San Jose State University
SJSU ScholarWorks

Master's Projects | Master's Theses and Graduate Research

2006

Bluetooth Security Protocol Analysis and Improvements

Chi Shing Lee
San Jose State University

Follow this and additional works at: http://scholarworks.sjsu.edu/etd_projects

Recommended Citation
Lee, Chi Shing, "Bluetooth Security Protocol Analysis and Improvements" (2006). *Master's Projects*. 122.
http://scholarworks.sjsu.edu/etd_projects/122

This Master's Project is brought to you for free and open access by the Master's Theses and Graduate Research at SJSU ScholarWorks. It has been accepted for inclusion in Master's Projects by an authorized administrator of SJSU ScholarWorks. For more information, please contact scholarworks@sjsu.edu.

Bluetooth Security Protocol Analysis and Improvements

A Writing Project

Presented to

The Faculty of the Department of Computer Science

San Jose State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Computer Science

By

Chi Shing Lee

May 2006

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

_____

Dr. Mark Stamp

_____

Dr. Melody Moh

_____

Dr. David Blockus

_____

APPROVED FOR THE UNIVERSITY

ABSTRACT

Since its creation, Bluetooth has transformed itself from a cable replacement technology to a wireless technology that connects people and machines. Bluetooth has been widely adapted on mobile phones and PDAs. Many other vendors in other industries are integrating Bluetooth into their products. Although vendors are adapting to the technology, Bluetooth hasn't been a big hit among users. Security remains a major concern. Poor implementation of the Bluetooth architecture on mobile devices leads to some high profiled Bluetooth hacks. Weak security protocol designs expose the Bluetooth system to some devastating protocol attacks.

This paper first explores four Bluetooth protocol-level attacks in order to get deeper insights into the weakness of the Bluetooth security design. It then proposes enhancements to defense against those attacks. Performance comparison will be given based on the implementation of those enhancements on a software based Bluetooth simulator.

TABLE OF CONTENT

1 Introduction……. 6

2 History …… 8

3 Technology ……. 8

4 Bluetooth Security Architecture ……. 9

4.1 Device Modes …….. 9

4.2 Security Modes …… 9

4.3 Key Managements …… 10

4.3.1 Initialisation Keys …….. 10

4.3.2 Combination / Unit Keys …… 11

4.3.3 Master Key ……. 11

4.3.4 Encryption Key …….. 12

4.4 SAFER+ …… 12

4.5 Hash Functions …….. 14

4.5.1 E22 …….. 14

4.5.2 E21 …….. 15

4.5.3 E1 …… 17

4.5.4 E3 …… 18

4.6 Pairing …….. 19

4.7 Authentication ……23

4.8 Encryption ……. 25

5 Research …….. 26

5.1 Environment …….. 27

5.2 Attacks …….. 29

5.2.1 Passive PIN cracking …….. 29

5.2.2 Active PIN cracking…… 33

5.2.3 Denial-of-Service Attack …… 36

5.2.4 Message Replay Attack ……. 38

6 Enhancements …… 41

6.1 Online/offline PIN attacks ……... 41

6.1.1 Password-based Encrypted Key Exchange (PW-EKE) ……. 42

A. Overview ……. 42

B. PW-EKE against different attack scenarios…….. 43

i) Man-In-the-Middle attack……... 43

ii) Brute-Force PIN attack…….. 44

iii) Brute-force attack against A & B …… 45

C. Pros and Cons …….. 45

6.1.2 MANA III variant (MANual Authentication III) …… 45

A. Overview …….. 45

B. MANA III variant against different attack scenarios …….. 50

i) MiM attack on the MANA III …… 50

ii) Diffie-Hellman MiM attack …….. 52

iii) Passive Brute-Force attack on SR …… 52

C. Pros and Cons …….. 53

6.2 Denial-Of-Service Attack ……. 53

6.3 Encryption Replay Attack …… 54

7 Conclusion …….. 55

8 References …….. 56

1. INTRODUCTION

Bluetooth, a short ranged wireless technology, was invented back in 1996. It was originally designed to replace clumsy cables that connect computers. However, the energy efficient nature of Bluetooth's design makes Bluetooth a practical technology to be applied on small portable mobile devices. The fact that Bluetooth design requires no state of the art components makes it a low cost wireless solution that can be widely afforded by the public [2, 11, 19]. Couple years ago, it was being seen as one of the technologies that will revolutionize how mobile devices connects to each other. Also, the fact that lots of big telecommunication moguls and software giants have made huge investments into Bluetooth projects makes Bluetooth a promising technology.

Bluetooth has been adapted by many different industries. Nowadays, lots of mobile phones on the market are equipped with Bluetooth which allows them to synchronize their data, such as phone books, wirelessly over a short range with other mobile phones, computers, and handheld devices. According to IDC research, about 13 percent of mobile phones shipped in the United States have Bluetooth. The number will grow to about 53 percent globally and 65 percent in United States by 2008 [21]. Some of the high-ended car models even have keyless entry and ignition that utilize Bluetooth. Some models of BMW have built-in Bluetooth hands-free system [14]. Microsoft has also started to have Bluetooth support after its XP SP1 release.

Although the Bluetooth technology can now be seen everywhere in our daily lives, it has not gain any significant popularity from its users over the past few years. Lots of people don't even know what Bluetooth is. Vendors also haven't aggressively push Bluetooth out to the market. Most mobile phones have their Bluetooth features

turned off by default so that only knowledge users will turn them on. Mobile phone vendors recommend users to turn off the Bluetooth features on their mobiles when they are not in use in order to minimize the risk of being attack by hackers. Bluetooth designers also recommend people not to pair Bluetooth devices in public places. If Bluetooth is such a great technology, why do vendors turn Bluetooth features off on their mobile phones by default? Why can't mobile users leave their Bluetooth connections on 24 hours a day? The answer lies in the fact that the Bluetooth security design is still insufficient for many applications.

Over the past few years, many security issues on Bluetooth have surfaced. During 2004, the famous Cabir worms that target mobile phones spread themselves through Bluetooth connections. British government required members of Parliament to disable the Bluetooth functions on their mobile phones [43]. Typing the words "bluetooth hack" on Google results in numerous links about all kinds of Bluetooth attacks. The two famous Bluetooth attacks are Bluejack and Bluesnarf. Bluejack is not exactly a hack. It does no real damage to its victims. A bluejacker merely abuses the "Name" field, which is long enough to embed a message, on a Bluetooth handshake packet to send anonymous messages to victims' devices. Bluejackers cannot steal information from their victims. Nor does the Bluejack attack allow attackers to take control of victims' devices [12, 13, 15]. The Bluesnarf attack, on the other hand, allow attackers to steal confidential data, such as phone books, calendars, images, pins, etc., from victims' phones [13]. Bluetooth profiling allows attackers to keep track of victims' locations because each Bluetooth device is a transceiver with a unique MAC address. Carrying a Bluetooth enabled mobile phone becomes tagging oneself with a RFID tag. All these issues are painting a rather negative picture for Bluetooth.

2. HISTORY

Bluetooth was named after the 10th century Danish King Harald Bluetooth. It was originally developed by Erission Mobile Communication. On 1998, a join initiative from couple big telecommunication giants gave birth to the Bluetooth Special Interest Group (Bluetooth SIG). The main goal of the Bluetooth SIG is to standardize and regulates the Bluetooth technology. The number of members of the Bluetooth SIG grew from a handful of companies, such as Ericsson, Nokia, Intel, IBM, in 1998 to more than 3000 today [40].

3. TECHNOLOGY OVERVIEW

Bluetooth data transmit on the unlicensed 2.4GHz ISM band. Bluetooth uses a frequency-hopping scheme in order to minimize the interferences with other technologies and applications such as 802.11, microwave ovens, cordless phones, etc. The connection range of off-the-shelf Bluetooth devices vary from 10 meters to 100 meters. Their data rate varies from 1Mbps to 2Mbps. Each Bluetooth device has a globally unique 48bit MAC address. The first 24 bits of the Bluetooth address is vendor specific. Figure 3.1 shows a typical Bluetooth address.

Bluetooth is an ad hoc networking technology in which no fix infrastructure (e.g. LAN) exists. Connections between Bluetooth devices are created “on the fly”. There is a master and a slave in each Bluetooth connection. A Bluetooth master can have up to 7 active slaves and unlimited passive (parked) slaves. Active slaves are devices that are in sync with the master and are ready to communicate. A master and its associated slaves form a piconet. A scatternet is formed by two or more piconets

that share common Bluetooth nodes. Figure 3.2 shows the different types of Bluetooth topology.

Bluetooth defines two procedures for establishing a connection between two Bluetooth devices. A Bluetooth device first uses the inquiry procedure to discover other close-by devices. It then uses the paging procedure to establish a connection with a target device. Two Bluetooth nodes are considered to be in sync when they share the same clock value and frequency-hopping pattern.

00:14:9A:C9:20:10

Figure 3.1. A Bluetooth MAC address

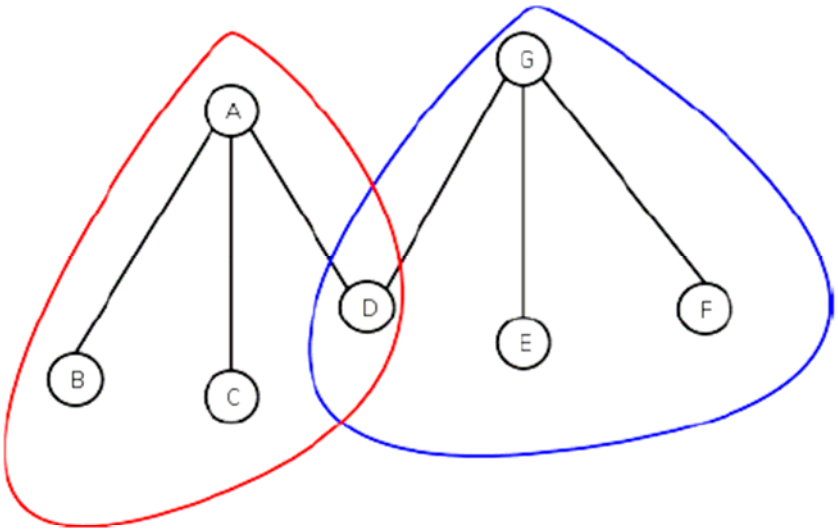

Figure 3.2. A scatternet consisting of two piconets

4. BLUETOOTH SECURITY ARCHITECTURE

4.1 DEVICE MODES

Bluetooth specification defines two device modes to control the visibility and availability of Bluetooth devices. A device is in discoverable mode if it responses to inquiries from other devices. Otherwise, it is in a non-discoverable mode. A device is in a connectable mode if it responses to paging requests from other devices.

Otherwise, it is in non-connectable mode. Paging will be explained further in the later sections.

4.2 SECURITY MODES

Bluetooth specification defines three security modes to control when and where authentication and encryption occur. In security mode 1, no authentication and encryption will be initialized on any connections. This mode is being provided mostly for Bluetooth devices where security is not necessary and is thus considered to be an overhead. Bluetooth wireless mouse is one of those applications. Mode 2 is a policy-based service level security mode. Security procedures are initialized only after the connection establishment on the L2CAP level. By assigning different security policies and trust levels to each connection, a Security manager control access to a device and the services that the device offers. In essence, this security mode provides authentication, confidentially, and authorization. Security mode 3 provides link level security. It is a build-in security mechanism that is transparent to the upper application layers [41].

4.3 KEY MANAGEMENT

Bluetooth security architecture is based on the symmetric key cryptography where two Bluetooth devices share a common link key for authentication and encryption. Figure 4.3.1 shows the Bluetooth key structure.

4.3.1 Initialization Key

Bluetooth specification defines a pairing process for two Bluetooth devices that have never establish any connection before to derive a common key for

authentication and encryption. The initialization key (K_{init}) is the first key being generated in the pairing process. It is being used to derive combination / unit keys later on in the pairing process. Once a combination / unit key is derived, the initialization key will be discarded. Note that the strength of this key solely relies on a 4 to 16 bytes PIN.

4.3.2 Combination / unit key

Combination keys (K_{ab}) and unit keys (K_a) are semi-permanent in a sense that devices store them permanently unless the they are being updated through the link key update procedures or the broadcast encryption scheme. These keys can be reused in multiple sessions by the devices that share them. The main difference between unit keys and combination keys is that two different random numbers, one from the master and the other from the slave, are used to derive combination keys. In other words, combination keys are unique for each connection. On the other hand, unit keys are generated by a single device and can be shared by different Bluetooth connections. Due to the inherited insecure nature of unit keys, the usage of unit keys is being depreciated.

4.3.3 Master key

Sometimes it is desirable for a master of a piconet to encrypt broadcast traffic. But using combination keys to encrypt broadcast traffic involves the overhead of encrypting the same packet using different combination keys associated with different slaves. Bluetooth specification defines shared master keys to allow piconet masters to encrypt broadcast traffic. Copies of a single master key are distributed to all the slaves within a piconet. The master then uses the master key to encrypt payloads and

broadcast them to all the slaves. As a result, the master has avoided the overhead of using combination keys to encrypt broadcast traffic.

4.3.4 Encryption Key

Encryption keys (K_c) are derived from the current link keys and are automatically updated each time the devices enter the encryption mode. K_C is used to generate cipher stream K_{Cipher} that in turn will be XORed with payloads.

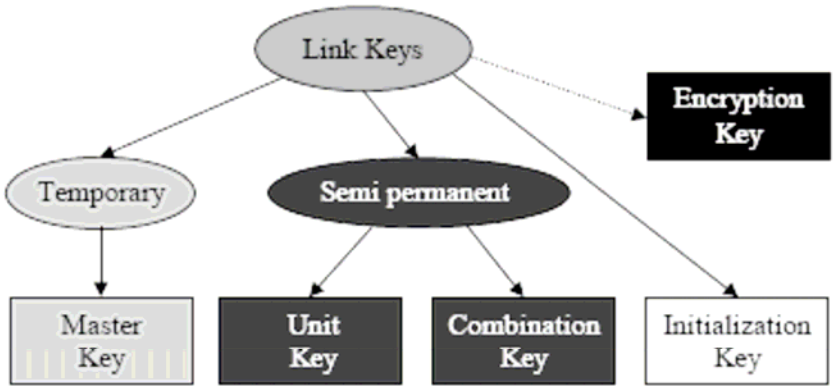

Figure 4.3.1. Bluetooth key structure

4.4 SAFER+

SAFER+, invented by Cylink Corporation, is a modified version of SAFER block cipher. It was one of the 15 submissions of AES [10]. Bluetooth security architecture uses SAFER+ in all key generating hash functions. SAFER+ has three main parts:

1. A Key Scheduling Algorithm (KSA) which takes a 128 bit key and generates 17 different sub-keys (K_1 to K_{17} in Figure 4.4.1)
2. Eight identical rounds. Each round takes two keys from KSA and a 128 bit input value to generate a 128-bit output. The inner design of each SAFER+ round is showed in Figure 4.4.2.
3. The 128-bit output from the last round is XORed with K_{17} to produce the final 128 bit output

Note that the Bluetooth design uses two slightly different versions of SAFER+ (Ar and Ar'). Ar represents the original design of SAFER+, which is being shown in Figure 4.4.2. Ar' differs with Ar only in the design of Round 3. In Ar', the 128-bit input value of round 3 is XORed with the input value of round 1 such that Ar' becomes non-invertible [2]. Figure 4.4.3 shows the inner design of Ar'.

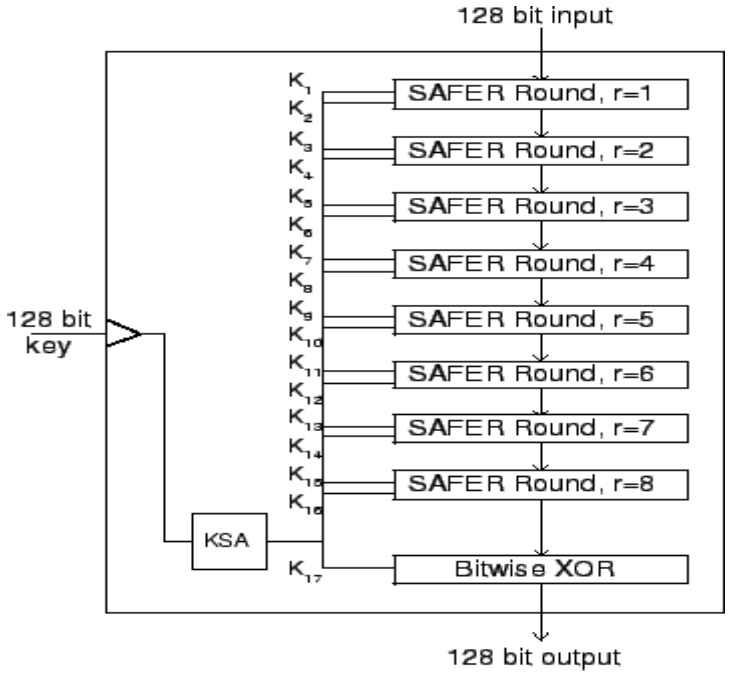

Figure 4.4.1 The inner design of SAFER+ block cipher (Ar) [1]

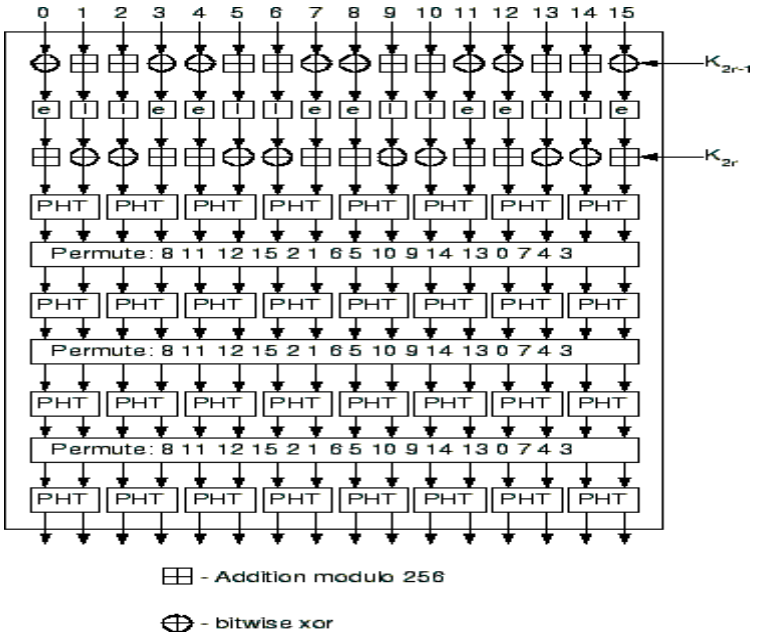

Figure 4.4.2 The inner design of a SAFER+ round [1]

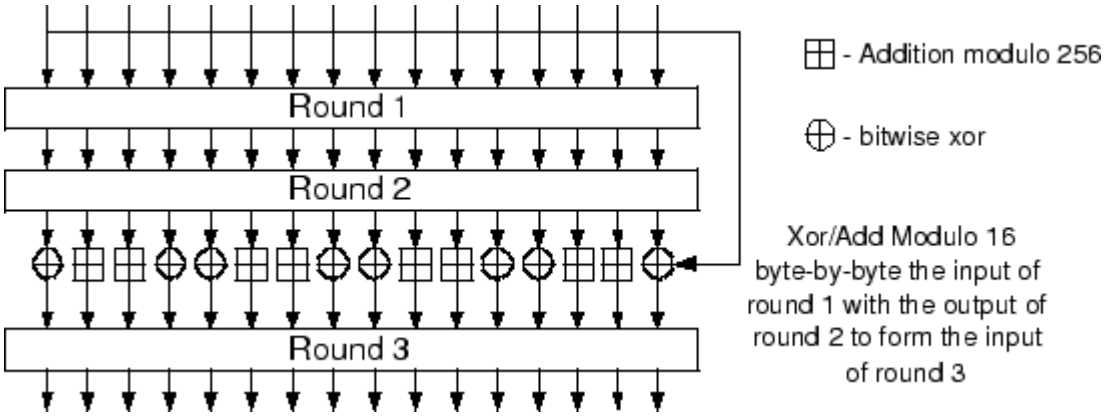

Figure 4.4.3 The inner designed of the slightly modified SAFER+ (Ar') [1]

4.5 HASH FUNCTIONS

Four hash functions are used in pairing, authentication, and encryption. The heart of all four functions is a SAFER+ block cipher.

4.5.1 E22

Bluetooth design uses E22 to generate initialization keys (K_{init}). The equation that depicts the design of E22 is shown in Figure 4.5.1. E22 takes a 48-bit Bluetooth address (BD_ADDR), a PIN, and a 128 bit random number (RAND) to generate a 128-bit $K_{init.}$ The maximum size of PIN is 16 bytes. If PIN's size (L) is less than 16 bytes, it will first combined with BD_ADDR to form PIN'. If PIN' is still less than 16 bytes, it will then be expanded cyclically to become a 16 byte PIN'' (or "X" in Figure 4.5.1). The 15th byte of RAND is XORed with the L', which is the lesser number between 16 (the maximum size of a PIN) and L + 6 (6 is the size BD_ADDR), to form Y. X and Y will then be feed into Ar' to create a 128-bit K_{init} [1,2].

$$E_{22}: \{0,1\}^{8L'} \times \{0,1\}^{128} \times \{1,2,\ldots,16\} \rightarrow \{0,1\}^{128}$$
$$(\mathrm{PIN}', \mathrm{RAND}, \mathrm{L}') \mapsto A'_r(X, Y)$$

$$PIN' = \begin{cases} PIN[0 \ldots L-1] \cup BD\_ADDR[0 \ldots \min\{5, 15-L\}], & L < 16, \\ PIN[0 \ldots L-1], & L = 16, \end{cases}$$

where

$$\begin{cases} X = \bigcup_{i=0}^{15} PIN'[i \pmod{L'}], \\ Y = RAND[0 \ldots 14] \cup (RAND[15] \oplus L'), \end{cases}$$

and $L' = \min\{16, L+6\}$ is the number of octets in PIN'.

Figure 4.5.1 Equations of E22 [2]

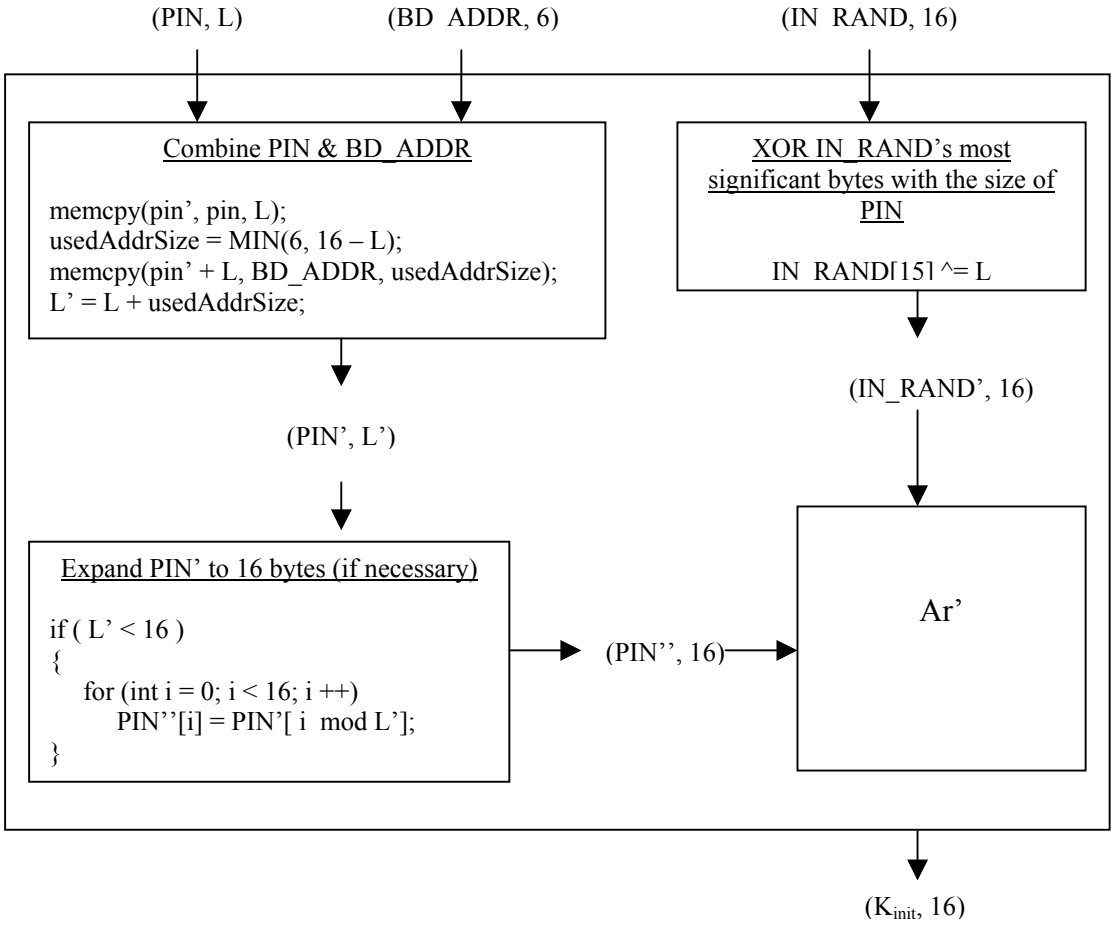

Figure 4.5.2 A graphical representation of E22

4.5.2 E21

Bluetooth design uses E21 to generate unit keys (LK_K_A/LK_K_B). The equation that depicts the design of E21 is shown in Figure 4.5.3. E21 takes a 128-bit

random number (RAND) and a 48-bit Bluetooth address (address) as its input. The 15th byte of RAND is XORed with a constant number 6 (the size of a Bluetooth address in byte) to form X. "address" is being cyclically expanded from 6 bytes to 16 bytes to form Y. X and Y are then being feed to Ar' to create a unit key. Two unit keys (LK_K_A and LK_K_B) will then be combined to form a combination key K_{ab} [1, 2].

$$E_{21}: \{0,1\}^{128} \times \{0,1\}^{48} \rightarrow \{0,1\}^{128}$$

$$(\mathrm{RAND}, \mathrm{address}) \mapsto A'_r(X, Y)$$

where

$$\begin{cases} X = \mathrm{RAND}[0 \ldots 14] \cup (\mathrm{RAND}[15] \oplus 6) \\ Y = \bigcup_{i=0}^{15} \mathrm{address}[i \pmod 6] \end{cases}$$

Figure 4.5.3 Equation of E21 [2]

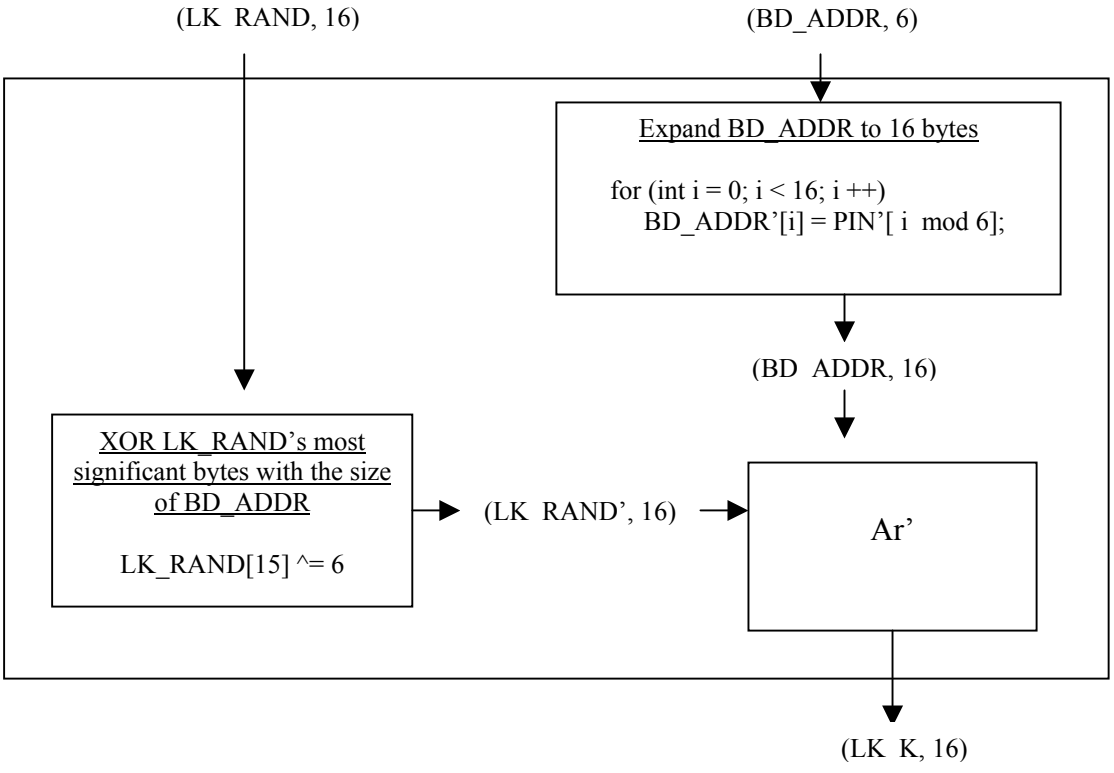

Figure 4.5.4 A graphical representation of E21

4.5.3 E1

Bluetooth design uses E1 to generate authentication responses (SRES). The equation that depicts the design of E1 is shown in Figure 4.5.5. E1 takes a 128-bit combination key (K), a 128-bit random number (RAND), and a Bluetooth address (address) as its inputs. RAND and K from Figure 4.5.6 are being feed to Ar. The 128-bit output is XORed with RAND and then added with a cyclically expanded address (Ar_out). Transforming K with an offset table will form K'. The complete offset table can be found in the Bluetooth specification. Ar_out and K' are being feed to Ar' to generate a 128 bit value. The first 32 bit of that value will become SRES. The rest of the 128-bit output value will become ACO [1, 2].

$$E_1:\ \{0,1\}^{128}\times\{0,1\}^{128}\times\{0,1\}^{48}\rightarrow\{0,1\}^{32}\times\{0,1\}^{96}$$
$$(K,\mathrm{RAND},\mathrm{address})\mapsto(\mathrm{SRES},\mathrm{ACO}),$$

$$\mathrm{SRES}=Hash(K,\mathrm{RAND},\mathrm{address},6)[0,\ldots,3]$$

$$Hash:\{0,1\}^{128}\times\{0,1\}^{128}\times\{0,1\}^{8\times L}\times\{6,12\}\rightarrow\{0,1\}^{128}$$
$$(K,I_1,I_2,L)\mapsto A'_r([\tilde{K}],[E(I_2,L)+_{16}(A_r(K,I_1)\oplus_{16}I_1)]),$$

and where

$$E:\{0,1\}^{8\times L}\times\{6,12\}\rightarrow\{0,1\}^{8\times 16}$$
$$(X[0,\ldots,L-1],L)\mapsto(X[i(\mathrm{mod\ L})]\ \text{ for } i=0...15),$$

Figure 4.5.5 Equations of E1 [2]

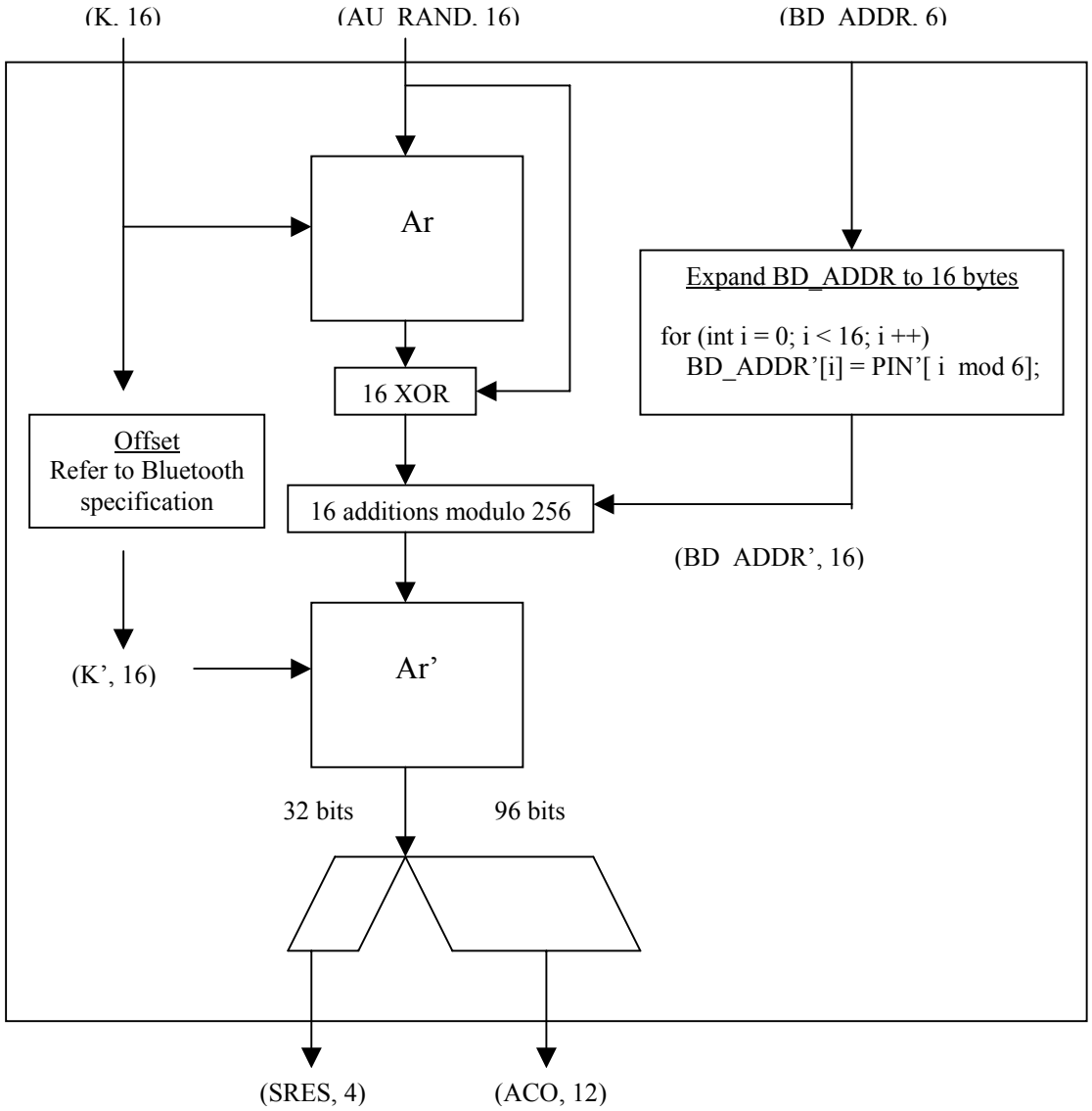

Figure 4.5.6 A graphical representation of E1

4.5.4 E3

Bluetooth design uses a hash function E3 to generate ciphering key K_c, which will then be used by system E0 to generate cipher streams for encrypting message payloads in encryption. The equations that depict the design of E1 are shown in Figure 4.5.7. K is the current link key. RAND is a 128bit random number that is generated by the master. Depends on the type of encryption (i.e. point to point or point to multi-points), COF is either the union of the master's address or the ACO generated by the previous authentication.

$$E_3:\ \{0,1\}^{128}\times\{0,1\}^{128}\times\{0,1\}^{96}\rightarrow\{0,1\}^{128}$$

$$(K, \mathrm{RAND}, \mathrm{COF}) \mapsto Hash(K, \mathrm{RAND}, \mathrm{COF}, 12)$$

$$\mathrm{COF} = \begin{cases} \mathrm{BD\_ADDR} \cup \mathrm{BD\_ADDR}, & \text{if link key is a master key} \\ \mathrm{ACO}, & \text{otherwise.} \end{cases}$$

Figure 4.5.7 Equations of E3 [2]

4.6 PAIRING

Before two Bluetooth devices can establish a connection and send data to each other, they have to go through a pairing procedure, which is essentially a process for creating a common key for authentication and encryption between two Bluetooth devices. The device that initializes the pairing is, by definition, the master of the whole process. The other device is considered to be the slave [1, 2]. Two Keys, K_{init} and $K_{ab,}$ are being generated from the process. Figure 4.6.1 and 4.6.2 show a simplified and a detailed pairing respectively.

An overview of the pairing process is described as follows:

Index:

Bluetooth Address – A 48 bit mac address that uniquely identify each individual Bluetooth device

BD_ADDR_A – The Bluetooth address of the master

BD_ADDR_B – The Bluetooth address of the slave

1. User A enters a PIN to the master (Device A) Bluetooth device
2. The master generates a 128 bit random number (IN_RAND)
3. The master uses IN_RAND along with the PIN and BD_ADDR_B to generate an initialization key (K_{init})

4. The master sends IN_RAND to the slave
5. User B enters the same PIN as User A did to the slave device
6. The slave uses the PIN, IN_RAND, and it's own address BD_ADDR_B to generate the same $K_{init.}$ At this point, both the master and the slave share the same initialization key.
7. The master generates a new 128 bit random number (LK_RAND_A)
8. The master uses LK_RAND_A along with BD_ADDR_A to generate a unit link key (K_a)
9. The master encrypts LK_RAND_A by using K_{init}
10. The master sends the encrypted LK_RAND_A to the slave
11. The slave decrypts the encrypted random number by using its own K_{init}
12. The slave generates K_a by using LK_RAND_A and BD_ADDR_A
13. The slave generates a new 128 bit random number LK_RAND_B
14. The slave uses LK_RAND_B along with BD_ADDR_B to generate K_b
15. At this point, the slave has both K_a and $K_{b.}$ It XOR two unit link keys to form a new 128 bit combination key K_{ab}
16. The slave encrypts LK_RAND_B by using K_{init}
17. The slave sends the encrypted LK_RAND_B to the master
18. The master decrypts the encrypted LK_RAND_B by using its own K_{init}
19. The master generates K_b by using LK_RAND_B and BD_ADDR_B.
20. At this point, the master has both K_a and $K_{b.}$ It XOR two unit link keys to form the same combination key K_{ab}

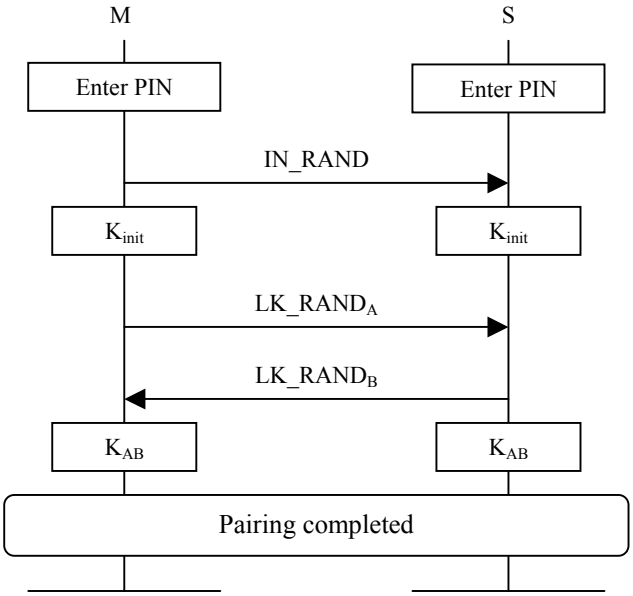

Figure 4.6.1 A simplified Bluetooth pairing protocol

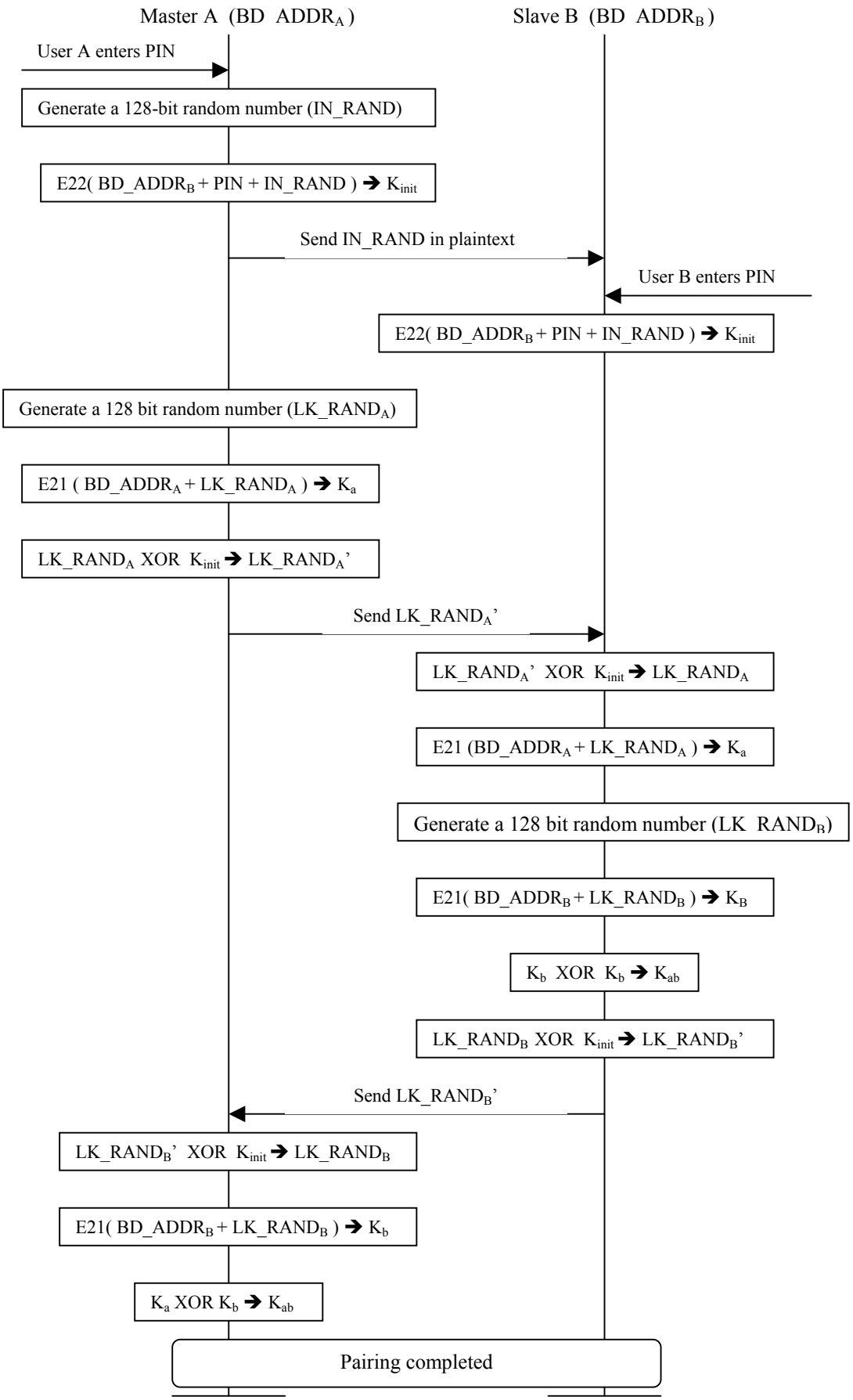

Figure 4.6.2 A detailed Bluetooth pairing protocol

4.7 AUTHENTICATION

Bluetooth security architecture uses a challenge-response authentication scheme. Figure 4.7.1 and 4.7.2 show a simplified and a detailed authentication respectively.

An overview of the authentication process is described as follows:

1. The master is the verifier. The slave is the claimant.
2. The master generates a 128 bit random number AU_RAND_A
3. The master uses AU_RAND_A along with K_{ab} and BD_ADDR_B to compute a 32 bit values $SRES_A$
4. The master sends AU_RAND_A to the slave as plaintext
5. The slave computes a 32 bit response $SRES_A$' using AU_RAND_A, K_{ab}, and BD_ADDR_B.
6. The slave sends the $SRES_A$' back to the master
7. The master compares the $SRES_A$' it received from the slave against $SRES_A$ to verify the validity of the slave's K_{ab}
8. Upon the success in verifying the validity of slave's K_{AB}, a new round of authentication begins. This time, the slave becomes the verifier. The master becomes the claimant
9. The slave generates a 128 bit random number AU_RAND_B
10. The slave uses AU_RAND_B along with K_{ab} and BD_ADDR_A to compute a 32 bit values $SRES_B$
11. The slave sends AU_RAND_B to the master
12. The master computes $SRES_B$' using AU_RAND_B, K_{ab}, and it's own address BD_ADDR_A.
13. The master sends $SRES_B$' back to the slave

14. The slave compares $SRES_B$' against $SRES_B$ to verify the validity of the master's K_{ab}
15. Upon the success in verifying the validity of master's $K_{AB,}$ a mutual authentication is completed

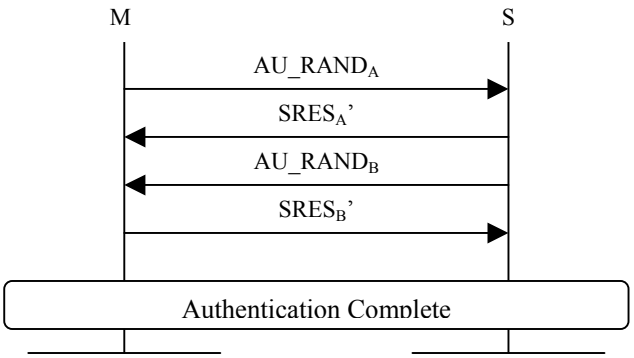

Figure 4.7.1 A simplified Bluetooth authentication protocol

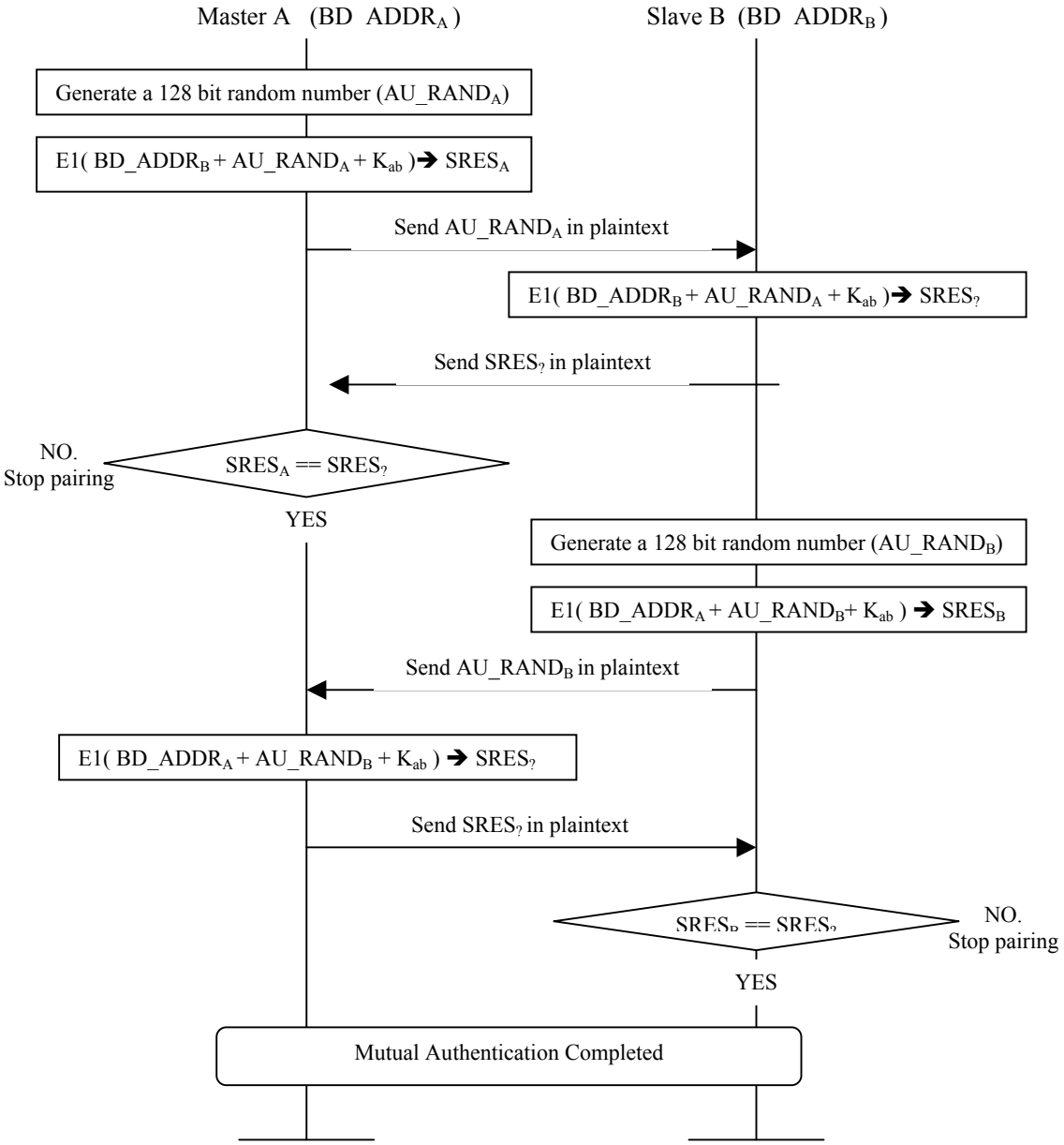

Figure 4.7.2 A detailed Bluetooth authentication protocol

4.8 ENCRYPTION

After at least one authentication has been performed, encryption can be used to protect message payloads. The master first negotiates the encryption key size with the slave. The master and the slave then derive the same ciphering key K_c. K_c will be used by the E0 system to generate cipher streams (K_{cipher}) for encrypting packet payloads. Figure 4.8.1 shows the encryption procedure.

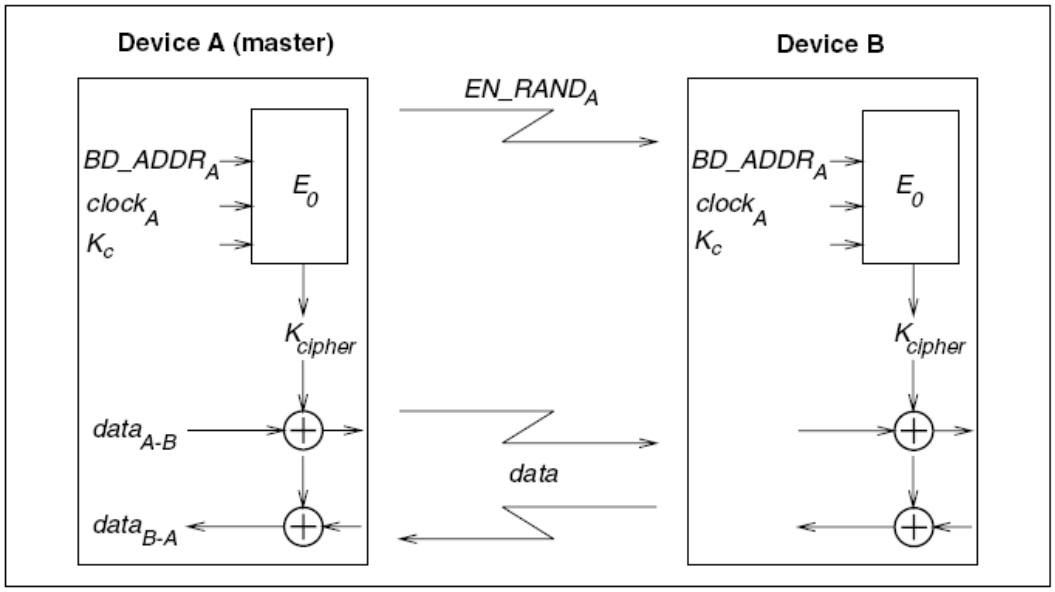

Figure 4.8.1 An overview of Bluetooth encryption protocol [1]

5 RESEARCH

This research first explores four Bluetooth security issues through the analysis and implementation of four attacks. The first attack is a passive PIN cracking attack. The attack attempts to use an offline brute-force approach to recover the secret PIN that is shared by two Bluetooth devices during their paring process. The second attack is an active version of the first attack. Instead of taking the PIN calculation offline, an attacker attempts to pair with a victim device repeatedly in a short period of time using different PINs until he recovered the secret PIN. Since those consecutive pairings happens in real time, speed will become a crucial factor in this attack. Thus, a PIN dictionary will be used along with this attack to enhance the PIN recovery speed. The third attack is a denial-of-service attack. The attack attempts to prevent legitimate users from connecting to a master Bluetooth device (e.g. a Bluetooth access point). The fourth attack is an encrypted message replay attack. The main goal of this attack is to make a victim do the same thing twice using some

previously captured messages. All these messages are encrypted. The attack does not need to know the current link key in order to carry out this attack.

After the analysis and implementations of those attacks, this research then suggests some security improvements to defense against those four attacks.

5.1 ENVIRONMENT

The research is based upon an open-sourced Bluetooth network simulator named UCBT, which stands for University of Cincinnati – Bluetooth [3]. There are two other open-sourced Bluetooth simulators available for download on the Internet. Bluehoc, which was developed by IBM in 1996, is the first generation of Bluetooth simulator [4]. A newer simulator is named Blueware, which is a project from couple MIT students [5]. Both Blueware and UCBT are built on top of Bluehoc. All three simulators incorporate the framework provided by NS-2, a well-known network simulator [7]. UCBT is chosen for this research because it is the most updated Bluetooth simulator that is designed based on the more widely adapted Bluetooth 1.1 and 1.2 specifications.

UCBT is designed for the Linux platform. It is written in C++. In this research, Linux Mandrake 10.0 [8] is being selected to host UCBT because it is notorious for the ease of its installation and configuration. Mandrake is installed as a virtual host operating system on VMWare [9] so that the research can be conducted in Windows' environment. The Bluetooth design incorporates SAFER+ as the core block cipher for couple encryption and key generation functions such as E22, E21, E1, etc. More detailed information regarding those functions is discussed in the previous sections. The codes for SAFER+ in UCBT are being extracted from a cryptographic

library named “LibTomCrypt” [8]. Table 5.1.1 provides a summary of the research environment.

UCBT is specifically designed to simulate the Baseband Bluetooth stack layer. The Bluetooth architecture divides into three distinct layers. A L2CAP layer, which stands for Logical Link Control and Adaptation Protocol, sits on the top of the stack. The main function of the L2CAP layer is to create and manage channels for the application layer, which is not considered to be a part of the Bluetooth architecture. A radio layer lies on the bottom of the stack. It contains a radio transceiver that transmits and receives Bluetooth packets. A baseband layer lies in between the L2CAP layer and the radio layer. It contains a scheduler that grants time slots for the L2CAP channels to send packets through the radio layer. It negotiates quality of services between Bluetooth entities. It is also responsible for encoding and decoding Bluetooth packets [2]. UCBT relies on NS-2 to provide the L2CAP layer that it needs. The radio layer and the physical wireless medium that allows Bluetooth devices to connect to each other are not being simulated [3].

One of the major challenges for using UCBT in this research is the fact that UCBT’s designers intentionally bypassed all the security aspects of the Bluetooth specification. In other words, the original UCBT package does not contain any modules for pairing, authentication, and encryption. As part of this research, pairing, authentication, and encryption (based on the Bluetooth 1.2 specification) have been added to the simulator.

In order to verify that all security modules have been implemented correctly, couple test samples from the Bluetooth 1.2 specification have been used. Figure 5.1.1 shows a sample input and its associated output for the E22 function. The inputs of E22 from the sample test data are a 128 bit random number (rand), a 16-byte pin

(PIN), and a 48-bit Bluetooth address (address). In the figure, "round 1" represents the input value for the first round in Ar'. Each SAFER+ round requires two keys. "Key [1]" and "Key [2]" represent the two input keys for the first round. The expected final key is represented by "Ka". The rest of the test sets can be found in Part G, Vol 2 of the Bluetooth 1.2 specification [2].

```
rand    :67ed56bfcf99825f0c6b349369da30ab
PIN     :7885b515e84b1f082cc499976f1725ce
round  1:67ed56bfcf99825f0c6b349369da30bb
Key [ 1]:7885b515e84b1f082cc499976f1725ce
Key [ 2]:72445901fdaf506beb036f4412512248
round  2:6b160b66a1f6c26c1f3432f463ef5aa1
Key [ 3]:59f0e4982e97633e5e7fd133af8f2c5b
Key [ 4]:b4946ec77a41bf7c729d191e33d459ab
```

Figure 5.1.1 A sample test data for E22 [2]

Virtual Machine Software	VMWare Workstation 5.0.0 build 13124
Host Operating System	Mandrake 10.0
Network Simulator	NS-2 version 2.27
Bluetooth Simulator	UCBT 0.9.8.2
SAFER+	LibTomCrypt 1.06
Compliers	gcc & g++

Table 5.1.1 Summary of all software used in the research

5.2 ATTACKS

5.2.1 Passive PIN cracking

This passive pin-cracking analysis was first being disclosed to the public by O. Whitehose at the CanSecWest '04 conference. At that time, only the attack framework and its performance analysis were discussed [20]. Two researchers, Yaniv Shaked and Avishai Wool, followed the lead and made a more detailed analysis of the

Bluetooth pin attack. They implemented a pin-cracking program along with couple speed improvements on their algorithm. They tested and evaluated their program against pins that are 4 to 7 digits long [1]. Since they did not release the source codes of their cracking program to the public, an independent implementation of the cracking program is included in this research. All Bluetooth pairing messages that are needed by the pin-cracking algorithm are summarized in Table 5.2.1.

The pin-cracking algorithm is a brute-force algorithm. It repeatedly generates different hypothetical PIN' and goes through a series of pairing and authentication steps to generate hypothetical SRES'. It then compares the hypothetical SRES' with $SRES_A$ and $SRES_B$ in order to recover the correct PIN [1]. Notice that the algorithm assumes that the attacker has successfully eavesdropped the entire pairing process and has retrieved all the necessary messages that are listed in Table 5.2.1. Figure 5.2.1 describes the complete pin-cracking process. Since the Bluetooth specification requires the length of the pins to be at least 4 digits, the crack program starts to enumerate all possible pin combinations from the pin "0000". Some performance improvement codes have been added to the SAFER+ that is being used by the crack program. Figure 5.2.2 shows the pseudo codes for the crack program. The following assumption has been made for the attack:

1. The attacker has eavesdropped the entire pairing process between the targets.
2. The following data are known prior to the pin cracking starts
    - The Bluetooth address of both master and slave (BD_ADDR_A and BD_ADDR_B)
    - All messages listed in Table 1
    - The internal designs of E22, E21, and E1

#	Src	Dst	Data	Length (bit)	Notes
1	Master	Slave	IN_RAND	128	Plaintext
2	Master	Slave	LK_RAND_A	128	XORed with Kinit
3	Slave	Master	LK_RAND_B	128	XORed with Kinit
4	Master	Slave	AU_RAND_A	128	Plaintext
5	Slave	Master	$SRES_A$	32	Plaintext
6	Slave	Master	AU_RAND_B	128	Plaintext
7	Master	Slave	$SRES_B$	32	Plaintext

Table 5.2.1 Messages used by the pairing process [1]

```
// Load all sniffed messages
in_rand               = getMsg(IN_RAND);          // in_rand for Kinit
encrypted_lk_rand_a   = getMsg(E_LK_RAND_A); // encrypted LK_RAND_A
encrypted_lk_rand_b   = getMsg(E_LK_RAND_B); // encrypted LK_RAND_B
au_rand_a             = getMsg(AU_RAND_A);    // authentication AU_RAND_A
au_rand_b             = getMsg(AU_RNAD_B);    // authentication AU_RAND_B
sres_expected_a     = getMsg(SRES_A);                 // response SRES_A
sres_expected_b     = getMsg(SRES_B);                 // response SRES_B
pin_found = false;

while(!pin_found)
{
        // Guess a new pin
        guess_pin = GetNewPin();

        // Initialisation key (Kinit)
        key_init = E22(guess_pin, slave_addr, in_rand);

        // Decrypt random numbers using Kinit
        lk_rand_a = XOR(encrypted_lk_rand_a, key_init);
        lk_rand_b = XOR(encrypted_lk_rand_b, key_init);

        // Generate unit keys Ka and Kb
        key_a = E21(master_addr, lk_rand_a);
        key_b = E21(slave_addr, lk_rand_b);

        // Generate combination key Kab
        key_ab = XOR(key_a, key_b);

        // Generate authentication response using
        // master-generated random number
        sres_a = E1(au_rand_a, slave_addr, key_ab);

        // Compare guessed authentication response with
        // the expected one
        if( sameResponse(sres_a, sres_expected_a))
        {
                // Generate authentication response using
                // slave-generated random number
                sres_b = E1(au_rand_b, master_addr, key_ab);

                // Compare guessed authentication response with
                // the expected one
                if( sameResponse(sres_b, sres_expected_b))
                        pin_found = true;
        }
}
```

Figure 5.2.2 Pseudo codes for the passive pin-cracking program

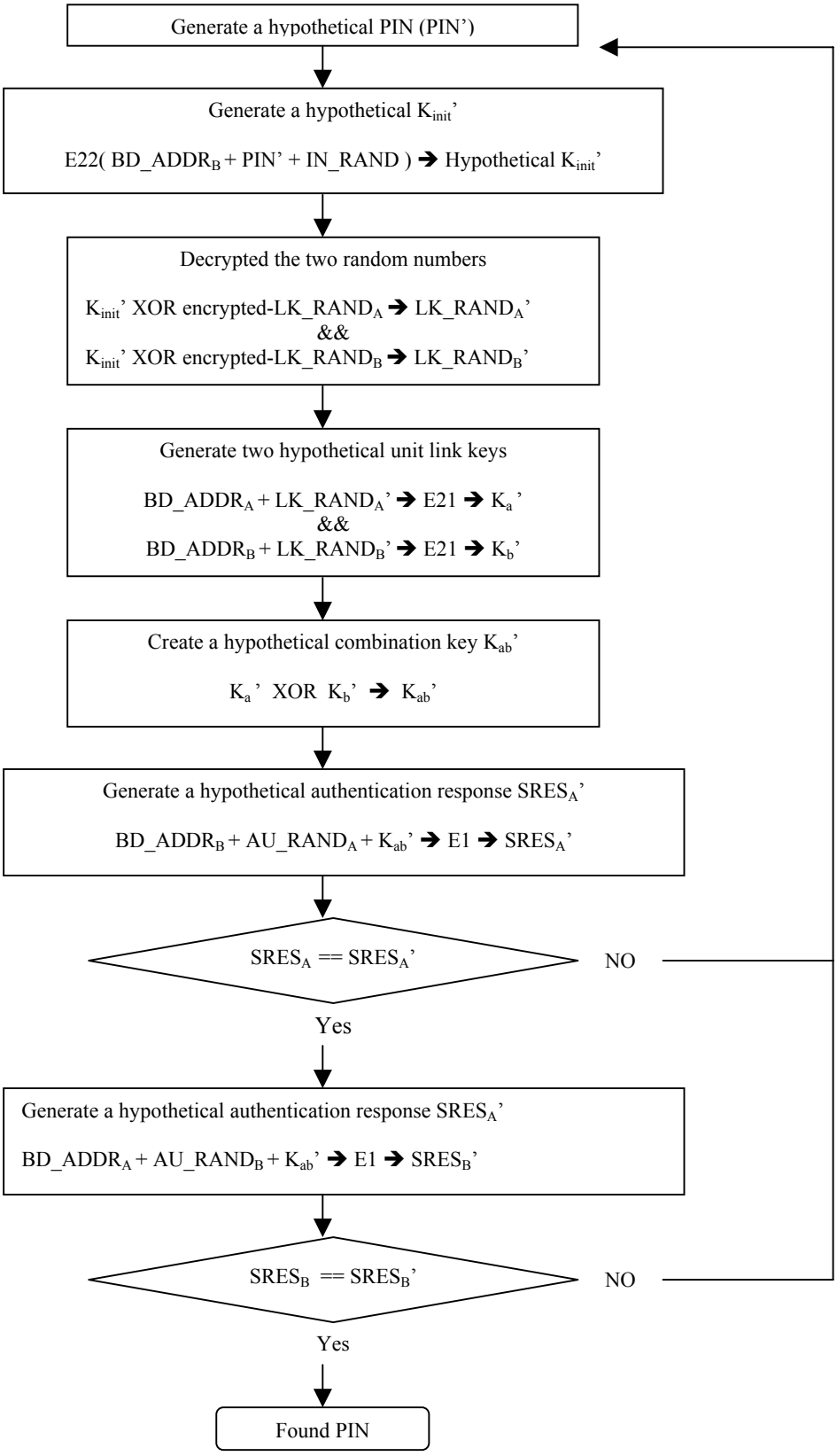

Figure 5.2.1 The passive PIN-cracking algorithm

The following are the results of running the pin-cracking program against messages that were encrypted with different pin sizes.

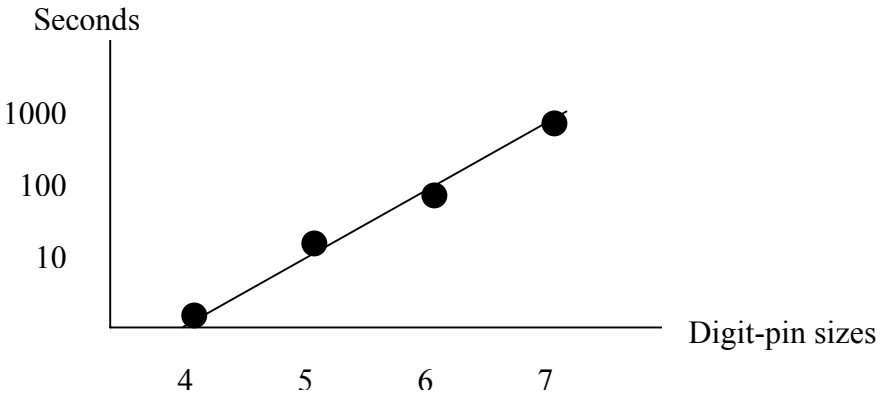

Figure 5.2.3 Performance measurement of the crack program against different sized pins

5.2.2 Active PIN cracking

Some Bluetooth devices, such as hands-free headphones, do not have a user interface. Thus, manufactures have to embed fixed PINs into those devices. The Bluetooth specification specifies that two devices cannot be paired if both of them have fixed PINs. In other words, for a pairing to occur, at least one device has to have a variable PIN. This active PIN attack is specifically targeting the fix-pined Bluetooth devices.

As the name implies, the attack involves communicating actively with the victim. The nature of this attack is very similar with the passive PIN attack. For the passive attack, the calculation is being taken offline. For this active attack, an attacker will initialize a pairing and an authentication with a victim device using a random PIN. Notice that the attacker is always the master. It means that the attacker should always be the first one who send out the challenge and receive the response in an mutual authentication. Once the attacker completed one pairing and collected a

pair of challenge and response for authentication, he will have enough information to launch a brute-force attack. If the attacker can retrieve the PIN before the challenge from the victim expires, she can generate a correct response to complete the authentication. If not, the attacker will have to initialize another round of pairing and authentication. In order to prevent intruders from trying a large number of different pins in a short period of time, the Bluetooth design specifies that a wait interval should be passed before a device response to an authentication attempts coming from the same claimant who has failed the authentication. The wait interval should also be exponentially increased [2]. Therefore, the attacker's MAC address will be stored in the victims' "Black List" in the subsequence rounds of failed pairing and authentication. But the only information that the victim can use to uniquely identify each failed attempt is the MAC address. An attacker can bypass the wait interval as long as he uses a different Mac address for each authentication attempt. To minimize the number of rounds, the attack can utilize a numeric pin dictionary and generates more common PIN candidates such as "1111", "1234", etc. Figure 5.2.4 shows the active PIN- cracking algorithm.

Since this active attack is very similar to the offline version of the PIN cracking and the offline version is much more efficient than the online one, one might wonder why the attacker would use this active attack at all. The reason is that sometimes it may not be easy, if not impossible, for the attacker to eavesdrop the complete pairing process between two target devices.

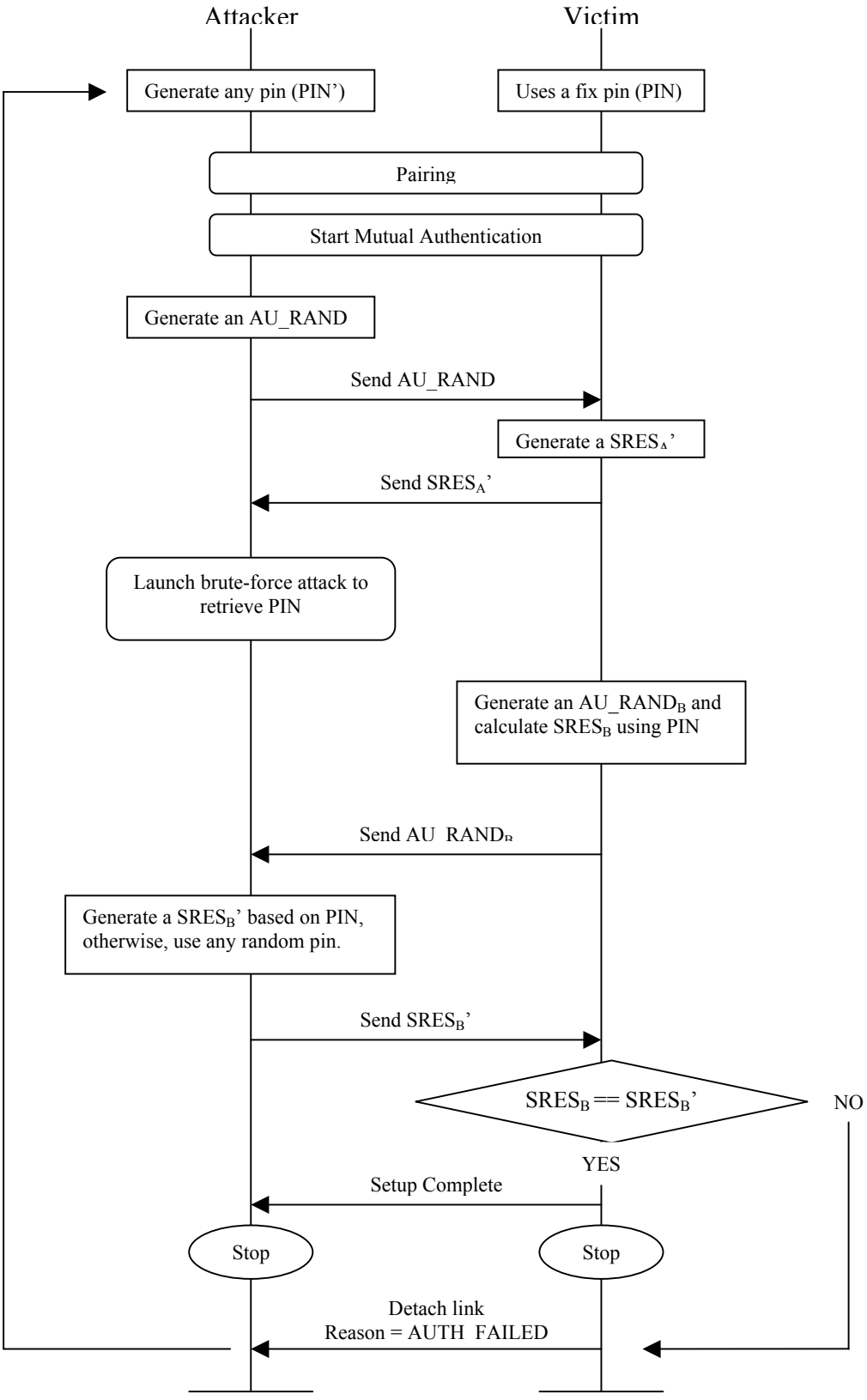

Figure 5.2.4 The active PIN-attack algorithm

5.2.3 Denial-of-Service Attack

The main goal for this attack is to flood a master Bluetooth device, such as an access point, with false authentications in attempt to prevent legitimate users from successfully pairing and authenticating with that master device. An attacker can accomplish this attack by taking advantages of the security measurement that is designed to prevent repeated authentication attempts with different PINs in a relatively short period of time.

To prevent repeated authentication, a device is recommended to store the MAC address that is associated with each failed authentication attempt. A wait time should pass before the device accepts new authentication requests from any of those MAC addresses. If the attacker uses the MAC address of a legitimate user and a fake PIN to authenticate with the master access point, the authentication will most likely fail. The access point will then "memorize" the MAC address of the legitimate user. In consequence, the access point will reject any further pairing and authentication requests coming from the legitimate user until the wait time has passed.

Bluetooth MAC addresses are 48 bits long. There are roughly 2^{48} unique MAC addresses. It would be impractical for the attacker to flood the access point using all possible addresses. But let say that the access point belongs to a mid-size company. The company will mostly provide its employees with Bluetooth devices manufactured by couple specific companies. Since the first three bytes of a Bluetooth MAC address are vendor specific, the attacker will only have to loop through all possible addresses (around 16 million addresses) from those brands. Furthermore, some companies assign fixed 7^{th} hex digit to the address of their products. For example, Sony Ericsson uses 00:0A:D9:E as the first 7 hex digits of the MAC address of their P900 mobile phones [42]. To speed up the denial-of-service attack further, a couple of probing

Bluetooth devices, each one target a different address range, can be used. Figure 5.2.5 describes a normal scenario of a legitimate device connects to an access point. Figure 5.2.6 shows the denial-of-service attack.

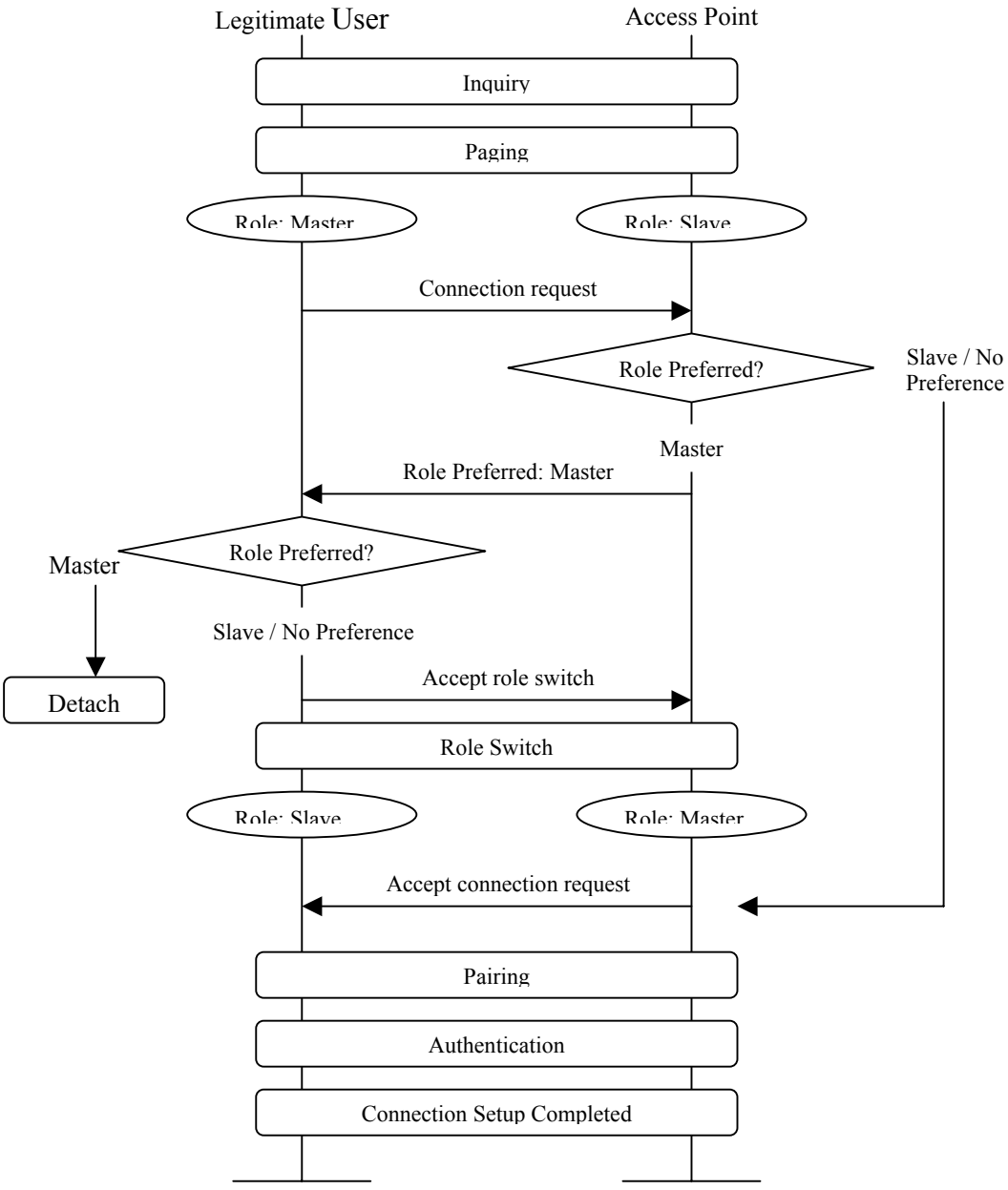

Figure 5.2.5 A legitimate device connects to an access point

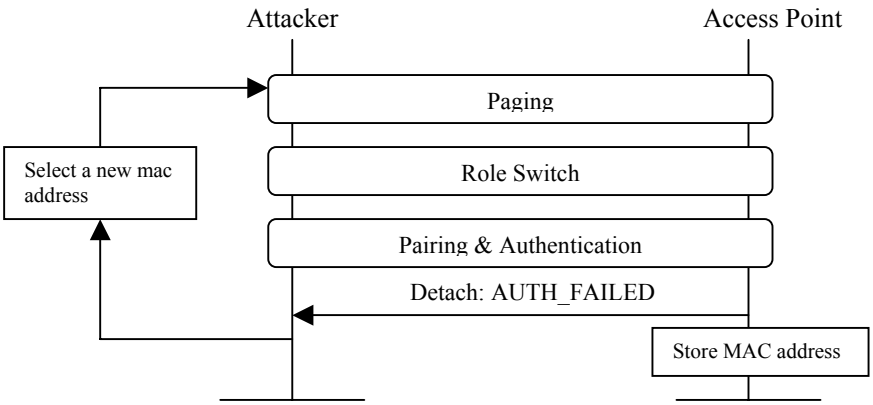

Figure 5.2.6 The Denial-of-Service Attack

5.2.4 Message Replay Attack

In this attack, an attacker replays previously captured messages to a victim without actually decrypting those messages. The attacker does not need to know the encryption key to conduct this attack [27]. This attack divides into two phases as follows:

Phase I:

Two victims (Alice & Bob) are attempting to set up a secure connection. We assume that they have previously paired and they share a secret link key (K). We further assume that Alice is the master of the piconet. Figure 5.2.7 depicts the interactions between Alice and Bob in Phase I. Alice and Bob first mutually authenticate each other using the secret key. Alice then initializes the encryption sequence by sending Bob a random number EN_RAND. They calculate K_c and K_{Cipher} using E3 and E0 respectively. They then encrypt the rest of the messages by XORing the payloads with the cipher streams. The attacker, Trendy, passively listens to the whole conversation between Alice and Bob. The messages and random numbers that Trendy needs (all red messages in Figure 5.2.7) to launch the Phase II attack are AU_RAND_A, EN_RAND, and the rest of the encrypted messages.

Phase II:

In this phase, Trendy initializes a mutual authentication with Bob. Trendy send AU_RAND_A, which he captured during Phase I, to Bob as the challenge such that Bob generates the same ACO_A and $SRES_A$ as in Phase I. Trendy then ignores the response $SRES_A$ coming from Bob. Since Trendy doesn't know K, he has no way to generate a correct response to Bob. But Trendy can relay the challenge to Alice posing as Bob and forwards the response from Alice to Bob. After the mutual authentication between Trendy and Bob has completed, Trendy sends the same encryption random number EN_RAND that he captured in Phase I to Bob. A key observation here is that since ACO_A and EN_RAND in Phase II are the same as those in Phase I, the K_C and K_{Cipher} that Bob generates in Phase II will also be the same as those in Phase I. Trendy can then replay the rest of the messages that he captured in Phase I to Bob. Figure 5.2.8 depicts the entire Phase II attack.

This attack poses some serious threats despite the fact that Trendy cannot decrypt those encrypted messages. For example, Trendy can force Bob to send data in plaintext by sending him an old encrypted STOP_ENCRYPTION command.

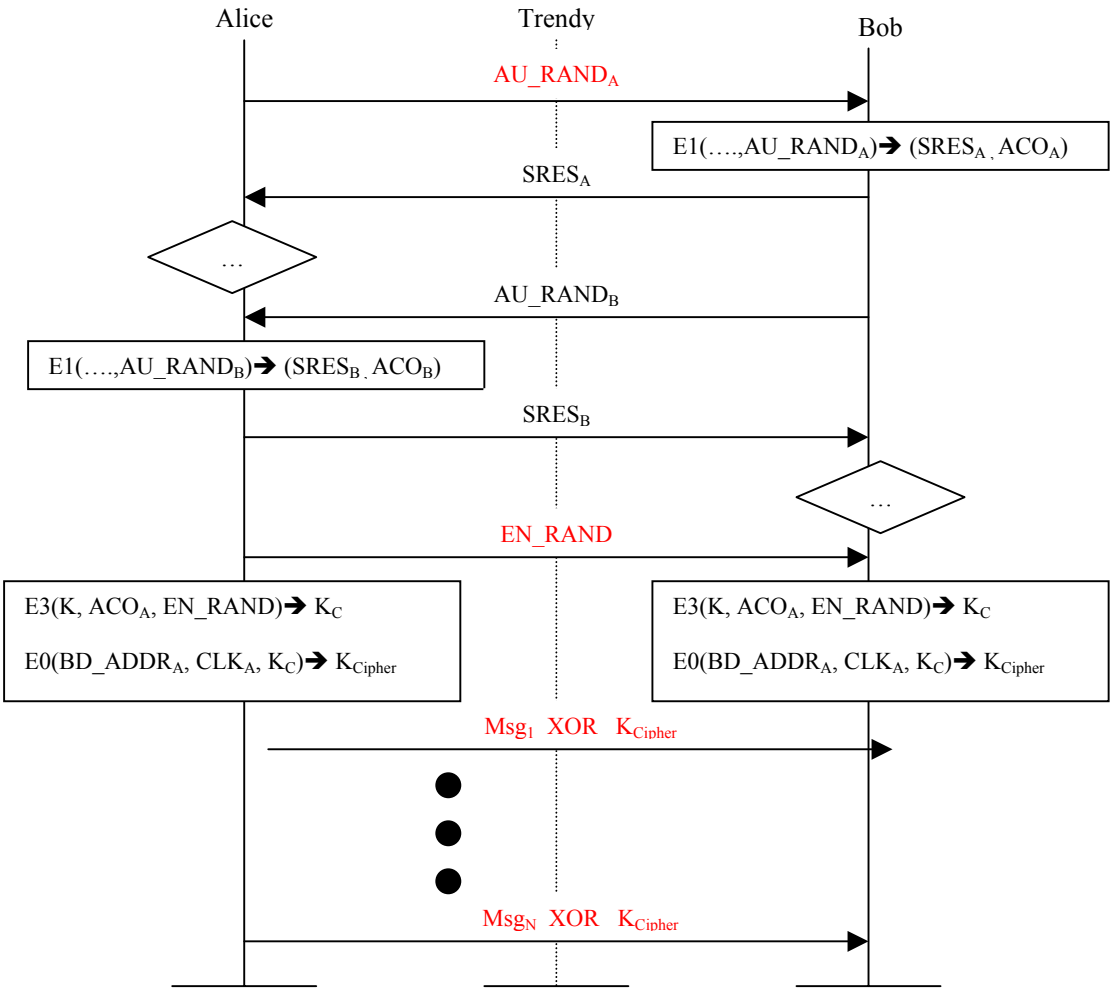

Figure 5.2.7 Phase I of the message Replay Attack

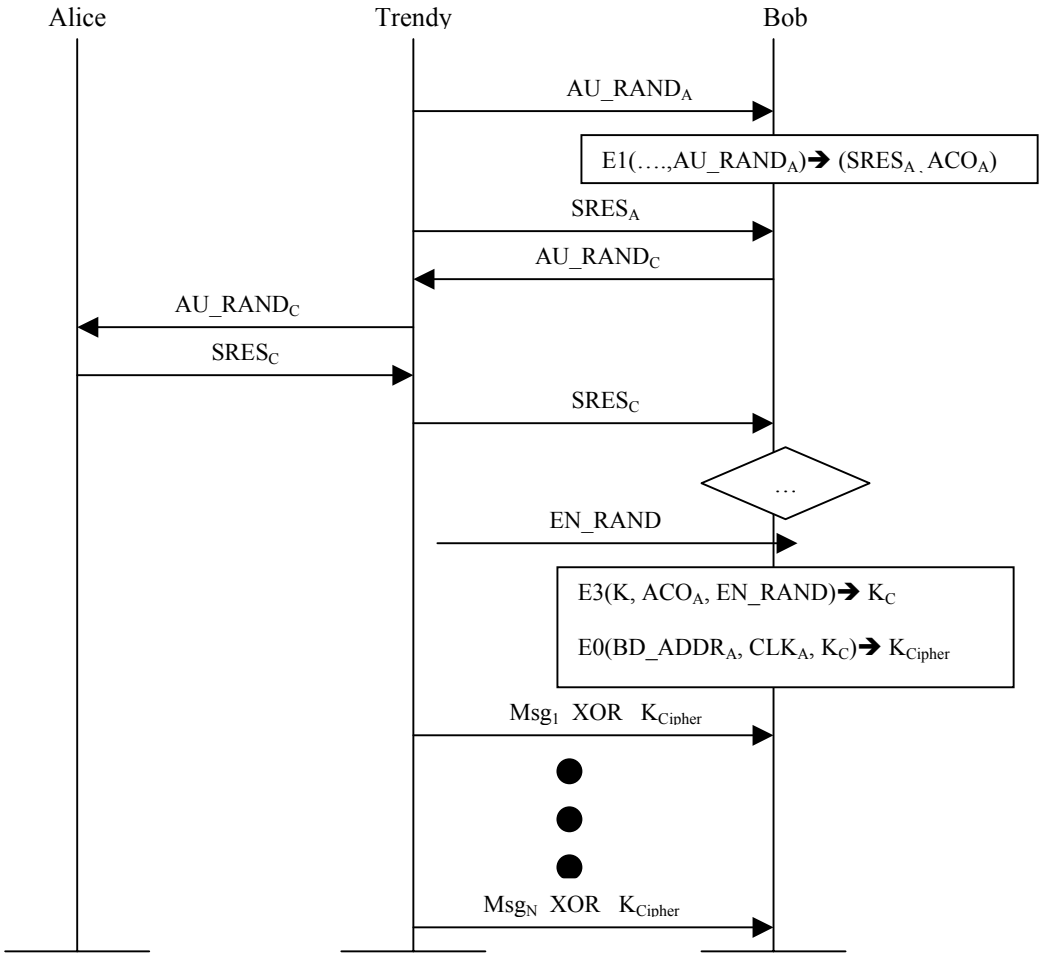

Figure 5.2.8 Phase II of the message Replay Attack

6 ENHANCEMENTS

In this section, several enhancements to the Bluetooth security protocol will be proposed in attempt to defense against the attacks described in the previous section. All proposed enhancements are on the protocol level. In other words, all lower-leveled cryptographic features and functions, such as the SAFER+ block cipher and various hash functions, will not be discussed.

6.1 Online/offline PIN attacks

Two different enhancements to the Bluetooth pairing and authentication protocols are proposed to address both online and offline PIN attacks. The first

enhancement is based on the Encrypted Key Exchange (EKE) protocol suggested by Steven Bellovin and Michael Merritt [28, 29, 30]. The second enhancement is based on MANA III (MANual Authentication III), a multi-channel authentication protocol [23, 24, 25, 26]. The Diffie-Hellman key exchange protocol is the foundation of both enhancements.

6.1.1 Password-based Encrypted Key Exchange (PW-EKE)

A. Overview

The goal of the password-based EKE protocol is to exchange a common key between two parties over an insecure channel, such as a wireless interface, using a shared weak PIN number, such as a 4 digit PIN. That is exactly what the pairing process in the Bluetooth design trying to accomplish. The design of PW-EKE incorporates the usage of both symmetric and asymmetric systems. Figure 6.1.1 shows the modified pairing and authentication using password-based EKE.

PW-EKE is based on the Diffie-Hellman key exchange protocol. A master (M) and a slave (S) try to derive g^{AB} mod p as their common session key by exchanging g^{A} mod p and g^{B} mod p in plaintext. Doing so will not weaken the protocol because (g^{A} mod p) (g^{B} mod p) does not equal (g^{AB} mod p). To recover the key using those two random numbers, the attack will have to solve the discrete log problem. But the Diffie-Hellman key exchange does not provide authentication. Thus, it is prone to the Man-In-The-Middle (MiM) attack. PW-EKE solves this problem by hashing the two random numbers with a common PIN. A simple XOR operation will suffice because the randomness of the two random numbers will provide enough security to protect the weak PIN number. Furthermore, by hashing PIN numbers to the two random

numbers, the strength of those PINs has been "amplified". An important property of this protocol is that weak PINs (such as a 4 digit PIN) will not weaken the protocol.

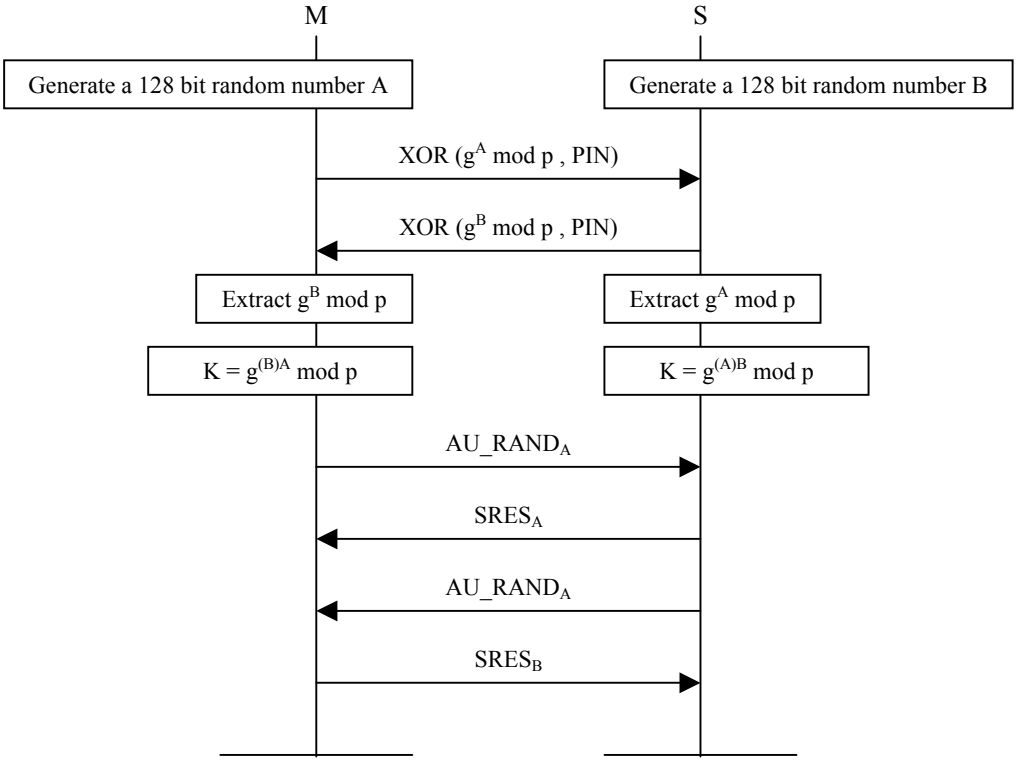

Figure 6.1.1 PW-EKE Based Pairing and Authentication Protocol

B. PW-EKE against different attack scenarios

This subsection demonstrates how the new protocol defense against different attacks.

i) Man-In-the-Middle attack

The Diffie-Hellman key exchange protocol is known to be vulnerable to the MiM attack because it does not provide authentication. Thus, we first have to make sure that the new protocol is well protected against the MiM attack. Figure 6.1.2 shows how MiM works under the original DH protocol. In the new protocol, a PIN is used to provide authentication between the master and the slave. The long random numbers, in return, protect the PIN. Since Trendy doesn't know the PIN, he can't extract the two random numbers from the hashes.

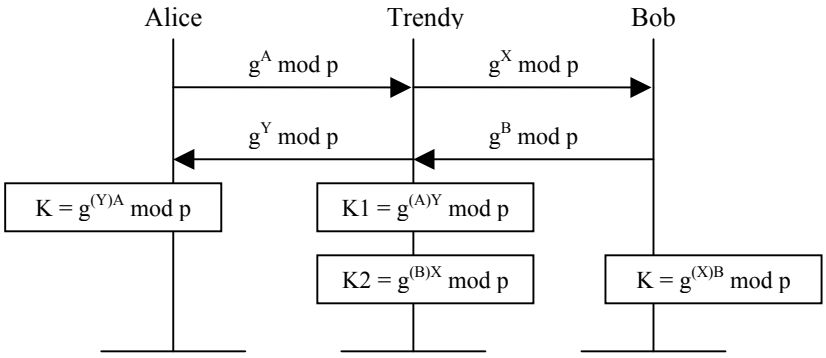

Figure 6.1.2 A MiM attack under the original DH protocol

ii) Brute-Force PIN attack

The feasibility of a brute-force PIN attack depends on how quickly an attacker can verify the correctness of a candidate PIN. Assuming the wireless interface is completely insecure, an attacker will be able to capture every message. Figure 6.1.3 shows how an attacker attempts to launch a brute-force search on the PIN number. Notice that step 3 is not feasible. The attacker has no way to verify the correctness of a candidate PIN. Thus, a brute-force PIN attack cannot be applied on PW-EKE.

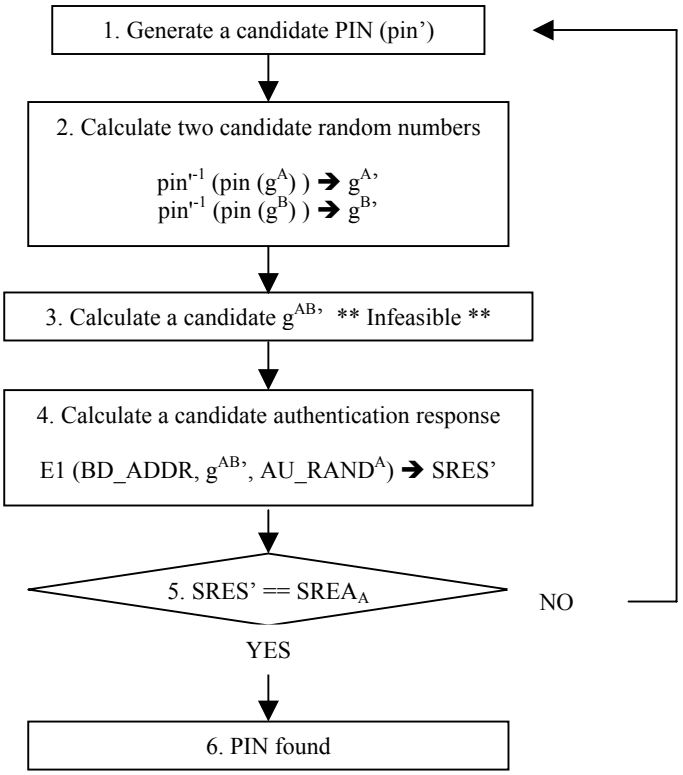

Figure 6.1.3 An attempted brute-force PIN search under PW-EKE

iii) Brute-force attack against A & B

An attacker can potentially retrieve the link key g^{AB} by generating candidate A and B and verify them against the challenge and response pair. But since A and B are long nonces (128 bits), this attack is not feasible.

C. Pros and Cons

A major benefit of PW-EKE is that weak PINs will not weaken the whole protocol. From the brute-force PIN attack in the previous section, we concluded that short PINs make the original Bluetooth security weak. In PW-EKE, the main purpose for the PINs is to provide authentication. The strength of the session keys does not depend on the length of the PINs. Given how often users pick short PINs, this benefit gives PW-EKE an edge over other protocols.

In terms of modification, PW-EKE does not require any changes to the original device interface requirement. The new protocol also needs users to enter PINs to both the master and the slave during a pairing process. For devices that have no keypads, such as headsets, the fixed-pin scheme from the original Bluetooth specification can be applied. In addition, the original challenge-response authentication procedures can also be reused.

6.1.2 MANA III variant (MANual Authentication III)

A. Overview

The original Bluetooth pairing and authentication protocol can be seen as having two communication channels. The first one is an insecure wireless channel that has unlimited bandwidth because there is theoretically no limit on how much data can be exchanged through this channel. The other channel is a physical channel

where PINs are being entered. This channel has a very limited bandwidth because people hate to remember long numbers and they hate to enter long numbers using tiny keypads. The MANA III protocol also has the same channels. But instead of asking users to enter PINs into both devices, it requires users to read a short number from one device and enter it into the other device. Since the short number is randomly generated by one device, it is no longer a personal identification number. In other words, users do not have to remember any PIN under the MANA III protocol. Figure 6.1.4 shows the MANA III protocol.

MANA III is also based on the Diffie-Hellman key exchange protocol. It uses g^{AB} mod p as the session key. Two devices first exchange two exponential random numbers and derive g^{AB} mod p as their intermediate keys. One device then generates a short random number (SR) and displays it on its screen. A user then enters the same number into the other device. The number is XORed with the intermediate key to form a session key. For authentication, instead of using a challenge-response scheme, MANA III uses a hash commitment scheme. Each device generates a 128 bits random number (R) and calculates a long hashed commitment (H) based on R, the session key, and a device ID. They first exchange their commitment to each other. They then release R to each other in order to verify the correctness of the commitments that they have received. The sequence of exchanging commitments and releasing random numbers is important. One device must not release its R unless it has received a commitment from the other device. At the end of the protocol, the authentication results have to be communicated back to the users through the physical channel again. This last step is essential because without it Trendy can use Bob's commitment to brute-force search for the SR. Because SR is a short digit number, recovering SR by brute-force searching on the commitment and response pair will be

easy. Once Trendy recovered the SR, he can complete the mutual authentication with Alice by generating a valid commitment. Figure 6.1.5 describes this attack.

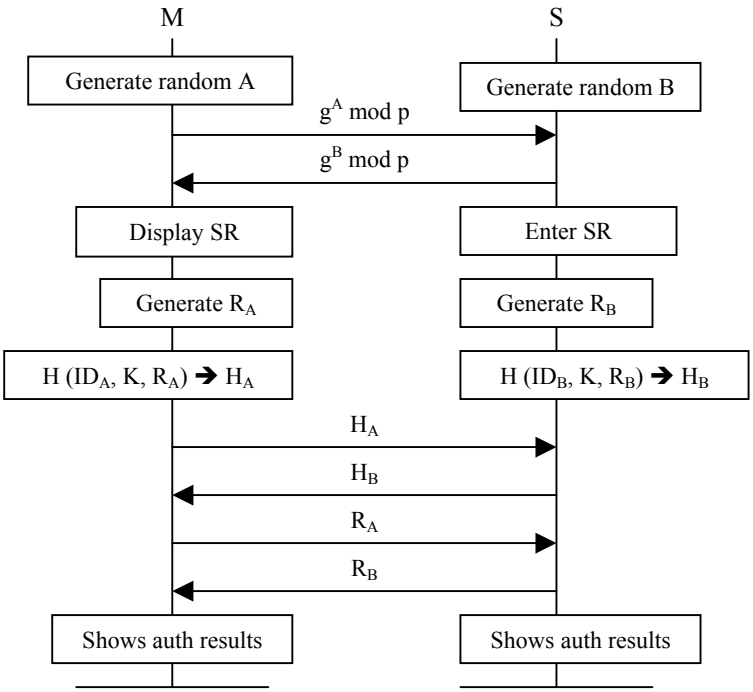

Figure 6.1.4 MANA III

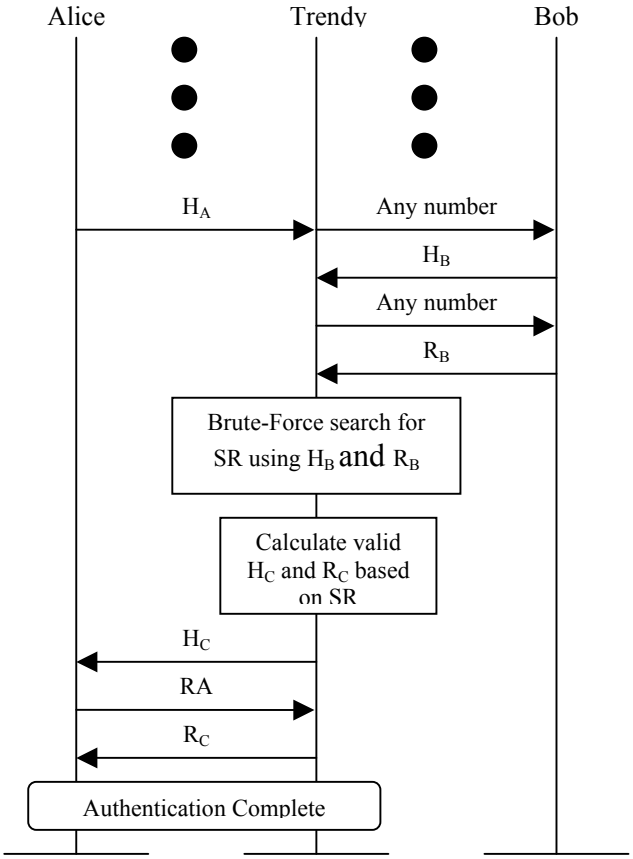

Figure 6.1.5 A MiM attack on MANA III (without displaying authentication results)

The fact that user interactions are needed twice (exchanging SR and displaying Auth Results) is a very undesirable property of the protocol. Thus, a MANA III variant is proposed in this paper. Figure 6.1.6 shows the MANA III variant protocol. Couple modifications are made in the original MANA III in order to eliminate the need to communicate authentication results back to the users.

1. SR will be displayed only after the master has sent its commitment. The protocol pauses until the user entered SR into the slave device.
2. The slave will only disclose its R_B if and only if H_A is valid.

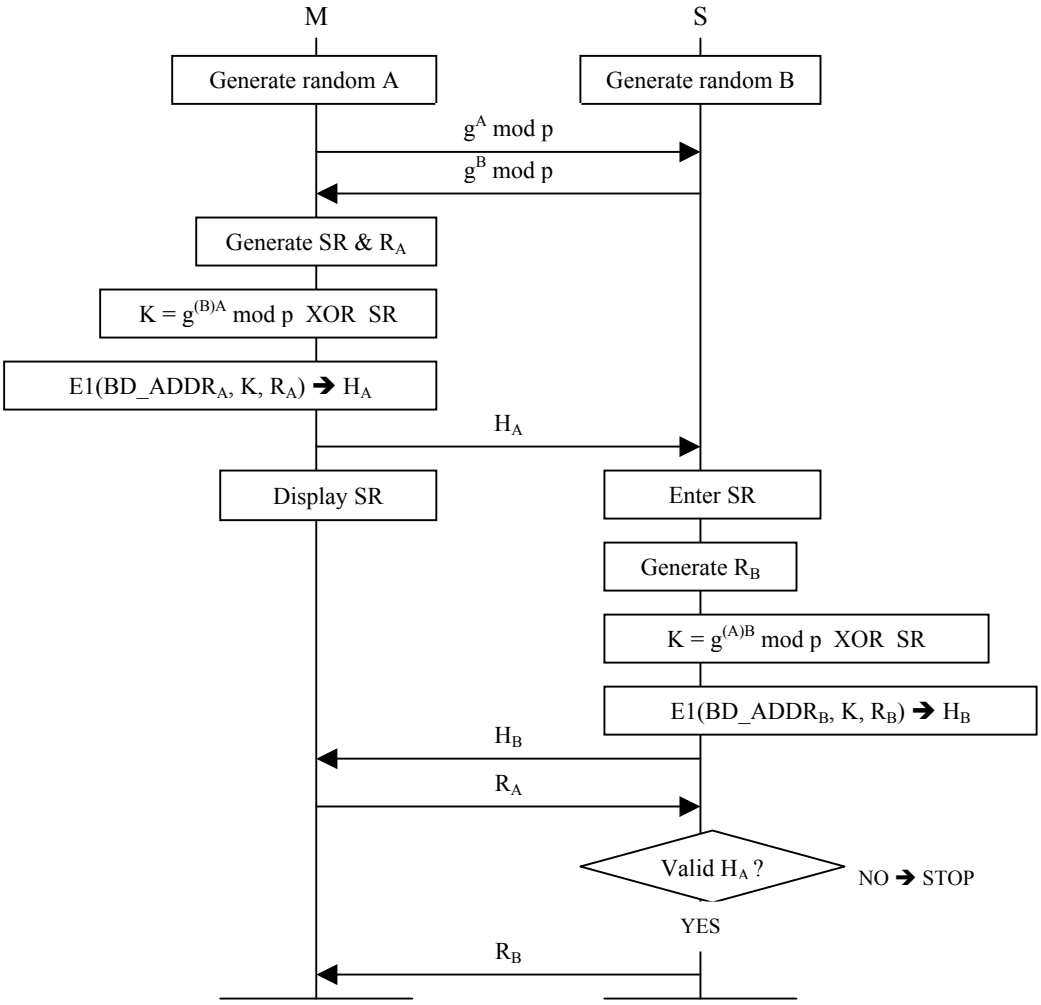

Figure 6.1.6 MANA III Variant

Notice that the challenge-response scheme used by the original Bluetooth design cannot be used in this protocol. Otherwise, a MiM attack will be feasible. Figure 6.1.7 demonstrates this attack. By using a challenge-response pair from Bob, Trendy can launch a brute-force search for SR. Once Trendy recovered SR, he will be able to complete the mutual authentication with Alice by calculating a correct SRES.

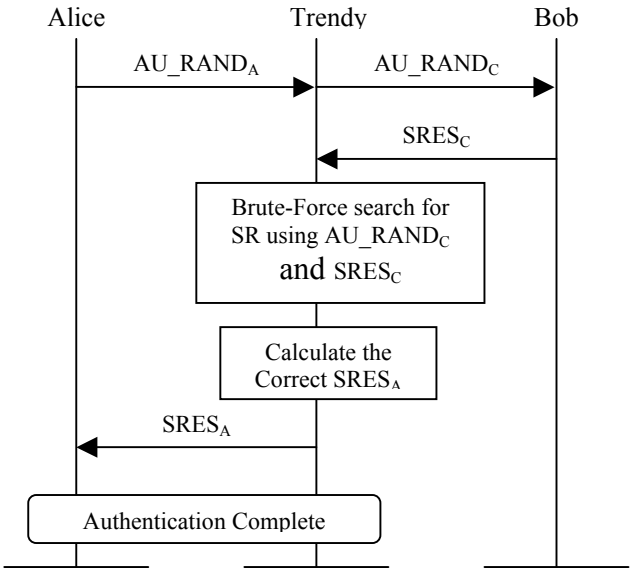

Figure 6.1.7 A MiM attack on MANA III (with challenge-response scheme)

B. MANA III variant against different attack scenarios

This subsection demonstrates how the new protocol defense against different attacks

i) A MiM attack on the MANA III that doesn't displaying authentication results

How does the MANA III variant eliminate the last step from the original MANA III protocol and, at the same time, protect itself from the MiM attack that we have described earlier? Lets take a look at couple different situations where Trendy targets a different victim.

a) Bob as the victim

In this case, Trendy is trying to pair with Bob. Figure 6.1.8 shows the attack. An important observation is that if Trendy doesn't commit H_C to Bob, Bob will never show the prompt for entering SR. Assume that Trendy has recovered SR, he still cannot find a valid R_C for generating H_C because he has already made a commitment to Bob. Since R_C is a long nonce and E1 is a one-way hash function, it's not

feasible for Trendy to brute-force search for a valid R_C that is associated with his commitment to Bob.

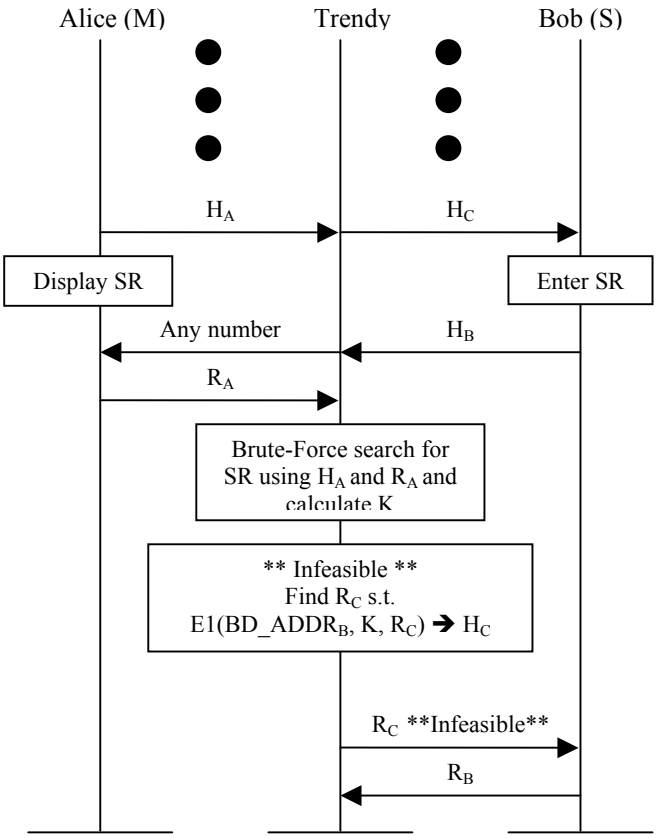

Figure 6.1.8 MiM attack on MANA III variant (I)

b) Alice as the victim

In this scenario, Trendy is trying to pair with Alice. Figure 6.1.9 shows the attack. Since Trendy doesn't know SR at the beginning of the attack, H_C will not be a valid commitment. Bob will end the transaction because of the invalid H_C. In other words, he will never send out R_B that he used to calculate H_B. Trendy will have nothing to validate the correctness of a candidate SR.

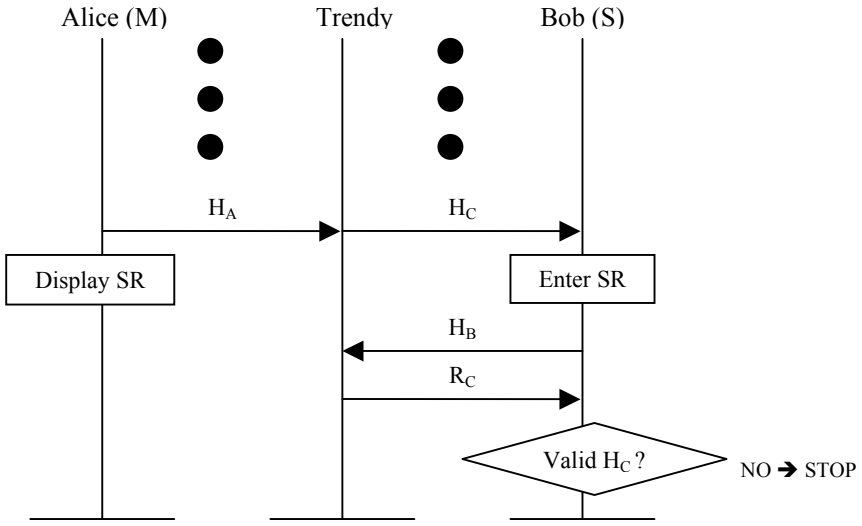

Figure 6.1.9 MiM attack on MANA III variant (II)

ii) Diffie-Hellman MiM attack

The short random number provides authentication to the protocol. Although Trendy can still substitutes his own g^X and g^Y as in Figure 6.1.2, he will not be able to get the session key because he doesn't know SR.

iii) Passive Brute-Force attack on SR

In this attack, Trendy attempted to launch an offline brute-force attack on SR by using all the messages that he has captured during a pairing and authentication session between Alice and Bob. Figure 6.1.10 shows this attempted attack. The attack essentially fails on step five. If the candidate hash H does not equal to H_A, Trendy cannot conclude that the candidate SR is wrong because he doesn't know g^{AB} mod p'. He could have guessed the correct SR and H' would still not equal to H_A due to an incorrect candidate g^{AB} mod p'.

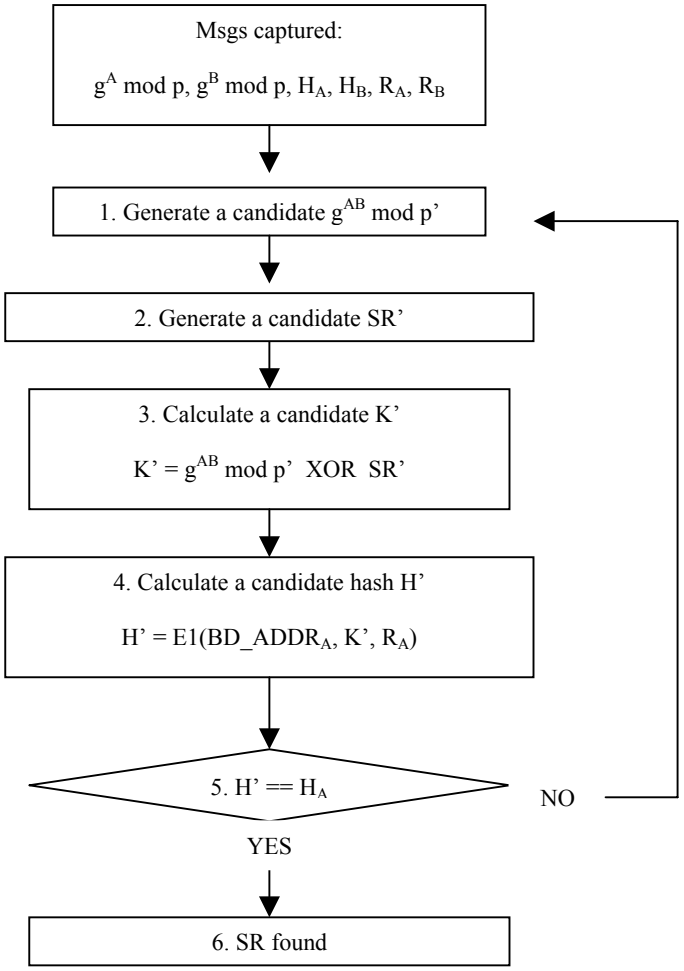

Figure 6.1.10 An attempted brute-force SR search under MANA III Variant

C. Pros and Cons

In this protocol, an SR is no longer a personal identification number because it is being generated randomly every time. This also means that users will no longer be required to memorize their PINs. Furthermore, only one manual entry of numbers is required for each round of pairing and authentication.

A drawback regarding MANA III variant is that it has a different interface requirement comparing to the original Bluetooth design. MANA III variant requires masters to have output interfaces to display short numbers.

6.2 Denial-Of-Service Attack

The main reason why the DOS attack will work is because of the exponentially increased authentication wait time mechanism recommended by the Bluetooth specification. Without this feature, the DOS attack will not work. Thus, by using protocols with manual channels, such as the one in the MANA III variant suggested in the previous section, the exponential wait time mechanism is no longer required. The DOS attack is no longer feasible.

Here is why a manual channel can prevent Trendy from trying different PINs in a short period of time. If an extra step of copying and entering a short number is introduced in the original Bluetooth authentication protocol (assuming that the short number is used to generate an authentication response), Trendy can on longer write a script to automate the authentication process. He has to physically read a short number from the master device and enter the number to his probing slave for each PIN that he tries. Since each trial takes more time to finish, recovering PINs using this technique is not feasible. The exponentially increased authentication wait time mechanism is no longer necessary. The DOS attack will then be prevented.

6.3 Encryption Replay Attack

An important observation regarding this replay attack is that only random numbers from the master are used to generate the 96 bits ACO and K_C for encryption. This attack can be prevented as long as both sides contribute to the creation of the encryption keys and cipher streams.

One straightforward solution is to use the ACOs from both sides to generate KC. But since mutual authentication is optional, there may be chances where only one side has an ACO. In order to guarantee that cipher streams depend on both

masters and slaves, an extra EN_RAND can be exchanged. Figure 6.3.1 shows the protocol.

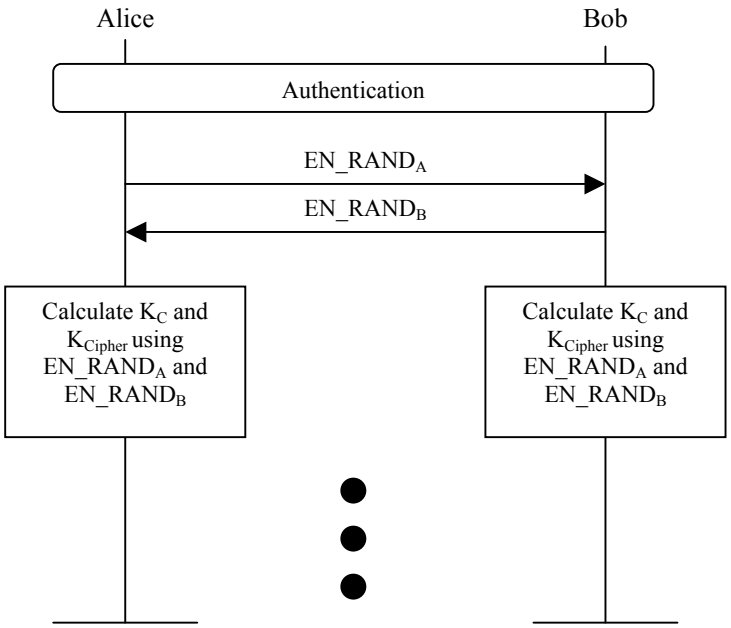

Figure 6.3.1 Encryption protocol with two encryption random numbers

7 Conclusion

This paper has studied four attacks, an online PIN attack, an offline PIN attack, a denial-of-service attack, and an encrypted message replay attack, on the Bluetooth protocol. They revealed the weaknesses on the pairing, challenge-response authentication, and encryption protocols. The paper proposed PW-EKE and MANA III variant as alternatives for the original pairing and authentication protocol. By adding an extra manual channel to the authentication protocol, repeated PIN trying on fix-pined Bluetooth devices is not feasible. The DOS attack can then be prevented because the exponential wait time mechanism is no longer required. The reason why the Bluetooth system is vulnerable to the encrypted message replay attack is because only random numbers from the master are used to derive cipher streams. Adding an extra step to exchange an encryption random number generated by the slave will protect the encryption protocol from generating the same cipher keys and streams.

8 References

[1] Yaniv Shaked and Avishai Wool, Cracking the Bluetooth PIN, at http://www.eng.tau.ac.il/~yash/shaked-wool-mobisys05/

[2] Bluetooth Specification v1.2, at https://www.bluetooth.org/spec/

[3] Qihe Wang, UCBT Bluetooth simulator, at http://www.ececs.uc.edu/~cdmc/ucbt/

[4] IBM, Bluehoc Bluetooth simulator, at http://sourceforge.net/projects/bluehoc/

[5] Godfrey Tan, Blueware Bluetooth simulator for ns, at http://nms.lcs.mit.edu/projects/blueware/software/

[6] LibTomCrypt cryptographic library, at http://libtomcrypt.org/

[7] NS-2 network simulator, at http://www.ececs.uc.edu/~cdmc/ucbt/

[8] Distribution for Mandrake 10.0, at http://www.linuxiso.org/distro.php?distro=29

[9] VMWare, at http://www.vmware.com

[10] Cylink Corporation, AES SAFER, at http://csrc.nist.gov/CryptoToolkit/aes/round1/conf1/saferpls-slides.pdf

[11] Prentice Hall, What is Bluetooth?, at

http://www.developer.com/ws/proto/print.php/10948_1433291_4

[12] A dedicated Bluejacking website, at http://www.bluejackq.com/

[13] http://www.trifinite.org/

[14] BMW, BMW Bluetooth hands-free system, at

http://www.bmwtransact.com/bluetooth/

[15] Ollie Whitehouse, War Nibbling: Bluetooth Insecurity, at

http://www.atstake.com/research/ reports/acrobat/atstake_war_nibbling.pdf

[16] Adam Laurie, Serious flaws in bluetooth security lead to disclosure of personal data, at http://www.thebunker.net/security/bluetooth.htm

[17] Marcel Holtmann, Bluetooth and Linux, at

http://www.holtmann.org/linux/bluetooth/

[18] J. Kelsey, B. Schneier, and D. Wagner, Key Schedule Weakness in SAFER+, at

http://www.schneier.com/paper-safer.html

[19] Bluetooth tutorials, at http://www.palowireless.com/infotooth/tutorial.asp

[20] Wired News, Bluetooth Mobile Phone statistics, at

http://www.wired.com/news/privacy/0,1848,64463,00.html

[21] Ryan Naraine, Source Code for Cabir Cell Phone Worm Released, at

http://www.eweek.com/article2/0,1759,1745949,00.asp

[22] Ollie Whitehouse, CanSecWest/core04, at

http://www.cansecwest.com/csw04/csw04-Whitehouse.pdf

[23] Frank Stajano and Ross Anderson. "The Resurrecting Duckling: Security Issues for Ad-hoc Wireless Networks". Springer-Verlag Berlin Heidelberg, 1999.

[24] Ford-Long Wong and Frank Stajano. "Multi-channel protocols". Springer-Verlag Berlin Heidelberg, 2005.

[25] Christian Gehrmann, Chris J. Mitchell, and Kaisa Nyberg. "Manual authentication for wireless devices". 2004.

[26] McCune, Perrig, and Reiter. "Seeing is believing: Using Camera Phones for Human-Verifiable Authentication". Carnegie Mellon Univerity, 2005.

[27] Eric Gauthier. "A man-in-the-middle attack using Bluetooth in a WLAN interworking environment". Orange, 2004.

[28] Bellovin and Merritt. "Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks". In Proceedings of the IEEE Symposium on Research in Security and Privacy, Oakland, 1992.

[29] Mihir Bellare and Phillip Rogaway. "The AuthA Protocol for Password-Based Authenticated Key Exchange". IEEE Computer Society, 2000.

[30] Daivd P. Jablon. "Strong Password-Only Authenticated Key Exchange". ACM Computer Communication Review, October 1996.

[31] Hager and Midkiff. "An Analysis of Bluetooth Security Vulnerabilities", IEEE Computer Society, 2003.

[32] Markus Jakobsson and Susanne Wetzel. "Security Weakness in Bluetooth". Proceeding of the RSA Conference, LNCS 2020, 2001.

[33] Wong, Stajano, and Clulow. "Repairing the Bluetooth Pairing Protocol". Thirteenth International Workshop in Security Protocols, Apr 2005.

[34] C. Gehrmann and K. Nyberg. "Enhancements to Bluetooth Baseband Security". Proceedings of Nordsec 2001, Nov 2001.

[35] Ford-Long Wong and Frank Stajano. "Location Privacy in Bluetooth". Springer-Verlag Berlin Heidelberg, 2005.

[36] Fathi Taibi and Mazliza Othman. "A Proposed Bluetooth Service-level Security". In Proceedings of the International Conference on Information Technology and Multimedia atUNITEN, Aug 2001.

[37] Keijo M.J. Haataja. "Detailed descriptions of new proof-of-concept Bluetooth seuciryt analysis tools and new secrutiy attacks". Dept of Computer Science, University of Kuopio, 2005.

[38] Levi, Cetinatas, Aydos, et al. "Relay Attacks on Bluetooth Authentication and Solution". Springer-Verlag Berlin Heidelberg, 2004.

[39] Karl E Persson and D. Manivanan. "Secure Connections in Bluetooth Scatternets". Proceedings of the 36th Hawaii International Conference on System Sciences, 2003.

[40] Dave Singelee and Bart Preneel. "Security Overview of Bluetooth". COSIC Internal Report, June 2004.

[41] Tom Karygiannis and Les Owens. "Wireless Network Security – 802.11, Bluetooth, and Handheld Devices". National Institute of Standards and Technology Special Publication 800-48, Nov 2002.

[42] Marek Bialoglowy, Bluetooth Security Review, Part 1, at http://www.securityfocus.com/infocus/1830

[43] Guy Kewney, Bluetooth Scare a Load of Hooey, at

http://www.eweek.com/article2/0,1759,1591184,00.asp