A Uniform Framework for Cryptanalysis of the Bluetooth E_0 Cipher

Ophir Levy and Avishai Wool
School of Electrical Engineering,
Tel Aviv University, Ramat Aviv 69978, ISRAEL.
ofirl@nice.com, yash@acm.org .

***Abstract*— In this paper we analyze the E_0 cipher, which is the encryption system used in the Bluetooth specification. We suggest a uniform framework for cryptanalysis of the E_0 cipher. Our method requires 128 known bits of the keystream in order to recover the initial state of the LFSRs, which reflects the secret key of this encryption engine. In one setting, our framework reduces to an attack of D. Bleichenbacher. In another setting, our framework is equivalent to an attack presented by Fluhrer and Lucks. Our best attack can recover the initial state of the LFSRs after solving 2^{86} boolean linear systems of equations, which is roughly equivalent to the results obtained by Fluhrer and Lucks.**

I. Introduction

A. Background

Bluetooth is a proposed standard for short range wireless communication of (potentially mobile) devices, such as cellular phones, wireless headsets, printers, cars, and turnstiles, allowing such devices to communicate with each other. Bluetooth offers methods for generating keys, authenticating users, and encrypting data.

The data encryption mechanism of the Bluetooth standard is a stream cipher that is generated by an LFSRs[1]-based keystream generator, E_0, and XORed with the plaintext. E_0 is a relatively new LFSR-based cipher, which comprises of 4 LFSRs of different lengths, which are combined by non-linear combiner logic.

A Bluetooth device resets the E_0 key after every 240 output bits, severely limiting the amount of known keystream that may be available to the cryptanalyst. In this paper we focus on "short key" attacks, that still manage to recover the key despite this limitation.

B. Related work

Several crypt-analytical results regarding E_0 ([1],[2],[3],[4],[5],[6],[7],[8]) have appeared over past years. We distinguish between two types of attacks: "short key" attacks, that need at most 240 known keystream bits and are applicable in the Bluetooth setting; and "long key" attacks, that are applicable only if the E_0 cipher is used outside the Bluetooth mode of operation.

1) Short Key Attacks: D. Bleichenbacher has shown in [1] that an attacker can guess the content of the registers of the three smaller LFSRs and of the E_0 combiner state registers with a probability of 2^{-93}. Then the attacker can compute the the contents of the largest LFSR (of length 39 bit) by "reverse engineering" it from the outputs of the other LFSRs and the combiner states. This attack requires a total of 128 bits of known plaintext and ciphertext. The reverse engineering and verification takes approximately 2^7 operations. making the total complexity of the attack 2^{100}.

Fluhrer and Lucks have described in [3] an optimized backtracking method of recovering the secret key with a computational complexity of about 2^{84}, which can be reduced to 2^{73} if a long (2^{43} bit) keystream is available.

Currently, the best short-key attack against E_0 has been suggested by Krause [5]. His attack uses Free Binary Decision Diagrams (FBDDs), a data structure for minimizing and manipulating boolean functions, for attacking LFSR-based key stream generators. The attack requires $O((2^{77})$ space. For a fixed initial state of the combiner the attack uses $O(2^{77})$ time, giving a total time complexity of $O(2^{81})$.

2) Long Key Attacks: Saarinen has showed an attack [9] that recovers the session key in a similar way to what D. Bleichenbacher showed in his attack, only that Saarinen assumed much more keystream is available within a packet and therefore the overall complexity was closer to $O(2^{93})$.

Ekdahl and Johansson have shown in [6] a method of reconstructing the initial state of the E_0 keystream generator given $O(2^{61})$ work and $O(2^{50})$ known keystream. This attack works by exploiting some weak linear correlations between the outputs of the LFSRs and the keystream output to verify if a guess on one of the LFSRs

[1]Linear Feedback Shift Register

is accurate. Previous to that, Hermelin and Nyberg published in [10] an attack with recovered initial state with $O(2^{64})$ work and $O(2^{64})$ known keystream bits. These are theoretical attacks since they require a far larger amount of consecutive keystream output than is available in a Bluetooth frame.

Golić, Bagini and Morgani show in [7] the recovery of the secret key, with $O(2^{70})$ work, along with a pre-computation stage of complexity $O(2^{80})$. They show in their work how to recover the level 2 internal state for a single packet with $O(2^{64})$ work, using linear correlations between the internal LFSR state and the observed keystream.

Currently, the best long-key attack against E_0 is by Lu and Vaudenay [8]. Their attack is a fast correlation attack, and works in $O(2^{39})$ time given $O(2^{39})$ keystream bits.

Shaked and Wool [11] have recently shown an efficient brute force attack against the Bluetooth authentication and key negotiation protocol, that relies on the fact that manufacturers limit the length of the secret PIN to inadequately short lengths (4 decimal digits is typical).

C. Contribution

In this paper we introduce a uniform framework for the derivation of the initial state of the LFSRs of the E_0 cipher. Our attacks can be viewed as a generalization of Bleichenbacher's attack [1] and Fluhrer and Lucks attack [3]: guessing the initial states of some of the LFSRs, backtracking through the different possible states of the Combiner logic, solving sets of linear equations, and checking for consistency with the given output stream cipher. Our best attack can recover the initial state of the LFSRs after solving $2^{85.38}$ boolean linear systems of equations, which is roughly equivalent to the results obtained by Fluhrer and Lucks.

Organization This paper is organized as follows: In section II , a description of the E_0 cipher is given. In section III a structured analysis of the Combiner logic is described. In section IV we present the basic method of our framework, including theoretical analysis and simulations. Sections V and VI describe some variants of our cryptanalysis framework, and Section VII concludes the work.

II. The E_0 Cipher

A detailed specification of Bluetooth security mechanisms can be found in part H of Vol 2 of [12]. In the Bluetooth system, the security layer is one of the baseband layers, which the upper layers control. The

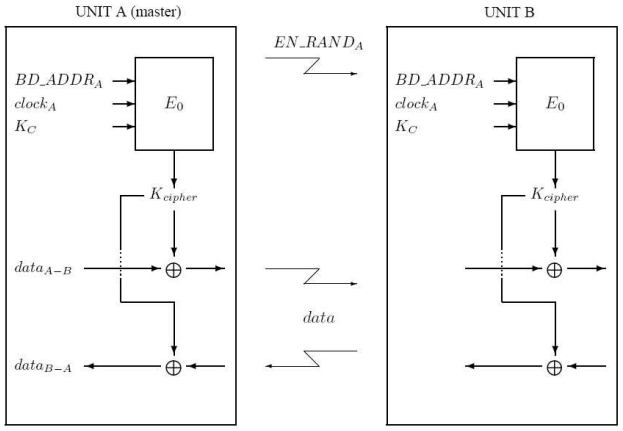

Fig. 1. Functional description of the encryption procedure.

security layer includes key management, key generation mechanisms, user/device authentication, and data encryption. The data encryption engine used within the Bluetooth security architecture is the E_0 key stream generator.

When two Bluetooth devices need to communicate securely, they first undergo an authentication and key exchange protocol that completes with each unit agreeing on a shared secret (the link key), which is used to generate the encryption key (K_C). The encryption key (K_C) is derived from the current link key, a ciphering offset number (COF), that is known from the authentication procedure done prior to the encryption phase, and a public known random number (EN_RAND). To encrypt a packet, this private key (K_C) is modified into another key denoted as K'_C. Then K'_C is used in a linear manner, along with the publicly known values, the master device Bluetooth address, and a clock, which is distinct for each packet, to form the initial state for a two level keystream generator (see Figure 1). The encryption algorithm E_0 generates a binary keystream, K_{cipher}, which is XORed with the plain text. The cipher is symmetric; decryption is performed in exactly the same way using the same key as used for encryption.

The E_0 keystream generator consists of 4 LFSRs with a total length of 128 bits, and a 4 bit finite state machine (called the Summation Combiner Logic and Blend). At each clock tick all LFSRs are clocked once, and their output bits are XORed together with one output bit of the finite state machine to produce the keystream bit. Then, the 4 LFSR's output bits are summed together. The two most significant bits of this 3 bit sum are used to update the state of the finite state machine. Table I describes the four feedback polynomials of the LFSRs, and Figure 2 describes the E_0 keystream generator engine concept.

Formally, let x_t^i denote the t^{th} symbol of $LFSR_i$.

LFSR Number	LFSR Length	LFSR Feedback Polynomial	Hamming Weight
1	25	$t^{25}+t^{20}+t^{12}+t^{8}+1$	5
2	31	$t^{31}+t^{24}+t^{16}+t^{12}+1$	5
3	33	$t^{33}+t^{28}+t^{24}+t^{4}+1$	5
4	39	$t^{39}+t^{36}+t^{28}+t^{4}+1$	5

TABLE I

THE FOUR PRIMITIVE FEEDBACK POLYNOMIALS.

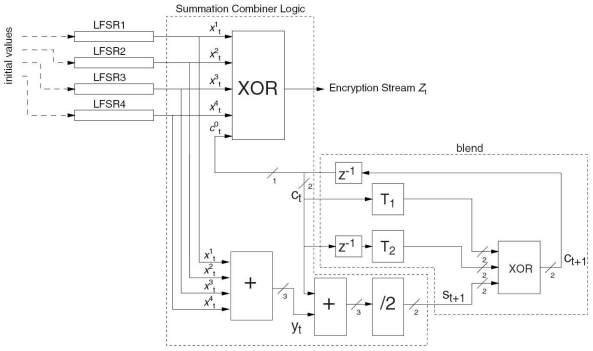

Fig. 2. E_0 keystream generator engine concept.

From the four-tuple $x_t^1, x_t^2, x_t^3, x_t^4$ we derive the value y_t as follows:

$$y_t = \sum_{i=1}^{4} x_t^i \ , \tag{1}$$

where the sum is over the integers. Thus y_t can take the values 0,1,2,3,4. The Summation Combiner uses 4 state bits, denoted $c_t^1, c_t^0, c_{t-1}^1, c_{t-1}^0$. The output of the SCLB[2] at time t (i.e., the output keystream bit) is given by the following equation:

$$z_t = x_t^1 \oplus x_t^2 \oplus x_t^3 \oplus x_t^4 \oplus c_t^0 \ . \tag{2}$$

The intermediate value of the SCLB, S_{t+1}, as described in Figure 2, is defined by:

$$s_{t+1} = \begin{bmatrix} s_{t+1}^0 \\ s_{t+1}^1 \end{bmatrix} = \left\lfloor \frac{y_t + c_t}{2} \right\rfloor \in \{0,1,2,3\} \ . \tag{3}$$

c_t is the value of (c_t^0, c_t^1) viewed as an integer in $[0,3]$.

$$c_{t+1} = \begin{bmatrix} c_{t+1}^0 \\ c_{t+1}^1 \end{bmatrix} = s_{t+1} \oplus T_1[c_t] \oplus T_2[c_{t-1}] \ , \tag{4}$$

where $T_1[.]$ and $T_2[.]$ are two different linear bijections over $GF(4)$. The mapping T_1 and T_2 are defined as described in Table II.

The mappings are linear, and therefore can be realized using XOR gates; i.e.,

[2]Summation Combiner Logic and Blend

x	$T_1[x]$	$T_2[x]$
00	00	00
01	01	11
10	10	01
11	11	10

TABLE II

T_1 AND T_2 LINEAR BIJECTIONS.

$$\begin{aligned} T_1: &\quad (x_1, x_0) \mapsto (x_1, x_0), \\ T_2: &\quad (x_1, x_0) \mapsto (x_0, x_1 \oplus x_0), \end{aligned}$$

The keystream generator needs to be loaded with an initial value for the four LFSRs (128 bits in total) and the 4 bits that specify the values of c_t and c_{t-1}. During the initialization phase these 4 bits are zeroed. The 128-bit initial value of the LFSRs is derived from four inputs by using the keystream generator itself. The input parameters are the secret key (K_c), a 128-bit random number ($RAND$), a 48-bit Bluetooth address, and the 26 master clock bits (CLK_{26-1}).

When the encryption key has been created, the LFSRs are loaded with their initial values. Then, 200 stream cipher bits are created by operating the generator. Of these bits, the last 128 are fed back in parallel into the keystream generator as an initial value of the four LFSRs. The value of c_t and c_{t-1} are kept. From this point on, when clocked, the generator produces the encryption (or decryption) sequence, which is XORed with the transmitted (or received) payload data. For a detailed description of the encryption process refer to [12],[13].

III. STRUCTURAL ANALYSIS OF THE SUMMATION COMBINER LOGIC AND BLEND

In this section we will describe the work that we did in defining a uniform framework for the cryptanalysis of the E_0 cipher. We will view the SCLB as a finite state machine that has four input bits ($x_t^1, x_t^2, x_t^3, x_t^4$), four bits that define its state ($c_t^1, c_t^0, c_{t-1}^1, c_{t-1}^0$) and one output bit ($Z_t$). For each clock of the LFSRs, 4 new input bits (the output of the LFSRs) enter the state machine, the state machine transfers to its next state and 1 output bit is produced.

A. The SCLB as a Finite State Machine

By running the SCLB over all possible inputs (4 input bits) and over the different states (4 bits states), we computed a 16x16 table that defines the output bit and the next state, for each possible state and each valid input value: a simple (input,state)⇒(output,next state) state machine table (see Table III).

We observe that the state machine is *only* dependent on the current state and on the *sum* of input bits (and not on the different values of the input bits). This observation allows us to reduce Table III into a 16 states x 5 sums table, in which each cell holds the next state and the output bit that is received for the relevant current state and sum of input bits. A similar observation was made in [8].

Looking at Table III, we can see that each row in the table holds exactly 3 possible next states (and not 5). Moreover, in each row, two next-states correspond to one output bit value, and three next-states correspond to the other output bit value. Therefore, we arranged the information in a different form. We gathered all of the sums that generate an output bit $Z_t = 0$ in one column, and all of the sums that generate an output bit $Z_t = 1$ in another column, and created a table $F(q, z)$: for a 4 bit state q and an output bit z, $F(q, z)$ contains either 2 or 3 tuples of (sum, next state). $F(q, z)$ appears in Table IV. Notice that for each state, exactly 3 sums produce the bit value z, and 2 sums produce $\overline{z}$.

Our attack is based on these findings; If we assume to know the current state of the SCLB, and the current output bit, then:

1) There are either 2 or 3 possible next-states.
2) For each such next-state we can derive linear equations on the LFSRs initial state bits.

IV. The basic method

A. Chains

Definition 4.1: Let $Z = z_1, \ldots, z_T$ be an output sequence of E_0, and denote the initial state of the SCLB by $q_0 \in [0, 15]$. A *chain* for Z is a sequence $a_1, \ldots, a_T$ of values $a_t \in \{0, 1, 2, 3, 4\}$, such that for each t,

$$a_t = F(q_{t-1}, z_t).sum[i] \qquad \text{for } i \in \{0, 1, 2\} \ , \quad (5)$$

and

$$q_t = F(q_{t-1}, z_t).NextState[i] \quad \text{for the same } i \ . \quad (6)$$

During the backtracking phase we produce a set of chains. For each chain we wish to get 128 binary linear equations of 128 unknown bits of the initial state. Solving this linear equation will supply an initial state of the LFSRs, that should be tested to be the initial state. Table IV allows us to move from one state to another according to the sum of the input bits and the known output. In each cell of the state machine table we can see that either 2 or 3 options are possible. Given T known output bits, we will guess an initial state $q_0 = [c_t^1, c_t^0, c_{t-1}^1, c_{t-1}^0]$ of the state machine. For each position in the range $[1, T]$ we guess the sum of input bits, $y_t = \sum_{i=1}^{4} x_t^i$, out of the possible sum values according to the $F(q, z)$ table, for the given output bit. We receive a chain of T sums that we have guessed. Each guess will lead us to the next state, and according to the next output bit we may guess the next sum. Note that the chains need not be all of the same length.

B. The linear equations

For each sum, we always get one linear equation, based on the parity of the sum:

$$x_t^1 \oplus x_t^2 \oplus x_t^3 \oplus x_t^4 = c_t^0 \oplus z_t \ , \quad (7)$$

where z_t is known and c_t^0 is extracted from the state machine. However, depending on the guessed sum value, we may be able to write additional equations. For a guess of $y_t = 0$ or $y_t = 4$ we get 4 linear equations.

For a sum of $y_t = 0$ we have:

$$x_t^1 = 0 \ , \ x_t^2 = 0 \ , \ x_t^3 = 0 \ , \ x_t^4 = 0$$

and for a sum of $y_t = 4$ we have:

$$x_t^1 = 1 \ , \ x_t^2 = 1 \ , \ x_t^3 = 1 \ , \ x_t^4 = 1$$

Note that the basic parity equation is linearly dependent on the 4 equations.

For a guess of $y_t = 1$ or $y_t = 3$ we get 4 different options of 4 linear equations each. Thus we have two possible strategies in the backtracking process: (1) for each sum of 1 or 3, try all 4 options, each giving 4 equations, or (2) only use the basic parity equation (Equation 7).

For a guess of $y_t = 2$ we have 6 different options of 4 linear equations each.

Obviously, a sum of 0 or 4 is very cheap in terms of backtracking cost, since we receive 4 equations for 1 output bit. However, on average only $2/5$ of the sums in a chain fall into the category. A sum of 1 or 3 costs us 4 possibilities and each will supply 4 equations. A sum of 2, which is the most expensive option, will supply 6 possibilities and 4 equations each. Therefore, in our backtracking process whenever we guess a sum of 2, we use only the basic parity equation.

In general, the goal is to go over all of the possible chains for the given output bit stream. Each chain ends when 128 equations are obtained. For each chain we solve a boolean linear system, that gives us the initial state of the LFSRs. Then we check if this solution is consistent with all of the given output keystream. We assume that we can use a hardware module that efficiently solves a linear system of 128 boolean equations of 128 boolean unknowns, therefore we count the computational complexity for such a step as $O(1)$.

State	Input	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	Output bit	0	1	1	0	1	0	0	1	1	0	0	1	0	1	1	0
	Next state	0	0	0	4	0	4	4	4	0	4	4	4	4	4	4	8
1	Output bit	0	1	1	0	1	0	0	1	1	0	0	1	0	1	1	0
	Next state	12	12	12	8	12	8	8	8	12	8	8	8	8	8	8	4
2	Output bit	0	1	1	0	1	0	0	1	1	0	0	1	0	1	1	0
	Next state	4	4	4	0	4	0	0	0	4	0	0	0	0	0	0	12
3	Output bit	0	1	1	0	1	0	0	1	1	0	0	1	0	1	1	0
	Next state	8	8	8	12	8	12	12	12	8	12	12	12	12	12	12	0
4	Output bit	1	0	0	1	0	1	1	0	0	1	1	0	1	0	0	1
	Next state	5	1	1	1	1	1	1	13	1	1	1	13	1	13	13	13
5	Output bit	1	0	0	1	0	1	1	0	0	1	1	0	1	0	0	1
	Next state	9	13	13	13	13	13	13	1	13	13	13	1	13	1	1	1
6	Output bit	1	0	0	1	0	1	1	0	0	1	1	0	1	0	0	1
	Next state	1	5	5	5	5	5	5	9	5	5	5	9	5	9	9	9
7	Output bit	1	0	0	1	0	1	1	0	0	1	1	0	1	0	0	1
	Next state	13	9	9	9	9	9	9	5	9	9	9	5	9	5	5	5
8	Output bit	0	1	1	0	1	0	0	1	1	0	0	1	0	1	1	0
	Next state	14	14	14	2	14	2	2	2	14	2	2	2	2	2	2	6
9	Output bit	0	1	1	0	1	0	0	1	1	0	0	1	0	1	1	0
	Next state	2	2	2	14	2	14	14	14	2	14	14	14	14	14	14	10
10	Output bit	0	1	1	0	1	0	0	1	1	0	0	1	0	1	1	0
	Next state	10	10	10	6	10	6	6	6	10	6	6	6	6	6	6	2
11	Output bit	0	1	1	0	1	0	0	1	1	0	0	1	0	1	1	0
	Next state	6	6	6	10	6	10	10	10	6	10	10	10	10	10	10	14
12	Output bit	1	0	0	1	0	1	1	0	0	1	1	0	1	0	0	1
	Next state	11	7	7	7	7	7	7	3	7	7	7	3	7	3	3	3
13	Output bit	1	0	0	1	0	1	1	0	0	1	1	0	1	0	0	1
	Next state	7	11	11	11	11	11	11	15	11	11	11	15	11	15	15	15
14	Output bit	1	0	0	1	0	1	1	0	0	1	1	0	1	0	0	1
	Next state	15	3	3	3	3	3	3	7	3	3	3	7	3	7	7	7
15	Output bit	1	0	0	1	0	1	1	0	0	1	1	0	1	0	0	1
	Next state	3	15	15	15	15	15	15	11	15	15	15	11	15	11	11	11

TABLE III

THE 16X16 SCLB STATE MACHINE TABLE.

The state and next state are shown as decimate integer in [0,15].
The 4 state bits are $(c_t^1, c_t^0, c_{t-1}^1, c_{t-1}^0)$, with c_t^1 as the MSB.

Output bit	0		1	
State	Sum	Next State	Sum	Next State
0	0	0	1	0
	2	4	3	4
	4	8		
1	0	12	1	12
	2	8	3	8
	4	4		
2	0	4	1	4
	2	0	3	0
	4	12		
3	0	8	1	4
	2	12	3	0
	4	0		
4	1	1	0	5
	3	13	2	1
			4	13
5	1	13	0	9
	3	1	2	13
			4	1
6	1	5	0	1
	3	9	2	5
			4	9
7	1	9	0	13
	3	5	2	9
			4	5

Output bit	0		1	
State	Sum	Next State	Sum	Next State
8	0	14	1	14
	2	2	3	2
	4	6		
9	0	2	1	2
	2	14	3	14
	4	10		
10	0	10	1	10
	2	6	3	6
	4	2		
11	0	6	1	6
	2	10	3	10
	4	14		
12	1	7	0	11
	3	3	2	7
			4	3
13	1	11	0	7
	3	15	2	11
			4	15
14	1	3	0	15
	3	7	2	3
			4	7
15	1	15	0	3
	3	11	2	15
			4	11

TABLE IV

SCLB STRUCTURE REARRANGEMENT, $F(q, z)$.

C. Estimated number of chains

We first estimate the number of chains we need to backtrack over. From Table IV it can be seen that in each step of the backtracking process, there are either 2 or 3 next states, and that for either value of the output z_t, the number of next states is distributed uniformly. Therefore, we can model these possibilities as a random choice and assume that with probability $1/2$ there are 2 next states, and with probability $1/2$ there are 3. Therefore we can recursively calculate the number of chains in the backtracking process $W(T)$ for each additional output bit:

$$W(t) = \frac{1}{2}\cdot 2W(t-1) + \frac{1}{2}\cdot 3W(t-1) = \frac{5}{2}\cdot W(t-1) \quad (8)$$

and so, if we backtrack to the depth of T we get

$$W(t=T) = \left(\frac{5}{2}\right)^T \quad (9)$$

chains.

D. Estimated backtracking chain lengths

Next, we would like to estimate the average length of the backtracking chains that are required. The lengths of the chains are calculated out of the number of equations that we receive from the process. Let $N(t)$ denote the expected number of equations that we have obtained using t output bits.

1) Using sums 0,1,3,4: Using the backtracking method from the previous section, we always get 1 equation for every output bit. We get 3 additional equations per a sum of $0/1/3/4$. In the enumeration space we backtrack through all possibilities, so the 5 values of the sum y_t are uniformly distributed, and therefore the probability of each sum is $1/5$. Hence, the probability of having a sum of 0/1/3/4 is $4/5$.

Now we can define the expectation of the number of equations through a recursive equation, by adding another output bit in each step, and derive the expected length of the backtracking chains:

$$N(t) = N(t-1) + 1 + \frac{4}{5}\cdot 3 = N(t-1) + \frac{17}{5}\ . \quad (10)$$

So for a backtracking depth of T, we get, on average,

$$N(T) = \frac{17}{5}\cdot T \quad (11)$$

equations.
Since the requirement is to have a sufficient number of equations (more than 128), we get a bound on the depth T:

$$N(T) = \frac{17}{5}\cdot T \geq 128 \quad (12)$$

$$\implies\ T \geq \frac{128\cdot 5}{17} = 37.64\ , \quad (13)$$

Note that the minimum number of equations is 39 (the size of the largest LFSR) so all variables appear in the set of equations, and therefore, this will be the estimate.

2) Using only sums 0,4: Similarly, for the other strategy, in which the default parity equation is counted for a sum of 1 or 3, we received the following results: Each sum contributes 1 equation (the default one) and only a sum of 0 or 4 contributes 3 additional equations. As previously explained, the probability for a sum of 0 or 4 will be in this case $2/5$.

$$N(t) = N(t-1) + 1 + \frac{2}{5}\cdot 3 = N(t-1) + \frac{11}{5}\ . \quad (14)$$

The recursive function can be translated into a linear one, giving us a bound on the depth T:

$$N(T) = \frac{11}{5}\cdot T \geq 128 \quad (15)$$

$$\implies\ T \geq \frac{128\cdot 5}{11} = 58.15\ . \quad (16)$$

Thus, in this case the estimated depth of the backtracking chains is 59.

E. Computational complexity of the backtracking process

Finally, we estimate the number of linear systems that should be solved during the process. Let T denote the number of required output bits (the length of the chains). For this estimate, we are assuming that all of the chains have the same length. Assume that R positions out of the total T positions in a chain hold the sums of 0/1/3/4. The other $T-R$ positions hold sum of 2. Note that among the positions $t \in R$, we have that

$$Pr\{y_t = 0 \text{ or } 4\} = \frac{1}{2} \qquad \forall t \in \mathrm{R} \quad (17)$$

$$Pr\{y_t = 1 \text{ or } 3\} = \frac{1}{2} \qquad \forall t \in \mathrm{R}\ . \quad (18)$$

For a sum of 0 or 4, there is only one option to backtrack through. For a sum of 1 or 3 there are 4 options to backtrack through. Therefore, going over all possible combinations of sums in R, the expected number of linear equations systems that should be solved per chain can be estimated as,

$$E = \sum_{i=0}^{R} \left(\begin{array}{c} R \\ i \end{array} \right) \left(\frac{1}{2}\right)^{i} \left(\frac{1}{2}\right)^{R-i} \cdot 1^{i} \cdot 4^{R-i} =$$

$$\frac{1}{2^R} \sum_{i=0}^{R} \left(\begin{array}{c} R \\ i \end{array} \right) 4^{R-i} \cdot 1^{i} = \frac{1}{2^R} (4+1)^R = \left(\frac{5}{2}\right)^R \quad (19)$$

i.e., for R positions we have $(2.5)^R$ systems to solve.

Combining the above estimates, we can calculate the *total* number of linear systems that should be solved. As seen in previous subsection the number of chains is $(5/2)^T$. For each output bit we have 1 default equation (parity equation) and for R bits out of the T bits we have 3 additional equations. By equation (19), for each chain we have $(5/2)^R$ equations to solve. We require 128 equations in order to solve the LFSRs initial state. Therefore, our constraint is: $T + 3R \geq 128 \;\; \rightarrow \;\; R = (128 - T)/3$.

Hence, estimated total number of equations to solve is:

$$\left(\frac{5}{2}\right)^T \cdot \left(\frac{5}{2}\right)^R = \left(\frac{5}{2}\right)^{T+R} = \left(\frac{5}{2}\right)^{T+\frac{128-T}{3}}$$

For a minimal T we chose $T = 39$ (the size of the largest LFSR) and we get the estimation for the total number of linear systems that should be solved: $2^{90.63}$. For the total computational complexity we should add the initial state 4 bit which brings to a total complexity of $2^{94.63}$.

F. Simulations

The analysis in the previous sections was somewhat inaccurate, since we assumed that all the chains are of equal length, which we estimated as an expectation. To validate these calculations we performed simulations. The simulations ran on a Pentium 4 processor with 256Mbyte of RAM. We examined the strategy of adding 4 options to a guess of a sum=1 or 3. As we ran the simulation, we calculated the computational effort for increasing values of T, and extrapolated the results to a system of 128 equations. From the simulations that we did, the computational complexity was extrapolated to be $2^{91.93}$ (close to the estimation above) for enumerating the total tree, solving 1 boolean linear equations system for each chain that supplies 128 equations of 128 unknowns. In addition there are 4 initial state bits that should be guessed (which costs 2^4). This brings us to a total computational complexity of $2^{95.93}$. This is significantly worse than the result of Fluhrer and Lucks [3].

V. Guessing a single register

In this phase of our framework we tried to reduce the computational complexity by guessing the contents of the first (shortest) LFSR, and implementing a similar process but with shorter chains in the backtracking analysis process.

Assuming that the first LFSR is guessed (which costs us 2^{25} operations), we can create a process similar to that described in Section IV, and is based on both the output bit and the guessed $LFSR_1$ bit. We built a table similar to Table IV that holds the sum and next state for a given output bit, current state and a guessed bit of $LFSR_1$. The possible sums in the table are 0/1/2/3 (we have 1 bit less). Considering the fact that the first LFSR is guessed then the number of unknowns is $128 - LFSR_1 = 103$. This means that only 103 equations are required, and the boolean linear system to be solved is of a size of 103 equations and unknowns. The parity equation now looks like this:

$$x_t^2 \oplus x_t^3 \oplus x_t^4 = c_t^0 \oplus z_t \oplus x_t^1 \;, \quad (20)$$

where x_t^1 is the guessed bit of the $LFSR_1$ at time t.

Let

$$y'_t = \sum_{i=2}^{4} x_t^i \;. \quad (21)$$

For a guessed sum of $y'_t = 0$ or 3 we get 3 equations. For a sum of $y'_t = 1$ or 2 we have 3 different options of 3 linear equations each, as follows. For a sum of 1 we get:

$$\begin{array}{ccccc} x_t^2 = 1 & & x_t^2 = 0 & & x_t^2 = 0 \\ x_t^3 = 0 & or & x_t^3 = 1 & or & x_t^3 = 0 \\ x_t^4 = 0 & & x_t^4 = 0 & & x_t^4 = 1, \end{array}$$

and for a sum of 2 we get:

$$\begin{array}{ccccc} x_t^2 = 0 & & x_t^2 = 1 & & x_t^2 = 1 \\ x_t^3 = 1 & or & x_t^3 = 0 & or & x_t^3 = 1 \\ x_t^4 = 1 & & x_t^4 = 1 & & x_t^4 = 0. \end{array}$$

A. Simulations

In order to estimate the complexity of this method, we modified the simulation from the previous section. As explained in the previous section, in some of the chains there are 3 options to be examined (for sum =1 or 2) and therefore in each such step the computational complexity is increased by a factor of 3. In the enumeration space there is a uniform distribution of sums, and therefore in 50% of the positions we will have the sum of 1 or 2. We examined the 3 possible strategies for dealing with sums of $y'_t = 1$ or 2.

a) Strategy 1: Backtracking through 3 options of 3 equations each. This option actually runs though all of the possible chains that exists in the tree. All options of chains are backtracked through, with no complexity reduction (as done in previous method). Thus, this strategy is equivalent to a full brute force attack.

b) Strategy 2: Using only the parity equation for a sum of 1 or 2. The simulation shows that using this Strategy, the computational complexity of backtracking the entire tree for achieving 103 equations of 103 unknowns, is approximately $2^{57.68}$ operations. Adding to this the guessed $LFSR_1$ bits and the initial state bits brings us to $2^{57.68} \cdot 2^{25} \cdot 2^4 = 2^{86.68}$ operations. This solution brings us quite close to the computational complexity that Fluhrer and Lucks received in [3].

c) Strategy 3: (Backtracking through 3 options for sum=1). This option was investigated and was found to be less efficient than strategy 2: we found that the number of operations is $2^{65.5}$, which gives a total computational complexity of $2^{94.5}$.

VI. Guessing more than one register

A. Guessing two registers

Assuming that the first and the second LFSRs are guessed (which costs us 2^{56} operations) we can create a similar backtracking process, based on the output bit and the guessed bits of $LFSR_1$ and $LFSR_2$. Using this method, we have possible sums of 0/1/2. Furthermore, the number of unknowns is $128-(LFSR_1+LFSR_2) = 72$. This means the boolean linear system to be solved is of size of 72 equations and unknowns. In this case the parity equation is: $x_t^3 \oplus x_t^4 = c_t^0 \oplus z_t \oplus x_t^1 \oplus x_t^2$, where x_t^1 and x_t^2 are the guessed bits of the $LFSR_1$ and $LFSR_2$ at time t, respectively. Let

$$y_t'' = \sum_{i=3}^{4} x_t^i \,. \tag{22}$$

Then, for $y_t'' = 0$ or 2 we get 2 equations. For a sum of 0 we get

$$\begin{aligned} x_t^3 &= 0 \\ x_t^4 &= 0, \end{aligned}$$

and for a sum of 2 we get:

$$\begin{aligned} x_t^3 &= 1 \\ x_t^4 &= 1. \end{aligned}$$

For $y_t'' = 1$ we get 2 options of 2 equations each. A sum of 0 or 2 is cheap in terms of backtracking cost, since we receive 2 equations for 1 output bit. A sum of 1 costs us 2 possibilities and each will supply 2 equations, so we only use the parity equation. This is, effectively, the method used by Fluhrer and Lucks in [3].

B. Simulations

We have performed simulations in order to calculate the computation complexity of the suggested backtracking method in this subsection. In 1/3 of the locations we have a sum of 1, giving 1 equation only, for such a location. The results show that computational complexity of backtracking the whole tree for achieving 72 equations of 72 unknowns, is approximately $2^{25.38}$ operations. Adding to this the guessed $LFSR_1$ and $LFSR_2$ bits and the initial state bits brings us to $2^{25.38} \cdot 2^{56} \cdot 2^4 = 2^{85.38}$ operations, which agrees with the result of Fluhrer and Lucks [3].

C. Guessing three registers

Completing the uniform framework that was described in this work, leads us to the last method of guessing 3 registers and backtracking through a single register. This is the exact case that D. Bleichenbacher used in [1] as described in Section I-B.

VII. Conclusions

We have presented a uniform framework for short-key cryptanalysis of the E_0 cipher, based on a backtracking process in the enumeration space of the different states of the Summation Combiner Logic and Blend unit. During the analysis we used the fact that the input to the state machine is based on the sum of input bits and not on the value of the bits themselves. Furthermore, after revising the state machine table we found that depending on the input bit, each state can advance to only 2 or 3 next states (and not to all 16 states). These properties allowed a relatively efficient backtracking process through the enumeration space. We identified several variants of the general method, two of which are equivalent to the attacks of Bleichenbacher [1] and of Fluhrer and Lucks [3]. The variant with the best performance in terms of computational complexity achieved about $O(2^{86})$ operations, with no significant memory requirements. We believe that we have explored this framework to its limits, and that faster attacks may be possible only if other techniques are used, such as the BDD methods of Krause [5].

REFERENCES

[1] D. Bleichenbacher. Personal communication in [2].

[2] M. Jakobsson and S. Wetzel, "Security weaknesses in Bluetooth," in *Proc. RSA Security Conf. – Cryptographer's Track, LNCS 2020*, pp. 176–191, Springer-Verlag, 2001.

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

[4] F. Armknecht, "A linearization attack on the Bluetooth key stream generator." Cryptology ePrint Archive, report 2002/191, available from http://eprint.iacr.org/2002/191/ , 2002.

[5] M. Krause, "BDD-based cryptanalysis of keystream generators," in *Advances in Cryptology – EUROCRYPT'02, LNCS 1462* (L. Knudsen, ed.), pp. 222–237, Springer-Verlag, 2002.

[6] P. Ekdahl and T. Johansson, "Some results on correlation in Bluetooth stream cipher," in *Proceedings of the* 10^{th} *Joint Conference on Communication and Coding*, Obertauern, Austria, March 2000.

[7] J. Golić, V. Bagini, and G. Morgani, "Linear cryptanalysis of Bluetooth stream cipher," in *Proceedings of Eurocrypt 2002*, Springer, 2002.

[8] Y. Lu and S. Vaudenay, "Faster correlation attack on Bluetooth keystream generator E0," in *Advances in Cryptology – CRYPTO'04, LNCS 3152*, pp. 407–425, Springer-Verlag, 2004.

[9] M. Saarinen, "Re: Bluetooth und E_0." Posting to sci.crypt.research, September 2000.

[10] M. Hermelin and K. Nyberg, "Correlation properties of the Bluetooth combiner generator," in *Information Security and Cryptology, LNCS 1787*, pp. 17–29, Springer-Verlag, 1999.

[11] Y. Shaked and A. Wool, "Cracking the Bluetooth PIN," in *Proc. 3rd USENIX/ACM Conf. Mobile Systems, Applications, and Services (MobiSys)*, (Seattle, WA), June 2005. To appear.

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

[13] "Wireless medium access control (MAC) and physical layer (PHY) specifications for wireless personal area networks (WPANs)." IEEE std 802.15.1-2002, IEEE computer society.