where $n = n_0 + \dots + n_{k-1}$. The initial states $x \in \{0,1\}^n$ for L are formed by the initial states $x^r \in \{0,1\}^{n_r}$, $r = 0, \dots, k-1$, of $L^0, \dots, L^{k-1}$. L produces in each time unit $j \geq 0$ the bit $L_j(x)$ according to the rule

$$L_j(x) = L_s^r(x^r),$$

where $r = j \mod k$ and $s = j \operatorname{div} k$. Observe that for all $j \geq 0$ $L_j(x)$ is a GF(2)-linear function in x.

The motivation for taking LFSRs as building blocks for keystream generators is that they can be easily implemented using n register cells connected by a feedback channel. Moreover, if the generator polynomial is primitive, they produce bit streams with nice pseudorandomness properties (maximal period, good auto correlation and local statistics). See, e.g., the monograph by *Golomb* [9] or the article by *Rueppel* [13] for more about the theory of shift register sequences.

Clearly, LFSR-sequences alone do not provide any cryptographic security. Thus, the aim of the compression function $C : \{0,1\}^* \longrightarrow \{0,1\}^*$ is to destroy the low linear complexity of the internal linear bit stream while preserving its nice pseudorandomness properties. Many keystream generators occuring in practice are LFSR-based in the above sense. In this paper we investigate the following LFSR-based generators.

The Connekt-k construction combines k parallele LFSRs $L^0, \dots, L^{k-1}$ with an appropriate connection function $c : \{0,1\}^k \longrightarrow \{0,1\}$. The keystream $y = y_0, y_1, y_2, \dots$ is defined via the rule

$$y_j = c(L_j^0(x^0), \dots, L_j^{k-1}(x^{k-1})),\ j \geq 0,$$

where x^r denotes the initial state for L^r, for $r = 0 \dots, k-1$. The Connect-k construction is one of the classical constructions of nonlinear keystream generators which use LFSRs as building blocks, see again [9] and [13] for more detailed considerations of such constructions.

The Self-Shrinking Generator was introduced by *Meier* and *Staffelbach* in [12]. It consists of only one LFSR L. The compression is defined via the shrinking function

$$shrink : \{0,1\}^2 \longrightarrow \{0,1,\varepsilon\},$$

defined as $shrink(ab) = b$ if $a = 1$, and $shrink(ab) = \varepsilon$, the empty word, otherwise. The shrinking-function can be extended to bit-strings of even length r as

$$shrink(z_0 z_1 \dots z_{r-1}) = y_0 y_1 \dots y_{r/2-1},$$

where $y_i = shrink(z_{2i} z_{2i+1})$ for $i = 0, \dots, r/2 - 1$. For each initial state x for L, the self-shrinking generator produces the keystream y according to

$$y = shrink(L(x)).$$

The E_0 Generator is the keystream generator used in the Bluetooth wireless LAN system [3]. It is defined as $E_0(x) = C(L(x))$, where the linear bitstream generator L of E_0 consists of 4 LFSRs $L^0, \dots, L^3$. The compression is organized by a finite automaton M with external input alphabet $\{0,1,2\}$, state space $Q = \{0,1,\dots,15\}$ and output alphabet $\{0,1\}$, which is defined by an output function $a : Q \times \{0,1,2\} \longrightarrow \{0,1\}$ and a state transition function $\delta : Q \times \{0,1,2\} \longrightarrow Q$. The exact specification of M is published in [3] but does not matter for our purpose and is therefore omitted.

The compression $C(z) = y = y_0 y_1 \dots y_{m-1}$ of an internal bit-stream

$$z = z_0^0 z_0^1 z_0^2 z_0^3 z_1^0 z_1^1 z_1^2 z_1^3 \dots z_{m-1}^0 z_{m-1}^1 z_{m-1}^2 z_{m-1}^3$$

is defined as $y_j = a(q_j, s_j) \oplus t_j$, where $s_j = (z_j^0 + z_j^1 + z_j^2 + z_j^3)$ div 2 and $t_j = (z_j^0 + z_j^1 + z_j^2 + z_j^3)$ mod 2, for all $0 \leq j \leq m-1$. The actual inner state is updated in each cycle according to the rule $q_{j+1} = \delta(q_j, s_j)$, where q_0 denotes the initial state of M. In practice, the E_0 generator is used with key length 128, the four LFSRs have lengths 39, 33, 31, 25.

The E_0 Encryption Standard (Two-Level Mode). In the Bluetooth system, the keystream is generated via a generator E_0^2 (of key length n) which combines two E_0 devices of internal keylength $N \geq n$ in the following way.

For $x \in \{0,1\}^n$ it holds $y = E_0^2(x) = E_0(z)$, where z denotes the prefix of length N of $E_0(u)$, and where

$$u = (x_0, \dots, x_{n-1}, U_n(x) \dots, U_{N-1}(x)).$$

U_i, $i = n \dots N-1$, are public GF(2)-linear functions in $(x_0, \dots, x_{n-1})$. In practice, the string u results from putting n secret bits together with $N-n$ known dummy bits into the LFSRs and running them a certain number of steps. The Bluetooth system uses $N = 128$, and n can be chosen as 8, 16, 32, or 64. It is a bit more involved to write the E_0^2 generator in a $y = C(L(x))$ fashion. We will do this in the appendix, subsection A.2. The reason for choosing a larger internal key length N is to achieve a larger effective key length in n.

The A5/1 generator is used in the GSM standard of mobile telephones. The definition was discovered by *Briceno et. al.* [5] via reverse engineering. The A5/1 generator of key-length n consists of 3 LFSRs L^0, L^1, and L^2 of key-lengthes n_0, n_1, and n_2. In each time step i, the output key bit y_i is the XOR of the actual output bits of the 3 LFSRs. A clock control decides in each timestep which of the 3 LFSRs are shifted, and which not. The clock control takes for all $k \in \{0,1,2\}$ a control value c_k from the $N_k - th$ register cell of L^k, and computes the control value $m = maj_3(c_0, c_1, c_2)$. [2] LFSR L^k is shifted if $m = c_k$, for $k = 0,1,2$. The control positions N_k are fixed and fulfil $N_k \in \left\{\lceil \frac{n_k}{2} \rceil - 1, \lceil \frac{n_k}{2} \rceil \right\}$.

This keystream generation rule can be written down in a $y = C(L(x))$ fashion in the following way. Given an internal bitstream

$$z = (z_0^0, z_0^1, z_0^2, \dots, z_m^0, z_m^1, z_m^2, \dots)$$

the keystream $y = C(z)$ is defined as follows. In each timestep, C holds 3 output positions $i[1], i[2], i[3]$ and 3 control positions $j[1], j[2], j[3]$. C outputs $x_{i[1]}^1 \oplus x_{i[2]}^2 \oplus x_{i[3]}^3$, computes the new control value $m = maj_3(x_{j[1]}^1, x_{j[2]}^2, x_{j[3]}^3)$, and updates the i- and j-values via $i[k] := i[k] + 1$ and $j[k] := j[k] + 1$, for those $k \in \{0,1,2\}$ for which $m = x_{j[k]}^k$. The output positions are initialized by 0. The control positions are initialized by N_1, N_2, N_3. Note that in the GSM standard the A5/1 generator is used with key length 64, the 3 LFSRs have lengthes 19, 22 and 23

3 Binary Decision Diagrams (BDDs)

For m a natural number let X_m denote the set of m Boolean variables $\{x_0, \dots, x_{m-1}\}$. A BDD P over X_m is an acyclic directed graph with inner nodes of

[2] maj_3 is defined to output $c \in \{0,1\}$ iff at least 2 of its 3 arguments have value c.

outdegree 2, a distinguished inner node of indegree 0, the source, and two sink nodes of outdegree 0, one 0-sink and one 1-sink. All inner nodes, i.e. nodes of outdegree > 0, are labelled with queries $x_i?$, $0 \leq i \leq m-1$, and are left by one edge labelled 0 (corresponding to the answer $x_i = 0$) and one edge labelled 1 (corresponding to the answer $x_i = 1$).

Each assignment b to the X_m-variables defines a unique computational path in P, which will be called the b-path in P. The b-path starts at the source, answers always b_i on queries $x_i?$ and, thus, leads to a unique sink. The label of this sink is defined to be the output $P(b) \in \{0,1\}$ of P on input $b \in \{0,1\}^m$. We denote by $One(P) \subseteq \{0,1\}^m$ the set of inputs accepted by P,

$$One(P) = \{b \in \{0,1\}^m;\ P(b) = 1\}.$$

Each BDD P over X_m computes a unique function $f : \{0,1\}^m \longrightarrow \{0,1\}$, by $f(b) = 1 \iff b \in One(P)$. The size of P, $|P|$, is defined to be the number of inner nodes of P. Two BDDs are called equivalent if they compute the same function.

We call an BDD P to be a *free* binary decision diagram (FBDD) if along each computational path in P each variable occurs at most once. In [8] and [14] it was observed that FBDDs can be efficiently minimized with respect to all equivalent FBDDs which read the input variables in an equivalent order. The equivalence of orders of reading the input variables is expressed by using the notion of graph orderings.

Definition 1. *A graph ordering G of X_m is an FBDD over X_m with only one (unlabelled) sink, for which on each path from the root to the sink all m variables occur.*

Graph orderings are not designed for computing Boolean functions. Their aim is to define for each assignments $b = (b_0, \ldots, b_{m-1})$ to X_m a unique variable ordering $\pi_G(b) = (x_{i_1(b)}, \ldots, x_{i_m(b)})$, namely the ordering in which the variables are requested along the unique b-path in G.

Definition 2. *An FBDD is called G-driven, for short, G-FBDD, if the ordering in which the variables are requested along the b-path in P respects $\pi_G(b)$, for all assignments b. I.e., there do not exist assignments b, variables x_i and x_j such that x_i is requested above x_j at $\pi_G(b)$, but below x_j at the b-path in P.*

A special, extensively studied variant of FBDDs are Ordered Binary Decision Diagrams (OBDDs). An FBDD P is called OBDD with variable ordering π (for short π-OBDD) if all pathes in P respect π.

We need the following nice algorithmic properties of graph-driven FBDDs. Let $f, g : \{0,1\}^m \longrightarrow \{0,1\}$ be Boolean functions, let G be a graph ordering for X_m, and let P and Q be G-driven FBDDs for f and g, respectively.

Property 3.01 *There is an algorithm MIN which computes from P in time $O(|P|)$ the (uniquely defined) minimal G-driven FBDD* $\min(P)$ *for f.*

Property 3.02 *It holds that* $|\min(P)| \leq m|One(P)|$.

Property 3.03 *There is an algorithm $SYNTH$ which computes in time $O(|P||Q||G|)$ a G-driven FBDD $P \wedge Q$, $|P \wedge Q| \leq |P||Q||G|$, which computes $f \wedge g$.*

Property 3.04 *There is another algorithm SAT which enumerates all elements in $One(P)$ in time $O(|One(P)||P|)$.*

See, e.g., the book by *Wegener* [15] for a detailed description and analysis of the OBDD- and FBDD-algorithms. FBDDs together with the procedures MIN, *SYNTH* and *SAT* will be the basic data structure used in our cryptanalysis. FBDDs for a given decision problems $F \subseteq \{0,1\}^*$ can be constructed using the following folklore result, which, for instance, is given in the monograph by *Meinel* [10].

Theorem 1. *Each $s(n)$-space bounded algorithm for F can be efficiently transformed into a sequence of $2^{O(s(n)+\log(n))}$-space bounded BDDs for F. Moreover, if the algorithm reads each input bit at most once then the resulting BDDs are FBDDs.* □

Thus, one way to prove the existence of polynomial size FBDDs for given Boolean functions is to look for logarithmically space bounded read-once algorithms.

4 FBDD-Aspects of Key-Stream Generators

Let E be a LFSR-based keystream generator of key-length n with linear keystream generator L and compression function $C : \{0,1\}^* \longrightarrow \{0,1\}^*$. Let $x \in \{0,1\}^n$ denote an initial state for L.

Definition 3. *For all $m \geq 1$ let G_m^C denote the graph ordering, which assigns to each internal bitstream z the order in which C reads the first m bits of z.*

Observe that for the E_0 generator, the self-shrinking generator, as well as for Connect-k generators, the order in which the compressor reads the internal bits does not depend of the internal bitstream itself, i.e., G_m^C has size m and G_m^C-driven FBDDs are OBDDs. But in the case of the A5/1 generator, this order is governed by the clock control, and can be different for different inputs. The efficiency of our cryptanalysis algorithm is based on the following FBDD assumption on E.

FBDD Assumption. The graph ordering G_m^C has polynomial size in m. Moreover, for arbitrary keystreams y, the minimal G_m^C-driven FBDDs which decide for $z \in \{0,1\}^m$ whether $C(z)$ is prefix of y have polynomial size in m.

It is quite easy to see that the compression function of a Connect-k generators, defined by a function $c : \{0,1\}^k \longrightarrow \{0,1\}$, fulfils the FBDD-assumption. The compressor reads the internal bits in the canonical order $\pi = 0, 1, 2, 3, \ldots$. Polynomial size (even linear size) π-OBDDs which decide whether $z \in \{0,1\}^m$ generates the first $\lfloor m/k \rfloor$ bits of a given keystream y via c can be constructed, via Theorem 1, according to the following algorithm

```
1. For $j := 0$ to $\lfloor m/k \rfloor$
2.     if $c(z_j^0, \ldots, z_j^{k-1}) \neq y_j$ then stop(0)
3. stop(1)
```

which is obviously $O(\log(m))$-space bounded.

Polynomial (even quadratic) size π-OBDDs which decide for $z \in \{0,1\}^m$ whether $shrink(z)$ is prefix of a given keystream y can be constructed, via Theorem 1, according to the following algorithm

```
1. $k := 0$, $j := 0$
2. while $j < m - 1$
3.     if $z_j = 0$
4.         then $j := j + 2$
5.         else
6.           if $z_{j+1} = y_k$
```

```
7.          then $j := j + 2$, $k := k + 1$
7.          else stop(0)
8. stop(1)
```

which is obviously $O(\log(m))$-space bounded. The FBDD constructions for all the E_0-, the E_0^2-, and the A5/1 generator are given in the appendix.

We still need to estimate the size of FBDDs which decide whether a given $z \in \{0,1\}^m$ is a linear bit-stream.

Lemma 1. *For all $m \geq n$, the decision whether $z \in \{0,1\}^m$ is generated via linear bitstream generator L of keylength n can be computed by a G_m^C-driven FBDD of size at most $|G_m^C| 2^{m-n}$.*

Proof: Let V_m denote the set of inner nodes of G_m^C. We construct a G_m^C-driven FBDD R_m with the set $W_m = V_m \times \{0,1\}^{m-n}$ of inner nodes.

For all initial states $x \in \{0,1\}^n$ and all internal positions $j, n \leq j \leq m-1$, write $L_j(x)$ as

$$L_j(x) = \bigoplus_{k=0}^{n-1} L_{k,j} x_k.$$

G_m^C ensures that x_k is always read before x_j if $L_{k,j} = 1$.

Let the root of R_m be the node $(v_0, \vec{0})$ where v_0 denotes the root of G_m^C. Let all nodes (v,b) have the same label as v does in G_m^C. The edges of R_m are defined according to the following rules. Let $v \in V_m$ and $b = (b_n, \ldots, b_{m-1}) \in \{0,1\}^{m-n}$ be arbitrarily fixed. For $c \in \{0,1\}$ let $v(c)$ be the c-successor of v in G_m^C. We have to distinguish two cases.

- v is labelled with some x_k, $0 \leq k \leq n-1$. Then, for all $c \in \{0,1\}$, the c-successor of (v,b) is $(v(c), b(c))$, where $b(c) = (b_0 \oplus L_{k,n}c, \ldots, b_{r-1} \oplus L_{k,m-1}c)$.
- v is labelled with some x_j, $n \leq j \leq m-1$. Then, for all $c \in \{0,1\}$, if $b_{j-n} \neq c$, the c-successor of (v,b) is the 0-sink. If $b_{j-n} = c$ and $v(c)$ is the *-sink, then let the c-successor of (v,b) be the 1-sink. Otherwise let the c-successor of (v,b) be $(v(c), b)$.

It can be easily checked that R_m (after removing non-reachable nodes) matches all requirements of the Lemma. □

5 The Main Result

We fix an LFSR-based keystream generator of key-length n with linear bit-stream generator L and a compression function C. We assume that for all $m \geq 1$ the probability that $C(z)$ is prefix of y for a randomly chosen and uniformly distributed $z \in \{0,1\}^m$ is the same for all keystreams y. Observe that all generators occuring in this paper have this property. Let us denote this probability by $p_C(m)$.

The cost of our cryptanalysis algorithm depends on two parameters of C. The first is the information rate (per bit) which a keystream y reveales about the first m bits of the underlying internal bitstream. It can be computed as

$$\frac{1}{m} I(Z^{(m)}, Y) = \frac{1}{m}\left(H(Z^{(m)}) - H(Z^{(m)}|Y)\right) =$$

$$= \frac{1}{m}\left(m - \log(p_C(m)2^m)\right) = -\frac{1}{m}\log(p_C(m)). \quad (2)$$

where $Z^{(m)}$ denotes a random $z \in \{0,1\}^m$ and Y a random keystream.

As the compression algorithm computes the keystream in an online manner, the time difference between two succeeding key bits should be small in the average, and not vary too much. This implies the following partition rule: Each internal bitstream z can be divided into consecutive elementary blocks $z = z^0 z^1 \dots z^{s-1}$, such that $C(z) = y_0 y_1 \dots y_{s-1}$ with $y_j = C(z^j)$ for all $j = 0, \dots, s-1$, and the average length of the elementary blocks is a small constant. This partition rule implies that $p_C(m)$ can be supposed to behave as $p_C(m) = 2^{-\alpha m}$, for a constant $\alpha \in (0, 1]$. Due to (2), α coincides with the information rate of C.

The second parameter of C is the maximal number of output bits which C produces on internal bitstreams of length m. Due to the partition rule, this value can be supposed to behave as γm, for some constant $\gamma \in (0, 1]$. We call γ to be the (best case) compression ratio of C.

Observe that if C always reads the same number k of internal bits for producing one output bit, then $\alpha = \gamma = \frac{1}{k}$. If this number is not a constant then α can be obtained by the formulae

$$2^{-\alpha m} \;=\; p_C(m) \;=\; \sum_{i=0}^{\lceil \gamma m \rceil} 2^{-i} Prob_z\left[|C(z)| = i\right], \tag{3}$$

where z denotes a random, uniformly distributed element from $\{0,1\}^m$. Observe that (3) yields $\gamma \geq \alpha$, i.e. $\gamma\alpha^{-1} \geq 1$.

For all $x \in \{0,1\}^n$ and $m \geq 1$ let $L_{\leq m}(x)$ denote the first m bits of $L(x)$. Note the following design criterion for well-designed keystream generators.

Pseudorandomness Assumption For all keystreams y and all $m \leq \lceil \alpha^{-1} n \rceil$ it holds that

$$Prob_z\left[C(z) \text{ is prefix of y}\right] \;\approx\; Prob_x\left[C(L_{\leq m}(x)) \text{ is prefix of y}\right],$$

where z and x denote uniformly distributed random elements from $\{0,1\}^m$ and $\{0,1\}^n$, respectively.

Lemma 2. *If the keystream generator fulfils the above pseudorandomness assumption then for all keystreams y and $m \leq \alpha^{-1} n$ there are approximately $2^{n-\alpha m}$ initial states x for which $C(L_m(x))$ is prefix of y.* □

Observe that a severe violation of the pseudorandomness assumption implies the possibility of attacking the cipher via a correlation attack. Our main result can now be formulated as

Theorem 2. *Let E be an LFSR-based keystream generator of key-length n with linear bit-stream generator L, and compression function C of information rate α and (best case) compression ratio γ. Let C and L fulfil the BDD- and the pseudorandomness assumption. Then there is an $n^{O(1)} 2^{(1-\alpha)/(1+\alpha)n}$-time bounded algorithm, which computes the secret initial state x from the first $\lceil \gamma\alpha^{-1} n \rceil$ consecutive bits of $y = C(L(x))$.*

As usual, we define the *effective key length* of a cipher of key length n to be the minimal number of polynomial time operations that are necessary to break the cipher. We obtain a bound of $\frac{1-\alpha}{1+\alpha} n$ for the effective key length of keystream generators which fulfil the above conditions.

6 The Algorithm

Let us fix n, E, L, C, α and γ as in Theorem 2. For all $m \geq 1$ let G_m denote the graph ordering defined by C on internal bitstreams of length m. Let y be an arbitrarily fixed keystream which was generated via E. For all $m \geq 1$ let Q_m denote a minimal G_m-FBDD which decides for $z \in \{0,1\}^m$ whether $C(z)$ is prefix of y. Observe that Q_m has to read the first $\lceil \gamma m \rceil$ bits of y. The FBDD-assumption yields that Q_m has polynomial size in m.

For $m \geq n$ let P_m denote the minimal G_m-driven FBDD which decides whether $z \in \{0,1\}^m$ is a linear bitstream generated via L and if $C(z)$ is prefix of y. Observe that by Property 3.03 and Lemma 1

Lemma 3. $|P_m| \leq |Q_m||G_m|^2 2^{m-n}$ *for all* $m \geq n$. □

The strategy of our algorithm is simple, it dynamically computes P_m for $m = n, \ldots, \lceil \alpha^{-1} n \rceil$. Lemma 2 implies that for $m = \lceil \alpha^{-1} n \rceil$ with high probability only one bit-stream z^* will be accepted by P_m. Due to property 3.04 this bit-stream can be efficiently computed. The first n components of z^* form the initial state that we are searching for.

For all $m \geq n$ let S_m denote a minimal G_m-FBDD which decides for $z = (z_0, \ldots, z_m)$ whether $z_m = L_m(z_0, \ldots, z_{n-1})$. From Lemma 1 we obtain that $|S_m| \leq 2|G_m|$. Now our algorithm can be formulated as

```
(1) $P := Q_n$
(2) For $m := n+1$ to $\lceil \alpha^{-1} n \rceil$
(3)         $P := \min(P \wedge Q_m \wedge S_{m-1})$
```

For the correctness of the minimization in step (3) observe that the definition of G_m implies that G_m is $G_{m'}$-driven for all $m' \geq m$. It follows from the definitions that for all $m \geq n$ P coincides with P_m after iteration m.

The FBDD-operation $\min(P \wedge Q_m \wedge S_{m-1})$ takes time $p(m)|P_{m-1}|$ for some polynomial p. Consequently, the running time of the algorithm can be estimated by

$$n^{O(1)} \max\{|P_m|,\ m \geq n\}.$$

Observe that on the one hand, by Lemma 3, $|P_m| \leq p'(m)2^{m-n}$ for some polynomial p', while on the other hand, by Property 3.02 and Lemma 2, $|P_m| \leq m|One(P_m)|$, where

$$|One(P_m)| \approx 2^{n-\alpha m} = 2^{(1-\alpha)n - \alpha(m-n)}.$$

Consequently, $|P_m|$ does not exceed $n^{O(1)}2^{r(n)}$, where $r(n)$ is the solution of

$$2^{r(n)} = 2^{(1-\alpha)n - \alpha r(n)}$$

which yields $r(n) = \frac{1-\alpha}{1+\alpha} n$. We have proved Theorem 2. □

7 Applications

We apply Theorem 2 to the keystream generators introduced in section 2. We suppose that these generators fulfill the pseudorandomness assumption, otherwise the running time estimations of our cryptanalysis hold on average. It remains to determine the information rate and the compression ratio, and to prove that the FBDD-assumption is true. For the Connect-k construction it holds $\alpha = \gamma = \frac{1}{k}$. The FBDD-assumption has shown to be true in section 4.

Theorem 3. *For all $k \geq 2$ and all stream ciphers E of key-length n which are a Connect-k construction, our algorithm computes the secret initial state $x \in \{0,1\}^n$ from the first n bits of $y = E(x)$ in time $n^{O(1)} 2^{\frac{k-1}{k+1}n}$.* □

This is, as far as we know, the first general upper bound on the effective key-length of the Connect-k construction.

For the E_0-encryption standard in the one-level mode we obtain $\alpha = \gamma = \frac{1}{4}$. That E_0 fulfils the FBDD-assumption is shown in the appendix-subsection A.1. We obtain

Theorem 4. *For the E_0-encryption standard with key-length n, our algorithm computes the secret initial state $x \in \{0,1\}^n$ from the first n bits of $y = E_0(x)$ in time $n^{O(1)} 2^{0.6n}$.* □

Observe that $128 \cdot 0.6 \approx 77$. Note that the best known attack against the E_0 generator of key length 128 was derived by *Fluhrer* and *Lucks* [6] and yields a tradeoff result between time and length of available keystream. It varies from $O(2^{84})$ necessary operations if 132 bit are available to $O(2^{73})$ necessary operations if 2^{43} bits are available. Observe that the FBDD attack yields $\lceil 128 \cdot 0.6 \rceil = 77$. Observe that due to our general assumption that the initial states of the LFSRs are the only secret information we suppose that the initial state of M is public. If this is not the case we have to run our algorithm 16 times, one round for each possible state of M.

Let us consider the E_0 generator in the two level mode with real key length n and internal key length $N \geq n$. Observe that E_0^2 needs $4 \cdot 4 = 16$ internal bits per key bit for producing the first $N/4$ key bits, while for later key bits only 4 internal bits per key bit are needed (see appendix, subsection A.2 for the details). Observe further that our algorithm reaches maximal FBDD-size in iteration $m^* := n + \frac{1-\alpha}{1+\alpha} n$. For $\alpha = 1/16$ this gives $m^* = 32/17n$. As $m^*/16 < N/4$ we obtain $\alpha = \gamma = 1/16$ as relevant parameters for our algorithm on E_0^2. That E_0^2 fulfils the FBDD-assumption is shown in the appendix, subsection A.2. Taking into account that $\frac{1-\alpha}{1+\alpha} = \frac{15}{17} \approx 0.8824$ we get

Theorem 5. *For the E_0^2-encryption generator with key-length n, our algorithm computes the secret initial state $x \in \{0,1\}^n$ from the first n bits of $y = E_0^2(x)$ in time $n^{O(1)} 2^{0.8824n}$.* □

As far as we know this is the first nontrivial upper bound on the key length of the E_2^0 generator.

Concerning the self-shrinking generator observe that for all even m and all keystreams y, $shrink(z)$ is prefix of y for exactly $3^{m/2}$ strings z of length m. We obtain an information rate $\alpha = 1 - \log(3)/2 \approx 0.2075$ for the self-shrinking generator by evaluating the relation $2^{-\alpha m} 2^m = 3^{m/2}$. The (best case) compression ratio of the self-shrinking generator is obviously 0.5. That the self-shrinking generator fulfils the FBDD-condition was already shown in section 4. Taking into account that for $\alpha = 0.2075$ it holds $\frac{1-\alpha}{1+\alpha} \approx 0.6563$ and $0.5\alpha^{-1} \approx 2.41$ we get

Theorem 6. *For the self-shrinking generator of an n-LFSR L, our algorithm computes the secret initial state $x \in \{0,1\}^n$ from the first $\lceil 2.41n \rceil$ bits of $y = shrink(L(x))$ in time $n^{O(1)} 2^{0.6563n}$.* □

Observe that the best short-keystream attacks previously known against the self-shrinking generator were given by *Meier* and *Staffelbach* [12] ($2^{0.75n}$ polynomial time operations) and *Zenner et. al.* [17] ($2^{0.694n}$ polynomial time operations). *Mihaljević* [11] presented an attack which yields a tradeoff between time and length of available keystream. It gives $2^{0.5n}$ necessary polynomial time operations if $2^{0.5n}$ bits of keystream are available, and matches our bound of $2^{0.6563n}$ necessary polynomial time operation if $2^{0.3n}$ bits of keystream are available, which is a quite unrealistic assumption.

The difficulty in applying our algorithm to the A5/1 generator is that the compression algorithm reads most of the internal bits twice, one time for the clock control and a certain time later for producing an output key bit. Read-twice BDDs do not have any of the nice algorithmic properties 3.01 - 3.04, unless $P = NP$. For making the A5/1 generator accessable to our approach we have to modify the keystream generation rule. We define the internal bitstream to be mixed of 6 LFSR-sequences $L^0, \ldots, L^5$, instead of 3. The first 3 LFSR-sequences are generated by the 3 LFSRs of the A5/1 generator. They are used for producing the output bits. The sequences L^3, L^4, L^5 are used for computing the control values. They are shifted copies of the first 3 sequences, defined by the rule $L_j^{3+k} = L_{j+N_k}^k$, for $k = 0, 1, 2$. After this modification, the compression algorithm can be designed in such a way that each internal bit is read exactly once (see the appendix, subsection A.3 for the details).

The (best case) compression ratio of the modified version of A5/1 is $\gamma = \frac{1}{4}$, as either 4 or 6 new internal bits are used for producing the next output bit. It is proved in the appendix-subsection A.4 that the information rate α is the solution of

$$2^{1-4\alpha} = \frac{1}{4}\left(3 + 2^{2\alpha}\right),$$

which yields $\alpha \approx 0.2193$. That the (modified) A5/1 generator fulfils the FBDD-assumption is shown in subsection A.3. Taking into account that $\frac{1-\alpha}{1+\alpha} \approx 0.6403$ and $\gamma\alpha^{-1} \approx 1.14$ we obtain

Theorem 7. *For an A5/1 generator E of key length n, our algorithm computes the secret initial state x from the first $\lceil 1.14n \rceil$ bits of $y = E(x)$ in time $n^{O(1)} 2^{0.6403n}$.*
$\square$

The best previously known short-keystream attack was given by *Golić* [7]. It is against a version of A5/1 generator with keylength 64, which slightly deviates from the specification discovered in [5]. A tight analysis of the time behaviour of *Golić's* attack, when applied to the real A5/1 generator, was given by *Zenner* in [16] and yields 2^{42} polynomial time operations. We get a marginal improvement, as $\lceil 64 \cdot 0.6403 \rceil = 41$. The best long-keystream attacks were given by *Biryukov*, *Shamir* and *Wagner* in [2], and *Biham* and *Dunkelman* in [1]. After a preprocessing of 2^{42} operations the first attack in [2] breaks the cipher within seconds on a modern PC if around 2^{20} bits of keystream are available. The second attack in [2] breaks the cipher within minutes after a preprocessing of 2^{48} operations and under the condition that around 2^{15} bits of keystream are available. The attack in [1] breaks the cipher within $2^{39.91}$ A5/1 clockings on the basis of $2^{20.8}$ available keystream bits.

8 Discussion

There are classical design criterions for keystream generators like a large period, a large linear complexity, correlation immunity and good local statistics. In this paper we suggest a new design criterion: resistance against FBDD-attacks. We have seen that there are two strategies to achieve this resistance. The first is to highly compress the internal bitstream (as in the case of E_0^2). This implies a low bit-rate which is not desirable. The second strategy is to design the compression function C in such a way that the decision if for an internal bitstream z it holds that $C(z)$ is a prefix of a given keystream y requires exponential size FBDDs. It is an interesting challenge to look for such constructions of compression functions which match also practical requirements. For demonstrating the universality of our approach we presented the FBDD-attack in a very general setting. The obvious disadvantage of this general setting is that the algorithm needs a lot of space as all intermediate FBDDs have to be explicitely constructed. It is an interesting open question if the algorithmic idea of FBDD-minimization can be used in a more subtle way for getting, at least for some ciphers, an algorithm which is less space consuming. Another interesting direction of further research is to check whether the FBDD-attack could be successfully combined with other more sophisticated methods of cryptanalysis like the tradeoff attacks suggested in [7], [2] and [1]. Moreover, it would be interesting to get some experimental results for smaller key lengths with real implementations of the FBDD-attack. How much do the real sizes of the minimized intermediate FBDDs deviate from the pessimistic upper bounds proved in our analysis?

Acknowledgement

I would like to thank Stefan Lucks, Erik Zenner, Christoph Meinel, Ingo Wegener, and Rüdiger Reischuk for helpful discussions.

References

1. E. Biham, O. Dunkelman. Cryptanalysis of the A5/1 GSM Stream Cipher. Proc. of INDOCRYPT 2000, LNCS 1977, 43-51.
2. A. Biryukov, A. Shamir, D. Wagner. Real Time Cryptanalysis of A5/1 on a PC. Proc. of Fast Software Encryption 2000, LNCS 1978, 1-18.
3. Bluetooth SIG. Bluetooth Specification Version 1.0 B, http//:www.bluetooth.com/
4. R. E. Bryant. Graph-based algorithms for Boolean function manipulations. IEEE Trans. on Computers 35, 1986, 677-691.
5. M. Briceno, I. Goldberg, D. Wagner. A pedagogical implementation of A5/1. http//:www.scard.org, May 1999.
6. S. R. Fluhrer, S. Lucks. Analysis of the E_0 Encryption System. Technical Report, Universität Mannheim 2001.
7. J. D. Golić. Cryptanalysis of alleged A5/1 stream cipher. Proc. of EUROCRYPT'97, LNCS 1233, 239-255.
8. J. Gergov, Ch. Meinel. Efficient Boolean function manipulation with OBDDs can be generalized to FBDDs. IEEE Trans. on Computers 43, 1994, 1197-1209.
9. S. W. Golomb. Shift Register Sequences. Aegean Park Press, Laguna Hills, revised edition 1982.
10. Ch. Meinel. Modified Branching Programs and their Computational Power. LNCS 370, 1989.
11. M. J. Mihaljević. A faster Cryptanalysis of the Self-Shrinking Generator. Proc. of ACIPS'96, LNCS 1172, 182-189.
12. W. Meier, O. Staffelbach. The Self-Shrinking Generator. Proc. of EUROCRYPT'94, LNCS 950, 205-214.
13. R. A. Rueppel. Stream Ciphers. Contemporary Cryptology: The Science of Information Integrity. G.Simmons ed., IEEE Press New York, 1991.
14. D. Sieling, I. Wegener. Graph driven BDDs - a new data structure for Boolean functions. Theoretical Computer Science 141, 1995, 283-310.
15. I. Wegener. Branching Programs and Binary Decision Diagrams. SIAM Monographs on Discrete Mathematics and Applications. Philadelphia 2000.
16. E. Zenner. Kryptographische Protokolle im GSM Standard: Beschreibung und Kryptanalyse (in german). Master Thesis, University of Mannheim, 1999.
17. E. Zenner, M. Krause, S. Lucks. Improved Cryptanalysis of the Self-Shrinking Generator. Proc. of ACIPS'2001, LNCS 2119, 21-35.

Appendix

A.1 FBDD-Constructions for the E_0 generator

Let us fix a keystream y. We construct polynomial (even linear) size π-OBDDs Q_m which decide for an internal bit-stream

$$z = (z_0^0, z_0^1, z_0^2, z_0^3, z_1^0, z_1^1, z_1^2, z_1^3, \ldots, z_{m-1}^0, z_{m-1}^1, z_{m-1}^2, z_{m-1}^3),$$

using this variable ordering π, whether $C(z) = (y_0, \ldots, y_{m-1})$. The Q_m follow, via Theorem 1, from the following $O(\log(m))$-space bounded algorithm.

```
1. $q := q_0$
2. For $j := 0$ to $m - 1$
3.     if $a(q, d_j) \oplus p_j = y_j$
4.         then $q := \delta(q, d_j)$
5.         else stop(0)
6. stop(1)
```

where $d_j = (z_j^0 + z_j^1 + z_j^2 + z_j^3)$ div 2 and $p_j = (z_j^0 + z_j^1 + z_j^2 + z_j^3)$ mod 2 for $j = 0, \ldots, m - 1$.

We assume here that the initial state q_0 of M is public. If this is not the case we have to run our algorithm 16 times, one round for each possible state of M.

A.2 FBDD-Constructions for the E_0-Generator in the Two-Level Mode

Let us fix a keystream y. We describe how FBDDs Q_m, S_m and P_m from section 6 have to be defined in the case of the E_0^2 generator with key-length and internal key-length $N > n$. For the sake of simplicity we suppose that N is divisible by 4. Inputs for Q_m and S_j are internal bit-streams looking like

$$u_0, \ldots, u_3, z_0, u_4, \ldots, u_7, z_1, \ldots, u_{4N-4}, \ldots, u_{4N-1}, z_{N-1}, z_N, z_{N+1}, \ldots$$

which are read in this order π, i.e., Q_m and S_j will be π-OBDDs.

Let L denote the linear bit-stream generator of the E_0 generator of key-length N. Let q_0 and q_0' denote the (public) initial states of the two E_0 generators of E_2^0. Let C and C' denote the the compression functions of the E_0 generator with initial states q_0 and q_0', respectively.

For all $m \geq n$, the program P_m reads the first m bits of the internal bit-stream and tests if

- $u_r = U_r(u_0, \ldots, u_{n-1})$ for $r = n, \ldots, N - 1$,
- the bit-stream $u = u_0, u_1, u_2, \ldots$ is generated via L and $C(u)$ is prefix of the bit-stream z,
- the bit-stream z is generated via L and $C'(z)$ is prefix of y.

These tests will be distributed to the programs S_m and Q_m as follows. The programs S_m test the linear restrictions and are minimal π-OBDDs which do the following

- If at position m stands some u_r, for $n \leq r \leq N - 1$, then S_m tests if $u_r = U_r(u_0, \ldots, u_{n-1})$.
- If at position m stands some u_r, for $N \leq r \leq 4N - 1$, then S_m tests if $u_r = L_r(u_0, \ldots, u_{N-1})$.
- If at position m stands some z_s, for $s \geq N$, then S_m tests if $z_s = L_s(z_0, \ldots, z_{N-1})$.

For all other positions m, S_m is defined to answer always 1.

For all $m \geq n$, the programs Q_m are minimal π-OBDDs which decide

- whether $C(u)$ is prefix of z and
- whether $C'(z)$ is prefix of y,

where u and z denote the strings of all u_r-bits and z_s bits, respectively, occuring among the first m bits of the internal bit-stream. Observe that running our cryptanalysis algorithm from section 6 with these Q_m and S_m yields th desired P_m for E_0^2.

For $0 \leq j \leq N-1$ we write

- $d_j = (u_{4j} + u_{4j+1} + u_{4j+2} + u_{4j+3})$ div 2,
- $p_j = (u_{4j} + u_{4j+1} + u_{4j+2} + u_{4j+3}) \mod 2$.

For $j \geq N/4$ we write

- $d'_j = (z_{4j} + z_{4j+1} + z_{4j+2} + z_{4j+3})$ div 2,
- $p'_j = (z_{4j} + z_{4j+1} + z_{4j+2} + z_{4j+3}) \mod 2$.

For all $m \geq n$ let $\gamma(m)$ denote the number of key-bits produced by the first m bits of the internal bit-stream. Let us fix some m for which $\gamma(m) \geq N/4$. The programs Q_m can be constructed, via Theorem 1, according to the following read-once algorithm which is obviously $O(\log(m))$-space bounded.

```
1.  $q := q_0$, $q' := q'_0$
2.  For $j := 0$ to $N/4 - 1$
3.      $S := 0$
4.      For $k := 0$ to 3
5.          if $a(q, d_{4j+k}) \oplus p_{4j+k} = z_{4j+k}$
6.              then $S := S + z_{4j+k}$, $q := \delta(q, d_{4j+k})$
7.              else stop(0)
8.      $d := S$ div 2, $p = S \mod 2$,
9.      if $a(q', d) \oplus p = y_j$
10.         then $q' := \delta(q', d)$
10.         else stop(0)
12. For $j := N/4$ to $t(m) - 1$
13.     if $a(q', d'_j) \oplus p'_j = y_j$
14.         then $q' := \delta(q', d'_j)$
15.         else stop(0)
16. stop(1)
```

For all $j = 0, \ldots, N/4 - 1$, while checking $C(u)_k = z_k$, for $k = z_{4j}, \ldots, z_{4j+3}$ the algorithm stores in S all information necessary for checking that $C'(z)_j = y_j$.

A.3 FBDD-Constructions for the $A5/1$-Generator

Let us fix 3 LFSRs L^0, L^1, L^2 of key-lengthes n^0, n^1 and n^2, where $n = n^0 + n^1 + n^2$. Let us further fix 3 control positions N^0, N^1 and N^2, where for $k = 0, 1, 2$

$$N^k \in \{\lfloor n^k/2 \rfloor - 1, \lfloor n^k/2 \rfloor\}.$$

As already mentioned, for getting the read-once property for the compression, we have to define the A5/1 generator with respect to the linear bit-stream $L(x)$, $x \in \{0,1\}^n$, which is mixed of 6 LFSR-sequences $L^0(x), \ldots, L^5(x)$ as

$$L(x) = L_0^0(x), \ldots, L_0^5(x), \ldots, L_m^0(x), \ldots, L_m^5(x), \ldots$$

where, for $k = 0, 1, 2$, $L^{3+k}(x)$ corresponds to $L^k(x)$ shifted by N^k, i.e.,

$$L_j^{3+k}(x) = L_{j+N_k}^k(x).$$

for all $j \geq 0$. The internal state x consists of x^0, x^1, x^2, we write $L^k(x)$ instead of $L^k(x^k)$.

The definition of the compression function C is as follows. In each timestep i, the compressor holds 3 control positions $i[0]$, $i[1]$, $i[2]$, outputs the key-bit

$$y_i = L_{i[0]}^0 \oplus L_{i[1]}^1 \oplus L_{i[2]}^2,$$

computes the control value

$$c = maj_3\left(L_{i[0]}^3, L_{i[1]}^4, L_{i[2]}^5\right),$$

and updates for $k = 0, 1, 2$ the control positions according to

$$\text{if } c = L_{i[k]}^{3+k} \text{ then } i[k] := i[k] + 1.$$

for $k = 0, 1, 2$. The order in which C reads the internal bits is governed by the control positions. As for internal bit-streams of length m there are at most $O(m^3)$ possible assignments to the 3 control values, G_m^C can be shown to have size $O(m^4)$.

Polynomial size G_m^C-driven FBDDs which decide for a bit-stream

$$z = z_0^0, \ldots, z_0^5, z_1^0, \ldots, z_1^5, \ldots,$$

of length m whether $C(z)$ is prefix of a given keystream y, can be constructed (via Theorem 3) according to the following $O(\log(m))$-space bounded read-once algorithm. For the sake of simplicity let m be divisible by 6.

```
 0. $j := 0$
 1. for all $k = 0, 1, 2$ let $i[k] := 0$
 2. for all $k = 0, 1, 2$ let $status[k] := read$
 3. while for all $k = 0, 1, 2$ $i[k] < m/6$ do
 4.     for $k = 0, 1, 2$
 5.         if $status[k] = read$ then $out[k] := z_{i[k]}^k$; $c[k] := z_{i[k]}^{3+k}$
 6.     if $y_j \neq out[1] \oplus out[2] \oplus out[3]$ then **stop(0)**
 7.     $m := maj_3(c[1], c[2], c[3])$
 8.     for $k = 1, 2, 3$
 9.         if $m = c[k]$
10.             then $i[k] := i[k] + 1$; $status[k] := read$
11.             else $status[k] := notread$
12.     j:=j+1
13. **stop(1)**
```

Observe that the control variable $status(k)$ ensures that no x^k and no x^{3+k} variable will be read more than once while $i[k]$, $out[k]$ and $c[k]$ are updated.

A.4 Estimating the Information Rate of the A5/1 generator

Let C denote the compression function of the modified A5/1 generator. We need to consider how C produces their output bits on a random uniformly distributed internal bitstream of fixed length m. For $i = 0, 1, 2, \ldots$, let Y_i denote the random variable corresponding to the number of internal bits which are necessary for producing the i-th output bit. Observe that, for all i, either 6 or 4 bits are necessary, depending on whether all three control bits are equal or not. It is quite straightforward to check that Y_i and Y_j are independent for all $i \neq j$ and that

$$Prob[Y_i = 6] = \frac{1}{4} \text{ and } Prob[Y_i = 4] = \frac{3}{4}.$$

Let α denote the information rate (per bit) of the A5/1 generator. For all integers $m \geq 0$ and $k \leq m$ let $p(m) := 2^{\alpha m}$ and

$$p(m,k) := Prob_{x \in_U \{0,1\}^m}[|C(x)| = k].$$

We derive an estimation for α from the relation

$$p(m) = \sum_{k=m/6}^{m/4} p(m,k) 2^{-k} \tag{4}$$

On each internal bitstream of length m at least $m/6$ output bits are produced. We denote by Z the random variable corresponding to the number of internal bits which are processed after producing the first $m/6$ output bits. It holds that

$$Z = 6Z' + 4(m/6 - Z'),$$

where Z' is an $(m/6, 1/4)$-binomially distributed random variable, which corresponds to the number of those of the first $m/6$ output bits, which used 6 internal bits. Observe that $m - Z$, the number of internal bits remaining after the production of $m/6$ output bits, can be written as

$$m - Z = m - 6Z' - \frac{2}{3}m + 4Z' = m/3 - 2Z'.$$

Now, the probability $p(m)$ can be written as

$$\begin{aligned} p(m) &= \sum_{i=0}^{m/6} 2^{-m/6} Prob[Z' = i] \sum_{j=(m/3-2i)/6}^{(m/3-2i)/4} p(m/3 - 2i, j) 2^{-j} \\ &= \sum_{i=0}^{m/6} 2^{-m/6} \binom{m/6}{i} \frac{1}{4}^i \frac{3}{4}^{m/6-i} p(m/3 - 2i). \end{aligned}$$

We get the following recurrence for α.

$$\begin{aligned} 2^{-\alpha m} &= 2^{-m/6} \sum_{i=0}^{m/6} \binom{m/6}{i} \frac{1}{4}^i \frac{3}{4}^{m/6-i} 2^{-\alpha(m/3-2i)} \\ &= 2^{-(1/6+\alpha/3)m} \sum_{i=0}^{m/6} \binom{m/6}{i} \frac{1}{4}^i \frac{3}{4}^{m/6-i} 2^{2\alpha i}. \end{aligned} \tag{5}$$

For a further simplification of this expression we use the following technical lemma.

Lemma 4. *For all integers $n \geq 0$, $p \in (0,1)$ and reals β it holds*

$$\sum_{i=0}^{n} \binom{n}{i} p^i (1-p)^{n-i} 2^{\beta i} = \left(1 - p + p2^{\beta}\right)^n.$$

Proof. For all integers $n \geq 0$ let $f(n) := \sum_{i=0}^{n} \binom{n}{i} p^i (1-p)^{n-i} 2^{\beta i}$. Observe that $f(0) = 1$. Using the well-known relation that for all $i,\ 1 \leq i \leq n$, it holds

$$\binom{n}{i} = \binom{n-1}{i} + \binom{n-1}{i-1}$$

one can derive that

$$f(n) = \left(1 - p + p2^{\beta}\right) f(n-1),$$

which immediately yields the lemma. $\square$ $\square$

Applying the lemma to relation (5) we obtain

$$\begin{aligned} 2^{-\alpha m} &= 2^{-(1/6+\alpha/3)m} \frac{1}{4} \left(3 + 2^{2\alpha}\right)^{m/6}, \text{ i.e.} \\ 2^{-6\alpha} &= 2^{-(1+2\alpha)} \frac{1}{4} \left(3 + 2^{2\alpha}\right), \text{ i.e.} \\ 2^{1-4\alpha} &= \frac{1}{4} \left(3 + 2^{2\alpha}\right). \end{aligned}$$

Evaluating this expression yields $\alpha \approx 0.2193$.