

BDD-based Cryptanalysis of the A5/1 Keystream Generator - Experimental Results

Dirk Stegemann
University of Mannheim, Germany

Abstract

The A5/1 keystream generator as part of the GSM standard for mobile telephones is one of the most widely used keystream generators. [Kra02] proposed a generic, binary decision diagram based attack on generators that consist of a small number of parallel linear feedback shift registers (LFSRs) and some nonlinear compression function combining the output of the LFSRs to the output keystream. Applied to the A5/1 keystream generator, this attack yields an $n^{O(1)}2^{0.6403n}$ time and space bounded algorithm for computing the secret initial state of the generator from the first $\lceil 1.14n \rceil$ consecutive bits of the keystream. We survey the generic result, give a detailed description of its application to the A5/1 generator and present first experimental results showing that the performance of the attack in practice does not deviate from the theoretical bound by more than a constant factor. With the current implementation, we are able to compute the initial state of A5/1 generators with keylength 28 in less than 15 minutes on a standard PC. For larger keylengths, the availability of main memory turns out to be the main bottle neck for practically feasible attacks.

1 Introduction

The A5/1 keystream generator that is used in the GSM standard for mobile telephones consists of three linear feedback shift registers (LFSRs) and a clock control that computes the keystream from the output bitstream of the LFSRs. In a more generic way, we can think of the A5/1 generator as a device containing a linear bitstream generator L that consists of three LFSRs and produces an internal bitstream z , and a nonlinear compression function C that transforms the internal bitstream z into the output keystream y . We will refer to such devices as LFSR-based keystream generators $K = (L, C)$.

The main purpose of LFSR-based keystream generators is the online-encryption of bitstreams $p \in \{0, 1\}^*$ that have to be transmitted over an insecure channel, e.g. for encrypting speech data to be transmitted from and to a mobile phone over the air interface. The output keystream $y \in \{0, 1\}^*$ of the generator is bitwise XORed to the plaintext stream p in order to obtain the ciphertext stream $e \in \{0, 1\}^*$, i.e. $e_i = p_i \oplus y_i$ for all i . Based on an initial state $x \in \{0, 1\}^n$, which has to be exchanged between sender and legal receiver prior to the transmission in a suitable way, the receiver can compute the keystream y from x in the same way as the sender computed it and decrypt the message using the above rule.

[Kra02] proposed an attack on LFSR-based keystream generators that makes use of free binary decision diagrams (FBDDs) and showed that LFSR-based generators are generally vulnerable against these attacks, but gave only pessimistic upper bounds on the performance. Similarly to [Sch02] providing experimental results for the application of the algorithm to the E_0 encryption standard [blu] and the self-shrinking generator [MS95], our goal is to verify how the performance of the attack in practice deviates from the theoretical bounds in the case of the A5/1 keystream generator.

The article is organized as follows. Section 2 provides the basic definitions of LFSR-based keystream generators and states their most important properties, while section 3 gives a brief introduction to FBDDs. Section 4 surveys the generic result of [Kra02] and section 5 describes its application to the A5/1 generator in detail. Finally, section 6 presents implementational issues and our experimental results.

2 LFSR-based Keystream Generators

2.1 Definitions

A *Linear Feedback Shift Register* (LFSR) of length n with a coefficient vector $c = (c_1, \dots, c_n) \in \{0, 1\}^n$ consists of n binary register cells q_1, \dots, q_n that are connected by a feedback channel. The LFSR is periodically clocked, and in each clock cycle $t > 0$, the LFSR outputs the content of q_t and copies for $i \in \{2, \dots, n\}$ the content of q_i into q_{i-1} . The new value for q_n is computed via the feedback channel from the current contents of the register cells according to

$$q_n^{new} := c_1 q_1 \oplus c_2 q_2 \oplus \dots \oplus c_n q_n$$

where c_i indicates whether or not q_i is connected to the feedback channel. At the beginning of the computation, i.e. in clock cycle $t = 0$, the register cells are initialized with an initial state $x = \{0, 1\}^n$. For the produced bitstream

$$L(x) = L_0(x), L_1(x), \dots, L_i(x), \dots$$

it holds that

$$L_i(x) = \begin{cases} x_i & \text{for } 0 \leq i \leq n-1 \\ c_1 L_{i-n}(x) \oplus c_2 L_{i-n+1}(x) \oplus \dots \oplus c_n L_{i-1}(x) & \text{for } i > n-1 \end{cases},$$

i.e. it is $L_i(x) = \bigoplus_{k=0}^{n-1} c_{k+1} \cdot L_{i-n+k}(x)$ for $i > n-1$.

Note that the first n bits in the output of an LFSR are the initial state bits while the following bits are linear combinations of the initial state. We can therefore equivalently compute the produced bitstream as

$$L_i(x) = \bigoplus_{j=0}^{n-1} L_{j,i} \cdot x_j$$

$$\text{where } L_{j,i} = \begin{cases} 1 & \text{if } i \leq n-1 \text{ and } i = j \\ 0 & \text{if } i \leq n-1 \text{ and } i \neq j \\ \bigoplus_{k=0}^{n-1} c_{k+1} \cdot L_{j,i-n+k} & \text{otherwise} \end{cases}$$

Obviously, the initial state $x = (0, \dots, 0)$ yields $L_i(x) = 0$ for all i . We will therefore assume x to be different from 0. If the feedback polynomial

$$F(x) := x^n + \sum_{i=0}^{n-1} c_i x^{i-1},$$

that is associated with the LFSR is primitive, the output bitstream has several nice pseudo-randomness properties such as maximum period, good auto correlation and local statistics. See [Gol82] for details on these properties.

A *Linear Bitstream Generator* L consists of $k \geq 1$ parallel LFSRs L^r of length n_r , $r \in \{0, \dots, k-1\}$ and $n_0 + \dots + n_{k-1} = n$. L produces a bitstream

$$L(x) = L_0(x), L_1(x), \dots, L_i(x), \dots$$

with

$$L_i(x) := L_{s(i)}^{r(i)} \left(x^{r(i)} \right) \quad \text{where} \quad \begin{cases} r(i) = i \bmod k \\ s(i) = i \operatorname{div} k \end{cases},$$

i.e. the i -th output bit of L corresponds to the $s(i)$ -th output bit of LFSR $L^{r(i)}$. The initial states x^p of the LFSRs L^p , $p \in \{0, \dots, k-1\}$, form the initial state $x \in \{0, 1\}^n$ of L .

For all $x \in \{0, 1\}^n$ and $m \geq 1$, let $L_{\leq m}(x)$ denote the first m bits of $L(x)$, i.e. $L_{\leq m}(x) := (L_0(x), L_1(x), \dots, L_{m-1}(x))$.

Let $I_m(L)$ and $C_m(L)$ denote the indices of initial state bits and linearly combined bits in the output bitstream of L , respectively. More precisely, we define

$$\begin{aligned} I_m(L) &:= \{j \in \{0, \dots, m-1\} \mid s(j) < n_{r(j)}\} \\ \text{and } C_m(L) &:= \{j \in \{0, \dots, m-1\} \mid s(j) \geq n_{r(j)}\} = \{0, \dots, m-1\} \setminus I_m \end{aligned}$$

Moreover, let $n' := \min_i \{i \in C_m(L)\}$ denote the index of the first linearly combined bit in the bitstream produced by L .

For a given internal bitstream $z = (z_0, \dots, z_{m-1})$ let $i_m(L, z)$ and $c_m(L, z)$ denote the initial state bits and the linearly combined bits, respectively, i.e.

$$\begin{aligned} i_m(L, z) &:= (z_{j_1}, \dots, z_{j_{|I_m(L)|}}) \text{ with } j_p \in I_m(L) \forall p \in \{1, \dots, |I_m(L)|\} \text{ and } j_p < j_q \forall p < q \\ c_m(L, z) &:= (z_{j_1}, \dots, z_{j_{|C_m(L)|}}) \text{ with } j_p \in C_m(L) \forall p \in \{1, \dots, |C_m(L)|\} \text{ and } j_p < j_q \forall p < q \end{aligned}$$

An *LFSR-based Keystream Generator* $K = (L, C)$ uses a linear bitstream generator L consisting of k LFSRs L^0, \dots, L^{k-1} to compute an internal bitstream $z = L(x)$ and produces a keystream

$$y = C(z) = C(L(x)) \in \{0, 1\}^*$$

where C denotes a nonlinear compression function $C : \{0, 1\}^* \rightarrow \{0, 1\}^*$. C computes the keybits in an online manner, i.e. there exists a function $\delta : \mathbb{N} \rightarrow \mathbb{N}$ with $\delta(i) < \delta(j)$ for $i < j$, such that the value of the i -th keybit only depends on the first $\delta(i)$ bits of z . Moreover, C reads the internal bits in the order in which they are produced by the LFSRs, i.e. for $s > 0$ and all $r \in \{0, \dots, k-1\}$, $L_{k \cdot s + r}(x)$ is not read before $L_{k \cdot (s-1) + r}(x)$.

We call two initial states $x, x' \in \{0, 1\}^n$ of an LFSR-based keystream generator *equivalent*, if $C(L(x)) = C(L(x'))$, i.e. if K produces the same keystream for both x and x' .

2.2 Important Properties

Since the keystream is produced in an online manner, the time difference between the output of two keybits should be small on average and not vary too much. We therefore expect the following assumption to hold:

Assumption 1 (Partitioning Rule). *Any internal bitstream $z = L(x)$ of an LFSR-based keystream generator can be divided into elementary blocks z^0, z^1, \dots, z^{l-1} with $|z^j| \geq 1$ such that*

$$C(z) = y^0, y^1, \dots, y^{l-1}$$

where $y^j = C(z^j)$ and $|y^j| = 1$ for all $j = 0, \dots, l-1$. The average length β of the elementary blocks is given by

$$\beta = \frac{\sum_{j=0}^{l-1} |z^j|}{l} \geq 1,$$

i.e. on average, β bits of the internal bitstream z are used to produce a single keybit.

Let γ denote the best-case compression ratio of C , i.e. γm is the maximum number of keybits that C produces from internal bitstreams of length m . Assumption 1 immediately yields $\gamma \in (0, 1]$.

For a randomly chosen and uniformly distributed internal bitstream $Z^{(m)} \in \{0, 1\}^m$ and a random keystream Y , let

$$\alpha := \frac{1}{m} I(Z^{(m)}, Y) \in (0, 1]$$

denote the average information that Y reveals about $Z^{(m)}$.

For a randomly chosen and uniformly distributed internal bitstream $z \in \{0, 1\}^m$, the probability of the keybits $C(z)$ being a prefix of a given keystream $y \in \{0, 1\}^*$ can be expressed as

$$\text{Prob}_z[C(z) \text{ is prefix of } y] =$$

$$\sum_{i=0}^{\lceil \gamma m \rceil} \text{Prob}_{z \in \{0,1\}^m} [|C(z)| = i] \cdot \text{Prob}_{z \in \{0,1\}^m, |C(z)|=i} [C(z) = (y_0, \dots, y_{i-1})]$$

Concerning this probability, we will make the following assumption:

Assumption 2 (Independence-Assumption). *For all $m \geq 1$, a randomly chosen, uniformly distributed internal bitstream $z \in \{0,1\}^m$ and all keystreams $y \in \{0,1\}^*$, we have*

$$\text{Prob}_z [C(z) \text{ is prefix of } y] = p_C(m),$$

i.e. the probability of $C(z)$ being a prefix of y is the same for all y .

Under Assumption 2, we can write the information rate α as (see [Kra02], [Ste04])

$$\alpha = -\frac{1}{m} \log_2 p_C(m)$$

Finally, we assume the keystream y to behave pseudorandomly, or more precisely:

Assumption 3 (Pseudorandomness-Assumption). *For all keystreams y and all $m \leq \lceil \alpha^{-1} n \rceil$ it holds that*

$$\text{Prob}_z [C(z) \text{ is prefix of } y] \approx \text{Prob}_x [C(L_{\leq m}(x)) \text{ is prefix of } y],$$

where z and x denote randomly chosen, uniformly distributed elements of $\{0,1\}^m$ and $\{0,1\}^{\lceil I_m(L) \rceil}$, respectively.

Note that a severe violation of Assumption 3 would imply a vulnerability of K via a correlation attack.

3 Free Binary Decision Diagrams (FBDDs)

A *Binary Decision Diagram* (BDD) over a set of variables $X_n = \{x_1, \dots, x_n\}$ 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 $v.\text{label} \in X_n$, whereas the two sinks are labelled $s_0.\text{label} = 0$ and $s_1.\text{label} = 1$. There is exactly one node with indegree 0, the root of the BDD. We define the size of a BDD to be the number of vertices in G , i.e. $|G| := |V|$. Each node $v \in V$ represents a Boolean Function $f_v \in B_n = \{f | f : \{0,1\}^n \rightarrow \{0,1\}\}$ in the following manner: For an input $a = (a_1, \dots, a_n) \in \{0,1\}^n$, 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.

For a BDD G over X_n , let $\text{One}(G) \subseteq \{0,1\}^n$ denote the set of all inputs accepted by G , i.e. all inputs $a \in \{0,1\}^n$ such that the path determined by a beginning in the root of G leads to the 1-sink.

An *oracle graph* $G_0 = (V, E)$ over a set of variables $X_n = \{x_1, \dots, x_n\}$ is a modified BDD that contains only one sink s , labelled by $*$, and for all $x_i \in X_n$ and all paths P from the root in G to the sink, there exists at most one node in P which is labelled by x_i .

A *Free Binary Decision Diagram* with respect to an oracle graph G_0 (G_0 -FBDD) over a set of variables $X_n = \{x_1, \dots, x_n\}$ is a BDD in which the following property holds for all inputs $a \in \{0,1\}^n$: Let the list $G_0(a)$ contain the variables from X_n in the order in which they are tested on the path defined by a in G_0 . Similarly, let the list $G(a)$ contain the variables from X_n in the order of testing in G . If x_i and x_j are both contained in $G(a)$, then they occur in $G(a)$ in the same order as in $G_0(a)$.

We call a BDD G an FBDD, if there exists an oracle graph G_0 such that G is a G_0 -FBDD.

The definition of FBDDs implies their important *read-once property*, i.e. on each path in an FBDD G , each variable in X_n is tested at most once.

An FBDD G is called *Ordered Binary Decision Diagram* (OBDD) if there exists an oracle graph G_0 such that G is a G_0 -FBDD and G_0 is degenerated into a linear list.

FBDDs possess several algorithmic properties that will prove useful in our context. Let G_0 denote an oracle graph over $X_n = \{x_1, \dots, x_n\}$ and let the G_0 -FBDDs G_f, G_g and G_h represent Boolean functions $f, g, h : \{0, 1\}^n \rightarrow \{0, 1\}$, respectively.

Property 1. *There exists an algorithm MIN which computes for G_f in time $O(|G|)$ the (uniquely defined) minimal G_0 -FBDD representing f .*

Property 2. *There exists an algorithm SYNTH which computes for G_f, G_g and G_h in time $O(|G_0| \cdot |G_f| \cdot |G_g| \cdot |G_h|)$ a G_0 -FBDD G of size $|G| \leq |G_0| \cdot |G_f| \cdot |G_g| \cdot |G_h|$ which represents the function $f \wedge g \wedge h$.*

Property 3. *There exists an algorithm SAT-ENUM which enumerates for a G_0 -FBDD G_f all elements in $\text{One}(G_f)$ in time $O(n \cdot |\text{One}(G_f)|)$.*

Property 4. *For every minimal G_0 -FBDD G over X_n it holds that $|G| \leq n \cdot |\text{One}(G)|$.*

See [Weg00] or [Ste04] for detailed descriptions of these algorithms.

FBDDs for a given decision problem $F \subseteq \{0, 1\}^*$ can be constructed with the help of the following result, which is given in [Mei89].

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. \square*

Therefore, we can prove the existence of polynomial size FBDDs for a given Boolean function f by identifying logarithmically space bounded read-once algorithms for f .

4 Cryptanalysis of LFSR-based Keystream Generators using FBDDs

For the cryptanalysis of LFSR-based keystream generators, we suppose that the only secret information is the initial state x and that all other parameters of the generator such as the coefficient vector c and the lengths of the LFSRs are publicly known. Moreover, we assume that the attacker is able to obtain the first s plaintext-ciphertext pairs $(p_0, e_0), \dots, (p_{s-1}, e_{s-1})$ of the transmission, from which he can easily compute the first s keybits via $y_i = p_i \oplus e_i$. The attacker's goal is then to compute an initial state $x \in \{0, 1\}^n$ that produces the observed keystream. Observe that the trivial exhaustive search attack, i.e. systematically testing all possible initial states, needs time $n^{O(1)}2^n$. Looking for a more efficient approach, we observe that since the initial state is contained in the first bits of z , the problem reduces to finding an internal bitstream $z \in \{0, 1\}^m$ with the following two properties:

- (i) z can be produced by the linear bitstream generator L , i.e. $z = L_{\leq m}(i_{|z|}(L, z))$.
- (ii) $C(z)$ is prefix of the observed keystream y .

For each $m \geq 1$, the internal bitstreams $z \in \{0, 1\}^m$ fulfilling these properties can be represented with the help of the following FBDDs:

Let G_m^C denote the oracle graph that defines for each z the order in which the bits of z are read by the compression function C .

Let R_m denote the minimal G_m^C -FBDD that decides whether $z = L_{\leq m}(i_m(L, z))$.

Let S_m denote the minimal G_m^C -FBDD that decides whether $z_{m-1} = L_{m-1}(i_m(L, z))$.

Let Q_m denote the minimal G_m^C -FBDD that decides whether $C(z)$ is prefix of y .

Let P_m denote the minimal G_m^C -FBDD that decides whether $z = L_{\leq m}(i_m(L, z))$ and $C(z)$ is prefix of y .

Algorithm 1 FBDD-COMPUTE-x(L,C,y)

$P \leftarrow Q_{n'}$
for $m \leftarrow n' + 1$ **to** m^* **do**
 $P \leftarrow \min(P \wedge Q_m \wedge S_m)$
return $i_m(L, z^*)$ for a $z^* \in \text{One}(P)$

Starting with $P_{n'}$, we can dynamically compute the FBDD P_m for $m \in \{n', \dots, m^*\}$, where m^* denotes the length of the internal bitstream for which with high probability only one internal bitstream z is accepted by P_{m^*} (see Algorithm 1).

Obviously, $P = P_m$ after each iteration m . Moreover, the definition of compression functions implies that for $m' \geq m$, every G_m^C -FBDD is a $G_{m'}^C$ -FBDD, which in turn yields the correctness of the operation $\min(P \wedge Q_m \wedge S_m)$.

An estimation for the value m^* can be derived from the following Lemma:

Lemma 2 ([Kra02], [Ste04]). *If the Pseudorandomness-Assumption holds for K , then for all keystreams y and all $m \leq \lceil \alpha^{-1}n \rceil$ it holds that*

$$|\{x \in \{0, 1\}^{|I_m(L)|} : C(L_{\leq m}(x) \text{ is prefix of } y)\}| \approx 2^{|I_m(L)| - \alpha m} \leq 2^{n - \alpha m},$$

i.e. there exist approximately $2^{n - \alpha m}$ initial states x such that $C(L_{\leq m}(x))$ is prefix of y . \square

Since Lemma 2 holds especially for the observed keystream y , the value m^* is the smallest $m \leq \lceil \alpha^{-1}n \rceil$ for which there exists with high probability only one initial state x such that $C(L_{\leq m}(x))$ is prefix of y . Setting

$$|\{x \in \{0, 1\}^{|I_m(L)|} : C(L_{\leq m}(x) \text{ is prefix of } y)\}| = 1, \text{ we obtain } m^* = \lceil \alpha^{-1}|I_m(L)| \rceil \leq \lceil \alpha^{-1}n \rceil.$$

According to Property 1 and Property 2, the runtime $t(n)$ of the *for*-loop in Algorithm 1 can be bounded by

$$t(n) \leq \sum_{m=n'+1}^{\lceil \alpha^{-1}n \rceil} |G_m^C| |P_{m-1}| |Q_m| |S_m|$$

We therefore need to estimate the sizes of G_m^C and the G_m^C -FBDDs P_m , Q_m and S_m . Concerning the FBDD-sizes, we will make the following assumption:

Assumption 4 (FBDD-Assumption). *For all $m \geq n'$, it holds that*

$$|G_m^C| \in m^{O(1)}, |Q_m| \in m^{O(1)}, |R_m| \leq |G_m^C| 2^{m - |I_m(L)|} \text{ and } S_m \leq O(1) \cdot |G_m^C|$$

In section 5 we will show that the A5/1 keystream generator indeed fulfils these requirements. We are now able to prove an upper bound on the size of P_m :

Lemma 3. *If K fulfils the FBDD-assumption, then $|P_m| \leq n^{O(1)} \cdot 2^{\frac{1-\alpha}{1+\alpha}n}$ for all $n' \leq m \leq \lceil \alpha^{-1}n \rceil$.*

Proof. From the definitions of P_m , R_m and Q_m it follows that $P_m = R_m \wedge Q_m$ for $n' \leq m \leq \lceil \alpha^{-1}n \rceil$. Property 2 implies

$$|P_m| \leq |G_m^C| |R_m| |Q_m|$$

In order to simplify the notation, let $n^* := |I_m(L)|$. Using Assumption 4 as well as $P_{n'} = Q_{n'}$, we get

$$|P_m| \leq |G_m^C|^2 2^{m - n^*} |Q_m| \leq p(m) \cdot 2^{m - n^*} \quad (1)$$

where $p(m) = |G_m^C|^2 \cdot |Q_m|$.

On the other hand, Property 4 implies $|P_m| \leq m \cdot |\text{One}(P_m)|$. It follows from $|\text{One}(P_m)| \approx 2^{n^* - \alpha m}$ (Lemma 2) for all $n' \leq m \leq \lceil \alpha^{-1}n \rceil$ that

$$|P_m| \leq m |\text{One}(P_m)| \approx m \cdot 2^{n^* - \alpha m} = m \cdot 2^{(1-\alpha)n^* - \alpha(m - n^*)} \quad (2)$$

Now, (1) and (2) imply for $n' \leq m \leq \lceil \alpha^{-1}n \rceil$

$$\begin{aligned} |P_m| &\leq \min \left\{ p(m) \cdot 2^{m-n^*}, m \cdot 2^{(1-\alpha)n^* - \alpha(m-n^*)} \right\} \\ &= \min \left\{ p(m) \cdot 2^{r(n^*)}, m \cdot 2^{(1-\alpha)n^* - \alpha r(n^*)} \right\} \text{ for } r(n^*) := m - n^* \\ &\leq \min \left\{ p(m) \cdot 2^{r(n^*)}, p(m) \cdot 2^{(1-\alpha)n^* - \alpha r(n^*)} \right\} \text{ since } m \leq p(m) \\ &\leq p(m) \cdot 2^{r^*(n^*)} \end{aligned}$$

where $r^*(n^*)$ denotes the solution of $2^{r(n^*)} = 2^{(1-\alpha)n^* - \alpha r(n^*)}$, which yields $r^*(n^*) = \frac{1-\alpha}{1+\alpha}n^*$.

Due to $n' \leq m \leq \lceil \alpha^{-1}n \rceil$ it holds that $p(m) \in n^{O(1)}$, and $n^* = |I_m(L)| \leq n$ finally implies the statement of the Lemma. \square

From the upper bound on $|P_m|$, we can easily obtain an upper bound on the runtime and space requirements of Algorithm 1:

Corollary 4. *For the runtime $t(n)$ and the required memory $s(n)$ of the for-loop in Algorithm 1 it holds that*

$$t(n), s(n) \leq n^{O(1)} 2^{\frac{1-\alpha}{1+\alpha}n}$$

Proof. Due to Property 1 and Property 2, a single synthesis and minimize operation needs time and space of at most $|G_m^C| |P_{m-1}| |Q_m| |S_m|$. The FBDD-Assumption and Assumption 4 imply that

$$t(n) \leq \sum_{m=n'+1}^{\lceil \alpha^{-1}n \rceil} |P_{m-1}| \underbrace{|G_m^C| |Q_m| |S_m|}_{\leq m^{O(1)}} \leq \sum_{m=n'+1}^{\lceil \alpha^{-1}n \rceil} m^{O(1)} |P_{m-1}| \leq n^{O(1)} \max_{n'+1 \leq m \leq \lceil \alpha^{-1}n \rceil} \{|P_m|\}$$

$|P_m| \leq n^{O(1)} 2^{\frac{1-\alpha}{1+\alpha}n}$ for all $n' \leq m \leq \lceil \alpha^{-1}n \rceil$ (Lemma 3) yields

$$t(n) \leq n^{O(1)} 2^{\frac{1-\alpha}{1+\alpha}n}$$

A similar argument holds for $s(n)$. \square

The FBDD-Assumption implies $|Q_{n'}| \leq n^{O(1)}$, and according to Property 3, the enumeration of all satisfying assignments is possible with time and space

$$O(\lceil \alpha^{-1}n \rceil \cdot \text{One}(P_{\lceil \alpha^{-1}n \rceil})) \approx O(\lceil \alpha^{-1}n \rceil \cdot 1) = O(\lceil \alpha^{-1}n \rceil)$$

Therefore, the runtime and memory requirements of the remaining operations in Algorithm 1 are polynomial in n .

We obtain the following main result:

Theorem 5. *Let $K = (L, C)$ be an LFSR-based keystream generator with initial state $x \in \{0, 1\}^n$, information ratio α and best-case compression ratio γ . If K fulfils the Independence Assumption, the Pseudorandomness Assumption and the FBDD-Assumption, an initial state \tilde{x} with $C(L(\tilde{x})) = y$ can be computed in time and space $n^{O(1)} 2^{\frac{1-\alpha}{1+\alpha}n}$ from the first $\lceil \gamma \alpha^{-1}n \rceil$ consecutive Bits of the keystream $y = C(L(x))$. \square*

5 Application to the A5/1 Keystream Generator

5.1 Definition of the A5/1 generator

The A5/1 generator is used in the GSM standard for mobile telephones. According to [BGW99], who obtained its design by reverse engineering, the generator consists of 3 LFSRs R_0, R_1, R_2 of length n_0, n_1, n_2 , respectively, and a clock control ensuring that the keybits do not linearly depend on the initial states of the LFSRs. For each $r \in \{0, 1, 2\}$, a register cell

q_{N^r} , $N^r \in \{\lceil \frac{n_r}{2} \rceil - 1, \lceil \frac{n_r}{2} \rceil\}$, is selected in LFSR R_r as input for the clock control. The GSM standard uses the parameters $(n_0, n_1, n_2) = (19, 22, 23)$ and $(N^0, N^1, N^2) = (11, 12, 13)$.

In each timestep t , the keybit y_t is produced as XOR of the current output bits of the LFSRs. Then, the clock control reads the contents $q_{N^0}^0, q_{N^1}^1, q_{N^2}^2$ of the register cells q_{N^0}, q_{N^1} and q_{N^2} and clocks the LFSR R_r if and only if $q_{N^r}^r = \text{maj}_3(q_{N^0}^0, q_{N^1}^1, q_{N^2}^2)$ where $\text{maj}_3 : \{0, 1\}^3 \rightarrow \{0, 1\}$ is defined to output $c \in \{0, 1\}$ if and only if at least 2 of the 3 arguments have value c .

5.2 Simulation of the A5/1 Generator by an LFSR-based Keystream Generator

5.2.1 Basic Construction

In order to apply the FBDD-based cryptanalysis to the A5/1 generator, we need to simulate the A5/1 generator by an LFSR-based keystream generator as defined in the introductory section.

Let L' denote a linear bitstream generator that consists of three LFSRs L^0, L^1, L^2 corresponding to the LFSRs R_0, R_1 and R_2 in the original generator, i.e. for $r \in \{0, 1, 2\}$ the length and the initial state of L^r are equal to those of R_r . Thus, L' produces a bitstream

$$z = L'(x) = z_0^0, z_1^0, z_2^0, \dots, z_s^0, z_s^1, z_s^2, \dots \quad \text{with } z_s^r = L_s^r(x^r)$$

Additionally, let C' denote a compression function that stores for $r \in \{0, 1, 2\}$ in $i[r]$ the current output position and in $j[r]$ the current control position in the internal bitstream produced by L' . In each Iteration t , C' outputs the new keybit $y_t := z_{i[0]}^0 \oplus z_{i[1]}^1 \oplus z_{i[2]}^2$, computes the new control value $p := \text{maj}_3(z_{j[0]}^0, z_{j[1]}^1, z_{j[2]}^2)$ and updates $i[r]$ and $j[r]$ according to $i[r] := i[r] + 1$ if $z_{j[r]}^k = p$ and $j[r] := j[r] + 1$ if $z_{j[r]}^k = p$.

$i[r]$ and $j[r]$ are initialized with 0 and N^r , respectively. Obviously, the non-linearity of maj_3 ensures the non-linearity of C' . It can be easily checked that for each initial state, the keystream produced by $K' = (L', C')$ is equal to the keystream computed by the original A5/1 generator.

5.2.2 Ensuring the read-once Property of the Compression Function

The compression function C' uses each internal bit z_s^r for the computation of the control bit p and again a few iterations later for producing the keybits. However, in order to construct the oracle graph G_m^C and the G_m^C -FBDDs, we need to represent the A5/1 generator by an LFSR-based keystream generator $K = (L, C)$, where C reads each internal bit at most once.

The linear bitstream generator L extends L' by three LFSRs L^3, L^4, L^5 . The LFSRs L^0, L^1, L^2 are now used exclusively for producing the keybits whereas the control value in each clock cycle is calculated from the three additional LFSRs. L^3, L^4, L^5 therefore correspond to L^0, L^1 and L^2 , shifted by N^0, N^1 and N^2 , respectively. More precisely, for $r \in \{0, 1, 2\}$ it holds that $n_{3+r} = n_r$ and

$$\begin{aligned} L_s^{3+r}(x) &= L_{s+N^r}^r(x) \text{ for all } s \geq 0 \\ \text{or equivalently } L_s^r(x) &= L_{s-N^r}^{3+r}(x) \text{ for all } s \geq N^r \end{aligned}$$

The initial state of L^{3+r} is equal to the initial state of L^r shifted by N^r positions, i.e. if L^r has the initial state $x^r = (x_0, \dots, x_{n_r-1})$, then the initial state of L^{3+r} is given by

$$\begin{aligned} x^{3+r} &= (L_{N^r}^r(x^r), \dots, L_{n_r+N^r-1}^r(x^r)) \\ &= (x_{N^r}, \dots, x_{n_r-1}, L_{n_r}^r(x^r), \dots, L_{n_r+N^r-1}^r(x^r)) \end{aligned}$$

Thus, L computes a bitstream $z = L(x) = z_0^0, \dots, z_0^5, \dots, z_s^0, \dots, z_s^5, \dots$ with $z_s^r = L_s^r(x^r)$ and $z_s^r = z_{s-N^r}^{3+r}$ for all $r \in \{0, 1, 2\}$ and $s \geq N^r$.

In contrast to C' , the compression function C only stores the current read position $i[r]$ for each $r \in \{0, 1, 2\}$ and computes in each iteration t the keybit y_t as $y_t := z_{i[0]}^0 \oplus z_{i[1]}^1 \oplus z_{i[2]}^2$

and the control value p by $p := \text{maj}_3(z_{i[0]}^3, z_{i[1]}^4, z_{i[2]}^5)$. The current read position $i[r]$ is updated according to $i[r] := i[r] + 1$ if $z_{i[r]}^{3+r} = p$.

The constructions of L and C ensure that a single internal bit is not read during the computation of the control value and again for computing the keybits, but in case a read position $i[r]$ is not incremented in iteration t , the internal bit $z_{i[r]}^r$ is used for the computation of both y_t and y_{t+1} . A similar argument holds for $z_{i[r]}^{3+r}$. Thus, in order to ensure the read-once property, bits that are reused in subsequent cycles have to be stored in a suitable way.

Algorithm 2 computes the keystream $y = C(z)$ from a given internal bitstream z without reading an internal bit z_s^r more than once. The computation of y is split into a function *output* for producing the keybits and a function *control* that computes the control value. The definition of maj_3 implies that in each iteration, there exists at most one $r \in \{0, 1, 2\}$ such that $i[r]$ is not incremented. The algorithm stores this unchanged index in the variable u , the corresponding input value for the control bit computation $z_{i[r]}^{3+r}$ in the variable v and the keybit input value $z_{i[r]}^r$ in the variable w .

In order to simplify the notation, Algorithm 2 assumes the sum r of two arrays p and q of length n to be defined component-wise, i.e. $r[i] = (p + q)[i] := p[i] + q[i]$ for all $i \in \{0, \dots, n - 1\}$.

5.3 FBDD-Constructions

Having a read-once algorithm for C at hand, we can now construct the oracle graph G_m^C and the G_m^C -FBDDs Q_m , R_m and S_m .

An algorithm $A(G_m^C)$ that for a given internal bitstream $z \in \{0, 1\}^m$ computes the order in which the bits of z are read by the compression function can easily be derived from Algorithm 2. In fact, $A(G_m^C)$ is slightly simpler than Algorithm 2 since it need not actually compute the keybits and can therefore omit maintaining the variables *out* and *w*. Obviously, the space that $A(G_m^C)$ requires in addition to the input is logarithmic in m . Theorem 1 therefore yields $|G_m^C| \in m^{O(1)}$.

Similarly, we can transform Algorithm 2 into an algorithm $A(Q_m)$ that decides for a given internal bitstream $z \in \{0, 1\}^m$ whether $C(z)$ is prefix of a given keystream $y \in \{0, 1\}^*$. We only need to replace the output of the computed keybit by a test whether the computed keybit matches the corresponding bit in the given keystream. Again, based on the logarithmically bounded space requirements of the resulting algorithm, Theorem 1 yields $|Q_m| \in m^{O(1)}$.

Giving explicit constructions for G_m^C and Q_m , [Ste04] obtained the following tighter bounds on $|G_m^C|$ and $|Q_m|$:

Lemma 6. *For the oracle graph G_m^C and the G_m^C -FBDD Q_m of the A5/1 keystream generator, it holds that $|G_m^C| \in O(m^3)$ and $|Q_m| \in O(m^4)$. \square*

In the general case, the G_m^C -FBDDs R_m and S_m are independent of the compression function and can therefore be designed in a generic way (see [Kra02], [Ste04]). However, for the A5/1 generator the interdependences of the LFSRs in L demand a slightly more evolved construction.

In order to decide for a given $z \in \{0, 1\}^m$ whether $z = L_{\leq m}(i_m(L, z))$, we first need to check the following linearity condition:

$$z_s^r = \bigoplus_{j=0}^{n_r-1} L_{j,s}^r \cdot z_j^r \text{ for } r \in \{0, \dots, 5\} \text{ and } s \geq n_r \quad (3)$$

Since for $r \in \{0, 1, 2\}$ the output bitstream of LFSR L^{3+r} corresponds to the shifted output bitstream of L^r , we additionally need to check whether

$$z_s^r = z_{s-N^r}^{3+r} \text{ for } r \in \{0, 1, 2\} \text{ and } s \geq N^r \quad (4)$$

Algorithm 2 $C(z)$

```
// compute the output bitstream  $y$  from the given internal bitstream  $z$  of length  $m$ 
//  $z$  is interpreted as  $z = z_0^0, \dots, z_0^5, \dots, z_s^0, \dots, z_s^5, \dots, z_{m-1}^{m-1 \bmod 6}, \dots, z_{m-1}^{m-1 \div 6}$ 
output([0, 0, 0], NIL, NIL, NIL)
return
function output( $i, u, v, w$ )
//  $i$ =current read position
//  $u$ =unchanged index  $\in \{0, 1, 2, NIL\}$ ,  $v$ =unchanged control value  $\in \{0, 1, NIL\}$ 
//  $w$ =unchanged output value  $\in \{0, 1, NIL\}$ 
if  $\exists r \in \{0, 1, 2\} \setminus \{u\} : 6 \cdot i[r] + r \geq m$  then
  stop
  Read  $\leftarrow \{0, 1, 2\} \setminus \{u\}$ 
  Let  $r_0, \dots, r_{|Read|-1}$  the elements of Read in ascending order, i.e.  $r_m < r_n$  for  $m < n$ 
  newOut[ $r_0$ ]  $\leftarrow z_{i[r_0]}^{r_0}$ 
  newOut[ $r_1$ ]  $\leftarrow z_{i[r_1]}^{r_1}$ 
  if  $u \neq NIL$  then //  $\exists$  an unchanged index
    newOut[ $u$ ]  $\leftarrow w$  // copy the unchanged output value
  else // all read positions incremented
    newOut[ $r_2$ ]  $\leftarrow z_{i[r_2]}^{r_2}$  // read the third output value
  keybit  $\leftarrow$  newOut[0]  $\oplus$  newOut[1]  $\oplus$  newOut[2]
  write keybit
  control( $i, u, v, newOut$ )
end function
function control( $i, u, v, out$ )
// out[0..2]=current output values of the LFSR  $L^0, L^1, L^2$ ,  $out[r] \in \{0, 1\}$ 
if  $\exists r \in \{0, 1, 2\} \setminus \{u\} : 6 \cdot i[r] + 3 + r \geq m$  then
  stop
  Read  $\leftarrow \{0, 1, 2\} \setminus \{u\}$ 
  Let  $r_0, \dots, r_{|Read|-1}$  the elements of Read in ascending order, i.e.  $r_m < r_n$  for  $m < n$ 
  c[ $r_0$ ]  $\leftarrow z_{i[r_0]}^{3+r_0}$ 
  c[ $r_1$ ]  $\leftarrow z_{i[r_1]}^{3+r_1}$ 
  if  $u \neq NIL$  then //  $\exists$  an unchanged index
    c[ $u$ ]  $\leftarrow v$  // copy the unchanged control value
  else // all read positions incremented
    c[ $r_2$ ]  $\leftarrow z_{i[r_2]}^{3+r_2}$  // read the third control value
  controlbit  $\leftarrow$  maj3(c[0], c[1], c[2])
  if  $\exists r \in \{0, 1, 2\} : c[r] \neq controlbit$  then
    // By definition of maj3,  $\exists$  at most one such  $r$ .
    newU  $\leftarrow r$  // set unchanged index
    newV  $\leftarrow controlValue \oplus 1$  // set unchanged control value
    newW  $\leftarrow out[r]$  // set unchanged output value
  else // all read positions incremented
    newU  $\leftarrow NIL$ 
    newV  $\leftarrow NIL$ 
    newW  $\leftarrow NIL$ 
  for  $l = 0, 1, 2$  do
     $\Delta i[l] \leftarrow \begin{cases} 0 & \text{for } l = newU \\ 1 & \text{for } l \neq newU \end{cases}$ 
  output( $i + \Delta i, newU, newV, newW$ )
end function
```

It follows immediately from Algorithm 2, that for any reading order π defined by G_m^C for the variables z_0, \dots, z_{m-1} , $z_s^r = z_{6 \cdot s + r}$, it follows that

$$\pi(z_{s-N^r}^{3+r}) < \pi(z_s^r) \text{ for } r \in \{0, 1, 2\} \text{ and } s \geq N^r,$$

i.e. on each path in G_m^C , the control bit $z_{s-N^r}^{3+r}$ is read before the corresponding output bit z_s^r . Hence, if all z_s^{3+r} with $r \in \{0, 1, 2\}$ and $s \geq n_r$ fulfil the linearity condition, and the shift condition holds for all z_s^r , $r \in \{0, 1, 2\}$ and $s \geq N^r$, then these z_s^r automatically fulfil the linearity condition. Only for the first N^r linearly combined bits of the LFSRs L^0 , L^1 and L^2 not covered by the shift condition, the linearity condition has to be checked explicitly. In order to decide whether a given internal bitstream z can be constructed by L , it therefore suffices to check the following requirements:

$$z_s^r = \bigoplus_{j=0}^{n_{r-3}-1} L_{j,s}^r \cdot z_j^r \text{ for } r \in \{3, 4, 5\} \text{ and } s \geq n_{r-3} \quad (5)$$

$$z_s^r = \bigoplus_{j=0}^{n_r-1} L_{j,s}^r \cdot z_j^r \text{ for } r \in \{0, 1, 2\} \text{ and } n_r \leq s < n_r + N^r \quad (6)$$

$$z_s^r = z_{s-N^r}^{3+r} \text{ for } r \in \{0, 1, 2\} \text{ and } s \geq N^r \quad (7)$$

Based on these observations, the following bound on the size of R_m can be shown (see [Ste04]):

Lemma 7. *For the G_m^C -FBDD R_m of the A5/1 keystream generator it holds that $|R_m| \leq |G_m^C| 2^{m-|I_m(L)|}$. \square*

The G_m^C -FBDD S_m , which decides for an internal bitstream $z = (z_0, \dots, z_{m-1})$ whether $z_{m-1} = L_{m-1}(i_m(L, z))$, has to store only one comparison bit for $m-1 \bmod 6 \in \{3, 4, 5\}$ and at most two comparison bits in case $m-1 \bmod 6 \in \{0, 1, 2\}$. S_m can therefore be directly derived from G_m^C , and we obtain:

Corollary 8. *For the G_m^C -FBDD S_m of the A5/1 keystream generator it holds that $|S_m| \leq 4 \cdot |G_m^C|$. \square*

The A5/1 generator therefore fulfils the FBDD-Assumption.

5.4 Theoretical Results

Algorithm 2 implies that the number of keybits produced from an internal bitstream of length m is maximum, if four internal bits are consumed for producing each keybit. The best-case compression ratio of the A5/1 keystream generator is therefore $\gamma = \frac{1}{4}$. Moreover, the A5/1 generator can be shown to have an information ratio $\alpha \approx 0.2193$ and to fulfil the independence assumption (see [Kra02], [Ste04]). Applying the general result for FBDD-based attacks (Theorem 5) to the A5/1 generator yields

Theorem 9. *From the first $[1, 14n]$ consecutive keybits of an A5/1 keystream generator of keylength n , an initial state \tilde{x} producing the observed keystream can be computed in time and space $n^{O(1)} 2^{0.6403n}$. \square*

6 Experimental Results

6.1 Implementation Aspects

In order to provide a fast implementation of the FBDD algorithms, an FBDD-library was developed based on the publicly available OBDD package CUDD (see [Som01]). Using straightforward bottom-up algorithms, the oracle graph G_m^C and the G_m^C -FBDDs Q_m and S_m can be constructed in linear time.

When applying the generic cryptanalysis algorithm to the A5/1 generator, we need to modify Algorithm 1 in two ways. First, the shift condition (4) implies that we additionally need to include the G_m^C -FBDDs S_m for $m < n'$ in order ensure that this condition is met by the internal bitstreams. Moreover, since the A5/1 generator is irregularly clocked, for each $m \geq 1$ there exist internal bitstreams $z \in \{0, 1\}^m$ and corresponding paths in G_m^C such that some of the internal bits in $\{z_0, \dots, z_{m-1}\}$ are only considered by the compression function C in subsequent iterations when z is extended by further bits. In order to enforce the restrictions of S_m on these bits, we always work with $G_{\lceil \alpha^{-1}n \rceil}$ instead of G_m^C in the construction of S_m and when applying the SYNTH algorithm. Note that the asymptotic analysis of Algorithm 1 continues to hold for the modified version of the algorithm. We refer the reader to [Ste04] for details on the implementation of both FBDD library and the cryptanalytic algorithms.

The experiments were run on a standard Linux-PC with two 3,4 GHz Intel Pentium 4 processors, an Intel D865PERLX motherboard with FSB 800 and two 512 MB PC-400 DDR-SDRAM modules. All implementation was done in C using the gcc-compiler version 3.3.3.

On the given hardware, our implementation is able to effectively attack generators with keylengths of up to $n = 28$ in less than 15 minutes. However, for keylengths above this value, the size of the intermediate FBDDs forces the system to utilize secondary memory. Due to the nearly random memory access patterns of the FBDD algorithms, attacks for larger keylengths turned out to be practically infeasible in the limits of the available main memory.

6.2 Size of P_m for increasing Keylengths n

Since the runtime of the cryptanalysis depends fundamentally on the maximum size of the intermediate FBDDs P_m , we investigate how much experimentally obtained values of $|P_m|$ deviate from the theoretical figures.

We consider A5/1 generators of keylengths between 13 and 28 bits, each one containing LFSRs of lengths between 3 and 14. All considered feedback polynomials are primitive and were obtained from [Cha]. Similarly to [Sch02], we distinguish dense and sparse polynomials and consider a polynomial as sparse if less than one half of the coefficients has value 1 and dense otherwise. Since there exist only few primitive polynomials of low degree, we define in these cases the polynomials to be both dense and sparse. For each keylength $n \in \{13, \dots, 16\}$, we randomly generate 4 A5/1 generators and 10 generators for $n \in \{17, \dots, 28\}$ by choosing the lengths n_1, n_2 of the first two LFSRs randomly from $[\frac{n}{3} - 3, \dots, \frac{n}{3} + 3]$ and defining the length n_2 of the remaining LFSR as $n_2 := n - n_0 - n_1$. For each of these generators, we perform $p(n)$ attacks, where $p(n) = 15$ for $n \leq 16$, $p(n) = 10$ for $n \in \{17, \dots, 26\}$ and $p(n) = 5$ for $n \in \{27, 28\}$. Each attack is performed in the following way: First, we randomly choose a non-zero initial state $x \in \{0, 1\}^n$ and let the generator produce at least $\lceil \gamma \alpha^{-1} n \rceil$ keybits. We feed these keybits into the cryptanalysis algorithm and record the maximum size of P_m that occurs during the computation of a consistent initial state \tilde{x} .

Figures 1 and 2 compare the obtained values to the asymptotic theoretical bound

$$\max\{|P_m|\} \leq \max_{1 \leq m \leq \lceil \alpha^{-1}n \rceil} \left\{ \min\{m^{10} \cdot 2^{m-n^*}, m \cdot 2^{n^* - \alpha m}\} \right\}$$

which was obtained in the proof of Lemma 3.

The average gradients for sparse and dense polynomials are 0.72 and 0.71, respectively. Thus, with each additional keybit, the maximum size of P_m increases by a factor of 1.65. Similarly to the experiments run by [Sch02], the minimum and maximum values are relatively close and only deviate from the theoretical bound by a constant factor.

6.3 Size of P_m for increasing m

We consider 10 different generators of keylength 20 and for each of these generators, we randomly choose 10 initial states and feed the computed keystreams into the cryptanalysis

algorithm. In this experiment, the algorithm runs 120 iterations, thus exceeding $m^* = \lceil \alpha^{-1}n \rceil = \lceil 0.2193^{-1} \cdot 20 \rceil = 92$ by almost 30 iterations. Figures 3 and 4 show that the graph for $|P_m|$ has the same shape as expected by theory, although the size of P_m seems to decrease faster for dense polynomials than for sparse polynomials. Similarly to the observations in [Sch02], the distance between the minimum and the maximum sizes is considerably larger for sparse polynomials than for dense polynomials.

7 Summary

In this article, we described the application of the generic FBDD-based attack of [Kra02] to the A5/1 keystream generator and presented experimental results for smaller instances of the generator. As long as sufficient main memory is available, the generator can be efficiently attacked within minutes, but as soon as secondary memory has to be used, the cryptanalysis becomes practically infeasible. With the hardware resources that were available for our experiments, A5/1 instances of keylengths greater than 28 can therefore not be processed within a reasonable amount of time. Particularly the original A5/1 generator, whose keylength is more than twice this threshold, can be considered practically secure from this point of view. However, the algorithm can possibly be improved in the direction of time/space tradeoffs in the sense of getting around the explicit construction of all intermediate FBDDs by allowing more time-consuming operations, which might yield a better overall performance, especially on systems with moderate main memory.

References

- [BGW99] M. Briceno, I. Goldberg, and D. Wagner. *A pedagogical implementation of A5/1*, May 1999. <http://jya.com/a51-pi.htm>.
- [blu] *Bluetooth specification version 1.1*. <http://www.bluetooth.com>.
- [Cha] F. Chabaud. Primitive polynomials over GF(2). URL: <http://fchabaud.free.fr/English/default.php>.
- [Gol82] S. W. Golomb. *Shift Register Sequences*. Aegean Park Press, Laguna Hills, California, USA, 1982.
- [Kra02] M. Krause. BDD-based cryptanalysis of keystream generators. In *EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 222–237. Springer Verlag, Heidelberg, 2002.
- [Mei89] C. Meinel. *Modified Branching Programs and Their Computational Power*, volume 370 of *Lecture Notes in Computer Science*. Springer, 1989.
- [MS95] W. Meier and O. Staffelbach. The self-shrinking generator. In *Advances in Cryptology-EUROCRYPT'94*, volume 950 of *Lecture Notes in Computer Science*, pages 205–214, 1995.
- [Sch02] F. Schleer. Einsatz von OBDDs zur Kryptanalyse von Flusschiffren (in german). Master's thesis, University of Mannheim, Mannheim, Germany, 2002.
- [Som01] F. Somenzi. *CUDD: CU decision diagram package*. University of Colorado, Boulder, CO, USA, March 2001. <http://vlsi.colorado.edu/~fabio/>.
- [Ste04] D. Stegemann. Fbdd-basierte Kryptanalyse des A5/1 Schlüsselstromgenerators (in german). Master's thesis, University of Mannheim, Mannheim, Germany, 2004.
- [Weg00] I. Wegener. *Branching Programs and Binary Decision Diagrams: Theory and Applications*. SIAM Monographs on Discrete Mathematics and Applications, 2000.

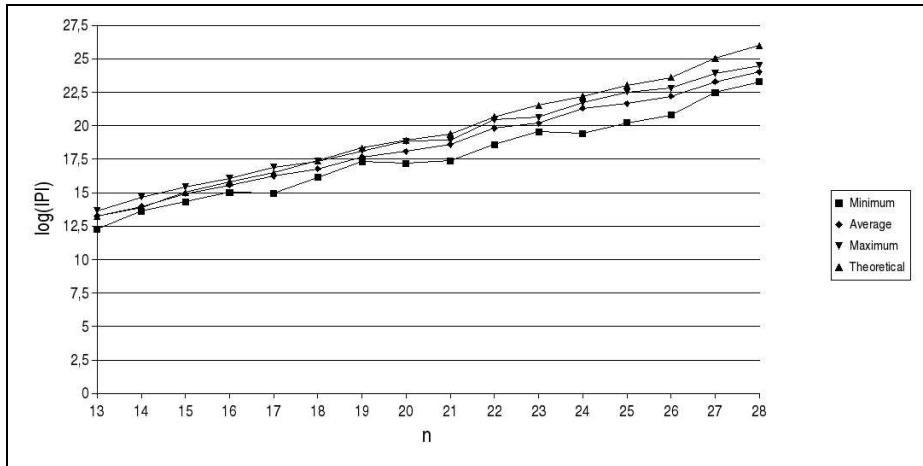


Figure 1: Maximum size of P_m for increasing keylengths n and sparse feedback polynomials

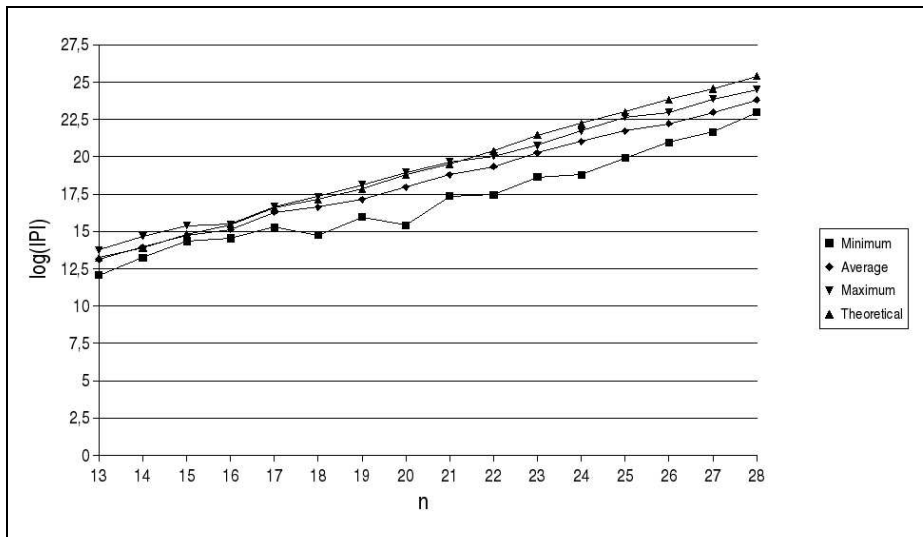


Figure 2: Maximum size of P_m for increasing keylengths n and dense feedback polynomials

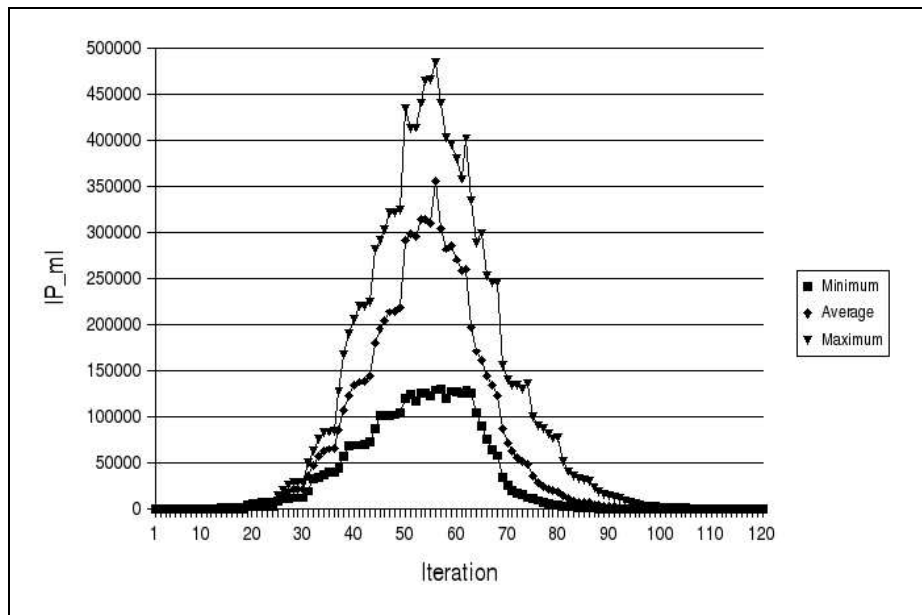


Figure 3: Size of P_m for increasing m and sparse feedback polynomials

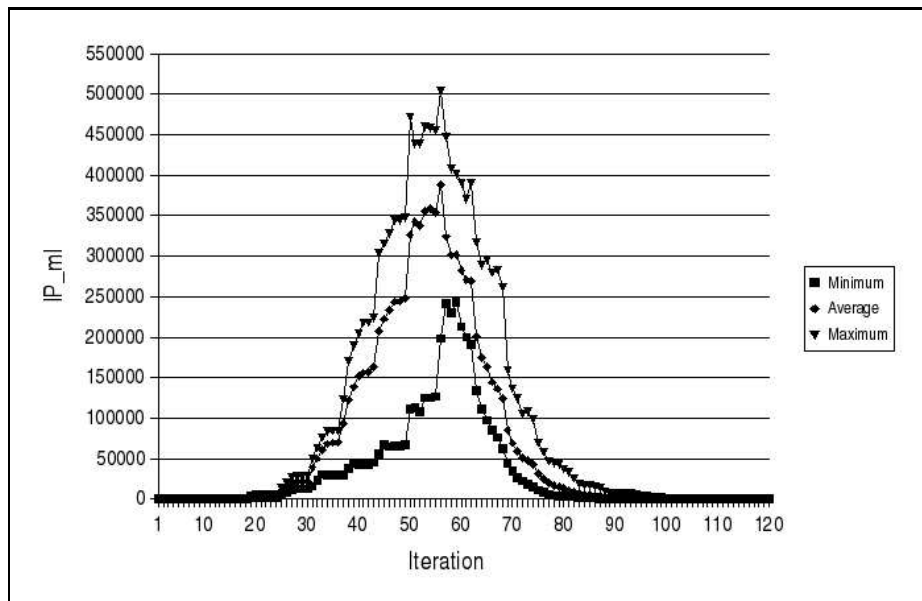


Figure 4: Size of P_m for increasing m and dense feedback polynomials