AN ENHANCED PASSKEY ENTRY PROTOCOL FOR SECURE SIMPLE PAIRING IN BLUETOOTH

by

Sai Swaroop Madugula

A thesis submitted to the faculty of graduate studies
Lakehead University
in partial fulfillment of the requirements for the degree of
Masters of Science in Computer Science

Department of Computer Science

Lakehead University

April 2020

To my parents, for supporting me through everything.

Contents

List of Tables

List of Figures

Abstract

Bluetooth devices are being used very extensively in today's world. From simple wireless headsets to maintaining an entire home network, the Bluetooth technology is used everywhere. However, there are still vulnerabilities present in the pairing process of Bluetooth which leads to serious security issues resulting in data theft and manipulation. We scrutinize the passkey entry protocol in Secure Simple Pairing in the Bluetooth standard v5.2. In this thesis, we propose a simple enhancement for the passkey entry protocol in the authentication stage 1 of Secure Simple Pairing (SSP) using preexisting cryptographic hash functions and random integer generation present in the protocol. Our research mainly focuses on strengthening the passkey entry protocol and protecting the devices against passive eavesdropping and active Man-in-the-middle (MITM) attacks in both Bluetooth Basic Rate/Enhanced Data Rate (BR/EDR) and Bluetooth Low Energy (BLE). In addition to increasing the security of the protocol, our proposed model will also significantly reduce the computation cost and the communication cost of the protocol. This model can be implemented for any Bluetooth device which uses the passkey entry protocol and is of version 4.2 or greater.

Chapter 1

Introduction

1.1 The Bluetooth

The Bluetooth is a simple wireless technology developed to exchange data over short ranges of area. The Bluetooth technology usually employs a wireless personal area network (PAN), also known as a piconet, in which data between two devices are exchanged securely. With the introduction of Bluetooth v4.0, namely Bluetooth Low Energy (Bluetooth LE), it became largely famous for its low power consumption. Bluetooth LE largely has its applications in the healthcare, fitness, security, and home network systems. This is because Bluetooth LE offered very low energy consumption while maintaining a similar communication range to that of Bluetooth Basic Rate/Enhanced Data Rate (BR/EDR). As the Bluetooth LE is known for its low energy consumption, it is most suited for the Internet of things (IoT) and is extensively used in it. The Bluetooth LE is ideal to send small amounts of data between devices continuously or periodically. Although different from each other, most security protocols in Bluetooth LE work like the BR/EDR. This is to ensure that both devices which support BR/EDR and Bluetooth LE are compatible to connect with each other. The BR/EDR and Bluetooth LE has a secure way of pairing and generating the link key which is essential for communication between any two Bluetooth devices. This method is known as Secure Simple Pairing (SSP). The two main goals of SSP protocol are to protect the devices against passive eavesdropping and active attacks like MITM.

SSP employs four association models that are used for authentication between two devices. One model is selected out of the four based on the input-output compatibility (IOcap) of the two devices. In these models, the passkey entry protocol uses a 6-digit passkey which is shared on both sides by the user and is used as an authentication token. The Bluetooth standard describes that the passkey entry protocol provides protection against passive eavesdropping and active man-in-the-middle (MITM) attacks. However, there are vulnerabilities present in the passkey entry model which allows an attacker or an adversary to guess the passkey if it is reused and hijack the session which enables the attacker to gain access to the Bluetooth devices and retrieve sensitive information. For instance, many users monitor their heart rate and health using Bluetooth technology. In some hospitals, Bluetooth technology is used to identify the ID of the patients and retrieve their health reports. Now, if an adversary makes use of the vulnerabilities present in SSP and obtains the Long Term Key (LTK) for the SSP session. Then, they can retrieve sensitive data about the patients. And this is not limited to healthcare, Bluetooth technology is also used in maintaining smart home networks which control different aspects of a house. So the main goal of any Bluetooth pairing process should be to prevent the attacker from obtaining the LTK from the legitimate devices.

The most recent countermeasure to the passkey reuse issue was proposed by Da-zhi Sun and Mu in their improved passkey entry protocol. Their improved protocol successfully prevents an adversary from deducing the passkey using passive eavesdropping. However, in our research, we have deduced that their protocol is still vulnerable to our simple brute-force attack which is a variant of the normal brute-force method. By using our attack, the attacker will be able to deduce the correct passkey which will help them to conduct a successful MITM attack allowing them to gain access to the LTK.

1.2 Problem Definition

To understand the vulnerability in the passkey entry protocol, we need to consider a few points.

- Security of passkey entry protocol according to Bluetooth standard.
- Vulnerability in passkey entry protocol.
- How does the vulnerability arise?
- Assumptions for an adversary to exploit the passkey entry protocol vulnerability.
- Goal of an attacker.
- Research Goal.

1.3 Security of passkey entry protocol according to Bluetooth standard

According to the Bluetooth standard v5.2 [1], the passkey entry protocol makes use of a 6-digit passkey which is entered in both the devices. The 6-digit passkey is entered by the user and is of 20-bit length. The passkey entry protocol uses a method of gradual disclosure of each bit of the passkey to thwart MITM attacks.

And it also uses the Elliptic Curve Diffie Hellman (ECDH) under secure connections mode to protect the devices against passive eavesdropping. It describes that a simple brute-force guesser succeeds guessing the passkey with a probability of 0.000001 thus making it very difficult for an attacker to execute a successful MITM attack.

1.4 Vulnerability in Passkey Entry Protocol

The passkey entry protocol normally protects the devices against passive and active attacks. The Bluetooth standard describes that in an ideal case, the user should never reuse the passkey from the previous session. However, if the passkey is reused, it creates a vulnerability which the attacker can exploit. If an attacker can passively capture the public key exchange packets and the passkey entry protocol communication packets, they can easily deduce the passkey and conduct a MITM attack during the user's next SSP session.

1.5 How does the vulnerability arise?

As we described earlier, ideally it is said that the user should not reuse the same passkey again. Just like a user tends to use the same password for every account, the user might find it simple or easy to reuse the same passkey for every Bluetooth connection. The attacker can make use of this vulnerability and exploit the devices. Alternatively, the protocol which is generating the random passkey might reuse the same passkey to save space or its computational power which makes the passkey a static key. Since the random passkey generation protocol does not belong to the Bluetooth standard, it does not necessarily adhere to its rules. Therefore, there is a chance that it will use the same passkey for every connection.

1.6 Assumptions on the capabilities for an adversary to exploit the vulnerability

In our research work, we are assuming that an attacker has a set of capabilities that enables them to perform passive eavesdropping and active MITM attacks. The capabilities are defined below.

- The attacker has the knowledge of the frequency hopping pattern which is shared by the two legitimate Bluetooth devices.
- The attacker has the ability to passively capture the transmission packets exchanged by the two legitimate Bluetooth devices.

- The attacker has the ability to modify packets that are transmitted in real-time between the legitimate Bluetooth devices thus enabling them to perform a MITM attack.

1.7 Goal of an attacker

The obtaining of the 6-digit passkey in the passkey entry protocol run using passive eavesdropping is only one of the goals of an adversary. The main goal of an attacker is to obtain the Long Term Key (LTK) and the encryption key which is derived after the secure simple pairing process is finished successfully. If an attacker can successfully conduct a MITM attack using the derived 6-digit passkey, then they can obtain the LTK and encryption key and will be considered as a trusted device by the legitimate devices. By using these keys, an attacker can simply make a request of sensitive data to the legitimate user. The legitimate user accepts the request since the connection was successful and sends sensitive data to the attacker thinking that it is an authentic device. Alternatively, the attacker can intercept the data being exchanged by legitimate devices and modify the data. Therefore, the vulnerability of the passkey entry protocol can lead to serious data manipulation and data theft.

1.8 Research Goal

Our research goal can be divided into achieving the following results:

- Protection against passive eavesdropping in passkey entry protocol.
- Protection against active MITM attacks in passkey entry protocol.
- Protection against passive and active attacks even when a user reuses the same passkey.

Keeping these goals in mind, we have proposed a new enhanced passkey entry protocol that provides protection against passive eavesdropping and MITM attacks even in case the user reuses the same passkey while significantly decreasing the communication cost and the computation cost making it very efficient. Our protocol also completely prevents an attacker from obtaining the correct passkey by using our brute-force attack. By using our protocol, a legitimate device can successfully prevent an attacker from obtaining the LTK.

In the coming chapters, we will learn about Bluetooth technology in detail. In chapter 2, we will discuss the types of Bluetooth technology and the security overview of Bluetooth. Next, we will talk about the different security features provided by the Bluetooth technology to protect the devices from malicious entities. Next, we will discuss some of the most well-known vulnerabilities present in Bluetooth legacy versions and how the attacks are performed. Finally, we discuss the related works and research works done on the Bluetooth passkey entry model and their countermeasures for the vulnerabilities present in the protocol. In chapter 3, we will go through the secure simple pairing protocol in detail by discussing the architecture of SSP and its various phases. In each phase, we will discuss the working of the SSP and how it provides protection. Finally, we will discuss the vulnerability of the passkey entry protocol in detail and also know how an adversary performs the passive and active attacks to obtain the LTK. In chapter 4, we will discuss the proposed enhanced passkey entry protocol. First, we will talk about the related research work and how they intend to protect the devices. Next, we will be discussing the vulnerability present in their improved protocol and discuss in detail about the attack on their proposed protocol. Next, we will discuss our proposed enhanced passkey entry protocol and how it works. Finally, we will discuss the security of our enhanced passkey entry model under different scenarios. In chapter 5, we will talk about the performance cost of the original and the improved passkey entry protocols proposed by related works. We will compare our enhanced passkey entry protocol with different models and discuss the differences. In chapter 6, we provide a brief conclusion and the scope of future work present for our research work.

Chapter 2

Literature Review

To better understand the working of Secure Simple paring and the passkey entry protocol, we will review the concepts and the related research works done on the secure simple pairing.

2.1 Bluetooth

Bluetooth is a simple wireless technology developed to exchange data over a short range of distance. Bluetooth devices are being used extensively in today's world. It is mostly used for low bandwidth wireless communication. The Bluetooth technology is maintained and organized by the Bluetooth Special Interest Group (SIG). The Bluetooth SIG foundation provides a specification standard that describes the requirements and rules needed to follow to create a Bluetooth device. Currently, the latest version of the Bluetooth standard is version 5.2. According to [2], the main tasks of the SIG organization are to publish the Bluetooth specifications, protect the Bluetooth trademarks and evangelize Bluetooth wireless technology.

The connection ranges of Bluetooth devices are divided into three classes. The class 1 has a range of 100 meters and these types of devices are usually used for industrial purposes. The class 2 Bluetooth devices are the most common devices providing a range of up to 10 meters. These devices usually include smartphones, tablets, laptops, etc. Finally, class 3 devices provide a range of fewer than 10 meters. Bluetooth employs a master/slave architecture for communication. When two Bluetooth devices want to connect, the advertising device (usually known as the peripheral device) broadcasts the advertisement packets within its range. When the other device wants to connect to this advertising device, it initiates a connection with the peripheral device by sending its clock information, unique 48-bit Bluetooth Address (BD_ADDR), and the frequency hopping pattern. Here, the initiating device is known as the master and the advertising device is known as the slave device. When the slave receives the above-mentioned data from the master, it sets its clock and synchronizes itself with the master device. The frequency hopping pattern helps the slave device to communicate with the master at different channels.

The Bluetooth technology uses Frequency Hopping Spread Spectrum (FHSS) to avoid interference with other devices on the communication channels. [2] A Bluetooth device divides the data into several packets and transmits each packet on one of 79 designated Bluetooth channels. Due to the FHSS, Bluetooth devices perform 1600 hops per second which helps the devices to avoid interference and passive eavesdropping. However, since the frequency hopping pattern is not encrypted when it is transmitted from a master device to a slave device, an attacker can capture this packet and learn the frequency hopping pattern which enables them to perform passive eavesdropping on the subsequently transmitted data packets.

Primarily, in Bluetooth, there are two main technologies present: Bluetooth BR/EDR and Bluetooth LE. The Bluetooth LE is more recently developed and was introduced in version 4.0. We will delve a bit deeper into both technologies below.

2.1.1 Bluetooth BR/EDR

The Bluetooth Enhanced data rate, also known as classic Bluetooth, was originally designed for streaming data transfer. It is a packet-based protocol with a master/slave architecture. In a piconet, there can be usually up to 7 slaves and one master. All the operations are done through the master. [3] The classic Bluetooth operates in the 2400-2483.5 MHz range within the ISM 2.4 GHz frequency band. Although the classic Bluetooth has a better data transfer rate at 2-3 Mbps, it suffers from high power consumption. The classic Bluetooth is more ideal for devices that require constant or continuous data streaming.

The Bluetooth BR/EDR architecture is given in Figure 2.1. Note that for simplicity, we have not included all the protocols present in the architecture. We are only focusing on the ones which are pertaining to our research work. The classic Bluetooth architecture can be divided into the following layers or features.

- **Bluetooth Radio Layer:** This layer mostly takes care of the features like defining frequency bands, frequency hopping specifications supported by the device. Aside from this, the radio layer is also responsible for defining the modulation technique that is required and the transmission power classes. It is in this layer that the frequency hopping pattern is generated and sent to the slave device. There is no security provided in this layer as it strictly is limited to only defining the hardware specifications and frequency hopping patterns.

- **Baseband Layer:** This is also known as the physical layer in the classic Bluetooth architecture. This layer is responsible for defining how the data needs to be transmitted across the devices. For instance, the baseband layer defines the addressing schemes, packet frame format, and power control algorithms that are required for connection establishment between the devices present in a piconet. All the devices in the piconet must adhere to schemes and frame formats specified in this layer.

- **LMP layer:** The LMP layer, also known as the Link Manager Protocol, is responsible for the pairing and their subsequent processes. It is in this layer, the connection establishment between two or more devices is performed. The LMP layer defines how the link should be established and also maintained. LMP layer provides the devices with specifications for authentication and encryption. In addition to this, the negotiation of the packet sizes is also done in this layer.

- **L2CAP layer:** The Logical Link Control and Adaptation Protocol (L2CAP) layer resides in the data link layer just above the baseband layer. It is responsible for converting upper layer frames to baseband layer frame format and vice versa. L2CAP can support both connection-oriented and connectionless services.

- **SDP:** The Service Discovery Protocol (SDP) is responsible for searching the services offered by other devices, and their associated parameters. For instance, if a user connects their phone to a wireless headset, then the SDP searches for the services related to the headset and uses Bluetooth profiles like Headset profile or hands-free profile which belong to the wireless headset service and allocates them. Each service is defined by a unique ID.

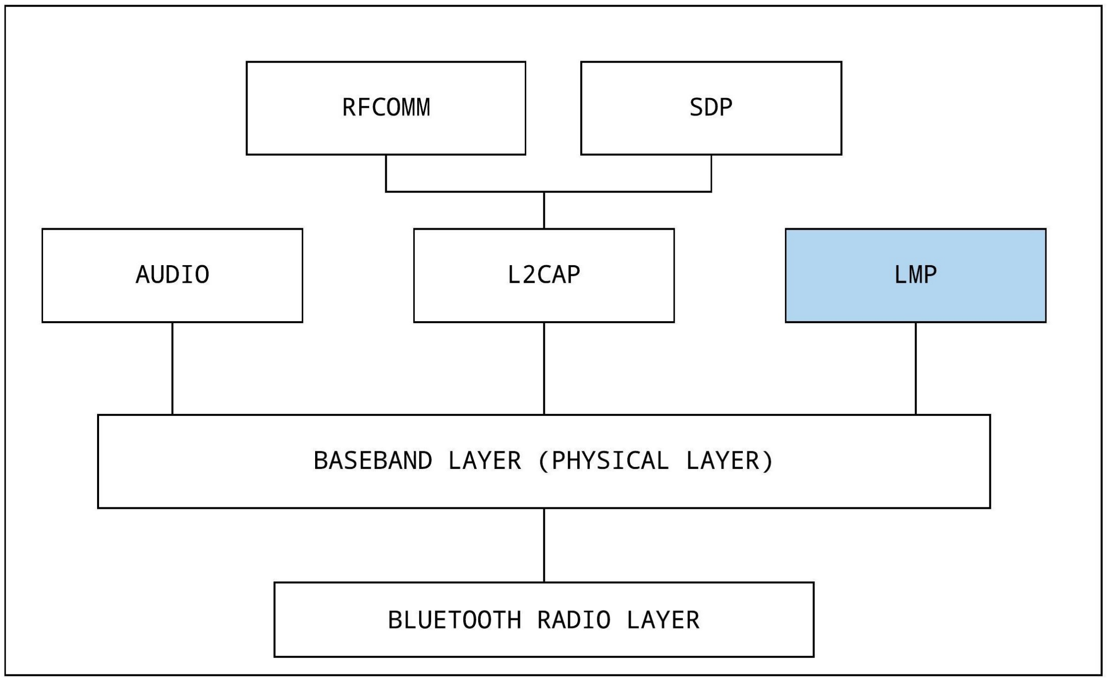

Figure 2.1: Classic Bluetooth Architecture

Although the classic Bluetooth has been widely adopted in wireless communications, with the introduction of Wi-Fi, it got overlapped. The Wi-Fi produced around 11Mbit/s data rate when it was introduced. It also did not have a limitation on the slave devices as classic Bluetooth. And since version 4.0, the Bluetooth focused more on the low energy consumption and compatibility to the smart devices by introducing Bluetooth LE.

2.1.2 Bluetooth Low Energy

The Bluetooth Low Energy was first introduced in the Bluetooth standard v4.0 by Bluetooth SIG. It was mainly aimed at applications in healthcare, fitness, and smart systems such as a smart home network. The main advantage of Bluetooth LE is its low power consumption for operations.

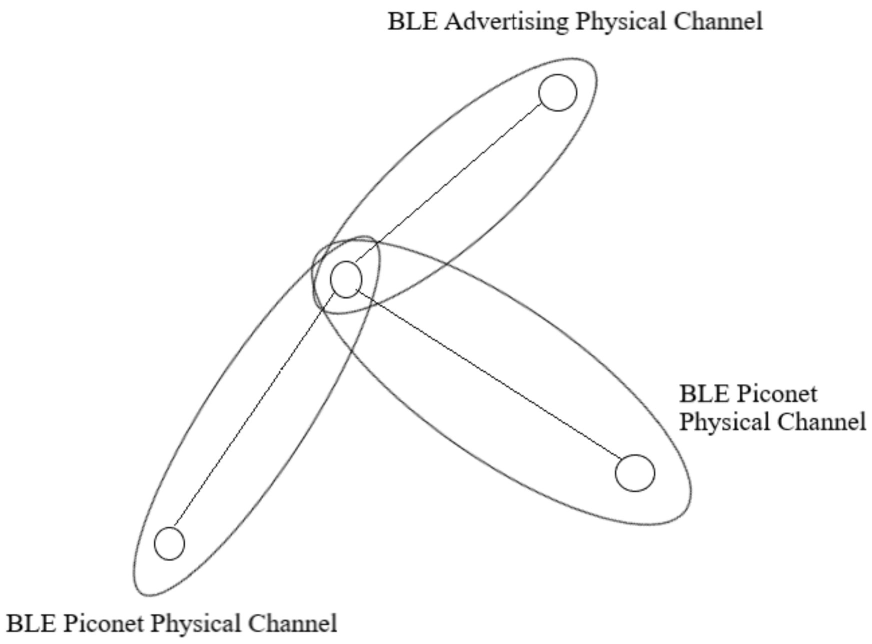
BLE Advertising Physical Channel
BLE Piconet Physical Channel
BLE Piconet Physical Channel

Figure 2.2: Bluetooth Piconet

A Bluetooth LE device connects differently when compared to the classic Bluetooth device. [4] In classic Bluetooth piconet, every slave listens for incoming connections requiring them to be on constant standby whereas Bluetooth LE slave devices invite connections and are in control of when to consume power. In the Figure 2.2, we can see how a piconet is formed where the slave device has a separate physical channel to communicate with the master device. In any Bluetooth LE piconet, a physical channel will consist of two devices.

Unlike classic Bluetooth, the Bluetooth LE is used in cases where there is a need to send data periodically at regular intervals. [5] A study in the beacon software company, Aislelabs, reported that peripherals usually function for 1-2 years with a 1000mAh coin cell battery. This is possible because of the power efficiency of Bluetooth Low Energy protocol which only transmits small packets.

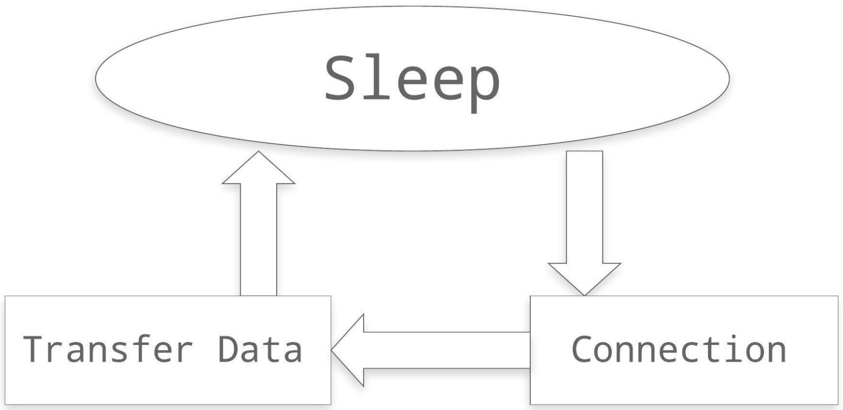

Figure 2.3: Bluetooth Low Energy States

Bluetooth LE technology is vastly used in the Internet of Things (IoT). The sensors of IoT need to transfer small amounts of data to the central system periodically. However, they do not need to keep a channel open constantly as it increases power consumption. This is where Bluetooth LE comes into place. The Bluetooth LE has a special mechanism where the Bluetooth LE only wakes up when a sensor must transfer data between devices. After the data transfer, the Bluetooth LE again goes to sleep thus saving power. [6] The Bluetooth LE over the air data rate is $\boldsymbol{1Mbit/s}$ using Gaussian Frequency Shift Keying (GFSK) modulation. The GFSK enables the Bluetooth LE to allow the reuse of many existing elements of Bluetooth radios.

In Figure 2.4, we can see the architecture of the Bluetooth LE technology. Although somewhat similar to the classic Bluetooth, the Bluetooth LE uses a different set of protocols to provide connection and data transfer between devices. The working of layers such as the physical layer, L2CAP, and Link layer is similar to the classic Bluetooth. The main difference in Bluetooth LE is that the pairing process and the security protocols are provided by the Security Manager Protocol (SMP) which resides in the host. Usually, Bluetooth devices perform operations in host and controller. In simple terms, the host is the operating system that is on the device and the controller can be defined as the Bluetooth integrated circuit (IC). Therefore, if a vulnerability was discovered, it is much easier to patch the vulnerability in the host than in the controller. For instance, the key generation process in classic Bluetooth is performed in the controller, and in case of Bluetooth LE, it is done in the host. And, if a vulnerability arises in the key generation process, it is much easier to apply the necessary countermeasures on Bluetooth LE devices than in classic Bluetooth devices.

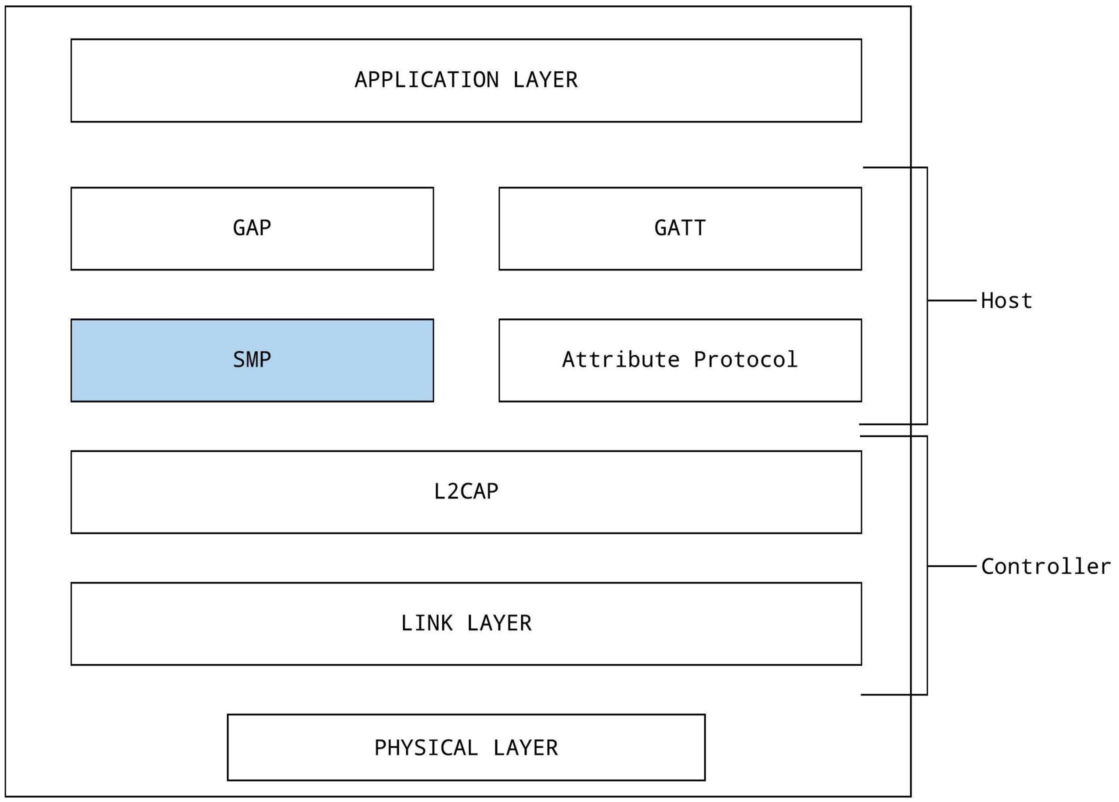

Figure 2.4: Bluetooth Low Energy Architecture

Unlike classic Bluetooth, the Bluetooth LE technology makes use of profiles that are based on the GATT(Generic Attribute Protocol) and ATT(Attribute Protocol). According to Bluetooth Specification standard v5.2 [1], "The GATT server sends responses to requests and when configured, sends indication and notifications asynchronously to the GATT client when specified events occur on the GATT server.". It should also be noted that there can be one or more services present in a GATT profile. The data is stored specifically in these services. In a GATT architecture, the peripheral is called as the GATT server, which consists of the ATT lookup data and service and characteristic definitions. The GATT client (usually a phone) is the device that sends requests to this server.

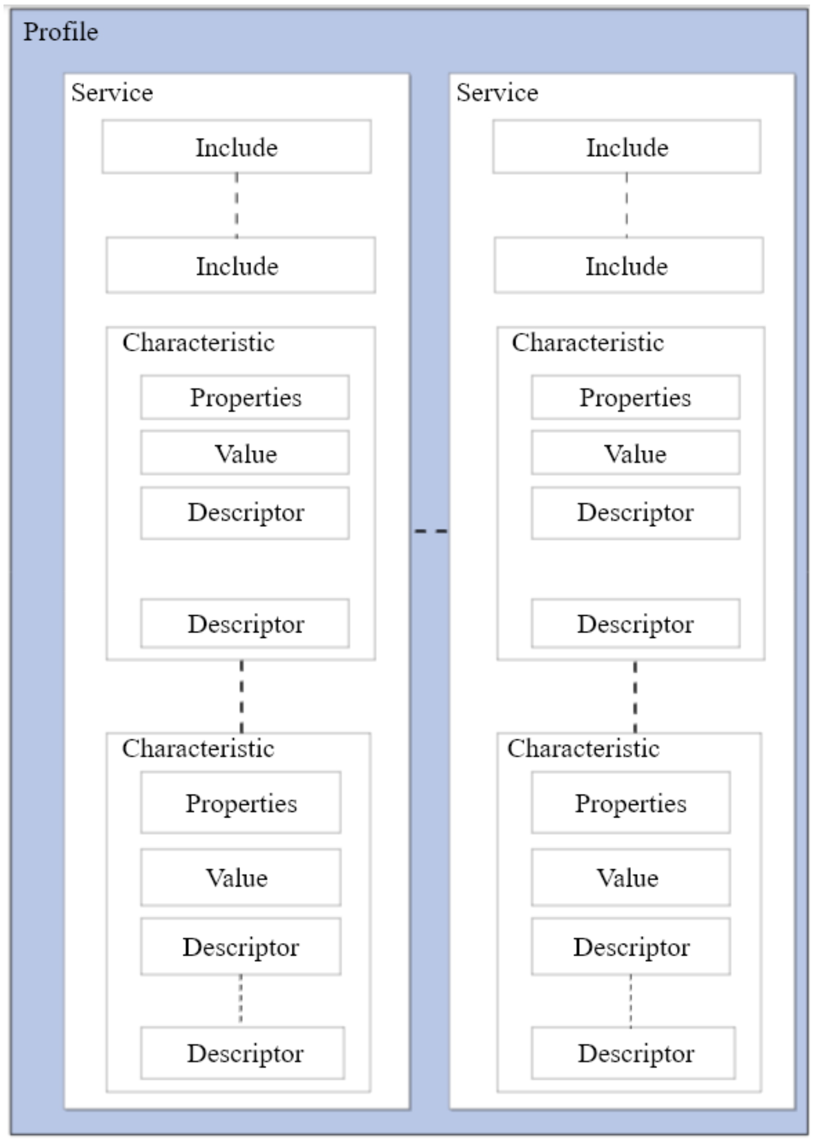

Figure 2.5: Bluetooth GATT Architecture

[7] The GATT uses services and characteristics to define the way that two Bluetooth LE devices transfer data back and forth. The ATT protocol is used in conjunction with GATT to store the services, characteristics, and data related to the devices using 16-bits IDs for each entry. A GATT enables the Bluetooth device to form a centralized structure where the Bluetooth device is the central system and the Bluetooth LE peripherals are connected to this central system individually. Note that a Bluetooth LE peripheral can only be connected to one central device. As a master-slave architecture, if two peripheral devices want to connect, they must communicate through the central device. As shown in figure 2.5, a service can consist of several characteristics values. Each service is denoted by a unique ID

(UUID). [7] The lowest level concept in GATT transactions is the Characteristic, which encapsulates a single data point or an array of related data, such as X/Y/Z values from a 3-axis accelerometer. Characteristics are mainly used to interact with the Bluetooth LE peripherals and are used to transfer data back to the Bluetooth LE peripheral.

2.2 Bluetooth Classic vs Bluetooth Low Energy

Although classic Bluetooth and Bluetooth Low Energy are similar in some respects, they are targeted for different scenarios. For instance, classic Bluetooth is most useful where higher data rate is required provided that the power is available. And Bluetooth LE is most extensively used when there is a low power consumption requirement. Therefore, Bluetooth LE is vastly used in smart devices that need smaller data transfer at regular intervals. [8] The classic Bluetooth typically takes 100ms to send data whereas a Bluetooth LE device takes only 3ms. In the case of power consumption, [9] the Bluetooth LE can be as much as 100 times lower compared to the classic Bluetooth. Coming to the throughput, from the previous information, we can easily deduce that Bluetooth LE will produce low throughput when compared to the classic Bluetooth. Therefore, the classic Bluetooth will have a throughput of up to 2.1 Mbps whereas Bluetooth LE will provide only around 0.3Mbps. [10] Despite their typical use, in connecting peripheral devices at close range, both classic Bluetooth and Bluetooth LE are capable of transmitting data up to 100m.

2.3 Security Overview of Bluetooth

The Bluetooth technology is found in almost every smartphone and IoT device in this world. Hence, it is very important that the system should enforce strict security principles such as authentication, confidentiality, authorization, and integrity. If not, the users might be a victim to data theft and manipulation. According to the Bluetooth standard v5.2 [1], the Bluetooth security model offers five distinct security features: pairing, bonding, device authentication, encryption, and message integrity. Each feature has a specific purpose in connecting the Bluetooth devices.

- **Pairing:** Pairing refers to the process where two devices which are connecting for the first time create shared secret keys called link keys which are used in later stages.
- **Bonding:** After pairing is successful, the keys are stored which are used for later to form a trusted device pair. [11] Bonding establishes a semi-permanent trusted relationship between two Bluetooth devices. When two devices have bonded successfully, communication is much faster for future connection establishments.

- **Device authentication:** In this, both devices compare the keys they computed and verify them. Bluetooth provides authentication at the device level.
- **Encryption:** As the name suggests, it provides encryption using different algorithms based on the IO capabilities of the devices.
- **Message Integrity:** Bluetooth uses Connection Signature Resolving Key (CSRK) to sign the data to prevent message modification.

From a security perspective, pairing is the most essential step in ensuring the security of the two devices. It is in this phase that a shared link key is generated which is the foundation of communication between the paired devices. In the legacy versions of Bluetooth, the encryption was based on the E0 algorithm derived from the Massey-Rueppel algorithm while the authentication and key generation was based on the SAFER+ algorithm. However, several vulnerabilities were found in the legacy versions of the Bluetooth. According to [12], “a number of security breaches have been discovered over the years where the most severe cases consisted of an attacker gaining access to restricted data or taking complete control over the device.”

Considering these security vulnerabilities, the Bluetooth standard introduced Secure Simple Pairing (SSP) for classic Bluetooth in version 2.1 which used the FIPS-approved algorithms. Although SSP was considered secure, it still had a few vulnerabilities in the Bluetooth security architecture which could be easily exploited by the attackers. Legacy SSP used the E0 algorithm for encryption and SAFER+ for Authentication. HMAC-SHA256 was used for Link key generation. With the release of Bluetooth version 4.0, the Bluetooth LE also adopted the Secure simple pairing (now Legacy SSP). However, the main difference between SSP in classic Bluetooth and Bluetooth LE was that Bluetooth LE used did not use ECDH for public key exchange. This led to certain vulnerabilities like MITM attacks in secure message communication between devices. To mitigate these issues, in December 2013, Bluetooth SIG released the secure connections feature to classic Bluetooth. [13] This was an upgrade to the existing Secure Simple Pairing algorithms for pairing, device authentication, and message integrity.

In December 2014, Bluetooth SIG released version 4.2 which introduced features for the Internet of Things (IoT). In addition to this, they released the secure connections only feature to provide greater security for the Bluetooth Low Energy devices. The secure connections mode in Bluetooth used SSP’s four association models to protect the devices against passive and active attacks. Although the secure connections mode is highly secure, it greatly depends on the compatibility of both devices. One of the main improvements in version 4.2 was the adoption of ECDH for pairing in Secure Simple Paring protocol.

	Legacy	**Secure Simple Pairing**	**Secure Connections**
Encryption	E0	E0	AES-CCM
Authentication	SAFER+	SAFER+	HMAC-SHA256
Key Generation	SAFER+	P192 ECDH and HMAC-SHA-256	P256 ECDH and HMAC-SHA-256

Table 2.1: Bluetooth Security Mechanisms

2.4 Security modes in Bluetooth BR/EDR and Bluetooth LE

To securely connect with Bluetooth devices, security modes along with security levels were defined for every Bluetooth device. Each device selects one of the modes and communicates with the other devices. Each security mode defines a specific set of security protocols that need to be followed by both devices. A security mode is selected based on the support and IO compatibility of both devices.

2.4.1 Security modes in Bluetooth BR/EDR

In the classic Bluetooth technology, there are four security modes from which one is selected by both devices to communicate with each other. The security modes of classic Bluetooth are described below.

- **Security mode 1:** This mode is used when both devices have a higher priority of data transfer rather than providing security. This level enables the devices to communicate with no security whatsoever. Therefore, as a result, devices operating in this mode do not employ any mechanisms to prevent other Bluetooth-enabled devices from establishing or interrupting connections [14].

- **Security mode 2:** This mode supports FIPS-approved algorithms such as AES-CMAC encryption during the communication phase of unpaired devices. This level is enforced at the service level of the Bluetooth Architecture. The security mode 2 is selected only if the attributes such as authorization, authentication, and encryption are absolutely supported.

- **Security mode 3:** This mode is executed at the link level. However, for this mode to execute the Bluetooth device must already be paired. Otherwise, the advertising device may reject the Host connection request. This mode also supports encryption.

- **Security mode 4:** The security mode 4 is the most secure out of all the 4 modes. It is enforced at the service level and requires ECDH for the public key exchange. The security mode 4 supports two requirements which are authenticated link key and unauthenticated link key. This mode provides protection against MITM attacks if there is an authenticated link key. The unauthenticated link key uses the SSP legacy model which does not use ECDH. The secure connections mode uses security mode 4.

2.4.2 Security modes in Bluetooth LE

The security modes in Bluetooth LE are similar to those present in classic Bluetooth. In addition to this, the Bluetooth LE employs different levels among three security modes. A Bluetooth LE device can select only one security mode at a time. However, it can maintain a mixed security mode if required. The three security modes in Bluetooth LE are defined as below.

The security mode 1 of the Bluetooth LE enforces the security through encryption and contains four levels of security [15].

- **Security Level 1:** This level like classic Bluetooth does not provide any security. It has no authentication and encryption.

- **Security Level 2:** The security level defines the unauthenticated pairing with encryption. The encryption algorithms used are FIPS-compliant.

- **Security Level 3:** Level 3 provides authenticated pairing with AES-CCM encryption.

- **Security Level 4:** The security level 4 like classic Bluetooth is the most secure level. It provides authenticated LE secure connections pairing with encryption. Due to the secure connections, it uses ECDH for the public key exchange phase and AES-CCM for encryption.

The security mode 2 of Bluetooth LE concerns itself with the signing of data and contains two levels of security. For Bluetooth LE security mode 2, data signing is done with the help of Connection Signature Resolving Key (CSRK).

- **Security Level 1:** Provides unauthenticated pairing with data signing.

- **Security Level 2:** Provides authenticated pairing with data signing.

The mixed security mode provides the ability for a device to support multiple combinations of security modes and levels. For instance, this mode can be used when a device is required to support both security mode 1 and 2. If both Bluetooth devices are trying to connect using the secure connections only mode, then it is essentially security mode 1 with security level 4.

2.5 Well-known Bluetooth Attacks

Since the introduction of the Bluetooth technology, several strides have been made to keep it secure. In the earlier versions of Bluetooth such as version 2.0, 3.0, and 4.0, several vulnerabilities were found by researchers which compromised the security of Bluetooth devices. [16] presented a brief overview of Bluetooth Technology in IoT including its security and vulnerabilities. We will discuss the vulnerabilities present in the legacy versions of Bluetooth and some of the attacks executed on the Bluetooth devices. According to [16], the Bluetooth version 2.0 and less used a 4-digit PIN to conduct the pairing sessions. However, since the PIN is too short, the attackers could easily guess the PIN using simple brute-force schemes. Coming to versions 2.1 and 3.0, security modes had backward compatibility which enabled the attacker to connect their devices with security mode 1 which offered no security. In addition to this, the usage of static keys led to the MITM attacks. Several attacks were introduced when Bluetooth technology was released. We are going to investigate some of the most well-known Bluetooth attacks.

2.5.1 MAC Spoofing Attack

The MAC spoofing attack allows an attacker to get unauthorized access to the Bluetooth device by masquerading the MAC address or specifically the 48-bit unique Bluetooth address (BD_ADDR) of the device. This attack is performed when the link keys are being generated. [17]

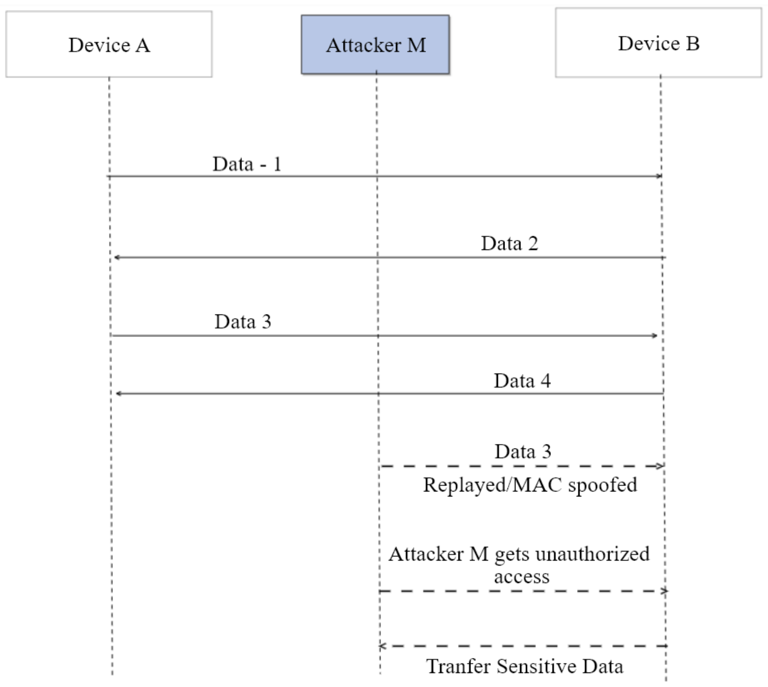

Figure 2.6: MAC Spoofing Attack with Malicious Node M

Here, the attacker simply imitates as the legitimate user A and sends a spoofed packet over to device B. The device B thinking the packet is authentic accepts the connection which gives the attacker access to the device. After that, device B sends sensitive data to the attacker thinking it is the authentic device. The attacker will intercept this data and send the modified data to the device A undetected. For this attack to work, the attacker will use tools to intercept and modify data between the two legitimate devices.

2.5.2 BlueBump Attack

This attack enables the attacker to get complete access to the victim's Bluetooth device and their data. [18] However the attack can be performed in a very limited range, usually around 10 meters for the smartphones. For this attack, the attacker first needs to establish a connection to the victim's device. This is usually done with social engineering tactics where the user believes the attacker's device to be authentic. After a successful connection establishment, the attacker exploits its connection and deletes the link key. This forces the user to create a new link key with unlimited access to the hacker device as per the attacker's request packets. If this step is successful, the attacker will have full access to the user's device until the link key is deleted.

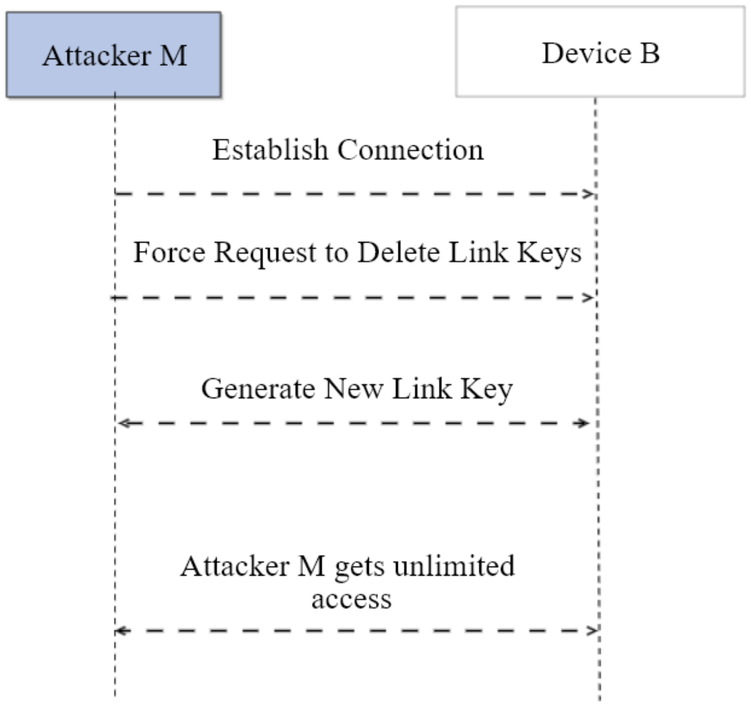
Attacker M
Device B
Establish Connection
Force Request to Delete Link Keys
Generate New Link Key
Attacker M gets unlimited access

Figure 2.7: BlueBump Attack

2.5.3 BluePrinting Attack

The BluePrinting attack is a technique or a method for extracting information and data about a remote Bluetooth device [19]. The attacker can extract information such as device firmware, manufacture information, etc. to exploit their specific vulnerabilities. To execute the BluePrinting Attack, the attacker first finds out the BD_ADDR of the targeted Bluetooth device and then tries to extract detailed information of the device.

2.5.4 BlueSnarfing Attack

BlueSnarfing is the unauthorized access of sensitive information from a Bluetooth device by connecting to the target device silently without the user's knowledge. Legacy Bluetooth devices communicate with each other using a certain protocol known as Object Exchange (OBEX). The OBEX protocol has a vulnerability where there is no authentication required which is exploited by the attacker. According to [20], it is possible for an attacker to connect to the target device without alerting the owner of the request, and gain access to restricted portions of the stored data on the phone. The attacker can extract sensitive information such as contacts from the phonebook, message logs, and more. BlueSnarfing attack, in most of the cases, can be prevented by turning off the discoverability mode of the Bluetooth device. However, certain studies have shown that some tools can be used to guess the device's MAC address or BD_ADDR by brute-forcing techniques [21]. In other cases, [20], stated that the Bluesnarf attack can also be extended by combining it with a backdoor attack. If successful, the attacker can not only gain sensitive information, but they can also get access to certain services like sending an SMS without the user's knowledge.

2.5.5 Bluetooth KNOB Attack

Most recently, in August 2019, a new security vulnerability was introduced by [22], which exploits the low entropy in the encryption key negotiation phase of Bluetooth BR/EDR. This attack enables an attacker to spy and decrypt the encrypted connections after which they can manipulate the information and resend it to legitimate devices. The attack known as Key Negotiation of Bluetooth (KNOB), works when an attacker forces two or more target Bluetooth devices to agree on an encryption key with only one byte of entropy. Because of the decrease in entropy, the attacker can brute-force the encryption key and decrypt the communications. [22] states that "The encryption key length negotiation process in Bluetooth BR/EDR Core v5.1 and earlier is vulnerable to packet injection by an unauthenticated, adjacent attacker that could result in information disclosure and escalation privileges."

2.6 Related Research Works on Passkey Entry Protocol

Da-Zhi Sun and Li Sun [23] designed a formal security model to evaluate SSP's association models and authenticated link key security. They discussed the security and the possible vulnerabilities present in the SSP's association models. This model simulates the networking of the Bluetooth devices and detects attack vectors when an SSP session is run over an insecure public channel. Eli Biham and Lior Neumann [24] discovered a new vulnerability in the ECDH key exchange process. Their attack successfully allows an adversary to recover the session encryption which is used for secure data transfer with a success rate of 50% during the pairing attempts. They accomplished this by modifying the y-coordinate of the public keys of the device and setting them to zero. By doing this, the computed DHkey value will always be infinity. Therefore, an adversary could calculate the encryption key this way. This attack is performed during the public key exchange phase of SSP. Samta

Gajbhiya [25] proposed SSP with an enhanced security level which involves SSP with authenticated public key exchange and delayed-encrypted capability exchange. The main goal of their protocol is to prevent an attacker from forcing the legitimate devices to adopt the association model as per the attacker's convenience. Giwon Kwon [26] introduced a method to increase the length of the Temporary Key (TK) by repeating it.

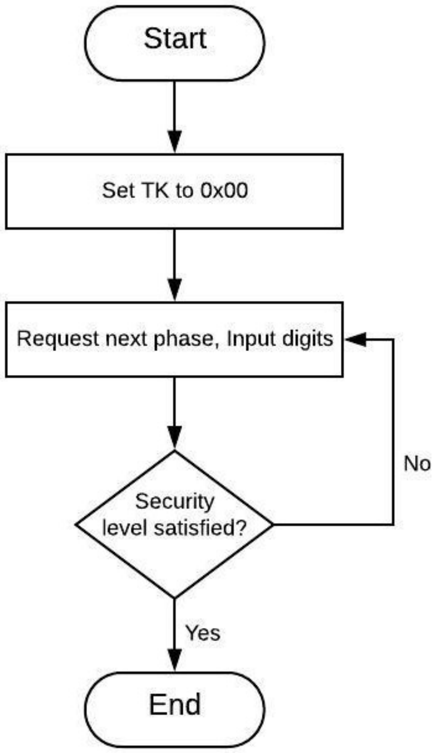

Figure 2.8: Giwon Kwon Proposed Method

They defined a model in which the master is required to select and transmit a security level through the security level field in the PairingRequest packet. When a user enters the passkey, the ***PasskeyRequest*** packet sets its phase field to 00 which represents the first phase. Then the packet is sent to the response device where digits are on the display with the requested phase. Now, the requesting device enters the digits displayed on the response device after which the digits are concatenated to the TK on both sides. This process is repeated until a satisfactory security level is reached. They proved that in a conventional Bluetooth LE protocol, it takes less than 20 seconds for a 3.4 GHz CPU to predict a 6-digit TK which has about 1,000,000 combinations. However, as the length increases, it becomes exponentially difficult to brute-force the TK. However, for this protocol, the user has to enter different passkeys in all devices until a security level is satisfied on all devices which is redundant and cumbersome for the user.

Da-Zhi Sun [27] proposed a novel method of protecting the passkey entry model in secure simple pairing protocol against MITM even if passkey is reused. After the passkey is injected into two devices, they generated a random nonce on both devices A and B and exchange it. Then, A computes a hash using the HMAC-SHA256 algorithm based on the passkey entered and sets the value $\mathbf{r}_a^*$ by taking the six most significant digits of r. The same process is also done at device B and the rest of the protocol is kept the same. Due to the random nonces, the $\mathbf{r}^*$ values computed from the hash are different for each connection.

Barnickel [28] researched on the vulnerability on the passkey entry model of Bluetooth v2.1+ when the user reuses the same passkey for another SSP session. They implemented the MITM attack with a successful probability of 90% using the GNU radio platform on Universal Software Radio Peripheral (USRP) and USRP2 devices. They proposed two countermeasures for the MITM attacks. The first is to record all the previously entered passkeys by the user and reject them if they are used again. However, this would mean there would be a very high storage cost of the protocol since it has to store every passkey entered for every SSP session. In addition to this, it is difficult for a user to enter a new password every time they want to connect to a new device since even they have to remember the previous passkeys used. The second countermeasure involved using encrypting and decrypting the random nonces which would prevent a passive attacker from eavesdropping. However, it greatly increases the complexity and the performance cost of the protocol due to the encryption and decryption functions.

Chapter 3

Secure Simple Pairing and Passkey Entry Protocol

For any kind of data transfer between two or more Bluetooth devices, they first need to establish a connection and pair. To make it secure, the devices use protocols such as Secure Simple Pairing under different security modes. The end goal of any pairing process between two Bluetooth devices is to generate a shared long-term link key on both sides which is used to transfer data. In classic Bluetooth and Bluetooth LE, there are a total of five phases in the SSP protocol. SSP strives to provide security while maintaining minimal user interaction. The primary focus of our research work is the passkey entry protocol present in the authentication stage 1 phase.

3.1 Architecture of SSP

The SSP is the most important pairing protocol in Bluetooth Pairing. Before SSP, the Bluetooth technology used a 4-digit PIN code to pair with other devices. This PIN code was also a part of the Link Key calculation process. Due to the short length of the PIN code, several vulnerabilities were revealed, and to mitigate them, SSP was used. [29] The SSP protocol makes use of extremely large random numbers (usually 128-bit) for seeding Link key calculations. Thus, an attacker cannot brute-force it. The end goal of the SSP is to generate a shared Link key between two Bluetooth devices. A shared link key must be linked with a strong encryption algorithm to give the user protection against passive eavesdropping. Therefore, SSP uses Elliptic Curve Diffie Hellman (ECDH) to generate the Link Key (LK) and is used to maintain the pairing. However, The SSP process can still be subjected to man-in-the-middle (MITM) attacks. To prevent MITM attacks, the Bluetooth standard has introduced association models that are implemented in the authentication stage 1 (phase 2) of the SSP.

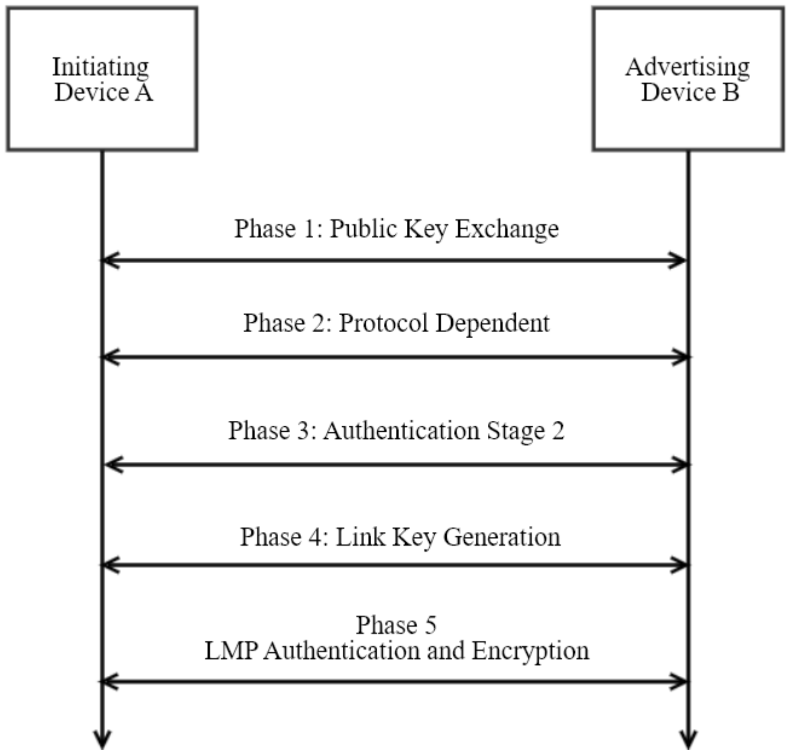
Initiating
Device A
Advertising
Device B
Phase 1: Public Key Exchange
Phase 2: Protocol Dependent
Phase 3: Authentication Stage 2
Phase 4: Link Key Generation
Phase 5
LMP Authentication and Encryption

Figure 3.1: SSP Phases

As shown in Figure 3.1, the working of SSP is the same for any device, excluding phase 2 which is protocol dependent. All phases of SSP must work in the same way in every Bluetooth device. As for phase 2, when the IO capabilities are exchanged before phase 1 of SSP, both devices show what protocols are supported by them. Based on the compatibility and the requirements, a protocol is selected and executed.

3.2 How SSP provides security

Usually, an attacker has two ways of obtaining data. The first technique is to execute passive eavesdropping (usually performed with specially designed tools like Ubertooth) and capture the transmission packets. By doing this, the attacker can try to compute the LTK based on the captured data in phase 1 and phase 2. If the attacker obtains the LTK this way, then they can simply calculate the encryption key which is originally derived from the LTK. Thus, when two legitimate devices transfer data using this encryption key, the attacker can again capture the data packets and simply decrypt them using the deduced encryption. This way, they can gain access to sensitive data.

The second technique is to execute a MITM attack. This technique is more dangerous because if successful, the attacker has a trusted connection with the legitimate device and can request data or send malicious files to the target device. In the case of passive eavesdropping, the attacker can only access the data which is transferred between two legitimate devices. However, when it comes to active attacks such as MITM, the attacker and the target devices share the same LTK. Therefore, an attacker can enable data transfer requests whenever they require provided that the target device has its Bluetooth switched on.

In terms of security, the SSP protocol has two goals. The first goal is to protect the devices from passive eavesdropping. The second goal is to protect the devices from MITM attacks. SSP provides protection from these attacks in the authentication stage 1 and authentication stage 2. In authentication stage 1, the SSP tries to prevent any MITM attacks by using either the numeric comparison or the passkey entry protocol based on the IOcap of devices. That way, if an attacker was able to intercept the communication in phase 1 of SSP, it prevents the attacker from obtaining the LTK by performing authentication in phase 2. In authentication stage 2, the DHkey which is computed in phase 1 is verified which protects the device from passive eavesdropping. This way, an attacker cannot obtain the LTK which is needed to transfer data between devices.

3.3 SSP Phase 1: Public Key Exchange

When two Bluetooth devices want to transfer data between each other, they first need to establish a secure connection between them. For this purpose, SSP is used which provides authentication, confidentiality, and integrity. Whenever a Bluetooth device wants to connect with another device, it broadcasts advertisement packets within its range. The device which is broadcasting the advertisement packets is known as the advertising device. When a device is in advertising mode, it is discoverable by all other Bluetooth devices within its vicinity. The device which wants to connect to the advertising device is known as the initiating device and is responsible for sending an initiate packet to the advertising device. After the initiating device establishes a connection with the responding device, the input-output capabilities (IOcap) are exchanged between the two devices to select a security mode. It is in this step that both devices select the required SSP association model as the pairing protocol and proceed with the protocol. The IOcap is generated by combining the input and output capabilities. Every Bluetooth device has a specific set of capabilities. The input capabilities of any Bluetooth device are given in Table 3.1

Capability	**Description**
No Input	Device does not have any input capabilities to indicate any message (like ’Yes’ or ’No’).
Yes / No	Device has at least one button which the user has to press to indicate an ’Yes’, otherwise the device assumes the value as ’No’. Alternatively, device might contain two buttons indicating ’Yes’ and ’No’.
Keyboard	Device has a keyboard through which the user can enter a passkey and also has two button for confirmation and rejection.

Table 3.1: Input Capabilities of a Bluetooth Device

Similarly, the output capabilities of a Bluetooth device are given in Table 3.2

Capability	**Description**
No Output	Device does not have any output capabilities such as a display to indicate any message or a passkey.
Yes / No	Device has the capability to communicate or display the 6-digit passkey.

Table 3.2: Output Capabilities of a Bluetooth Device

In each Bluetooth device, the input and output capability is combined to generate a single IOcap specific to that device. This IOcap is then used in the pairing process. The IOcap mapping of Bluetooth devices are given in Table 3.3

	Local Output capacity	
	No Output	**Numeric Output**
No input	NoInputNoOutput	DisplayOnly
Yes/No	NoInputNoOutput	DisplayYesNo
Keyboard	KeyboardOnly	KeyboardDisplay

Table 3.3: IOcap of a Bluetooth Device

In the first phase of SSP, the public keys of both devices are exchanged between each other. On both sides, a public and private key pair are generated by using the ECDH algorithm.

3.3.1 ECDH

The Secure Simple pairing protocol was first introduced in the Bluetooth version 2.1. Later, Bluetooth LE also adopted the SSP for connections in version 4.0. However, the version 4.0 SSP (now legacy SSP) in Bluetooth LE did not use ECDH for public key exchange. This led to a few vulnerabilities where an attacker could passively capture packets and try to deduce the shared link key. Therefore, in version 4.2, secure connections mode was introduced where ECDH was for the public key exchange phase to prevent passive eavesdropping. The ECDH algorithm is a public key cryptographic system that allows two devices, to establish a shared secret over an insecure channel without explicitly communicating the secret keys of both sides. The shared secret key is known as the Diffie-Hellman Key (DHKey) and it is calculated at both sides. [30] The ECDH is a variant of the Diffie-Hellman protocol using elliptic-curve cryptography. Each device will have its public-private key pair used to establish a connection. And to generate them, ECDH uses six parameters or variables as shown in table 3.4.

Parameters	**Description**
p	Prime number that specifies the size of the finite field.
a,b	Coefficients of the elliptic curve equation.
G	Base point that generates the subgroup.
n	Order of the subgroup.
h	Co-factor of the subgroup.

Table 3.4: ECDH domain parameters

In ECDH, the private key of the device is generated by choosing a random integer $\boldsymbol{d}$ from [*1,...,n-1*] where n is the order of the subgroup [31]. The public key is derived as a point $\boldsymbol{Q} = \boldsymbol{d} \cdot \boldsymbol{G}$ where G is the base point of the subgroup. The public key is sent when the connection is needed to be established.

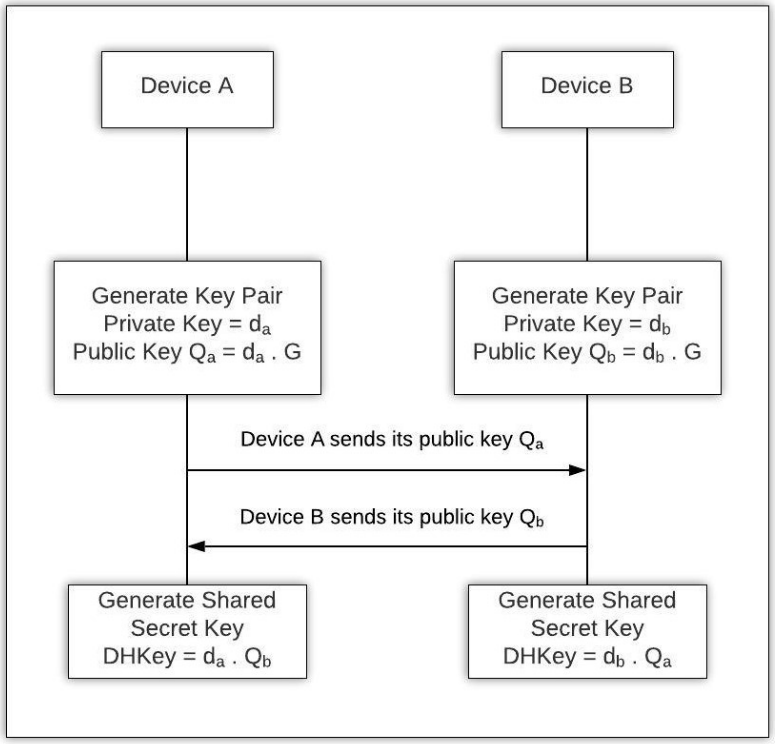
Device A
Device B
Generate Key Pair
Private Key = d_a
Public Key $Q_a = d_a . G$
Generate Key Pair
Private Key = d_b
Public Key $Q_b = d_b . G$
Device A sends its public key Q_a
Device B sends its public key Q_b
Generate Shared Secret Key
DHKey = $d_a . Q_b$
Generate Shared Secret Key
DHKey = $d_b . Q_a$

Figure 3.2: Communication using ECDH

As shown in Figure 3.2, if device A wants to connect to device B, the first step for both devices is to generate their public and private key pairs. After the key pairs are generated they exchange their public keys with each other on an insecure channel. If the device A has a key pair of (d_a, Q_a) and the device B has a key pair of (d_b, Q_b), then both devices exchange their public key values (Q_a, Q_b) with each other. Here, d_a and d_b are the secret keys of the device A and B respectively and are not disclosed to either side. After the public key exchange, the device A computes a point from its elliptic curve DHkey = $(d_a \cdot Q_b)$. Likewise, device B computes a point from its own elliptic curve DHkey = $(d_b \cdot Q_a)$. Thus, a shared secret key (DHKey) is calculated on both sides. [30] The shared secret calculated by both parties is equal, because

$$d_a \cdot Q_b = d_a \cdot d_b \cdot G = d_b \cdot d_a \cdot G = d_b \cdot Q_a$$

In this run, the only information an attacker can passively capture are public keys of both devices. However, the attacker has no way of computing the secret keys from the public keys unless they solve the discrete logarithm problem. Although the ECDH algorithm is secure, it provides no authentication during its public key exchange phase. As a result, an attacker can execute a MITM attack on two devices that are trying to communicate with each other. If an attacker executes the attack in such a way, then the legitimate device and the attacker will have the same DHKey. Therefore, additional security measures like AES ciphers and HMAC are necessary when working with ECDH. It is also advised that the public-private key pair of ECDH must be changed regularly to avoid a lack of entropy.

Coming back to the execution of phase 1 in SSP, the initiating device A sends its public key to the advertising B. After receiving the public key of A, device B also sends its own public key to A. When the public key exchange is successful, both devices start computing the DHKey based on their secret key and the public key of the other device. For example, at device A, the DHkey is calculated as P256(SK$_a$, PK$_b$) or P192(SK$_a$, PK$_b$) depending on the length of the elliptic curve P. This length is determined by the support of the two device's controllers. Here the SK$_a$is the secret key of the device A and the PK$_a$ is the public key of the device A. If there is no secure connections support, then the protocol will use P192. Alternatively, if secure connections mode is active, then P256 is used during the calculation of DHKey. Thus, at the end of this phase, a shared DHKey is generated which is used to generate Link Key in a later phase. [32] The elliptic curve P256 provides roughly equivalent security 128-bit secret key, yet the output shared secret is 256 bits. [33] In phase one, both devices select which pairing method should be used based on the Attribution protocol (ATT) values in the L2CAP layer.

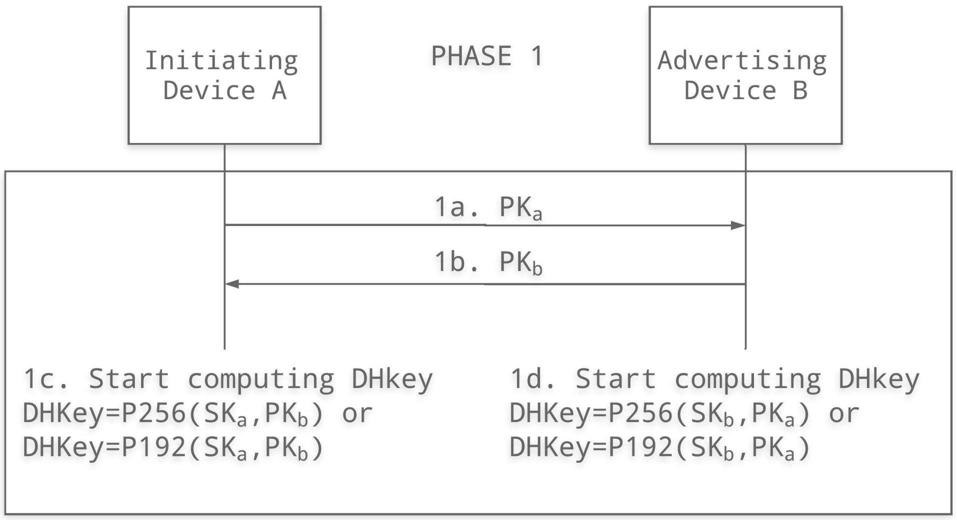
Initiating Device A
PHASE 1
Advertising Device B
1a. PK_a
1b. PK_b
1c. Start computing DHkey
DHKey=P256(SK_a,PK_b) or
DHKey=P192(SK_a,PK_b)
1d. Start computing DHkey
DHKey=P256(SK_b,PK_a) or
DHKey=P192(SK_b,PK_a)

Figure 3.3: SSP Phase 1: Public Key Exchange

If an adversary knows the frequency hopping pattern of both devices and performs a passive capture on the connection in this phase, they can obtain the public keys of device A and device B. This information is crucial to the attacker because the public keys are used to compute the commitment values in authentication stage 1. Alternatively, an attacker can also perform a MITM attack by intercepting the connection between the two devices and transmitting their public keys to both devices. This way, the attacker will have two DHKeys, where device A and the attacker will share the same DHkey and the other key will be shared with device B and the attacker.

3.4 SSP Phase 2: Authentication Stage 1

This stage consists of 3 association models. In Bluetooth LE, the just works model integrates with the numeric comparison model. Aside from these two models, we have the out of band model and the passkey entry model. This phase is dependent on the IO capabilities of both devices. Based on the IOCap, the compatible association model is selected. Since our research focus is on the passkey entry model, we will not be discussing the other association models in detail.

To prevent MITM attacks, the Bluetooth standard employs three association models that are implemented in the authentication stage 1 (phase 2) of the SSP. The models are as follows:

- **Just Works Model:** This model is ideally used for scenarios where at least one device has no output display or any input capability, e.g. wireless speaker. The just works model does not provide any protection against MITM attacks.

- **Numeric Comparison model:** This model is used when both devices have display capabilities as well as input capability for the user to enter "yes" or "no". This user confirmation is required for the device to know that the other device is legitimate and authentic and is, in fact, the same device the user is trying to connect to. [1] The numeric comparison model offers limited protection against MITM attacks with the attacker having a success probability of around 0.000001 on each iteration of the protocol.

- **Out of Band model:** This model is used when both devices can support communication over additional wireless channels e.g. near field communication (NFC). The out of band (OOB) channel ideally is resistant to MITM attacks. If not, there is a possibility that the data might be compromised during the authentication phase.

3.4.1 Passkey Entry Model

This model is used in scenarios where one device has input capability but does not have any output display capability. In this case, the user enters an identical 6-digit passkey on both devices. Alternatively, the passkey might be randomly generated by an algorithm in one device and displayed, which the user then inputs into the other device. The passkey entry model architecture is shown in the Figure 3.4.

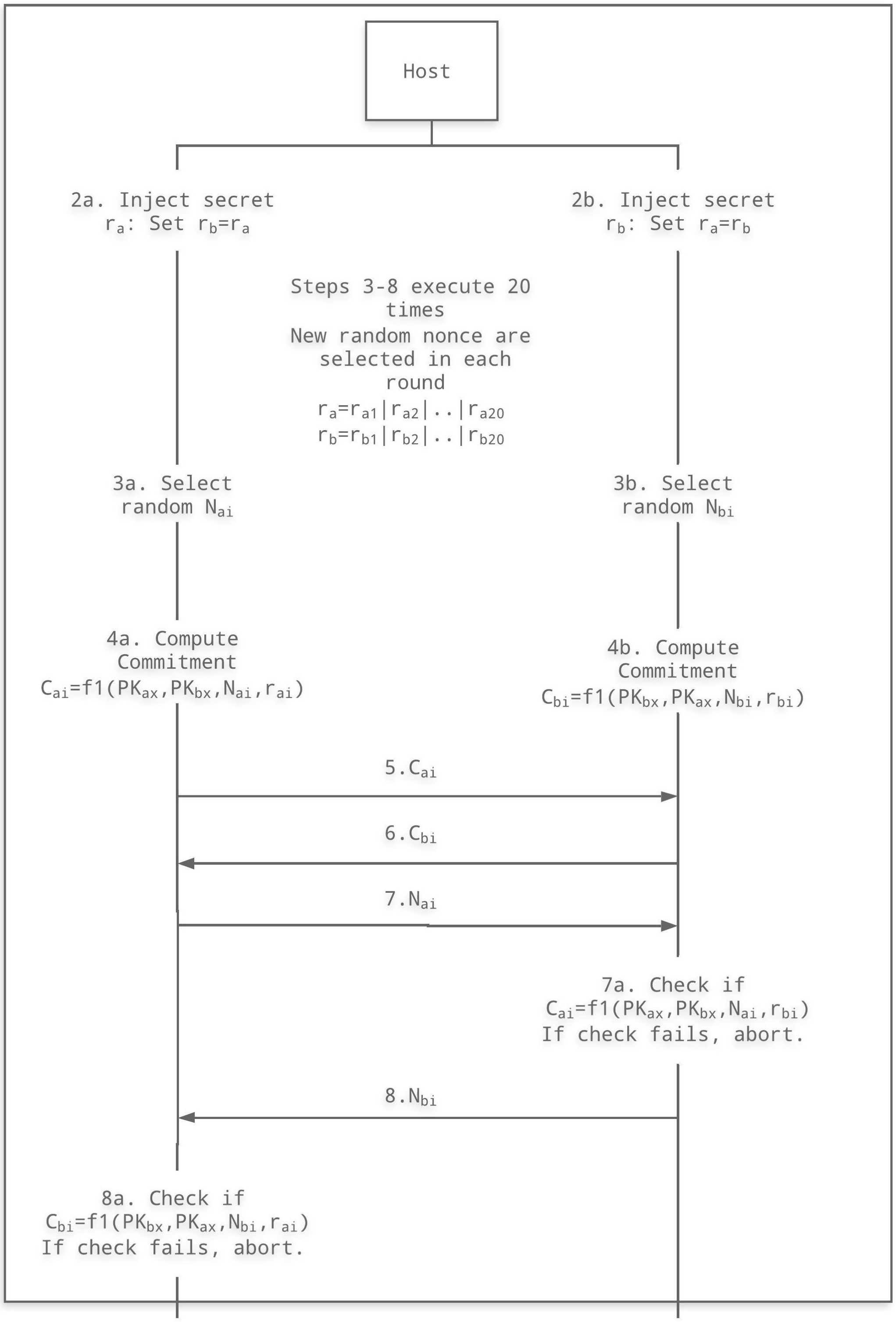
Host
2a. Inject secret r_a: Set r_b=r_a
2b. Inject secret r_b: Set r_a=r_b
Steps 3-8 execute 20 times
New random nonce are selected in each round
$r_a=r_{a1}|r_{a2}|..|r_{a20}$
$r_b=r_{b1}|r_{b2}|..|r_{b20}$
3a. Select random N_{ai}
3b. Select random N_{bi}
4a. Compute Commitment $C_{ai}=f1(PK_{ax},PK_{bx},N_{ai},r_{ai})$
4b. Compute Commitment $C_{bi}=f1(PK_{bx},PK_{ax},N_{bi},r_{bi})$
5.C_{ai}
6.C_{bi}
7.N_{ai}
7a. Check if $C_{ai}=f1(PK_{ax},PK_{bx},N_{ai},r_{bi})$ If check fails, abort.
8.N_{bi}
8a. Check if $C_{bi}=f1(PK_{bx},PK_{ax},N_{bi},r_{ai})$ If check fails, abort.

Figure 3.4: SSP Phase 2: Passkey Entry Protocol

Parameter	**Description**
$\mathbf{PK}_{ax}$	X-coordinate of public key of device A.
$\mathbf{PK}_{bx}$	X-coordinate of public key of device B.
$\mathbf{r}_a$ **or** $\mathbf{r}_b$	Random 6-digit passkey generated by User.
$\mathbf{r}_{ai}$	The i^{th} bit of 20-bit passkey sent by device A.
$\mathbf{r}_{bi}$	The i^{th} bit of 20-bit passkey sent by device B.
$\mathbf{N}_{ai}$	Random 128-bit Nonce.
$\mathbf{N}_{bi}$	Random 128-bit Nonce.
$\mathbf{C}_{ai}$	Commitment Value calculated by device A.
$\mathbf{C}_{bi}$	Commitment Value calculated by device B.

Table 3.5: Passkey Entry Parameters

In this model, the host or the user enters a 6-digit identical passkey on both devices A and B. The 6-digit passkey which is of 20-bit length is then stored into $\mathbf{r}_a$ and $\mathbf{r}_b$. After injecting the passkey, the steps 3-8 are repeated for 20 times and each round is denoted with the integer i which has a range of $1 <= i <= 20$. For every round, exactly one bit of passkey is sent to the other device. In each round, a 128-bit random nonce $\mathbf{N}_{ai}$ and $\mathbf{N}_{bi}$ is generated accordingly at both sides. After that, the device A computes a commitment value $\mathbf{C}_{ai}$ using $f1(\mathrm{PK}_{ax},\mathrm{PK}_{bx},\mathrm{N}_{ai},\mathrm{r}_{ai})$ and sends it to device B. B also generates the commitment value $\mathbf{C}_{bi}$ using $f1(\mathrm{PK}_{bx},\mathrm{PK}_{ax},\mathrm{N}_{bi},\mathrm{r}_{bi})$ and sends it to device A. After exchanging the commitment values, the device A sends its $\mathbf{N}_{ai}$ value to B. Now, the device B, uses the $\mathbf{N}_{ai}$ value sent by A to check if $\mathrm{C}_{ai} = f1(\mathrm{PK}_{ax},\mathrm{PK}_{bx},\mathrm{N}_{ai},\mathrm{r}_{bi})$ and if the check passes, the device B will also send its $\mathbf{N}_{bi}$ value. If the check does not pass, the protocol is aborted and the SSP session will need to be run again. The same procedure is followed at the device A also. According to the Bluetooth standard, the gradual disclosure of the passkey in this way prevents leakage of more than 1 bit of the passkey making it difficult for an MITM attacker to crack it. At the end of this stage, the values of $\mathbf{N}_a$ and $\mathbf{N}_b$ are set to $\mathbf{N}_a 20$ and $\mathbf{N}_b 20$ which are later used in the next stage, authentication stage 2. Here, $f1()$ is a cryptographic hash function used to calculate the commitment values of $\mathbf{C}_{ai}$ and $\mathbf{C}_{bi}$.

3.4.2 f1() function

The f1() function in the passkey entry phase model is cryptographic hash function which uses HMAC-SHA256. The 128-bit commitment values computed in step 4 uses this algorithm along with a 128-bit key for HMAC. This 128-bit key is the random nonce $\mathbf{N}_{\boldsymbol{a}}$ and $\mathbf{N}_{\boldsymbol{b}}$. For the simple pairing f1() function, there are a total of four inputs. The f1() function is defined as

$$f1() = \text{HMAC-SHA256}_X(U||V||Z)/2^{128}$$

Input Parameter	**Size in P192**	**Size in P256**
U	192	256
V	192	256
X	128	128
Z	8	8

Table 3.6: f1() function Inputs

At device A, the commitment values are generated by taking the $\mathbf{PK}_{\boldsymbol{ax}}$, $\mathbf{PK}_{\boldsymbol{bx}}$, $\mathbf{N}_{\boldsymbol{ai}}$, and $\mathbf{r}_{\boldsymbol{ai}}$ as inputs to f1() function. The same procedure at device B. The random 128-bit nonce acts as the input X and is used as the key for HMAC function. Using the 128-bit random nonce provides security as it is highly difficult for an attacker to brute a 128-bit passkey. In addition to this, a new random nonce is generated in each round. [1] $\mathbf{r}_{\boldsymbol{ai}}$ is one bit value of the passkey expanded to 8 bits (either 0x80 if $\mathbf{r}_{\boldsymbol{a}} = 0$ or 0x81 if $\mathbf{r}_{\boldsymbol{a}} = 1$).

3.4.2.1 HMAC-SHA-256

The Hashed Message Authentication Code (HMAC) is a variant of the MAC which involves a cryptographic hash function like SHA-2 and a secret cryptographic key. The main goal of a MAC function is to provide data integrity and the authenticity of a message. [34] When a message of any length less than 2^{64} bits (for SHA-256) is input to a hash algorithm, the result is an output called a message digest (MD). The SHA-256 is a secure hash algorithm because it produces a 256-bit unique MD which is impossible for an attacker to brute-force through. In the Bluetooth standard, the SHA-256 is used as the hash algorithm with the HMAC function. The HMAC is run in two phases.

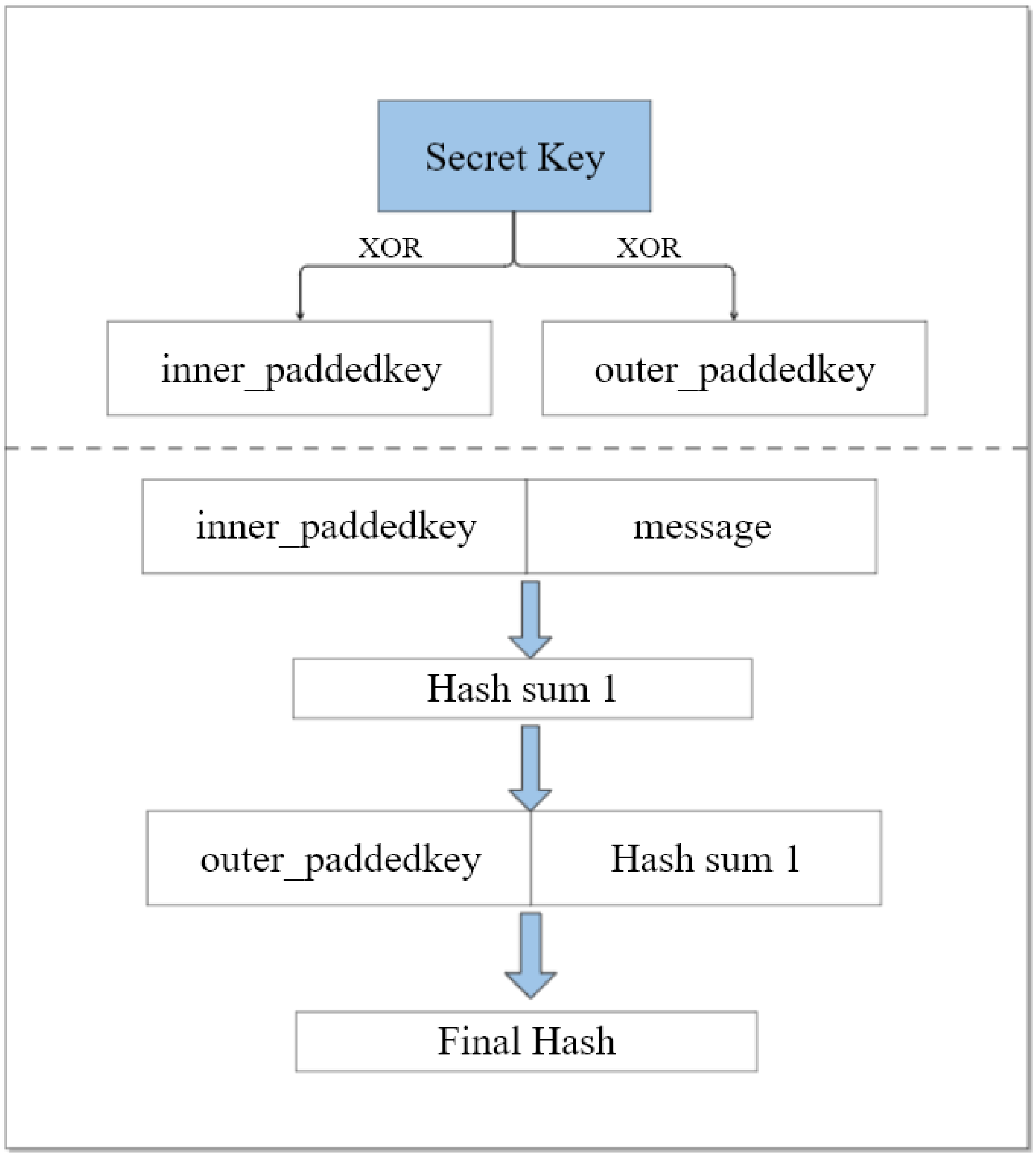

Figure 3.5: HMAC-SHA256 basic operation

As shown in Figure 3.5, when a secret key is selected, the HMAC uses the secret key to generate two different keys. The secret key is XORed and two keys are derived by performing inner padding and outer padding. In the next phase, the input message is concatenated with the ***inner_paddedkey*** which uses repeating bytes of value 0x5c and the SHA-256 is used to generate a unique hash. After this, the resultant hash is concatenated with the ***outer_paddedkey*** which uses repeating bytes of value 0x36 and a new hash is generated. The resultant hash is the final output of the HMAC function. [35] The reason for using the hash function two times is to provide better protection against length extension attacks. The strength of an HMAC function greatly depends upon the length of the secret key and the hash algorithm used.

3.5 SSP Phase 3: Authentication Stage 2

With the completion of the authentication stage 1, the devices move to the next phase of the protocol to complete the second stage of the authentication which is the message exchange between the devices.

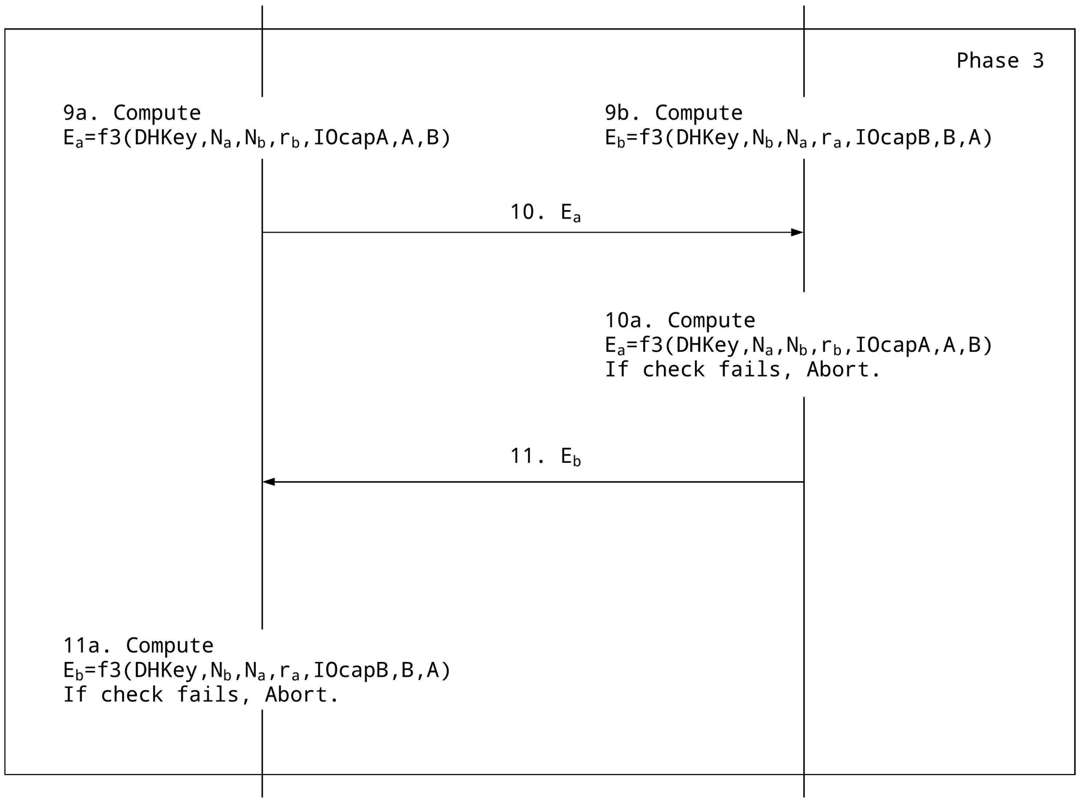
Phase 3
9a. Compute
E_a=f3(DHKey,N_a,N_b,r_b,IOcapA,A,B)
9b. Compute
E_b=f3(DHKey,N_b,N_a,r_a,IOcapB,B,A)
10. E_a
10a. Compute
E_a=f3(DHKey,N_a,N_b,r_b,IOcapA,A,B)
If check fails, Abort.
11. E_b
11a. Compute
E_b=f3(DHKey,N_b,N_a,r_a,IOcapB,B,A)
If check fails, Abort.

Figure 3.6: SSP Phase 3: Authentication Stage 2

Parameter	**Description**
r$_a$ or r$_b$	Entire 20-bit passkey.
IOcapA or IOcapB	IO capabilities of device A and device B.

Table 3.7: SSP Phase 3 Parameters

In this phase, the device A computes a new confirmation value Ea using $f3(DHkey, N_a, N_b, r_b, IOcapA, A, B)$ and B computes a confirmation value E_b using $f3(DHkey, N_b, N_a, r_a, IOcapB, B, A)$ simultaneously. After the computations, the device A sends the E_a to device B where B checks the E_a with $f3(DHKey, N_a, N_b, r_b, IOcapA, A, B)$. If the check fails, the protocol aborts. If the check passes, the value of E_b is sent to device A where A also checks the E_b with $f3(DHKey, N_b, N_a, r_a, IOcapB, B, A)$. If the check fails, the protocol aborts. Here the value of $\mathbf{r_a}$ and $\mathbf{r_b}$ is the entire 20-bit passkey. With this step, the message exchange between both devices is complete and the DHkey values are verified.

3.5.1 f3() function

The f3() function uses the HMAC-SHA-256 algorithm to generate and verify the confirmation values E_a and E_b. It uses the shared DHkey as the secret key for HMAC function. The f3() function has a total of six inputs. The f3() function is defined as

$$f2() = \text{HMAC-SHA256}_W(N1||N2||R||IOcap||A_1||A_2)/2^{128}$$

Here, the input A_1 is the 48-bit unique BD_ADDR of device A and the input A_2 is the BD_ADDR of device B. The R is defined as the 6-digit passkey.

Input Parameter	Size in P192	Size in P256
W	192	256
N_1	128	128
N_2	128	128
IOcap	24	24
A_1	48	48
A_2	48	48

Table 3.8: f3() function Inputs

3.6 SSP Phase 4: Link Key Calculation

After the authentication stage 2 is successful between two devices, a shared link key is generated by using $f2(DHkey, N_a, N_b, btlk, BD\_ADDR_a, BD\_ADDR_b)$ at both sides. The order of the parameters must be the same in order to calculate the same key. The end goal of an attacker is to obtain this key.

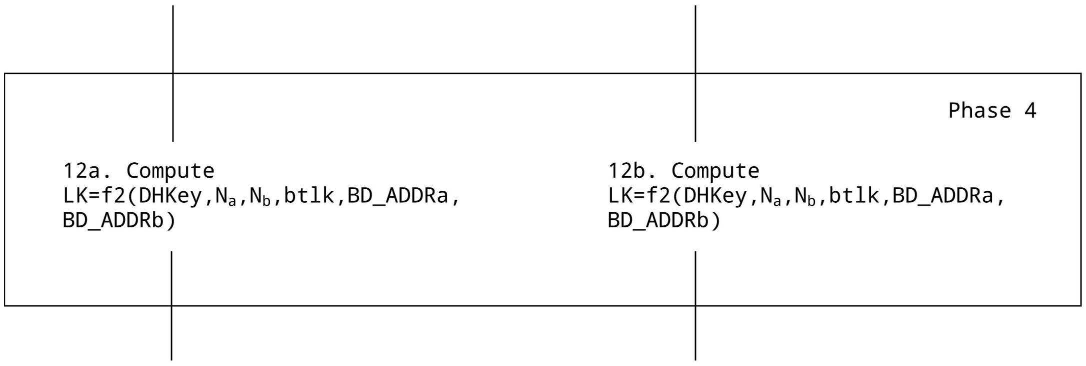
Phase 4
12a. Compute LK=f2(DHKey,Na,Nb,btlk,BD_ADDRa, BD_ADDRb)
12b. Compute LK=f2(DHKey,Na,Nb,btlk,BD_ADDRa, BD_ADDRb)

Figure 3.7: SSP Phase 4: Link Key Calculation

3.6.1 f2() function

Similar to the f3() function in phase 2 of SSP, this uses the HMAC-SHA-256 algorithm with the DHKey as the key for HMAC. As shown in Figure 3.7, the f2() function is mainly used for the generation of a Link key. For the simple pairing f2() function, there are a total of six inputs. [1] The output of the f2 function is taken as the 128 most significant bits of the output of HMAC-SHA-256. The f2() function is defined as

$$f2() = \text{HMAC-SHA256}_W(N1||N2||keyID||A_1||A_2)/2^{128}$$

Input Parameter	**Size in P192**	**Size in P256**
W	192	256
N_1	128	128
N_2	128	128
keyID	32	32
A_1	48	48
A_2	48	48

Table 3.9: f2() function Inputs

3.7 SSP Phase 5: LMP Authentication and Encryption

Once the shared link key is established at both sides, the authentication process and the encryption key generation process is completed based on the link key. The steps followed in this phase are identical to the ones in the legacy pairing. Since we are only focusing on the passkey entry model, we do not delve into the details of this phase.

3.8 Vulnerability in Passkey Entry Protocol

Consider a scenario where a peripheral Bluetooth device starts advertising within its range. The master device initiates a connection with the peripheral device by sending its BD_ADDR, clock information, and the frequency hopping pattern. Both devices are now synchronized with each other and form a piconet where the initiating device is the master and the advertising device is the slave. If an attacker was able to capture the frequency hopping pattern of these devices in the piconet, they can establish passive eavesdropping for the subsequent transmission packets. When the two legitimate devices exchange their IOcap and start the SSP process, the attacker can start passively capturing packets. Note that the Bluetooth standard uses ECDH public key exchange in phase 1 to prevent passive eavesdropping. It is very important to make sure the DHkey is always secure in case of passive eavesdropping. An attacker can then passively capture all the packets sent during the public key exchange and the passkey entry protocol and try to deduce the passkey used. In the Phase 2 or authentication stage 1 phase, the Bluetooth standard uses the commitment protocol to make sure that both parties have the same passkey. That method works well if the user will change the passkey each time when the connection has a problem. Unfortunately, in practice very likely users will use the same passkey for many times, which will cause serious security problems.

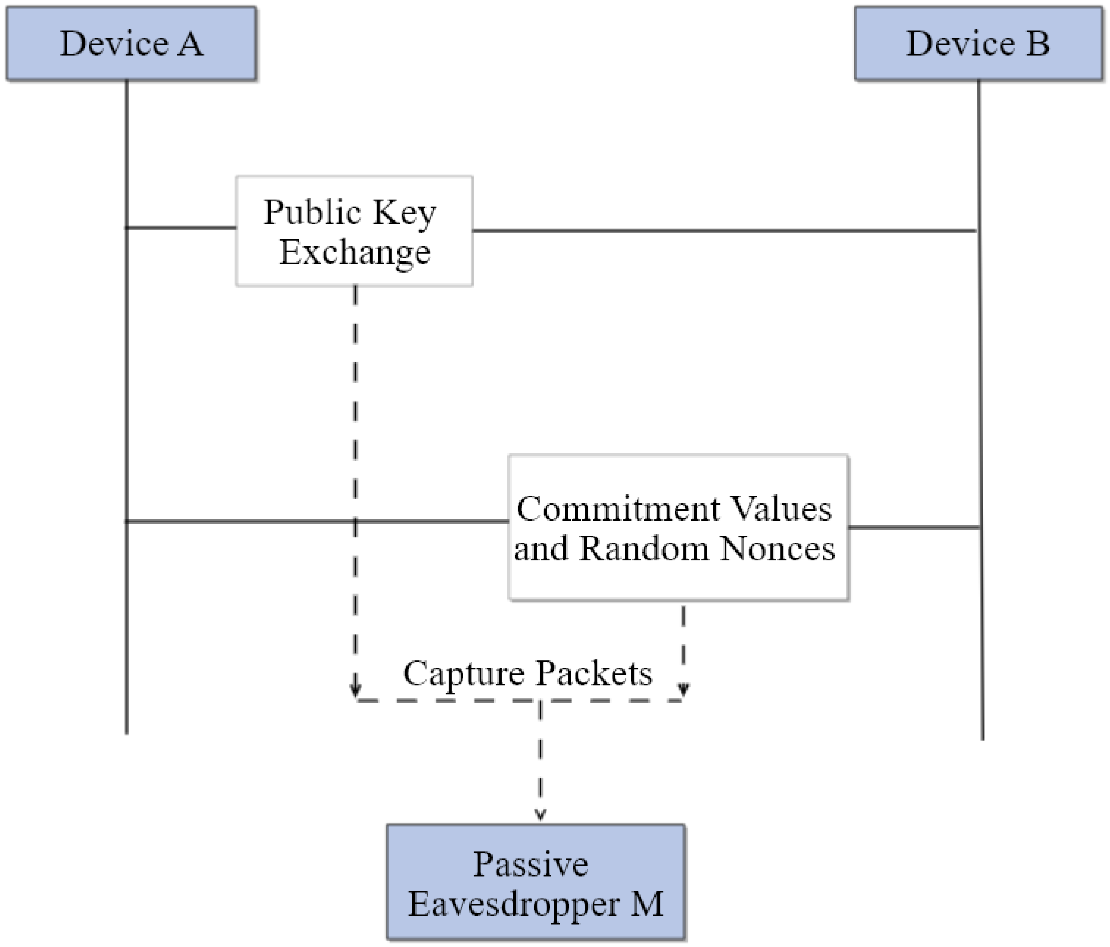

Figure 3.8: Passive Eavesdropping on SSP session

For instance, a passive attacker can capture all the commitment values (C_{ai} and C_{bi}), random nonce (N_{ai} and N_{bi}) and deduce each bit of passkey `r` by computing their own commitment $f1(Pk_{ax}, PK_{bx}, N_{ai}, r'_{ai})$ and comparing it with the original value. Since the value of the r$_{ai}$ is always either a 1 or 0, the attacker can easily obtain the entire passkey. In this way, the attacker can figure out the correct r_{ai}. Ideally, we should not reuse the same passkey again, however, most users tend to reuse the old passkey again for simplicity and ease. This enables the attacker to launch a MITM attack with the known passkey successfully communicating with the two devices simultaneously. In this way, after capturing all the packets and deducing the passkey, the attacker will proceed to launch a MITM attack with the obtained passkey.

To launch a MITM attack, the attacker will follow the same procedure as that of the passive capture. They will first acquire the frequency hopping pattern. Then, they will intercept the communication packets exchanged between the two legitimate devices and modify the data with their data. Since the public keys used in Bluetooth do not have certificates, the Bluetooth standard uses phase 2 to check the authentication. The main idea is that if there is a MITM attack in Phase 1, then the Bluetooth standard will try to prevent the attacker from connecting by using one of the association models in Phase 2. So we can always assume that there is a MITM attack in Phase 1, and then check if the authentication process in Phase 2 always works. As we have already indicated that the attacker can perform a MITM attack in Phase 1, we can see that the attacker will be able to know all the information except the passkey $\mathtt{r}$ in Phase 2. However, if the attacker has already deduced the passkey using passive eavesdropping technique described earlier, then they can execute a MITM attack with the passkey knowledge and complete the authentication stage 1 successfully. By achieving this, the attacker and device A will compute the same LTK and the device B and the attacker will compute the same LTK. Based on this, an encryption key is derived. Here, device A and device B will believe that the connection was successful and begin to transfer data. But in reality, the device A and B will be sending the data to the attacker. In addition to this, the attacker can also request for sensitive data with the two devices. Note that if a MITM attack is successful, then the connection between device A and device B will only work when the attacker is present because device A and device B will have different LTKs.

As shown in Figure 3.9, after deducing the passkey from passive eavesdropping, the attacker M can simply execute a MITM attack. Since ECDH does not provide any authentication, the attacker can spoof their public keys and initiate a connection with device A and B. The attacker sends their public key $\mathrm{PK}'_{\boldsymbol{e}}$ to device A and device B. Here, the device A thinking that the public key belongs to device B accepts the public key of attacker E and computes the DHkey. The device B follows similar steps as of A. After the public key exchange is completed, the attacker will enter the same passkey which was used for their previous session. Since the attacker deduced the passkey from passive eavesdropping, they follow the same steps as the other devices. In step 3, attacker will generate a random nonce $\mathbf{N}'_{\boldsymbol{mi}}$ and compute the commitment value $\mathbf{C}'_{\boldsymbol{ei}}$. The computed commitment values are sent to device A and B. Both devices, A and B, verify the commitment values by using the $\mathrm{r}_{\boldsymbol{a}}$ and $\mathrm{r}_{\boldsymbol{b}}$. The verification will be successful and the next bit is sent. This process is repeated for 20 rounds where one bit of the 20-bit passkey is exchanged between the attacker and the target devices A and B. Since the value of the passkey is the same, the passkey entry protocol will successfully finish and the attacker will go to the next phase of SSP. The authentication stage 2 will be completed and the link keys will be calculated. Here, two link keys are generated where device A and the attacker will have the same link key and the device B and the attacker will have another same link key.

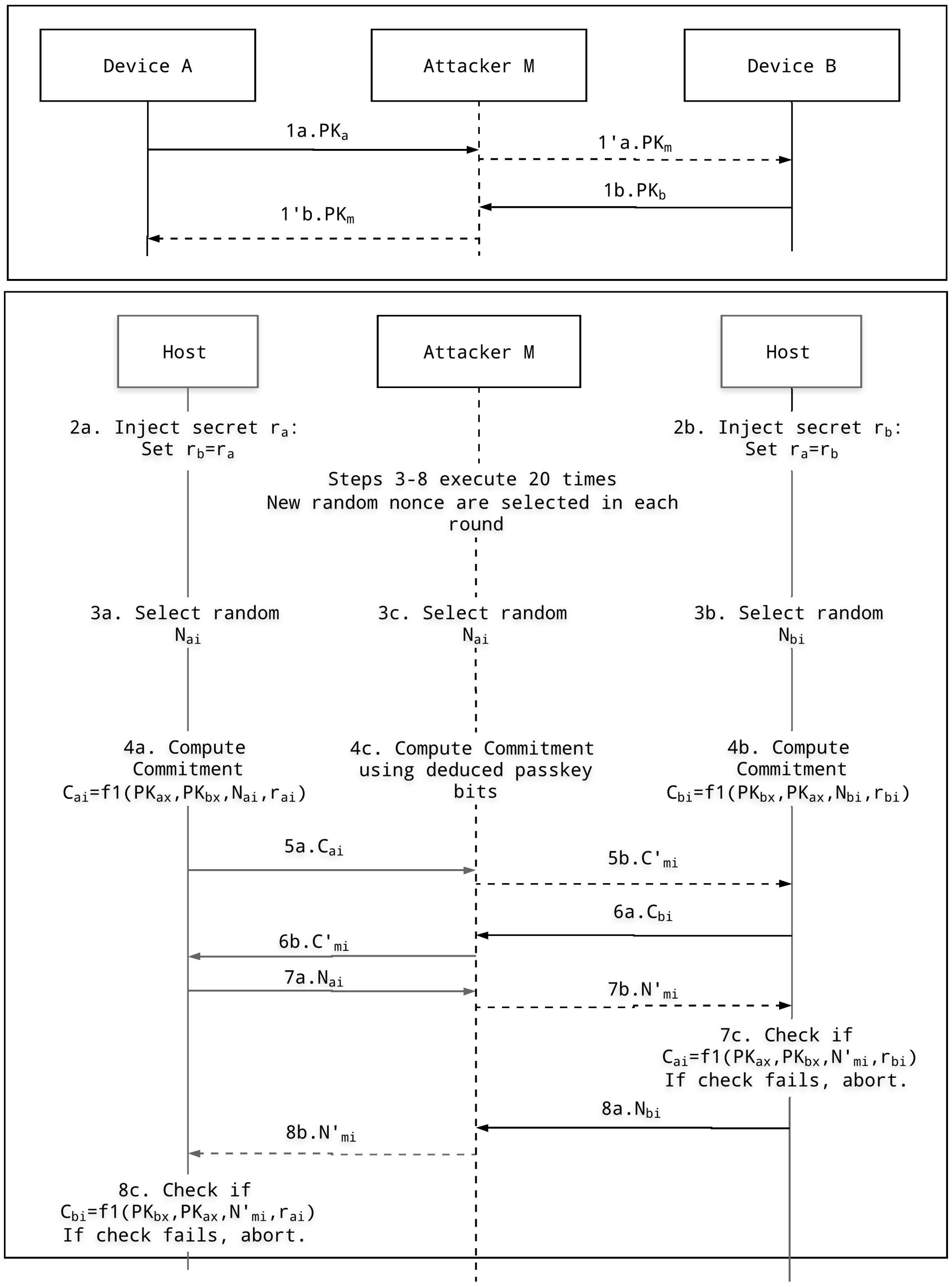
Device A
Attacker M
Device B
1a.PK_a
1'a.PK_m
1b.PK_b
1'b.PK_m
Host
Attacker M
Host
2a. Inject secret r_a: Set r_b=r_a
2b. Inject secret r_b: Set r_a=r_b
Steps 3-8 execute 20 times New random nonce are selected in each round
3a. Select random N_ai
3c. Select random N_ai
3b. Select random N_bi
4a. Compute Commitment C_ai=f1(PK_ax,PK_bx,N_ai,r_ai)
4c. Compute Commitment using deduced passkey bits
4b. Compute Commitment C_bi=f1(PK_bx,PK_ax,N_bi,r_bi)
5a.C_ai
5b.C'_mi
6a.C_bi
6b.C'_mi
7a.N_ai
7b.N'_mi
7c. Check if C_ai=f1(PK_ax,PK_bx,N'_mi,r_bi) If check fails, abort.
8a.N_bi
8b.N'_mi
8c. Check if C_bi=f1(PK_bx,PK_ax,N'_mi,r_ai) If check fails, abort.

Figure 3.9: MITM on SSP session

Chapter 4

Enhanced Passkey Entry Protocol

4.1 An Improved protocol by Sun and Mu

Sun and Mu in [27] proposed an improved passkey entry protocol as a countermeasure to passkey reuse. We will refer to this protocol as the SM protocol. They proposed an improved passkey entry protocol in which before the run of sending each bit of passkey (steps 3-8) in Figure 3.4, they generate a new random nonce on both sides and exchange them. Then, they compute $r = f2(DHkey, N_{a0}, N_{b0}, r_a)$ at device A and $r = f2(DHkey, N_{a0}, N_{b0}, r_b)$ and set the six most significant digits of r to r_a^*. This protocol needs a generation of two additional 128-bit random nonces as well as two additional cryptographic hash function f2() which is originally used in the generation of link key in SSP phase 4.

As shown in Figure 4.1, after injecting the passkey r on both sides, the devices A and B generate two random nonces Na0 and Nb0. The two nonces are sent to each other. Then, on device A, a new hash value is computed as $r = f2(DHkey, N_{a0}, N_{b0}, r_a)$ and the six most significant digits of r is set to $\mathbf{r_a^*}$. The device B computes the hash value $r = f2(DHkey, N_{a0}, N_{b0}, r_b)$ and sets the six most significant digits of r to $\mathbf{r_b^*}$. The rest of the protocol, the steps 5-10, is run the same as that of the original passkey entry model. In their protocol, the random passkey r is used as a seed of the authentication passkey.

Now, even if the user uses the same passkey next time, the attacker cannot easily attack since each time the values of $\mathbf{r_a^*}$ and $\mathbf{r_b^*}$ will be different.

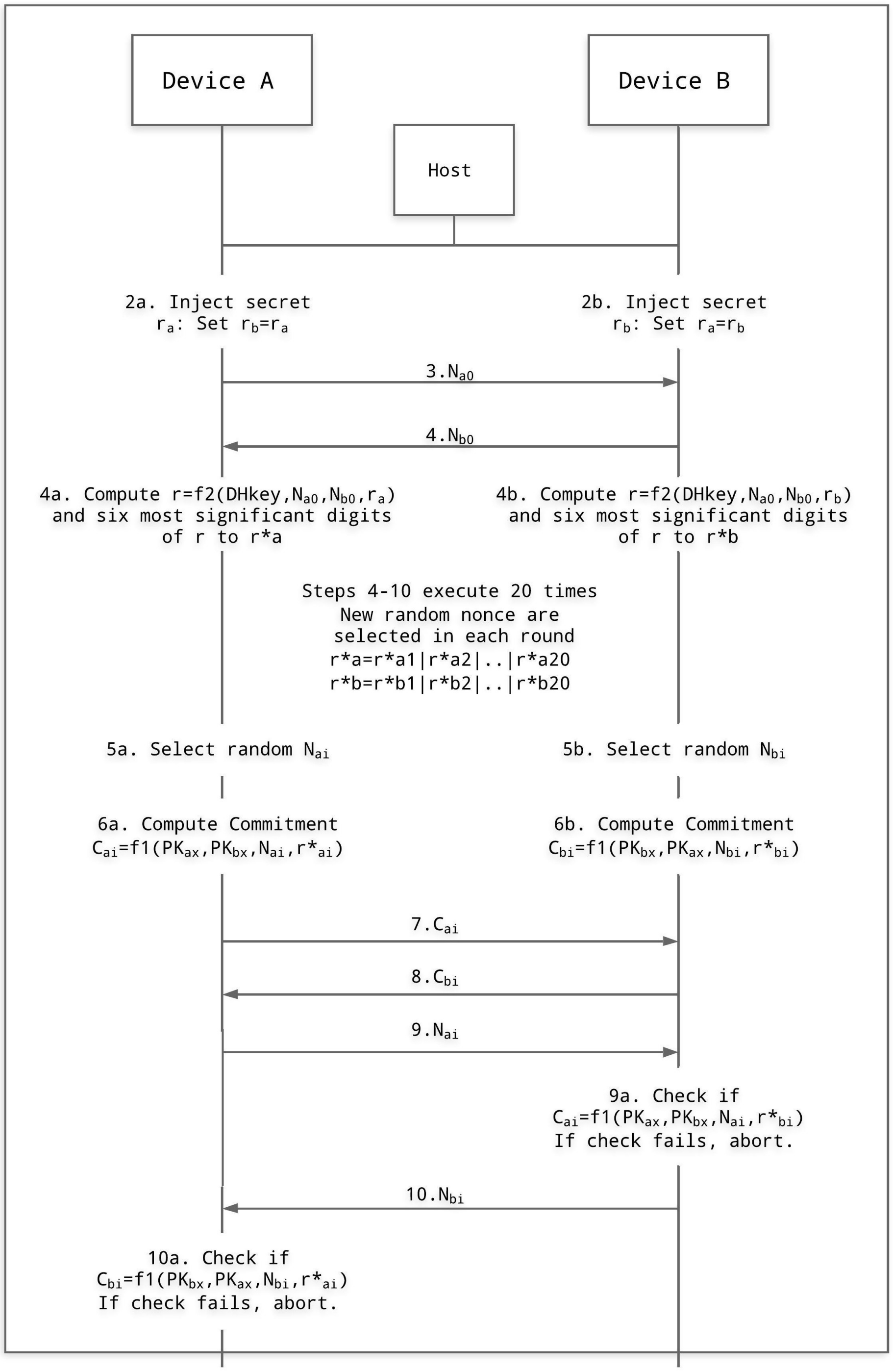
Device A
Device B
Host
2a. Inject secret r_a: Set $r_b=r_a$
2b. Inject secret r_b: Set $r_a=r_b$
3.N_{a0}
4.N_{b0}
4a. Compute $r=f2(DHkey,N_{a0},N_{b0},r_a)$ and six most significant digits of r to r*a
4b. Compute $r=f2(DHkey,N_{a0},N_{b0},r_b)$ and six most significant digits of r to r*b
Steps 4-10 execute 20 times
New random nonce are selected in each round
r*a=r*a1|r*a2|..|r*a20
r*b=r*b1|r*b2|..|r*b20
5a. Select random N_{ai}
5b. Select random N_{bi}
6a. Compute Commitment $C_{ai}=f1(PK_{ax},PK_{bx},N_{ai},r^*_{ai})$
6b. Compute Commitment $C_{bi}=f1(PK_{bx},PK_{ax},N_{bi},r^*_{bi})$
7.C_{ai}
8.C_{bi}
9.N_{ai}
9a. Check if $C_{ai}=f1(PK_{ax},PK_{bx},N_{ai},r^*_{bi})$ If check fails, abort.
10.N_{bi}
10a. Check if $C_{bi}=f1(PK_{bx},PK_{ax},N_{bi},r^*_{ai})$ If check fails, abort.

Figure 4.1: Da-zhi Sun Improved Passkey Entry Model

4.1.1 An attack on SM protocol

For our attack to work, we are assuming that an adversary has achieved the following.

- They know the frequency hopping pattern and have intercepted the communication packets.
- Modify the communication packets and send the modified data to both legitimate devices so that one computed DHkey will be equal with device A and the attacker and other DHkey will be equal with the attacker and device B.
- Attacker has obtained at least three to four r* bits from the MITM attacks conducted on SSP sessions.

Consider a scenario where an attacker is simultaneously trying to establish a connection with both devices A and B. If we assume that the device A and device B have a fault tolerance system where they allow three to four consecutive SSP sessions with minimal delay in the event of failure, the attacker can try to establish consecutive sessions to device A and device B event in the event of failure of prediction of bits. When an attacker has successfully completed the public key exchange, then they have two DHKey values, one with device A and another with device B. If we assume that an attacker was able to correctly predict around 8-bits of the r_a^* and r_b^*, then a simple brute-force would greatly increase the chances of the attacker to predict the correct passkey. This is because aside from the passkey r, the attacker knows the values of DHkey, N_{a0}, and N_{b0}. So after the SSP session fails due to the incorrect passkey bit, the attacker can simply brute-force all the combinations of passkey and match the output with the 8-bits of r_a^*. Out of the passkeys which match, there is going to be the correct passkey. The attacker can now take the list of the matched passkeys and perform a dictionary attack and try those combinations with the first 8 bits of r_b^* of another SSP session. By doing this, the attacker can eventually obtain the correct passkey within 2-3 SSP sessions. When the passkey is known, the attacker will simply execute another MITM attack but this time, they will be successful because of the correct passkey and will be able to obtain the LTK.

By using our brute attack given in Algorithm 1, an attacker can easily find the passkey by trying the potential passkey list. For our simulation of the brute-force attack, we used a desktop computer with an Intel i7 2.20GHz processor. The simulation took less than 12 seconds to obtain the correct passkey for 2 consecutive SSP sessions. Note that for our experiment, we are considering that an attacker has successfully obtained at least 6 bits of r_a^* and r_b^* for one SSP session. Even with a shorter r_a^* and r_b^*, the attacker can still try to brute-force but will most likely need to execute MITM on more SSP sessions. But eventually, the list of potential passkeys will only decrease and the attacker will obtain the correct passkey. The main reason for this attack to succeed is that we have a limited keyspace of 2^{20}.

If the attacker knows partial digits of the passkey then it becomes even easier for them to crack it possibly in one session.

In the table 4.1, we use our brute-force algorithm to try and deduce the correct passkey. The inputs of the algorithm are as follows.

- The DHkey value which is shared between the target device and the adversary.
- The obtained r* bits from the SSP session with the target device.
- The 128-bit random nonce, N$_0$, sent by the target device to the adversary.
- The 128-bit random nonce, N$_{x0}$, generated by adversary X.

For each session, these values are updated again. In our attack, we are assuming that the attacker knows 8 bits of r$_a^*$ and r$_b^*$. And since the attacker already knows the values of DHkey, N$_{a0}$, and N$_{b0}$, we compute $r = f2(DHkey, N_{a0}, N_{x0}, i)$ where i is the potential passkey that the legitimate user has used. Now, we set the n most significant bits of r to r$'_x$ where n is the number of bits an attacker has obtained during the MITM attack. If the value of r$'_x$ is equal to the first n obtained r bits, then the value of i is added to the list of potential passkeys. A new combination i is tried and if the r$'_x$ is equal to the first n obtained of r bits, then the potential passkey list is updated. This process is repeated until all combinations are exhausted and the potential passkey list is obtained. Then, using that potential passkey list, we move on to the next round, where we use the obtained potential passkeys from round 1 and execute $r = f2(DHkey, N_{x0}, N_{b0}, i)$ where i is one of the potential passkeys. Similar to round 1, we set the n most significant bits of r to r$'_x$ and compare the value with obtained r bits of device B. We repeat the process until the list of passkeys is tried. If equal, we update our potential passkey list. Note that this is still only one SSP session where the attacker has obtained the r$_b^*$ bits from device B. We repeat the rounds with an updated potential passkey list for every round until there is only one potential passkey left which is the correct passkey. The attacker can also significantly increase the chances of deducing the right passkey if they know at least 1 or 2 digits of the passkey as it greatly reduces the number of combinations to brute-force. Note that we have executed our brute-force attack by taking every possible combination of the 6-digit passkey whereas ideally, it is said that the user must enter at least a 4-digit passkey. If the attacker knows that the length of the passkey should always be greater than 3, then the possible combinations will also decrease increasing the attack success probability.

	No. of Potential Passkeys (Output)		
Values	**Test Case 1**	**Test Case 2**	**Test Case 3**
Round 1 **Execute f2(DHKey, N$_{a0}$, N$_{x0}$, i) obtained from device A and attacker X in 1st SSP session.**	7710	7835	7933
Round 2 **Execute f2(DHKey, N$_{x0}$, N$_{b0}$, i) obtained from device B and attacker X in 1st SSP session.**	56	70	67
Round 3 **Execute f2(DHKey, N$_{a0}$, N$_{x0}$, i) obtained from device A and attacker X in 2nd SSP session.**	1	1	1
Correct Passkey	893562	120736	544557
Execution Time	8.75 Seconds	8.86 Seconds	8.88 Seconds

Table 4.1: Experiment Test Cases

Algorithm 1 brute-force algorithm

```
 1: Initialise DHkey, $N_{a0}$, $N_{b0}$, newlist[] and count = 0 obtained from first SSP
    session.
 2: for i in range of (0..1000000) do
 3:     if i$\leq$1000000 then
 4:         prepend zeros to i to make it 6-digit.
 5:     end if
 6:     Generate $r_x$ = HMAC-SHA256(DHkey, $N_{a0}$, $N_{b0}$, i)
 7:     Set n most significant of $r_x$ to $r'_x$ where n is the obtained number of $r^*_a$ bits
    by the attacker.
 8:     if $r'_x$ ==$r^*_a$ then
 9:         newlist[count] = i (Adding i to the list of potential passkeys.)
10:         count = count + 1
11:         print(Match found. Potential Passkey = i)
12:     end if
13: end for
14: procedure CONSECUTIVEBRUTE(DHkey, $N_{a0}$, $N_{b0}$, newlist, count)
15:     Initialise DHkey, $N_{a0}$, $N_{b0}$ obtained from next consecutive SSP session.
16:     newcount = 0
17:     for i in range of (0..count) do
18:         if newlist[i]$\leq$100000 then
19:             prepend zeros to newlist[i] to make it 6-digit.
20:         end if
21:         Generate $r_x$ = HMAC-SHA256(DHkey, $N_{a0}$, $N_{b0}$, newlist[i])
22:         Set n most significant of $r_x$ to $r'_x$ where n is obtained number of $r^*_a$ bits
    by the attacker.
23:         if $r'_x$ ==$r^*_a$ then
24:             newlist[newcount] = i (Adding i to the list of potential passkeys.)
25:             newcount = newcount + 1
26:             print(Match found. Potential Passkey = newlist[i])
27:         end if
28:     end for
29: end procedure
30: Execute consecutivebrute(DHkey, $N_{a0}$, $N_{b0}$, newlist, count) with a new list of
    potential passkeys until the correct passkey is found.
```

We have done a simulation where we tested the resilience of the SM passkey entry model against the brute-force attack. For this specific simulation of the brute-force attack, we used a desktop computer with an Intel i7 2.20GHz processor. We have performed the brute-force attack when 4,5,6 and 7 bits of r* are known to the attacker. The brute-force attack was conducted 50 times with each known r* value and the average SSP sessions required were calculated for each r* value as shown in Figure 4.2.

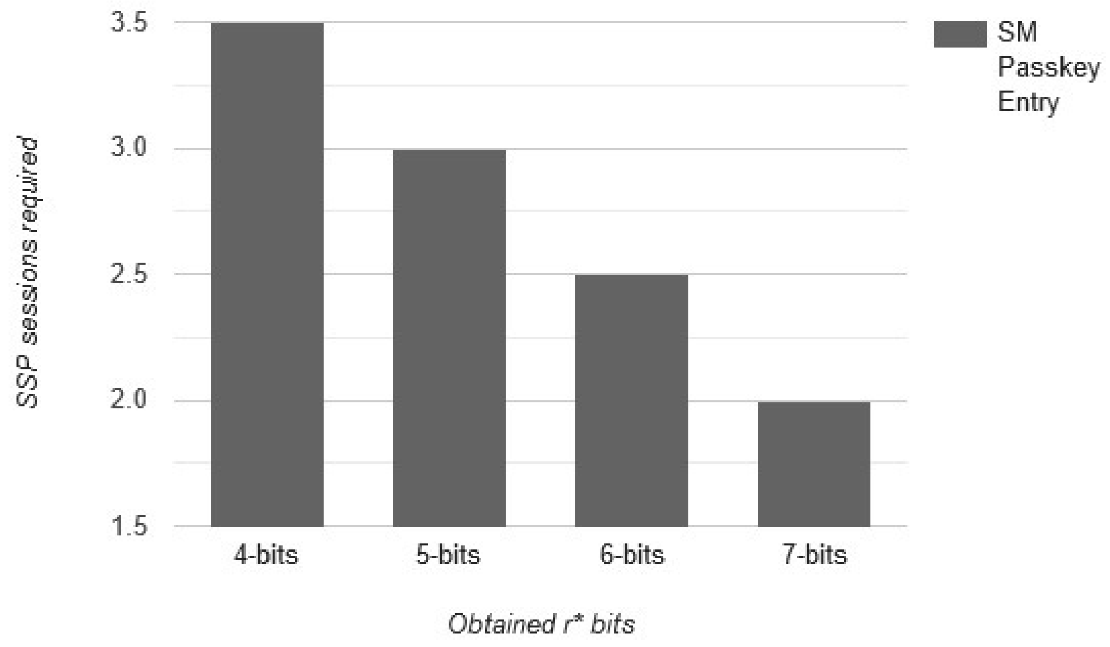

Figure 4.2: SM protocol resistance against brute-force attack

It is easy to see that if the attacker knew the DHkey, then they will be easily able to get the value of r_a^*. Alternatively, the attacker can try to set up a random 20-bit string $\hat{r_a^*}$. Then use the same formula to calculate C_i^a, except using the bit from $\hat{r_a^*}$ instead of r_a^*. By comparing the value with the value C_i^a downloaded, they can correct $\hat{r_a^*}$ bit by bit to get the exact value of r_a^*.

We note that, for security reasons, we may assume that an attacker knows all the information except the value or r_a in the authentication stage of SSP Phase 3. At this point, we will think that the SM protocol is not resistant enough against the aforementioned brute-force attack to prevent the reusing passkey problem.

4.2 Proposed Enhanced Passkey Protocol

To improve the security of the passkey protocol, we propose an alternate to [27], our enhanced passkey entry protocol. Our protocol does not differ much from the original protocol in Bluetooth standard and in addition to that, there are only 10 rounds in our protocol which significantly reduce the power and communication costs of the devices. The main idea of our protocol is to never reveal the original passkey until phase 3 of the SSP. We also make use of the properties of the DHKey in the ECDH algorithm to generate a 256-bit hash from passkey r. Therefore, we use a derivative hash of the original passkey r and use that to run the entire passkey entry protocol. In our protocol, there are 2 different bits exchanged and verified at each side for each round resulting in 20-bits being exchanged altogether.

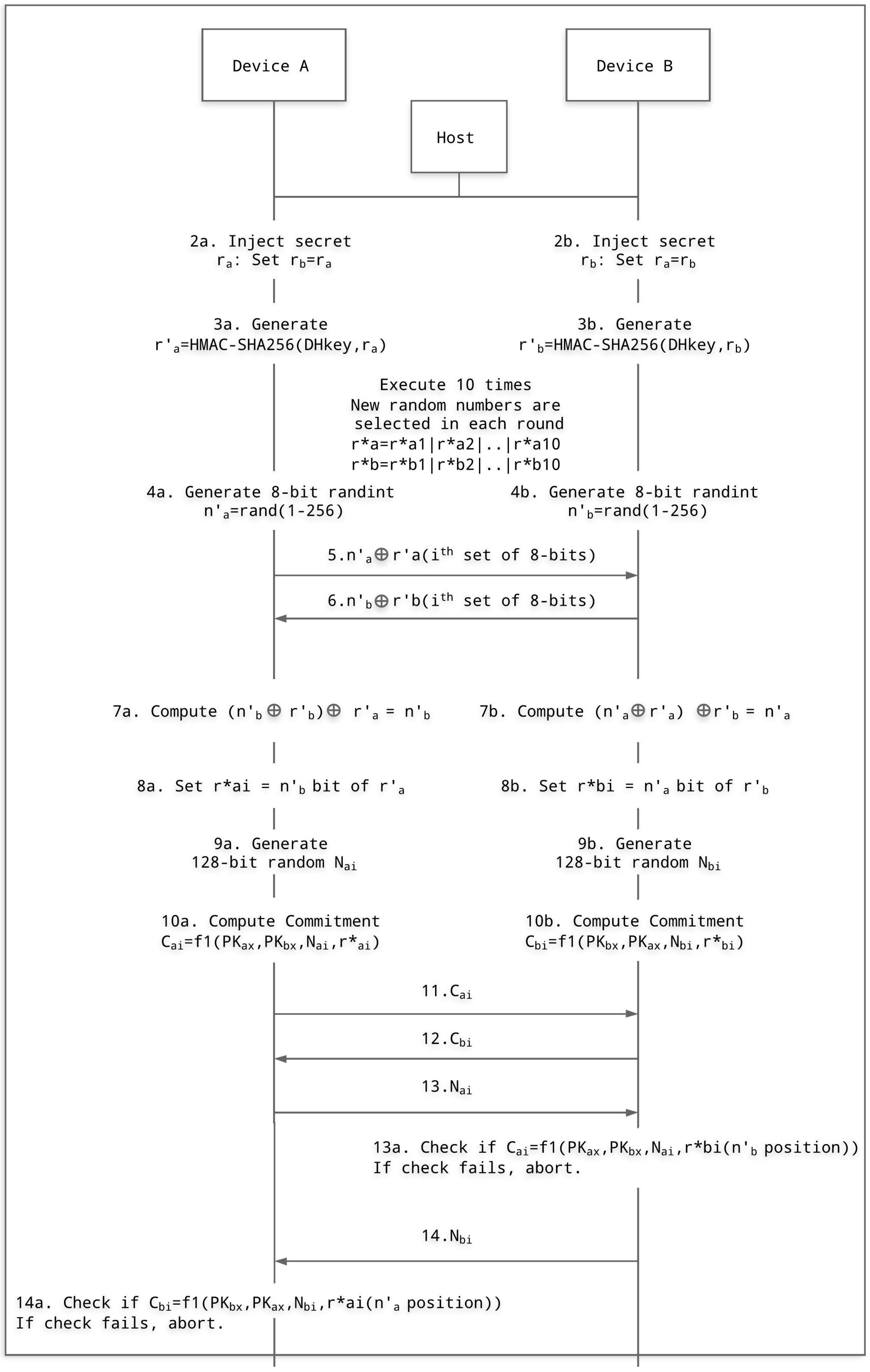
Device A
Device B
Host
2a. Inject secret r_a: Set r_b=r_a
2b. Inject secret r_b: Set r_a=r_b
3a. Generate r'_a=HMAC-SHA256(DHkey,r_a)
3b. Generate r'_b=HMAC-SHA256(DHkey,r_b)
Execute 10 times
New random numbers are selected in each round
r*a=r*a1|r*a2|..|r*a10
r*b=r*b1|r*b2|..|r*b10
4a. Generate 8-bit randint n'_a=rand(1-256)
4b. Generate 8-bit randint n'_b=rand(1-256)
5.$n'_a \oplus$ r'a(ith set of 8-bits)
6.$n'_b \oplus$ r'b(ith set of 8-bits)
7a. Compute $(n'_b \oplus r'_b) \oplus r'_a = n'_b$
7b. Compute $(n'_a \oplus r'_a) \oplus r'_b = n'_a$
8a. Set r*ai = n'_b bit of r'_a
8b. Set r*bi = n'_a bit of r'_b
9a. Generate 128-bit random N_{ai}
9b. Generate 128-bit random N_{bi}
10a. Compute Commitment C_{ai}=f1(PK_{ax},PK_{bx},N_{ai},$r*_{ai}$)
10b. Compute Commitment C_{bi}=f1(PK_{bx},PK_{ax},N_{bi},$r*_{bi}$)
11.C_{ai}
12.C_{bi}
13.N_{ai}
13a. Check if C_{ai}=f1(PK_{ax},PK_{bx},N_{ai},r*bi(n'_b position)) If check fails, abort.
14.N_{bi}
14a. Check if C_{bi}=f1(PK_{bx},PK_{ax},N_{bi},r*ai(n'_a position)) If check fails, abort.

Figure 4.3: Enhanced Passkey Entry Model

Parameter	Description
$\mathbf{PK}_{ax}$	X-coordinate of public key of device A.
$\mathbf{PK}_{bx}$	X-coordinate of public key of device B.
$\mathbf{r}_a$ or $\mathbf{r}_b$	Random 6-digit passkey generated by User.
$\mathbf{r}'_a$ or $\mathbf{r}'_b$	256-bit hash value of the six-digit passkey.
$\mathbf{r}^*_{ai}$	The n'_bth bit of 256-bit hash sent by device A.
$\mathbf{r}^*_{bi}$	The n'_ath bit of 256-bit hash sent by device B.
$\mathbf{n}'_a$ or $\mathbf{n}'_b$	8-bit random Integer with range 0-255.
$\mathbf{N}_{ai}$	Random 128-bit Nonce.
$\mathbf{N}_{bi}$	Random 128-bit Nonce.
$\mathbf{C}_{ai}$	Commitment Value calculated by device A.
$\mathbf{C}_{bi}$	Commitment Value calculated by device B.

Table 4.2: Enhanced Passkey Entry Parameters

As seen in Figure 4.3, when the user enters the random passkey on both devices, we make use of the f2() hash function of the original passkey entry protocol to create a 256-bit hash value from the input passkey r. We use the shared DHKey calculated in the public key exchange phase as the key for the HMAC function and the input message, 20-bit passkey. The resultant hash is denoted as $\mathbf{r}'_a$ and $\mathbf{r}'_b$ respectively. The computed hashes $\mathbf{r}'_a$ and $\mathbf{r}'_b$ calculated in steps 3a and 3b have the same value since the passkey is same at both devices. After generating these values, the steps 4-12 are executed for 10 times. In steps 4a and 4b, an 8-bit random integer $\mathbf{n}'$ (where $1 <= n' <= 255$) is generated at both sides i.e., $\mathbf{n}'_a$ and $\mathbf{n}'_b$ for device A and device B respectively. This random integer defines the position at which the passkey bit is to be taken from $\mathbf{r}'_a$ and $\mathbf{r}'_b$. After generating the random integers, both devices perform an XOR operation with the 8-bit random integer $\mathbf{n}'$ and the $\boldsymbol{i^{th}}$ set of 8-bits of the $\mathbf{r}'$. Both values are exchanged in steps 5 and 6. Next, at device A, we compute the XOR as $(n'_b \oplus r'_b) \oplus r'_a$ and obtain the $\mathbf{n}'_b$ value. Then, we set the $\mathbf{n}'_b$th bit of 256-bit $\mathbf{r}'_a$ hash value to the $\mathbf{r}^*_{ai}$ bit. For instance, if the random number $\mathbf{n}'_b$ is 20, then we set the value of the 20^{th} bit of

$\mathbf{r'_a}$ hash as $\mathbf{r^*_{ai}}$. The same process is followed at the device B where we calculate $(n'_a \oplus r'_a) \oplus r'_b$ to obtain the $\mathbf{n'_a}$. Since the values of the $\mathbf{r'_a}$ and $\mathbf{r'_b}$ are equal, the XOR operation will derive the correct values. After setting the $\mathbf{r^*_{ai}}$ and $\mathbf{r^*_{bi}}$ values, both devices generate a 128-bit random nonce. In steps 10a and 10b, the device A computes a commitment value $C_{ai}= f1(PK_{ax}, PK_{bx}, N_{ai}, r^*_{ai})$ and device B computes a commitment value $C_{bi}= f1(PK_{bx}, PK_{ax}, N_{bi}, r^*_{bi})$. Both commitment values are exchanged between the two devices. Then, the device A sends the 128-bit random nonce $\mathbf{N_{ai}}$. After receiving the random nonce $\mathbf{N_{ai}}$, the device B sets the $\mathbf{r^*_{bi}}$ with the value at $\mathbf{n'_b}^{th}$ bit of $\mathbf{r'_b}$ hash. Next, the commitment value $\mathbf{C_{ai}}$ is checked by verifying $C_{ai}= f1(PK_{ax}, PK_{bx}, N_{ai}, r^*_{bi}(n'_b position))$. If the two $\mathbf{C_{ai}}$ values match, then the protocol is continued. The device B sends the random nonce $\mathbf{N_{bi}}$ and random integer $\mathbf{n'_b}$. The same process is followed at device A and the commitment values are verified. This process is repeated for 10 rounds where two bits of the 256-bit $\mathbf{r'}$ hash are sent randomly by two devices in each round.

In our proposed protocol, only 10 rounds are needed due to the fact that two different bits are exchanged in each round. If an attacker conducted a MITM attack, they have to predict the correct $\mathbf{r^*_{ai}}$ bit with device A and the correct $\mathbf{r^*_{bi}}$ bit with device B in the same round. Even if the attacker predicts a correct $\mathbf{r^*_{ai}}$ bit with device A, they cannot use the same value with device B because the random integer will be different thereby enabling device B to select another value in the random integer position from the $\mathbf{r'}$ hash. Therefore, in 10 rounds, we are effectively providing the security of 20 rounds.

The main idea of our proposed protocol is to use bits from the random position of the hash value of the passkey to do the commitment instead of using the bits of passkey. To keep the position information secure, we again use the hash value as the one-time pad encryption key to encrypt the information of positions. In our protocol, the passkey is never used to communicate between two parties. Since the random position is used, Algorithm 1 does not work for our protocol.

4.3 Security of Enhanced Passkey Entry Protocol under passive Eavesdropping and MITM attacks

Compared to the original passkey entry protocol, our enhanced protocol provides protection against passive and active attacks. Consider a scenario where the attacker X captures all the commitment values and the random nonce of the passkey entry protocol run using passive eavesdropping. Note that here, we are assuming that the attacker already has the knowledge of the frequency hopping pattern of the target devices. By doing this, the attacker effectively obtains the public keys of device A and device B. Then in the passkey entry protocol run, the commitment values and the random nonces are obtained. Now, the attacker X can try to deduce the value of $\mathbf{r^*}$ bits by comparing the commitment values but X cannot find out the original passkey r from the $\mathbf{r^*}$ bits. Therefore, even though X knows $\mathbf{r^*}$, they cannot deduce the original passkey using just that because the $\mathbf{r^*}$ bits are generated from the hash function which uses the original passkey r as the message and the DHkey as the key for the HMAC operation. At most, the attacker would have 10 bits of the $\mathbf{r^*_{ai}}$ and $\mathbf{r^*_{bi}}$ values which are exchanged in each round. However, these values are not enough to deduce the entire hash. In addition to this, even if the DHkey is known which is extremely difficult to deduce, the attacker would still not know the position of the $\mathbf{r^*}$ bits which are exchanged due to XOR operations. In addition to this, the $\mathbf{r'}$ value will be different for every SSP session which leads to an entirely different and unique hash value $\mathbf{r'}$ in another SSP session. This way we eliminate the possibility of an attacker that uses passive eavesdropping to deduce the original passkey. To summarise, it would be impossible for an attacker to deduce the full hash value of the passkey even if they passively eavesdropped the target device's SSP sessions continuously several times because of the following reasons.

- Even if the passkey is reused, the hash value obtained in step 3 of Figure 4.3 will be different for every SSP session. Each time, the target device connects to a different device, the DHKey value will change and as a result, the hash value will also change.
- The ECDH private-public key pair is generated randomly and is not static.
- Since the n' bits are hidden using the XOR operations, an adversary cannot guess the correct passkey due to a one-time pad technique.

We have also compared the security of our protocol to the SM protocol defined in Section 4.1. While the SM protocol can be attacked using our brute-force attack mentioned in algorithm 1, the enhanced passkey entry model completely prevents an adversary from establishing the attack. This is because of the XOR operations done on both sides to hide the correct value of the $\mathbf{n'_a}$ and $\mathbf{n'_b}$. From an attacker's perspective, the XOR'ed value which is sent during steps 5 and 6 is not useful to them. Unless the attacker already knows the random integer $\mathbf{n'}$ or the i^{th} set of 8-bits of the correct hash value, they can obtain neither the correct $\mathbf{n'}$ or the i^{th} set of $\mathbf{r'}$ value. In the XOR operation, the value of the $\mathbf{n'}$ and the set of 8-bit $\mathbf{r'}$ used is different for each round and is not repeated anywhere. Therefore, the XOR operation that is executed in steps 5 and 6 technically uses the properties of the one-time pad technique [36] making it very secure.

In case of a MITM attack, when the public key exchange phase is completed between the attacker and the devices A and B. Now, the attacker will share one DHkey with device A and one DHkey with device B. In our proposed protocol, after the legitimate user enters the 6-digit passkey, we generate a hash using the DHkey and the 6-digit passkey. Since the attacker does not know the passkey, they have 1 million possibilities of the passkey. Even if one digit of the passkey is guessed wrong, the hash value which is computed is changed drastically. In addition to this, the hash value of device A and device B will be different due to the different DHkey values. Due to this, when the XOR operation is done with the 8-bit random integer and the 8-bit set of $\mathbf{r'}$ and the result is exchanged between the devices, the derived value will be different leading to an incorrect $\mathbf{n'}$ value. Therefore, even if an attacker predicts a correct $\mathbf{r^*}$ bit in one round, they would not know the correct value of the position $\mathbf{n'}$ which is needed to execute the brute-force attack described in Algorithm 1.

Therefore, our protocol not only thwarts off passive eavesdropping but also Man-in-the-middle attacks even when a passkey is repeated.

Chapter 5

Performance Evaluation

5.1 Performance Cost of Original Passkey Entry Model

In the original passkey entry protocol for each round, there are hash operations of f1 at steps 4a, 4b, 7a, and 8a as shown in Figure 3.4. Therefore, the computation cost between the two devices is 80 hash functions. Coming to the commitment values, each value has a size of 128bits. And there are exactly 2 commitment values exchanged in each round for 20 rounds. Therefore, the communication cost for the commitment values can be calculated as

$$20(rounds) * 2(commitmentvalues) * 128(sizeofcommitment) = 5120bits$$

And in addition to that, the two devices exchange exactly two random nonce for each round in $1 <= i <= 20$ which leads to

$$20(rounds) * 2(RandomNonce) * 128(sizeofRandomnonce) = 5120bits$$

Altogether, we have a total of 10,240 bits being exchanged in the protocol run. Finally, the storage cost of the protocol would be around 192 or 256 bits depending on the length of the DHkey. Taking the size of the passkey into account, we also need an additional 20-bits. Then, the total storage cost would be 212 or 276 bits.

5.2 Performance Cost of Enhanced Passkey Entry Model

Our enhanced protocol performs in the same way as the original. The main difference of our protocol is that it significantly reduces the number of hash computations required due to the 10 rounds. We also need two extra variables to store the calculated 256-bit hash $\boldsymbol{r'}$ and the 8-bit random integer $\boldsymbol{n'}$. In addition to this, there are

2 additional hash computations present before the execution of steps 4-14 in Figure 4.3. These hash computations are done on both sides. We are not considering the XOR operations done on both sides as the operation cost is almost negligible. In our protocol, aside from two hash computations in step 3a and 3b, in each round, there are 4 hash computations which are used to compute the commitment values and verify them at both sides. So, for 10 rounds we have 4(Hash) x 10(rounds) = 40 Hash computations. Therefore, the computation cost of our protocol adds up to a total of 42 Hash computations. Regarding the communication cost, our protocol follows the same steps as that of the original protocol aside from the fact that two 8-bit random integers are exchanged in each round. The size of the commitment values and the random nonces remain the same. Therefore, the communication cost can be calculated as

$$10(rounds) * 2(randint) * 8(randintsize) + 10(rounds) * 2(commitmentvalues) * 128(commitmentsize) + 10(rounds) * 2(randomnonces) * 128(randomnoncesize) = 5280bits$$

The storage cost of our protocol is 468 or 532 bits (including the 20-bit passkey) depending on the length of DHkey. Although the storage cost is higher than the original protocol, it significantly increases the security of the protocol by providing two layers of security and protecting devices against passive attacks and active MITM attacks.

5.3 Performance Evaluations of Different Passkey Entry Models

Protocol	**Computation Cost**	**Communication Cost**	**Storage Cost**
Enhanced Passkey Entry	42H	5280 bits	468 or 532 bits
Original Protocol	80H	10,240 bits	212 or 276 bits
Da-zhi yun Improved passkey entry	82H	10,496 bits	212 or 276 bits
Barnickel's countermeasures	80H + 80(E+D)	10,496 bits	212 or 276 bits

Table 5.1: Performance Evaluations of different passkey entry models

Here, we calculate the storage cost by combining the size of the DHKey and also the 20-bit passkey. Barnickel's countermeasures required the device to have encryption and decryption capabilities. Furthermore, it requires 2 encryption and 2 decryption functions for every round which adds up to a total of 80 encryption and decryption functions in addition to the 80 hash computations. Although the performance cost Da-zhi's model is not high and is easy to implement, it is vulnerable to our brute-force attack. Whereas our enhanced passkey entry significantly reduces the computation cost as well as the communication cost in half. Due to the decrease in the computation cost, the device will not take as much power as the original protocol. In addition to this, the passkey entry protocol runs faster due to the decrease in the communication cost as we only need to send half of the bits compared to the original protocol.

5.4 Discussion

In Bluetooth standard, the passkey entry model uses the commitment scheme to check the passkey bit by bit. As we have already seen that this is good only for cases that the passkey is a one-time passkey. We proposed a method that still uses the commitment scheme. The difference is that instead of the passkey, our protocol checks the hash value of the passkey. Since the main part of the input for the hash function is changed each time, the hash value will also change each time. In this way, it's more difficult for an attacker to figure out the passkey.

The reason that we still use the commitment scheme is that we try to modify the standard protocol as little as possible so that the devices using the standard protocol will be easy to adapt to our modification.

On the other hand, it is not necessary to use the commitment schemes in the passkey scheme. It is possible to verify the whole hash value instead of commitment bit by bit. The passkey protocol will be simpler and more efficient if we just check the 256 bits hash value once and decide if the passkey is correct.

Chapter 6

Conclusion

In our research, we have introduced a new enhanced passkey entry protocol that successfully protects the Bluetooth devices against passive attacks and MITM attacks while reducing the computation and communication costs by half. It should be noted that we have designed our protocol without deviating too much from the original protocol. This is done so that our enhanced protocol can be simply integrated with the Bluetooth devices in the form of a patch. We have made a theoretical approach to the passkey reuse issue and protection against MITM in the passkey entry model. Since our research is theoretical, for our future work, we will be considering to employ our proposed protocol in different Bluetooth devices and test the protocol under different scenarios.

References

[1] SIG. Bluetooth core specification version 5.2. Specification of the Bluetooth System, 2019.

[2] Wikipedia contributors. "Bluetooth." Wikipedia, The Free Encyclopedia. Web. Accessed September 2019.

[3] Brian Ray. Bluetooth vs Bluetooth Low Energy: What's The Difference? `https://www.link-labs.com/blog/bluetooth-vs-bluetooth-low-energy`, November 2015. Accessed December 2019.

[4] James C. Lin, University of Chicago. "BLE Overview". `http://www.summitdata.com/blog/ble-overview/`, February 2013. Accessed October 2019.

[5] Wikipedia contributors. "Bluetooth Low Energy." Wikipedia, The Free Encyclopedia. Web. Accessed October 2019.

[6] Mikhail Galeev. Bluetooth 4.0: An introduction to Bluetooth Low Energy. `https://www.eetimes.com/bluetooth-4-0-an-introduction-to-bluetooth-low-energy-part-i/`, July 2011. Accessed October 2019.

[7] Kevin Townsend. `https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt`, March 2014. Accessed December 2019.

[8] Bluetooth Classic Vs Bluetooth Low Energy (BLE): Which one is right for you?. `https://www.plugintoiot.com/bluetoothclassic-vs-bluetoothlowenergy/`, June 2016. Accessed November 2019.

[9] Dawid Kliszowski. Bluetooth Classic vs. Bluetooth Low Energy on Android - Hints & Implementation Steps. `https://www.thedroidsonroids.com/blog/bluetooth-classic-vs-bluetooth-low-energy-on-android-hints-implementation-steps`, June 2017. Accessed October 2019.

[10] Jeremy Martin, Douglas Alpuche, Kristina Bodeman, Lamont Brown, Ellis Fenske, Lucas Foppe, Travis Mayberry, Erik Rye, Brandon Sipes, and Sam Teplov. Handoff All Your Privacy – A Review of Apple's Bluetooth Low Energy Continuity Protocol. `https://arxiv.org/pdf/1904.10600.pdf`, June 2019. Accessed October 2019.

[11] Vinh Pham and Janne Hagen. Bluetooth Security and Threats. https://ffi-publikasjoner.archive.knowledgearc.net/bitstream/handle/20.500.12242/1115/15-01091.pdf?sequence=1&isAllowed=y, November 2015. Accessed November 2019.

[12] Zegeye and Wondimu K. Exploiting Bluetooth Low Energy Pairing Vulnerability in Telemedicine. https://repository.arizona.edu/bitstream/handle/10150/596383/ITC_2015_15-02-04.pdf?sequence=1&isAllowed=y, October 2015. Accessed October 2019.

[13] Michael. BLE secure connections security modes. https://security.stackexchange.com/questions/141336/difference-between-bluetooth-le-secure-connections-security-mode-1-and-level-3-a, November 2016. Accessed September 2019.

[14] Electronic Notes. Bluetooth Security. https://www.electronics-notes.com/articles/connectivity/bluetooth/security.php. Accessed December 2019.

[15] Alexis Duque. Deep Dive into Bluetooth LE Security. https://medium.com/rtone-iot-security/deep-dive-into-bluetooth-le-security-d2301d640bfc, Mar 2018. Accessed December 2019.

[16] Angela M. Lonzetta, Peter Cope, Joseph Campbell, Bassam J. Mohd, and Thaier Hayajneh. Security Vulnerabilities in Bluetooth Technology as Used in IoT. https://res.mdpi.com/jsan/jsan-07-00028/article_deploy/jsan-07-00028.pdf?filename=&attachment=1, July 2018. Accessed November 2019.

[17] Nateq Be-Nazir Ibn Minar and Mohammed Tarique. Bluetooth Security Threats and Solutions: A survey. https://www.researchgate.net/publication/267200901_Bluetooth_Security_Threats_And_Solutions_A_Survey, February 2012. Accessed January 2020.

[18] Sanyam J. BlueBump Attack. https://iq.opengenus.org/bluebump-attack/. Accessed December 2019.

[19] Shrinath. Bluetooth Hacking. https://exploitbyte.com/bluetooth-hacking/, November 2019. Accessed December 2019.

[20] Adam Laurie. Serious flaws in bluetooth security lead to disclosure of personal data. https://seclists.org/fulldisclosure/2003/Nov/409, November 2003. Accessed October 2019.

[21] Wikipedia contributors. "Bluesnarfing." Wikipedia, The Free Encyclopedia. Web. Accessed October 2019.

[22] Daniele Antonioli, Nils Ole Tippenhauer, and Kasper B. Rasmussen. The KNOB is Broken: Exploiting Low Entropy in the Encryption Key Negotiation Of Bluetooth BR/EDR. `https://www.usenix.org/system/files/sec19-antonioli.pdf`, August 2019. Accessed September 2019.

[23] Da-Zhi Sun, Li sun. On Secure Simple Pairing in Bluetooth standard v5.0-Part I: Authenticated Link Key Security and Its Home Automation and Entertainment Applications. March 2019. Accessed December 2019.

[24] Eli Biham, Lior Neumann. Breaking the Bluetooth Pairing - Fixed Coordinate Invalid Curve Attack. July 2018. Accessed January 2020.

[25] Samta Gajbhiye, Sanjeev Karmakar. Bluetooth Secure Simple Pairing with an enhanced security level. Journal of information security and applications 44 (2019). pp.170-183.

[26] Giwon Kwon, Jeehyeong Kim. Bluetooth Low Energy Security Vulnerability and Improvement Method. IEEE ICCE Asia, 2016. Accessed December 2019.

[27] Da-Zhi Sun, Yi Mu. Man-in-the-middle attacks on Secure Simple Pairing in Bluetooth standard v5.0 and its countermeasure. Personal and Ubiquitous Computing, February 2018. pp.55-67.

[28] Barnickel J., Wang J., Meyer U. Implementing an attack on Bluetooth 2.1+ secure simple pairing in passkey entry mode. IEEE computer society (2012), pp.17-24.

[29] Ellisys. Secure Simple Pairing Explained. `https://www.ellisys.com/technology/een_bt07.pdf`, 2011. Accessed January 2020.

[30] Wikipedia contributors. "Elliptic-curve Diffie–Hellman." Wikipedia, The Free Encyclopedia. Web. Accessed December 2019.

[31] Andrea Corbellini. Elliptic Curve Cryptography: ECDH and ECDSA. `https://andrea.corbellini.name/2015/05/30/elliptic-curve-cryptography-ecdh-and-ecdsa/`, May 2015. Accessed December 2019.

[32] Neil Madden. Ephemeral elliptic curve Diffie-Hellman key agreement in Java. `https://neilmadden.blog/2016/05/20/ephemeral-elliptic-curve-diffie-hellman-key-agreement-in-java/`, April 2017. Accessed December 2019.

[33] Mark Loveless. Understanding Bluetooth Security. `https://duo.com/decipher/understanding-bluetooth-security`, Jan 2018. Accessed December 2019.

[34] FIPS PUB 180-4. Secure Hash Standard (SHS). `https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf`, August 2015. Accessed November 2019.

[35] Wikipedia contributors. ”HMAC.” Wikipedia, The Free Encyclopedia. Web. Accessed December 2019.

[36] Wikipedia contributors. ”One-time pad.” Wikipedia, The Free Encyclopedia. Web. Accessed December 2019.

Appendix A

Brute-force attack Code Link

To code the brute-force attack algorithm, we have used the python3.8 programming language. The code is available at `https://github.com/saiswaroop7/Bluetooth-SSP-BruteForce`