

Chapter 2

Overview of Bluetooth Security

The basic Bluetooth security configuration is done by the user who decides how a Bluetooth device will implement its connectability and discoverability options. The different combinations of connectability and discoverability capabilities can be divided into three categories, or *security levels* [1, 2].

1. *Silent*: The device will never accept any connections. It simply monitors Bluetooth traffic.
2. *Private*: The device cannot be discovered, i.e., it is a so-called *non-discoverable device*. Connections will be accepted only if the *Bluetooth Device Address (BD_ADDR)* is known to the prospective master. A 48-bit BD_ADDR is normally unique and refers globally to only one individual Bluetooth device.
3. *Public*: The device can be both discovered and connected to. It is therefore called a *discoverable device*.

The 48-bit BD_ADDR is divided into three parts: the 16-bit Nonsignificant Address Part (NAP), the 8-bit Upper Address Part (UAP), and the 24-bit Lower Address Part (LAP). The first three bytes of BD_ADDR (NAP and UAP) refer to the manufacturer of the Bluetooth chip and represent *company_id*. The last three bytes of BD_ADDR (LAP), called *company_assigned*, are used more or less randomly in different Bluetooth device models. *Company_id* values are public information and are listed in the Institute of Electrical and Electronics Engineers' (IEEE's) Organizationally Unique Identifier (OUI) database [2, 5].

There are also four different *security modes* that a device can implement. In Bluetooth technology, a device can be in only one of the following security modes at a time [1, 2]:

1. *Nonsecure*: The Bluetooth device does not initiate any security measures.
2. *Service-level enforced security mode*: Two Bluetooth devices can establish a non-secure ACL link. Security procedures, namely authentication, authorization, and optional encryption, are initiated when a Logical Link Control and Adaptation Protocol (L2CAP) Connection-Oriented (CO) or an L2CAP Connection-Less (CL) channel request is made.

3. *Link-level enforced security mode*: Security procedures are initiated when an ACL link is established.
4. *Service-level enforced security mode*: This mode is similar to mode 2, except that only Bluetooth devices using SSP can use it, i.e., only Bluetooth 2.1+EDR or later devices can use this security mode.

Authentication is used for proving the identity of one piconet device to another. The results of authentication are used for determining the client's *authorization level*, which can be implemented in many different ways: for example, access can be granted to all services, only to a subset of services, or to some services while other services require additional authentication. *Encryption* is used for encoding the information being exchanged between Bluetooth devices in such a way that eavesdroppers cannot read its contents [2].

Bluetooth uses *Secure And Fast Encryption Routine + (SAFER+)* [6] with a 128-bit key as an algorithm for authentication and key generation in Bluetooth versions up to 3.0+HS (High Speed), while Bluetooth 4.0 (i.e. Bluetooth Low Energy) replaces SAFER+ with the more secure 128-bit *Advanced Encryption Standard (AES)* [1, 2, 7].

SAFER+ [6] was developed by Massey et al. in 1998. It was submitted as a candidate for the AES contest, but the cipher was not selected as a finalist. SAFER+ is a block cipher with the following main features. It has a block size of 128 bits and three different key lengths (128, 192, and 256 bits). SAFER+ consists of nine phases (eight identical rounds and the output transformation) and a Key Scheduling Algorithm (KSA) in the following way. KSA produces 17 different 128-bit subkeys. Each round uses two subkeys and a 128-bit input word from the previous round to calculate a 128-bit word that is a new input word for the next round. The last subkey is used in the output transformation, which is a simple bitwise XOR of the last round's output with the last subkey. Although some optimizations for faster breaking of SAFER+ exist (for example, in [8, 9]), it is still considered quite secure [1, 2, 6, 8, 9].

AES [7] was published by the National Institute of Standards and Technology (NIST) in 2001 after the evaluation process of the AES contest. Rijndael was the winner of the contest and NIST selected it as the algorithm for AES. AES is a symmetric block cipher that is intended to replace the Data Encryption Standard (DES) as the approved standard for a wide range of applications, but this process will take many years. NIST anticipates that the Triple Data Encryption Standard (3DES) will remain an approved algorithm for the foreseeable future, at least for U.S. government use. AES encryption consists of 10–14 rounds in which data blocks are processed step-by-step in the following way (except the final round; it is noteworthy that AES decryption is symmetric to AES encryption) [1, 2, 7, 10, 11]:

1. *Byte substitution*: Byte substitution uses an S-box to perform a byte-by-byte substitution of the block.
2. *Row shifting*: Row shifting is a simple permutation.

3. *Column mixing*: Column mixing is a substitution which makes use of arithmetic over $GF(2^8)$. Galois Field $GF(2^8)$ is a finite field of 256 elements, which can be denoted by strings of eight bits or by hexadecimal notation.
4. *Round key adding*: Round key adding is a simple bitwise XOR of the current block with a portion of the expanded key.

The final round of AES encryption (and AES decryption) is slightly different [1, 2, 7, 10, 11]:

1. *Byte substitution*.
2. *Row shifting*.
3. *Round key adding*.

AES is considered secure, it is very fast and compact (about 1 kB of code), its block size is a multiple of 32 (typically 128 bits), its key length is also multiples of 32 (typically 128, 192, or 256 bits), and it has a very neat algebraic description [2, 7, 10, 11].

Because Bluetooth is a wireless communication system, there is always a possibility that the transmission could be deliberately jammed or intercepted, or that false or modified information could be passed to the piconet devices [2].

Bluetooth security is based on building a chain of events, none of which should provide meaningful information to an eavesdropper. All events must occur in a specific sequence for security to be set up successfully [1, 2].

In order for two Bluetooth devices to start communicating, a procedure called *pairing* must be performed. As a result of pairing, two devices form a trusted pair and establish a link key which is used later on for creating a data encryption key for each session [2].

In Bluetooth versions up to 2.0+EDR, pairing is based exclusively on the fact that both devices share the same *Personal Identification Number (PIN)*, or *passkey*, that is used for generating several 128-bit keys as Fig. 2.1 illustrates. When the user enters the same passkey in both devices, the devices generate the same shared secret which is used for authentication and encryption of traffic exchanged by them [1, 2].

An *initialization key* (K_{init}) is generated when Bluetooth devices meet for the first time and it is used for securing the generation of other more secure 128-bit keys, which are generated during the next phases of the security chain of events. K_{init} is derived from a 128-bit pseudorandom number IN_RAND , an L -byte ($1 \leq L \leq 16$) PIN code, and a BD_ADDR . It is worth noting that IN_RAND is sent via air in unencrypted form [1, 2].

The output of a certain key generation function can be expressed in terms of the function itself and its inputs. K_{init} is produced in both devices using the formula $K_{init} = E_{22}(PIN', L', IN_RAND)$. The PIN code and its length L are modified into two different quantities called PIN' and L' before they are sent to the E_{22} function. If the PIN is smaller than 16 bytes, it is augmented by appending bytes from the device's BD_ADDR until either PIN' reaches a total length of 16 bytes or the entire BD_ADDR is appended, whichever comes first. If one device has a fixed PIN code,

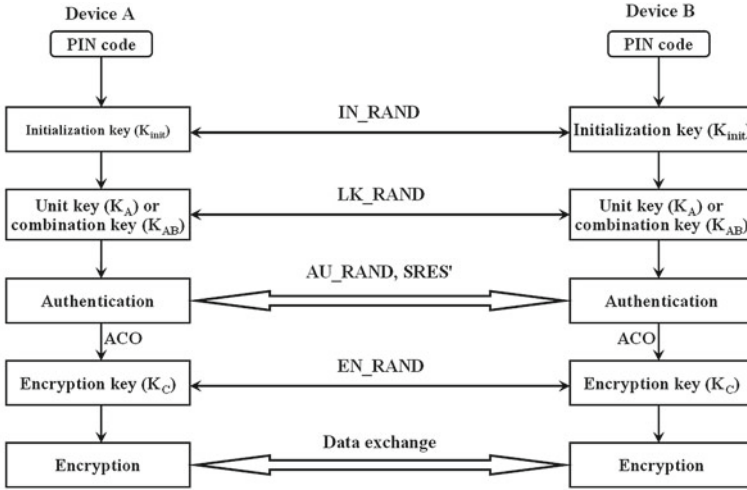


Fig. 2.1 Summary of Bluetooth security operations [1, 2]

the BD_ADDR of the other device is used. If both devices can support a variable PIN code, the BD_ADDR of the device that received IN_RANDOM is used [1, 2].

K_{init} is used to encrypt a 128-bit pseudorandom number (RAND or LK_RANDOM), i.e., $RAND \oplus K_{init}$ or $LK_RAND \oplus K_{init}$, exchanged in the next phase of the security chain of events when a link key (a unit key or a combination key) is generated [1, 2].

A *unit key* (K_A) is produced from the information of only one device (device A) using the formula $K_A = E_{21}(BD_ADDR_A, RAND_A)$. Device A encrypts K_A with K_{init} , i.e., $K_A \oplus K_{init}$, and sends it to device B. Device B decrypts K_A with K_{init} , i.e., $(K_A \oplus K_{init}) \oplus K_{init} = K_A$, and now both devices have the same K_A as a link key [1, 2].

A Bluetooth device that uses a unit key has only one key to use for all of its connections. This means that the same key is shared with all other trusted Bluetooth devices. In addition, any trusted Bluetooth device using the same unit key can eavesdrop on any traffic between two other Bluetooth devices that share the same unit key. Moreover, any trusted Bluetooth device using the same unit key can impersonate the target device just by duplicating its BD_ADDR. Thus, only devices that have limited resources, i.e., no memory to store several keys, should use K_A , because it provides only a low level of security. Therefore, Bluetooth specifications do not recommend using K_A anymore. More information about unit key weaknesses can be found in [1, 2, 12].

A *combination key* (K_{AB}) is dependent on two devices and therefore it is derived from the information of both devices. K_{AB} is produced in both devices using the formula $K_{AB} = E_{21}(BD_ADDR_A, LK_RAND_A) \oplus E_{21}(BD_ADDR_B, LK_RAND_B)$. It is worth noting that generating K_{AB} is nothing more than a simple bitwise XOR between two unit keys, i.e. $K_{AB} = K_A \oplus K_B$. Each device can produce its own unit

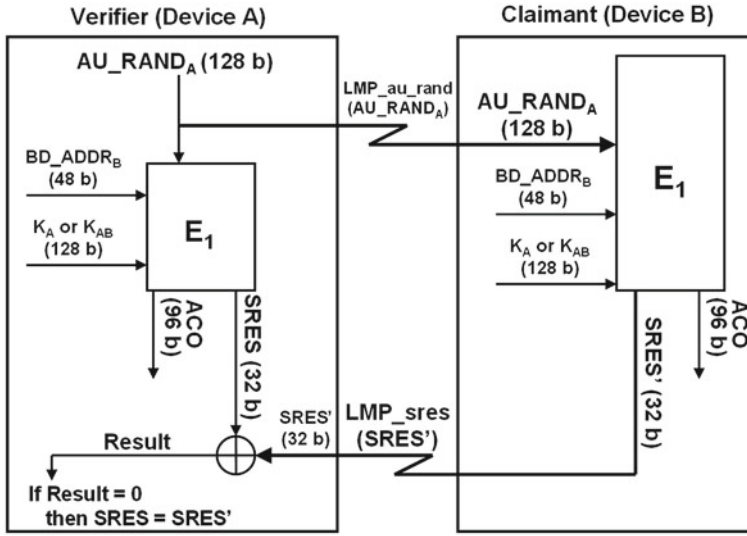


Fig. 2.2 Bluetooth challenge-response authentication [1, 2]

key and each device also has the BD_ADDR of the other device. Therefore, two devices have to exchange only their respective pseudorandom numbers in order to produce each other's unit key [1, 2].

Device A encrypts LK_RAND_A with the current key K , i.e., $LK_RAND_A \oplus K$ where K can be the K_{init} , the K_A , or the K_{AB} that was created earlier, and sends it to device B. K is K_{init} if the devices create a link key for the first time together. K is K_A if the link key is being upgraded to a combination key, and it is K_{AB} if the link key is being changed. Device B decrypts LK_RAND_A with K , i.e., $(LK_RAND_A \oplus K) \oplus K = LK_RAND_A$, and can now produce K_A . Correspondingly, device B encrypts LK_RAND_B with K , i.e., $LK_RAND_B \oplus K$, and sends it to device A. Device A decrypts LK_RAND_B with K , i.e., $(LK_RAND_B \oplus K) \oplus K = LK_RAND_B$, and produces K_B . Finally, both devices can produce K_{AB} by XORing K_A with K_B , i.e., $K_{AB} = K_A \oplus K_B$ [1, 2].

The next phase of the security chain of events is the *challenge-response authentication* in which a claimant's knowledge of a secret link key is checked, as Fig. 2.2 illustrates. During each authentication, a new 128-bit pseudorandom number AU_RAND is exchanged via air in unencrypted form. Other inputs to the authentication function E_1 are the BD_ADDR of the claimant and the current link key (K_A or K_{AB}) [1, 2].

A 32-bit result (*SRES*, Signed Response) and a 96-bit result (*ACO*, Authenticated Ciphering Offset) are produced in both devices by the $E_1(AU_RAND_A, BD_ADDR_B, \text{Link key})$ function, where the Link key is K_A or K_{AB} . The claimant sends $SRES'$, i.e., the $SRES$ value produced by the claimant, via air in unencrypted form to the verifier. The verifier compares the generated $SRES$ value with the received

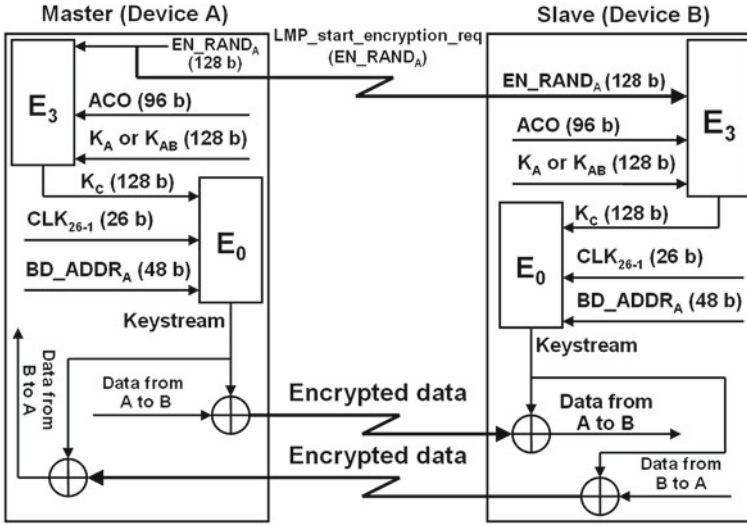


Fig. 2.3 Bluetooth data encryption [1, 2]

SRES value, and if these values match, the authentication is completed successfully. The ACO is used in the next phase of the security chain of events when an encryption key is generated [1, 2].

It is worth noting that SRES and SRES' are 32-bit values, not 128-bit values. The 32-bit SRES provides reasonable protection against an attacker who is trying to guess the value, while it also reduces the chance that the PIN code will be compromised by an attacker who has somehow determined the correct SRES value [2].

Figure 2.3 illustrates Bluetooth data encryption between two Bluetooth devices. The ACO, the current link key (K_A or K_{AB}) and a 128-bit pseudorandom number EN_RAND are the inputs to the encryption key generation function E_3 that is used for generating an *encryption key* (K_C). The master (device A) generates EN_RAND and sends it to the slave (device B) via air in unencrypted form. K_C is produced in both devices using the formula $K_C = E_3(EN_RAND_A, ACO, \text{Link key})$, where the Link key is K_A or K_{AB} [1, 2].

The keystream generator function E_0 (see Fig. 2.3) makes symmetric encryption possible by generating the same cipher bit stream, or *keystream* (also referred to as a *running key*), in both devices. The inputs to the E_0 function are K_C , the BD_ADDR of the master (BD_ADDR_A), and the 26 bits of the master's real-time clock (CLK_{26-1}). The keystream is generated by the $E_0(K_C, CLK_{26-1}, BD_ADDR_A)$ function that is reinitialized for every new sent or received Baseband packet, i.e., CLK_{26-1} is updated for every new Baseband packet. This means that inputs to the E_0 function are never identical longer than the lifetime of one Baseband packet and therefore a new keystream is generated for every new Baseband packet [1, 2].

The sender encrypts the plaintext by XORing it with the keystream, i.e., $\text{Plaintext} \oplus \text{Keystream} = \text{Ciphertext}$, and sends the produced ciphertext to the receiver. The receiver decrypts the ciphertext by XORing it with the same keystream, i.e., $\text{Ciphertext} \oplus \text{Keystream} = (\text{Plaintext} \oplus \text{Keystream}) \oplus \text{Keystream} = \text{Plaintext}$. It is worth noting that only the payload of the Bluetooth Baseband packet is encrypted (not an access code or a header), and therefore an attacker cannot use the regularly repeating information (that is easy for the attacker to guess) of the access code and the header in order to facilitate a cryptanalysis of the cipher [2].

As already discussed in this chapter, the PIN is the only source of entropy for the shared secret in Bluetooth versions up to 2.0+EDR. As the PINs often contain only four decimal digits, the strength of the resulting keys is not enough for protection against passive eavesdropping on communication. Even with longer 16-character alphanumeric PINs, full protection against active eavesdropping cannot be achieved: it has been shown that MITM attacks on Bluetooth communications (versions up to 2.0+EDR) can be performed [2, 13–15].

Let us assume that Alice and Bob are communicating with each other and they want to secure their communication by using some public-key encryption method. In a *MITM attack*, Mallory (an attacker) intrudes between Alice and Bob. Mallory can eavesdrop on messages, modify messages, delete messages, and generate new messages between Alice and Bob in such a way that his presence is unrevealed, i.e., Alice and Bob do not know that the link between them is compromised by Mallory. Mallory is also able to imitate Bob when talking to Alice and vice versa. This simple example of a MITM attack works in the following way [2, 10, 16]:

1. Alice sends her public key to Bob, but Mallory is able to intercept it. Mallory sends Bob his own public key for which he has the matching private key. Now Bob wrongly thinks that he has Alice's public key.
2. Bob sends his public key to Alice, but Mallory is able to intercept it. Mallory sends Alice his own public key for which he has the matching private key. Now Alice wrongly thinks that she has Bob's public key.
3. Alice sends Bob a message encrypted with Mallory's public key, but Mallory is able to intercept it. Mallory decrypts the message with his private key, keeps a copy of the message, re-encrypts the message with Bob's public key, and sends the message to Bob. Now Bob wrongly thinks that the message came directly from Alice.
4. Bob sends Alice a message encrypted with Mallory's public key, but Mallory is able to intercept it. Mallory decrypts the message with his private key, keeps a copy of the message, re-encrypts the message with Alice's public key, and sends the message to Alice. Now Alice wrongly thinks that the message came directly from Bob.

Even if the public keys of Alice and Bob are stored on a database, a MITM attack will work. Mallory can intercept Alice's (or Bob's) database inquiry and substitute his own public key for Bob's (or Alice's) public key. He can also somehow break into the database and substitute his key for both Alice's public key and Bob's public key. A MITM attack works, because Alice and Bob have no way to verify that they are

truly using each other's correct public keys. If Mallory does not cause any noticeable delays to the communication, Alice and Bob have no idea that Mallory has intruded between them [2, 10, 11, 16–18].

Without any verification of the public keys, MITM attacks are generally possible (in principle) against any message sent using public-key technology. One solution to this problem is to use *public key certificates* (also referred to as *digital identity certificates*) [17], which use digital signatures to bind together public keys with the information of their respective users, i.e., information such as the name of the user, the address of the user, and so on. Each user is associated with a trusted authority, a *Certification Authority (CA)*, and each certificate is created by such a CA. A certificate establishes a verifiable connection between the user and his public keys. The users know their CA's public key and therefore they can verify the signatures of their CA. The certificate is stored in a directory. Only the CA is allowed to write in this directory, but all users of the CA can read the information in the directory [2, 10, 11, 16–18].

Defences against MITM attacks use authentication techniques which are based on *public key certificates*, *two-way authentication* (also referred to as *mutual authentication*), *secret keys*, *passwords*, and other methods (such as *voice recognition* and *other biometrics*) [2, 10, 11, 16–18].

Bluetooth versions 2.1+EDR, 3.0+HS, and 4.0 add a new specification for the pairing procedure, namely SSP [1]. Its main goal is to improve the security of pairing by providing protection against passive eavesdropping and MITM attacks [1, 2].

Instead of using (often short) passkeys as the only source of entropy for building the link keys, SSP employs Elliptic Curve Diffie-Hellman (ECDH) public-key cryptography. To construct the link key, devices use public-private key pairs, a number of nonces, and Bluetooth addresses of the devices. Passive eavesdropping is effectively thwarted by SSP, as running an exhaustive search on a private key with approximately 95 bits of entropy is currently considered to be infeasible in reasonable time [1, 2].

In order to provide protection against MITM attacks, SSP either uses an OOB channel (e.g., Near Field Communication, NFC), or asks for the user's help: for example, when both devices have displays and keyboards, the user is asked to compare two six-digit numbers. Such a comparison can also be thought of as an OOB channel which is not controlled by the MITM. If the values used in the pairing process have been tampered with by the MITM, the six-digit integrity checksums will differ with probability 0.999999 [1, 2].

SSP uses four *association models*. In addition to the two association models mentioned previously, *OOB* and *Numeric Comparison*, models named *Passkey Entry* and *Just Works* are defined [1, 2].

The Passkey Entry association model is used in cases when one device has input capability, but no screen that can display six digits. A six-digit checksum is shown to the user on the device that has output capability, and the user is asked to enter it on the device with input capability. The Passkey Entry association model is also used if both devices have input but not output capabilities. In this case the user chooses a 6-digit checksum and enters it in both devices. Finally, if at least one of the devices has neither input nor output capability, and an OOB cannot be used, the

Table 2.1 Device capabilities and SSP association models [1, 2]

Device 1	Device 2	Association model
DisplayYesNo	DisplayYesNo	Numeric comparison ^a
	DisplayOnly	Numeric comparison
	KeyboardOnly	Passkey Entry ^a
	NoInputNoOutput	Just works
DisplayOnly	DisplayOnly	Numeric comparison
	KeyboardOnly	Passkey entry ^a
	NoInputNoOutput	Just Works
KeyboardOnly	KeyboardOnly	Passkey entry ^a
	NoInputNoOutput	Just works
NoInputNoOutput	NoInputNoOutput	Just works

^aThe resulting link key is considered *authenticated*

Just Works association model is used. In this model the user is not asked to perform any operations on numbers: instead, the device may simply ask the user to accept the connection [1, 2].

The choice of the association model depending on the device capabilities is shown in Table 2.1. *DisplayYesNo* indicates that the device has a display and at least two buttons that are mapped to “yes” and “no”: using the buttons the user can either accept the connection or decline it. Other notation in the table is self-explanatory [1, 2].

SSP is comprised of six phases: [1, 2]

1. *Capabilities exchange*: The devices that have never met before or want to perform re-pairing for some reason, first exchange their Input/Output (IO) capabilities (see Table 2.1) to determine the proper association model to be used.
2. *Public key exchange*: The devices generate their public-private key pairs and send the public keys to each other. They also compute the Diffie-Hellman key.
3. *Authentication stage 1*: The protocol that is run at this stage depends on the association model. One of the goals of this stage is to ensure that there is no MITM in the communication between the devices. This is achieved by using a series of nonces, commitments to the nonces, and a final check of integrity checksums performed either through the OOB channel or with the help of the user.
4. *Authentication stage 2*: The devices complete the exchange of values (public keys and nonces) and verify their integrity.
5. *Link key calculation*: The parties compute the link key using their Bluetooth addresses, the previously exchanged values, and the Diffie-Hellman key constructed in phase 2.
6. *LMP authentication and encryption*: Encryption keys are generated in this phase, which is the same as the final steps of pairing in Bluetooth versions up to 2.0+EDR.

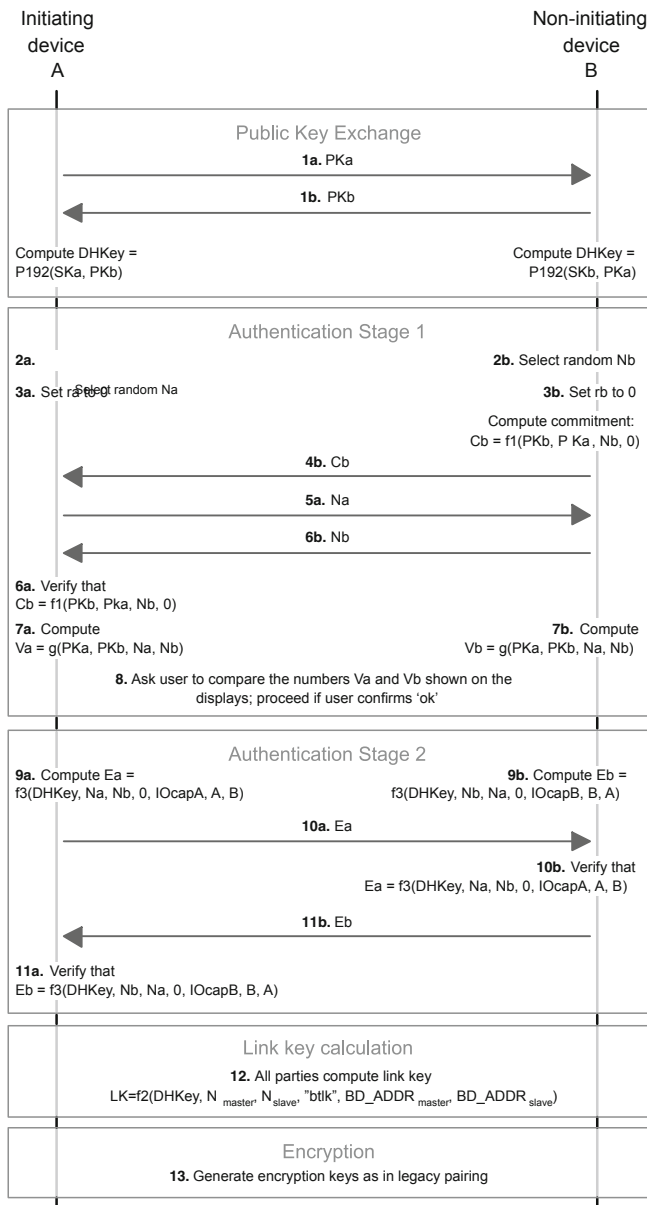


Fig. 2.4 SSP with the Numeric Comparison association model [1, 2]

The contents of messages sent during the SSP phase are outlined in Fig. 2.4 and the notations used are explained in Table 2.2.

Table 2.2 SSP protocol notation [1, 2]

Term	Definition
PK _x	Public key of device X
SK _x	Private key of device X
DHKey	Diffie-Hellman key generated after key exchange
N _x	Nonce generated by device X
rx	Random number generated by device X; equals 0 in the Numeric Comparison association model
C _x	Commitment value from device X
f1	One-way function used to compute commitment values
f2	One-way function used to compute the link key
f3	One-way function used to compute check values
g	One-way function used to compute numeric check values
IOcapX	Input/Output capabilities of device X
BD_ADDR	48-bit Bluetooth device address

Even though SSP improves the security of Bluetooth pairing, it has been shown that MITM attacks against Bluetooth 2.1+EDR, 3.0+HS, and 4.0 devices are possible by forcing victim devices to use the Just Works association model [2, 19–23]. Moreover, at least one of the proposed MITM attacks against Bluetooth SSP has already been implemented and mounted in practice [24]. Thus, the security of SSP should be further improved.

Bluetooth Security Attacks

Comparative Analysis, Attacks, and Countermeasures

Haataja, K.; Hyppönen, K.; Pasanen, S.; Toivanen, P.

2013, VII, 93 p. 31 illus., Softcover

ISBN: 978-3-642-40645-4