Math-Net.Ru

Общероссийский математический портал

М. Hamann, М. Krause, А. Moch, Улучшение атак на поточные шифры, использующих диаграммы последовательных бинарных решений, *Матем. вопр. криптогр.*, 2020, том 11, выпуск 2, 137–151

DOI: <https://doi.org/10.4213/mvk327>

Использование Общероссийского математического портала Math-Net.Ru подразумевает, что вы прочитали и согласны с пользовательским соглашением
<http://www.mathnet.ru/rus/agreement>

Параметры загрузки:

IP: 107.22.136.117

26 апреля 2023 г., 18:07:10

Improving OBDD attacks against stream ciphers

M. Hamann, M. Krause, A. Moch

Universität Mannheim, Germany

Получено 05.XI.2019

Abstract. OBDD-attacks against stream ciphers compute the secret initial state by generating a sequence of $\mathcal{O}(n)$ ordered binary decision diagrams (OBDDs) of maximal width $\mathcal{O}(2^{\frac{1-\alpha}{1+\alpha}n})$, where n denotes the inner state length and $\alpha \in (0, 1)$ is the *compression rate* of the cipher. We propose and experimentally verify the following strategy of avoiding the huge storage demand of $\mathcal{O}(2^{\frac{1-\alpha}{1+\alpha}n})$. (1) Generate in parallel two OBDDs P and Q such that $P \wedge Q$ has only a few satisfying assignments. (2) Compute the set $(P \wedge Q)^{-1}(1)$, containing the secret inner state, by a new breadth-first-search based algorithm. We show that this approach improves standard OBDD-attacks drastically.

Keywords: symmetric cryptography, stream ciphers, OBDD attacks

Улучшение атак на поточные шифры, использующих диаграммы последовательных бинарных решений

М. Хаманн, М. Краузе, А. Мох

Университет Мангейма, Германия

Аннотация. При атаках на поточные шифры диаграммами последовательных бинарных решений (ДПБР) секретное начальное состояние вычисляется с помощью набора из $\mathcal{O}(n)$ ДПБР максимальной ширины $\mathcal{O}(2^{\frac{1-\alpha}{1+\alpha}n})$, где n — длина внутреннего состояния, а $\alpha \in (0, 1)$ — *степень сжатия* шифра. Предложен и экспериментально проверен следующий способ обходиться без памяти большого объема $\mathcal{O}(2^{\frac{1-\alpha}{1+\alpha}n})$. (1) Строить одновременно пары таких ДПБР P и Q , что для $P \wedge Q$ существует мало решений. (2) Вычислять множество $(P \wedge Q)^{-1}(1)$, содержащее секретное внутреннее состояние, новым алгоритмом, основанным на поиске в ширину. Показано, что этот подход существенно улучшает характеристики стандартных ДПБР-атак.

Ключевые слова: симметричная криптография, поточные шифры, диаграммы последовательных бинарных решений

1. Introduction

Stream ciphers are symmetric encryption algorithms intended for the online encryption of plaintext bitstreams X which have to pass an insecure channel. The encryption is performed by bitwise addition of a keystream S , which is generated in dependence of a secret symmetric session key k and, possibly, a public initial value IV . The legal recipient, who also knows k , decrypts the encrypted bitstream $Y = X \oplus S$ by generating S and computing $X = Y \oplus S$. In this paper, we consider KSG-based stream ciphers, i.e., stream ciphers that generate the keystream using a so-called keystream generator (KSG).

KSGs are stepwise working devices that may be formally specified by finite automata. KSGs are defined by an inner state length n and the corresponding set of inner states $\{0, 1\}^n$, a state update function $\pi: \{0, 1\}^n \rightarrow \{0, 1\}^n$ and an output function $\text{out}: \{0, 1\}^n \rightarrow \{0, 1\}$. Starting from an initial state $q_0 \in \{0, 1\}^n$, in each clock cycle $i \geq 0$, the KSG produces a keystream bit $z_i = \text{out}(q_i)$ and changes the inner state according to $q_{i+1} = \pi(q_i)$. The output bitstream $S(q_0)$ is defined by concatenating all the outputs $z_1 z_2 z_3 \dots$.

The main security requirement for stream ciphers is the following: when a secret initial state is chosen randomly, it must be hard to distinguish the generated keystream from a truly random bitstream. This implies the hardness of the problem S^{-1} : given a piece z of keystream of length $\ell \geq n$, compute an initial state q which generates z in the sense that z is a prefix of $S(q)$. In this paper, we focus on analyzing the security of KSGs with regard to S^{-1} -attacks, which try to solve the S^{-1} -problem for a given piece z of keystream.

The main building blocks of many KSG-constructions are so-called Feedback Shift Registers (FSRs). FSRs are defined over a number m of register cells and a feedback function $f: \{0, 1\}^m \rightarrow \{0, 1\}$. In each clock cycle, given the inner state $(b_1, \dots, b_m)$, the bit b_1 is the output bit and the state changes to $(b_2, \dots, b_{m-1}, f(b_1, \dots, b_m))$. The FSR is called a linear FSR (LFSR) if the feedback function is $GF(2)$ -linear, and it is called a nonlinear FSR (NFSR) otherwise.

The output sequences defined by LFSRs with a primitive connection polynomial have several very useful pseudorandomness properties, e.g., the maximal period of $2^m - 1$. However, an LFSR alone does not represent a secure KSG as the problem S^{-1} may be efficiently solved by inverting an $(n \times n)$ -matrix.

Many KSG-constructions use a small number of FSRs (LFSRs or

NFSRs) to generate a secret inner bitstream, which has to pass a non-linear filter function to produce the output keystream. This allows to split the keystream generation process into two parts: (1) the generation of the inner bitstream $\text{IB}(q)$ from an initial state q , consisting of the bits produced by the FSRs, and (2) the generation of the public output keystream $S(q) = C(\text{IB}(q))$ that is generated from the secret inner bitstream $\text{IB}(q)$ using an online compression function C . The compression rate of a keystream generator is defined to be $\alpha = 1/A$, where A denotes the average number of inner keystream bits needed to produce one output bit. Note that the secret initial state q is a part (often the prefix) of the inner bitstream $\text{IB}(q)$.

Concerning S^{-1} -attacks against stream ciphers, one distinguishes short-keystream attacks (e.g., OBDD-attacks) and long-keystream attacks (e.g., time-memory-data tradeoff (TMD-TO) attacks). Short-keystream attacks solve the S^{-1} -problem where only a small sequence of keystream bits was observed. In this case, the sequence is usually not significantly longer than n . Long-keystream attacks require a lot more keystream bits (e.g., $2^{n/2}$ for the TMD-TO attacks of Babbage [1] and Golić [5]) to solve the S^{-1} -problem.

Ordered binary decision diagrams (OBDDs) are a graph based data structure for representing Boolean functions. Due to their specific algorithmic properties, there is a wide range of practical applications of OBDDs, especially in the field of hardware verification (see, e.g., [9]). OBDD-attacks were introduced by Krause in [6] and further studied in, e.g., [4, 7, 11]. They represent the most efficient kind of short-keystream attack against stream ciphers. The number of necessary keystream bits nearly matches the information-theoretic lower bound, which corresponds to the unicity distance of the cipher.

OBDD-attacks refer to the decision if for a given initial state q , a given piece of inner bitstream y and a given piece of keystream z it holds that y is prefix of $\text{IB}(q)$ and $C(y)$ is prefix of z . For many KSGs, this problem may be formulated as a set $\mathcal{R} = \{R_1, \dots, R_t\}$ of easy relations between the q -bits and the y -bits, and between the y -bits and the z -bits, which all may be tested by small OBDDs.

OBDD-attacks compute the OBDDs $Q(I)$ for increasing subsets $I \subseteq \{1, \dots, t\}$, which test if all relations R_i , $i \in I$, are fulfilled. They are based on certain algorithmic properties of OBDDs:

- The conjunction $P \wedge Q$ of two OBDDs may be computed within time and space $\mathcal{O}(|P| \cdot |Q|)$.

- OBDDs P may be efficiently minimized in time $\mathcal{O}(|P|)$.
- The width of a minimized OBDD P is bounded from above by the amount of satisfying assignments of P .

In the standard OBDD-attack, one generates a sequence $Q(I_1) \rightarrow Q(I_2) \rightarrow \dots \rightarrow Q(I_s)$, where $Q(I_1)$ is small and the number of satisfying assignments of $Q(I_1)$ is bounded by $2^{(1-\alpha)n}$. For $j = 2$ to s , I_j is derived from I_{j-1} by adding one new relation from $\mathcal{R}$. This implies that on average $|Q(I_j)^{-1}(1)| \leq 2^\alpha \cdot |Q(I_{j-1})^{-1}(1)|$ and $\text{width}(Q(I_j)) \leq \min\{2 \cdot \text{width}(Q(I_{j-1})), |Q(I_j)^{-1}(1)|\}$. It follows that on average after $r = \frac{1-\alpha}{1+\alpha}n$ iterations the maximal width of $\mathcal{O}(2^{\frac{1-\alpha}{1+\alpha}n})$ is reached and that after $\frac{1}{\alpha}n$ iterations we obtain an OBDD for which the secret initial state is the only satisfying assignment, i.e., which solves the problem.

OBDD-attacks (and their generalization based on free binary decision diagrams (FBDDs)) yield the best known short-keystream attacks against several practically used ciphers such as the A5/1-generator included in the GSM-standard [3], the E_0 -generator included in the Bluetooth standard [2], or the self-shrinking generator [8].

1.1. Our results

We propose the following modification of OBDD-attacks for circumventing the bottleneck consisting in the huge amount of storage needed for the intermediate OBDDs in the iterations near the critical round number $r = \frac{1-\alpha}{1+\alpha}n$.

- (1) Generate two disjoint subsets I and J of relations from $\mathcal{R}$ in parallel such that the OBDDs $Q(I)$ and $Q(J)$ are of moderate size and the set $Q(I \cup J)^{-1}(1)$ of satisfying assignments of $Q(I) \wedge Q(J) = Q(I \cup J)$ is small. Then compute $Q(I \cup J) = Q(I) \wedge Q(J)$ using the standard OBDD-synthesis algorithm.
- (2) Suppose that strategy (1) leads to sets I and J for which the set $Q(I \cup J)^{-1}(1)$ is not just “small” but even comprises of only a *single* element. Then compute this satisfying assignment directly from $Q(I)$ and $Q(J)$ by means of our new breadth-first-search-based algorithm instead of performing the OBDD synthesis $Q(I) \wedge Q(J)$.

The first part of our experimental results, presented in Section 4, shows that strategy (1) is a lot more space efficient than computing $Q(I \cup J)$ using the classical OBDD-attack suggested in [6] (and employed in all follow-up

works since). Strategy (2) allows us to investigate the complexity of the following *Bounded Synthesis Problem*: given two OBDDs P_1 and P_2 for which it is known that $P_1 \wedge P_2$ has only one satisfying assignment, compute this satisfying assignment.

The standard solution is to compute $P_1 \wedge P_2$ using the standard OBDD-synthesis algorithm and then to minimize the resulting OBDD. Note that for solving the bounded synthesis problem, it is sufficient to find the only existing directed path connecting the root to the 1-sink of $P_1 \wedge P_2$, as this path corresponds to the only satisfying assignment of $P_1 \wedge P_2$.

In Section 5, we describe a depth-first-search (DFS) approach and a breadth-first-search (BFS) approach for solving the Bounded Synthesis Problem. Our most promising candidate is the BFS-approach as it allows to identify nodes in $P_1 \wedge P_2$ which do not occur in the minimized OBDD.

Structure of the paper: The remaining part of this paper is organized as follows. In Section 2, we first provide the basic information about OBDDs. We do this by considering a subfamily of OBDDs, so-called leveled OBDDs (LOBDDs). The reason for considering LOBDDs is that the relevant algorithmic properties of OBDDs may be explained more easily for LOBDDs, and in our cryptographic context we obtain LOBDDs in a natural way. Note that all of our results about LOBDDs hold also for OBDDs. In Section 3, we describe the classic BDD attack against stream ciphers on the basis of a toy example. Note that like this toy example, also the generators used in our experiments do not provide sufficient cryptographic hardness for modern practical applications. However, due to their feasible inner state size and simple definition, they are well suited for applying and verifying our algorithmic ideas. In Section 4, we describe strategy (1), the intuition behind it in further detail and provide corresponding experimental results. In Section 5, we present our algorithmic approaches for the bounded synthesis problem. Finally, in Section 6, we present the experimental results which compare our DFS- and BFS-approach to the standard synthesis algorithm, and conclude the paper.

2. Preliminaries

2.1. Leveled ordered binary decision diagrams (LOBDDs)

Definition 1. An LOBDD P over $X_n = \{x_1, \dots, x_n\}$ is a directed acyclic labeled Graph $G = (V, E)$ with one root and one sink, where V is the set of vertices and E is the set of edges. The following properties apply to P :

- The set of vertices V is partitioned into $n + 1$ pairwise disjoint levels, $L_1(P), \dots, L_{n+1}(P)$. The nodes in $L_i(P)$, $1 \leq i \leq n$, are labeled x_i .
- $L_1(P)$ contains only the root $\mathbf{root}(P)$, labeled x_1 .
- $L_{n+1}(P)$ contains only the sink $\mathbf{sink}(P)$, labeled 1.
- The edges are labeled with either 0 or 1. The labels correspond to the Boolean value of the respective variable.
- For all $v \in V \setminus \{\mathbf{sink}(P)\}$, there are either two outgoing edges with different labels or there is only one outgoing edge (labeled 0 or 1).
- For each edge $(u, v) \in E$, there exists some level $i \in \{1, \dots, n\}$ such that $u \in L_i(P)$ and $v \in L_{i+1}(P)$.
- The size of P is denoted by $|P|$ and it is defined to be equal to the amount of nodes, i.e., $|P| := |V|$.
- The width of P is denoted by $\mathbf{width}(P)$ and it is defined to be equal to the size of the largest level, i.e., $\mathbf{width}(P) := \max_{1 \leq i \leq n} \{|L_i(P)|\}$.

Fix some $i \in \{1, \dots, n + 1\}$ and fix some node $v \in L_i(P)$. A path leading from the root $\mathbf{root}(P)$ to v corresponds to a unique $\{0, 1\}$ -assignment of the variables $\{x_1, \dots, x_{i-1}\}$. Similarly, a path leading from v to the sink $\mathbf{sink}(P)$ corresponds to a unique $\{0, 1\}$ -assignment of the variables $\{x_i, \dots, x_n\}$. This motivates the following definitions:

Definition 2. For all $i \in \{1, \dots, n + 1\}$ and all nodes $v \in L_i(P)$, we define $\mathbf{Reach}_P(v)$ to be the set of all those $\{0, 1\}$ -assignments of $\{x_1, \dots, x_{i-1}\}$ which correspond to paths from $\mathbf{root}(P)$ to v .

Definition 3. For all $i \in \{1, \dots, n + 1\}$ and all nodes $v \in L_i(P)$, we define $\mathbf{Sat}_P(v)$ to be the set of all those $\{0, 1\}$ -assignments of $\{x_i, \dots, x_n\}$ which correspond to paths from v to $\mathbf{sink}(P)$.

Definition 4. The Boolean function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ computed by P is defined by

$$f(x) = 1 \iff x \in \mathbf{Sat}_P(\mathbf{root}(P)) = \mathbf{Reach}_P(\mathbf{sink}(P)).$$

It may be straightforwardly shown that for each Boolean function $f : \{0, 1\}^n \rightarrow \{0, 1\}$ there exists an LOBDD P over X_n that computes f . Further, there exists an efficient algorithm running in space and time $\mathcal{O}(|P|)$

that minimizes P , i.e., it computes the minimal LOBDD that represents the same function as P . Note that P is minimal if and only if each node of P is reachable from $\text{root}(P)$ and if for all $i \in \{2, \dots, n\}$ and for all $v \neq v' \in L_i(P)$ we have $\text{Sat}_P(v) \neq \text{Sat}_P(v')$. Otherwise v and v' could be merged into one node. We refer the reader to [12] for further details.

For any two LOBDDs P and Q over X_n , we can define the canonical LOBDD $P \wedge Q$ over X_n computing the logical conjunction of the functions computed by P and Q . For defining $P \wedge Q$, we again consider a graph $G = (V, E)$. The set of vertices is partitioned into $n+1$ levels $L_1(G), \dots, L_{n+1}(G)$ and the nodes in $L_i(G)$, $1 \leq i \leq n$, are labeled by x_i .

Definition 5. Let P and Q be two arbitrary LOBDDs over X_n . The LOBDD $G := P \wedge Q$ is defined as follows. For all $i \in \{1, \dots, n+1\}$:

- Level i is defined to be $L_i(G) := L_i(P) \times L_i(Q)$.
- There exists an edge $((u, u'), (v, v')) \in L_i(G) \times L_{i+1}(G)$ labeled $b \in \{0, 1\}$ if and only if (u, v) is an edge labeled b in P and (u', v') is an edge labeled b in Q .

It follows directly that $L_1(G) = \{(\text{root}(P), \text{root}(Q))\}$ and $L_{n+1} = \{(\text{sink}(P), \text{sink}(Q))\}$. Further, the LOBDD $P \wedge Q$ is formed by all nodes of G reachable from the root $(\text{root}(P), \text{root}(Q))$. It is easy to check that $P \wedge Q$ is an LOBDD computing the logical conjunction of the functions computed by P and Q . In general, $P \wedge Q$ is not minimal, even if P and Q are minimal. In the worst case, $P \wedge Q$ has the width $\Theta(\text{width}(P) \cdot \text{width}(Q))$.

The above definition may be straightforwardly generalized to compute the LOBDD $P_1 \wedge \dots \wedge P_k$ for given LOBDDs $P_1, \dots, P_k$. The LOBDD $P_1 \wedge \dots \wedge P_k$ computes the logical conjunction of the functions computed by $P_1, \dots, P_k$. In the worst case, it has the width $\Theta(\text{width}(P_1) \cdot \dots \cdot \text{width}(P_k))$.

2.2. Comparison of LOBDDs and OBDDs

In Subsection 2.1, we defined LOBDDs, which respect the canonical variable ordering $(x_1, x_2, \dots, x_n)$ on all paths from the root to the sink. Similarly, π -LOBDDs may be defined, which respect the variable ordering $(x_{\pi(1)}, x_{\pi(2)}, \dots, x_{\pi(n)})$ on all paths from the root to the sink, where π is some permutation on $\{1, \dots, n\}$. LOBDDs differ from general OBDDs in two ways:

- 1) OBDDs are usually defined to have a 1-sink and a 0-sink. Moreover, each node has exactly two edges, labeled 0 and 1, respectively. A missing

edge in our LOBDD definition is equivalent to an edge pointing to the 0-sink. In fact, this is just a matter of notation.

- 2) General OBDDs do not have to be leveled, i.e., there may be an edge (u, v) from a node u labeled x_i to a node v labeled x_j , where $1 \leq i < j \leq n + 1$. If $j > i + 1$, this implies that the subfunction computed at v does not depend on the variables $x_{i+1}, \dots, x_{j-1}$. In LOBDDs, on contrast, we always demand $j = i + 1$.

Each OBDD P may be easily converted to an LOBDD P' with $|P'| \leq n \cdot |P|$. Consider the edge (u, v) , where u is a node labeled x_i , v is a node labeled x_j , and $j > i + 1$. In the LOBDD, (u, v) is replaced by a sequence of $j - i - 1$ dummy nodes labeled $x_{i+1}, \dots, x_{j-1}$.

Note that if an OBDD P is satisfying the property that for all $x \neq x' \in \text{Sat}_P(\text{root}(P))$ the Hamming distance between x and x' is larger than one, then P has to be an LOBDD. As all functions considered in our cryptographic context have this property, we decided to work with LOBDDs in most parts of the paper (esp. in Section 4) since minimization and the construction of $P \wedge Q$ may be defined more easily in the model of LOBDDs.

3. Classical BDD attacks against stream ciphers

Let us consider the toy example of a simple KSG with inner state size five, whose secret inner bitstream $x_1, x_2, \dots$ is defined by the feedback relation

$$x_{t+5} := x_t \oplus x_{t+2} \quad \text{for } t \geq 1, \quad (1)$$

where $(x_1, \dots, x_5)$ denotes the secret initial state. Moreover, let the output function of this KSG be given as

$$z_t := x_{t+2} \cdot x_{t+4} \quad \text{for } t \geq 1. \quad (2)$$

It is easy to check that for this KSG, the initial state $(x_1, \dots, x_5) = (0, 1, 1, 0, 1)$ leads to the keystream prefix $(z_1, \dots, z_7) = (1, 0, 1, 0, 1, 0, 1)$.

Classical OBDD-based cryptanalysis now proceeds as follows. An attacker who gets hold of the above keystream prefix starts by turning his information about step $t = 1$ into an OBDD. From Eq. (1), he knows that

$$x_1 \oplus x_3 \oplus x_6 = 0 \quad (3)$$

holds w.r.t. the newly generated inner state stream bit x_6 . This knowledge is represented by the OBDD R_1 depicted in Fig. 1, whose satisfying assignments (leading to the OBDD's 1-sink) are those satisfying Eq. (3).

Figure 1: OBDDs R_1 (left), Q_1 (mid.), $P_1 = R_1 \wedge Q_1$ (right). The solid (resp. dashed) edges denote that the variable labeling the edge's source node takes the value 1 (resp. 0)

Also at $t = 1$, the attacker learns from the observed first keystream bit $z_1 = 1$ together with Eq. (2) that

$$x_3 \cdot x_5 = 1 \quad (4)$$

must hold. This knowledge is represented by the OBDD Q_1 in Fig. 1.

Through AND-synthesis of R_1 and Q_1 , the attacker finally obtains the OBDD P_1 depicted in Fig. 1, whose satisfying assignments are exactly those assignments to x_1, x_3, x_5, x_6 which simultaneously fulfill Eqs. (3) and (4).

For the next step, $t = 2$, the attacker proceeds analogously. More precisely, he builds the OBDDs R_2 and Q_2 corresponding to the relations $x_2 \oplus x_4 \oplus x_7 = 0$ and $x_4 \cdot x_6 = 0$ (as $z_2 = 0$), respectively. The new main OBDD P_2 is computed as $P_2 := P_1 \wedge R_2 \wedge Q_2$.

The general attack strategy is now as follows. The attacker will treat the subsequent iterations $t = 3, 4, \dots$ accordingly, obtaining further growing OBDDs P_3, P_4 , and so on. However, as explained in detail in [6], at some point, the size of the OBDDs P_t will eventually reach a maximum and henceforth (usually quickly) decrease. Note that this maximum actually dominates the overall complexity of the attack.

In the case of our toy example, after only seven steps, the main OBDD P_7 has degraded into a list as depicted in Fig. 2. The only satisfying assignment to the first twelve bits of the inner state stream may be derived from P_7

Figure 2: The OBDD P_7 , which contains the solution of our cryptanalysis

directly as $(x_1, \dots, x_{12}) = (0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0)$. The first five bits $(x_1, \dots, x_5)$ are the secret initial state that underlies the attacked keystream prefix and, hence, represent the solution of our OBDD-based cryptanalysis. From this initial state, an attacker may now generate the full keystream.

4. Attack improvement and experimental results

Instead of building only one main OBDD P_t (see our example in Section 3) we now suggest to work with ‘two main OBDDs’ P_t^1 and P_t^2 as follows.

Algorithm 1 A new approach for more efficient, parallelizable OBDD attacks.

$X \leftarrow$ merging point parameter (see explanation below)

$(P_0^1, P_0^2) \leftarrow (1\text{-OBDD}, 1\text{-OBDD})$

for $t = 1$ to X **do**

if t is odd **then**

$(P_t^1, P_t^2) \leftarrow (P_{t-1}^1 \wedge R_t \wedge Q_t, P_{t-1}^2)$

else

$(P_t^1, P_t^2) \leftarrow (P_{t-1}^1, P_{t-1}^2 \wedge R_t \wedge Q_t)$

end if

end for

$\tilde{P}_X \leftarrow P_X^1 \wedge P_X^2$

$t \leftarrow X$

while $\tilde{P}_t$ has more than one satisfying assignments **do**

$t \leftarrow t + 1$

$\tilde{P}_t \leftarrow \tilde{P}_{t-1} \wedge R_t \wedge Q_t$

end while

return $\tilde{P}_t$

The intuition behind the efficiency of our new approach is the following.

Let $t^{\max}$ denote the step in which the single main OBDD P_t would have reached its maximum size $|P_{t^{\max}}|$ and remember that for $t > t^{\max}$, the sizes $|P_t|$ would henceforth constantly decrease until only one satisfying assignment is left. Then, for a properly chosen parameter X (in particular, $X > t^{\max}$), the result $\tilde{P}_X := P_X^1 \wedge P_X^2$ in Algorithm 1 will already have much less nodes than $P_{t^{\max}}$ with the effect that the respective synthesis operation will have advantageous (as compared to working with $P_{t^{\max}}$) expected time and memory consumption. Note that for the new approach to be actually more efficient than the classical one, X must be still small enough such that the sizes of the OBDDs P_X^1 and P_X^2 still stay significantly below $|P_{t^{\max}}|$.

In order to assess the efficiency gains achievable with our new approach, we performed an experimental evaluation using the well-known BDD package CUDD [10]¹. As the attack target, we considered a simple KSG of size 39 bits, whose inner state stream is defined by the relation

$$x_{t+39} := x_t \oplus x_{t+13} \oplus x_{t+5} \cdot x_{t+17} \oplus x_{t+24} \cdot x_{t+29} \quad \text{for } t \geq 1,$$

and whose output function is $z_t := x_{t+9} \oplus x_{t+19} \oplus x_{t+29}$, $t \geq 1$.

The classical approach described in Section 3 leads to a memory consumption of almost 900 MB and takes about 215 seconds. The single main OBDD used there reaches its maximum size at $t^{\max} = 17$. In contrast, the same attack using our new approach in Algorithm 1 (with $X = 22$) requires only about 110 MB of memory and is completed in as few as 8 seconds.

Also note that the OBDDs P_t^1 and P_t^2 are built on the basis of completely separate information. Thus, they could as well be easily computed in parallel on different CPUs in order to further speed up the attack.

5. A related special synthesis problem for LOBDDs and algorithmic approaches

We have seen in the previous sections that given a piece of keystream z , the problem of computing the secret inner state x which generates z may be reduced to computing a sequence $P_1, \dots, P_k$ of (L)OBDDs of a moderate size such that x is a satisfying assignment of $P_1 \wedge \dots \wedge P_k$ and the number of satisfying assignments of $P_1 \wedge \dots \wedge P_k$ is moderately bounded. In Section 4, we described a corresponding approach with $k = 2$ and verified through

¹Detailed setup specification: The attack approaches were implemented on the basis of the C/C++ BDD package CUDD [10], version 2.5.1, ported to Windows. The employed IDE/compiler environment was Visual Studio 2015 - Visual C++ 14.0. The resulting executable programs were run on a Windows 7 (32 bit) machine equipped with an Intel Core i7-3820QM (4x 2.70 GHz, 8MB cache) CPU and 4 GB (1600 Mhz, DDR3) RAM.

experiments that this approach was indeed significantly more efficient than the classical BDD-attack described in Section 3.

The standard way of computing the set of satisfying assignments of $P_1 \wedge \dots \wedge P_k$ from $P_1, \dots, P_k$ is to construct $P_1 \wedge \dots \wedge P_k$ as described in Subsection 2.1 and to minimize it. This algorithm needs space and time $\Theta(|P_1 \wedge \dots \wedge P_k| + \sum_{i=1}^k |P_i|)$, where in the worst case we have $|P_1 \wedge \dots \wedge P_k| = \Theta(\prod_{i=1}^k |P_i|)$. The resulting question is the following:

Can we find more efficient algorithms to compute the set S of satisfying assignments of $P_1 \wedge \dots \wedge P_k$ if $|S|$ is small?

We are convinced that this question represents a fundamental open problem in complexity theory. So far, we cannot yet present an algorithm for this problem which has an asymptotic worst case running time essentially better than $\Theta(\prod_{i=1}^k |P_i|)$. Nonetheless, we are able to come up with new algorithmic approaches which are much simpler than the standard synthesis algorithm and which explicitly use the fact that the set of satisfying assignments of $P_1 \wedge \dots \wedge P_k$ is small. Our experimental results in Section 6 show that our algorithms are more space efficient than the standard synthesis.

In the following, we restrict ourselves to the case that $k = 2$ and $|S| = 1$; i.e., we are given two LOBDDs P_1 and P_2 over the set of Boolean variables $X_n = \{x_1, \dots, x_n\}$ for which we know $|(P_1 \wedge P_2)^{-1}(1)| = 1$. The problem is to compute the only satisfying assignment x of $P_1 \wedge P_2$ in a more efficient way than by the standard synthesis algorithm described above. One obvious consequence of the assumption $|(P_1 \wedge P_2)^{-1}(1)| = 1$ is that we do not have to compute the complete LOBDD $P_1 \wedge P_2$. It is sufficient to compute the only path in $P_1 \wedge P_2$ leading from the root (v_1^0, v_2^0) to the sink (s_1, s_2) .

There are two elementary graph algorithms that compute all nodes of a given directed graph which are reachable from a given starting point: depth-first search (DFS) and breadth-first search (BFS). Hence, we present two algorithmic approaches for computing $(P_1 \wedge P_2)^{-1}(1)$: a *DFS approach* and a *BFS approach*.

5.1. The DFS approach

The underlying data structure is a stack S used to store the nodes of $P_1 \wedge P_2$. The stack maintains a pointer $\text{head}(S)$ to the node on top of the stack. During the execution of the algorithm, some nodes of $P_1 \wedge P_2$ will be labeled *gray* (discovered) and some will be labeled *black* (discovered but useless). The algorithm starts by labeling the root (v_1^0, v_2^0) gray and pushing it on top of the stack. In each iteration, the node $(v, v') = \text{head}(S)$ on top

of the stack is considered. If there is a successor (w, w') of (v, v') which is not black, then (w, w') will be labeled gray and pushed onto S , i.e., $\text{head}(S) = (w, w')$. If not, (v, v') will be labeled black and removed from the stack. The algorithm stops if it discovers the sink (s_1, s_2) . In this case, the nodes in the stack identify the only assignment in $(P_1 \wedge P_2)^{-1}(1)$.

It is a well known fact that this algorithm has time and space costs of $\mathcal{O}(|P_1 \wedge P_2|)$. While the stack always contains at most n nodes, the space consuming part of this approach results from the necessity to store all nodes labeled black. In the worst case, a large part of the nodes of unminimized $P_1 \wedge P_2$ have to be discovered before finding the path to (s_1, s_2) .

5.2. The BFS approach

This approach uses the following easy fact resulting from $|(P_1 \wedge P_2)^{-1}(1)| = 1$.

Lemma. *For all nodes (v, v') in $P_1 \wedge P_2$, the property $|\text{Sat}_{P_1 \wedge P_2}(v, v')| \leq 1$ is fulfilled, and if $|\text{Reach}_{P_1 \wedge P_2}(v, v')| > 1$, then $\text{Sat}_{P_1 \wedge P_2}(v, v') = \emptyset$.*

The underlying data structure for BFS is a queue Q used to store the nodes of $P_1 \wedge P_2$. The queue Q is equipped with the pointers $\text{tail}(Q)$, pointing to the back of Q , and the pointer $\text{head}(Q)$, pointing to the front of Q . The nodes (v, v') of the LOBDD $P_1 \wedge P_2$ have an additional property $(v, v').\text{path}$ which contains the assignment of the Boolean variables corresponding to the only path in $P_1 \wedge P_2$ from the root to (v, v') .

During the execution of the algorithm, some nodes of $P_1 \wedge P_2$ will be labeled *gray* (discovered) and some nodes will be labeled *black* (discovered but useless). The algorithm starts by labeling the root (v_1^0, v_2^0) gray and putting it into the queue Q , i.e., $\text{tail}(Q) = \text{head}(Q) = (v_1^0, v_2^0)$.

In each iteration, the node $(v, v') = \text{head}(Q)$ is removed from Q . If (v, v') is gray, each successor (w, w') of Q is processed in the following way: If (w, w') was not discovered before, then $(w, w').\text{path}$ is computed by adding the label of the edge $((v, v'), (w, w'))$ to $(v, v').\text{path}$; (w, w') will be labeled gray and put into the queue Q , i.e., $\text{tail}(Q) = (w, w')$. If (w, w') is gray (already discovered and contained in Q), it is labeled black. The algorithm stops when (s_1, s_2) is labeled gray. In this case, $(s_1, s_2).\text{path}$ yields the only satisfying assignment of $P_1 \wedge P_2$.

Note that if a node (v, v') is black, then $|\text{Reach}_{P_1 \wedge P_2}(v, v')| > 1$. This implies $\text{Sat}_{P_1 \wedge P_2}(u, u') = \emptyset$ for all nodes (u, u') reachable from (v, v') in $P_1 \wedge P_2$. Consequently, if a black node is dequeued from Q , its successors will not be enqueued into Q . Compared to the DFS approach, it is not necessary

to store nodes removed from Q . Hence, the space consumption equals the maximum size of Q . It may be upper bounded by $\text{width}(P_1) \cdot \text{width}(P_2)$.

6. Further experimental results and conclusion

In order to assess the space reduction achievable with our new algorithmic approaches presented in Section 5, we performed corresponding experiments based on size-reduced KSG prototypes. But unlike in Section 4, we could not use an existing BDD package such as CUDD, which is highly optimized for real-world usage and would not have easily allowed for the kind of algorithmic analysis (w.r.t. implementation-independent metrics) required in our context. Consequently, we had to write our own LOBDD package from scratch in order to derive experimental results for

- a 28-bit Geffe-generator with three maximum-period LFSRs defined by $l_{t+7} := l_t \oplus l_{t+1}$, $m_{t+10} := m_t \oplus m_{t+3}$, and $n_{t+11} := n_t \oplus n_{t+2}$, and the output function $z_t := l_t \cdot m_t \oplus m_t \cdot n_t \oplus n_t$,
- a 26-bit NFSR with feedback function $x_{t+26} := x_t \oplus x_{t+13} \oplus x_{t+5} \cdot x_{t+17}$ and output function $z_t := x_{t+9} \oplus x_{t+19}$.

As described in Section 5, the relevant memory consumption metrics for our new algorithms are the maximum size (i.e., number of elements) of the set of black nodes for the DFS approach and of the queue for the BFS approach. We compare this to the approach that the LOBDD $P_1 \wedge P_2$ is actually computed in order to find the only assignment in $(P_1 \wedge P_2)^{-1}(1)$. The relevant metric there is the size (i.e., number of nodes) of the LOBDD *before minimization*. The results in the following table are an average based on 100 randomly sampled (*Initial State, Keystream Prefix*)-pairs per KSG.

	DFS-based (black nodes)	BFS-based (queue nodes)	Synthesis-based (LOBDD nodes)
Geffe-KSG	4003455	667694	6530494
NFSR-KSG	7631335	557169	10977359

Clearly, both of our new algorithmic approaches are more space efficient than using standard synthesis to compute $(P_1 \wedge P_2)^{-1}(1)$. Especially the BFS-based variant seems extremely promising and could initiate a new phase of BDD-based cryptanalysis. As future work, we suggest to integrate our new approaches into standard BDD packages like CUDD.

Acknowledgement: We are grateful to anonymous reviewers, whose comments helped improve the presentation of this paper.

References

- [1] Babbage S.H., “Improved "exhaustive search" attacks on stream ciphers”, European Convention on Security and Detection, 1995, 161–166.
- [2] *Bluetooth SIG*, Bluetooth Core Specification 4.2, 2014, https://www.bluetooth.org/DocMan/handlers/DownloadDoc.ashx?doc_id=286439.
- [3] Briceno M., Goldberg I., Wagner D., “A pedagogical implementation of A5/1”, 1999, <http://www.scard.org/gsm/a51.html>.
- [4] Eibach T., Pilz E., Völkel G., “Attacking Bivium using SAT solvers”, SAT 2008, Lect. Notes Comput. Sci., **4996**, Springer, 2008, 63–76.
- [5] Golić J. Dj., “On the security of nonlinear filter generators”, FSE 1996, Lect. Notes Comput. Sci., **1039**, 1996, 173–188.
- [6] Krause M., “BDD-based cryptanalysis of keystream generators”, EUROCRYPT 2002, Lect. Notes Comput. Sci., **2332**, 2002, 222–237.
- [7] Krause M., Stegemann D., “Reducing the space complexity of BDD-based attacks on keystream generators”, FSE 2006, Lect. Notes Comput. Sci., **4047**, 2006, 163–178.
- [8] Meier W., Staffelbach O., “The self-shrinking generator”, EUROCRYPT '94, Lect. Notes Comput. Sci., **950**, 1994, 205–214.
- [9] Meinel C., Theobald T., Algorithms and data structures in VLSI design, 1998, xii+268 pp.
- [10] Somenzi F., “CUDD”, 2015, <http://vlsi.colorado.edu/~fabio/>, Website (accessed on February 05, 2018).
- [11] Stegemann D., “Extended BDD-based cryptanalysis of keystream generators”, SAC 2007, Lect. Notes Comput. Sci., **4876**, 2007, 17–35.
- [12] Wegener I., *Branching Programs and Binary Decision Diagrams: Theory and Applications*, SIAM e-books. Society for Industrial and Applied Mathematics, 2000.