Improved key recovery of level 1 of the Bluetooth Encryption System

Scott R. Fluhrer[1]

Cisco Systems, Inc.
170 West Tasman Drive, San Jose, CA 95134
sfluhrer@cisco.com

Abstract. The encryption system E_0 , which is the encryption system used in the Bluetooth specification, is a two level system where a key and a packet nonce is given to a level 1 key stream generator, which produces the key for a level 2 key stream generator, whose output is used to encrypt. We give a method for recovering the key for the level 1 key stream generator given the internal keys for two or three level 2 key stream generators. This method, combined with published methods for recovering keys for the level 2 key stream generator, can be used to recover the E_0 second key with $O(2^{65})$ work, and $O(2^{80})$ precomputation time.
Although this attack is of no advantage if E_0 is used with the recommended security parameters (64 bit encryption key), it shows that no addition security would be made available by enlarging the encryption key, as discussed in the Bluetooth specification.

1 Introduction

E_0 uses a two level rekeying mechanism, using the key and a nonce (CLK_{26-1} in the Bluetooth specification) to initialize 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 give an algorithm for deriving the initial state of the keystream generator used within E_0 given the intermediate keystream generated by the same key, and two or three distinct nonces. This algorithm takes several minutes on a desktop computer to rederive the key when given three internal keystreams. With two distinct nonces, it takes approximately $O(2^{50})$ time. Combined with existing methods for reconstruction the internal level 2 state[4], this leads to a method for reconstruction the secret key with $O(2^{65})$ work, two maximum sized packet, and $O(2^{80})$ precomputation effort[1].

This method is an improved version of one of the algorithms given in [3]. In addition, experimental results showing the workability of the algorithm is given.

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 the fundamentals

[1] As the method in [4] require precomputation. The method described in this paper require no precomputation.

of this attack, and 5 gives the actual algorithm. Section 6 gives experimental results, and section 7 concludes and discusses how it can be combined with previous results, and the ramifications on the Bluetooth system.

2 Description of Bluetooth encryption

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)[2]. Then, K'_C is used in a linear manner, along with the publicly known values, the Bluetooth address, and a clock (CLK_{26-1}) 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, referred to as the blender FSM. For each bit output, each LFSR is clocked once, and the output of all four LFSRs and an output from the finite state machine is exclusive-or'ed together, and is the keystream output. Then, the 4 LFSR outputs are summed together to form a 3 bit output. The upper 2 bits of that sum is 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.

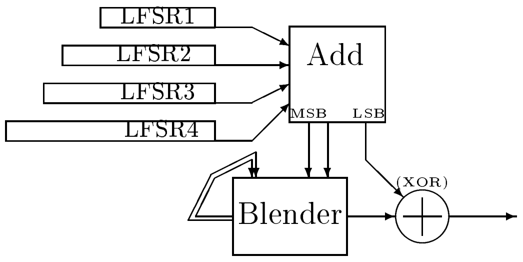

Fig. 1. The E_0 keystream generator.

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

[2] This attack actually recovers 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 the most effective attack to date, Golić, Bagini and Morgani show in [4] how to recover K'_C with $O(2^{70})$ work, along with a precomputation stage of complexity $O(2^{80})$. In particular, they show 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. This paper uses this observation, and uses two or three such level 2 internal states to reconstruct the level 1 internal state with comparatively little additional effort.

Fluhrer and Lucks have shown in [3] how to recover K'_C with work of between $O(2^{73})$ and $O(2^{84})$ work, depending on the amount of available keystream. This works by doing an optimized backtracking algorithm.

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. His attack has a complexity of $O(2^{93})$.

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 [5] 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.

4 Fundamentals of the Attack

The problem we are trying to solve is, given two or three initial states that we produced by the same secret key for the level 2 LFSRs (and the nonces that generated them), to reconstruct the secret key $K'C$ (or equivalently, the initial state of the level 1 LFSRs).

The basic approach is to use a backtracking approach, similar to the attacks found in [3]. One distinction between the approach found in [3] and this is that we don't assume LFSR1 – we backtrack across all 4 LFSRs simultaneously. The procedure steps through the output of the level 1 keystream generators bit by bit, keeping track of the state of the blender FSM and the known linear equations that hold between the internal bits of the LFSRs. This search can be aided by several observations:

1. The key setup sets the FSM state of the level 2 keystream generator to be the final contents of the FSM state after the level 1 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 invertable given a current LFSR state. This implies that if we logically run the cipher backwards, we get the initial FSM state (which is the final FSM state when the cipher is run in the forward direction) at zero cost.
2. Every step of the way, we need to assume the FSM states for the next[3] state for all level 1 ciphers we are tracking. We note that often there are level LFSR outputs that give rise to the same set of FSM states, and that these LFSR outputs can often be expressed in terms of a linear relationship. Hence, instead of assuming the actual LFSR states (which would give us a factor of 8 branching at every point), we often have a factor 4 or 5 branching.
3. The nonce, public information and the private information are mixed linearly to form the level 1 LFSR state. Hence, if we know the xor differential in all the inputs (which we do, as we know the actual nonce values, and we know everything else has zero differential), we know the xor differential of the first level LFSRs at all times.
4. Everytime we step to another state, the next[4] output bit is formed by the xor of the 4 LFSRs and an output from the FSM:

$$Output = FSM \oplus LFSR_1 \oplus LFSR_2 \oplus LFSR_3 \oplus LFSR_4 \qquad (1)$$

(where $Output$ is the output bit, FSM is the output from the FSM, and $LFSR_x$ is the output of LFSR x. Since we are tracking two or three level 1 LFSRs and FSM states at the same time, we know the FSM outputs and output bits for all of them:

$$Out\hat{p}ut = F\hat{S}M \oplus LF\hat{S}R_1 \oplus LF\hat{S}R_2 \oplus LF\hat{S}R_3 \oplus LF\hat{S}R_4 \qquad (2)$$

If our guesses up to this point are correct, we know that values of $Output$, FSM, $Out\hat{p}ut$, $F\hat{S}M$. In addition, because of the linear LFSR mixing, we know the values of $LFSR_x \oplus LF\hat{S}R_x$. This gives us enough information to check the above equation for consistency, and if are guesses are inaccurate, the above equations will be inconsistent with probability $\frac{1}{2}$ with two level 2 states and probability $\frac{3}{4}$ with three level 2 states.

[3] Actually, the previous state, because we are stepping through the cipher state backwards.

[4] Actually, the previous output bit, because we are stepping through the cipher state backwards.

5 Attack Details

When you combine all the above, we come up with the following algorithm, with the following inputs:

1. An integer N which is the number of level 2 states we have. We examine the $N = 2$ and $N = 3$ conditions in this paper.
2. $Z, \dots Z'$ N different initial states for the level 2 LFSRs. This is also the keystream generated by the level 1 keystream generators.
3. The N FSM initial states for the level 2 keystream generators. This is also the final FSM state of the level 1 keystream generators.
4. The $N - 1$ differentials between the level 1 LFSRs $LFSRDiff_x$.

First, initialize the set of linear equations $\mathcal{L}$ to empty, and initialize the blender FSMs to the initial FSM values. Then, you perform the below depth-first search:

1. Call the state we are examining n and the output of the FSMs $FSM_n, \dots FSM'_n$.
2. Determine whether the state is self consistent by computing (taking the keystream states pairwise):

   $$Z_n \oplus Z'_n \oplus FSM_n \oplus FSM'_n \oplus LFSRDiff_1 \oplus LFSRDiff_2 \oplus LFSRDiff_3 \oplus LFSRDiff_4 \qquad (3)$$

   If the equation evaluates to 1 for any pair, the state is not self consistent and some assumption we made is incorrect and we backtrack to consider the next case.
3. For each of the 8 possible LFSR outputs for which

   $$Z_n = FSM_n \oplus LFSR_1 \oplus LFSR_2 \oplus LFSR_3 \oplus LFSR_4 \qquad (4)$$

   compute the predicted previous FSM states. Search for pairs that share FSM states over all N FSMs, and which differ in precisely two LFSRs. We branch and consider both the disjoint pairs, and singleton states which do not cooperate with any pairs. We include in $\mathcal{L}$ the linear equations that characterize the pair or the singleton. We check to see if the new equations are inconsistent with the equations in $\mathcal{L}$, and if they are, then some assumption we made is incorrect and we backtrack to consider the next case.
4. If $n \geq 25$, then we include in $\mathcal{L}$ the linear equation implied by the $LFSR1$ tap equations. If $n \geq 31$, then we include in $\mathcal{L}$ the linear equation implied by the $LFSR2$ tap equations. 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 all 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. If n is more than 128, then we have found with high probability the initial state of the encryption engine. If not, then we continue this search for state $n + 1$

6 Expiremental Results

We have implemented this attack for both $N = 2$ and $N = 3$.

In the case that $N = 3$, we have run it for 100 random secret keys and triples of nonces. Over those 100 trials, it found the secret key after examining an average of $10700000 \approx 2^{23.4}$ states, taking several minutes per secret key.

In the case that $N = 2$, the tree was too large to do an exhaustive search. By sampling a random subset of the search tree, we find that the tree has approximately 2^{50} nodes. We note that it takes somewhat more time to examine a node than a single step of Golić's algorithm [4], however because Golić's algorithm has $O(2^64)$ steps, it still bounds the running time of the combined attack.

7 Conclusions and Open Problems

The method presented can be combined with Golić's algorithm [4] in the following manner:

1. Use Golić's algorithm on two packets to recover the initial LFSR level 2 states (and initial FSM states).
2. Use the known nonces for those two packets to compute the xor differential for the level 1 LFSRs.
3. Use the initial LFSR level 2 states, initial FSM states and the xor differential with the above algorithm to recover the initial LFSR level 1 state.
4. Recover K'_C from the LFSR level 1 state.

Thus, the real security level of E_0 to be no more than 65 bits (depending the amount of keystream available to the attacker), and that larger key lengths suggested by the Bluetooth specification would not provide additional security.

In addition, because of the efficiency of the $N = 3$ case, we have shown that any method of recovering LFSR level 2 state can be used to break the system, as long as it can be done to three different packets. Because of this efficiency, we see that the two level structure of E_0 does not add additional security.

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. S. Fluhrer, S. Lucks, "Analysis of the E_0 Encryption System", Selected Areas in Cryptography 2001, Springer, 2001
4. J. Golić, V. Bagini, G. Morgani, "Linear Cryptanalysis of Bluetooth Stream Cipher", Proceedings of Eurocrypt 2002, Springer, 2002
5. M. Hermelin, K. Nyberg, "Correlation Properties of the Bluetooth Combiner", proceedings of ICISC '99, LNCS 1787, Springer, 1999
6. M. Saarinen, "Re: Bluetooth und E0", Posting to sci.crypt.research, 02/09/00