Testing for weak key management in Bluetooth Low Energy implementations

Tristan Claverie and José Lopes-Esteves
<prenom>.<nom>@ssi.gouv.fr

ANSSI

Abstract. Bluetooth Low Energy is a widely deployed protocol in the world of connected devices. As such, the question of the security level of communications is important. This paper analyses and summarises the previous work from a communication security point of view. It discusses the concept of key exchange and key generation in Bluetooth Low Energy as well as their inner working. Some new shortcomings of the standardised key exchange and key generation are discussed in this paper. Test procedures are developed, enabling one to verify that a device has none of the mentioned problems.

1 Introduction

Among the several wireless communication protocols deployed in electronic "smart" devices, Bluetooth Low Energy (BLE) has a prominent role. It is currently integrated by default in billions of devices [30], be it smartphones, Smart TVs, healthcare devices, locks, etc. Because it is deployed in a lots of different devices, contexts and scenarios, the concept of "security" in BLE can cover many cases. Common security research efforts tend to go in the following directions:

- **Device security**: in this context, researchers try to get access to the information or capabilities of the device. Smart locks or glucometers have been subject to those kind of studies;
- **Privacy**: in this context, researchers analyse the privacy implications of BLE devices and study the privacy-preserving modes implemented;
- **Tool manufacture**: in this part, developers and researchers craft tools to interact with BLE devices and test their security. Studies about sniffers and framework are represented;
- **Communication security**: in this context, researchers set to analyse the security properties of the BLE communication protocol.

The first three approaches will be briefly discussed in this paper, while a focus will be made on BLE communication security. Therefore related work of this nature will be examined more thoroughly.

Like many standards, the BLE specification is a rather complex document which does contribute to give neither a clear comprehension of the security mechanisms nor a precise view of the best way to implement those. Furthermore, backwards compatibility raises questions about the way security is implemented and managed in devices which are compatible with several versions of the standard.

In this study, the auditability of closed source BLE stacks against potential misinterpretations or bad implementations of specific security requirements of the standard is discussed.

First, an effort is made to decribe with clarity and pedagogy the security mechanisms supported by the standard. Then, an example description of two weaknesses that become exploitable when some specific security requirements from the standard are not correctly implemented is provided. The first one relies on a lack of entropy in one of the key generation methods described in the standard. An attacker with the ability to repeatedly bond with a device which uses this method is able to enumerate all the keys that the device will generate. This attack affects all Pairing procedures and requires the implementation of the Key Hierarchy key generation. The second one can be viewed as an extension of CVE-2018-5383 in the case key renewal is improperly implemented. It impacts the key exchange in BLE. Both attacks lead directly or indirectly to compromising the keys involved in the confidentiality, the integrity and the authenticity of communications. The effectiveness of both attacks is conditioned by the way delays are introduced between successive pairing attempts, as mandated by the standard.

It is to be noted that devices implementing correctly the security requirements of the latest version of the standard will not be impacted by those attacks. But it becomes pretty obvious that the possibility to determine if a target implementation is vulnerable to such attacks is of fundamental interest. To adequatly address this issue, it is suitable to design and release efficient test procedures which ideally would be benign, i.e. do not require or provide means to exploit the vulnerabilities to identify vulnerable devices.

Thus the main outcome of this study is the design, the evaluation and the implementation of test vectors allowing to test closed source implementations against the aforementioned vulnerabilities.

In section 2, necessary basics about Bluetooth protocols will be provided. The security mechanisms and keys involved will be detailed in an understandable way. Section 3 will present the state of the art with a focus on BLE communication security. In section 4, the inner workings of

BLE key exchange and key generation mechanisms will be discussed. Section 5 will explain the problems brought by the Key Hierarchy generation method and devise a testing method for it. The section 6 will focus on detailing the implementation flaws highlighted by Biham et al. [22]. It will discuss the analysis of their work made by Cremers et al. [27] and discuss the applicability of the key retrieval scenario. Some test vectors will be discussed in this section. Section 7 will present the challenges encountered when performing successive pairings on various type of devices. It will discuss the effectiveness of designed test procedures when testing an open implementation for the identified vulnerabilities. Finally, section 8 will conclude this paper.

2 Bluetooth technical background

This section provides an introduction to several protocols which were standardized under the "Bluetooth" denomination. Their main differences and the security mechanisms they provide are discussed in detail.

2.1 Bluetooth Classic and Bluetooth Low Energy basics

Bluetooth-related protocols are developed and standardised by the Bluetooth Special Interest Group (SIG) [7]. Bluetooth Classic (BT) and Bluetooth Low Energy are communication protocols, that is they allow to exchange data between two or more entities. When a communication link is established between two devices using either protocol, the communication follows a master-slave fashion on the lower layers.

The first version of the specification appeared in 1999, in which Bluetooth Classic was described. There have been significant changes in the security of BT in version 2.1, published in 2007, with the addition of security procedures under the name **Secure Simple Pairing** (SSP). Bluetooth Low Energy was officially added to the specification in version 4.0 in 2009 (though it lived a few years before under the denomination "Bluetooth Smart"). There have been some evolutions to the security of BLE in version 4.2 with the addition of security procedures called **LE Secure Connections** [1] (LESC). Retro-actively, the previous security procedures have been named **Legacy Pairing** (versions 4.0 and 4.1).

New security procedures defined in LESC are in fact those that had been defined in SSP, being rebranded, though cryptographic primitives

1. The term **Secure Connections** (without 'LE') refers to a security mode of Bluetooth Classic. The term **LE Secure Pairing** may be found in relevant literature, it is a synonym of **LE Secure Connections**.

used in LESC security procedures are not the exact same as the ones used in SSP security procedures. This means that some results apply equally to SSP and LESC, which is why part of the relevant literature for BLE communication security refers only to SSP security procedures and predates BLE itself.

2.2 Security in Bluetooth Low Energy

As a communication protocol, BLE attempts to provide four [2] security properties:

— Confidentiality;
— Integrity;
— Authenticity;
— Privacy.

To guarantee those properties, the specification defines several procedures, involving different cryptographic keys in the process. As mentioned, security of BLE has changed between versions 4.0 and 4.2. However, the various Bluetooth specifications require backwards compatibility: BLE devices compliant with the latest version of the specification will implement both security specifications. Between both versions, the properties have been kept, some procedures have changed and key management strategy has slightly changed. In the following descriptions, the elements are common to both versions, except when specifically noted.

The keys introduced in the specifications are as follows:

— **STK**: Short-Term Key (specific to Legacy Pairing);
— **LTK**: Long-Term Key;
— **CSRK**: Connection Signature Resolving Key;
— **IRK**: Identity Resolving Key.

The specifications also describe the following security-related procedures: [3]

— **Pairing**: this is the process of exchanging an ephemeral key and optionally authenticating two devices. Different procedures in 4.0 and 4.2;
— **Link Encryption**: this is the process of encrypting the communications between a master and a slave. It requires a LTK or a STK;

2. Some would argue that it also provides "Authorization", but this property is not standardised in the specification, which is why it has been excluded from the list.

3. Depending on the procotol layer considered, those procedures are renamed or aggregated into higher-level procedures.

— **Bonding**: this is the process of creating a bond between two devices. This bond is a persistent mapping between a device and an exchanged set of keys. It must be done after Pairing and over an encrypted link. Few differences between 4.0 and 4.2;
— **Data Signing**: this is the process of signing some commands [4] to a device. It is not available over an encrypted link. It requires the CSRK;
— **Private Address Resolution**: this is the process of resolving (∼decrypting) the address of a device from an advertised one. Requires the IRK.

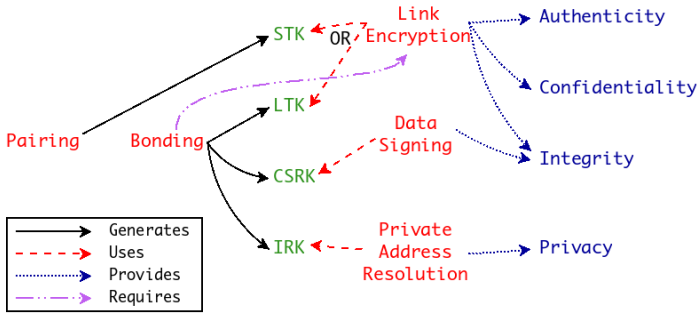

Fig. 1. Mapping between security properties, procedures and keys in Legacy Pairing

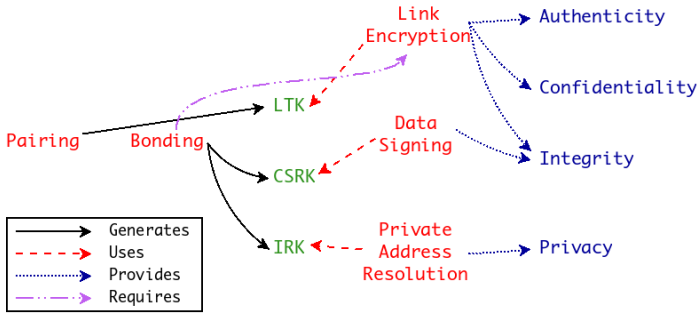

Fig. 2. Mapping between security properties, procedures and keys in LE Secure Connections

4. The only command that can be signed is "ATT Signed Write Command" defined in ATT protocol.

The relationships between the security properties, the aforementioned procedures and the cryptographic material is shown in Figures 1 for Legacy Pairing and 2 for LE Secure Connections. Those procedures are performed in a sequence as follows:

1. Both devices perform a Pairing procedure, after which they share a key (STK in Legacy Pairing and LTK in LE Secure Connections);
2. The Link Encryption procedure is ran using the shared key obtained after pairing;
3. If they wish for it, both devices perform the Bonding procedure, after which they may generate and exchange the CSRK, the IRK and the LTK. In LESC, the LTK generated by the Pairing procedure is stored and not re-generated during Bonding;
4. Subsequent connections will be encrypted with the LTK using the Link Encryption procedure.

The case of authentication is a bit more subtle, because it is optional. In order to know if the link is authenticated, devices have to remember if the key used to encrypt was authenticated, that is if the Pairing procedure provided this property.

Though not explicitely defined, it is possible to infer two attacker levels from the specification:

— **Passive**: this attacker can listen to all messages exchanged between two devices;
— **Active**: this attacker can listen, inject, intercept and forward messages between two devices.

From a communication security standpoint, the goal of an attacker is to get knowledge of the key used in the Link Encryption procedure (either STK or LTK). Once in possession of this key, it is possible to passively decrypt all communications between devices by capturing the Link Encryption procedure. For devices that use the specific Data Signing procedure, which doesn't require encryption, the goal of an attacker is to get knowledge of the CSRK. For devices that use the Private Address Resolution procedure, the goal of an attacker is to get knowledge of the IRK to be able to track a device across time.

Therefore, the way those keys are generated and/or derived is important for the security properties to hold. Those phases occur during the Pairing and the Bonding steps.

2.3 Pairing and Bonding in BLE

Pairing is the process in which two devices exchange a shared key and optionnally authenticate to each other. This process comes in several flavours as the standards describe seven different paring procedures. It is to be realized that the name of Pairing procedure reflects the user interaction required, not their security or the messages exchanged. In practical terms, some procedures bear the same name in Legacy Pairing and in LESC, but do not use the same cryptographic primitives nor exchange the same messages. Therefore, they do not exhibit the same security properties. This situation adds an unnecessary but more critically harmful complexity to the protocol.

In Legacy Pairing (BLE 4.0 and 4.1) were defined three procedures. After Pairing with one of those, both devices share the STK. The Pairing procedures are:

— **JustWorks**: no interaction required from the user;
— **Passkey Entry**: one device displays a six-digit number and the user inputs it in the other device;
— **Out of Band**: the devices use an other communication channel (such as Near-Field Communication (NFC), Infrared (IR), etc.) to share the STK.

In LE Secure Connections (BLE 4.2), four **additional** Pairing procedures were defined. After Pairing with one of those, both devices share the LTK. The Pairing procedures are:

— **JustWorks**: no interaction required from a user;
— **Passkey Entry**: one device displays a six-digit number and the user inputs it in the other device;
— **Numeric Comparison**: both devices display a six-digit number and the user must validate on both devices that they match;
— **Out of Band**: the devices use an other communication channel (such as NFC, IR, etc.) to exchange authentication data.

Therefore, to accurately talk about a specific Pairing procedure, the pairing mode should be mentioned e.g. LESC Passkey Entry. As discussed in section 2.1, devices which are compliant with the latest version of the specification implement those seven procedures.

Essentially, the way the pairing modes work is the following:

— **Legacy Pairing**: the Pairing procedure (JustWorks, Passkey Entry, Out of Band) serves to exchange a temporary secret, from which STK is derived. Authentication is performed based on user interaction, using a commitment scheme.

— **LE Secure Connections**: an ECDH key exchange (over curve P-256) is used to share a secret, from which LTK is derived. The Pairing procedure is used to authenticate the public key of the participants. This authentication relies on user interaction, using a different commitment scheme.

The Bonding procedure is used to exchange keys between devices. Each device asks for a set of keys the other party has to generate and send. For example in Legacy Pairing, the master could ask for a LTK, CSRK and IRK while the slave could ask for a LTK and CSRK. Then the slave will generate an LTK, CSRK and IRK and send them over the encrypted link. The master will generate a LTK and CSRK and send them over the encrypted link. Only the slave's set of key is used after bonding. The rationale for the master to send its own set of keys is in case both devices switch roles in an upcoming connection: the master becomes slave and the keys he generated become the ones used, without requiring a new pairing nor bonding.

3 Related work

Recent research efforts regarding BLE-enabled devices security have focused on smart locks [32,39], connected toothbrushes [20] or e-scooters [14]. In those cases, it is considered that an attacker has physical access to a device and is able to connect and pair to it. For some devices, this is a reasonable model, such as for smart locks whose function is to prevent a physically present attacker to open them.

On the privacy side, several researchers have tackled the issue. A usual target is the advertising mechanism, which broadcasts metadata about a device and leaks some identifiable information [25, 26, 28]. Other research has studied the possibility to fingerprint devices using other public information [43].

Several tools are available to work with BLE, starting with the most obvious which are the official Linux stack BlueZ [4] and its suite of tools or the official Android BLE API [3]. In order to test communication security, several sniffers exist. Proprietary sniffers are TI's one [8] and Adafruit BLEfriend [5]. Open-source sniffers are Ubertooth One [12], btlejack [24] and SniffLE [15]. There are also man-in-the-middle (MITM) frameworks for BLE, which are GATTacker [42], btlejuice [23] and Mirage [6].

All of them have different capacities, for example btlejack is able not only to sniff, but also to take control of a link: it actively disconnects the master and takes its place in the connection. Mirage is also more than a

MITM framework, it aggregates several third-party tools and exposes an interface to manipulate them. Using those capabilities, it is possible to reimplement more complex scenarios. It finally enables to interact directly with a BLE dongle, which can be leveraged to program custom behaviors. In addition, multiple libraries allow to interact with BLE dongles in various programming languages. The advantage of Mirage over those is that lower layers are directly accessible and not wrapped into high-level APIs.

Regarding communication security, researchers have mostly focused on the Pairing procedure, as it is this process which is used to derive a LTK or STK for both devices. Some results regarding BT SSP are provided in this section, however those shouldn't be mistaken for a complete state of the art of Bluetooth Classic communication security. In 2013, Ryan [40] showed that the Pairing procedures JustWorks and Passkey Entry in Legacy Pairing allowed a passive attacker to recover STK. The attacker could use it to passively decrypt all the subsequent communications, by capturing the Link Encryption procedure with knowledge of the key. If both devices bonded afterwards, the attacker was thus able to get the LTK of this bond, having decrypted the link over which it is sent The tool crackle [41] was released at the same time, which enables to decrypt a capture if it uses one of those two Pairing procedures or if the LTK is known in advance and the Link Encryption procedure is part of the capture.

Rosa [38] showed that the commitment scheme used in Legacy Pairing was flawed: the committed value can be changed after being produced. This means that an attacker could complete a pairing as a slave device using the Legacy Passkey Entry Pairing procedure without knowledge of the numeric code displayed by the master. More generally, this shows that the authentication property of this Pairing procedure does not hold. It still holds in the case of the OOB procedure if the exchanged secret stays so.

Regarding LESC security, one must look at research papers studying BT SSP security. Haattaja et al. [29] did a summary of their research on Bluetooth SSP security. The base element is called 'Nino' attack which is a MITM on the SSP JustWorks procedure, therefore applies equally to LESC JustWorks procedure. They show that an active attacker is able to perform a successful MITM on this Pairing procedure and complete pairing with both devices. This is a logical conclusion because this Pairing procedure does not provide authentication (i.e. explicitely does not protect against an active attacker), therefore the result is self-evident. Then, they explored various scenarios where an active attacker was able to downgrade

to a less secure Pairing procedure by carefully tampering with the messages exchanged during pairing. As a result, an active attacker is able to change the Pairing procedure that two devices are about to use. For example, it could downgrade the Pairing procedure from LESC Numeric Comparison which is authentified, to LESC JustWorks which is not. In this case, the attacker will be able to complete a successful pairing. A side note regarding BLE could be added, the same principle could be used to downgrade any LESC procedure to Legacy Passkey Entry, which is broken. Therefore, it is possible for an active attacker to combine Haattaja's approach and Ryan's results to downgrade any LESC procedure to Legacy Passkey Entry and to recover STK then the keys exchanged.

Lindell, A. [33] has shown that the SSP Passkey Entry procedure did not prevent eavesdropping of the numeric code used in an instance of the Pairing procedure. This means that if the code is static or can be guessed from previous ones, an attacker who would be able to force re-pairing devices could authenticate its own public key to both master and slave. As the attacker is in possession of the code used as authentication secret, he could successfully perform a MITM attack on the Pairing procedure. A second result is discussed, which is the ability for an attacker to 'read' the static code from a device. In case a device is preconfigured with a static code, an attacker could perform several pairing attempts and infer the code using the device as an oracle. In at most 20 attempts, an attacker will be able to retrieve the preconfigured code from any device using this scheme. Note that this result, developed for SSP Passkey Entry, is applicable to LESC Passkey Entry, but **not** to Legacy Passkey Entry because of the differences that exist between the uses of the commitment scheme of those modes.

Lindell, Y. [34] performed a formal proof of the security of the SSP Numeric Comparison procedure. This result should be considered encouraging for the security of LESC Numeric Comparison. However, the cryptographic primitives used in SSP and LESC are different, thus the proof may not directly apply to LESC Numeric Comparison.

More recently, Biham et al. [22] showed that vulnerable implementations of the ECDH key echange in SSP and LESC could allow an active attacker to compromise the LTK (or its equivalent in the context of SSP) without affecting the pairing process. This attack will be discussed in more details in Section 6.

Cremers et al. [27] extended the field of formal protocol analysis to add the modelisation of small subgroup and invalid curve attacks in the symbolic model. They implemented this work in the Tamarin prover [11].

Building up on Biham et al.'s work, they tested their implementation on the SSP Numeric Comparison procedure. It found the original attack and also a new one where the authentication property could still be broken if one of the two devices was vulnerable. Their attack applies equally to all SSP and LESC procedures. [5]

Finally, Antonioli et al. [19] explored the ability to reduce the key size down to vulnerable values in Bluetooth Low Energy, extending a previous work they did on Bluetooth Classic [18]. However, besides suggesting and verifying that the key size reduction was transposable to BLE, an in-depth anaysis of the consequences of such attack on the overall security of BLE communications is missing.

Overall, from a communication security standpoint, the Pairing process of BLE has been scrutinized and found vulnerable in various cases.

3.1 Synthesis

It must be mentioned that the NIST published guidelines to Bluetooth (Classic and LE) security in 2017 [37], which provide an accurate view of the threats to communication security up to the publication date, even though results presented in previous section are not referenced. This guide also provides good practices and verifications to properly integrate BT and BLE communications in an application.

There have been many attempts to provide descriptions of how the Pairing procedure occurs, the different steps it requires and how the choice of the Pairing procedure is made. However, from the authors's experience, those descriptions are generally paraphrasing the specification without improving the overall understandablitity of those over-complicated processes. Rather than explaining in details the Pairing and Bonding steps, an alternative description of those processes is proposed. It applies equally to all Pairing procedures:

— **Identification**: devices identify themselves and provide their abilities;
— **Key exchange**: devices exchange a key;
— **Authentication**: devices authenticate the key that has been exchanged;
— **Key generation**: devices generate several keys if needed
— **Key distribution**: devices provide their set of keys to the other.

5. In spite of a typo in the curve used (P-224 is used neither in SSP nor LESC), their results hold for both SSP and LESC.

This alternative description fails to take into account the subtleties of the specification and the fact that some elements are optional, but can be used to contextualise the various works presented.

Authors	Mode	Procedure	Element discussed	Impact
Ryan et al. [40]	Legacy	JustWorks, Passkey Entry	Key Exchange	Confidentiality and Integrity of those procedures do not hold
Rosa [38]	Legacy	Passkey Entry	Authentication	Authentication property in this Pairing procedure does not hold
Haataja et al. [29]	SSP (and LESC)	all	Identification	MITM on the JustWorks procedure is possible, various downgrade attacks to force the use of the JustWorks procedure.
Lindell, A. [33]	SSP (and LESC)	Passkey Entry	Authentication	Passkey is not protected against passive eavesdropper. Using predictable passkeys exposes to MITM attacks.
Lindell, A. [34]	SSP	Numeric Comparison	Authentication	Formal proof of the security of the SSP Numeric Comparison procedure
Biham et al. [22]	SSP (and LESC)	all	Key Exchange	Some implementations do not correctly verify received public key values and are susceptible to MITM attacks.
Cremers et al. [27]	SSP (and LESC)	all	Key Exchange	An implementation which does not correctly verify received public key does not guarantee the Authentication property.
Antonioli et al. [19]	Legacy and LESC	all	Identification	Discussion about the susceptibility of BLE to key length reduction attacks.

Table 1. Summary of relevant literature for Bluetooth Low Energy communication security

Table 1 summarises the results and known attacks on Bluetooth Low Energy. It also provides insight into the procedure step which has been impacted, using the terminology proposed above.

4 Key generation in Bluetooth Low Energy

4.1 Legacy Pairing

In Legacy Pairing, devices end the Pairing process with STK. The LTK, CSRK and IRK are generated as part of the Bonding procedure and exchanged over an encrypted link.

When encrypting the link, a device needs to know the security context (i.e. which key) to use. When persistently stored, the keys are associated with two pieces of information called EDIV and Rand. Those numbers are sent during the Link Encryption procedure in a specific message. When encrypting the link for the first time with a device (e.g. right after a Pairing procedure) those numbers are set to 0 because they have not been generated yet. Each device provides its own EDIV and Rand alongside the set of keys during the Bonding procedure.

Regarding the actual generation of the keys, the specification is loose on the details and doesn't require anything. It gives two examples of generation:

— Keys are randomly generated;

— Keys are generated using a **Key Hierarchy** mechanism.

As there is not much to say about random number generation at a protocol's level,[6] the **Key Hierarchy** option will be discussed in more details.

This option brings in three new keys:

— **ER**: Encryption Root

— **IR**: Identiy Root

— **DHK**: Diversifier Hiding Key

Furthermore, it defines another parameter called DIV, which serves as an identifier for a bond. In this method, EDIV is the masked version of DIV. The generation itself uses two functions called dm and $d1$. Operator $\cdot|\cdot$ denotes concatenation.

dm takes two arguments: a 128-bit key and a 64-bit value.

$$dm(k, r) = aes\_128(k, 0x0000000000000000|r) \mod 2^{16}$$

$d1$ takes three arguments: a 128-bit key and two 16-bit values.

$$d1(k, d, r) = aes\_128(k, 0x00000000000000000000000000|r|d)$$

The key generation works as follows:

6. Other than generic statements like "It should be cryptographically secure"

— Generate DIV (16 bits) and Rand (64 bits);
— Compute $LTK = d1(ER, DIV, 0)$;
— Compute $CSRK = d1(ER, DIV, 1)$;
— Compute $IRK = d1(IR, 1, 0)$;
— Compute $DHK = d1(IR, 3, 0)$;
— Compute $EDIV = DIV \oplus dm(DHK, Rand)$.

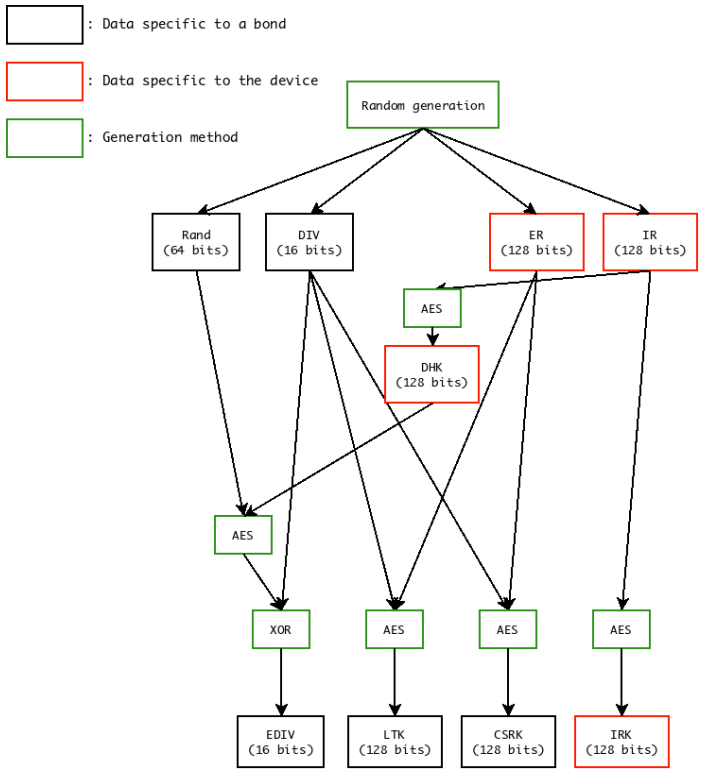

Fig. 3. Summary of the Key Hierarchy generation method

Figure 3 shows the key generation in LE Legacy Pairing. It summarizes the various elements involved as well as their size. It also shows that the IRK is static per device. Even without using the Key Hierarchy method for generating keys, the IRK must be unique: if two devices have paired with the same slave device, they must know the IRK that the slave uses in its advertising messages, hence the unicity.

4.2 LE Secure Connections

In LE Secure Connections, devices exit the Pairing process with a shared LTK. Only the CSRK and IRK are generated as part of the Bonding procedure and exchanged over an encrypted link.

The CSRK and IRK can be generated using the same processes as in Legacy Pairing: either using random generation or using the Key Hierarchy method. In the latter case, it uses the exact same keys, elements and algorithms as in Legacy Pairing.

As explained in section 2.3, all pairing methods of LE Secure Connections start with an ECDH key exchange over curve P-256. After this key exchange, both devices share the Diffie-Hellman Key: **DHKey**. This key is used to generate the LTK using two random numbers and the device addresses of the master and slave.

The random numbers and device addresses are transmitted over the BLE link, no matter which Pairing procedure involved (even OOB). This means that if an attacker is able to eavesdrop the communication and gains knowledge of the private key used by a device during the ECDH exchange, he will be able to retrieve DHKey and therefore the LTK derived by both devices. This is a desired property of BLE communications (from Bluetooth SIG's point of view) to enable debugging encrypted communications with knowledge of one of the used private keys and a capture of the pairing. The specification mentions that devices may implement a Debug mode, in which a default Debug keypair [7] is used for the ECDH exchange and implements exactly this feature.

5 Analysis of the Key Hierarchy generation

This section studies the Key Hierarchy generation method and determines a testing procedure to verify if a device uses it.

5.1 Issues with Key Hierarchy

The problem with this generation scheme is the lack of possible keys: LTK and CSRK are generated by encrypting a changing 16-bit value (DIV) with a 128-bit static key (ER). As described in section 4:

$$LTK = aes\_128(ER, 0x00000000000000000000000000|DIV|0)$$

$$CSRK = aes\_128(ER, 0x00000000000000000000000000|DIV|1)$$

7. This keypair is standardised and can be found in the specification.

There is no overlap between possible LTKs and CSRKs for a device due to the last bit of the cleartext which is fed into the AES, that changes depending on the key to generate. This means that a given device (thus a given ER) can generate only 2^{16} LTKs and 2^{16} CSRKs with this method. More precisely, for a given ER, a device can generate only 2^{16} (LTK, CSRK) key pairs.

This brings two issues: first, when a device pairs with a lot of devices, there is a high probability that two devices will derive the same LTK or CSRK. The birthday paradox indicates that if a device performs 302 pairings, there is a 50% chance that it will generate twice the same DIV, meaning that it will generate twice a given LTK and CSRK.

Second, this also means that if an attacker has access to a vulnerable device for long enough, he can pair it multiple times and enumerate all possible LTKs and CSRKs. Knowing the possible LTKs means that an attacker could decrypt all the past and future connections [8] for which the device has acted as a slave and used a bond. This would enable an attacker to decrypt communications **even without assisting to the Pairing procedure**, which is a significant change compared to existing results on BLE communication security. Knowing the possible CSRKs means that an attacker could sign data and impersonate one of the two devices of the bond. Alternatively, compromising ER of a given device one way or another would allow an attacker to generate all LTKs and CSRKs that would be generated by a device, for the same results.

The root of the issue here is that DIV is 16 bits, hence does not provide enough diversification in the generated keys. As the problem affects the Key Generation step, it is agnostic of the Pairing procedure used for a given Pairing mode. Putting DIV to a larger value (e.g. 127 bits, considering the one-bit switch between CSRK and LTK) would eliminate this issue, but is not possible with the current standards due to message formats.

Devices that use the Key Hierarchy generation algorithm for either Legacy Pairing or LE Secure Connections should change to the first proposed scheme which is random generation. Even though this problem affects both Legacy Pairing et LE Secure Connections, the impact is higher for the former as the LTK is also generated this way.

It should be noted that this issue is likely known by the Bluetooth SIG. In the current version of the specification (v5.2), the key generation procedures are discussed in Vol. 3, Part H, Appendix B., Paragraph 2.2: *"This method [Key Hierarchy] provides an LTK and CSRK with limited*

8. BLE does not offer the Forward Secrecy property

amount of entropy because LTK and CSRK are directly related to EDIV and may be less secure than other generation methods. To reduce the probability of the same LTK or CSRK value being generated, the DIV values must be unique for each CSRK, LTK, EDIV, and Rand set that is distributed."

5.2 Detecting the implementation of the Key Hierarchy generation method

In many cases, BLE devices are closed-source and their code cannot be audited. Even when the BLE stack is open-source, the information regarding the security settings and parameters is not accessible. Thus, there exists no generic approach to straightforwardly determine if the cryptographic material has been generated using a **Key Hierarchy** scheme. This section will provide a testing method applicable even for black-box devices to determine which key generation scheme is used. An important assumption is however made here regarding the key generation methods. It is assumed that the key generation scheme used is whether the **Key Hierarchy** or the random generation, given as examples in the standard.

First approach The main idea is to try to determine the size of the key space on the tested device by collecting CSRKs and LTKs it generates when performing successive pairings. If the **Key Hierarchy** generation method is implemented, it will allow to generate only up to 2^{16} different keys. A random generation of a 128 bit key will provide keys in a 2^{128} key space. Therefore, if after collecting $2^{16} + 1$ LTKs no collision is found, it means the other key generation method is used. However, if a collision occurs, this approach is inconclusive as a small probability exists that the collision is random.

Repeatingly pairing two devices has a non-negligible temporal cost as each pairing implies an update of the BLE state machine and the management of bonds for the slave device. Manual empirical tests on devices have shown that the pairing process is conducted rather quickly ($<$ 1sec). However, erratic behavior [9] can sometimes occur, introducing a longer waiting time required between two pairings. In order to roughly estimate the time necessary to perform the test, it seems reasonable to consider that the delay between two successive pairings ranges between 1

9. Typically, pairing process has been found to be interrupted by the slave device for no apparent reason when one pairing process was done too close to another.

and 10 seconds.[10] Based on those timings, enumerating $2^{16}+1$ LTKs and CSRKs for a device would take between 65537 and 655370 seconds, which corresponds respectively to 18h 12min and 7d 14h 3min.

Generalizing the approach The outcome of the first approach shows that testing for collisions for determining the key space raises a confidence question regarding the result of a test campaign. Indeed, if no collision if found after observing more than the entire smallest key space, there is a 100% confidence that the key generation method is not the vulnerable one. If only a subset of the key space is observed, this confidence decreases. On the other hand, if a collision is observed, the generation method can be either the **Key Hierarchy** or the random generation. Increasing the number of observations will also increase the probability of witnessing a random collision.

As such, the number of observations is a key parameter for estimating the efficiency of the test, by maximizing the probability of detection while minimizing the required testing time. In what follows, the formalization of this decision problem is proposed in order to provide a theroretical basis which can be used to tune the testing parameters in order to adapt the testing efficiency to the operational testing conditions (number of devices to test, total testing time available).

Estimating the test efficiency The following propositions are established:

T: the test is positive

$\overline{T}$: the test is negative

V: the device is vulnerable; it employs the **Key Hierarchy** generation mechanism.

$\overline{V}$: the device is not vulnerable; it does not employ the **Key Hierarchy** generation mechanism.

By hypothesis, a device that does not use **Key Hierarchy** will use the random generation, which yields 2^{128} possible keys. This also means that $P(V)+P(\overline{V})=1$.

The question that naturally arises is how to interpret the result of the test: what does it mean that T is true or false regarding the tested device's key generation method? The quantities of interest are therefore the True Positive rate $P(V|T)$, the False Positive rate $P(\overline{V}|T)$, the False Negative rate $P(V|\overline{T})$ and the True Negative rate $P(\overline{V}|\overline{T})$.

10. On some devices, additional steps are required to put them in pairing mode, in which cases this timing may change.

Using Bayes theorem, we can develop:

$$P(V|T) = \frac{P(V)P(T|V)}{P(V)P(T|V) + P(\overline{V})P(T|\overline{V})}$$

$$P(V|\overline{T}) = \frac{P(V)P(\overline{T}|V)}{P(V)P(\overline{T}|V) + P(\overline{V})P(\overline{T}|\overline{V})}$$

and

$$P(\overline{V}|T) = 1 - P(V|T)$$

$$P(\overline{V}|\overline{T}) = 1 - P(V|\overline{T})$$

Collision probabilities can be determined with the birthday paradox. To simplify notations, the function b is introduced to compute the probability of collision when taking at random t elements out of n:

$$b(t, n) = \left(\frac{n-1}{n}\right)^{\frac{t*(t-1)}{2}}$$

It is possible to know the probability to get at least one collision in t attempts knowing that a device can generate $n_0 = 2^{16}$ and $n_1 = 2^{128}$ keys respectively:

$$P(T|V) = b(t, n_0) = b_0(t)$$

$$P(T|\overline{V}) = b(t, n_1) = b_1(t)$$

After applying those values in the equation of $P(V|T)$, then substituting $P(\overline{V})$ with $1 - P(V)$:

$$P(V|T) = \frac{b_0(t) * P(V)}{b_0(t) * P(V) + b_1(t) * (1 - P(V))}$$

After applying those values in the equation of $P(V|\overline{T})$, then substituting $P(\overline{V})$ with $1 - P(V)$:

$$P(V|\overline{T}) = \frac{(1 - b_0(t))P(V)}{(1 - b_0(t))P(V) + (1 - b_1(t))(1 - P(V))}$$

The probability of encountering a vulnerable device $P(V)$ is unknown and therefore a definitive answer is out of reach, however those functions can be studied with $P(V)$ varying from 0 to 1. The functions $f1$ and $f2$ are defined over the interval $[0; 1] \times [2; 65535]$, using the equations of $P(V|T)$ and $P(V|\overline{T})$ respectively:

$$f1(p,t) = \frac{b_0(t)p}{b_0(t)p + b_1(t)(1-p)}$$

$$f2(p,t) = \frac{(1-b_0(t))\,p}{(1-b_0(t))\,p + (1-b_1(t))\,(1-p)}$$

If the test were perfect, it would always discriminate vulnerable devices from non-vulnerable ones. Using the previous notations, it would mean that $f1(p,t) = 1$ and $f2(p,t) = 0$.

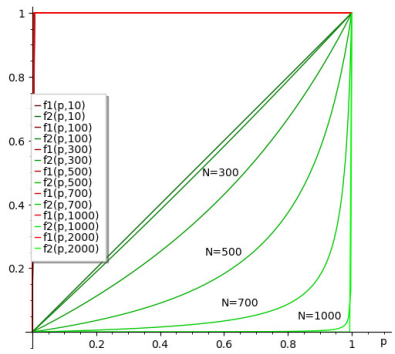

Fig. 4. Evolution of $f1(p,t)$ and $f2(p,t)$ depending on the number of trials

Figure 4 shows the impact of the number of trials on the expected accuracy of the test. Nothing significant can be seen at this scale regarding the evolution of the True Positive rate: if the test is positive then it means with near certainty that a device is vulnerable. What can be noticed instead is the variation of the False Negative rate. It still depends a lot on p, but choosing a large enough number of trials can reduce a lot the expected number of False Negative, hence the number of devices which are misclassified as not vulnerable.

Table 2 summarises some threshold numbers for several values of t. The first two columns provide a condition on the proportion of vulnerable devices p to get more than 99% of True Positive results (the lower the better) and less than 1% of False Negative results (the higher the better). The last two columns display the expected rate of True Positives and False Negatives under the assumption that there are 10% devices vulnerable.

The green curves in figure 4 represent the False Negative rate relatively to the proportion of vulnerable devices. It becomes sharper with an

	f1(p, t=N) >= 0.99	f2(p, t=N) <= 0.01	f1(0.1, t=N)	f2(0.1, t=N)
N=10	$p >= 1.907 * 10^{-32}$	$p <= 1.001 * 10^{-2}$	≈ 1.000	$9.994 * 10^{-2}$
N=100	$p >= 1.980 * 10^{-32}$	$p <= 1.078 * 10^{-2}$	≈ 1.000	$9.340 * 10^{-2}$
N=300	$p >= 2.633 * 10^{-32}$	$p <= 1.963 * 10^{-2}$	≈ 1.000	$5.307 * 10^{-2}$
N=500	$p >= 4.265 * 10^{-32}$	$p <= 6.347 * 10^{-2}$	≈ 1.000	$1.629 * 10^{-2}$
N=700	$p >= 7.292 * 10^{-32}$	$p <= 2.969 * 10^{-1}$	≈ 1.000	$2.651 * 10^{-3}$
N=1000	$p >= 1.454 * 10^{-31}$	$p <= 9.538 * 10^{-1}$	≈ 1.000	$5.440 * 10^{-5}$
N=2000	$p >= 5.816 * 10^{-31}$	$p <= 1 - \epsilon$	≈ 1.000	$6.290 * 10^{-15}$

Table 2. Summary of results regarding f1 and f2; with $\epsilon < 10^{-3}$

increasing number of trials. Optimizing the efficiency of the test becomes equivalent to choosing a value of t such as the False Negative rate is minimized and ideally independently from the value of p. The ideal green curve would be a straight line from (0,0) to (1,0). The problem can then be translated to a tradeoff between the number of trials and the minimization of the area under the green curve. In this case, this can be done by taking the definite integral of $f1$ and $f2$ over p. The functions $F1$ and $F2$ are defined:

$$F1(t) = \int_0^1 f1(p,t)dp$$

$$F2(t) = \int_0^1 f2(p,t)dp$$

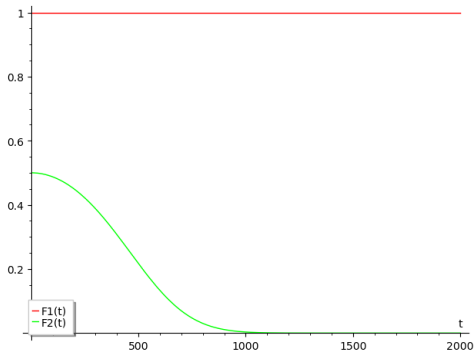

Fig. 5. F1(t) and F2(t) for $t \in [2; 2000]$

Figure 5 shows the evolution of the area under both curves with the number of trials, here varying from 2 to 2000. The curves remain flat from 2001 to 65536 trials, not shown here. This validates the observations made from Figure 4. Furthermore, it shows that the False Negative rate improves a lot with the number of trials at the beginning, then improves only marginally after that.

Overall, this shows that choosing a number of observations comprised between 500 and 1000 will result in an interesting tradeoff between temporal cost and test efficiency. The choice of parameters is ultimately a compromise between the time needed for the test and the desired precision. Furthermore, it can be deduced that between the naive test of $2^{16}+1$ pairings and 1000 pairings, the difference in accuracy is neglectable whereas the temporal cost difference is significant.

The numeric values and graphics in this section have been generated with SageMath [10]. The script to compute all the elements presented can be found in Appendix A.

6 Issues with the ECDH key exchange

This section will detail the work of Biham et al. [22] and the improvements of Cremers et al. [27]. It will introduce the need to study an overlooked element in those papers which is the key retrieval scenario. After defining it and discussing it in depth in the context of BLE, testing procedures are proposed to verify that implementations are not affected by this problem.

6.1 Previous Work

Biham et al. [22] studied the use of ECDH in SSP and LESC. They noticed that in the exchange, both devices send their full public key: the X and Y-coordinate of their public point. However, in the authentication process only the X-coordinate is verified, which means that an active attacker can change the Y-coordinate without affecting the authentication property of the Pairing procedure used.

Their attack consists in changing the Y-coordinate to 0 to reduce the number of possibilities DHKey can take. They have imagined two versions of this attack: one where they only affect the key exchange and has a 25% chance of success and another where they affect the key exchange and all subsequent messages which has a 50% chance of success. In both cases, the principle remains the same. The question that arises naturally is: what happens when an attacker can change the Y-coordinate?

As usual in ECDH, DHKey is generated by multiplying a device's private key with the peer's public key. When multiplying the point $(x, 0)$ with any private key the result will be $(x, 0)$ (the same point) or a specific point of the curve called **Point at infinity** P_∞.

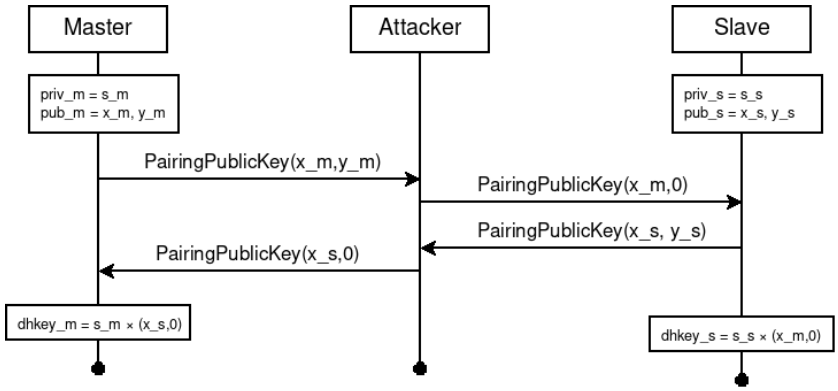

Fig. 6. Attack proposed by Biham and Neumann

The attack they proposed is depicted in figure 6. At the end, the master has derived $DHKey_m = (x_s, 0)$ or P_∞ and the slave has derived $DHKey_s = (x_m, 0)$ or P_∞ Therefore, there is a 25% chance that both devices agree on DHKey being P_∞,[11] in which case the attacker also knows it hence he will be able to retrieve the LTK generated. For more details about the rules of addition over elliptic curves that make this attack possible, one could refer to an article [35] published shortly after the original paper.

It should be noted that changing this coordinate invalidates the point: it is no longer on the chosen curve and this can be detected by the implementation. Another finding of Biham et al. was that some implementations did not implement this verification and were vulnerable to this attack. Also, to be successfully conducted this attack requires the two devices to be vulnerable. If one target implements the verification, then the pairing won't complete. This was given the CVE identifier 2018-5383 and published during summer 2018. Since then, the specification explicitly mentions that the ECDH public key must be validated for successfully completing the Pairing.

11. It was determined by the researchers that the memory representation of this value was 0 on the studied implementations.

Cremers et al. [27] built up on this result using the automatic security protocol prover Tamarin [11]. Their implementation found the same attack. It also found that when only one device is vulnerable, it is possible to break the authentication property of the Pairing procedure. Here is a quick outline of their attack:

1. The attacker replaces the Y-coordinate of the public key of the protected device by 0 but transmits the unmodified public key of the vulnerable device;
2. Both devices will complete the Pairing procedure without problem;
3. When the pairing ends, both devices will have unmatched DHKey;
4. The attacker then takes the place of the protected device by guessing the value of DHKey of the victim, it has 50% of success;
5. If successful, the protected device will notice a different DHKey and disconnect; the victim device will be connected and paired with the attacker.

Cremers et al. also found that in case of a static ECDH key, a vulnerable device is susceptible to a key retrieval attack. The specification has slightly changed between versions 5.0 and 5.1: requirements regarding public key validation are more precise in order to prevent these types of attacks. The reader will find the paragraph on the validation of received public keys in Vol 3, Part H, section 2.3.5.6.1 of the specification. The section detailing the private key renewal requirements is in Vol 3, Part H, section 2.3.6. In the latest version of the specification, it is mentioned that the received public key must be verified against the curve equation and that the private key should be rotated in the worst case every 8 pairing attempts.

Both research efforts brushed off the key retrieval scenario by mentioning that almost all devices did re-generate their keypair at each pairing attempt. However, during our observation of BLE devices, it was found that some embedded devices do not follow the specification and keep the same keypair for longer than specified. Therefore, the key retrieval scenario cannot be completely overruled and deserves to be studied.

6.2 Studying the complexity of key retrieval in BLE

In the context of elliptic curves, key retrieval is a type of invalid curve attack which has been described first by Biehl et al. [21] and developed by Antipa et al. [17]. This type of vulnerability has already been found in two TLS implementations [31], leading to a private key compromise when TLS_ECDH_* cipher suites were used by the server.

The generic idea behind this attack is that an attacker will send invalid points to an oracle, which will multiply those points by a constant secret value. The attacker can retrieve the result of this computation and compute the secret little by little. In the attack proposed by Biham et al. [22], replacing the Y-coordinate of the two public keys with 0 is a way of constructing two points which will generate each a group of order 2. This is another way to explain the 25% chance of success of their attack.

To retrieve the private key, an attacker needs to generate multiple points $p_0, p_1, p_2, \ldots$ which will generate groups of order $n_0, n_1, n_2, \ldots$. For one generated group, knowing which point was computed by the target device is equivalent to knowing the private key of the target device modulo the order of the group. For example, retrieving the computed key after sending the point $(x, 0)$ is equivalent to knowing the private key of the target modulo 2. The Chinese Remainder Theorem states that that when we retrieve a number s modulo $n_0, n_1, n_2, \ldots n_i$, it is possible to compute $s \mod N$, with $N = lcm(n_0, n_1, n_2, \ldots n_i)$ (the least common multiple). In particular if all n are relatively prime, we have $N = \prod_{j=0}^{i} n_j$. If an attacker retrieves enough moduli such that N is greater than the order of the curve used, then it is possible to completely retrieve the private key of the target.

The process for generating interesting invalid points and retrieving the complete private key from an oracle has already been developed in several publications [17, 21, 31]. The oracle is a way to retrieve the private key modulo a single n. By making several calls to the oracle with different n, it is possible to retrieve the complete private key. Next section will demonstrate how it is possible to build such an oracle from a vulnerable Bluetooth Low Energy device and discuss the cases of a master and slave device.

Figure 7 shows the scenario needed to perform this attack. Again, as in the key enumeration against the insecure **Key Hierarchy** generation method, an attacker needs to have access to a device for a certain amount of time and to be able to perform many pairing attempts with it.

Figure 8 depicts the pairing process in LE Secure Connections. Beyond the steps of identification and key exchange, the exact messages exchanged for authentication depend on the Pairing procedure used (JustWorks, Passkey Entry, ...). After those messages, the Pairing procedure ends with an exchange of messages of type **PairingDHKeyCheck** which are used, as the name suggests, to verify that DHKey derived at both ends of the connection matches.

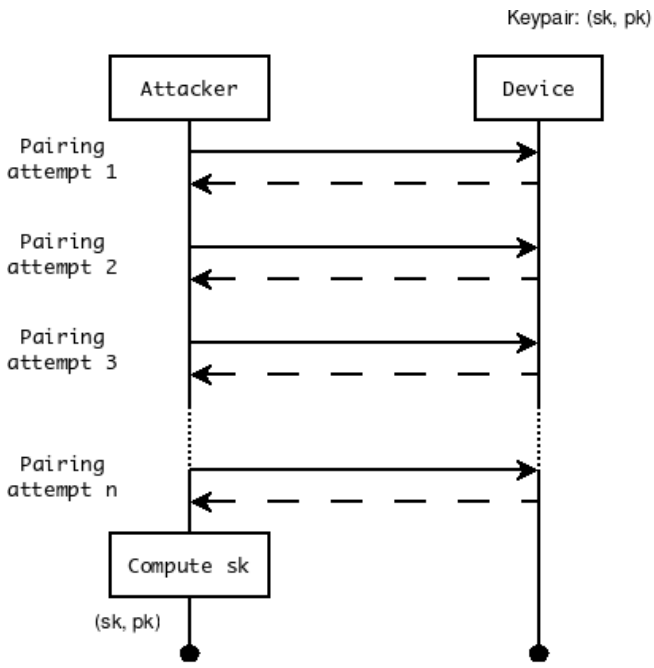

Fig. 7. Outline of the Key Retrieval process

In the studied case, one end of the connection is malicious and tries to retrieve the private key of the other device. What can be seen in Figure 8 is the asymmetry between the Master and Slave roles: the Master initiates the exchange of DHKeyCheck messages, while the Slave only answers to the Master.

In the easiest case, the attacker poses as a Slave and wants to retrieve the private key of the Master modulo n.

1. The Slave sends a point which generates a subgroup of order n;
2. The Slave receives the PairingDHKeyCheck sent from the Master;
3. The Slave aborts the Pairing procedure;
4. The Slave computes offline which subgroup element was used to generate the DHKey computed by the Master and which produces the gathered PairingDHKeyCheck value.

Therefore, to get the information of the private key modulo n, the attacker needs **1** pairing attempt and **n-1** offline computations by posing as a Slave.

In the opposite case, the attacker poses as a Master and wants to retrieve the private key of the Slave. Note that in this case, the attacker

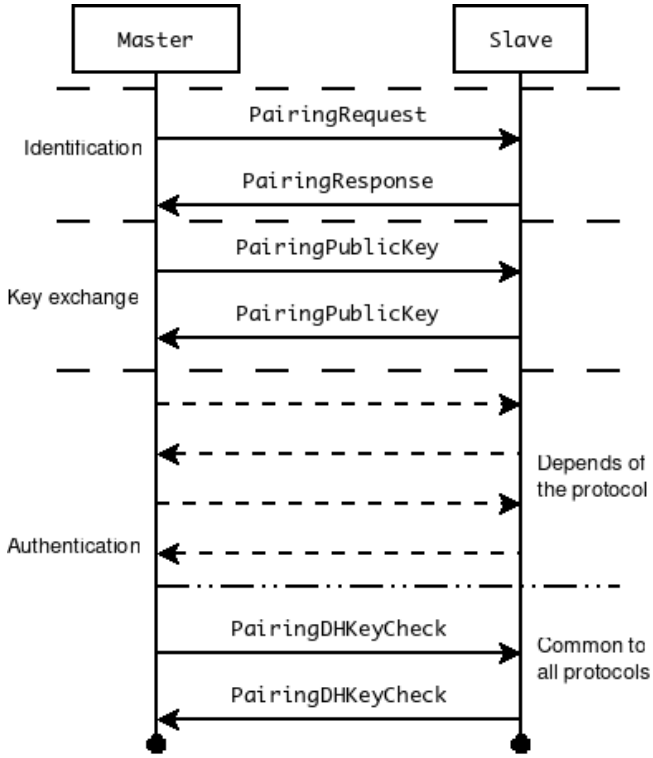

Fig. 8. Overview of the Pairing procedure in LE Secure Connections

has to guess values that produce a correct Master PairingDHKeyCheck in order to receive the Slave PairingDHKeyCheck value. This process is outlined as follow:

1. The Master sends a point which generates a subgroup of order n;
2. The Master takes the first element of the subgroup and sends a matching PairingDHKeyCheck;
   — If the Slave answers (i.e. the Pairing completes), the Master stops;
   — Else, the Master restarts the Pairing process using the same point and uses the next element in the subgroup.

Therefore, to get the information of the private key modulo n, the attacker needs in the worst case **n - 1** pairing attempts and as much offline computations by posing as a Master.

Hence, if a BLE device does not verify the public key received in LE Secure Connections and does not renew it's key material frequently (as mandated by the standard), it is possible for an attacker to retrieve

partially or completely the ECDH private key, depending on the role of the victim device and the number of pairing attempts performed. It has been shown how to use both a Slave and a Master as an oracle in order to finally retrieve information about this private key. In order to evaluate the exact cost of this attack, one would need more informations about the target role, the time between two pairing attempts and decide of a tradeoff regarding the amount of computation to do offline and the number of pairing attempts.

This does not represent a new vulnerability, but extends the work of Biham et al. [22] and Cremers et al. [27] by studying the impact of CVE-2018-5383 in other scenarios. Devices must check the public key received and not perform computations based on invalid ones. Furthermore, it is a good practice, in the case of BLE to regenerate the keypair regularly, ideally at each new pairing attempt Finally, imposing a delay between too many pairing attempts, either successful or failed, could prevent this type of attacks.

All those recommendations are clearly identified as mandatory in the latest versions of the Bluetooth specification.

6.3 Testing for ECDH keypair renewal

The specification mentions in Vol 3, Part H, paragraph 2.3.6 that devices should regenerate their keypairs every 8 pairing attempts in the worst case. Hence, if a device has kept the same public key during 9 successive pairings, then it most likely does not follow this line of the specification. Despite being a non-compliance to the standard, this element alone does not endanger communication security with regards to the considered attacker models.

In order to perform this test, it is just necessary to actually initiate several legitimate pairings and check when the public key changes.

6.4 Testing the validation of ECDH public keys

A testing methodology is described for determining if a target device is vulnerable to CVE-2018-5383 in [22]. A Bluetooth stack is modified so as to allow using invalid ECDH keypairs and is used in order to perform several pairing procedures with various devices. If the pairing succeeds it means that the device does not verify the validity of the public key and is therefore vulnerable.

The Bluetooth SIG has recently opened its test criterias, among which a test procedure for the validation of public keys. The drawback of the

described method is that it only ensures that a device does not accept a public key having a Y-coordinate at 0. The problem of this type of test is already addressed by Biham et al. [22] and they propose to choose invalid points of small order (e.g. 3) for testing devices. However, SIG's approach does not test with high confidence for invalid points whose coordinate is different than 0.

Another problem is that when testing a Slave device, some targets do not immediately reject a public key. In this case, one has to guess the value of DHKey that has been derived by the Slave in order to try to complete a successful pairing. This guess has only one chance over the order of the generated group to succeed, for example it has a 33.33% chance of success when providing a point which generates a group of order 3. The implication is the following: when an invalid public key is used and the slave has rejected the pairing at the DHKeyCheck step, it is impossible to distinguish if the rejection cause was an invalid public key or if it was a wrong guess for DHKey. For this reason, multiple pairing attempts are needed to have a higher confidence that a slave has rejected pairing because it truly verifies the public key.

It should be noted that there exists a Proof of Concept to test the presence of the vulnerability in Bluetooth Classic devices. This proof of concept is included in the framework InternalBlue [16] and requires the instrumentation of a Nexus 5 phone. In addition of being specific to Bluetooth Classic, it only supports Master mode, hence is unable to test Master devices.

7 Implementation and test results

This section will describe the choices made to implement the two tests aforementioned.

7.1 Existing implementations

On an implementation level, BT and BLE define two entities:

— The Controller, which handles the radio link and some other procedures;

— The Host, which gives orders to the Controller and implements higher-level protocols.

Both components communicate through the Host-Controller Interface (HCI). In Bluetooth Low Energy, it is easier to perform tests about the pairing process (relatively to Bluetooth Classic) for one main reason: the

pairing process is implemented completely by the Host, while in BT it is mainly implemented by the Controller.

Open-source BLE Host stacks include Linux's stack BlueZ [4], Android's stack Fluoride [9], Mynewt's stack NimBLE [1] for embedded devices and Zephyr project's stack [13] also for embedded devices.

Though those projects are interesting when the goal is to develop a BLE-enabled device, in order to test their security the framework Mirage [6] is more suited. It enables to develop new tests and applications in Python, thus to iterate quicker. Also, it interfaces with many type of devices natively. When performing tests about the Pairing and Bonding processes, the ideal case if to interface directly with a BLE chip using the HCI protocol, which is exactly what Mirage enables.

7.2 Testing for the Key Hierarchy generation method

In order to test the Key Hierarchy generation method, one has to bond many times with a device to observe the keys generated.

Perform multiple bondings with a device To bond with a device, one always has to perform an entire pairing attempt first In order to automate the pairing process, not all devices have the same abilities. In some cases, it is needed to instrument a device to pair it with another one or to put it in pairing mode. Rather than trying to provide an exhaustive list, different types of devices will be discussed.

First, some devices require no instrumentation at all: they advertise themselves and accept connections automatically. In this case, one only has to scan, connect, pair then disconnects as many times as needed to perform the test. This approach can be applied to test various BLE stacks, where it is possible to develop its own application using a provided API. Some commercial devices also exhibit this behavior.

In other cases, the device needs a user confirmation to accept a pairing attempt. This is the case, amongst others, of smartphones, whose leading operating systems at the time of writing are Android and iOS. To study phones, the application nRFConnect may be used to work more easily with BLE. With it, it is possible to put a device into advertising and connectable mode: it will accept connections and pairing requests (upon user confirmation). In order to automate the pairing accept, one has to send touch events to a device. In the case of Android, it is possible to send touch events using Android Debug Bridge and shell commands, as explained in [2]. For iOS, the authors are not aware of a similar way to

send touch events. Alternatively, one could replicate the approach used by Markert et al. [36] where they used a mecanical robot for this.

Other devices such as smartwatches may also need a user confirmation (e.g. physical keypress) to accept pairings. In addition, some devices must be put into pairing mode for each attempt, using dedicated buttons or combination of buttons. In those cases using a mecanical approach may be the least intrusive way to automate the pairing process, however no general rule can be given here.

Overall, the way a device is instrumented will also impact the performance of the test. If several actions are needed to perform one pairing, it will slow the testing process as much.

Performing multiple pairings on a controlled device In order to study the efficiency of the proposed test, it was chosen to use a controlled device. This device is a BLE Nano 2 from RedBear, based on the nRF 52832 chipset. The firmware used was a custom version of Mynewt's NimBLE. The example application 'bleprph' was installed, configured with Pairing and Bonding support. This implements the first case discussed: the device advertises itself automatically, no inputs are needed to put it into pairing mode or to make it accept the pairing.

There was a modification done to the underlying stack: by default, NimBLE uses the random generation mechanism. The stack was modified to implement the Key Hierarchy mechanism, hence the device tested was known to be vulnerable.

First, the test was ran using the parameter of 1000 tests. Mirage was used on the testing computer, with the internal BLE chipset. The Pairing procedure was set to Legacy JustWorks, that is the one with the least messages exchanged. The first collision was found after 389 pairing attempts, in 54 minutes and 6 seconds. Overall, in 1000 tests, there were 7 collisions in total. Then, 2000 pairings were performed to study the interval between two pairing attempts: this setup performs one pairing attempt every 8.5 seconds in average (standard deviation is 1.38s). The limiting factors were the time needed to scan (set to 2 seconds) and the interval between the end of an attempt and the next one (set to 2 seconds). Those delays are needed to let the different stacks reconfigure themselves. For example, without pause between two attempts, the chipset of the computer failed every few attempts.

Overall, this shows that it is possible to perform multiple pairings with devices. The value of 1 to 10 seconds per pairing attempt given in Section 5 proved a bit optimistic, with real values of at least several

seconds for non-optimised versions. This observation reinforces even more the use of a statistic test which observes only a subset of possible keys. During a test campaign on a dozen devices, it was found that at least one commercial implementation uses the Key Hierarchy key generation mechanism, which was successfully detected with the proposed method. None of the tested devices implemented the recommendation of gradually adding delays between pairing attempts. An ongoing responsible disclosure process prevents from providing more precise results.

7.3 Testing the ECDH key exchange

In its previous state, Mirage had a pairing module whith support for LE Legacy Pairing. Therefore, the new messages and cryptographic primitives added in LE Secure Connections had to be developed and were added to Mirage. Finally, the module **ble_pair** was modified to implement the new pairing methods specific to LE Secure Connections.

Overall, this module now supports:

— Pairing using LE Legacy Pairing as Master and Slave, for procedures JustWorks, Passkey Entry and Out of Band
— Pairing using LE Secure Connections as Master and Slave, for procedures JustWorks, Passkey Entry and Numeric Comparison

As mentioned in Paragraphs 6.3 and 6.4, there are two tests that can be performed on this key exchange:

1. Verify that the keypair is rotated as mandated by the specification
2. Verify that the public key is validated by the device under test

While the implementation of the first test is straightforward, the second one is necessarily imperfect. It was observed that some implementations reject an invalid point right after receiving it which seems to indicate that they verify it right away. However, some devices reject a pairing with an invalid point when performing the DHKeyCheck exchange. In this case, the test should be repeated several times, with several invalid points, to gain confidence that the device indeed validates the public key.

The functionality was tested against Android in particular to verify that the implementation works. The result are as expected: old Android versions are affected while recent ones are patched.

Regarding the proposed key retrieval attack, tests have concluded that some devices are vulnerable to CVE-2018-5383 and that some devices use static key pairs. As there was no overlap between those two sets, this particular attack could not be completed. Yet, these independent

observations show that the existence of such vulnerable devices cannot be completely overruled.

Overall, the tests on ECDH discussed in this paper showed their efficiency on commercial devices.

8 Conclusion

The security of Bluetooth Low Energy devices is of great importance due to their proliferation in recent years. As many standards, the BLE specification is a rather complex document which does contribute to give neither a clear comprehension of the security mechanisms nor a precise view of the best way to implement those. Furthermore, backwards compatibility raises questions about the way security is implemented and managed in devices which are compatible with several versions of the standard. After describing the various mechanisms meant to provide security properties to this protocol, the previous work related to BLE communication security was extensively discussed. This paper focused on the different processes for key exchange and key generation in use in Bluetooth Low Energy.

Then, a description of two weaknesses that become exploitable when some specific security requirements from the standard are not correctly implemented is provided. The first relies on a lack of entropy in one of the key generation methods described in the standard. The second one represents an extension of CVE-2018-5383, which impacts the key exchange in BLE.

This work showed that the Key Hierarchy key generation method, used to generate the LTK in Legacy Pairing and the CSRK in all Pairing modes suffers from a lack of entropy. An attacker with the ability to repeatedly bond with a Slave device which uses this method is able to get all the keys that the device will generate. Hence, this attacker is be able to compromise the security of procedures which uses these keys, in particular the Link Encryption and Data Signing procedures, which provide Confidentialy, Integrity and Authenticity in the BLE protocol. This proves to be significant change in the case of Legacy Pairing where existing attacks all require an attacker to capture the Pairing procedure. This attack affects JustWorks, Passkey Entry and Out of Band procedures, require the implementation of the Key Hierarchy key generation and it effectiveness will be conditioned by the way delays are introduced between several pairing attempts, as required by the standard.

It was already known that, when a device uses LE Secure Connections and uses a static or semi-static ECDH private key; then if it is vulnerable

to CVE-2018-5383, it is possible to mount a small subgroup key recovery attack to retrieve the private key in use. However, this scenario was swept away by previous relevant literature. Based on observations of live devices, it was considered plausible and therefore discussed in depth in this paper. The complexity of this attack for different scenarios has been provided. When in possession of the ECDH private key of a device, an attacker which captures the LESC Pairing procedures made by this device will naturally recover the LTK derived, hence will be able to decrypt communications between the victim device and its interlocutor. This however relies on implementation errors omitting to correctly implement the ECDH key verification and the ECDH key renewal as required by the latest standard.

That being said, devices implementing correctly the security requirements of the latest version of the standard will not be impacted by those attacks.

Those attacks being conditioned to implementation errors, an effort was made to provide a test methodology for each weakness. This will enable determining if a target device, which BLE stack might be closed source and misconfigured, is vulnerable to the two attack vectors that were described. The Key Hierarchy vulnerability being a diversity problem, a probabilistic approach based on collision finding is proposed. It has been shown that there exists a tradeoff between the number of pairing trials (i.e. the temporal cost of the test) and the precision of the test. This tradeoff was extensively discussed to enable a test operator to select the best parameter choice with regards to his operational requirements. Regarding the second scenario, one of the prerequisites being a vulnerability to CVE-2018-5383, it is proposed to test for a bad key verification by performing a pairing attempt with an invalid public key. Additionally, one can verify the key renewal mechanisms implemented.

Furthermore, those tests do not rely on trying to exploit the vulnerabilities in order to prove that devices are protected.

A Sage script

```
#!/usr/bin/env sage

from sage.all import *
import time

# Work with sufficiently precise numbers to handle little quantities
R300 = RealField(300)

# Birthday paradox
```

```
def collision_proba(M, N):
    """Returns the probability of getting a collision by picking M
        elements
    at random in a set of N elements"""
    e = (R300(M)*(R300(M)-1))/2
    return (1 - ((R300(N-1)/R300(N)))**e)

# Get f1(p, t=nb_trials) and f2(p, t=nb_trials)
def make_probas(nb_trials):
    proba_coll_key_hierarchy = collision_proba(nb_trials, 2**16)
    proba_coll_random = collision_proba(nb_trials, 2**128)

    p = var('p')
    f1 = proba_coll_key_hierarchy*p/(proba_coll_key_hierarchy*p +
        proba_coll_random*(1-p))
    f2 = (1-proba_coll_key_hierarchy)*p/((1-proba_coll_key_hierarchy
        )*p + (1-proba_coll_random)*(1-p))

    return (p, f1, f2)

fig = text( "p", (1.1, -0.03), color="black")
for (idx, i) in enumerate([10, 100, 300, 500, 700, 1000, 2000]):
    (p, f1, f2) = make_probas(i)
    fig += parametric_plot( (p, f1), (p, 0, 1), rgbcolor=(0.4 + 0.1*
        idx,0,0), legend_label="f1(p,{})".format(i))
    fig += parametric_plot( (p, f2), (p, 0, 1), rgbcolor=(0,0.4+0.1*
        idx,0), legend_label="f2(p,{})".format(i))
    print(i)
    sol1 = solve((f1 >= 0.99), p)
    print("f1(0.1, {}) = {}".format(i, R300(f1(0.1))))
    print("f1(p, {}) >= 0.99: p >= {}".format(i, R300(sol1[1][0].
        right())))
    sol2 = solve((f2 <= 0.01), p)
    print("f2(0.1, {}) = {}".format(i, R300(f2(0.1))))
    print("f2(p, {}) <= 0.01: p <= {}".format(i, R300(sol2[0][0].
        right())))

fig += text( "N=300", (0.59, 0.5), color="black")
fig += text( "N=500", (0.6, 0.25), color="black")
fig += text( "N=700", (0.65, 0.09), color="black")
fig += text( "N=1000", (0.9, 0.05), color="black")

show(fig)
fig.save("n_varies.png")

nb_trials = var('t')
e = (nb_trials*(nb_trials-1))/2
proba_coll_key_hierarchy = (1 - ((R300(2**16-1)/R300(2**16)))**e)
proba_coll_random = (1 - ((R300(2**128-1)/R300(2**128)))**e)

f1 = proba_coll_key_hierarchy*p/(proba_coll_key_hierarchy*p +
    proba_coll_random*(1-p))
f2 = (1-proba_coll_key_hierarchy)*p/((1-proba_coll_key_hierarchy)*p
    + (1-proba_coll_random)*(1-p))

F1 = f1.integral(p, 0, 1)
F2 = f2.integral(p, 0, 1)
```

```
fig = parametric_plot( (nb_trials, F1), (nb_trials, 2, 2000),
    rgbcolor=(1,0,0), legend_label="F1(t)")
fig += parametric_plot( (nb_trials, F2), (nb_trials, 2, 2000),
    rgbcolor=(0,1,0), legend_label="F2(t)")
fig += text( "t", (2000, +0.03), color="black")
fig.set_aspect_ratio('automatic')
fig.save("integrals.png")
```

Listing 1. Sage script

References

1. Apache Mynewt. https://mynewt.apache.org/.
2. Automating Input Events on Android – RPLabs – Rightpoint Labs. https://www.rightpoint.com/rplabs/automating-input-events-abd-keyevent.
3. Bluetooth low energy overview. https://developer.android.com/guide/topics/connectivity/bluetooth-le.
4. BlueZ. http://www.bluez.org/.
5. Introduction | Introducing the Adafruit Bluefruit LE Sniffer | Adafruit Learning System. https://learn.adafruit.com/introducing-the-adafruit-bluefruit-le-sniffer?view=all.
6. Mirage documentation — Mirage 1.1 documentation. http://homepages.laas.fr/rcayre/mirage-documentation/.
7. Our History. https://www.bluetooth.com/about-us/our-history/.
8. PACKET-SNIFFER SmartRF Protocol Packet Sniffer | TI.com. http://www.ti.com/tool/PACKET-SNIFFER.
9. platform/system/bt - Git at Google. https://android.googlesource.com/platform/system/bt/.
10. SageMath Mathematical Software System - Sage. http://www.sagemath.org/.
11. Tamarin Prover - Security protocol verification tool. https://tamarin-prover.github.io/.
12. Ubertooth One - Great Scott Gadgets. https://greatscottgadgets.com/ubertoothone/.
13. Zephyr Project | Home. https://www.zephyrproject.org/.
14. Don't Give Me a Brake - Xiaomi Scooter Hack Enables Dangerous Accelerations and Stops for Unsuspecting Riders. https://blog.zimperium.com/dont-give-me-a-brake-xiaomi-scooter-hack-enables-dangerous-accelerations-and-stops-for-unsuspecting-riders/, February 2019.
15. nccgroup/Sniffle. https://github.com/nccgroup/Sniffle, January 2020. original-date: 2019-08-17T05:26:18Z.
16. seemoo-lab/internalblue. https://github.com/seemoo-lab/internalblue, January 2020. original-date: 2018-09-04T14:17:46Z.
17. Adrian Antipa, Daniel Brown, Alfred Menezes, Rene Struik, and Scott Vanstone. Validation of Elliptic Curve Public Keys. page 13.
18. Daniele Antonioli, Nils Ole Tippenhauer, and Kasper Rasmussen. The KNOB is Broken: Exploiting Low Entropy in the Encryption Key Negotiation Of Bluetooth BR/EDR. page 16.

19. Daniele Antonioli, Nils Ole Tippenhauer, and Kasper Rasmussen. Low Entropy Key Negotiation Attacks on Bluetooth and Bluetooth Low Energy. page 13.
20. Axelle Apvrille. Ingénierie inverse d'une brosse à dents connectée. page 20.
21. Ingrid Biehl, Bernd Meyer, and Volker Muller. Differential Fault Attacks on Elliptic Curve Cryptosystems (Extended Abstract). page 16.
22. Eli Biham and Lior Neumann. Breaking the Bluetooth Pairing – Fixed Coordinate Invalid Curve Attack. page 26.
23. Damien Cauquil. Digitalsecurity/btlejuice. `https://github.com/DigitalSecurity/btlejuice`, 2020.
24. Damien Cauquil. virtualabs/btlejack. `https://github.com/virtualabs/btlejack`, January 2020. original-date: 2018-08-07T19:13:53Z.
25. Guillaume Celosia and Mathieu Cunche. Saving Private Addresses: An Analysis of Privacy Issues in the Bluetooth-Low-Energy Advertising Mechanism. pages 1–10, December 2019.
26. Guillaume Celosia and Mathieu Cunche. Discontinued Privacy: Personal Data Leaks in Apple Bluetooth-Low-Energy Continuity Protocols. *Proceedings on Privacy Enhancing Technologies*, 2020(1):26–46, January 2020.
27. Cas Cremers and Dennis Jackson. Prime, order please! revisiting small subgroup and invalid curve attacks on protocols using diffie-hellman. Cryptology ePrint Archive, Report 2019/526, 2019. `https://eprint.iacr.org/2019/526`.
28. Kassem Fawaz, Kyu-Han Kim, and Kang G Shin. Protecting Privacy of BLE Device Users. page 18.
29. Keijo Haataja, Konstantin Hyppönen, Sanna Pasanen, and Pekka Toivanen. *Bluetooth Security Attacks*. SpringerBriefs in Computer Science. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.
30. Dave Holl and er. The State of Bluetoothin 2018 and Beyond. `https://www.bluetooth.com/blog/the-state-of-bluetooth-in-2018-and-beyond/`, April 2018.
31. Tibor Jager, Jorg Schwenk, and Juraj Somorovsky. Practical Invalid Curve Attacks on TLS-ECDH. page 19.
32. Slawomir Jasek. Blue picking - hacking Bluetooth Smart Locks. page 228.
33. Andrew Y Lindell. Attacks on the Pairing Protocol of Bluetooth v2.1. page 10.
34. Andrew Y. Lindell. Comparison-Based Key Exchange and the Security of the Numeric Comparison Mode in Bluetooth v2.1. In Marc Fischlin, editor, *Topics in Cryptology – CT-RSA 2009*, volume 5473, pages 66–83. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.
35. Mario. You could have invented that Bluetooth attack. `https://blog.trailofbits.com/2018/08/01/bluetooth-invalid-curve-points/`, August 2018.
36. Philipp Markert, Daniel V. Bailey, Maximilian Golla, Markus Dürmuth, and Adam J. Aviv. This PIN Can Be Easily Guessed. In *IEEE Symposium on Security and Privacy*, SP '20, San Francisco, California, USA, May 2020. IEEE.
37. John Padgette, John Bahr, Mayank Batra, Marcel Holtmann, Rhonda Smithbey, Lily Chen, and Karen Scarfone. Guide to bluetooth security. Technical Report NIST SP 800-121r2, National Institute of Standards and Technology, Gaithersburg, MD, May 2017.

38. Tomáš Rosa. Bypassing Passkey Authentication in Bluetooth Low Energy. page 3.
39. Anthony Rose and Ben Ramsey. >>> Picking Bluetooth Low Energy Locks from a Quarter Mile Away. page 85.
40. Mike Ryan. Bluetooth: With Low Energy comes Low Security. page 7.
41. Mike Ryan. mikeryan/crackle. `https://github.com/mikeryan/crackle`, December 2019. original-date: 2014-01-27T03:15:35Z.
42. securing. securing/gattacker. `https://github.com/securing/gattacker`, March 2020. original-date: 2016-08-03T10:21:08Z.
43. Chaoshun Zuo, Haohuang Wen, Zhiqiang Lin, and Yinqian Zhang. Automatic Fingerprinting of Vulnerable BLE IoT Devices with Static UUIDs from Mobile Apps. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security - CCS '19*, pages 1469–1483, London, United Kingdom, 2019. ACM Press.