Analysis of the E_0 Encryption System

Scott Fluhrer[1] and Stefan Lucks[2]

[1] Cisco Systems, Inc., 170 West Tasman Drive, San Jose, CA 95134, USA
sfluhrer@cisco.com
[2] University of Mannheim, 68131 Mannheim, Germany
lucks@th.informatik.uni-mannheim.de

Abstract. The encryption system E_0, which is the encryption system used in the Bluetooth specification, is examined. In the current paper, a method of deriving the cipher key from a set of known keystream bits is given. The running time for this method depends on the amount of known keystream available, varying from $O(2^{84})$ if 132 bits are available to $O(2^{73})$, given 2^{43} bits of known keystream.
Although the attacks are of no advantage if E_0 is used with the recommended security parameters (64 bit encryption key), they provide an upper bound on the amount of security that would be made available by enlarging the encryption key, as discussed in the Bluetooth specification.

1 Introduction

We give algorithms for deriving the initial state of the keystream generator used within E_0 given some bits of keystream with less effort than exhaustive search. From this, we derive a method for reconstructing the session encryption key used by E_0 based on some amount of keystream output. E_0 uses a two level rekeying mechanism, using the key to initialialize the level 1 keystream generator to produce the initial state for the level 2 keystream generator, which produces the actual keystream used to encrypt the data.

We use a known keystream to reconstruct the initial state for the level 2 keystream generator, which we then use to reconstruct the initial state for the level 1 keystream generator, from which we can directly deduce the encryption key. Reconstructing the state of the level 2 keystream generator takes an expected $O(2^{76})$ to $O(2^{84})$ work effort (based on the amount of known keystream available). Another attack with even more keystream available takes $O(2^{72})$ work.

By reconstructing the state from either 1 or 2 packets that are encrypted during the same session, we can reconstruct the state of the level 1 keystream generator in an expected $O(2^{81})$ or $O(2^{51})$ time, which gives a total of $O(2^{73})$ to $O(2^{84})$ work effort.

This paper is structured as follows. In Section 2, the E_0 keystream generator, and how it is used within the Bluetooth system is described. In Section 3, previous analysis and results are summarized. Section 4 presents our base attack against the keystream generator, Section 5 describes how to use it against the level 2 generator. Section 6 deals with another approach to attack the keystream generator and the second level, if a huge amount of known keystream is available. Section 7 describes the basic attack on the first level of E_0 , given one state of the level 2 generator, while Section 2 deals with an attack given two such states. Section 9 comments on attacking the full E_0 system. Section 10 concludes and discusses the ramifications on the Bluetooth system.

2 Description of E_0

E_0 is an encryption protocol that was designed to provide privacy within the Bluetooth wireless LAN specification. When two Bluetooth devices need to communicate securely, they first undergo a key exchange protocol that completes with each unit agreeing on a shared secret, which is used to generate the encryption key (K_C). To encrypt a packet, this private key (K_C) is combined with a publicly known salt value (EN_RAND) to form an intermediate key (K'_C)[1]. Then, K'_C is used in a linear manner, along with the publicly known values, the Bluetooth address, and a clock which is distinct for each packet, to form the initial state for a two level keystream generator.

The keystream generator consists of 4 LFSRs with a total length of 128 bits, and a 4 bit finite state machine, refered to as the blender FSM. For each bit of output, each LFSR is clocked once, and their output bits are exclusive-or'ed together with one bit of output from the finite state machine. Then, the 4 LFSR outputs are summed together. The two most significant bits of this 3-bit sum are used to update the state of the finite state machine. We will refer to the 25 bit LFSR as LFSR1, the 31 bit LFSR as LFSR2, the 33 bit LFSR as LFSR3 and the 39 bit LFSR as LFSR4. We will also refer to the finite state machine as the blender FSM. The generator is shown in Figure 1. Note that the least significant bit (LSB) of the sum of the four LFSRs is their bit-wise XOR.

There are logically two such keystream generators. The key of the first level keystream generator is shifted into the LFSRs, while clearing the blender FSM. Then, 200 bits are generated and discarded. Then, the output of this keystream

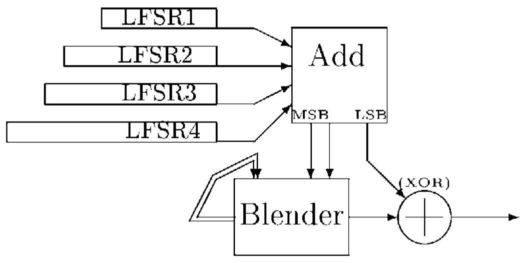

Fig. 1. The E_0 keystream generator.

[1] The attacks in the current paper actually provide the value of K'_C.

generator is collected, and is used to initialize the LSFRs of what we call the second level keystream generator, which is structurally identical to the first level keystream generator. This initialization is done by collecting 128 output bits, parallel loading them into the LSFRs, and making the initial second level FSM state be the final first level FSM state.

This output of this second generator is then used as an additive stream cipher to encrypt the packet.

3 Description of Previous Work

In a sci.crypt.research posting [6], Markku-Juhani O. Saarinen showed an attack that rederived the session key. This attack consisted of guessing the states of the 3 smaller LFSRs and the blender FSM, and using those states and the observed keystream to compute whether there is a consistent output from LFSR4 that is consistent with that assumption.

In the original posting, he estimated the attack to have overall complexity of $O(2^{100})$. However, he assumed that only 125 bits of keystream were available, and so he assumed a significant amount of time would be spent checking false hits. Since significantly more keystream is available within a packet, the true complexity is closer to $O(2^{93})$ expected.

Our attacks can be viewed as refinements of Saarinen's attack by taking the same basic approach of guessing the initial states of part of the cipher, and checking for consistency. However, our attacks take advantage of additional relationships within E_0 and use them to gain some performance.

Ekdahl and Johansson have shown in [2] how to extract the initial state from the keystream generator used in E_0 given $O(2^{61})$ time and $O(2^{50})$ known keystream. Their 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 is accurate. Previous to that, Hermelin and Nyberg published in [4] an attack which recovered the initial state with $O(2^{64})$ work and $O(2^{64})$ known keystream. However, these are theoretical attacks as they require a far larger amount of consecutive keystream output than is available.

A time-spaces tradeoff attack has been described by Jakobsson and Wetzel [5]. Given N key streams and running time T, it is possible to recover one of the N keys if $N * T > 2^{132}$. A similar attack on the A5 keystream generator has been previously described by Golic [3].

Our attacks resemble a general type of attack, the *linear consistency attack*, which has been described as early as 1989 by Zeng, Yang, and Rao [7].

4 Base Attack on the E_0 Keystream Generator

The base attack rederives the initial settings of the LFSRs, given a limited (132 or so bits) keystream output. We will later show how this attack can be separately optimized for both levels of the keystream generators. For this attack, you assume the initial settings of the blender FSM and the contents of $LFSR1$

and $LFSR2$, and maintain for each state the current settings of the blender FSM, and a set $\mathcal{L}$ of linear equations on the $LFSR3$ and $LFSR4$ output bits. We will refer to those output bits as $LFSR3_n$ and $LFSR4_n$.

First, you initialize the set $\mathcal{L}$ to empty. Then, you perform the below depth-first search:

1. Call the state we are examining n. Compute the exclusive-or of the output n of $LFSR1$ and $LFSR2$, the next output of the blender FSM (based on the current state), and the known keystream bit Z_n. If our assumptions are correct to this point, this must be equal to the exclusive-or of the outputs of $LFSR3$ and $LFSR4$.
2. If the exclusive-or is zero, then we branch and consider the cases that both $LFSR3$ and $LFSR4$ output a zero here, and that they both output a one. When we assume a zero, we include in $\mathcal{L}$ the two linear equations $LFSR3_n = 0$ and $LFSR4_n = 0$, and when we assume a one, we include in $\mathcal{L}$ the two linear equations $LFSR3_n = 1$ and $LFSR4_n = 1$.
3. If the exclusive-or is one, then we include in $\mathcal{L}$ the single linear equation $LFSR3_n \neq LFSR4_n$
4. If $n \geq 33$, then we include in $\mathcal{L}$ the linear equation implied by the $LFSR3$ tap equations. If $n \geq 39$, then we include in $\mathcal{L}$ the linear equation implied by the $LFSR4$ tap equations. In both cases, we check to see if the new equations are inconsistent with the equations already in $\mathcal{L}$. If they are, then some assumption we made is incorrect and we backtrack to consider the next case.
5. Compute the next state of the blender FSM. This is always possible, as the next state depends on the current state (which we know) and the number of LFSRs that output a one, which we know.
6. If n is more than 132, then we have found with high probability the initial state of the encryption engine. If not, then we continue this search for state $n + 1$

There are two ideas behind this algorithm. The first is that the next state function for the blender FSM depends only on the number of LSFRs that output a one. So, when we assume that the outputs of LFSR3 and LFSR4 differ, we need not decide which one outputs a zero and which one outputs a one – instead, we can just note the fact that they differ and continue the search.

The other idea is that systems of linear equations in $GF(2)$ can be quite efficiently examined for contradictions.

How efficient is this attack? We provide some heuristic arguments. First, consider the case that all the assumed bits of LFSRs 1 and 2 and the blender state are correct.

With every step we learn if the sum S of the two output bits is either (a) $S \in \{0, 2\}$ or (b) $S = 1$. Both cases (a) and (b) are equally likely.

Note Prob$[S = 1] = 0.5$, and Prob$[S = 0] =$ Prob$[S = 2] = 0.25$. If $S = 1$, we learn one linear equation on the state bits of LFSRs 3 and 4 (namely the XOR of the two current output bits). If $S \in \{0, 2\}$, we branch and consider both

$S = 0$ and $S = 2$. Both $S = 0$ and $S = 2$ provide us with two linear equations on the state bits of LFSRs 3 and 4.

On the average, we expect to learn 1.5 linear equations and branch 0.5 times for each step. Once we have learned in total 33+39=72 equations, we are in a leaf of the branch tree and know or "have guessed" all bits in the system. The number of such leaves describes the amount of work. (Note that this analysis is based on the heuristic assumption that no equations are redundant or contradictory, or rather, that the effects of redundant and contradictory equations on the amount of work cancel out.)

So, our branch tree has an "average" size determined by $2^{72/3} = 2^{24}$ leaves. We initially assumed 60 bits and can expect to have made a correct assumption after trying 2^{59} times, which gives us a running time of $O(2^{59+24}) = O(2^{83})$ on the average.

Experiments demonstrate that our heuristic arguments on the efficiency of the attack are reasonable, though perhaps a bit optimistic. For a random incorrect guess of initial state, the procedure examines an average of approximately 60 million (2^{26}) states before terminating. Thus we can reconstruct the encryption engine state in

$$O(2^{85}) \text{ expected time.}$$

However, for both the first level and the second level keystream generator, we can take advantage of special conditions that allow us to further optimize the attack.

5 Attack on the Second Level E_0 Keystream Generator

To optimize the attack against the second level keystream generator (which produces the observed keystream directly), we note that the base attack is more efficient if the outputs of LFSR3 and LFSR4 exclusive-or'ed together happens to have a high hamming weight. To take advantage of this, we extend the attack by assuming that, at a specific point in the keystream, the next $n + 1$ bits of LFSR3 exclusive-or'ed with LFSR4 are n ones followed by a zero, where n will be less than the length of the LFSRs. Since LFSR outputs are effectively random and independent with such a length (since both LFSRs can generate any $n + 1$ bit pattern at any time with approximately equal probability if $n < 32$), the probability a $n + k$ length output contains such a sequence is approximately $k \cdot 2^{-n}$ (for $k \ll 2^n$).

If the assumption that the LFSRs produce such an output at the specific point in the keystream is false, we will fail to discover the internal state. However, the amount of work required to make that determination turns out to be rather less than $O(2^{85-n})$, and so if we have 2^n or more starting places to test out, we will find a place where the above procedure discovers the initial state with high probability.

The expected amount of time the base attack will take when we precondition the assumed outputs of LFSR3 and LFSR4 can be experimentally obtained. The results are given in Table 1, together with the expected time for the full search.

Table 1. The expected complexity and plaintext required for various values of n. Base Search Time is the expected number of nodes traversed in a single run of the base attack. Expected Plaintext Required is the expected amount of plaintext we need to prosecute the attack. Expected Search Time is the expected total search time taken.

n	Base Search Time	Expected Plaintext Required	Expected Search Time
5	$2^{24.8}$	165 bytes	$2^{83.8}$
10	$2^{23.5}$	1157 bytes	$2^{82.5}$
15	$2^{22.1}$	33k	$2^{81.1}$
20	$2^{20.5}$	1M	$2^{79.5}$
25	$2^{18.8}$	32M	$2^{77.8}$
30	$2^{17.1}$	1G	$2^{76.1}$

Looking through this table, we can see that modest amounts of keystream reduce the expected work somewhat, however, vast quantities of keystream reduce the expected work only slightly further.

Formally, the algorithm is:

1. Select a position in the known keystream that is the start of more than 132 consecutive known bits.
2. Cycle through all possible combinations of 4 bits of blender FSM state, 25 bits of LFSR1 state and the last $30 - n$ bits of LFSR2 state
3. Compute the initial $n + 1$ bits of LFSR2 state that is consistent with the exclusive-or of LFSR3 and LFSR4 consisting of n ones and then zero.
4. Run the base attack on that setting. Stop if it finds a consistant initial setting.

The above algorithm runs the base attack 2^{59-n} times and has a 2^{-n} probability of success for a single location.

Note that, even though a single packet has a payload with a maximum of 2745 bits, we can have considerably more than 2745 bits of known keystream, if we know the plaintext of multiple packets. All the next phase of the attack needs to know is the initial state of the second level keystream generator for a packet – it does not matter which. If we have multiple packets, we can try all of them, and we will be successful if we manage to find the initial state for any of them.

6 Another Attack on the Second Level Generator

Given a huge amount of known keystream, there is another technique to attack the second level keystream generator more efficiently. The basic attack requires to assume the blender state and the states of both LFSR1 and LFSR2 (i.e. $4 + 25 + 31$ bits $= 60$ bits). Now, we start with assuming only the blender and LFSR1 states (29 bits), at the beginning of the attack. During the course of the attack, we continue to make assumptions on how the blender state is updated.

Denote the sum of the outputs of LFSR2, LFSR3, and LFSR4 by S. Obviously, $S \in \{0, 1, 2, 3\}$. Since we always know (based on previous assumptions) the current blender and LFSR1 state, we only need to know S in order to compute the next blender state. The current output bit tells if S is odd or not. Thus, we know if either (a) S in $\{0, 2\}$ or (b) S in $\{1, 3\}$.

Both cases (a) and (b) are equally likely. And in both cases we learn one linear equation, namely we learn the XOR of the output bits of the LFSRs 2–4.

Now consider the conditional probabilities Prob[S=2|(a)] and Prob[S=1|(b)]. Assuming the three output bits are independent uniformly distributed random bits (which they are, approximately), we get

$$\text{Prob}[S = 2|(a)] = \text{Prob}[S = 1|(b)] = 0.75.$$

Instead of branching, as we did in the base attack, we simply assume the likely case $S \in \{1, 2\}$, ignoring $S = 0$ and $S = 3$.

We need $31 + 33 + 39 = 103$ linear equations to entirely restore the states of the LFSRs 2–4. The assumptions we get here are linearily independent. If both our initial assumptions on the 29 state bits of blender and LFSR1 and our 103 assumptions on the sum S are correct, we have found restored the correct state. We can check so by computing δ output bits (with $\delta > 29$) and comparing the output stream we get by our assumed E_0 state with the true output stream.

Within these 103 clocks the random variable S takes 103 values $S_1, S_2, \ldots \in \{0, 1, 2, 4\}$ with Prob[$S_i \in \{1, 2\} = 0.75$]. The attack works if $S_1 \in \{1, 2\}$ and $S_2 \in \{1, 2\}$ and …and $S_{103} \in \{1, 2\}$. Making the heuristic (but apparently plausible) argument that the S_i behave like 103 independent random variables, the probability $p = \text{Prob}[\,S_1 \in \{1, 2\}$ and … and $S_{103} \in \{1, 2\}\,]$ is

$$p = 0.75^{103} \approx 1.35 * 10^{-13} \approx 2^{-42.7}.$$

If the initially assumed 29 bits are correct, the attack requires less than 2^{43} bits of known keystream and less than 2^{43} steps (each step means to solve a system of 103 linear equations). Thus the entire attack needs

less than 2^{43} bits of known keystream

and

less than 2^{72} steps.

7 Attack on the First Level E_0 Keystream Generator

To attack the first level keystream generator (which produces the initial LFSR and blender FSM states), we first note that the key setup sets the FSM state of the second level keystream generator to be the final contents of the FSM state after the first level generator has produced the last bit for the LFSR state. We also note that the next-state function of the cipher is invertible – the LFSRs can be run backwards as easily as forwards, and the FSM next state function is invertible given a current LFSR state. We can also test the base attack, and find that it works essentially as well on the backwards cipher as it does the forward cipher.

This suggests this attack: when given one state of the level 2 generator, cycle through all possible combinations of 25 bits of LFSR1 state and 31 bits of LFSR2 state, and use the base attack on the reversed cipher, using as the initial FSM contents the initial contents of the phase 2 FSM. Because we are cycling through an expected $O(2^{55})$ LFSR states, and each check is expected to take $O(2^{26})$ time, we should expect to find the first level initial position in $O(2^{81})$ time.

8 Attack on the First Level E_0 Keystream Generator Given Two Second Level Keystreams

Now, let us consider a possible attack if the attacker has the first level output for two distinct packets that were sent with the same key. In this case, we first note that both keystreams have a clock associated with it, and that the clock is the only thing that differs. We further note that the method of combination is linear, hence if we know the xor differential in the clock (which we do, because we know the actual clock values), we know the xor differential of the first level LFSRs.

We can use this to optimize the attack further, as follows, where we will indicate the two known sides with as x_A and x_B, and where $\mathcal{L}$ is a set of linear equations on the outputs of $LFSR2_A$, $LFSR3_A$, $LFSR4_A$.

Assume the contents of $LFSR1_A$ (which also gives you $LFSR1_B$, because of the known differential between the two).
Initialize the set $\mathcal{L}$ to empty.
Perform the following depth-first search

1. Call the state we are examining n. Compute the output n_A, n_B of $LFSR1_A$, $LFSR1_B$, the previous output of the blender FSMs based on the current state), and the known keystream bit Z^n_A, Z^n_B. If our assumptions are correct to this point, this must be equal to the exclusive-or of the outputs of $LFSR2_A$, $LFSR3_A$, $LFSR4_A$ and of $LFSR2_B$, $LFSR3_B$, $LFSR4_B$.
2. Check the known differential in $LFSR2_A$, $LFSR3_A$, $LFSR4_A$, $LFSR2_B$, $LFSR3_B$, $LFSR4_B$ to see if there is a setting of those bits that satisifies both the known xors and the known differentials. If there is not, then backtrack to consider the next case.
3. If we reach here, there are four possible settings of the outputs of $LFSR2_A$, $LFSR3_A$, $LFSR4_A$ which are consistent with known xors and differentials. At least two of those settings will also update both blender FSMs identically, and will differ in precisely two bits. Here, we branch and consider three cases: one case that corresponds to the two settings which updates both blender FSMs identically, and the other two cases corresponding to the other two settings. For the first case, we include in $\mathcal{L}$

the linear equation implied by the two bits that differ, and the linear equation implied by the third bit setting. For the other two cases, we include in $\mathcal{L}$ three linear equations giving the three bit settings.

4. If $n \geq 31$, then we include in $\mathcal{L}$ the linear equation implied by the $LFSR2_A$ tap equations.
5. If $n \geq 33$, then we include in $\mathcal{L}$ the linear equation implied by the $LFSR3_A$ tap equations. If $n \geq 39$, then we include in $\mathcal{L}$ the linear equation implied by the $LFSR4_A$ tap equations. In all three cases, we check to see if the new equations are inconsistent with the equations already in $\mathcal{L}$. If they are, then some assumption we made is incorrect and we backtrack to consider the next case.
6. Compute the previous state of the blender FSMs. This is always possible, as the next state depends on the current state (which we know) and the number of LFSRs that output a one, which we know.
7. If n is more than 128, then we have found with high probability the initial states of the encryption engines. If not, then we continue this search for state $n + 1$

Experiments show that the above procedure examines an expected $O(2^{51})$ nodes during the search.

9 Attack Against Full E_0

Below is how we can combine these attacks into an attack on the full E_0 encryption system.

Assume we have an amount of known keystream generated with an unknown session key, which may be from a single packet or it may be from multiple packets. We select n based on the amount of known keystream. We can then use the attack shown in Section 5 to find the initial LFSR and blender FSM settings for a packet generated by that session key. If the cost of finding the initial LFSR and blender FSM settings for a second packet is less than $O(2^{81})$, then we find a second one. Then, we either use the attack shown in Section 7 to find all possible initial LFSR settings that generated that initial setting (if we have one initial LFSR setting), or we use the attack shown in Section 8 if we have two initial LFSR settings. Once we find the initial LFSR settings that generates the observed output, we can step the LFSRs back 200 cycles, and use linear transformations to eliminate the Bluetooth address and the block to reconstruct the session key K'_C, and verify that potential key by using to to decrypt other packets.

If we denote the amount of effort to find a LFSR and blender setting given n bytes of known keystream as $F(n)$ (see table 1), then the total effort for this attack is

$$O(min(F(n) + 2^{81}, 2F(n/2) + 2^{51}))\,\text{work}.$$

This is $O(2^{84})$ if you have barely enough keystream to uniquely identify the session key (eg., 140 bits), and drops to $O(2^{77})$ if you have a gigabit of known keystream.

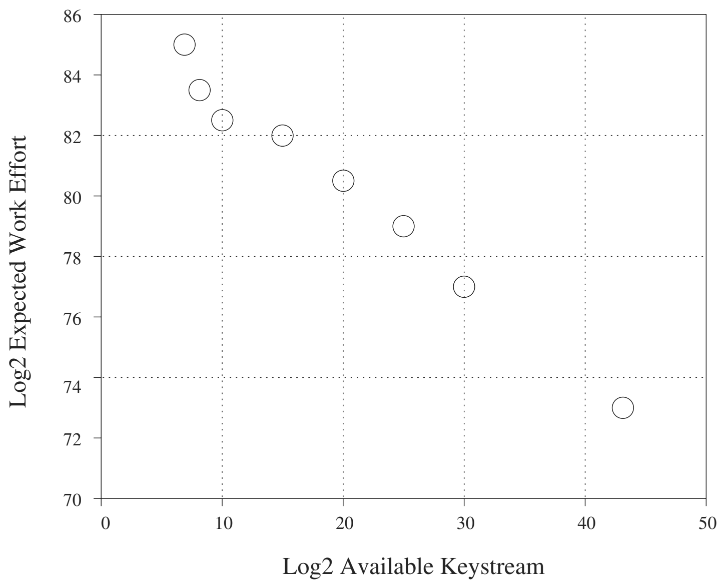

Fig. 2. Expected work effort required to recover session key, versus known keystream.

We can further reduce the effort down to

$$O(2^{73})\,\text{work},$$

if about 14 000 gigabit bits of keystream are available. We simply use the attack from Section 6 twice, to recover two states of the level 2 generator, and then continue with the attack from Section 8.

These results are summarized in Figure 2.

10 Conclusions and Open Problems

We described methods for rederiving the session key for E_0 given a limited amount of known keystream. This session key will allow the attacker to decrypt all messages in that session. We showed that the real security level of E_0 is no more than 73–84 bits (depending the amount of keystream available to the attacker), and that larger key lengths suggested by the Bluetooth specification[2] would not provide additional security.

[2] "For the encryption algorithm, the key size may vary between 1 and 16 octets (8-128 bits). The size of the encryption key shall be configurable for two reasons. [First is export provisions]. The second reason is to facilitate a future upgrade path for the security without a costly redesign of the algorithms and the encryption hardware; increasing the effective key size is the simplest way to combat increased computing power at the opponent side. Currently (1999) it seems that an encryption key size of 64 bits gives satisfying protection for most applications." [1, Section 14, page 148]

We empicically observed that the technique from Section 6 (assume the blender state and LFSR1 only, and build up a set of equations based on the states of LFSR2, LFSR3 and LFSR4) posed some practical problems, because the equations created are rather complex. Also, the technique requires a huge amount of known keystream. It would be interesting to develop improved techniques to handle the set of linear equations more efficiently. Also, it would be interesting to reduce the required amount of known keystream.

Another approach for more practical attacks on E_0 and Bluetooth would be to exploit the weak mixing of the clock into the first level LFSRs, which will, at attacker known times, leave three of the LFSRs with zero differential.

References

1. Bluetooth SIG, “Bluetooth Specification”, Version 1.0 B, `http://www.bluetooth.com/`
2. P. Ekdahl, T. Johansson, “Some Results on Correlations in the Bluetooth Stream Cipher”, Proceedings of the 10th Joint Conference on Communications and Coding, Obertauern, Austria, March 11-18, 2000.
3. J. Golic, Eurocrypt 1997.
4. M. Hermelin, K. Nyberg, “Correlation Properties of the Bluetooth Combiner”, proceedings of ICISC ’99, LNCS 1787, Springer, 1999.
5. M. Jakobsson, S. Wetzel, “Security Weaknesses in Bluetooth”, RSA Conference 2001.
6. M. Saarinen, “Re: Bluetooth und E0”, Posting to `sci.crypt.research`, 02/09/00.
7. K. Zeng, C.-H. Yang, T. Rao “On the Linear Consistency Test (LCT) in Cryptanalysis with Applications”, Crypto ’89, Springer LNCS 435, pp. 164–174.