To be published in: *Proceeding of Third International Conference on Mathematical Sciences (ICM 2008), Al Ain, UAE, March 2008.*

ZDD-Based Cryptanalysis of E_0 Keystream Generator

M. Ghasemzadeh - Ch. Meinel
HPI at the University of Potsdam
Helmert Str. 2-3, Potsdam, Germany
{meinel, ghasemzadeh}@hpi.uni-potsdam.de

M. Shirmohammadi - M.H. Shahzamanian
Computer - Math. Dept. at Yazd University
Pejoohesh Street, Safa-ieh, Yazd, Iran
shirmohammadi@stu.yazduni.ac.ir

Abstract

BDD is an efficient data structure that in last few years has been used effectively in computer science and engineering. BDD-based attack in key stream cryptanalysis is supposed to be one of the best forms of attack in its kind. In this paper, we propose a new key stream attack which is based on ZDD(a variant of BDD). We show how our ZDD-based key stream attack can be used against the E_0 type of the Bluetooth security mechanism. We implemented our algorithm using CUDD package. The experimental results witness the superiority of our method. We have also derived a mathematical proof for the algorithm, which shows that its behavior even under the worst circumstances is better than BDD attack.

1 Introduction

In cryptography, pseudo random sequences are frequently used. A pseudo random sequence generator requires to be uniformly distributed, independent, and non-correlated [8]. In implementation of key stream generators, the LFSR (Linear Feedback Shift Register) is being used because all above conditions are met and the corresponding algebraic analysis is also quite simple.

The LFSR-based key stream generators consist of two components: a linear bitstream generator L and a nonlinear compression function C, i.e. $K = (L, C)$. First they generate the key stream $Y = C(L(k))$, for the cipher key k, then Y and the plain text P are bitwise XORed to produce the cipher text E. In cryptanalysis of these generators, the encryption system is supposed to be known and we are interested in finding k.

BDD and its variants are data structures that are used effectively in computer science and engineering. These data structures give compact and canonical representations for Boolean functions. Recently, a new attack against LFSR-based key stream generators is introduced by Krause [4] which is based on FBDD. Later Shaked and Wool [9] introduced their OBDD-based attack to E_0 key stream generator. In this paper, we introduce a new attack to key stream generators which uses ZDD. Experimental results show that it makes a remarkable reduction in time and space complexity over OBDD and FBDD based attacks. We have also derived a proof which confirms the experimental results.

This paper is organized as follows. Section 2 provides the basic definitions and the main concepts: E_0 encryption system and a brief introduction to BDD and ZDD. In section 3 the proposed attack is introduced. First the FBDD attack is discussed, then the attack to E_0 with OBDD is reviewed. Finally our ZDD-based attack is introduced. Section 4 is dedicated to the theoretical complexity analysis of our method. Section 5 provides concludes.

2 Preliminaries

2.1 E_0 Key Stream Generator

E_0 is a LFSR-based key stream generator which is used in Bluetooth security mechanism. LFSR-based key stream generators consist of two components, a linear bitstream generator and a nonlinear compression function. After initialization, the linear bitstream generator L, generates the bitstream Z. It employs four Linear Feedback Shift Registers(LFSR), whose output is the input to the compression function C. The output of the compression function would be the key stream $Y = C(L(k))$. The lengths of the four LFSR are $|L_0| = 25$, $|L_1| = 31$, $|L_2| = 33$ and $|L_3| = 39$, and their feedback polynomials are:

$$p_0(x) = x^{25} + x^{20} + x^{12} + x^8 + 1$$

$$p_1(x) = x^{31} + x^{24} + x^{16} + x^{12} + 1$$

$$p_2(x) = x^{33} + x^{28} + x^{24} + x^4 + 1$$

$$p_3(x) = x^{39} + x^{36} + x^{28} + x^4 + 1$$

At the beginning, the linear generator needs to be loaded with an initial value for the four LFSRs(128 bits in total). Summation of the four output bits of the LFSRs make

the input of the compression function. The compression function is usually organized with a finite state machine $C_{E_0} : (Q, \Sigma, \Gamma, I, F, \delta)$ [1, 9, 5]. Figure 1 displays the transition function of this finite state machine.

	a=0		a=1		a=2		a=3		a=4	
q_n	b	q_{n+1}	b	q_{n+1}	b	q_{n+1}	b	q_{n+1}	b	q_{n+1}
0	0	0	1	0	0	4	1	4	0	8
1	0	12	1	12	0	8	1	8	0	4
2	0	4	1	4	0	0	1	0	0	12
3	0	8	1	8	0	12	1	12	0	0
4	1	5	0	1	1	1	0	13	1	13
5	1	9	0	13	1	13	0	1	1	1
6	1	1	0	5	1	5	0	9	1	9
7	1	13	0	9	1	9	0	5	1	5
8	0	14	1	14	0	2	1	2	0	6
9	0	2	1	2	0	14	1	14	0	10
10	0	10	1	10	0	6	1	6	0	2
11	0	6	1	6	0	10	1	10	0	14
12	1	11	0	7	1	7	0	3	1	3
13	1	7	0	11	1	11	0	15	1	15
14	1	15	0	3	1	3	0	7	1	7
15	1	3	0	15	1	15	0	11	1	11

Figure 1. The FSM transition function

2.2 OBDD And ZDD

A Binary Decision Tree is a directed acyclic graph over a set of Boolean variables $\forall m \in N, X_m = \{x_1, ..., x_m\}$; it can represent a Boolean function over X_m. A BDT comprises two kinds of vertices, nonterminal vertices, and terminal vertices. Each nonterminal vertex N is assigned a label $Var(N) \in X_m$, and has two children $Low(N)$ corresponding to the answer $x_i = 0$ and $High(N)$ corresponding to the answer $x_i = 1$. There is exactly one node with indegree 0, the root of BDT. Each terminal vertex T is labeled by either 0 or 1. Depending on its label, a terminal vertex is called a $0-T$ or $1-T$. A Boolean function composed of m Boolean variables, has 2^m assignments. Each assignment $b \in \{0,1\}^m$ defines a path from root to a terminal vertex.The label of this terminal vertex is shown as $F(b)$.

A Binary Decision Diagram is an efficient graph to represent a Boolean function. A BDD is derived from a BDT by applying the following reduction rules: merge all $0-T$ vertices and $1-T$ vertices, combine similar subgraphs, remove all "don't care" nonterminal vertices. Two subgraphs G_1 and G_2 are similar, if in their roots g_1 and g_2: $Var(g_1) = Var(g_2)$, $Low(g_1) = Low(g_2)$ and $High(g_1) = High(g_2)$. A "don't care" nonterminal vertex is one whose two children are the same, i.e., $Low(g_1) = High(g_1)$.

A Free Binary Decision Diagram is a BDD if along each path, from root to one of terminal vertices, each variable x_i occurs at most once.

An Oracle graph over X_m is a FBDD with only one (unlabeled) terminal for which on each path, from the root to the terminal, all m variables occur. Oracle graph is not designed for computing Boolean functions; its aims is to define a set of valid ordering for $x_i \in X_m$.

G-FBDD is a FBDD if there exists an Oracle graph G that every ordering of the variables requested in FBDD corresponds to a path of G.

A Reduced Ordered Binary Decision Diagram is a G-FBDD if its Oracle graph G has only one path. On the other hand, G is degenerated into a linear list that only shows one order for occurrence of variables.

Zero-suppressed Binary Decision Diagram is a variant of BDD that can represent a Boolean function. ZDD like BDD is derived from a BDT by applying similar reduction rules, i.e., merge all $0-T$ vertices and 1-T vertices and combine similar subgraphs, but with different reduction rule. The last rule for deriving ZDD, remove the nonterminal vertex N, if $High(N)$ is connected to $0-T$ [2, 6, 4, 3].

3 ZDD Based Cryptanalysis Of E_0

3.1 FBDD Based Cryptanalysis Of Key Stream Generator

In the algorithm proposed by Krause [4], the method first reduces the problem for the cryptanalysis of LFSR-based key stream generators. It assumes that except for key k, all other parameters are known. Moreover, it is assumed that the attacker is able to obtain the first bits of the key stream Y. The attacker's goal then is computing $k = \{0,1\}^n$ that produces the observed key stream. Since in an LFSR, the first output bits are the initialization value of LFSR, then $Z = L(k)$ would contain k in the first bits. Then the problem reduces to finding a bitstream Z with the following conditions:

1. Z can be produced by the linear bitstream generator L.
2. $C(Z)$ is prefix of the observed key stream Y.

For each $m \geq 1$, and the bitstream $z \in \{0,1\}^m$ is defined as:

- $G_m^C - FBDD$ denote the oracle graph that defines for each Z the order in which the bits of Z are read by the compression function C.
- R_m denote the minimal $G_m^C - FBDD$ that decides whether Z can be produced by L.
- Q_m denote the minimal $G_m^C - FBDD$ that decides whether $C(Z)$ is prefix of Y.

- P_m denote the minimal $G_m^C - FBDD$ that decides whether Z can be produced by L and $C(Z)$ is prefix of Y.

In [4], the key is considered to be n bits length then it computes m^*, where m^* denotes the length of the consecutive bitstream required for key discovery. Therefore, the following algorithm can compute k:

1. $P \leftarrow Q_n$.
2. for $m \leftarrow n+1$ to m^* do:
   $P \leftarrow (P \wedge Q_m \wedge R_m)$
3. return Z^* that $P(Z^*) = 1$.

On the other hand, iteration continues until P_{m^*} has only one assignment $z^* \in \{0,1\}^m$ that $P(Z^*) = 1$.

3.2 Reduction of FBDD-based Cryptanalysis With OBDD-Based Cryptanalysis

The algorithm described by Krause is generic and needs to be adapted for use on E_0, Shaked and Wool [9] made reductions and changes to the algorithm about E_0, by using OBDD instead of FBDD. Krause in [5] generalized OBDD attack to oblivious key stream generator.

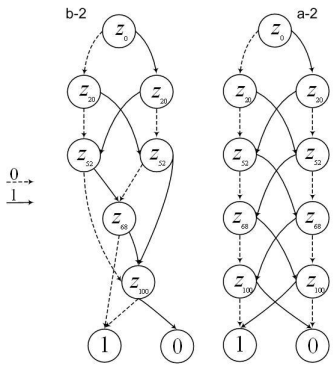

Figure 2. a) OBDD representing and b)ZDD representing which check bit z_{100}

OBDD attack marked the output bits of $L(k)$, for example, $Z = (..., z_{4j}, z_{4j+1}, z_{4j+2}, z_{4j+3}, ...)$, where $z_{4j+i} \in L_i(k_i)$. This bit ordering leads to following equations for linear key stream generator L:

$$\forall i = 4j : z_i = z_{i-32} \oplus z_{i-48} \oplus z_{i-80} \oplus z_{i-100} \quad (1)$$

$$\forall i = 4j+1 : z_i = z_{i-48} \oplus z_{i-64} \oplus z_{i-96} \oplus z_{i-124}$$

$$\forall i = 4j+2 : z_i = z_{i-16} \oplus z_{i-96} \oplus z_{i-112} \oplus z_{i-132}$$

$$\forall i = 4j+3 : z_i = z_{i-16} \oplus z_{i-112} \oplus z_{i-144} \oplus z_{i-156}$$

Then based on these equations, R_m graph is produced by building OBDD for every z_i. Figure 2 shows an OBDD representing which checks bit z_{100}. In building OBDDs which check bits of each L_i, algorithm calls the first $|L_i|$ bits in its bit stream. The goal of the algorithm is to compute these first bits of all L_i. According to the above equations, an algorithm must build OBDD for $z_j : |L| \leq j \leq 4|L|$. Each synthetic OBDD contains 5 variables and 11 vertices, therefore, it requires 4224 vertices. To compute the Q_m graph, they used a BDD structure called basic chain, to represent sum of 4 bit, refer to Figure 3. For each state and each of the 5 possible sums, Table 1 shows what the output bit should be. If it matches the bit given in known key stream Y, it can advance to the next chain, and test the next four bits; otherwise, this path will lead to $0 - T$. The Q_m graph is built from blocks, each consisting of 16 basic chain corresponding with 16 states of FSM. According to 2, each chain contains 4 variables and 10 vertices, therefore, a sequence of 128 blocks requires 20,480 vertices. Indeed, Shaked and Wool [9] demonstrated that the reduction rule of BDD reduces this size to 14,500 vertices rather than 20,480.

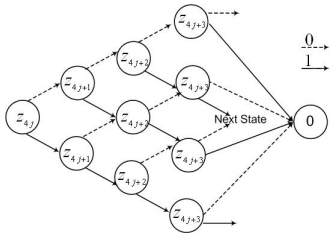

Figure 3. Basic chain representing sum of 4 bits

3.3 ZDD-Based Cryptanalysis Of E_0

Initially we stated the main motivation for using ZDD in relation to this probelm. Considering the structure and properties of BDD and ZDD, while BDDs are better suited for representing Boolean functions, ZDDs are better for representing sets. In fact, using ZDD is more efficient than using OBDD in problems that need to manipulate a set of combination [7]. A combination of "n" items can be represented by an n-bit binary vector, $(x_1, ...x_n)$, where each bit $x_i \in \{0,1\}$ expresses whether or not the item is included

in the combination, so a set of combination can be shown with a Boolean function by n-input variables. This Boolean function determines which combination are possible in a solution set of problem. Such Boolean functions are called characteristic functions. The set operations such as union, intersection, and difference can be performed on these functions.

OBDDs are more efficient in compression representing characteristic function of combinatorial set, than other data structures. Despite the efficiency of OBDDs, there is one severe defect in combinatorial sets, because of inappropriate reduction rules. Indeed, ZDD is a variant of BDD, where one of the reduction rules have been changed in a way which leads to more efficient representations for combinatorial sets. In a ZDD, each path from root to the 1-T corresponds to one of the combinations [6].

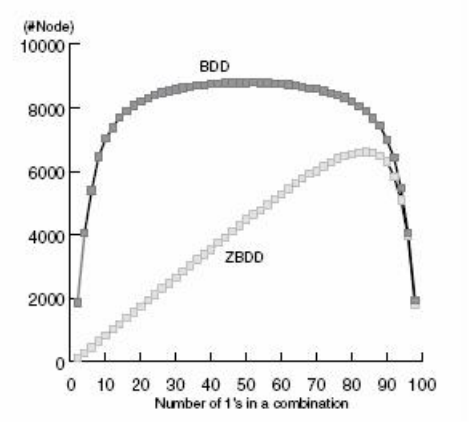

Figure 4. ZDD vs. BDD in combinatorial sets (Adapted from[6])

Minato in [6], for evaluating the ZDD and OBDD using representative combinatorial set, conducted a statistical experiment and produced a shown diagram in Figure 4. The diagram shows that ZDDs are much more compact than OBDD, especially in representing sets of sparse combinations.

The goal of key stream Cryptanalysis is to scan possible keys and find keys of known key streams. FBDD attack can be reduced for oblivious key stream generators by using OBDD data structure. These generators fulfill the same ordering, in building R_m and Q_m graphs, and in conclusion building P_m graphs. The compression nonlinear function of these generators can be shown with a finite state machine.

ZDD attack operates on this kind of key stream generators. We discussed ZDD attack against E_0 key stream generator, and stated that the method can be generalized to all oblivious key stream generators. ZDD attack is based on the general theoretical FBDD attack, but with different data structure and implementation method, which optimizes that attack. In the ZDD attack against E_0 generator, we implemented an R_m graph by similarity technique to OBDD attack, the only difference is using ZDD instead of OBDD. Figure2 shows a ZDD representing checks bit z_{100}; according to this figure, each synthetic ZDD contains 5 variables and 9 vertices, therefore, it requires 3456 vertices. We computed the Q_m graph by the following technique. Since finite state machine of E_0 generator has 16 states, we used 4 variables $0 \leq i \leq 3$, $q_i^n$to mark them. So the following function for Q_m can be computed:

$$F(q_3^{m+1}, q_2^{m+1}, q_1^{m+1}, q_0^{m+1}, z_{4m+3}, z_{4m+2}, , z_{4m+1}, .., z_0)$$

We can see that the Q_m function consists of $4m + 4$ variables. It stands for all the possible paths in the finite state machine after reading $m + 1$ input symbols. We implemented the Q_m function using the algorithm bleow:

1. If C_{E_0} machine includes transition rule $(q_m, a) \rightarrow q_{m+1}$ AND corespondent output rule $(q_m, a) \rightarrow b$ AND $b = b_m$(b_m is m th bit in known key stream Y):

   1.1 Compute q_m and q_{m+1} based on q_m^i:

   $$q_m = (q_m^3)^* \wedge (q_m^2)^* \wedge (q_m^1)^* \wedge (q_m^0)^*$$

   where $(q_m^i)^*$ is q_m^i or $(q_m^i)^*$ is $(q_m^i)'$ according to labels of the states of the machine. For example in step m, the 5th state is : $(q_m^3)' \wedge (q_m^2) \wedge (q_m^1)' \wedge (q_m^0)$.

   1.2 For all $\sum z_{4m+i} = a$, compute:

   $$X_j = (q_{m+1} \wedge z_{4m+3} \wedge z_{4m+2} \wedge z_{4m+1} \wedge z_{4m} \wedge q_m)$$

2. Compute Q'_m function based on:

$$Q'_0 = (X_0 \vee ... \vee X_j)$$

$$Q'_m = ((X_0 \wedge Q_{m-1}) \vee ... \vee (X_j \wedge Q_{m-1})$$

3. Compute Q_m by removing $(q_m^i)^*$ from Q'_m.

We need to mention that finally we are interested in computing Q_{128}. The constructed Q_m truly decides whether $C(Z)$ is prefix of Y or not. By scanning all the paths from root to $1-T$, we compute all Zs which produce same prefix to Y . In our proposed ZDD-based attack, the final states of machine and bit stream that read for reaching the states are enough to be maintained with the Q_m graph.

Due to pseudo random sequences requirements, the constructed Q_m would be a sparse graph. (A pseudo random sequence must be uniformly distributed, i.e., the probability of 0 occurrences must be equal to the probability of 1 occurrence.) In implementing our proposed attack by ZDD, we mapped the problem to a combinatorial set problem. In fact, in each iteration of computing Q_m, we checked all possible combinations of input bits and final states.

Most operation on sets are readily defined and implemented for ZDD, such as union, intersect, difference and another functions:

- Z.onset(N) selects the subset of the combinations including N, and then delete N from each combination.
- Z.count the number of combinations in Z.

and so on. We ran our algorithm in C along with the CUDD package[10]; Our algorithm can be displayed with the following pseudo codes:

For $\forall$ element $\in \delta$
  If $((q_m, a) \rightarrow q_{m+1} \wedge (q_m, a) \rightarrow b \wedge (b = b_m))$
  {
    $q_m = ZDDIntersect(q_m^3)^*, (q_m^2)^*, (q_m^1)^*, (q_m^0)^*)$
    $q_{m+1} = ZDDIntersect((q_{m+1}^3)^*, (q_{m+1}^2)^*,$
                $(q_{m+1}^1)^*, (q_{m+1}^0)^*)$
    For $\forall Z_{4j+i}, 0 \leq i \leq 3$
    {
      if $\sum Z_{4j+i} = a$
       $X_j = ZDDIntersect(q_{m+1}, z_{4m+3},$
                $z_{4m+2}, z_{4m+1}, z_{4m}, q_m)$
    }
  }
  $Q'_m \leftarrow ZDDUnion(\forall j, ZDDIntersct(X_j,$
      $Q_{m-1}.Oneset(q_m), if((q_m)_{X_j} == (q_m)_{Q_{m-1}})$
For every q_m $Q_m \leftarrow Q.Oneset(q_m)$

4 Theoretical Complexity Analysis

The time complexity of the algorithm is determined by the space complexity of the constructed ZDD during the entire process of construction. First, review the complexity of functions which are used in the algorithm:

- Time complexity of producing ZDD which is representing function $F(x_0, ..., x_n)$ is $O(|G_F|)$, where $|G_F|$ denotes the number of vertexes in constructed graph.
- Time complexity of each set operation such as union and intersect of two graph F,G is $O(|G_F|.|G_G|)$

In the algorithm, during the $|L_0|$ steps,it introduces 4 new variables, and one constraint $\sum z_{4j+i} = a$, then the number of assignments is multiplied by 2^3. After $|L_1|-|L_0|$ steps it has two constraints, $z_{4j} \in L_0$ is determined ,then the number of assignments is multiplied by 2^2. After $|L_2|-|L_1|$ steps it has three constraints, $z_{4j} \in L_0$ and $z_{4j+1} \in L_1$ are determined ,then the number of assignments is multiplied by 2^1. After $|L_3|-|L_2|$ step it has four constraints and there are no more choices, then the number of assignments will be constant. In the other steps, the number of assignments start to decrease to half. On the other hand, due to ZDD properties, the average number of vertices in each path are:

$$\frac{\sum C(4,i).i.m}{2^4} = 2m$$

Therefor, considering the above arguments, we can compute the higher bound as$(m : 0 \rightarrow 128)$:

$m \leq |L_0| : |P_m| = 2m \times 2^{3m}$
$|L_0| \leq m \leq |L_1| : |P_m| = 2m \times 2^{3|L_0|} \times 2^{2(m-|L_0|)}$
$|L_1| \leq m \leq |L_2| : |P_m| = 2m \times 2^{3|L_0|} \times 2^{2(|L_1|-|L_0|)} \times 2^{m-|L_1|}$
$|L_2| \leq m \leq |L_3| : |P_m| = 2m \times 2^{3|L_0|} \times 2^{2(|L_1|-|L_0|)} \times 2^{2(|L_2|-|L_1|)}$
$|L_3| \leq m : |P_m| = 2m \times 2^{3|L_0|} \times 2^{2(|L_1|-|L_0|)} \times 2^{|L_2|-|L_1|} \times 2^{|L_3|-m}$

On the other hand, P_m is obtained by intersection of Q_m and R_m, then we can compute the other higher bound:

$m \leq |L_0| : Time(P_m) = |Q_m| \times |R_{4m}| = |Q_m|$
$|L_0| \leq m \leq |L_1| : Time(P_m) = |Q_m| \times (m - |L_0|) \times 2^3$
$|L_1| \leq m \leq |L_2| : Time(P_m) = |Q_m| \times (m - |L_0| - |L_1|) \times 2^3$
$|L_2| \leq m \leq |L_3| : Time(P_m) = |Q_m| \times (m - |L_0| - |L_1| - |L_2|) \times 2^3$
$|L_3| \leq m : Time(P_m) = |Q_m| \times (m - |L_0| - |L_1| - |L_2| - |L_3|) \times 2^3 = |Q_m| \times (m - 128) \times 2^3$

In practice $|Q_m|$ is approximately 2^{14}. The overall higher bound of complexity can be obtained from intersection of the two calculated bounds, which will give a space complexity of 2_{23}, and with a time complexity of 2^{82}. we need to mention that this is a non-refined approximation bound, accurate analysis would give even better complexity. The theoretical bound of ZDD attack showed it made $O(2^8)$ reduction to OBDD attack.

5 Conclusion

This research shows how ZDD can be used to construct an attacker to the key stream generators. A formal proof shows that for E_0 generator attack it needs 2^8 nodes less than the former BDD based method.

References

[1] S.R. Fluhrer and S. Lucks. Analysis of the E0 encryption system. In *Proceedings of 8th Workshop on Selected Areas in Cryptography, LNCS 2259*. Springer-Verlag, 2001.

[2] M. Ghasemzadeh. An efficient data structure in Computer Science. In *Proceedings of 12th International CSI Computer Conference (CSICC'2007), Tehran, Iran*, 2007.

[3] M. Krause and D. Stegemann. Reducing the space complexity of BDD-based attacks on keystream generators. In *13th annual Fast Software Encryption Workshop.*, 2006.

[4] Matthias Krause. BDD-Based Cryptanalysis of Keystream Generators. In *EUROCRYPT*, pages 222–237, 2002.

[5] Matthias Krause. OBDD-Based Cryptanalysis of Oblivious Keystream Generators. *Theor. Comp. Sys.*, 40(1):101–121, 2007.

[6] S. Minato. Zero-Suppressed BDDs and Their Applications. In *Proceedings of International Journal on Software Tools for Technology Transfer*, pages 156–170. Springer.

[7] A. Mishchenko. An introduction to zero-suppressed binary decision diagrams. Technical report, Portland State University, 2001.

[8] M.J.B. Robshaw. Stream ciphers. Technical report, RSA Laboratories, July 1995.

[9] Y. Shaked and A. Wool. Cryptanalysis of the Bluetooth E0 cipher using OBDD's. In *Proceedings of 9th Information Security Conference, LNCS 4176*, pages 187–202.

[10] Fabio Somenzi. CUDD: Colorado University Decision Diagram Package. ftp://vlsi.colorado.edu/pub/.