Cryptanalysis of the Bluetooth E_0 Cipher using OBDD's

Yaniv Shaked and Avishai Wool

School of Electrical Engineering Systems,
Tel Aviv University, Ramat Aviv 69978, ISRAEL
shakedy@eng.tau.ac.il, yash@acm.org

March 18, 2006

Abstract. In this paper we analyze the E_0 cipher, which is the cipher used in the Bluetooth specifications. We adapted and optimized the Binary Decision Diagram attack of Krause, for the specific details of E_0. Our method requires 128 known bits of the keystream in order to recover the initial value of the four LFSR's in the E_0 system. We describe several variants which we built to lower the complexity of the attack. We evaluated our attack against the real (non-reduced) E_0 cipher. Our best attack can recover the initial value of the four LFSR's, for the first time, with a realistic space complexity of 2^{23} (84MB RAM), and with a time complexity of 2^{87}. This attack can be massively parallelized to lower the overall time complexity. Beyond the specifics of E_0, our work describes practical experience with BDD-based cryptanalysis, which so far has mostly been a theoretical concept.

Keywords: Stream cipher, Cryptanalysis, Bluetooth, BDD

1 Introduction

1.1 Background

Bluetooth, a technology used for short range fast communications, has quickly spread worldwide. Bluetooth technology is used in a large set of wired and wireless devices: mobile phones, PDA's, desktop and mobile PC's, printers, digital cameras, and dozens of other devices.

Bluetooth employs a stream cipher as the data encryption mechanism. This stream cipher, E_0, is based on 4 LFSR's (Linear Feedback Shift Registers) of different lengths, along with a non-linear combiner logic (finite state machine). The keystream is xor-ed with the plaintext, to create the ciphertext, and decryption is performed in exactly the same way using the same stream used for encryption.

1.2 Related work

A number of crypt-analytical results regarding E_0 ([JW01], [FL01], [LW05], [Kra02], [Saa00], [HN99], [EJ00], [GBM02], [LV04], [LMV05], [KS06]) have appeared over the last five years. These attacks can be organized into two classes: *Short Keystream attacks* - attacks that need at most 3,100 known keystream bits; and *Long Keystream attacks* - attacks that require more (usually much more) known keystream. Long keystream attacks are generally not applicable within the Bluetooth settings since a maximal Bluetooth continuous frame is shorter than 3,100 bits (5 slots, $625\mu sec$ each, $1Mbit$ burst rate; see page 59 of part B of Vol 2 of [Blu03]) after which Bluetooth rekeys the E_0 registers. Therefore, all long keystream attacks, except for the attack suggested in [LMV05], are applicable only if E_0 is used outside the Bluetooth system.

Short Keystream attacks

1. D. Bleichenbacher has shown in [JW01] that an attacker can guess the initial state of the three smaller LFSR's and the non-linear combiner; Then the attacker can compute the contents of the longest LFSR, (whose length is 39 bits) by "reverse engineering" it from the outputs of the other LFSR's and the combiner state. This attack requires approximately 132 bits of known keystream with a computational complexity of $O(2^{100})$.
2. S. Fluhrer and S. Lucks have shown in [FL01] an optimized backtracking method of recovering the secret key with a computational complexity of $O(2^{84})$ if a 132 bits are available.
3. O. Levy and A. Wool have shown in [LW05] a uniform framework for crypt-analysis, whose its best setting can recover the initial state of the LFSR's after solving $O(2^{86})$ systems of boolean linear equations.
4. The best reported short keystream attack against E_0 was suggested by Krause [Kra02] as part of a general framework. The general attack framework uses Free Binary Decision Diagram (FBDD's), a data structure that is used to represent a boolean function, for attacking LFSR-based key stream generators in general, and E_0 in particular. In his paper, Krause claims that for E_0 his attack requires $O(2^{77})$ space, and a time complexity of $O(2^{81})$, based on some quick estimations. Krause's attack is the starting point of this paper: we adapted and optimized his attack for the specifics of E_0, and evaluated the attack's viability.

The work closest to ours was very recent recently suggested, independently, by Krause and Stegemann [KS06]. They too attempt to make BDD-based crypt-analysis practical, via a divide-and-conquer strategy. They evaluated their attacks against reduced versions of E_0, with random feedback polynomials, and extrapolated a space complexity of $O(2^{42})$ against the real E_0, with roughly the same time complexity estimate of [Kra02]. In contrast, we evaluated our attacks against the *real* E_0 cipher, and show a greatly improved and practical space complexity of 2^{23} BDD nodes (without the $O()$ notation).

Currently, the best *long keystream attack* against E_0 is by Y. Lu, W. Meier and S. Vaudenay in [LMV05]. The attack is a conditional correlation attack on the two-level Bluetooth E_0, that fully recovers the original encryption key using the first 24 bits of $2^{23.8}$ frames with $O(2^{38})$ computations. Since it is against the two-level cipher, the attack is not limited to a single continuous Bluetooth frame—so the requirement of $2^{23.8}$ frames is attainable in principle.

Another BDD-based cryptanalysis attack against a different cryptosystem was presented by J.F Michon, P. Valarcher and J.B Yunés in [MVY03]. They used BDD's to implement a ciphertext only attack against HFE (Hidden Field Equations - a public key cryptosystem). They report that the attack was not efficient.

1.3 Contributions

In this paper we describe an implementation of an attack against E_0 that is based on the use of *Binary Decision Diagrams (BDD's)*. Our attack is based upon the theoretical BDD-based attack framework of M. Krause ([Kra02]). Krause's work covered several keystream generators including the E_0; Consequently, we needed to supply missing details to adjust the attack for the E_0 system. Furthermore, we discovered that Krause's general attack can be greatly simplified and optimized when it is used against E_0: We discovered that it is possible to use OBDD's rather than FBDD's throughout the algorithm; We re-engineered the algorithm to adjust to the different LFSR lengths; We developed an efficient composable BDD for the *compressor*; and after discovering that standard BDD algorithms and libraries are very inefficient for this algorithm we wrote our own BDD code that is optimized for attacking E_0.

In addition, we built several hybrid variants of the basic BDD-based algorithm. These variants include: (i) partially guessing LFSR's initial data, (ii) using an intentionally "defective" compressor, and (iii) enumerating the satisfying assignments and testing them. We evaluated our attacks against the full, non-reduced, E_0 cipher. Our best heuristics can recover the initial state of the LFSR's, for the first time, with a practical space complexity of 2^{23} (84MB RAM). Our time complexity is 2^{87}: slightly higher complexity than reported by [Kra02], [KS06]—however, the attack is massively parallelizable. In addition to the specifics of Bluetooth, our work describes practical experience with BDD-based cryptanalysis, which so far has mostly been a theoretical concept.

Organization: In Section 2 we give an overview of the E_0 cipher, a brief overview of Binary Decision Diagrams and a description of Krause's attack. Section 3 describes adapting the attack to E_0 and analyzes the theoretical complexity of the attack. Section 4 describes the implementation of the attack, the heuristics used to lower attack complexity, and the performance we achieved. Section 5 concludes our work. Appendix A includes a more detailed description of the E_0 system. Appendix B contains a detailed explanation of the bounds used in the theoretical complexity analysis of Section 3.4.

2 Preliminaries

2.1 Overview of E_0 System

A full specification of Bluetooth security mechanisms can be found in part H of Vol 2 of [Blu03]. The security layer of Bluetooth, which is a part of the link layer, includes key management and key generation mechanisms, a challenge-response authentication scheme, and a data encryption engine. The data encryption engine used within Bluetooth is the E_0 keystream generator.

E_0 is initialized using a 128 bit session key (denoted K_c'), the Bluetooth address of the master device and a clock, which is different for every packet. Details regarding the generation of K_c' appear in appendix A. E_0 generates a binary keystream, K_{cipher}, which is xor-ed with the plaintext. The cipher is symmetric; decryption is performed in exactly the same way using the same key as used for encryption.

The E_0 system employs four *linear shift feedback registers* (LFSR's), of lengths 25, 31, 33, and 39 (total length of 128 bits), a *Summation Combiner Logic* and a non-linear *Blend machine*. We can represent the *summation combiner logic* and the *blend machine* together as a 4 bit finite state-machine. At each clock tick the LFSR's are clocked once, and the output of the four LFSR's is xor-ed with the output bit of the finite state machine, to create the next output bit of the encryption stream K_{cipher}. The sum of the four output bits of the LFSR's is input into the finite state machine to update the state of the machine. In the remainder of this paper, the finite state machine will be denoted as the ***Compressor*** unit. The finite state machine transition function (following [LV04], [LW05]) can be found in Table 3 in Appendix A.

2.2 Binary Decision Diagrams

A binary decision diagram (BDD) is a data structure that is used to represent a Boolean function. Let $\boldsymbol{X}_n$ denote the set of boolean variables $(x_0, ..., x_{n-1})$ of some boolean function. A **BDD** P over X_n is a rooted, directed, acyclic graph where each non-terminating node is labeled by a query *(x_i?)* and has outdegree two, one edge labeled 0 and one edge labeled 1, connecting to child nodes. There are two terminating nodes: one 0-sink and one 1-sink. The root node is considered the source node. Each assignment $w(x_0 = w_0, x_1 = w_1, ..., x_{n-1} = w_{n-1})$ where $w_i \in \{0, 1\}$ defines a unique path in P, which starts at the source node, answers w_i on queries *(x_i?)* and always leads to a unique sink. The ending sink is the result of the boolean function under the assignment w. Two BDD's are considered equivalent if they compute the same boolean function.

A BDD is a **Free** Binary Decision Diagram (**FBDD**) if along each path in the BDD each variable appears at most once.

A BDD is an **Ordered** Binary Decision Diagram (**OBDD**) if on **all** paths in the BDD the variables respect a given ordering $x_0 < x_1 < x_2 < .... < x_{n-1}$ (While FBDD's allow different orderings along each path).

2.3 BDD-Based Cryptanalysis of E_0

Problem model: The general attack framework of Krause ([Kra02]) works as follows. Given some known keystream bits, we would like to calculate the initial value of the LFSR's. Let $L(x)$ denote the internal linear bitstream in the E_0 keystream generator. $L(x)$ is actually comprised of the output sequence of the *four parallel LFSR's* in E_0. E.g., for an E_0 keystream of 128 bits, $L(x)$ comprises of 512 bits. Let $C(z)$ denote the non-linear component in E_0. $C(z)$ is actually the *Compressor* unit, including the output xor operation that is used to derive the keystream. According to these declarations, K_{cipher} equals $C(L(x))$, where x is the secret initial value of the LFSR's.

Krause's observation is that finding a secret key x fulfilling $K_{cipher} = C(L(x))$ for a given keystream K_{cipher}, is equivalent to the problem of finding the minimal FBDD P for the decision whether x fulfills $K_{cipher} = C(L(x))$. This idea is the basis for the BDD attack against the E_0 system.

The algorithm: Let *L(x)*, *C(z)* and K_{cipher} be as before. Let n be the key length (=128).

1. For all $m \geq 1$ let Q_m denote a minimal FBDD which decides for $z \in \{0,1\}^m$ whether *C(z)* is a prefix of K_{cipher}. In other words, Q_m is a FBDD which is built based on the value of the known keystream bits (K_{cipher}). This FBDD receives prefixes of the internal bitstreams which are generated by each LFSR as input. If this internal bitstream generates a prefix of the known keystream bits (K_{cipher}) - the FBDD accepts it. Otherwise, the FBDD rejects the input.
2. For all $m \geq n$ let S_m denote a minimal FBDD which decides for $z = (z_0, z_1, ..., z_m)$ whether $z_m = L(z_0, z_1, ..., z_{n-1})$. In other words, S_m is a FBDD which is build based on the feedback polynomials of the LFSR's. This FBDD receives the initial value of the LFSR's as input. If this initial value generates the correct value of z_m (the m-th internal stream bit) - the FBDD accepts it. Otherwise, the FBDD rejects the input.
3. Construct a third set of FBDD's, denoted P_m, which is the minimal FBDD which decides whether $z \in \{0,1\}^m$ is a linear bitstream generated via L **and** if *C(z)* is a prefix of K_{cipher}. Note that P_m is actually the result of the intersection between Q_m and S_m: $P_m = SYNTH(Q_m, S_m)$ — where $SYNTH$ denotes the BDD synthesis operation (cf. [Weg00]).

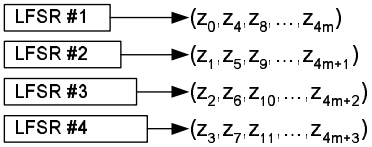
The strategy of Krause's algorithm is as follows: It incrementally computes P_m for increasing values of m until only one assignment will be accepted by P_m. This assignment is the initial value of the LFSR's generating K_{cipher}.

3 Adapting the attack to E_0

3.1 Reduction of the Algorithm

The algorithm described by Krause is generic, and needs to be adapted for use on E_0. We made the following reductions and changes before implementing the algorithm:

Fig. 1. Indexing method used in implementation



1. A key observation is that E_0 is regularly clocked. Every clock tick, one bit from each LFSR is input to the compressor, and each LFSR is stepped once. This regularity gives us two important advantages: First, E_0 induces a natural order on the internal bit stream Z: In our implementation, the variable ordering we used is: $\pi = (z_0, z_1, z_2, z_3, z_4, ..., z_j, ..., z_{511})$: for $j = 4*m+L_i-1$ we have that m is the clock tick index ($0 \leq m \leq 127$), and L_i is the index of the LFSR ($1 \leq L_i \leq 4$). Figure 1 describes the indexing method we used in implementation of the algorithm. Second, we can switch from using FBDD's to using OBDD's. This has critical implementation benefits, since the data structures for supporting OBDD's are much simpler and more efficient than those of FBDD's.
2. We needed to adjust for the fact that the four LFSR's in E_0 have different lengths. This changes the implementation details and the complexity analysis.
3. As Section 2.3 implies, we had to implement a synthesis operation between two BDD's. Our implementation was based on the synthesis algorithm suggested by Wegener (See Section 3.3 of [Weg00]). However, we found that (1) all our BDD's are OBDD's; (2) none of them contain a self loop; and (3) all our BDD's are already reduced (minimal in size); Therefore, the use of a hash table in the algorithm is redundant and can be eliminated. This modification made our code specific for the E_0 attack—but it tremendously improved the performance of the algorithm in comparison with general purpose BDD libraries that we tried to use.

3.2 Building the *LFSR* Consistency OBDD

As described in Section 2.3, S_i denotes the BDD that computes whether the internal bit z_i is consistent with the prefix $\{z_j\}_{j=1}^{i-1}$. Since each internal bit is produced by one of the LFSR's, its consistency depends on 4 earlier bits of the same LFSR as determined by the LFSR's taps. For example, for the shortest LFSR each bit must comply with the LFSR feedback polynomial: $t^{25} + t^{20} + t^{12} + t^8 + t^0$; meaning, bit z_i equals :

$$z_i = z_{i-8} \oplus z_{i-12} \oplus z_{i-20} \oplus z_{i-25} \tag{1}$$

Using our bit ordering (see Figure 1) changes the equation to:

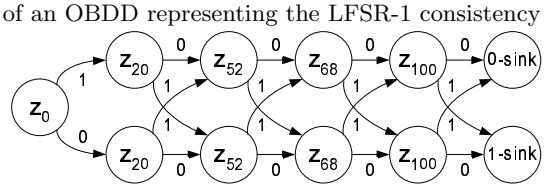
$$z_i = z_{i-32} \oplus z_{i-48} \oplus z_{i-80} \oplus z_{i-100} \tag{2}$$

Table 1 summarizes the basic consistency equations and the normalized consistency equations for all four LFSR's. Note that $LFSR_i$ produces bits with index j such that $j \equiv (i-1) \bmod 4$.

Table 1. LFSR's consistency equations

LFSR #	Basic consistency equation	Normalized consistency equation
1	$z_i = z_{i-8} \oplus z_{i-12} \oplus z_{i-20} \oplus z_{i-25}$	$z_i = z_{i-32} \oplus z_{i-48} \oplus z_{i-80} \oplus z_{i-100}$
2	$z_i = z_{i-12} \oplus z_{i-16} \oplus z_{i-24} \oplus z_{i-31}$	$z_i = z_{i-48} \oplus z_{i-64} \oplus z_{i-96} \oplus z_{i-124}$
3	$z_i = z_{i-4} \oplus z_{i-24} \oplus z_{i-28} \oplus z_{i-33}$	$z_i = z_{i-16} \oplus z_{i-96} \oplus z_{i-112} \oplus z_{i-132}$
4	$z_i = z_{i-4} \oplus z_{i-28} \oplus z_{i-36} \oplus z_{i-39}$	$z_i = z_{i-16} \oplus z_{i-112} \oplus z_{i-144} \oplus z_{i-156}$

Fig. 2. Example of an OBDD representing the LFSR-1 consistency check for bit Z_{100}



Notation: For register L_i of length $|L_i|$, we call the first $|L_i|$ bits in its bit stream (bits $\{Z_k\} : k = 4j + L_i - 1 \ \ for \ \ 0 \leq j \leq |L_i| - 1$) its *native bits*. The goal of the algorithm is to compute the native bits of all 4 LFSR's (128 bits in total).

An OBDD representing an LFSR consistency condition contains 5 variables and 11 nodes (including the 0 sink and 1 sink). Figure 2 shows the OBDD which checks the consistency condition for bit number 100. Note that a different number of OBDD's is created for each LFSR; this is because each LFSR is of different length and produces a different number of *non-native* bits. The number of *non-native* bits each LFSR produces equals to the keystream length minus the size of the LFSR. Therefore, the total number of OBDD's representing an LFSR consistency condition is $4n - 128$ which is 384 (since $n = 128$).

3.3 Building the *Compressor* OBDD

The OBDD representing the non-linear component of E_0 (denoted Q_m in Section 2.3) represents the *compressor* unit (see Section 2.1). This OBDD is built according to the known keystream bits, and according to the transition function of the compressor (see Table 3 in Appendix A).

As stated before, the compressor updates its value according to the sum of the LFSR's output bits. Therefore, we need a BDD structure to represent the sum of 4 bits. We call such a structure a *basic chain*. For state and each of the 5 possible sums, Table 3 tells us what the output bit should be. If it matches the bit given in the known keystream, we can advance to the next chain, and test the next four bits; Otherwise, this path will lead to the 0-sink. Figure 3 shows the structure of a basic chain. Table 3 shows that for all states, exactly half the paths advance to the next chain, and the other half are connected directly to the 0-sink.

The compressor BDD is built from blocks, each consisting of 16 basic chains (one for each possible state of the compressor). Half the paths from each block

Fig. 3. The structure of a single basic chain in the compressor

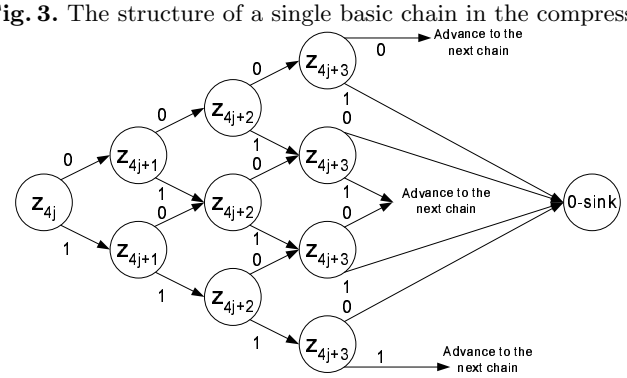
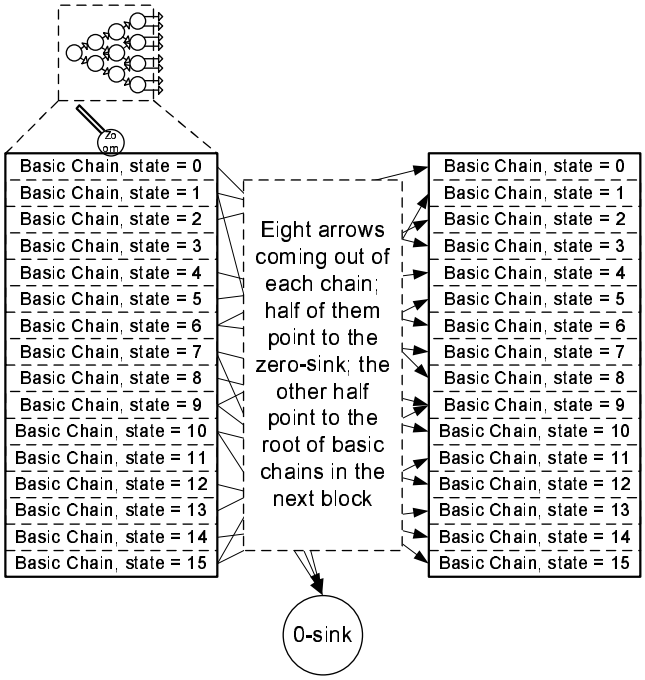


Fig. 4. Two consecutive blocks in an OBDD representing the compressor



lead to the 0-sink, while the other half advance to appropriate states on the next block. Figure 4 illustrates the full structure of the OBDD representing the compressor.

A single compressor block consists of 160 nodes and uses 4 (consecutively numbered) bits. Note, though, that each of the 4 bits "contributed" a different number of nodes to a block. Furthermore, attaching a sequence of blocks produces a non-minimal BDD, which can be reduced. For instance, for 128 blocks, the reduced compressor BDD consists of $\approx$ 14,500 nodes, rather than 20,480.

3.4 Theoretical Complexity Analysis

The time complexity of the algorithm is determined by the space complexity of the synthesized OBDD throughout the entire process of synthesis. At any stage in the process, the size of the synthesized OBDD is bounded by two bounds (See Wegener [Weg00]):

1. The number of assignments satisfying the OBDD bounds the size of the minimal OBDD representing that boolean function:

$$|P| \leq m \cdot |One(P)| \tag{3}$$

   where $One(P)$ denotes the set of satisfying assignments of the BDD P, and m is the number of variables the BDD contains ($m : 4 \longrightarrow 512$).
2. Each synthesis operation bounds the size of the synthesis result: In general, the bound is $|SYNTH(P,Q)| \leq |P| \cdot |Q|$. However, when P is an LFSR consistency check OBDD, we can use a tighter bound. This is mainly due to the structure of the OBDD's representing the LFSR's consistency check; These OBDD's effectively keep a parity bit to "remember" if the consistency is held at each point. This is why each variable appears twice in the LFSR consistency OBDD. When synthesizing another OBDD with an LFSR consistency OBDD, each node within the "window" of the parity between the lowest and highest numbered variables in the LFSR consistency OBDD is duplicated, therefore the resulting OBDD must be at most twice the size of the larger OBDD. This bound can be summed in:

$$|P| \leq |Q(m)| \cdot 2^{m-n} \tag{4}$$

   where $|Q(m)|$ is size of the OBDD representing the compressor, m is the number of variables ($m : 4 \longrightarrow 512$) and n is the amount of given keystream ($n : 1 \longrightarrow 128$ bits). Note that this bound is still loose because only nodes within the "window" of the parity are duplicated, while this bound assumes that all OBDD nodes are duplicated.

The bound on the size of the OBDD throughout the process is the lower envelope of bounds (3) and (4). Figure 5 shows the two bounds.

Using (3) (number of satisfying assignments), we get that during the first steps, each clock tick introduces 4 new variables, and one constraint since the output bit is known. This means the number of satisfying assignment is multiplied by 2^3 in each clock tick. Once we pass 25 clock ticks, all the native bits of LFSR #1 are fully determined, so the number of satisfying assumptions grows by a factor of 2^2 per clock tick. When the native bits of all four LFSR's are already set due to the consistency condition of the LFSR's (i.e., when $n \geq 39$), the number of satisfying assignments starts to decrease by half on each clock tick. The bound due to the number of satisfying assignment for $n \geq 39$ is $|P| \leq m \cdot 2^{128-n}$. See appendix B.1 for a detailed calculation of this bound.

Using (4) (magnitude of the synthesis result), we get that as long as we didn't start synthesizing with LFSR consistency OBDD's ($n \leq 25$), the OBDD size is

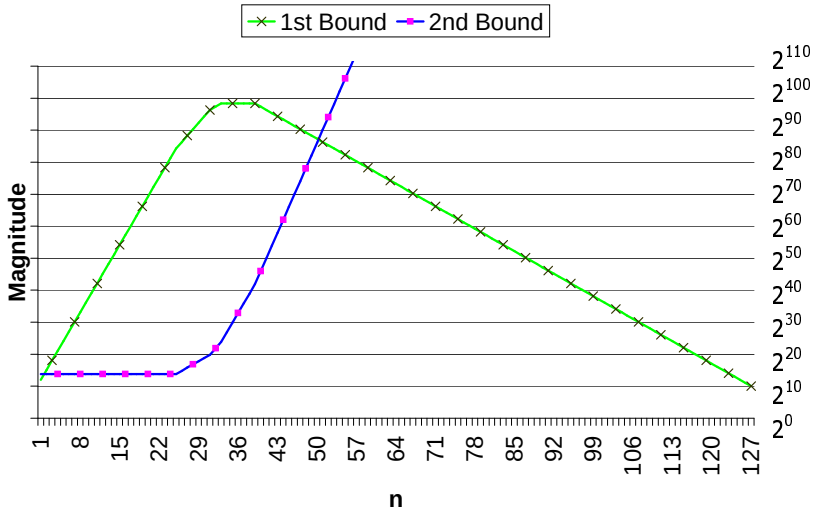


Fig. 5. The two bounds

at most the size of the OBDD representing the compressor (C_OBDD). When we begin the synthesis operation, the OBDD starts growing by a factor of 2 for each synthesis operation. Note that the number of synthesis operations for one tick depends on n. The bound due to the magnitude of the synthesis result for $n \geq 39$ is $|P| \leq |C\_OBDD| \cdot 2^{4n-128}$ (The size of C_OBDD is approximately 2^{14}). See appendix B.2 for a detailed calculation of this bound.

Calculating the intersection point of the two bounds, we get that the maximal size of the OBDD synthesized throughout the process is $|P| \approx 2^{86}$. This maximal size of the OBDD appears at clock tick $n = 50$. This gives a total time complexity of $O(2^{90})$, since we need to run the algorithm with different value of the 4 bits initializing the state machine. Note that this estimate is significantly larger than the quick estimation made by Krause. However this is still a relatively **loose** bound; The actual size of the OBDD synthesized throughout this process is in fact lower. To refine this bound, we ran a simulation which builds a histogram representing the number of nodes in the synthesized OBDD for each bit index. Using the simulation results we calculated that the maximal size of the OBDD synthesized during the process is $|P| \approx 2^{82.5}$. This gives a total time complexity of $O(2^{86.5})$, i.e., the BDD attack is roughly equivalent to the attack of [FL01] in terms of time complexity.

4 Advanced Heuristics

Since running the algorithm as-is would take impractically long, and would require an unreasonable amount of memory, we used several heuristics to lower the time and space complexity of our attack.

Table 2. Complexity results for different numbers of guessed bits

Total number of guessed bits in LFSR's #3+#4	**Maximal OBDD size (# nodes)**	**Total time complexity**
12	$2^{18.3}$	$2^{90.3}$
10	$2^{18.7}$	$2^{88.7}$
8	$2^{19.9}$	$2^{87.9}$
6	$2^{21.7}$	$2^{87.7}$
4	$2^{23.4}$	$2^{87.4}$

4.1 Guessing initial LFSR bits

The first idea was to **guess** the value of some initial LFSR bits and use the BDD method only in the remaining bits. This gives us two advantages: (a) Lower space complexity, since the size of the OBDD representing the compressor is lower, and more importantly the number of OBDD's one has to synthesize with is significantly lower. (b) This idea also allows parallelization of the attack, since one can run the algorithm with different values of guessed bits on different machines.

On our test computer (a Pentium IV with 1Gb RAM running WinXP) we were only able to run the BDD attack by guessing all 56 bits of LFSR's #1 and #2, plus a few bits of LFSR's #3 or #4 (or both). When we guessed fewer bits, the program exhausted all the available RAM and failed to complete. The best results were obtained when guessing the entire content of LFSR's #1 and #2 plus another four bits, two bits from each of the remaining LFSR's. The latter were located at the end of LFSR's #3 and #4. In this case the maximal size of the OBDD synthesized was $\approx 2^{23}$ nodes, which used 84Mb RAM[1]; Since we guess a total number of 60 bits (25+31+4), and we have to run the algorithm for all possible initial states of the compressor (4 bits), the total time complexity is $O(2^{87})$. Table 2 summarizes the results obtained when trying to run the algorithm with different numbers of guessed bits in LFSR #3 and #4.

4.2 Changing the position of the guessed bits

Another heuristics we tested was to change the position of the 4 guessed bits in LFSR #3 and LFSR #4. Recall that these guessed bits were originally selected at the end of the two LFSR's, so we decided to test how changing their location would affect the attack's complexity. The positions we tried include:

1. Guessing 2 *native* bits at the end of each LFSR (original position).

[1] The program needs to maintain two such data structures during the synthesis operation, plus various other data structures. We observed that the program's peak RAM usage reached about 400MB.

2. Guessing *native* bits that are positioned exactly where the LFSR taps are.
3. Guessing the first *non-native* bits of each LFSR.
4. Guessing bits only from one LFSR (#3 or #4).
5. Guessing bits from parallel positions in LFSR #3 and #4.

The reason for trying to guess bits on the LFSR taps positions (test #2) is that this can cause a single LFSR consistency OBDD (See Section 3) that is used during the synthesis procedure, to be totally eliminated.

However, the best results were obtained when the guessed bits were located at the end of the LFSR's (i.e., in the original bit positions). All the other alternatives increased the maximal OBDD size by factors of 2–4. Thus, the time complexity in this case is $O(2^{87})$ and the space complexity is $O(2^{23})$.

4.3 Using an intentionally defective compressor

A close examination of the transition function of the compressor (see Table 3) shows that from every state there are only 3 possible next states. Furthermore, the probability of entering each of these states is not uniform; For every state, there exists one next state that is reached with probability 1/16. For example, if we look at the reachable states from state #0, we note that state #8 is reachable with probability of 1/16. This leads to our next suggested heuristic: build a compressor that lacks the low-probability transition in every state. Naturally, this causes our attack to fail, if one bit of the known keystream was generated using such a transition. Therefore, instead of eliminating all the low probability transitions, we eliminate them only on the first 32 blocks of the compressor BDD. This means that the probability of performing a successful attack on a given known keystream is $(15/16)^{32} = 12.6\%$. This heuristic lowered the size of the synthesized OBDD by 14%. Thus, the overall complexity of the attack using an intentionally defective compressor has decreased, but is still around $O(2^{87})$.

4.4 Changing the order of synthesis

Another type of heuristic we tried was to change the order in which the OBDD's are synthesized: the order in which the various LFSR consistency OBDD's are synthesized does not affect the final outcome. The default synthesis order was by increasing bit index order. However, we conjectured that the OBDD will grow more slowly if we order the synthesis so all the LFSR OBDDs that "hit" some compressor block are synthesized consecutively, then those that hit some other compressor block, etc. We built a simulation to calculate the best order using the above criterion, and then ran the algorithm using the order produced by the simulation. Unfortunately, this heuristic produced poor results: the attack in which 4 bits of LFSR's #3 and #4 are guessed crashed for lack of memory (whereas the same attack using the default order ran to completion).

4.5 Enumerating satisfying assignments

The typical failure mode of the BDD attack is that all available memory is exhausted. However, just before such a failure occurs, we can trade time for the missing space, and still run the attack to completion. The idea is to stop the synthesis operation when the synthesized OBDD is close to the memory upper limit. Then, we enumerate all the satisfying assignments for the last synthesized OBDD, and test each assignment by generating the corresponding keystream for that assignment and comparing it to the given keystream. The overall complexity of this procedure is dominated by either the size of the synthesized OBDD or the number of satisfying assignments, whichever is larger. The time complexity of this approach is obviously poorer than using the previous heuristics—it's main advantage is that it allows one to obtain results even if the available RAM is insufficient.

5 Conclusion

We have presented an implementation of a BDD-based attack that is a short key cryptanalysis of the E_0 cipher. We have shown that several significant reductions and changes needed to be made to Krause's general attack. These changes include using OBDD's instead of FBDD's, using the exact size of the LFSR's, and skipping the use of a hash table in the implementation of the synthesis operation. We also performed an accurate complexity analysis of this attack. Furthermore, we presented some heuristics that lower the time and space complexity of this attack, and to allow parallelization of the attack on multiple machines. Our best heuristic has a time complexity which is roughly equivalent to that of the attacks of S. Fluhrer and S. Lucks ([FL01]) and O. Levy and A. Wool ([LW05]), and has significantly better space complexity than the recent work of Krause and Stegemann [KS06].

References

[Blu03] Specification of the Bluetooth system, v.1.2. Core specification, available from `http://www.bluetooth.org/spec`, 2003.

[EJ00] Patrik Ekdahl and Thomas Johansson. Some results on correlation in the bluetooth stream cipher. In *Proc. of the 10th Joint Conference on Communication and Coding*, 2000. Obertauern, Austria, March 2000.

[FL01] Scott R. Fluhrer and Stefan Lucks. Analysis of the E_0 encryption system. In *Proc. 8th Workshop on Selected Areas in Cryptography, LNCS 2259*. Springer-Verlag, 2001.

[GBM02] Jovan Dj. Golic, Vittorio Bagini, and Guglielmo Morgari. Linear cryptanalysis of Bluetooth stream cipher. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques: Advances in Cryptology*. Springer-Verlag, 2002.

[HN99] Miia Hermelin and Kaisa Nyberg. Correlation properties of the Bluetooth combiner generator. In *Information Security and Cryptology, LNCS 1787*, pages 17–29. Springer-Verlag, 1999.

[JW01] Markus Jakobsson and Susanne Wetzel. Security weaknesses in Bluetooth. In *Proc. RSA Security Conf. – Cryptographer's Track, LNCS 2020*, pages 176–191. Springer-Verlag, 2001.

[Kra02] Matthias Krause. BDD-based cryptanalysis of keystream generators. In L. Knudsen, editor, *Advances in Cryptology – EUROCRYPT'02, LNCS 1462*, pages 222–237. Springer-Verlag, 2002.

[KS06] Matthias Krause and Dirk Stegemann. Reducing the space complexity of BDD-based attacks on keystream generators. In *13th annual Fast Software Encryption Workshop (FSE 2006)*, Graz, Austria, March 2006. To appear.

[LMV05] Y. Lu, W. Meier, and S. Vaudenay. The conditional correlation attack: A practical attack on Bluetooth encryption. In *Advances in Cryptology – CRYPTO'05, LNCS 3621*, pages 97–117. Springer-Verlag, 2005.

[LV04] Y. Lu and S. Vaudenay. Faster correlation attack on Bluetooth keystream generator E_0. In *Advances in Cryptology – CRYPTO'04, LNCS 3152*, pages 407–425. Springer-Verlag, 2004.

[LW05] O. Levy and A. Wool. A uniform framework for cryptanalysis of the Bluetooth E_0 cipher. In *Proc. 1st International Conference on Security and Privacy for Emerging Areas in Communication Networks (SecureComm)*, pages 365–373, Athens, Greece, September 2005.

[MVY03] Jean-Francis Michon, Pierre Valarcher, and Jean-Baptiste Yunés. HFE and BDDs: A practical attempt at cryptanalysis. In *International Workshop on Coding Cryptography and Combinatorics, Huangshan (China)*, June 2003.

[Saa00] Markku-Juhani O. Saarinen. Re: Bluetooth und E_0. Post to sci.crypt.research, September 2000.

[SW05] Yaniv Shaked and Avishai Wool. Cracking the Bluetooth PIN. In *Proc. 3rd USENIX/ACM Conf. Mobile Systems, Applications, and Services (MobiSys)*, pages 39–50, June 2005.

[Weg00] Ingo Wegener. *Branching programs and binary decision diagrams.* SIAM, 2000.

Appendix

A Detailed specifications of the encryption system

When two Bluetooth devices wish to establish a secure communication link, they first undergo through the pairing and authentication process. The specific details of this process are not given in this paper, see [SW05] for the full details of this process. At the end of this process, both devices hold a 128 bit secret key (the link key, K_{ab}). This key is stored in a non-volatile memory area of the two devices, for future communication between these devices. This key is used to generate the *encryption key* (K_c), also known as the ***session key***. Using an algorithm (E_3), both devices derive the encryption key from the link key (K_{ab}), a ciphering offset number (COF), that is generated during the authentication process done prior to the encryption phase, and a public known random number (EN_RAND) that is exchanged between the devices. The encryption key (K_c) is then modified into another key denoted K'_c. This modification is done to lower the effective size of the session key, according to the effective length the devices have decided upon negotiation in a preliminary phase. K'_c is used in a linear manner, along

Table 3. The finite state machine transition function. NS stands for *Next State*. Each of the five main columns stands for a possible sum of the 4 LFSR bits that is input to the state machine

	Input									
Current	0		1		2		3		4	
State	Out	NS	Out	NS	Out	NS	Out	NS	Out	NS
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

with some publicly known values (the Bluetooth address of the master device and a clock, which is different for every packet) to form the initial value of E_0, for a two level keystream generator. E_0 generates a binary keystream, K_{cipher}, which is xor-ed with the plaintext. The cipher is symmetric; decryption shall be performed in exactly the same way using the same key as used for encryption.

B Detailed bounds calculation

B.1 Bound due to the number of satisfying assignments

Using the first bound term, we get that:

$n = 1$	$\lvert P\rvert \leq m \cdot 2^3$	On the first step, we have 3 free bits, and the last bit is determined
$n = 2$	$\lvert P\rvert \leq m \cdot 2^6$	Same for the next step
$n \leq 25$	$\lvert P\rvert \leq m \cdot 2^{3n}$	Same for the next steps, as long as we take initial bits from LFSR #1
$25 \leq n \leq 31$	$\lvert P\rvert \leq m \cdot 2^{75} \cdot 2^{2n-25}$	One bit is already set due to the consistency condition of LFSR #1; So we have two free bits, and the last bit is determined
$31 \leq n \leq 33$	$\lvert P\rvert \leq m \cdot 2^{75} \cdot 2^{12} \cdot 2^{n-31}$	Two bits are already set due to the consistency condition of LFSR's #1, #2; So we have one free bit, and the last bit is determined
$33 \leq n \leq 39$	$\lvert P\rvert \leq m \cdot 2^{75} \cdot 2^{12} \cdot 2^2$	Three bits are already set due to the consistency condition of LFSR's #1, #2, #3; The last bit is determined
$39 \leq n$	$\lvert P\rvert \leq m \cdot 2^{75} \cdot 2^{12} \cdot 2^2 \cdot 2^{39-n}$	Four bits are already set due to the consistency condition of LFSR's #1, #2, #3, #4; Only half of the satisfying assignments survive in each step

B.2 Bound due to magnitude of the synthesis result

$n \leq 25$	$\lvert C\_OBDD \rvert$	No synthesis operations done so far, since all bits are native
$25 \leq n \leq 31$	$\lvert C\_OBDD \rvert \cdot 2^{n-25}$	One synthesis operation per each bit produced by LFSR #1
$31 \leq n \leq 33$	$\lvert C\_OBDD \rvert \cdot 2^{n-25} \cdot 2^{n-31}$	Two synthesis operations per each tick; One for the bit produced by LFSR #1, the other for the bit produced by LFSR #2
$33 \leq n \leq 39$	$\lvert C\_OBDD \rvert \cdot 2^{n-25} \cdot 2^{n-31} \cdot 2^{n-33}$	Three synthesis operations per each tick; For the three bits produced by LFSR #1, #2, #3
$39 \leq n$	$\lvert C\_OBDD \rvert \cdot 2^{n-25} \cdot 2^{n-31} \cdot 2^{n-33} \cdot 2^{n-39}$	Four synthesis operations per each tick; For the four bits produced by LFSR #1, #2, #3, #4

Where $\lvert C\_OBDD \rvert$ denotes the size of the OBDD representing the compressor.