Zero-knowledge-like Proof of Cryptanalysis of Bluetooth Encryption

Eric Filiol*
Ecole Supérieure et d'Application des Transmissions
Laboratoire de virologie et de cryptologie
B.P. 18, 35998 Rennes Armées, FRANCE
eric.filiol@esat.terre.defense.gouv.fr

September 3, 2006

Abstract

This paper presents a protocol aiming at proving that an encryption system contains structural weaknesses without disclosing any information on those weaknesses. A verifier can check in a polynomial time that a given property of the cipher system output has been effectively realized. This property has been chosen by the prover in such a way that it cannot been achieved by known attacks or exhaustive search but only if the prover indeed knows some unknown weaknesses that may effectively endanger the cryptosystem security. This protocol has been denoted *zero-knowledge-like proof of cryptanalysis*. In this paper, we apply this protocol to the Bluetooth core encryption algorithm E0, used in many mobile environments and thus we prove that its security can seriously be put into question.

Keywords: Bluetooth encryption, Bluetooth security, Bluetooth protocol, stream cipher, zero-knowledge, cryptanalysis.

1 Introduction

Encryption is the most important part in computer security mechanisms and protocols: he who can bypass cryptographic protection, gains total control over the security. Password management, secure network transmission protocol, wireless protocol (Wep, WPA, Bluetooth, GSM...), integrity checking

*Also *Eric.Filiol@inria.fr*

(e.g. in antivirus software), data protection, login authentication... are well-known examples whose security heavily relies on cryptographic mechanisms.

As far as viral hazard is concerned, maintaining a high level of security is essential. Anyone who could break cryptographic mechanisms would be not only able to subvert some detection capabilities (file integrity checking) but also to make malware spread far more easily. One well illustrative example, among many others, is the Bluetooth protocol used for wireless communications between mobile devices: laptops, PDA, cell phones, printers, cars... Since July 2004, more than a hundred viral codes spreading via mobile environments and Bluetooth devices in particular have been recorded. The most famous one is the *Epoc.Cabir* virus [3]. Fortunately, these viral codes have a very limited effect. One of the main reason for this, is that Bluetooth protocol embeds cryptographic multi-level security. Cryptology ensures a high level of security for both encryption and authentication. This implies that nobody can remotely and unlegitimately either connect to or log in to Bluetooth devices and proceed to its infection. With the same considerations in mind, encryption of data stream between two or more communicating devices prevents anyone from manipulating or corrupting it (e.g. insertion of malicious code). The encryption cryptographic core uses the stream cipher E0, whose key entropy is 128 bits. The key length thus prevents any cryptanalysis by exhaustive search. Moreover, up to now, the encryption security of E0 has not been challenged from a practical point of view. A few attacks of theoretical interest only have been published [1, 4, 2, 6, 7, 10, 11, 13, 14, 15]. Unless irrealistic assumptions are to be made, E0 has not been broken yet and the cryptographic security of Bluetooth protocol is still very high.

However, publishing any potentially efficient cryptanalytic techniques that may practically put cryptographic security into question is an essential question. While it is important to make engineers, vendors and users aware of a real risk, there are maybe more important reasons not to technically explain what the level of risk really is:

- disclosing any technical, reproducible and usable data provides information that the "bad guys" will use to perform attacks. As far as embedded encryption is concerned (WEP, Bluetooth, GSM...), changing the core encryption algorithm is very costly and takes too much time. Months or even years are generally required before all weak devices are replaced. The vendors try to make their technical investments financially profitable enough before changing the whole cryptographic standard while users are generally reluctant to change their equipment. During this transition period, an important number of attacks may be performed.

- disclosing any technical information that allow to bypass or hurt computer security is generally prosecuted in many countries (one example among others is the French *Law for Confidence in the e-Economy* [1] [12]). Moreover, such disclosing can be prosecuted as copyright infringement as well. The best known-example is undoubtly the US *Digital Millenium Copyright Act* [20]. But retaining any information about any system vulnerability, a software flaw or any weakness and communicating it only to the developers may incitate them not to react for commercial purposes.

The question actually is the following one: how to prove in a uncriticizable way but without disclosing any useful, reproducible technical data that a cryptographic system can, may or might be broken in practise?

In this paper, we will consider the case of the Bluetooth encryption system (E0) and explain how to solve this problem. Recent and significant progress has been made in the cryptanalysis of symmetric ciphers. We detected and identified serious cryptographic weaknesses in E0 that could be used to break it in practice in a near future, especially to perform viral attacks through the Bluetooth protocol. This result is proved without giving any clue about the weaknesses and the way they can be exploited. Nonetheless, any reader with basic cryptographic knowledge will be able to be convinced that it is possible through a simple polynomial time verification. We call this method of proof *Zero-knowledge-like proof of cryptanalysis* (0-Kl proof for short).

The paper is organized as follows. Section 2 recalls the required background and notation in cryptography. Section 3 recalls how the Bluetooth encryption works. Section 4 then presents the Zero-knowledge-like proof of cryptanalysis itself. Section 6 then deals with future work and draws a conclusion. The E0 reference implementation is given in Appendix A. Appendix B contains detailed numerical values. Both appendices are essential to make Zero-knowledge-like proof of cryptanalysis work.

2 Background and Notation

First of all, we are going to define what *Zero-knowledge* is generally referred to. Zero-knowledge is a property attributed to interactive proofs, in which a

[1] According to Article 323-3-1 of the Penal Code, it is punishable by imprisonment not exceeding three years and a fine of up to 45,000 euros.

prover convinces a *verifier* of the validity of a given statement. The prover has no particular restriction whereas the verifier is restricted to use a (probabilistic) polynomial time algorithm. By Zero-knowledge proof, we mean that the verifier is convinced without the prover releasing any knowledge beyond the validity of the statement. This concept has been introduced for the first time by Goldwasser and al. [9]. The reader will find a detailed overview of zero-knowledge proofs in [8].

For the non cryptologist reader's sake, let us now recall the definition of a *stream cipher*. A stream cipher is a symmetric cipher – in other words, the same secret key is employed for both the encryption and the decryption – which operates in the following way: a sequence of plaintext bits $m_0, m_1, \ldots, m_i, \ldots$ is encrypted into a sequence of ciphertext bits $c_0, c_1, \ldots, c_i, \ldots$ by means of a pseudo-random sequence $s_0, s_1, \ldots, s_i, \ldots$ called the *keystream*. The most common way of encryption is given by

$$c_i = m_i \oplus s_i$$

The keystream is produced by a finite state automaton whose initial state is precisely the secret key shared by both emitter and receiver.

Like for any other encryption algorithm, performing an exhaustive search cryptanalysis consists to trying every possible key until the right key that had been used to produce a given keystream has been detected. In the context of the paper, we try to find secret keys that produce keystreams having some given properties. Up to now, no method except exhaustive search is known to achieve that property for a keystream.

Let us state more clearly the total amount of work required by an exhaustive search. Let us consider a secret key K of n bits and let us consider a property $\mathcal{P}$ for the corresponding (output) keystream of length n. Thus, finding a key which produces a keystream having property $\mathcal{P}$ requires $n^{\mathcal{O}(1)}.\mathcal{O}(2^{n-m})$ operations where 2^m represents the total number of keys for which $\mathcal{P}$ is satisfied. Let us notice that in classical exhaustive search $m = 0$ (there is only one key producing a fixed keystream).

A *random search* consists in trying at random a large enough number of keys until obtaining a given fixed property or, equivalently, keeping the keys that output sequences with non-trivial properties.

The *Hamming weight* of a sequence $(s_t)_{0 \leq t \leq n}$, denoted $wt((s_t)_{0 \leq t \leq n})$ is the number of non-zero bits:

$$wt((s_t)_{0 \leq t \leq n}) = \{0 \leq t \leq n | s_t = 1\}.$$

3 The Bluetooth Encryption

The Bluetooth security mechanisms are presented in part H of Volume 2 of [19]. In the Bluetooth standard, the security layer is one of the baseband layers (hardware level), which the upper layers control (host and application levels). The security mechanisms include key management, as well as key generation protocols, user/device authentication, and data encryption. The data encryption algorithm used within the Bluetooth security architecture is the E0 stream cipher.

Each time two Bluetooth devices need to communicate securely, they first undergo authentication and key exchange protocols whose purpose is to agree on a shared secret (the *link key*), which is used to generate the encryption key (K_C). This latter key is derived from the current link key, an encryption 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 cipher a payload packet, the private key K_C is modified into another key denoted K'_C. Then K'_C is used in a linear manner, along with the publicly known values, the master device Bluetooth address (MAC address), and a clock value, which is different for each payload packet, to form the initial state for a two-level stream cipher as depicted in Figure 1. The encryption algorithm E0 generates a binary keystream, K_{cipher}, which is bitwise xored with the plain text. Decryption is performed in exactly the same way using the same key as used for encryption (xor addition being involutive). Any real-life cryptanalysis of E0 will greatly challenge the overal security. Besides the fact that it would be then possible to manipulate the encrypted data stream between devices that communicate (insertion of malicious code for example), coupled with recent efficient attack of the Bluetooth authentication and key negotiation protocol [18], the ability of retrieving the secret encryption key could make other attacks on overal cryptographic security easier.

3.1 The E0 Stream Cipher

Let us now consider the encryption core denoted E0.

E0 stream cipher uses linear feedback shift registers (LFSRs) whose output is combined by a simple finite state machine (called the summation combiner) with 16 memory states. The output of this state machine is the keystream sequence, or, during initialization phase, the randomized initial start value. The algorithm uses an encryption key K_C, a 48-bit

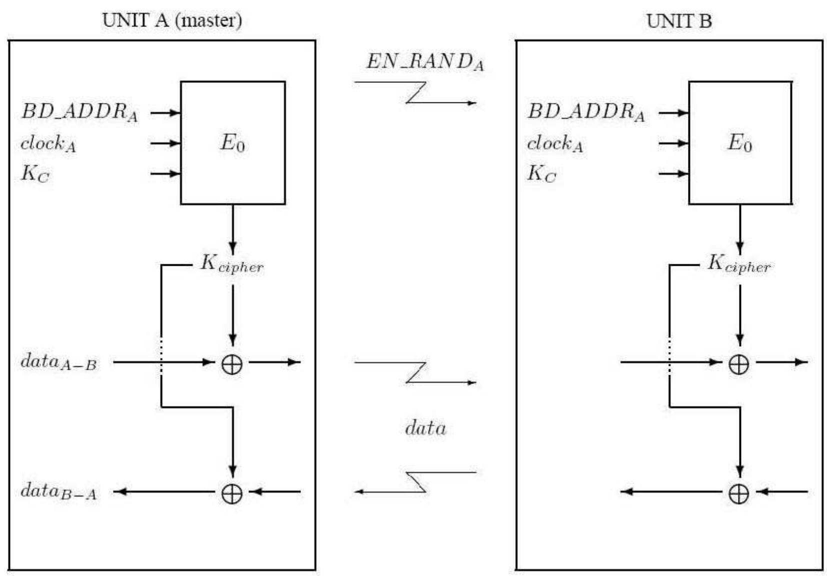

Figure 1: Functional description of the Encryption Procedure

Bluetooth address, the master clock bits CLK_{26-1}, and a 128-bit RAND value. Figure 2 shows the encryption engine setup. There are four LFSRs ($LFSR_1, \ldots, LFSR_4$) of lengths $L_1 = 25, L_2 = 31, L_3 = 33$ and $L_4 = 39$ with feedback polynomials as specified in Table 1. The total length of the registers is 128. These primitive polynomials have been chosen as they exhibit the best trade-off between hardware implementation constraints and excellent statistical properties of the output sequences. Let x_t^i denote the

i	L_i	Feedback polynomials
1	25	$x^{25} \oplus x^{20} \oplus x^{12} \oplus x^8 \oplus 1$
2	31	$x^{31} \oplus x^{24} \oplus x^{16} \oplus x^{12} \oplus 1$
3	33	$x^{33} \oplus x^{28} \oplus x^{24} \oplus x^4 \oplus 1$
4	39	$x^{39} \oplus x^{36} \oplus x^{28} \oplus x^4 \oplus 1$

Table 1: The Four Primitive Feedback Polynomials

t-th symbol of $LFSR_i$. The value y_t is derived from the 4-tuple $x_t^1, x_t^2, x_t^3, x_t^4$

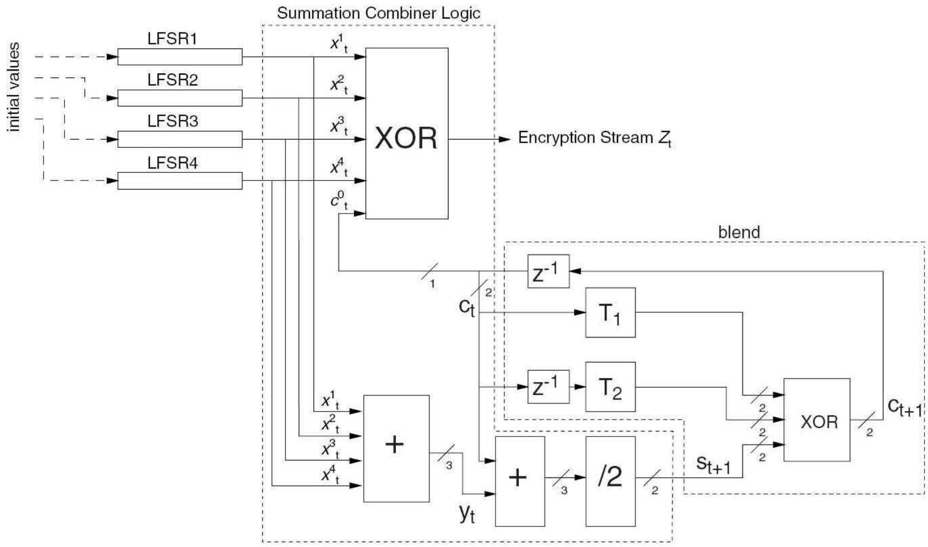

Figure 2: Functional description of the Encryption Procedure

using the following equation:

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

where the sum is over the integers. Thus y_t can take the values 0, 1, 2, 3 or 4. The output of the summation generator is obtained by the following equations:

$$
\begin{aligned}
z_t &= x_t^1 \oplus x_t^2 \oplus x_t^3 \oplus x_t^4 \oplus c_t^0 && \in \{0,1\}, \\
s_{t+1} &= (s_{t+1}^1, s_{t+1}^0) = \lceil \frac{y_t + c_t}{2} \rceil && \in \{0,1,2,3\}, \\
c_{t+1} &= (c_{t+1}^1, c_{t+1}^0) = s_{t+1} \oplus T_1[c_t] \oplus T_2[c_{t-1}],
\end{aligned}
$$

where $T_1[.]$ and $T_2[.]$ are two different linear bijections over $GF(4)$ summarized in Table 2. The E0 algorithm needs to be initialized with an initial value for the four LFSRs (the secret key K'_C) and the four bits that specify the values of c_{-1} and c_0. The key K'_C and the 4-bit value are produced by an initialisation step involving E0 and the secret key K_C, a 48-bit Bluetooth address, the master clock bits CLK_{26-1}, and a 128-bit RAND value.

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

Table 2: E0 Bijective Mappings

3.2 E0 Cryptanalysis State-of-the-Art

Stream cipher E0 is so far considered as a secure encryption algorithm. In particular, it has very good statistical properties and complies to the NIST statistical test suite [17]. No significant bias has been detected with respect to the tests of this suite.

A number of E0 cryptanalysis have been proposed so far. They can be divided into two sets, according to the number of frames required for the cryptanalysis to work :

- *Long keystream attacks.*- These attacks consider the Bluetooth encryption outside its real-life mode of operation. They require a too long keystream to actually challenge E0 security. They are for most of them correlation or fast correlation attacks, that is to say that they exploit some correlation between the outputs of the LFSR and the output sequence itself. Two attacks consider [1, 4] linearization of non linear equations whose unknowns are secret key bits. Table 3 summarizes the required amount of keystream bits (data) and the complexity (precomputation, time and memory) for each of those attacks.

Attacks	Data	Precomp.	Attack complexity	Memory
Fluhrer - Lucks [6]	2^{43}	-	2^{73}	2^{51}
Fluhrer [7]	$2^{12.4}$	2^{80}	2^{65}	2^{80}
Golic & al. [10]	2^{17}	2^{80}	2^{70}	2^{80}
Armknecht - Krause [1]	2^{24}	-	2^{68}	2^{48}
Courtois [4]	2^{24}	-	2^{49}	2^{37}
Lu & Vaudenay [14]	$2^{39.6}$	-	2^{40}	2^{35}
Lu & al. [15]	$2^{28.4}$	2^{38}	2^{38}	2^{33}

Table 3: Complexity Comparison for Best Long Keystream Attacks on E0

- *Short keystream attacks.*- These methods consider a very short known keystream (128 bits). Despite their still high complexity, these attacks are far more realistic than long keystream attacks. So far only a few such short key cryptanalysis are known: use of Binary Decision Diagrams [11] or backtracking methods [13] have opened a promising field of cryptanalytic research. Table 4 compares complexity of known short keystream attacks.

Attacks	Known keystream bits	Attack complexity
Bleichenbacher [2]	128	2^{100}
Krause [11]	128	2^{81}
Levy - Wool [13]	128	2^{86}

Table 4: Complexity Comparison for Best Short Keystream Attacks on E0

4 Zero-knowledge-like Proof of Cryptanalysis

The attacks presented in the previous section all require irrealistic assumptions to work in practice such as a huge amount of known plaintext bits and/or a dramatically high computing complexity. Let us consider now sequences of known plaintext of length n. The core idea of the zero-knowledge-like proof of cryptanalysis is to consider a mathematical property that cannot be achieved in real-life, unless to effectively knowing one or more weaknesses. The following definition will help us to make it clearer.

Definition 1 *(Zero-knowledge-like proof of cryptanalysis) Let be a cryptosystem S_K and a property $\mathcal{P}$ about the output sequence of length n produced by S denoted σ_K^n. No known method other than exhaustive search or random search can obtain property $\mathcal{P}$ for σ_K^n. Then, a* zero-knowledge-like proof of cryptanalysis *of S consists in exhibiting secret keys $K_1, K_2, \ldots, \ldots K_m$ such that the output sequences $(\sigma_{K_i}^n)_{1 \leq i \leq m}$ verify $\mathcal{P}$ and such that, checking it requires polynomial time complexity. Moreover, the property $\mathcal{P}$ does not give any information on the way it was obtained.*

The protocol proposed in Definition 1 is not a true zero-knowledge protocol (hence the term *zero-knowledge-like*) for the following reasons:

1. a paper is not an interactive medium;

2. the author of the cryptanalysis plays the role of the prover and answers questions which have not been asked by the verifier, e.g. the reader.

Another point worth considering is that the reader/verifier can bring up against the author/prover that some random keys has been taken, the keystream has been computed and afterwards been claimed that the keystreams properties have been desired. In other words, the author/prover tries to fool the verifier/reader by using exhaustive search to produce the properties that have been considered for the zero-knowledge-like proof protocol. Thus the relevant properties must be carefully chosen such that:

- the probability to obtain them by random search over the key space makes such a search untractable. This point is treated in Section 4.1, 4.2 and 4.3. In the contrary the verifier/reader would be able to himself exhibit secret keys producing keystream having the same properties by a simple exhaustive search;
- the known attacks cannot be applied to retrieve secret keys from a fixed keystream having the properties considered by the author/prover.
- to really convince the reader/verifier, a large number of secret keys must be produced by the author/prover, showing that "he was not lucky".

Since there do not exist any known method other than exhaustive search or random search to produce output sequences σ_K^n having property $\mathcal{P}$, and since the complexity of a successful search is too high in practice, anybody who is effectively able to exhibit a secret K producing such output sequences obviously has found some unknown weaknesses he used to obtain this result. The probability of realizing property $\mathcal{P}$ through an exhaustive search gives directly the upper bound complexity of the zero-knowledge-like proved cryptanalysis.

The last point to weigh up is to determine whether the fact knowing some flaw in a cryptographic design implies that it is possible to break it. The academic approach generally considers the following established cryptanalytic models:

- either an attacker knows some keystream bits and wants to recover the secret key,
- or the attacker wants to efficiently distinguish a keystream produced by a particular keystream generator from a truly random keystream[2].

[2]However, it remains an open problem which consists in proving that having a efficient

Well in the context of the present paper, the model we consider is not so far from the first case as long as the properties we considered are effectively not reproducible by another approach than a true cryptanalytic one: we fix some *a priori* keystreams exhibiting given properties and we must retrieve the corresponding secret key for each of them. In other words:

- in the classical case, a cryptanalyst aims at guessing $K = E_0^{-1}(\sigma_K)$ for some *a priori* fixed output sequence σ_K;
- in our case, we consider a subset $\mathcal{S}^{\mathcal{P}}$ of output sequence having a *a priori* fixed property $\mathcal{P}$, and we aim at recovering $K_{\mathcal{S}^{\mathcal{P}}} = E_0^{-1}(\mathcal{S}^{\mathcal{P}})$.

We will consider in the rest of the paper output sequence of length $n = 128$. This particular value is based on two reasons:

- this length value is more realistic when considering real use of E0 encryption in Bluetooth communication protocol (see Section 3);
- choosing a short sequence value clearly reinforces the level of attack efficiency and thus the 0-Kl proof of cryptanalysis.

Moreover, since E0 stream cipher exhibits all cryptographic properties that any strong cryptosystems fullfil and provided that secret keys are random variables over $\mathbb{F}_2^{128}$, any output sequence σ_K^{128} is a random variable as well which has uniform distribution over $\mathbb{F}_2^{128}$. Let us now consider two properties for our purpose of 0-Kl proof of cryptanalysis.

4.1 The Hamming Weight Property

We will first try to find secret keys K such that $\sigma_K^{128} = S_K$ has Hamming weight at most equal to some value k. This sequence will be denoted $\sigma_K^{128,k+}$. In particular, we will focus on small values for k since the sparsity of sequence is a property that is difficult to achieve. The same approach could consider in the same way sequence of weight at least equal to k for large value of k. To state things more clearly, the probability to obtain a sequence $\sigma_K^{128,k+}$ is given by Formula (1).

$$P[\sigma_K^{128,k+}] = \frac{1}{2^{128}} \times \left(\sum_{i=0}^{k} \binom{128}{i} \right) = p_{k+} \tag{1}$$

distinguiser at one's disposal is equivalent to effectively be able to break the relevant cryptosystem. The distinguisher issue relates more to a steganographic (transmission security or equivalently the security of the transmission channel itself) issue than to pure communication security issues.

This result is obvious when considering simple combinatorial properties.

In the same way, we may consider output sequences of length $n = 128$ that Hamming exactly equal to k. These sequences will be denoted $\sigma_K^{128,k}$. Then the probability of such a sequence is given by

$$P[\sigma_K^{128,k}] = \frac{1}{2^{128}} \times \binom{128}{k} = p_k.$$

Now considering the Hamming weight property implies that if we want to find a secret key K that outputs a sequence $\sigma_K^{128,k+}$ (respectively a sequence $\sigma_K^{128,k}$) no other method than a exhaustive search or random search of complexity $\mathcal{C}_{k+} = \frac{1}{p_{k+}}$ (resp. $\mathcal{C}_k = \frac{1}{p_k}$) is known unless using some undisclosed weaknesses. Table 5 gives numerical values for different values of k. These

k	$\mathcal{C}_{k+}$	$\mathcal{C}_k$	k	$\mathcal{C}_{k+}$	$\mathcal{C}_k$
30	$2^{30.52}$	$2^{31.04}$	22	$2^{46.36}$	$2^{46.69}$
29	$2^{32.27}$	$2^{32.76}$	21	$2^{48.66}$	$2^{48.97}$
28	$2^{34.08}$	$2^{34.54}$	20	$2^{51.04}$	$2^{51.33}$
27	$2^{35.97}$	$2^{36.39}$	19	$2^{53.51}$	$2^{53.78}$
26	$2^{37.90}$	$2^{38.31}$	18	$2^{56.06}$	$2^{56.31}$
25	$2^{39.91}$	$2^{40.30}$	15	$2^{64.27}$	$2^{64.48}$
24	$2^{41.99}$	$2^{42.35}$	10	$2^{80.18}$	$2^{80.31}$
23	$2^{44.14}$	$2^{44.48}$	5	$2^{99.96}$	$2^{100.02}$

Table 5: Complexity for the Hamming Weight Property (Random Search; $n = 128$)

results show that complexity $\mathcal{C}_{k+}$ is systematically lower than complexity $\mathcal{C}_k$. As a matter of fact, it is better to consider the property relevant to $\mathcal{C}_k$ with $k \leq 22$ at least, for our purposes.

4.2 The Run Property

The purpose is now to find secret keys for which S_K outputs a sequence whose r first bits are all zeroes (*runs* of zeroes). Sequences satisfying this property will be denoted $\sigma_K^{128,r}$. The probability of such a sequence is given by

$$P[\sigma_K^{128,r}] = \frac{2^{128-r}}{2^{128}} = \frac{1}{2^r} = p_r.$$

The proof is obvious when considering basic combinatorics. The resulting complexity to find such a sequence at random is

$$\mathcal{C}_r = 2^r.$$

As for the Hamming weight property, if we manage to exhibit a secret key K such that S_K output a sequence $\sigma_K^{128,r}$, for relatively large value of r, then we 0-Kl prove that we know a far more efficient attack than the exhaustive search which is untractable at the present time.

The reader will note that any fixed sequence with some structure could be used instead of runs, provided that any verifier is convinced that this sequence has not been chosen after an simple encryption process (*a posteriori* choice). This is the reason why we choose runs of zeroes which can be considered as a rather "remarquable sub-sequence".

4.3 Cumulating Hamming Weight and Run Properties

We now want to find secret keys such that S_K outputs a sequence whose r first bits is a run of zeroes and whose Hamming weight is equal to k. Such a sequence will be denoted $\sigma_K^{128,r,k}$. It is then easy to prove that the probability to find such a key at random (exhaustive search) is given by

$$P[\sigma_K^{128,r,k}] = \frac{\binom{128-r}{k}}{2^{128}} = p_{r,k}.$$

The resulting complexity to find such a key is given by

$$\mathcal{C}_{r,k} = \frac{2^{128}}{\binom{128-r}{k}}.$$

In the same way, we can consider output sequences with runs located anywhere in the sequence and not only at the beginning. Such a sequence is denoted $\sigma_K^{128,r+,k}$. The probability to find such a key at random (exhaustive search) is given by

$$P[\sigma_K^{128,r,k}] = \frac{(128-r)\binom{128-r}{k}}{2^{128}} = p_{r+,k}.$$

The resulting complexity to find such a key is given by

$$\mathcal{C}_{r+,k} = \frac{2^{128}}{(128-r)\binom{128-r}{k}}.$$

(r, k)	$\mathcal{C}_{k+}$	$\mathcal{C}_k$	$\mathcal{C}_r$	$\mathcal{C}_{r,k}$	$\mathcal{C}_{r+,k}$
(69, 29)	$2^{32.27}$	$2^{32.76}$	2^{69}	$2^{72.28}$	$2^{66.40}$
(69, 27)	$2^{35.97}$	$2^{36.39}$	2^{69}	$2^{72.57}$	$2^{66.69}$
(69, 25)	$2^{39.91}$	$2^{40.30}$	2^{69}	$2^{73.25}$	$2^{67.36}$
(68, 27)	$2^{35.97}$	$2^{36.39}$	2^{68}	$2^{71.71}$	$2^{65.80}$
(67, 26)	$2^{37.90}$	$2^{38.31}$	2^{67}	$2^{71.24}$	$2^{65.31}$
(67, 24)	$2^{41.99}$	$2^{42.35}$	2^{67}	$2^{72.27}$	$2^{66.34}$
(66, 29)	$2^{32.27}$	$2^{32.76}$	2^{66}	$2^{69.49}$	$2^{63.53}$
(66, 26)	$2^{37.90}$	$2^{38.31}$	2^{66}	$2^{70.45}$	$2^{64.50}$
(65, 28)	$2^{34.08}$	$2^{34.54}$	2^{65}	$2^{68.87}$	$2^{62.89}$
(65, 27)	$2^{35.97}$	$2^{36.39}$	2^{65}	$2^{69.23}$	$2^{63.25}$
(65, 26)	$2^{37.90}$	$2^{38.31}$	2^{65}	$2^{69.69}$	$2^{63.71}$
(64, 27)	$2^{35.97}$	$2^{36.39}$	2^{64}	$2^{68.44}$	$2^{62.44}$
(63, 29)	$2^{32.27}$	$2^{32.76}$	2^{63}	$2^{66.87}$	$2^{60.85}$
(63, 28)	$2^{34.08}$	$2^{34.54}$	2^{63}	$2^{67.23}$	$2^{61.20}$
(63, 27)	$2^{35.97}$	$2^{36.39}$	2^{63}	$2^{67.67}$	$2^{61.64}$
(62, 29)	$2^{32.27}$	$2^{32.76}$	2^{62}	$2^{66.04}$	$2^{59.99}$
(62, 28)	$2^{34.08}$	$2^{34.54}$	2^{62}	$2^{66.43}$	$2^{60.38}$
(62, 27)	$2^{35.97}$	$2^{36.39}$	2^{62}	$2^{66.91}$	$2^{60.86}$
(62, 26)	$2^{37.90}$	$2^{38.31}$	2^{62}	$2^{67.47}$	$2^{61.43}$
(60, 23)	$2^{44.14}$	$2^{44.48}$	2^{60}	$2^{68.52}$	$2^{62.43}$

Table 6: Complexity Comparison for Hamming Weight and Run Properties (Random Search; $n = 128$)

Table 6 compares the different complexity $\mathcal{C}_k, \mathcal{C}_{k+}, \mathcal{C}_r$ and $\mathcal{C}_{r,k}$ for different values of k and r. Note that the complexities we have given in the present section refer to the detection of one unique secret key K. Looking for ν such keys increases the relevant complexity in the same order of magnitude. In other words, we have to multiply the complexity by ν.

5 E0 Zero-knowledge-like proof of cryptanalysis

Important weaknesses have been identified for E0. To the author's knowledge, they have never been published so far. These weaknesses are mainly of combinatorial nature. The *CoHS*[3] and VAUBAN packages have been used

[3] *CoHS* stands for *Combinatorics over Huge Sets*

in a precomputing step. The first package is a combinatorial flaw scanner whereas the second one translates the detected flaws into one or more statistical estimators suitable for cryptanalysis. They both are non public packages.

All the three properties have been successfully considered. Each time secret keys have been found for different values of k and r. For the run property, without loss of generality, we considered run of zeroes. The memory bits c_{-1} and c_0 have been chosen equal to zero as well. This appears to be a more challenging choice: the null vector $K = (0, 0, 0, \ldots, 0)$ obviously produces in this setting the null sequence. Thus retrieving keys that zeroes the r first bits while having non zero Hamming weight increases the difficulty. However, the results could be obtained for any other settings (runs of ones and different memory bits initialisation).

The precomputing step with the two packages took approximatively one week of computing time on a Athlon 64. The work must be done only once (and for all). For each possible choice of runs and values k and r, the cryptanalysis step is performed (on four DEC 9000 machines). The first keys have been retrieved within the first hour while slightly more than five weeks have been necessary to retrieve slightly more than 48,000 keys. Some of the most significant sequences are given in Appendix B. Table 7 provides results about the number of secret keys retrieved for each property, during five weeks of computing (detailed results available upon request). The

k	Hamming weight property	(r, k)	Cumulated properties
19	1	(69, 29)	1
20	5	(69+, 27)	1
21	18	(69+, 25)	1
22	38	(66+, 26)	3

Table 7: Number of Keys Found With Respect to Properties ($n = 128$)

most significant results deals with the retrieval of a secret key K outputting a sequence $\sigma_K^{128,69,29}$. Finding such a key would require an exhaustive or a random search of $2^{72.28}$, in average. For the moment, this cannot be achieved with existing computing resources. Consequently, this implies to know weaknesses enabling to retrieve such a key faster than with exhaustive search.

The approximative equivalent complexity of the computation which enables to recover slightly more than 48,000 has been empirically evaluated by comparing the number of keys effectively treated by the attack with respect to the time that a simple exhaustive search would require. This yields a complexity of $\mathcal{O}(2^{35})$. The theoretical value of complexity has been computed but the proof will not be given in order the cryptanalysis to remain zero-knowledge-like. Let us mention that theoretical, expected and observed complexities do not significantly differ.

At last, the properties we have considered does not provide any information about the method to obtain them. In other words, the verifier cannot induce what weaknesses have been exploited. Of course, a rigorous proof could be given only by precisely exhibiting those weaknesses. Only the future will confirm or invalidate the fact that our approach is truly zero-knowledge. If nobody is able to reproduce such properties in the same amount of time, then we can consider that indeed it is.

At the present time, none of the known attacks can obtain the results we have presented in this paper: either building secret keys producing keystream with given properties or retrieving secret keys from fixed keystream with desired properties. However, it is an open problem to determine whether the attacks of Table 4 can be modified or improved to obtain the results presented before.

6 Future Work and Conclusion

In this paper, we have presented a scheme to prove the cryptanalysis of an encryption algorithm without disclosing any information on the nature of the cryptanalysis, while any verifier can check in a polynomial time the reality of that cryptanalysis. It becomes then acceptable to disclose information about the weaknesses of cryptosystems without fearing that "bad guys" will reproduce and use it for real attacks purpose. This scheme can be applied to any symmetric cryptosystem. Block ciphers can be considered as stream ciphers when fixing the plaintext block.

At the present time, the results exhibited in this paper allow to greatly put E0 security into question. In the future, viral attacks could occur by precisely bypassing Bluetooth security at the cryptographic level if any other people found equivalent or more important weaknesses in E0. That kind of risk cannot be denied.

As far as E0 stream cipher is concerned, current work is in progress to greatly improve the efficiency of our attack while new properties for 0-Kl proof of cryptanalysis will be considered. Other cryptosystems used in real transmission protocols are currently analysed with CoHS and Vauban packages in order to exhibit vulnerabilities that could be exploitable in practical cryptanalysis.

Acknowledgements

The author would like to thank all reviewers for their valuable comments which greatly help to improve the final quality of this paper.

References

[1] Armknecht F. and Krause M. (2003), "Algebraic Attacks on Combiners with Memory". In *Advances in Cryptology - CRYPTO'03*, LNCS 2729, pp. 162–175, Springer Verlag.

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

[3] Description of the *Epoc.Cabir* virus. `www.f-secure.com/v-descs/cabir.shtml`

[4] Courtois, N. (2003), "Fast Algebraic Attacks on Stream Ciphers with Linear Feedback". In *Advances in Cryptology - CRYPTO'03*, LNCS 2729, pp. 176-194, Springer-Verlag.

[5] Filiol E. (2005) *Computer Viruses: from Theory to Applications*, IRIS International series, Springer Verlag, ISBN 2-287-23939-1.

[6] Fluhrer S. and Lucks S. (2001), "Analysis of the E0 Encryption System", *Selected Areas in Cryptography - SAC 2001*, LNCS 2259, pp. 38–48, Springer-Verlag.

[7] Fluhrer S. (2002), "Improved Key Recovery of Level 1 of the Bluetooth Encryption System", available at `http://eprint.iacr.org/2002/068`

[8] Goldreich O. (2001), *Foundations of Cryptography – Basic Tools*. Cambridge University Press, Cambridge.

[9] Goldwasser S., Micali S. and Rackoff C. (1989), "The Knowledge-complexity of Interactive Proof Systems", *SIAM Journal on Computing*, Vol. 18, pp. 186–208.

[10] Golic J., Bagini V. and Morgani G., "Linear cryptanalysis of Bluetooth stream cipher". In *Advances in Cryptology - EUROCRYPT'02*, LNCS 2332, pp. 238–255, Springer, 2002.

[11] Krause M. (2002), "BDD-based cryptanalysis of keystream generators". In *Advances in Cryptology - EUROCRYPT 02*, LNCS 2332, pp. 222–237, Springer-Verlag.

[12] Loi pour la confiance en l'économie numérique (Law for Confidence in the e-Economy), Journal Officiel, June 22nd, 2004. A detailed presentation of this law as well as comments and legal explanation of this law can be found in English in [5, Chap. 5].

[13] Levy O. and Wool A. (2005), "A Uniform Framework for Cryptanalysis of the Bluetooth E0 Cipher". Available at `eprint.iacr.org/2005/107.pdf`

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

[15] Lu Y., Meier W. and Vaudenay S. (2005), "The Conditional Correlation Attack: A Practical Attack on Bluetooth Encryption". In *Advances in Cryptology - CRYPTO'05*, LNCS 3621, pp. 97–117, Springer Verlag.

[16] Saarinen, M.-J., "A Software Implementation of the BlueTooth Encryption Algorithm E0". Available at `http://www.jyu.fi/~mjos/e0.c`

[17] Revised NIST Special Publication 88-22 (2000), "A Statistical Test Suite for the Validation of Ramdom Number Generator and Pseudorandom Number Generator for Cryptographic Applications". National Institute of Standard and Technology, US Commerce Department's Technology Administration, `http://csrc.nist.gov/rng/rng2.html`

[18] Shaked Y. and Wool A. (2005), Cracking the Bluetooth PIN. In *Proc. 3rd USENIX/ACM Conf. Mobile Systems, Applications, and Services (MobiSys)*, Seattle, pp. 39–50, ISBN 1-931971-31-5.

[19] "Specification of the Bluetooth system", v.2.0. Core specification, 2004. Available from `http://www.bluetooth.org/foundry/adopters/document/Core_v2.0_EDR/en/1/Core_v2.0_EDR.zip`

[20] U.S. Copyright Office Summary (1998), "The Digital Millenium Copyright Act of 1998", http://www.copyright.gov/legislation/dmca.pdf

A E0 Reference Implementation

We give here the E0 implementation in C programming language, that has been used for this cryptanalysis. Is it mainly based on Saarinen's reference implementation [16]. The reader thus will be able to verify our results. As pointed out by a reviewer, this implementation differs from the official specification. In each iteration of the for-loop in the main encryption procedure, the LFSR-output has to be read *before* instead *after* the LFSRs are clocked. Consequently, the initial LFSR-states given in the Saarinen's implementation have to be clocked one step backwards to work with the original E0. Nonetheless, from a cryptanalytic point of view this minor difference does not have an impact in our approach and results.

A.1 Header File "include.h"

```
#include "stdio.h"
#include "stdlib.h"

#define mot64 unsigned long long int
#define mot32 unsigned long int
#define int32 long int
#define mot16 unsigned int
#define mot08 unsigned char
```

A.2 Header File "e0light.h"

```
#include <stdio.h>
typedef unsigned char mot08;
typedef unsigned long long mot64;

const mot08 e0_fsm[16][16] = {
{ 0,  0,  0,  1,  0,  1,  1,  1,  0,  1,  1,  1,  1,  1,  1,  2},
{ 5,  4,  4,  4,  4,  4,  4,  7,  4,  4,  4,  7,  4,  7,  7,  7},
{11, 11, 11,  8, 11,  8,  8,  8, 11,  8,  8,  8,  8,  8,  8,  9},
{14, 13, 13, 13, 13, 13, 13, 12, 13, 13, 13, 12, 13, 12, 12, 12},
{ 3,  3,  3,  2,  3,  2,  2,  2,  3,  2,  2,  2,  2,  2,  2,  1},
```

```
{ 6,  7,  7,  7,  7,  7,  7,  4,  7,  7,  7,  4,  7,  4,  4,  4},
{ 8,  8,  8, 11,  8, 11, 11, 11,  8, 11, 11, 11, 11, 11, 11, 10},
{13, 14, 14, 14, 14, 14, 14, 15, 14, 14, 14, 15, 14, 15, 15, 15},
{ 1,  1,  1,  0,  1,  0,  0,  0,  1,  0,  0,  0,  0,  0,  0,  3},
{ 4,  5,  5,  5,  5,  5,  5,  6,  5,  5,  5,  6,  5,  6,  6,  6},
{10, 10, 10,  9, 10,  9,  9,  9, 10,  9,  9,  9,  9,  9,  9,  8},
{15, 12, 12, 12, 12, 12, 12, 13, 12, 12, 12, 13, 12, 13, 13, 13},
{ 2,  2,  2,  3,  2,  3,  3,  3,  2,  3,  3,  3,  3,  3,  3,  0},
{ 7,  6,  6,  6,  6,  6,  6,  5,  6,  6,  6,  5,  6,  5,  5,  5},
{ 9,  9,  9, 10,  9, 10, 10, 10,  9, 10, 10, 10, 10, 10, 10, 11},
{12, 15, 15, 15, 15, 15, 15, 14, 15, 15, 15, 14, 15, 14, 14, 14}
};

static mot64 e0_r1, e0_r2, e0_r3, e0_r4;
static int e0_state, e0_x, e0_z;
```

A.3 Encryption Procedure

```
#include "e0light.h"

int e0(mot64 K1, mot64 K2, mot08 KA, mot08 * suite, mot64 nbbit)
 {
  unsigned long int i;
  int t;

  /* Register Initialisation */
  e0_r1 = (K1 & 0x1FFFFFFL);
  e0_r2 = ((K1 >> 25) & 0x7FFFFFFFL);
  e0_r3 = (((K1 >> 56) | (K2 << 8)) & 0x1FFFFFFFFLL);
  e0_r4 = (K2 >> 25);

  e0_state = KA;

  for(i = 0; i < nbbit;i++)
   {
    e0_r1 = ((e0_r1 << 1) & 0x1fffffe) | (((e0_r1 >> 7)
           ^ (e0_r1 >> 11) ^ (e0_r1 >> 19) ^ (e0_r1 >> 24)) & 1);
    e0_r2 = ((e0_r2 << 1) & 0x7ffffffe) | (((e0_r2 >> 11)
           ^ (e0_r2 >> 15) ^ (e0_r2 >> 23) ^ (e0_r2 >> 30)) & 1);
    e0_r3 = ((e0_r3 << 1) & 0x1fffffffell) | (((e0_r3 >> 32)
```

```
          ^ (e0_r3 >> 27) ^ (e0_r3 >> 23) ^ (e0_r3 >> 3)) & 1);
    e0_r4 = ((e0_r4 << 1) & 0x7fffffffell) | (((e0_r4 >> 38)
          ^ (e0_r4 >> 35) ^ (e0_r4 >> 27) ^ (e0_r4 >> 3)) & 1);

    e0_x = ((e0_r1 >> 23) & 1) | ((e0_r2 >> 22) & 2)
         | ((e0_r3 >> 29) & 4) | ((e0_r4 >> 28) & 8);

    e0_state = e0_fsm[e0_state][e0_x];
    t = e0_x ^ (e0_x >> 2);
    t ^= t >> 1;

    suite[i] = (t ^ (e0_state >> 2)) & 1;
   }
 }
```

A.4 Main Procedure

```
#include "include.h"

#define N 128
#define KA 0         /* Initial memory bits   */

int main(int argc, char * argv[])
 {
  mot64 i, j, i0, i1, i2 , i3, K[2];
  mot32 m;
  mot08 * suite, ka, k;

  K[0] = <------- bits  0 --  63 of secret key
  K[1] = <------- bits 64 -- 127 of secret key

  suite    = (mot08 *)calloc(N, sizeof(mot08));
  suite_sc = (mot08 *)calloc(N, sizeof(mot08));
  suite_ka = (mot08 *)calloc(N + 2, sizeof(mot08));

  e0(K[0], K[1], KA, suite, 128LL);
  printf("Output sequence\n\n");
  for(i = 0L; i < 128;i++) printf("%01d", suite[i]);
  printf("\n\n");
  free(suite);
```

```
}
```

B Proof Values of 0-K Cryptanalysis

In this section, we give the key producing the most significant properties. Detailed results are available upon request (slightly more than 48,000 keys). The notation is that of the `main()` procedure given in the previous section of the Appendix.

```
(67+, 24)
    K[0] = 0x73CD595AD3FD6A26 K[1] = 0x4E5BB736824EFAC4

(67+, 26)
    K[0] = 0x481AC9D68A265BB6 K[1] = 0x9C49E65F2C5AC7EC

(66+, 26)
    K[0] = 0x19D2C332127ACF17 K[1] = 0x3616434EA1A991A
    K[0] = 0xD15D3CA3C5240B4D K[1] = 0x11BDAC9BE5D608D2
    K[0] = 0x1168C994D63DBEE1 K[1] = 0xA52DB3C47F6E4B78

(66+, 29)
    K[0] = 0x4AA088310330E134 K[1] = 0x886554F41774B5DF

(65+, 26)
    K[0] = 0x499B5A23B09E73C7 K[1] = 0xBF9A060F485F8708

(65+, 27)
    K[0] = 0x49EE7FAEDE74A51B K[1] = 0x9EF861C90E85C6A0

(65+, 28)
    K[0] = 0xCB9E8BC74B91EA42 K[1] = 0x4575201CFBDC7FF9

(64, 27)
    K[0] = 0x09F51F2AEE52BBCC K[1] = 0x345991408FD0A40B

(63+, 27)
    K[0] = 0x3AF59A1AB3849A22 K[1] = 0xA8F0630AAB90E4EE

(63+, 29)
    K[0] = 0xC98D344092E7B8A6 K[1] = 0x18FFAA9AB4BB0FB2
```

```
    K[0] = 0x3395F4E0AA7F2AAA K[1] = 0x7D3C8F1CC1A9FB61
    K[0] = 0x60595B6C3F81FBC7 K[1] = 0x39608B22C62E8C79

(63+, 28)
    K[0] = 0x0DB55B6143A3DF6A K[1] = 0xC69A087CB6FA29E5

(62, 29)
    K[0] = 0x18C1077579DD290B K[1] = 0x5B672FC8D0CCE243

(62, 27)
    K[0] = 0xF11D6526C305E816 K[1] = 0x35BE571A69C9B6EA

(62+, 29)
    K[0] = 0xC88DDB3D2D6415F4 K[1] = 0xA219615A07B7BFFF
    K[0] = 0xC40BA27939383C32 K[1] = 0xC1692DEF036E7049
    K[0] = 0x9D45CC6215D1E5B3 K[1] = 0x39CB14370AEB1CB2

(61+, 29)
    K[0] = 0xF1F70889D3A6FF5D K[1] = 0x4DD6D71E317B540B
    K[0] = 0x1B9456D34AA3E596 K[1] = 0x9E183710E7B6138B

(62+, 27)
    K[0] = 0x44F646AB3AED19E0 K[1] = 0xC3BC20A780A2BA3E
    K[0] = 0x42461FB9C07F3F9D K[1] = 0x746A780C6A649D6B

(62+, 26)
    K[0] = 0x7B1B5463C802FFB5 K[1] = 0xA3FDF5940264D28B
    K[0] = 0x89E14644C0AD64BB K[1] = 0xC077883C768664D5
    K[0] = 0x33E24602D7A02C18 K[1] = 0xBF3C9A7CD53C865D

(62+, 28)
    K[0] = 0x125D85B3A3353C2A K[1] = 0xA8E12FDAD9269406

(61+, 27)
    K[0] = 0x2FA83A7A4959C2FE K[1] = 0xCCF65606210D32C9

(61+, 26)
    K[0] = 0xF01896F8455DDBD5 K[1] = 0x604AC5B5048A233D

(60+, 23)
```

`K[0] = 0xB8F7ABBACC30347F K[1] = 0xEEDC60766DAA3F32`