Cracking the Bluetooth PIN*

Yaniv Shaked and Avishai Wool

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

Abstract

This paper describes the implementation of an attack on the Bluetooth security mechanism. Specifically, we describe a passive attack, in which an attacker can find the PIN used during the pairing process. We then describe the cracking speed we can achieve through three optimizations methods. Our fastest optimization employs an algebraic representation of a central cryptographic primitive (SAFER+) used in Bluetooth. Our results show that a 4-digit PIN can be cracked in less than 0.3 sec on an old Pentium III 450MHz computer, and in 0.06 sec on a Pentium IV 3Ghz HT computer.

1 Introduction

1.1 Background

Bluetooth, a technology used for short range fast communications, has quickly spread worldwide. Bluetooth technology is used in a large set of wired and wireless devices: mobile phones, PDA's, desktop and mobile PC's, printers, digital cameras, and dozens of other devices. Being wireless, Bluetooth is potentially vulnerable to many attacks. It is very difficult to avoid Bluetooth signals from leaking outside the desired boundaries. The possible damage of a successful wireless attack starts with the ability to eavesdrop on the data transferred during the communication of two devices, and ends with the ability to fully impersonate other devices.

The Bluetooth technology has a significant security component, which includes key management, authentication and secrecy. However, the security of the whole system relies on the user's choice of a secret Personal Identification Number (PIN) - which is often much too short. Moreover, the Bluetooth designers invented several new cryptographic primitives, which were incorporated into the system. Cryptographers consider fielding new primitives to be risky, because new cryptography is less tested and may contain hidden flaws. Furthermore, Bluetooth is designed for short-range communication (nominal range of about 10m). This short-range is perceived as a security feature, since an attacker is supposed to be quite near the attack target - but recent history with IEEE 802.11 has shown that effective range-extenders can be built very cheaply [Reh03]. Finally, as Bluetooth gains popularity on PDAs and laptops, the information that lures attackers grows from cell-phone address books to valuable corporate data.

1.2 Related work

A number of crypt-analytical results regarding Bluetooth [HN99, FL01, Flu02, Kra02, Arm02, LV04, LW05] have appeared over the last five years. Most of the work has focused on the lowest level cipher, called E_0. This is a completely new cipher, designed specifically for Bluetooth. The current state of the art is that no practical attacks, with current technology, have surfaced, *yet*. However, it is already clear that the security of the cipher is much less than claimed: although E_0 uses 128-bit keys, its effective security is no more than an 84-bit system (approximately). If E_0 were to be used outside of the Bluetooth system, and allowed to produce a stream of several million bits, then [LV04] shows E_0 to be effectively a 39-bit system - which would make it much too weak for use. These are worrisome, if not yet fatal, developments.

As for a security analysis of the system as a whole, much less has been done. The most significant work so far is [JW01], which identified some weaknesses. Over the last two years some hacker tools are starting to emerge (with colorful names such as "bluesnarfing" [Lau03], "bluejacking" [Blu04], and "redfang" [Whi03]).

The work closest to ours was recently done by O. Whitehouse, independently, and announced at the

*Supported in part by a grant from Intel Corporation.

Term	Explanation
PIN	Personal Identification Number. The PIN code is 1-8 bytes long (8-128 bits). However, most devices use PIN sizes of 4 decimal digits.
BD_ADDR	Each Bluetooth device has a 48 bit unique address that is called the Bluetooth Device Address.
Pairing	The process in which two (or more) Bluetooth devices hook up to create a shared secret value called K_{init}. The K_{init} forms the basis for all future Bluetooth negotiations between these two devices.

Table 1: List of terms

CanSecWest '04 conference [Whi04]. His work contains a survey of many aspects of Bluetooth security. However, as far as PIN cracking goes, the author only describes the attack framework, with rough time estimates. Precise technical details of the attack (beyond the presentation slides) have not been published. Unlike our work, the author apparently did not implement a PIN-cracking program. Thus we believe that our implementation, measurements, and our optimization methods, are all novel.

1.3 Contribution

In this paper we introduce a passive attack, in which an attacker can find the PIN used during the Bluetooth pairing process. We then describe implementations of this attack, using three optimizations methods. For this purpose we wrote a special-purpose Bluetooth security suite from scratch. Our fastest optimization employs an algebraic representation of a central cryptographic primitive (SAFER+) used in Bluetooth. Our results show that a 4-digit PIN can be cracked in less than 0.3 sec on an old Pentium III 450MHz computer, and in 0.06 sec on a Pentium IV 3Ghz HT computer. We then sketch an additional attack that can force the Bluetooth devices to repeat the pairing process and make them vulnerable to the first attack.

Organization: In Section 2 we give an overview of Bluetooth security, focusing on the Bluetooth pairing and authentication mechanism. Section 3 describes the passive attack, in which an attacker can find the PIN used during the pairing process. Section 4 contains a description of five versions implementing such an attack, with their respective performance figures. Section 5 sketches the additional attack, which can force two devices to repeat the pairing process. Section 6 presents countermeasures that will reduce the probability of being subjected

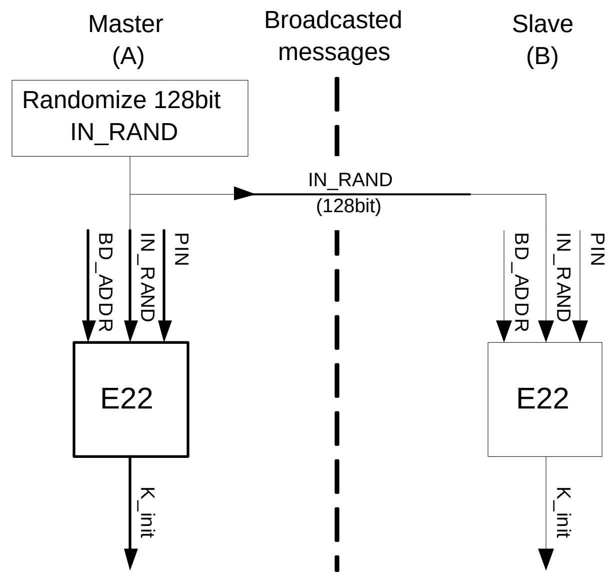

Figure 1: Generation of K_{init} using E_{22}

to both attacks and the vulnerability to these attacks, and we conclude our work in Section 7.

2 Overview of Bluetooth Security

A detailed specification of Bluetooth security mechanisms can be found in part H of Vol 2 of [Blu03]. A list of terms used repeatedly in this paper is given in Table 1.

This papers deals with the mechanisms used in Bluetooth Security Mode 3: The Link-level security mode. In this mode, a Bluetooth device will initiate security measures before a channel is established. This is a built-in mechanism, that is used regardless of the application layer security that may also be used. In security mode 3 terminology, establishing a channel between two Bluetooth devices is called pairing or bonding.

2.1 The Bluetooth pairing & authentication process

The Bluetooth initialization procedures consists of 3 or 4 steps:

1. Creation of an initialization key (K_{init}).
2. Creation of a link key (K_{ab}).
3. Authentication.

After the 3 pairing steps are completed, the devices can derive an encryption key to hide all future communication in an optional fourth step.

Before the pairing process can begin, the PIN code must be entered into both Bluetooth devices. Note that in

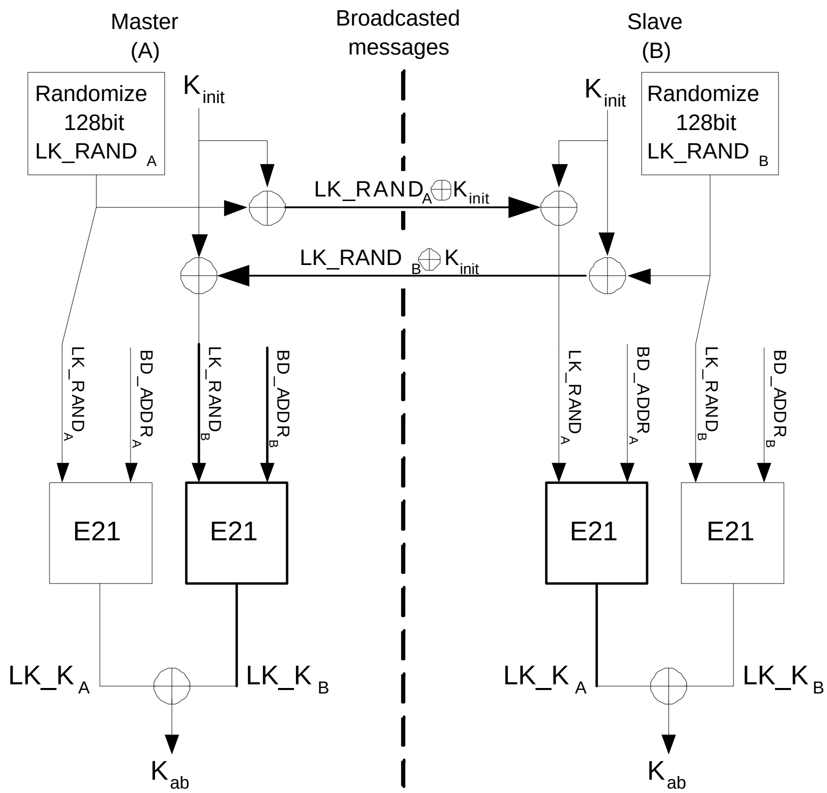

Figure 2: Generation of K_{ab} using E_{21}

some devices (like wireless earphones) the PIN is fixed and cannot be changed. In such cases, the fixed PIN is entered into the peer device. If two devices have a fixed PIN, they cannot be paired, and therefore cannot communicate. In the following sections we go into the details of the steps of the pairing process.

2.1.1 Creation of K_{init}

The K_{init} key is created using the E_{22} algorithm, whose inputs are:

1. a *BD_ADDR*.
2. the PIN code and its length.
3. a 128 bit random number *IN_RAND*.

This algorithm outputs a 128 bit word, which is referred to as the initialization key (K_{init}).

Figure 1 describes how K_{init} is generated using E_{22}. Note that the PIN code is available at both Bluetooth devices, and the 128 bit *IN_RAND* is transmitted in plaintext. As for the *BD_ADDR*: if one of the devices has a fixed PIN, they use the *BD_ADDR* of the peer device. If both have a variable PIN, they use the PIN of the slave device that receives the *IN_RAND*. In Figure 1, if both devices have a variable PIN, *BD_ADDR*$_B$ shall be used. The Bluetooth device address can be obtained via an inquiry routine by a device. This is usually done before connection establishment begins. A detailed explanation of the inner design of E_{22} can be found in Appendix B.1.

This initialization key (K_{init}) is used only during the pairing process. Upon the creation of the link key (K_{ab}), the K_{init} key is discarded.

2.1.2 Creation of K_{ab}

After creating the initialization key, the devices create the *link key* K_{ab}. The devices use the initialization key to exchange two new 128 bit random words, known as *LK_RAND*$_A$ and *LK_RAND*$_B$. Each device selects a random 128 bit word and sends it to the other device after bitwise xoring it with K_{init}. Since both devices know K_{init}, each device now holds both random numbers *LK_RAND*$_A$ and *LK_RAND*$_B$. Using the E_{21} algorithm, both devices create the *link key* K_{ab}. The inputs of E_{21} algorithm are:

1. a *BD_ADDR*.
2. The 128 bit random number *LK_RAND*.

Note that E_{21} is used twice is each device, with two sets of inputs. Figure 2 describes how the link key K_{ab} is created. A detailed explanation of the inner design of E_{21} can be found in Appendix B.2.

2.1.3 Mutual authentication

Upon creation of the link key K_{ab}, mutual authentication is performed. This process is based on a challenge-response scheme. One of the devices, the verifier, randomizes and sends (in plaintext) a 128 bit word called *AU_RAND*$_A$. The other device, the claimant, calculates a 32 bit word called *SRES* using an algorithm E_1. The claimant sends the 32 bit *SRES* word as a reply to the verifier, who verifies (by performing the same calculations) the response word. If the response word is successful, the verifier and the claimant change roles and repeat the entire process. Figure 3 describes the process of mutual authentication. The inputs to E_1 are:

1. The random word *AU_RAND*$_A$.
2. The link key K_{ab}.
3. Its own Bluetooth device address (*BD_ADDR*$_B$).

A detailed explanation of the inner design of E_1 can be found in Appendix B.3.

Note that as a side effect of the authentication process, a 96 bit word called ACO is calculated by both peers. This word is optionally used during the creation of the encryption key. The creation of this encryption key exceeds our primary discussion and shall not be described in this paper.

2.2 Bluetooth cryptographic primitives

As we described above, the Bluetooth pairing and authentication process uses three algorithms: E_{22}, E_{21}, E_1. All of these algorithms are based on the SAFER+ cipher [MKK98], with some modifications. Here we describe features of SAFER+ that are relevant to our attack.

2.2.1 Description of SAFER+

SAFER+ is a block cipher [MKK98] with a block size of 128 bits and three different key lengths: 128, 192 and 256 bits. Bluetooth uses SAFER+ with 128 bit key length. In this mode, SAFER+ consists of:

1. KSA - A key scheduling algorithm that produces 17 different 128-bit subkeys.
2. 8 identical rounds.
3. An output transformation - which is implemented as a xor between the output of the last round and the last subkey.

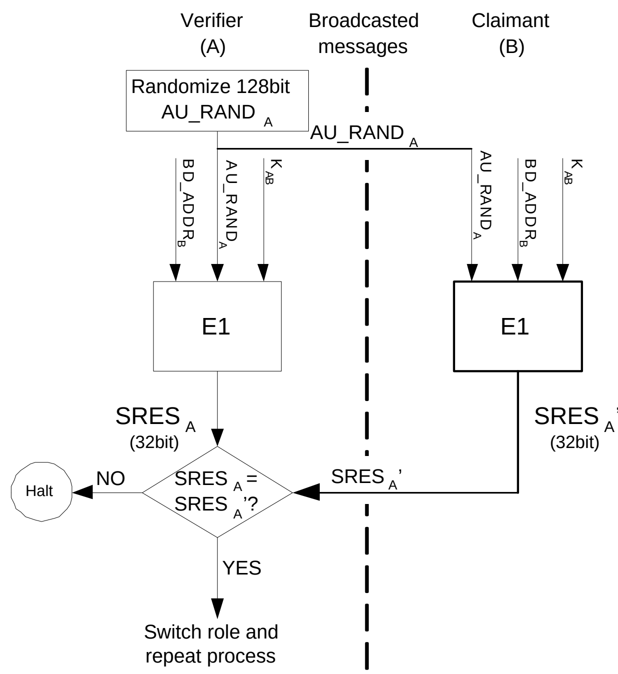

Figure 3: Mutual authentication process using E_1

Figure 4 describes the inner design of SAFER+, as it is used in Bluetooth.

The key scheduling algorithm (KSA)

The key scheduling algorithm used in SAFER+ produces 17 different 128-bit subkeys, denoted K_1 to K_{17}. Each SAFER+ round uses 2 subkeys, and the last key is used in the SAFER+ output transformation. The important details for our discussion are that in each step of the KSA, each byte is cyclic-rotated left by 3 bit positions, and 16 bytes (out of 17) are selected for the output subkey. In addition, a 128 bit bias vector, different in each step, is added to the selected output bytes. The entire process of key scheduling is detailed in Appendix A.1.

The SAFER+ Round

As depicted in Figure 4, SAFER+ consists of 8 identical rounds. Each round calculates a 128 bit word out of two subkeys and a 128 bit input word from the previous round. The central components of the SAFER+ round are the 2-2 Pseudo Hadamard Transform (PHT), the Armenian Shuffles, and the substitution boxes denoted "**e**" and "**l**".

The Pseudo Hadamard Transform takes two input bytes and produces two output bytes, as follows:

$$PHT[a, b] = [(2a + b) \bmod 256, (a + b) \bmod 256]$$

The Armenian Shuffle is a permutation of 16 bytes. See Figure 5 for the Armenian shuffle order.

The substitution boxes "**e**" and "**l**" are non-linear, both replace an input byte with an output byte. Their imple-

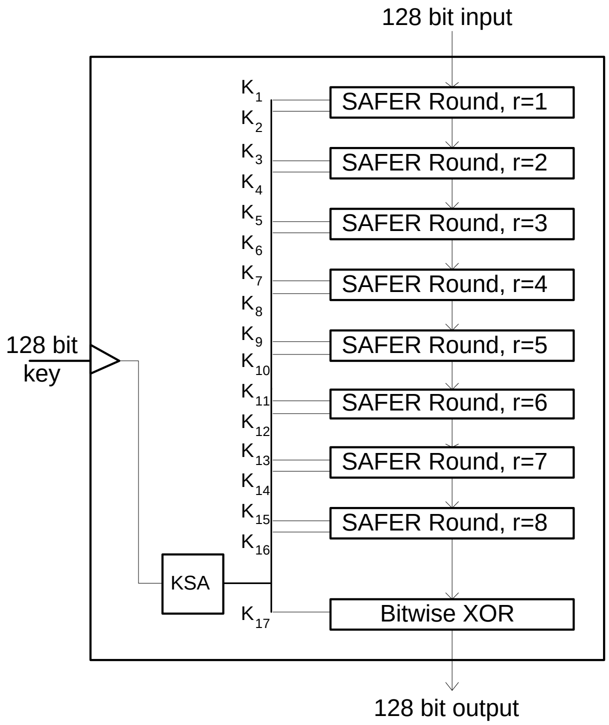

Figure 4: Inner design of SAFER+

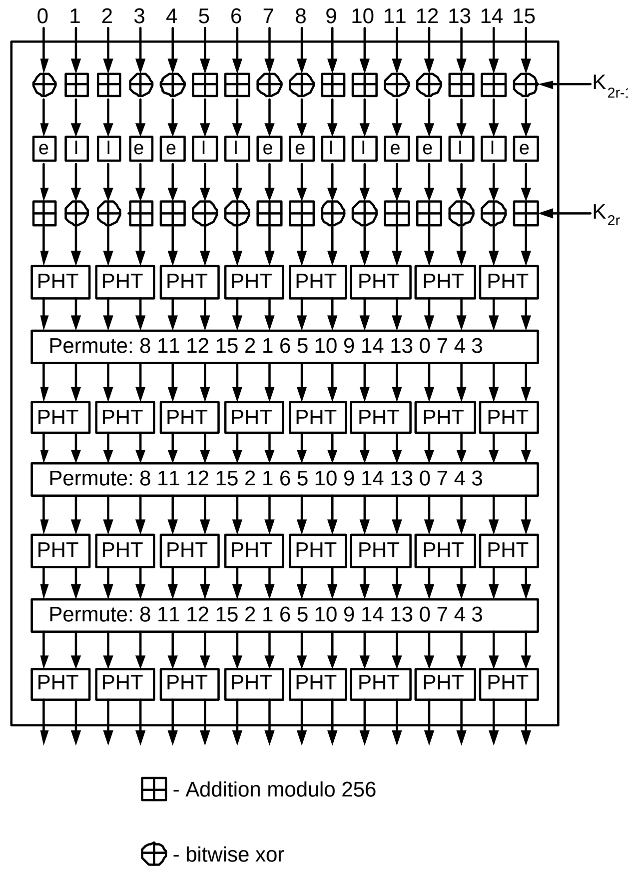

Figure 5: Structure of one SAFER+ round

mentation is given in equations (1) and (2):

$$e(x) = (45^x \pmod{257}) \bmod 256 \tag{1}$$

$$l(x) = y \;\; s.t. \;\; e(y) = x \tag{2}$$

Figure 5 describes the structure of one SAFER+ round.

3 Bluetooth PIN Cracking

3.1 The Basic Attack

Assume that the attacker eavesdropped on an entire pairing and authentication process, and saved all the messages (see Table 2). The attacker can now use a brute force algorithm to find the PIN used. The attacker enumerates all possible values of the PIN. Knowing *IN_RAND* and the *BD_ADDR*, the attacker runs E_{22} with those inputs and the guessed PIN, and finds a hypothesis for K_{init}. The attacker can now use this hypothesis of the initialization key, to decode messages 2 and 3. Messages 2 and 3 contain enough information to perform the calculation of the link key K_{ab}, giving the attacker a hypothesis of K_{ab}. The attacker now uses the data in the last 4 messages to test the hypothesis: Using K_{ab} and the transmitted AU_RAND_A (message 4), the attacker calculates *SRES* and compares it to the data of message 5. If necessary, the attacker can use the value of messages 6 and 7 to re-verify the hypothesis K_{ab} until the correct PIN is found. Figure 6 describes the entire process of PIN cracking.

#	Src	Dst	Data	Length	Notes
1	A	B	*IN_RAND*	128 bit	plaintext
2	A	B	LK_RAND_A	128 bit	XORed with K_{init}
3	B	A	LK_RAND_B	128 bit	XORed with K_{init}
4	A	B	AU_RAND_A	128 bit	plaintext
5	B	A	*SRES*	32 bit	plaintext
6	B	A	AU_RAND_B	128 bit	plaintext
7	A	B	*SRES*	32 bit	plaintext

Table 2: List of messages sent during the pairing and authentication process. “A” and “B” denote the two Bluetooth devices.

Note that the attack, as described, is only fully successful against PIN values of under 64 bits. If the PIN is

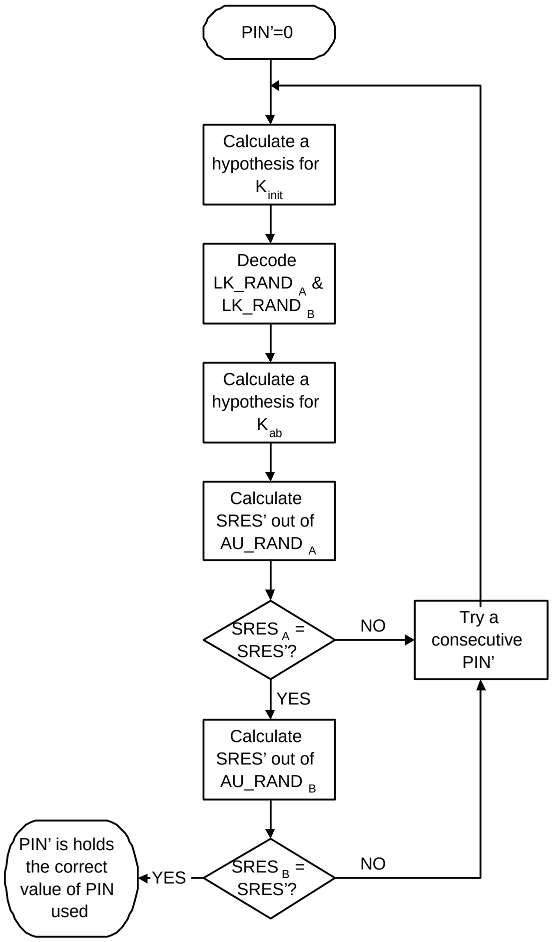

Figure 6: The Basic Attack Structure.

longer, then with high probability there will be multiple PIN candidates, since the two *SRES* values only provide 64 bits of data to test against. A 64 bit PIN is equivalent to a 19 decimal digits PIN.

4 Implementation

This section describes our implementation of the PIN cracking attack, through several optimization versions. We implemented all the versions in C with some embedded 80x86 assembly instructions. We used the Microsoft VC++ compiler on a PC running Microsoft Windows 98.

4.1 The Baseline

Before writing optimized versions of the code, we established two baseline implementations for comparison purposes, as follows.

4.1.1 The "as-is" version

This version is a non-optimized implementation of the attack, using C code only. The bias vectors (see Section 2.2.1) which are used during the SAFER+ key scheduling algorithm are calculated offline, and the substitution boxes **e** and **l** are implemented using two pre-calculated look-up tables.

4.1.2 The basic version

This version is identical to the "as-is" version, but with compiler optimizations to yield maximal speed[1].

4.2 Improved KSA & Expansion

Our first optimization technique focuses on the SAFER+ Key Scheduling Algorithm (KSA). We identified two effective optimizations in the KSA:

1. Caching the calculation result of the expansion operation in the E_{21} and E_{22} algorithms on the *BD_ADDR* of both peers. Since the input of *BD_ADDR* to E_{21} and E_{22} is nearly static (only two values of *BD_ADDR* are used during the PIN cracking attack), it is possible to perform the calculation of Expansion(*BD_ADDR*,6) (see Appendices B.1 and B.2) only once, and save the result for later use.

2. Enhancements of the implementation of the key scheduling algorithm. We found that the implementation of the byte-rotate operation (recall Section 2.2.1) using C code is expensive. Instead we used inline assembly code which employed the *ROL* instruction. Furthermore, we found that the modulo 17 operation used to extract specific bytes from a batch of 17 bytes during the key scheduling algorithm is very expensive. Instead, we used a pre-calculated look-up table.

4.3 PHT as lookup-table

In this version we used a large look-up table to implement the Pseudo Hadamard Transformation (PHT) operation, which is used 32 times during a single SAFER+ round. The look-up table is 65,536 entries long, since the transformation receives two bytes (2^8) and replaces them. The routine which implements the use of such a look-up table was written in pure assembly code. The look-up table was pre-calculated offline.

[1] Compile option /O2 yields maximal speed.

4.4 Algebraic Manipulation

Our most interesting and most effective optimization is the algebraic manipulation of the SAFER+ round. A key observation is that almost the entire SAFER+ round[2] can be implemented as a 16x16 matrix multiplying the vector of 16 input bytes (all operations modulo 256). This is possible since the operations used in the Armenian shuffles and Pseudo Hadamard Transformations are linear. By tracing back through the Shuffles and PHT boxes we computed the 16x16 matrix coefficients as follows:

$$\begin{pmatrix}
2 & 2 & 1 & 1 & 16 & 8 & 2 & 1 & 4 & 2 & 4 & 2 & 1 & 1 & 4 & 4 \\
1 & 1 & 1 & 1 & 8 & 4 & 2 & 1 & 2 & 1 & 4 & 2 & 1 & 1 & 2 & 2 \\
1 & 1 & 4 & 4 & 2 & 1 & 4 & 2 & 4 & 2 & 16 & 8 & 2 & 2 & 1 & 1 \\
1 & 1 & 2 & 2 & 2 & 1 & 2 & 1 & 4 & 2 & 8 & 4 & 1 & 1 & 1 & 1 \\
4 & 4 & 2 & 1 & 4 & 2 & 4 & 2 & 16 & 8 & 1 & 1 & 1 & 1 & 2 & 2 \\
2 & 2 & 2 & 1 & 2 & 1 & 4 & 2 & 8 & 4 & 1 & 1 & 1 & 1 & 1 & 1 \\
1 & 1 & 4 & 2 & 4 & 2 & 16 & 8 & 2 & 1 & 2 & 2 & 4 & 4 & 1 & 1 \\
1 & 1 & 2 & 1 & 4 & 2 & 8 & 4 & 2 & 1 & 1 & 1 & 2 & 2 & 1 & 1 \\
2 & 1 & 16 & 8 & 1 & 1 & 2 & 2 & 1 & 1 & 4 & 4 & 4 & 2 & 4 & 2 \\
2 & 1 & 8 & 4 & 1 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 4 & 2 & 2 & 1 \\
4 & 2 & 4 & 2 & 4 & 4 & 1 & 1 & 2 & 2 & 1 & 1 & 16 & 8 & 2 & 1 \\
2 & 1 & 4 & 2 & 2 & 2 & 1 & 1 & 1 & 1 & 1 & 1 & 8 & 4 & 2 & 1 \\
4 & 2 & 2 & 2 & 1 & 1 & 4 & 4 & 1 & 1 & 4 & 2 & 2 & 1 & 16 & 8 \\
4 & 2 & 1 & 1 & 1 & 1 & 2 & 2 & 1 & 1 & 2 & 1 & 2 & 1 & 8 & 4 \\
16 & 8 & 1 & 1 & 2 & 2 & 1 & 1 & 4 & 4 & 2 & 1 & 4 & 2 & 4 & 2 \\
8 & 4 & 1 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 1 & 2 & 1 & 4 & 2
\end{pmatrix}$$

Our goal is to implement the multiplication (a 16 coefficients vector by a 16x16 matrix) faster than the traditional implementation of the Armenian shuffles and the PHT. A naive implementation of multiplying the vector with each column of the matrix would have taken 16 multiplication operations and 16 add operations for each column: 32 operations for each column, yielding 512 operations for the entire matrix (plus load and store operations). Such an implementation is slower than the traditional one: We found that each PHT box consists of 7 operations, the Armenian shuffle consist of 32 operations. This yields 320 operations for the traditional implementation (32 PHT boxes and 3 Armenian shuffles).

However, a careful examination of the above matrix shows that we can do much better. A much faster implementation is possible because the matrix has a great deal of structure, and because all the coefficients are powers of 2.

Observe that every pair of consecutive columns, starting with the two leftmost columns, are identical in half of their coefficients. All other coefficients in the left column are equal to twice the value of the coefficients in the right column. This structure is very useful, since the result of multiplication of half the column can be used in both

[2]Except the lookup tables **e** and **l** and the key addition steps.

PIN Length (digits)	Time (seconds)
4	0.063
5	0.75
6	7.609
7	76.127

Table 3: Summary of results obtained running the last version on a Pentium IV 3Ghz HT computer.

columns. Furthermore, the product of the other coefficients can be calculated once and used for both columns, since they differ only by a factor of 2.

The fact that the coefficients are all powers of 2 is also helpful, since instead of using multiplication operations, the calculation is done using a shift left operation.

The next pseudo code depicts the calculation procedure for two columns. Note the saving in shift operations, done by arranging the add operation in an appropriate manner. The input vector is denoted by $X = (x_0, ..., x_{15})$, and we show the calculation of the outputs y_0 and y_1:

$$\begin{aligned}
h_1 &= x_1 + x_2 + x_3 + x_6 + x_7 + \\
&\quad 2(x_0 + x_5 + 2(x_4)) \\
h_2 &= x_8 + x_9 + x_{11} + \\
&\quad 2(x_{10} + x_{12} + x_{13} + 2(x_{15} + 2(x_{14}))) \\
y_1 &= h_1 + h_2 \\
y_0 &= y_1 + h_2
\end{aligned}$$

How fast is the new implementation?

This implementation consists of 5 shift left operations, 16 add operations, 2 load operations and 2 store operations. This yields 25 operations per 2 columns, **200** operations for the entire matrix multiplication: 30% fewer than needed in the normal implementation.

4.5 Results

This subsection presents the cracking time of the five versions. All the versions were run on an old Pentium III 450MHz Personal Computer. For each version we tried several PIN sizes, ranging from 4 to 7 decimal digits.

Figure 7 compares the results obtained from all five versions. The Y axis denotes the running time in seconds (logarithmic scale), and the X axis denotes the number of decimal digits the PIN contains.

The final version improves the cracking speed by a factor of 10, and brings the time to crack a 4-digit PIN down to 0.27 sec. To gain some insight on how the attack improves with stronger hardware, we also ran our best attack version on a Pentium IV 3Ghz HT. On this computer we were able to crack a 4-digit PIN in 63 msec

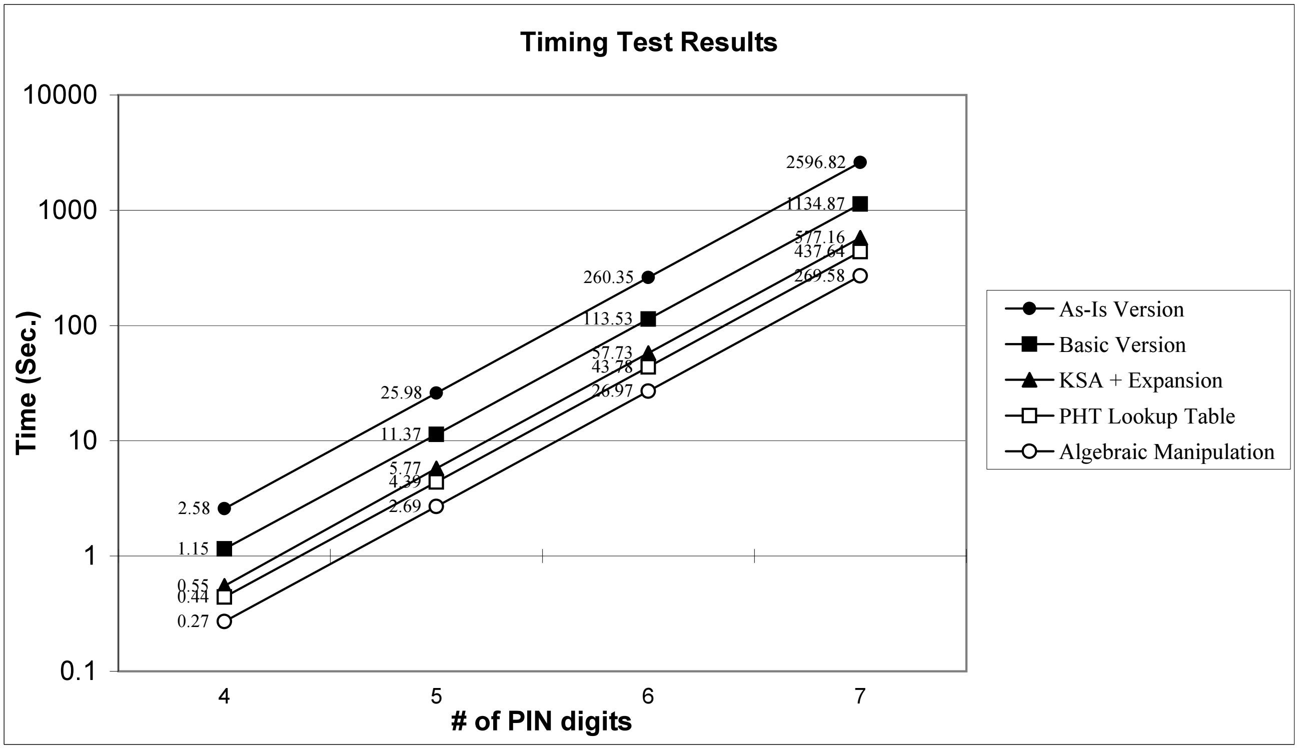

Figure 7: Timing results chart

(see Table 3) – 4 time faster than on the Pentium III. This makes the attack near-real-time.

5 The Re-Pairing attack

5.1 Background and motivation

This section describes an additional attack on Bluetooth devices that is useful when used in conjunction with the primary attack described in Section 3. Recall that the primary attack is only applicable if the attacker has eavesdropped on the entire process of pairing and authentication. This is a major limitation since the pairing process is rarely repeated. Once the link key K_{ab} is created, each Bluetooth device stores it for possible future communication with the peer device. If at a later point in time the device initiates communication with the same peer - the stored link key is used and the pairing process is skipped. Our second attack exploits the connection establishment protocol to force the communicating devices to repeat the pairing process. This allows the attacker to record all the messages and crack the PIN using the primary attack described in this paper.

5.2 Attack details

Assume that two Bluetooth devices that have already been paired before now intend to establish communication again. This means that they don't need to create the link key K_{ab} again, since they have already created and stored it before. They proceed directly to the *Authentication* phase (Recall Figure 3). We describe three different methods that can be used to force the devices to repeat the pairing process. The efficiency of each method depends on the implementation of the Bluetooth core in the device under attack. These methods appear in order of efficiency:

1. Since the devices skipped the pairing process and proceeded directly to the *Authentication* phase, the master device sends the slave an *AU_RAND* message, and expects the *SRES* message in return. Note that Bluetooth specifications allow a Bluetooth device to forget a link key. In such a case, the slave sends an *LMP_not_accepted* message in return, to let the master know it has forgotten the link key (see subsection 4.2.1.2 *"Claimant has no link key"* of Part C of Vol 2 of [Blu03]). Therefore, after the master device has sent the *AU_RAND* message to the slave, the attacker injects a *LMP_not_accepted* message toward the master. The master will be convinced that the slave has lost the link key and pairing

will be restarted (see subsection 5.1 *"AUTHENTICATION"* of Part C of Vol 3 of [Blu03]). Restarting the pairing procedure causes the master to discard the link key (see subsection 6.5 *"BONDING"* of Part C of Vol 3 of [Blu03]). This assures pairing must be done before devices can authenticate again.

2. At the beginning of the *Authentication* phase, the master device is supposed to send the *AU_RAND* to the slave. If **before** doing so, the attacker injects a *IN_RAND* message toward the slave, the slave device will be convinced the master has lost the link key and pairing is restarted. This will cause the connection establishment to restart.

3. During the *Authentication* phase, the master device sends the slave an *AU_RAND* message, and expects a *SRES* message in return. If, after the master has sent the *AU_RAND* message, an attacker injects a random *SRES* message toward the master, this will cause the *Authentication* phase to restart, and repeated attempts will be made (see subsection 5.1 *"REPEATED ATTEMPTS"* of Part H of Vol 2 of [Blu03]). At some point, after a certain number of failed authentication attempts, the master device is expected to declare that the authentication procedure has failed (implementation dependent) and initiate pairing (see subsection 5.1 *"AUTHENTICATION"* of Part C of Vol 3 of [Blu03]).

The three methods described above cause one of the devices to discard its link key. This assures the pairing process will occur during the next connection establishment, so the attacker will be able to eavesdrop on the entire process, and use the method described in Section 3 to crack the PIN.

In order to make the attack "online", the attacker can save all the messages transferred between the devices after the pairing is complete. After breaking the PIN (0.06-0.3 sec for a 4 digit PIN), the attacker can decode the saved messages, and continue to eavesdrop and decode the communication on the fly. Since Bluetooth supports a bit rate of 1 Megabit per second (see Part A of Vol 1 of [Blu03]), a 40KB buffer is more than enough for the common case of a 4 digit PIN.

Notes:

1. The Bluetooth specification does allow devices to forget link keys and to require repeating the pairing process. This fact makes the re-pairing attack applicable.

2. Re-Pairing is an active attack, that requires the attacker to inject a specific message at a precise point in the protocol. This most likely needs a custom Bluetooth device since off-the-shelf components will be unable to support such behavior.

3. If the slave device verifies that the message it receives is from the correct *BD_ADDR*, then the attack requires the injected message to have its source *BD_ADDR* "spoofed" - again requiring custom hardware.

4. If the attack is successful, the Bluetooth user will need to enter the PIN again - so a suspicious user may realize that his Bluetooth device is under attack and refuse to enter the PIN.

6 Countermeasures

This section details the countermeasures one should consider when using a Bluetooth device. These countermeasures will reduce the probability of being subjected to both attacks and the vulnerability to these attacks.

Since Bluetooth is a wireless technology, it is very difficult to avoid Bluetooth signals from leaking outside the desired boundaries. *Therefore, one should follow the recommendation in the Bluetooth standard and refrain from entering the PIN into the Bluetooth device for pairing as much as possible*. This reduces the risk of an attacker eavesdropping on the pairing process and finding the PIN used.

Most Bluetooth devices save the link key (K_{ab}) in non-volatile memory for future use. This way, when the same Bluetooth devices wish to communicate again, they use the stored link key. However, there is another mode of work, which requires entering the PIN into both devices every time they wish to communicate, even if they have already been paired before. *This mode gives a false sense of security!* Starting the pairing process every time increases the probability of an attacker eavesdropping on the messages transferred. We suggest not to use this mode of work.

Finally, the PIN length ranges from 8 to 128 bits. Most manufacturers use a 4 digit PIN and supply it with the device. Obviously, customers should demand the ability to use longer PINs.

7 Conclusion

This paper describes the implementation of an attack on the Bluetooth security mechanism. Our results show that using algebraic optimizations, the most common Bluetooth PIN can be cracked within less than 0.06-0.3 seconds. If two Bluetooth devices perform pairing in a hostile area, they are vulnerable to this attack.

References

[Arm02] Frederik 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.

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

[Blu04] Bluejackq. http://www.bluejackq.com/, 2004.

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

[Flu02] Scott R. Fluhrer. Improved key recovery of level 1 of the Bluetooth encryption system. Cryptology ePrint Archive, report 2002/068, available from http://eprint.iacr.org/2002/068/, 2002.

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

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

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

[Lau03] Adam Laurie. Serious flaws in Bluetooth security lead to disclosure of personal data. http://www.bluestumbler.org, 2003.

[LV04] Y. Lu and S. Vaudenay. Faster correlation attack on Bluetooth keystream generator E0. In *Advances in Cryptology – CRYPTO'04, LNCS 3152*, pages 407–425. Springer-Verlag, 2004.

[LW05] Ophir Levy and Avishai Wool. A uniform framework for cryptanalysis of the Bluetooth E_0 cipher. Cryptology ePrint Archive, Report 2005/107, 2005. http://eprint.iacr.org/2005/107.

[MKK98] J. L. Massey, G. H. Khachatrian, and M. K. Kuregian. SAFER+. In *Proc. First Advanced Encryption Standard Candidate Conference*. National Institute of Standards and Technology (NIST), 1998.

[Reh03] Gregory Rehm. 802.11b homebrew WiFi antenna shootout. http://www.turnpoint.net/wireless/has.html, 2003.

[Whi03] Ollie Whitehouse. War nibbling: Bluetooth insecurity. http://www.atstake.com/research/reports/acrobat/atstake_war_nibbling.pd%f, 2003.

[Whi04] Ollie Whitehouse. Bluetooth: Red fang, blue fang. CanSecWest/core04. Available from http://www.cansecwest.com/csw04/csw04- Whitehouse.pdf, April 2004. Vancouver, CA.

A Detailed specifications of SAFER+

A.1 SAFER+ Key Scheduling Algorithm

In continuation to section 2.2.1, the key scheduling algorithm uses 16 constant **Bias vectors**. The **Bias vectors**, denoted B_2 to B_{17}, are derived from the following equation:

$$B_c[i] \;=\; ((45^{(45^{17c+i+1} \bmod 257)} \bmod 257) \bmod 256), \quad \text{for } i = 0, .., 15.$$

One bias vector is used in each step, except for the first step. Note that the first step doesn't contain the cyclic rotate. Figure 8 describes the entire process of the key scheduling algorithm in SAFER+.

A.2 SAFER+ modified version

Besides using SAFER+ as is, Bluetooth uses a slightly modified version of SAFER+. This modified version is identical to the original SAFER+ implementation, only it also combines the input of SAFER+'s round 1 to the input of round 3: Some bytes are xored and some are added. This combination is done to make the modified version non-invertible. Figure 9 describes how the input of round 1 is combined with the input of round 3.

As stated before, all of the algorithms used during Bluetooth pairing and authentication process, use SAFER+ as is, or the modified version of SAFER+. In the remainder of this paper, SAFER+ is denoted as A_r, and the modified version of SAFER+ is denoted as $A_r^{'}$. Next subsections describe how E_{22}, E_{21}, E_1 are implemented using SAFER+.

B SAFER+ Based Algorithms

B.1 Inner design of E_{22}

As described in subsection 2.1, E_{22} is used to generate the initialization key. The inputs used are:

1. a *BD_ADDR* (48 bits long).
2. the PIN code and its length L.
3. a 128 bit random number *IN_RAND*.

At first, the PIN and the *BD_ADDR* are combined to create a new word: if the PIN contains less than 16 bytes, some of the *BD_ADDR* bytes are appended to the PIN. If the PIN is less than 10 bytes long, all bytes of *BD_ADDR* shall be used. Let **PIN'** denote the new word created, and **L'** denote the number of bytes the new word contains. Now, if L' is less than 16, the new word is cyclic expanded till it contains 16 bytes. Let **PIN"** denote this

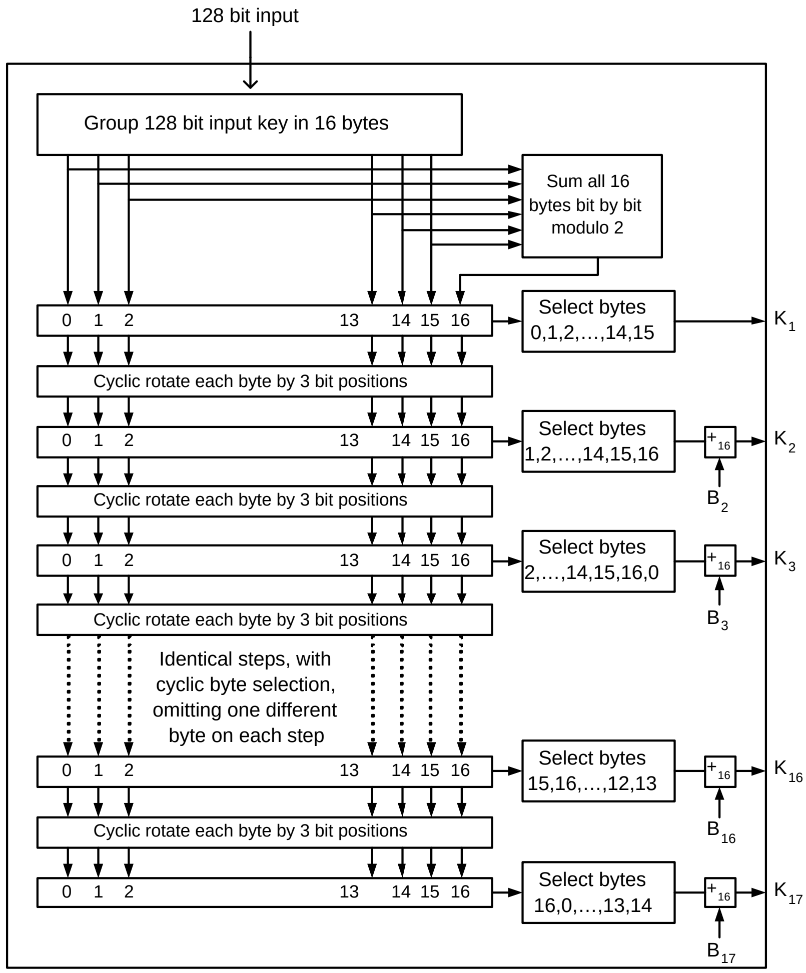

Figure 8: Inner design of SAFER+'s key scheduling algorithm

second new word. PIN" is used as the 128 bit input key of $A_r^{'}$. *IN_RAND* is used as the 128 bit input data, after xoring the most significant byte with L'. Figure 10 describes the inner design of E_{22}.

B.2 Inner design of E_{21}

As described in subsection 2.1, E_{21} is used to generate the link key. The inputs used are:

1. a *BD_ADDR* (48 bits long).
2. a 128 bit random number *LK_RAND*.

At first, the *BD_ADDR* is cyclic expanded to form a 128 bit word which is used as the input data of $A_r^{'}$. The key used for $A_r^{'}$ consists of the 128 bit random number *LK_RAND*, after xoring its most significant byte with 6 (result denoted ***LK_RAND'***). Figure 11 describes the inner design of E_{21}.

B.3 Inner design of E_1

As described in subsection 2.1, E_1 is used to perform mutual-authentication. The inputs used are:

1. A random word AU_RAND_A.
2. The link key K_{ab}.
3. a *BD_ADDR* (48 bits long).

The inner design of E_1 contains both A_r and $A_r^{'}$. The link key is used twice. Once, it is supplied as is for the key input of A_r. Later, it goes through a transformation denoted **Offset** and supplied as the key input of $A_r^{'}$. The "Offset" transformation consists of adding and xoring its bytes with some constants. For the full description of this transformation see page 778 of part H of Vol 2 of [Blu03].
As for the *BD_ADDR*, it is cyclic expanded to form a 128 bit word denoted *BD_ADDR'*. The inner design of E_1 is depicted in figure 12.

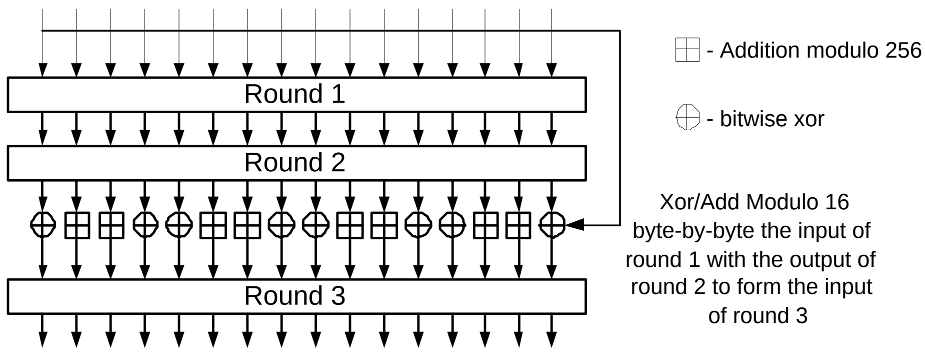

Figure 9: Combining input of round 1 with round 3 in SAFER+ modified version

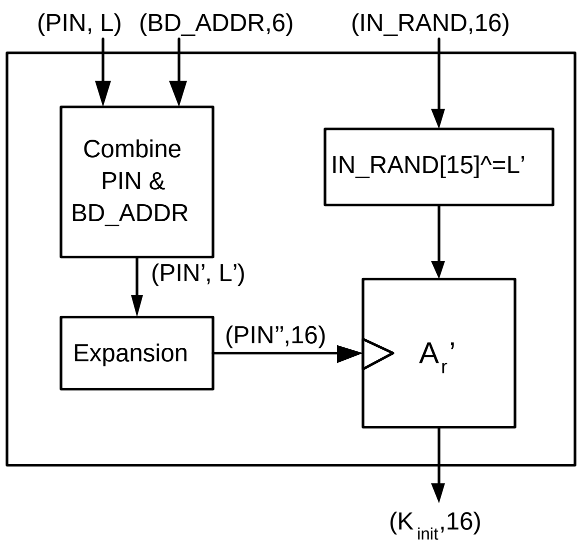

Figure 10: Inner design of E_{22}

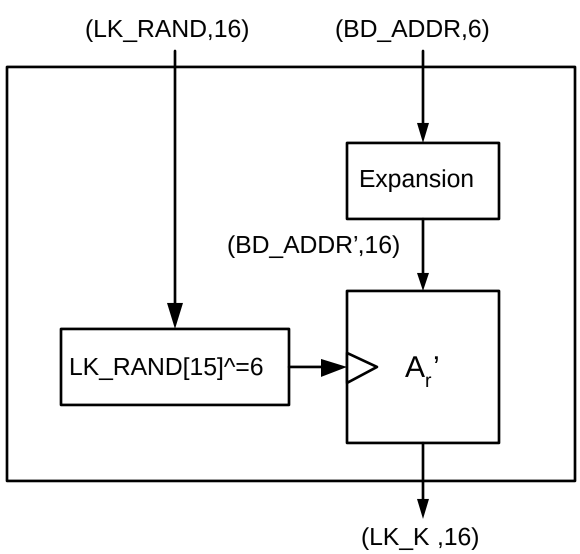

Figure 11: Inner design of E_{21}

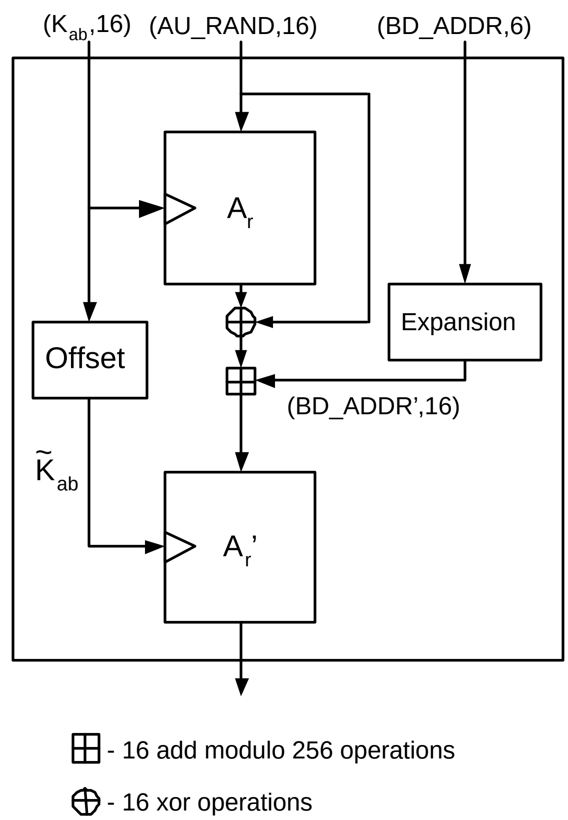

Figure 12: Inner design of E_1