Breaking the Bluetooth Pairing – Fixed Coordinate Invalid Curve Attack

Eli Biham
biham@cs.technion.ac.il

Lior Neumann
lior.neumann@cs.technion.ac.il

Department of Computer Science
Technion – Israel Institute of Technology
Haifa 3200003, Israel

Abstract

Bluetooth is a widely deployed platform for wireless communications between mobile devices. It uses authenticated Elliptic Curve Diffie-Hellman for its key exchange. In this paper we show that the authentication provided by the Bluetooth pairing protocols is insufficient and does not provide the promised MitM protection. We present a new variant of an *Invalid Curve Attack* that preserves the x-coordinate of the public keys. The attack compromises the encryption keys of all of the current Bluetooth authenticated pairing protocols, provided both paired devices are vulnerable. Specifically, it successfully compromises the encryption keys of 50% of the Bluetooth pairing attempts, while in the other 50% the pairing of the victims is terminated. The affected vendors are currently working to patch their products.

1 Introduction

Bluetooth is a wireless communication standard for exchanging data over short distances. The protocol provides confidentiality and access authentication at the link layer. Thanks to its embedded security and flexibility Bluetooth has become one of the most popular communication protocols for mobile devices.

This paper presents a new cryptographic attack on the ECDH protocol and its application to all of the current Bluetooth versions. Our attack provides a new tool for attacking protocols with insufficient MitM authentication as we illustrate on Bluetooth.

As a result of our disclosure, all the major Bluetooth vendors including Qualcomm, Broadcom, Intel and Google have addressed the issue and are currently working together with the Bluetooth Special Interest Group in order to provide a valid mitigation and patch their products. Accordingly, CVE-2018-5383 was assigned for this vulnerability in the Bluetooth protocol.

1.1 Bluetooth Versions

Bluetooth is a set of protocols evolved during many years of development. The two main Bluetooth protocols are *Bluetooth BR/EDR*, commonly used by audio peripheral and old Bluetooth equipment, and *Bluetooth Low Energy*, mainly used by IoT and smart devices.

In this paper we concentrate on the two associated pairing protocols, i.e., *Secure Simple Pairing* (*SSP*) and *LE Secure Connections* (*LE SC*). The SSP protocol is used as part of the original Bluetooth BR/EDR protocol, while the recent LE SC protocol is used by the newer Bluetooth Low Energy protocol. The full list of Bluetooth protocols and sub-protocols is summarized in Appendix A.

1.2 The Elliptic Curve Diffie-Hellman Protocol

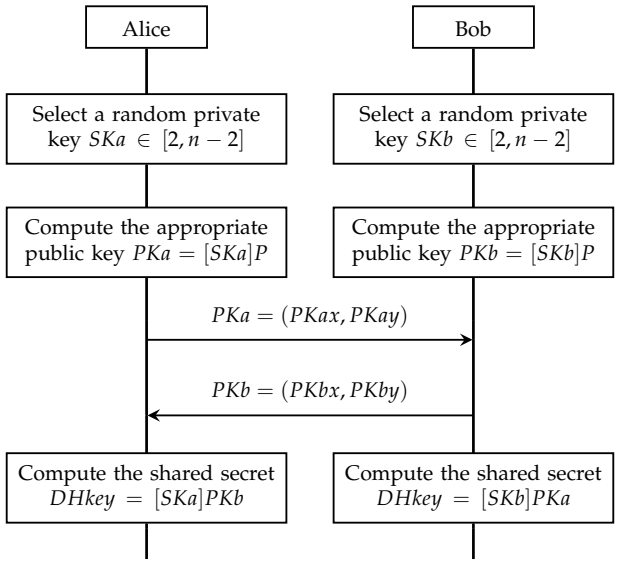
The *Elliptic Curve Diffie-Hellman* (*ECDH*) protocol, introduced in the 1980s by Miller [20] and Koblitz [13], is a variant of the Diffie-Hellman key exchange protocol [3]. It utilizes the algebraic structure of elliptic curves over finite fields in order to exchange cryptographic symmetric keys over a public compromised channel.

The domain parameters of ECDH consist of the order q of the underlying finite field $\mathbb{F}_q$, the equation of the elliptic curve $y^2 = x^3 + ax + b$ and a base point P with a prime order n. Examples of such parameters are the NIST domain parameters specified in the FIPS 186-2 standard [14], which are used by the Bluetooth pairing protocol.

A prerequisite to the protocol is for both parties to agree on the domain parameters $D = \{q, a, b, P, n\}$. Denote $[a]P$ to be the repeated addition of point P to itself a times. The first step of the protocol is key-pair generation, in which each party generates an ECDH key-pair. A key-pair consists of a private scalar SK and a public point PK, such that $PK = [SK]P$. Both parties then send their public point to their correspondent. Finally both parties compute the shared-point by multiplying their private scalar by their correspondent's public point. A diagram of the protocol is outlined in Figure 1. Note that a scalar multiplication over elliptic curves is commutative. This fact ensures that both parties computed the same shared key $[SKa]PKb = [SKa]([SKb]P) = [SKb]([SKa]P) = [SKb]PKa$.

A major advantage of elliptic curve based DH over multiplicative modulus groups based DH is efficiency. The best known attacks against the elliptic curve discrete logarithm problem for a general curve requires $O(\sqrt{n})$ group operations. As a result the same security properties of multiplicative modulus groups can be achieved with much smaller group sizes, which makes ECDH far more space efficient than the original DH protocol. This property is making ECDH better suited for low-bandwidth radio based communication between embedded devices.

Figure 1: Elliptic Curve Diffie-Hellman Protocol Diagram



1.3 Previous Work

The first version of Bluetooth BR/EDR was not intended to be secure against MitM attacks. It used a short PIN code as its primary cryptographic secret. Bluetooth BR/EDR version 2.1 introduced SSP which added authenticated ECDH to achieve eavesdropping and MitM protection. A patent assigned by Peter Landrock and Jan Ulrik Kjaersgaard [12] in 2008 describes how the Invalid Curve Attack could break SSP in cases where appropriate public key validation is not performed. It explains how a malicious attacker can extract the private ECDH key from a poorly implemented platform. The patent proposed multiple mitigations against the attack, which was adopted by the Bluetooth SIG. The protocol designers suggest three optional methods in order to "prevent an attacker from retrieving useful information about the device's private key using invalid public keys", as described in [7, Part H, Section 5.1]. The proposed methods are:

- "Change its private key after three failed attempts from any BD_ADDR and after 10 successful pairings from any BD_ADDR; or after a combination of these such that 3 successful pairings count as one failed pairing".
- "Verify that the received public keys from any BD_ADDR are on the correct curve".
- "Implement elliptic curve point addition and doubling using formulas

that are valid only on the correct curve".

Most implementors chose the first mitigation.

The first version of the Bluetooth Low Energy pairing scheme also did not protect against MitM attacks. In an article called "With Low Energy Comes Low Security" [17] by Mike Ryan, the author pointed out that BTLE "Legacy Pairing" is vulnerable to an eavesdropping attack. His article describes how an attacker with low resources can easily sniff BTLE traffic and decrypt it without any interaction with the victim. His attack relies on the same weakness that was found in the original Bluetooth pairing. Prior to the authentication phase a 6-digit decimal PIN is generated and displayed by the slave. The user then enters this PIN to the master device. The session key is exchanged encrypted using the PIN as a mutual temporary key. Mike Ryan also provided an open-source software [16] that recovers the session key from captured Legacy Pairing traffic within a fraction of a second.

Bluetooth version 4.2 addressed the weaknesses found in the Legacy Pairing protocol by introducing LE Secure Connections, a new pairing protocol based on ECDH. Since the release of Core Specification 4.2 [7] no new security related issues regarding the pairing of Bluetooth Low Energy have been published.

Small-subgroups based attacks were described several times throughout history. Most recently, in 2015 Tibor Jager, Jrg Schwenk and Juraj Somorovsky [11] presented practical Invalid Curve Attack on specific implementations of TLS which indicated that these attacks are still widely effective on modern software.

1.4 Our Results

In [7, Part A, Section 5.2.3], the protocol designers state that "Secure Simple Pairing protects the user from MITM attacks with a goal of offering a 1 in 1,000,000 chance that a MITM could mount a successful attack". Since LE SC is almost identical to SSP, it is safe to assume that the same goal was intended for it as well. We present a new MitM attack on both SSP and LE SC.

Our attack exploits improper validation of ECDH public keys using a new variant of the Invalid Curve Attack. Provided that both paired devices are vulnerable, our attack can compromise 50% of the Bluetooth pairing attempts, while in the rest the pairing fails.

In this paper we present two variations of our attack: semi-passive and fully-active. In both cases, our attack recovers the session encryption key on success, while on failure our attack causes a denial of service. The semi-passive attack requires packet interception and transmission only twice during the ECDH key exchange, and provides success probability of 25%. The fully-active attack generalizes the semi-passive attack by requiring packet interception and transmission throughout the entire connection. The fully-active attack provides better success rate of 50%.

Finally, combining our attack with the already known Bluetooth pairing (Bluetooth BR/EDR) and "Legacy Pairing" (Bluetooth LE) insecurity against MitM, we show that all of the currently available Bluetooth pairing protocols are insecure.

1.5 Structure of the Paper

This paper is organized as follows: in Section 2 we introduce the Bluetooth pairing schemes. Section 3 summarizes the original version of the Invalid Curve Attack. Section 4 is the core of our paper, where we describe our new attack. Section 5 describes the design flaws and suggests possible mitigations. Section 6 presents the platforms that we found to be vulnerable to our attack, and our testing methods. Section 7 discusses the practicality of our attack. Finally, the paper is summarized in Section 8.

2 Bluetooth Pairing

The Bluetooth pairing protocol is the part of the Bluetooth link layer protocol which provides the encryption keys for the rest of the protocol. In this section we review the Bluetooth pairing protocols and their authentication mechanisms.

Throughout this paper SSP and LE SC are interchangeable. We arbitrarily chose to elaborate LE SC, and then discuss the differences from SSP.

2.1 LE Secure Connections

Bluetooth Low Energy has two pairing schemes, "Legacy Pairing" and "LE Secure Connections". Oddly, Legacy Pairing was not intended to be protected against malicious eavesdroppers. The newer LE SC, introduced in Bluetooth version 4.2, promised to solve this problem and provide MitM protection using contemporary cryptographic primitives such as key exchange, commitments and MACs (message authentication codes).

The LE SC pairing scheme has four association models: "Just Works", "Numeric Comparison", "Passkey Entry" and "Out-Of-Band". The association model is chosen according to the IO capabilities of the participating devices. We do not address the "Out-Of-Band" mode in this paper since it requires a vendor specific protocol over a proprietary private channel, which is not fully-defined by [7]. The Just Works mode is equivalent to Numeric Comparison, but without the user interaction, and therefore provides no authentication.

In the next subsections, after some required notations and definitions, we discuss the various phases of the Bluetooth LE SC pairing. The first phase of the pairing is feature exchange. We skip discussing this phase as as it is irrelevant in the context of this paper.

2.1.1 Notations and Definitions

Protocol variables:

- **A, B** (6 Bytes) – The BD_ADDR of each party.
- **IOcapA, IOcapB** (1 Byte) – The advertised IO capabilities of each party (exchanged during the first phase).
- **PKa, PKb** (64 Bytes) – The public key of each party.
- **SKa, SKb** (32 Bytes) – The private key of each party.
- **PKax, PKbx** (32 Bytes) – The x-coordinate of each party's public key.
- **DHKey** (32 Bytes) – The shared Diffie-Hellman secret.
- **Na, Nb** (16 Bytes) – Nonces used for Numeric Comparison association model.
- **Nai, Nbi** (16 Bytes) – Nonces used for Passkey Entry association model.
- **rai, rbi** (1 Bytes) – Represents a single bit of the passkey.

Cryptographic functions (based on AES-CMAC [19]):

- **Function f4** – Commitment Value generation function, defined by: $\text{f4}(U, V, X, Y) = \text{AES-CMAC}_X(U \parallel V \parallel Y)$
- **Function g2** – User Confirm Value generation function, defined by: $g2(U, V, X, Y) = \text{AES-CMAC}_X(U \parallel V \parallel Y) \pmod{2^{32}}$
- **Function f5** – Key Derivation function, defined by:

$$T = \text{AES-CMAC}_{SALT}(DHKey)$$
$$\text{f5}(DHKey, N1, N2, A1, B2) =$$
$$\text{AES-CMAC}_T(0 \parallel \text{`}btle\text{'} \parallel N1 \parallel N2 \parallel A1 \parallel A2 \parallel 256) \parallel$$
$$\text{AES-CMAC}_T(1 \parallel \text{`}btle\text{'} \parallel N1 \parallel N2 \parallel A1 \parallel A2 \parallel 256)$$

  Where SALT is the 128-bit constant value defined in [7].

- **Function f6** – Check Value generation function, defined by:

$$\text{f6}(W, N1, N2, R, IOcap, A1, A2) =$$
$$\text{AES-CMAC}_W(N1 \parallel N2 \parallel R \parallel IOcap \parallel A1 \parallel A2)$$

2.1.2 Phase 2 – ECDH Key Exchange

Both participating devices exchange ECDH public keys using the standard NIST curve P-256 domain parameters. Each party computes the shared secret DHKey as described in Subsection 1.2.

2.1.3 Phase 3 – Authentication (Numeric Comparison)

The Numeric Comparison association model is used when both participating devices are capable of displaying a 6-digit decimal number and at least one of them can accept a "Confirm or Deny" input.

1. Each party selects a random 128-bit nonce Na and Nb.
2. The non-initiator commits (Cb) to Nb and public keys using Function f4, such that $Cb = \text{f4}(PKax, PKbx, Nb, 0)$.
3. Both parties reveal their nonces, first the initiator and then the non-initiator.
4. The initiator validates the commitment.
5. Both sides display the six least significant decimal digits of the User Confirm Value (Va and Vb), which are computed by $g2(PKax, PKbx, Na, Nb)$.
6. The user compares the values and confirms or denies accordingly.

Note that the y-coordinates of the public keys are not authenticated during this phase. This observation will be used later.

The third phase using Numeric Comparison is outlined in Figure 2, taken from [7].

2.1.4 Phase 3 – Authentication (Passkey Entry)

The Passkey Entry association model is used when at least one of the participating devices is capable of receiving numeric input from the user while the other is capable of displaying a six-digit number.

1. The Passkey is generated and displayed on one device, and the user then enters it into the other.
2. Each party selects a random nonce Na1 and Nb1 (128-bit).
3. Each party commits to their nonce, public keys and the first bit (ra1 and rb1) of the Passkey using Function f4, first the initiator ($\text{f4}(PKax, PKbx, Na1, ra1)$), and then the non-initiator ($\text{f4}(PKbx, PKax, Nb1, rb1)$).
4. Both parties reveal their nonces and validate the commitments, first the initiator and then the non-initiator.
5. Steps 2–4 are repeated 20 times, once per each Passkey bit.

Similarly to Section 2.1.3 the y-coordinates are not authenticated.

The third phase using Passkey Entry is outlined in Figure 3, taken from [7].

Figure 2: Phase 3 – Authentication (Numeric Comparison)

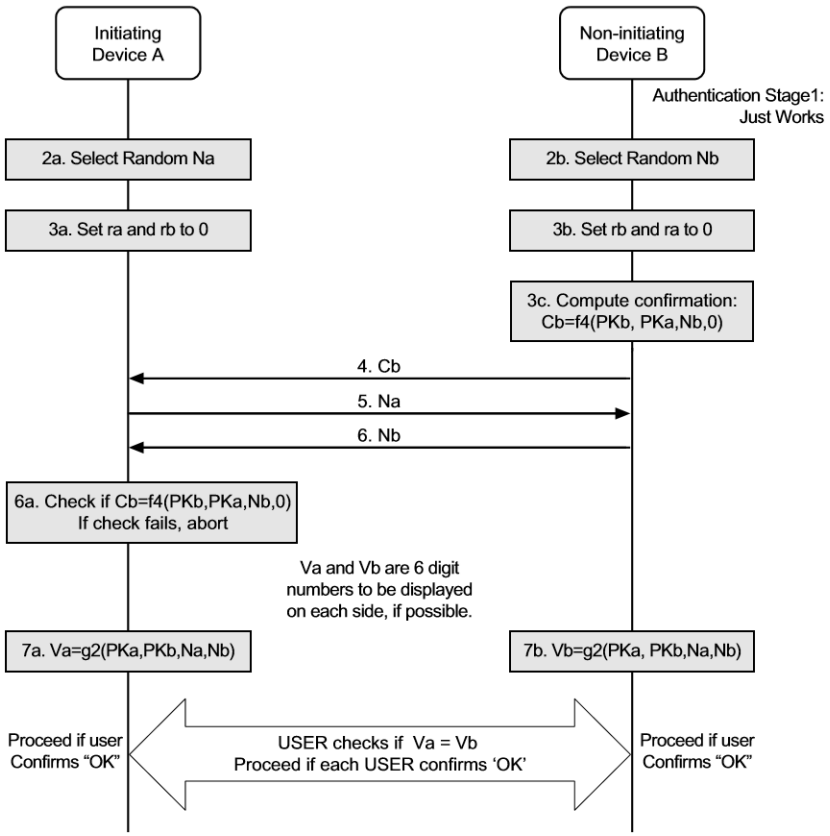


Figure 3: Phase 3 – Authentication (Passkey Entry)

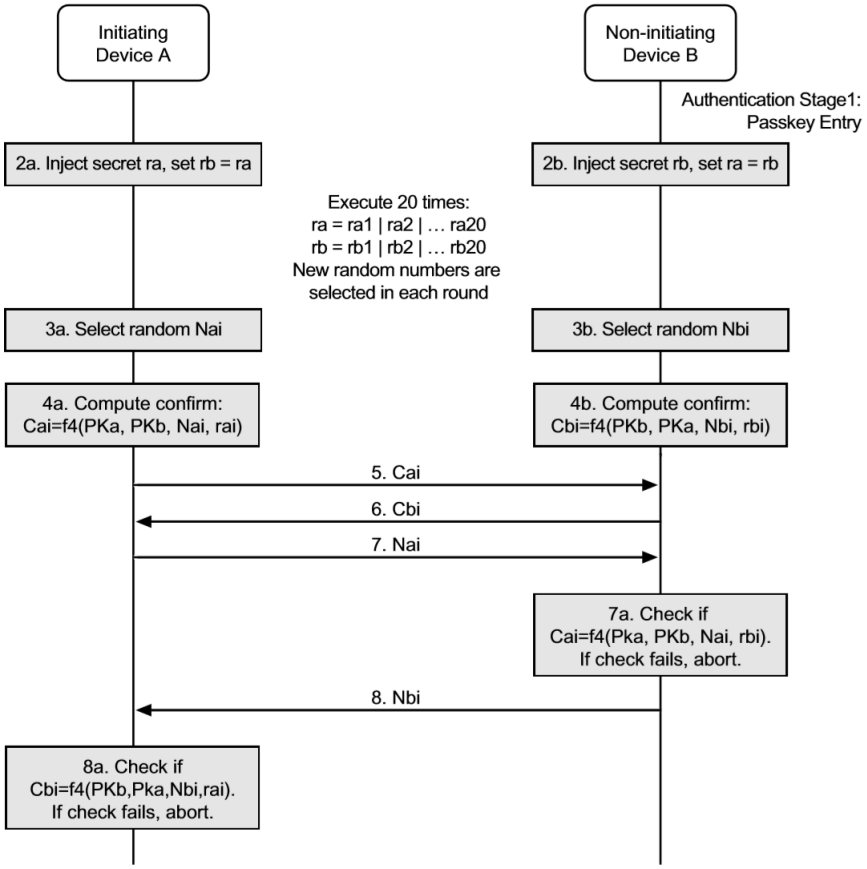


Table 1: Function f6 Inputs

	Numeric Comparison	Passkey Entry
Ea	f6$(MacKey, Na, Nb, 0, IOcapA, A, B)$	f6$(MacKey, Na20, Nb20, rb, IOcapA, A, B)$
Eb	f6$(MacKey, Nb, Na, 0, IOcapB, B, A)$	f6$(MacKey, Nb20, Na20, ra, IOcapB, B, A)$

2.1.5 Phase 4 – Session Key Derivation and Validation

The fourth phase of the pairing is responsible for session key derivation.

1. Both parties derive the session keys (MacKey and LTK) from the $DHKey$ using f5$(DHKey, Na, Nb, A, B)$.
2. Each device computes its Check Value (Ea and Eb) using Function f6. The inputs of Function f6 are listed in Table 1.
3. The initiator sends his Check Value to the non-initiator, which responds in turn with his Check Value.
4. Each side validates its correspondent's Check Value.

The final phase is outlined in Figure 4, also taken from [7].

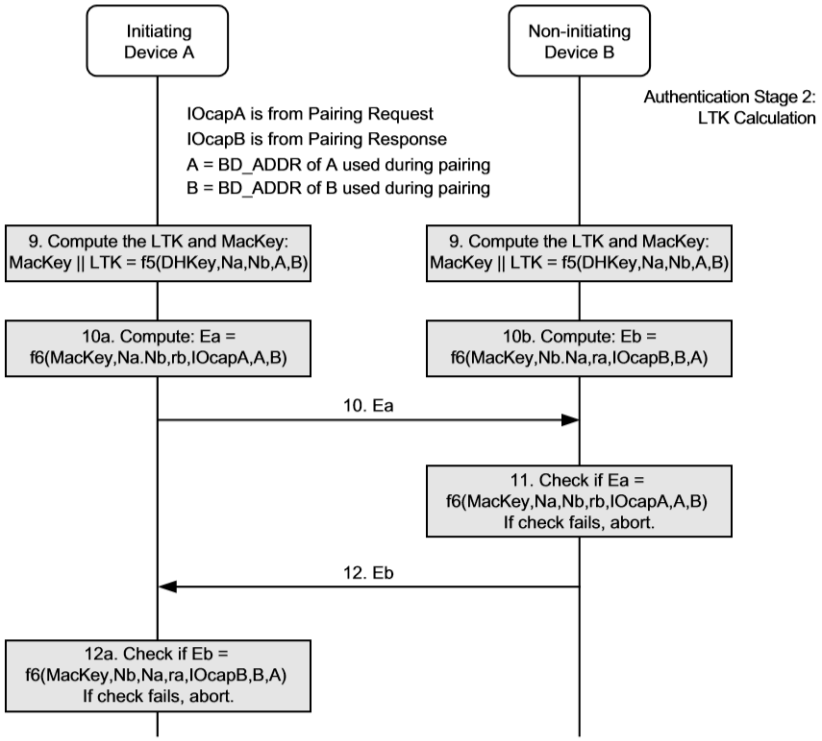
2.2 Secure Simple Pairing (SSP)

Secure Simple Pairing was the inspiration behind LE SC. There are two main differences between the two protocols. The first difference is the support of both P-192 and P-256 curves for the key exchange in SSP. The second difference is that SSP uses SHA-256 based HMAC functions, while LE SC uses AES-CMAC based functions. Other than those differences, SSP and LE SC are almost identical. Both of them have the same stages with similar functions and inputs. Due to these similarities they are interchangeable for the sake of this paper, and every statement regarding the security of one of them applies to the other.

3 Invalid Curve Attack

The Invalid Curve Attack, presented by Biehl et al. [2] and further described by Antipa et al. [1], exploits implementations of ECDH which improperly validate the received public keys. The invalid curve attack belongs to a larger family of attacks named *Small subgroup key recovery attacks*. This group of attacks utilizes small subgroups of finite groups in order to extract non-ephemeral secret information.

Figure 4: Phase 4 – Session Key Derivation and Validation



3.1 Background

Scalar multiplications over elliptic curves consist of repeating two basic operations, point-addition and point-doubling. Given domain parameters $D = \{q, a, b, P, n\}$, the group operations are defined as follows:

- **Point Addition** – Consider the points $P = (Px, Py)$ and $Q = (Qx, Qy)$ such that $P \neq Q$. The point addition $R = P + Q$, where $P \neq -Q$, is defined by drawing the line which intersects with both P and Q. This line also intersects with a third point on the curve. The sum point R is the reflection of this point across the x-axis. This computation is expressed by the following formulae:

$$\begin{aligned} s &\equiv (Py - Qy)(Px - Qx)^{-1} \pmod{q} \\ Rx &\equiv s^2 - Px - Qx \pmod{q} \\ Ry &\equiv Py - s(Rx - Px) \pmod{q} \end{aligned}$$

  If $P = -Q$ the result is the identity element $\infty \in E$.

  It can be seen that these formulae do not involve the curve parameters a and b.

- **Point Doubling** – Consider the point $P = (Px, Py) \in E$. Adding a point to itself $R = P + P = [2]P$, where $Py \not\equiv 0 \pmod{q}$, is defined by drawing the tangent line of the curve at point P. This line intersects with a second point on the curve. The sum point R is computed by reflecting this point across the x-axis. This computation is expressed by the following formulae:

$$\begin{aligned} s &\equiv (3Px^2 + a)(2Py)^{-1} \pmod{q} \\ Rx &\equiv s^2 - 2Px \pmod{q} \\ Ry &\equiv Py - s(Rx - Px) \pmod{q} \end{aligned}$$

  If $Py \equiv 0 \pmod{q}$ the result is the identity element $\infty \in E$.

  It can be seen that these formulae do not involve the curve parameter b.

3.2 Private Key Retrieval

Let E' be a different group with the curve equation $y^2 = x^3 + ax + b'$, such that there exists a point $Q_1 \in E'$ with a small prime order p_1. The attacker provides Q_1 as his ECDH public-key. Denote SK as the private key of the victim. The victim then calculates $x = [SK]Q_1$ and send $H(x)$, where H is a publicly known one-way function. The attacker then exhaustively searches the value of x. This is computationally feasible, since the order of the Q_1 is low. The resultant discrete log a_1 provides the information $x = [a_1]Q_1 = [SK]Q_1$, therefore $SK \equiv a_1 \pmod{p_1}$. The attacker then repeats this process using a different point Q_i, which has a different small prime order p_i. This

exchange repeats until the product of the primes satisfies $\prod_{i=1}^{k} p_i > n$. Finally, the attacker recovers the victim's private key using the Chinese-Remainder-Theorem.

4 Our attack

In this section we introduce the Fixed Coordinate Invalid Curve Attack, a new variant of the Invalid Curve Attack in which we exploit the ability to forge low order ECDH public keys that preserves the x-coordinate of the original public-keys. Our attack is based on the observations that only the x-coordinate of each party is authenticated during the Bluetooth pairing protocol and on the fact that the protocol does not require its implementations to validate whether a given public-key satisfies the curve equation.[1] Our new attack can be applied to both SSP and LE SC.

As opposed to the classical Invalid Curve Attack our attack belongs to the family of attacks named *Small subgroup confinement attack*. In this family the attacker attempts to compromise the shared secret by forcing a key to be confined to an unexpectedly small subgroup.

4.1 Semi-Passive Attack

The Invalid Curve Attack exploits the fact that given an elliptic curve in Weierstrass notation $y^2 = x^3 + ax + b$, both point-doubling and point-addition operations are independent of b. Given an elliptic curve group E and a point $Q = (Qx, Qy), Q \in E$, let $Q' = (x, 0)$ be its projection on the x-axis. We can easily find a different curve E' over the equation $y^2 = x^3 + ax + b'$, such that $P' \in E'$, using the same curve parameter a and a different parameter b'. According to the group definition, given a point $Q = (Qx, Qy)$, the inverse of Q is found by reflecting it across the x-axis $Q^{-1} = (Qx, -Qy)$. Therefore, every point of the form $(Qx, 0)$ equals its own inverse, thus has order two. An example of such a point is outlined in Figure 5. Since the y-coordinate is forced to zero on the given point, the new curve parameter b' is calculated using the equation $b' \equiv -x^3 - ax \pmod{q}$.

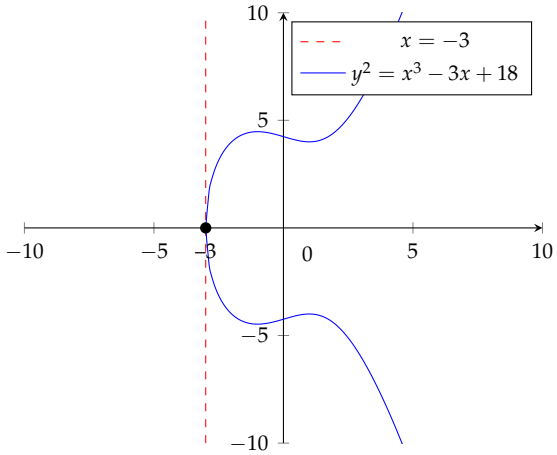
Using this observation we describe the Fixed Coordinate Invalid Curve Attack. In our attack, the attacker confines each of the public keys to a subgroup of order two by projecting it on the x-axis. This confinement implies that the attack succeeds only if both private keys SKa and SKb are even. In this case, which occurs with probability 0.25, we get $DHKey_a = DHKey_b = \infty$. This manipulation is not detected since the x-coordinates are left unchanged.

The semi-passive attack is thus as follows:

1. Eavesdrop both parties throughout the pairing protocol.
2. Let both parties perform the first phase of the pairing (feature exchange).

[1] Note that all of the implementations we tested did not add this validation voluntarily.

Figure 5: Example of an elliptic curve with an order-two point at $(-3, 0)$



3. Let the parties transmit their ECDH public keys.
4. Modify the y-coordinate of both public keys to zero during the transmission.
5. Do not intervene with rest the of the pairing protocol.
6. Observe if the pairing succeeded, otherwise, quit.
7. Derive the symmetric session keys (LTK and MacKey) using the expected $DHKey = \infty$ and the public parameters.
8. After the pairing is finished, forge or passively decrypt packets sent between the participating devices using the derived keys.

The message interception during the second phase is illustrated in Figure 6. After the interception of the second phase the rest of the communication can be eavesdropped without further interaction, as outlined in Figure 7.

4.2 Fully-Active MitM Attack

The attack described in Subsection 4.1 requires a message interception only during the second phase of the pairing. The rest of the communication can be passively eavesdropped. We can further improve the attack success probability to 0.5 by also intercepting messages during the fourth phase. In order to achieve this better success rate we should consider the four possible values of $DHKey_x$ computed by each party after the interception of the second phase, as listed in Table 2. The first row in the table represents a successful attack as in Subsection 4.1.

Figure 6: Fixed Coordinate Invalid Curve Attack – Phase 2

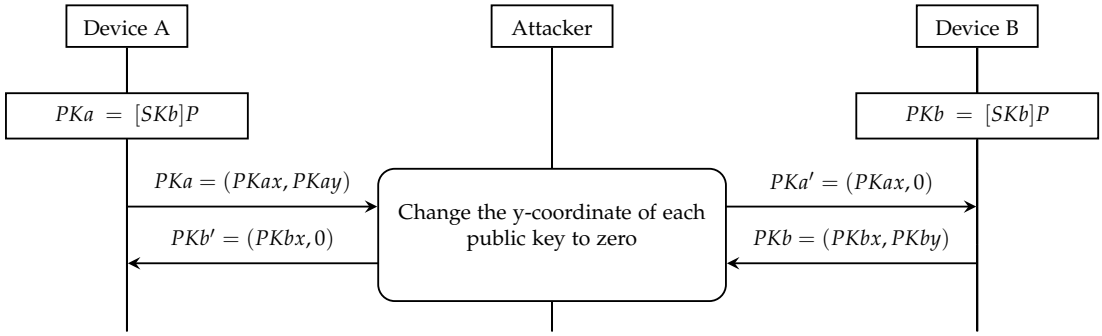


Figure 7: Semi-Passive MitM Attack – Passive Eavesdropping

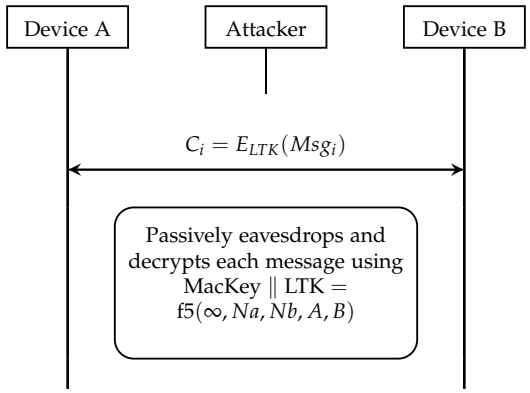


Table 2: The possible values for $DHKey_a$ and $DHKey_b$

$DHKey_a$	$DHKey_b$
∞	∞
∞	PKa'
PKb'	∞
PKb'	PKa'

During the fourth phase (see Subsection 2.1.5) both devices derive the session keys, and exchange check values to validate their correspondent's session key. Recall that the session key is derived using the shared $DHKey$, the nonce Na, the nonce Nb, the address of device A, the address device B, ra and rb. At the beginning of the fourth phase all of the values are publicly known, except for the $DHKey$, ra and rb. The values ra and rb differ between Passkey Entry and Numeric Comparison. In Numeric Comparison $ra = rb = 0$, while in Passkey Entry these are 6-digit decimal values expressed as 128-bit integers. The passkey is not known to the attacker, but could still be retrieved with a small effort. The attacker could retrieve the passkey by exhaustively searching through the $10^6 \approx 2^{20}$ possible values. An even better way for extracting the passkey is by iterating through all possible values for rai and rbi, used during the third phase. The inputs of Function f4, $PKax$, $PKbx$, Nai and Nbi, are publicly known at the fourth phase. The remaining values rai and rbi represent a single bit of the passkey each, and are left concealed. The attacker can iterate through all $2 \cdot 20 = 40$ options to determine the correct values of ra and rb.

During the first message of the fourth phase, the initiator reveals information about his $DHKey_a$ by sending his check-value (Ea) computed using Function f6. The attacker uses this information to determine $DHKey_a$ out of the two possible values, PKb' or ∞. If $DHKey_a = \infty$, the attacker continues as in semi-passive attack, without further interception.

If $DHKey_a = PKb'$, the attacker guesses the yet unknown value of $DHKey_b$ to be either ∞ or PKa', and calculates the appropriate check-value accordingly. If the guess was incorrect, the non-initiator would reply with the message "Pairing Failed (Check Value Failed)". Otherwise, the non-initiator replies with his check-value. The fully-active attack is thus as follows:

1. Apply steps 1–3 as in the semi-passive attack.
2. Do not intervene with the third phase of the pairing protocol.
3. If the association model is Passkey Entry, find the correct value of ra and rb using one of the methods described above, otherwise, assume $ra = rb = 0$.
4. In the fourth phase of the pairing, receive the check-value (Ea), but distort it so it will not reach its destination (i.e. by destroying the checksum or preamble).
5. Use the check-value in order to validate whether $DHKey_a$ equals ∞ or PKb'.
   (a) If the $DHKey_a = \infty$, send the original check-value, and continue as with the semi-passive attack.
   (b) If $DHKey_a = PKb'$, randomly guess the value of $DHKey_b$ to be either ∞ or PKa'.
6. Compute and transmit the value of Ea' according to the guess of $DHKey_b$, instead of the distorted value from step 4.

Table 3: Success Rate – Semi-Passive Attack

$DHKey_a$ \ $DHKey_b$	∞	PKa'
∞	Success	Failure
PKb'	Failure	Failure

Total Semi-Passive Attack: **25%**

7. Compute the session keys $LTK_a, MacKey_a$ and $LTK_b, MacKey_b$ associated with $DHKey_a$ and $DHKey_b$.
8. If the guess is incorrect, the pairing protocol is terminated with a "Pairing Failed (Check Value Failed)" message.
9. Otherwise, act as a relay or forge packets sent between the participating devices by decrypting and re-encrypting each message.

The message interception during the fourth phase considering $DHKey_a = PKb'$ is illustrated in Figure 8. Note that using this improvement the attacker has to continually act as MitM for the rest of the session. Each message sent between the victim devices has to be intercepted by the attacker, decrypted using the sender's key, re-encrypted using the recipient's key, and sent to the recipient. The relay operation is illustrated in Figure 9.

4.3 Success Rate

The success rate of our attack is calculated under the assumption that the private keys SK_x are chosen uniformly at random. For the semi-passive attack suppose that the shared keys computed by both parties are the point-at-infinity, as described in Subsection 4.1. Denote this event by $V = (DHKey_a = \infty) \cap (DHKey_b = \infty)$. The probability of this event is $\Pr(V) = 25\%$.

For the fully-active attack suppose that the shared key as derived by the initiator is PKb', and that the attacker correctly guesses the shared key as derived by non-initiator. Denote this event by $U = (DHKey_a = PKb') \cap (DHKey'_b = DHKey_b)$. The probability of this event is $\Pr(U) = 25\%$. Consequentially the success probability of the fully-active attack is $\Pr(U \cup V) = 25\% + 25\% = 50\%$.

The success probabilities of the various cases of our attack are summarized in Tables 3, 4 and 5.

5 Design Flaws and Mitigations

We found that there were multiple flaws in the design of the pairing protocol. In this section we discuss these flaws, and point to which mitigations should

Figure 8: Fully-Active MitM (Considering $DHKey_a = PKb'$) – Phase 4

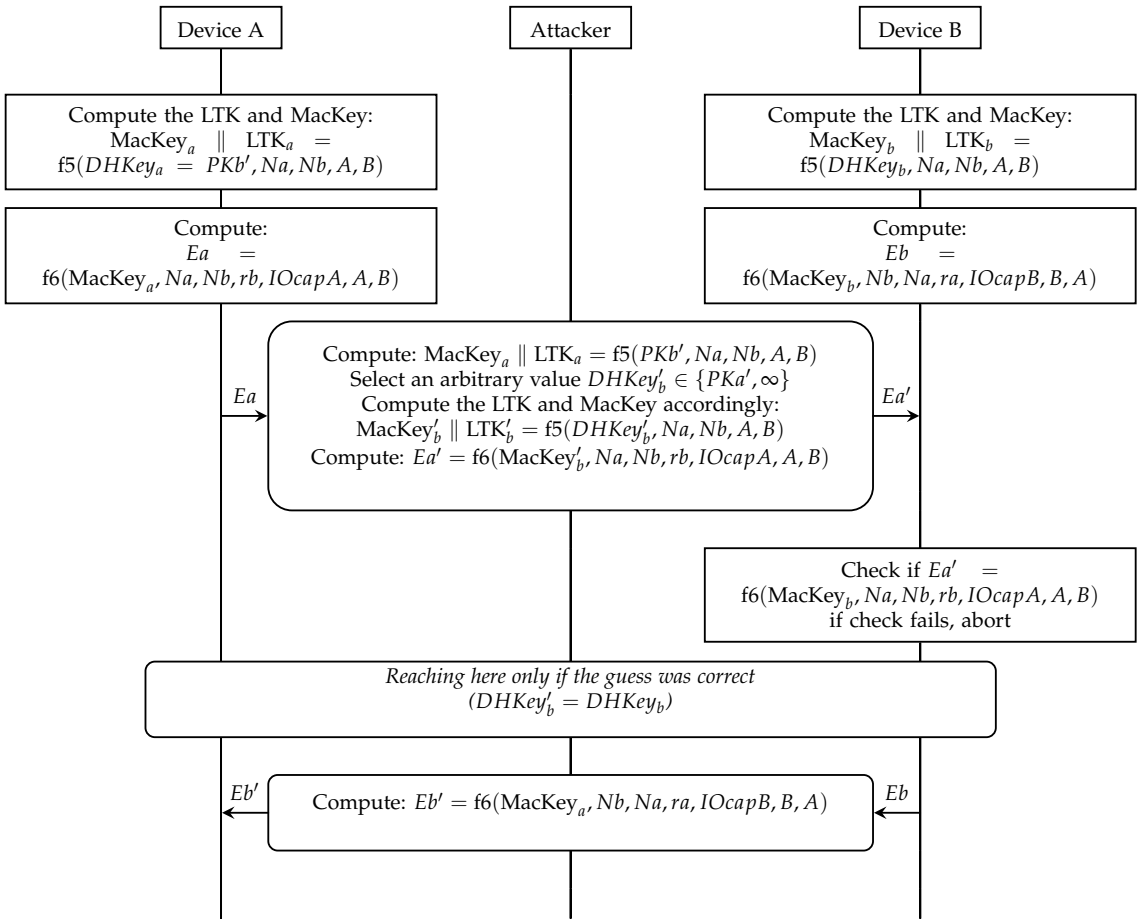


Table 4: Success Rate – Fully-Active Attack (when guessing $DHKey'_b = \infty$)

$DHKey_a$ \ $DHKey_b$	∞	PKa'
∞	Success	Failure
PKb'	Success	Failure

Total Fully-Active Attack: **50%**

Figure 9: Fully-Active MitM – Relay

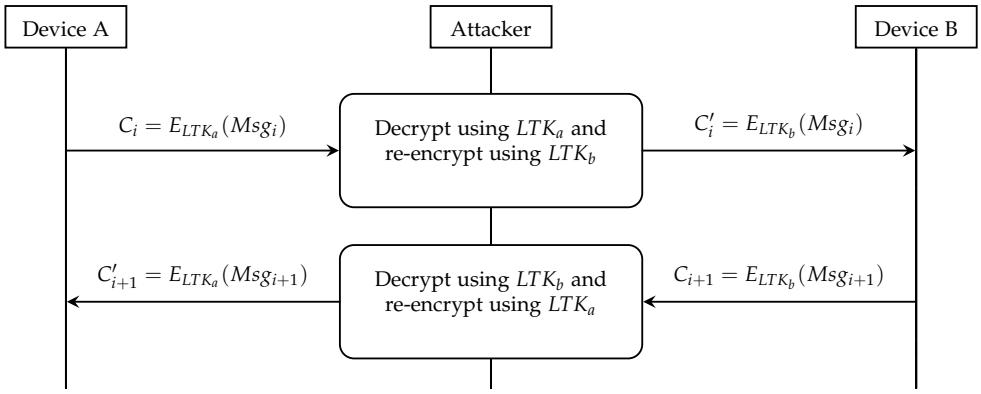


Table 5: Success Rate – Fully-Active Attack (when guessing $DHKey'_b = PKa'$)

$DHKey_a$ \ $DHKey_b$	∞	PKa'
∞	Success	Failure
PKb'	Failure	Success

Total Fully-Active Attack: **50%**

be applied in order to protect a platform. We also warn from mitigations which are insufficient to protect against our attack.

5.1 Design Flaws

We would like to point out two major design flaws that make our attack possible. The first design flaw is sending both the x-coordinate and the y-coordinate during the public key exchange. This is unnecessary and highly inadvisable, since it greatly increases the attack surface, while calculating the y-coordinate from a given x-coordinate is simple.

The second major flaw is that although both coordinates of the public keys are sent during the second phase of the pairing, the protocol authenticates only the x-coordinate. We are not aware of any reason why the designers decided to leave the y-coordinate unauthenticated, other than for saving a tiny computational effort. Even though the point validity should be checked by the implementation, our attack could have also been avoided if both coordinates were authenticated.

Another less significant flaw is that in [7, Part H, Section 5.1] the protocol designers state that "To protect a device's private key, a device should implement a method to prevent an attacker from retrieving useful information about the device's private key using invalid public keys. For this purpose, a device can use one of the following methods". In this quote, the specification uses the term "should" (as opposed to "must"). Therefore, implementors may skip the instruction as it is not mandatory for compliance with the specification.

5.2 Mitigations

The obvious (and recommended) mitigation against our attack is to test whether the given ECDH public-key satisfies the curve equation.

Assuming that the order n of the base point P is prime, another acceptable mitigation is to validate whether the given public key (PK) satisfies $[n]PK = \infty$. This mitigation applies since it verifies the order of the given key, thus preventing public keys with smaller orders.

These mitigations are applicable to both LE SC and SSP.

5.3 Insufficient Mitigation

The commonly used mitigation proposed by the Bluetooth Core Specification [7] against Invalid Curve Attack is refreshing the ECDH key pair every pairing attempt. We stress that our attack applies even when this mitigation is applied.

We must also note that although checking whether the y-coordinate equals zero protects against our specific attack, it is not a sufficient mitigation. An extended version of our attack can still be devised using slightly higher order

points with a decreased success probability. For example, given the same domain parameters $D = \{q, a, b, P, n\}$ and a valid public key $PK = (PKx, PKy)$, we can easily forge a public key $PK' = (PKx, PKy')$ with order four, such that $PKy' \neq 0$. Let $Q = (Qx, Qy)$ be the point defined by $Q = [2]PK'$. Such PK' point with order four can be generated by forcing $Qy = 0$. Substituting (PKx, PKy') into the point doubling equations we get

$$\begin{aligned} s &\equiv (3PKx^2 + a)(2PKy')^{-1} \pmod{q} \\ Qx &\equiv s^2 - 2PKx \pmod{q} \\ Qy &\equiv PKy' - s(Qx - PKx) \equiv 0 \pmod{q} \end{aligned}$$

from which we can deduce

$$\begin{aligned} PKy' &\equiv (3PKx^2 + a)(2PKy')^{-1}(Qx - PKx) \pmod{q} \\ 2PKy'^2 &\equiv (3PKx^2 + a)(Qx - PKx) \pmod{q} \\ PKy' &\equiv \pm\sqrt{(3PKx^2 + a)(Qx - PKx)2^{-1}} \pmod{q} \end{aligned}$$

Ignoring the negligible chance that $3PKx^2 + a \equiv 0 \pmod{q}$, we can assign any value Qx that satisfies $Qx \neq PKx$ for which $(3PKx^2 + a)(Qx - PKx)2^{-1}$ is a quadratic-residue modulo q. This generalized form of our attack provides success rate of 25%, rather than 50%. Therefore, checking whether the y-coordinate equals zero is not a sufficient mitigation.

6 Vulnerable Platforms

We have tested the vulnerability of multiple devices from several manufacturers and found that our attack is applicable to most currently available Bluetooth devices. Every vulnerable implementation we discovered can be used in both master and slave roles, depending on the configuration. For the sake of simplicity our tests used CY5677 programmable USB Bluetooth dongle by Cypress with modified dedicated firmware. The following modifications were added to the dongle's Bluetooth stack:

1. We zeroize the y-coordinate of our public key just before transmission.
2. We zeroize the y-coordinate of the remote public key immediately after it is received.

We then examined the target responses, measured the pairing success rate and compared it to our expectations.

We used this "synthetic " validation method since it provides the most realistic setting while giving us enough flexibility to precisely test our attack. We stress that this validation proves that the target device is vulnerable since it performs exactly the same manipulations as a MitM implementation of the attack. An in depth discussion of over-the-air MitM is presented in Section 7.

6.1 Vulnerable Bluetooth Low Energy Platforms

Since LE SC is implemented in the host, the vulnerability is found in the host's operating system, regardless of the Bluetooth adapter. We found the Android Bluetooth stack, "Bluedroid", to be vulnerable[2]. On the other hand, Microsoft Windows does not support LE SC.

6.2 Vulnerable Bluetooth BR/EDR Platforms

The vulnerability in SSP depends on the Bluetooth chip firmware implementation, since unlike LE SC, the key exchange is performed by the chip, rather than by the host. During our research we found that devices of most major chip vendors are affected. In particular, Qualcomm's, Broadcom's and Intel's implementations[3] are vulnerable, which together constitutes most of the Bluetooth chips market. We stress that every device (e.g., mobile phone, laptop or car) that uses such a chip is vulnerable.

7 Practical Considerations

In the following section we consider the requirements and obstacles facing attackers who wish to implement our attack in real-life environment.

7.1 Over the Air Implementation

Our attack requires the ability to manipulate the Bluetooth communication between two victim devices. In a wireless environment this is usually done by jamming the receiver while eavesdropping to the sender, before transmitting the modified message. Due to physical obstacles (such as frequency hopping and different clock synchronizations) Bluetooth eavesdropping and traffic interceptions are much more difficult than in other radio based protocols (such as 802.11). Until recent years, in order to sniff Bluetooth traffic one had to purchase expensive equipment such as the "Ellisys Bluetooth Explorer", which costs in the range of 10K–20K US dollars.

In 2010 Michal Ossman introduced an open-source project called "Project Ubertooth" [15], which introduced a highly accessible and inexpensive utility for Bluetooth sniffing, that costs about $100. Moreover, Mike Ryan suggested in [17] how an attacker can break the frequency hopping parameters used in Bluetooth Low Energy, allowing to easily sniff Bluetooth traffic using Ubertooth. A similar technique is used in Ubertooth in order to sniff Bluetooth Basic Rate traffic, but not yet intercept. It is therefore clear that frequency hopping is not a valid security mitigation against MitM.

[2]Tested on Nexus 5X devices with Android version 8.1.

[3]The examined Bluetooth adapters were: Qualcomm's QCA6174A, Broadcom's BCM4358 and Intel's 8265.

After the success of Ubertooth a lot of new products were introduced improving the capabilities provided by Ubertooth and adding the ability to perform over the air MitM to Bluetooth LE traffic. One of the most popular solutions is "GATTack" [18] presented during Blackhat USA 2016. This tool is a software based framework providing over the air BLE MitM capabilities with equipment that costs about $10. Unfortunately, all of the solutions we found are limited to Bluetooth 4.0 and do not support Bluetooth 4.2 due to its larger packet size. It is safe to assume that the next generation of these products will support Bluetooth 4.2 as it becomes more popular.

7.2 Public Key Manipulation

In addition to the straight-forward MitM technique, there is a much simpler technique for attacking SSP. The SSP packet size is limited to 16-bytes, while the ECDH public-key is either $192 \cdot 2 = 384\text{bits} = 48\text{bytes}$ or $256 \cdot 2 = 512\text{bits} = 64\text{bytes}$. To accompany this restriction the public key is divided into either 3 or 4 packets, allowing the attacker to "override" the packet of the y-coordinate, without affecting the x-coordinate.

It is interesting to note that the P-192 variant is actually harder to override than the P-256 variant, as the second packet in the P-192 variant contains data of both coordinates (x and y), while the P-256 variant separates the coordinates to different packets.

The maximal packet size in LE SC is much larger, limited by 256 bytes instead of 16, thus both public key coordinates could be encoded in a single packet.

7.3 Encoding of the Point-at-Infinity

In the description of our attack we repeatedly consider the point-at-infinity (aka. the identity element ∞) as a possible solution for the scalar multiplication. This requires a delicate discussion since the encoding of this point is not defined by the Bluetooth specification nor there is any standard encoding for it. The encoding of this point is important for ensuring that when both participating devices compute the shared secret to be the point-at-infinity, they actually compute the same key. However, during our tests we found that all the popular implementations represent the point-at-infinity as (0, 0).

8 Summary

In this paper we introduced the Fixed Coordinate Invalid Curve Attack which provides a new tool for attacking the ECDH protocols, and presented the application of our new attack to the Bluetooth pairing protocol.

During our research we discovered multiple design flaws in the Bluetooth specification. We proceeded our research by testing different Bluetooth im-

plementations and found that a large majority of the Bluetooth devices are vulnerable.

We highly recommend the Bluetooth Special Interest Group to fix the design flaws in future Bluetooth versions. The specification must require manufacturers to add the necessary validations, rather than having it optional. We also urge the manufacturers to add the proposed validations, even if it is not mandatory. Although we expect that most of the currently used Bluetooth peripherals will never be patched, patching all the mobile-phones and computers (where a software update is relatively easy) will greatly decrease the risk of this vulnerability.

Finally, shortly after we discovered the attack we informed the affected vendors through the Cert Coordination Center (CERT/CC). As a result, CVE-2018-5383 was assigned to this vulnerability in the Bluetooth protocol. Meanwhile, Google, Broadcom, Qualcomm and Intel have responded with the intention to roll out a patch addressing this vulnerability in the near future. The Bluetooth SIG had also been informed and is currently investigating the issue.

References

[1] Antipa A., Brown D.R.L., Menezes A., Struik R., and Vanstone S.A. Validation of elliptic curve public keys. *Public Key Cryptography PKC 2003*, 2567:211–223, 2003.

[2] Ingrid Biehl, Bernd Meyer, and Volker Mller. Differential fault attacks on elliptic curve cryptosystems. *Advances in Cryptology CRYPTO 2000*, 1880:131–146, 2000.

[3] Whitfield Diffie and Martin E. Hellman. New directions in cryptography. *Transactions on Infromation Theory*, IT-22, NO. 6:644–654, 1976.

[4] Bluetooth Special Interest Group. Specification of the bluetooth system v2.0. 0, 2004.

[5] Bluetooth Special Interest Group. Specification of the bluetooth system v3.0. 0, 2009.

[6] Bluetooth Special Interest Group. Specification of the bluetooth system v4.0. 0, 2010.

[7] Bluetooth Special Interest Group. Specification of the bluetooth system v4.2. 0, 2014.

[8] Bluetooth Special Interest Group. Specification of the bluetooth system v5.0. 0, 2016.

[9] IEEE. Specification of the bluetooth system v1.0b. 1, 1999.

[10] IEEE. Specification of the bluetooth system v1.1. 1, 2001.

[11] Tibor Jager, Jrg Schwenk, and Juraj Somorovsky. Practical invalid curve attacks on tls-ecdh. *Computer Security – ESORICS 2015*, 1880:407–425, 2000.

[12] Peter Landrock and Jan Ulrik Kjaersgaard. Protecting against security attack. 2013. US Patent 8077866 B2.

[13] Koblitz N. Elliptic curve cryptosystems. *Mathematics of Computation*, 48:203–209, 1987.

[14] National Institute of Standards and Technology. Federal information processing standards publication 186-2. 2000.

[15] Michael Ossmann. Project ubertooth. *http://ubertooth.sourceforge.net*.

[16] Mike Ryan. Crackle cracks ble encryption. *https://github.com/mikeryan/crackle*.

[17] Mike Ryan. With low energy comes low security. *USENIX WOOT*, pages 4–4, 2013.

[18] securing. Gattack. *http://gattack.io*.

[19] JH. Song, R. Poovendran, J. Lee, and T. Iwata. The AES-CMAC Algorithm. (4493):1–20, June 2006.

[20] Miller V.S. Use of elliptic curves in cryptography. *Advances in Cryptology CRYPTO 85 Proceedings*, 218:417–426, 1986.

A Bluetooth Versions

Bluetooth has several versions. Each new version introduces extended capabilities or a complete new set of sub-protocols.

The initial releases of Bluetooth, versions 1.0 and 1.0B [9], had many problems, and manufacturers had difficulty making their products interoperable. The manufacturers included a mandatory Bluetooth hardware device address (BD_ADDR) for transmission in the connecting process, which made anonymity impossible at the protocol level. This was a major setback for certain services planned for use in Bluetooth environments.

Bluetooth versions 1.1 [10] introduced major improvements over their predecessors and addressed many of the errors found in v1.0B. New features were added, among them: RSSI for measurement of the power present in a received radio signal, faster connection, faster discovery, adaptive frequency-hopping and higher transmission speeds.

Version 2.0 [4] was released in 2004. It introduced an even faster data transfer with throughput of up to 3 Mbit/s. The throughput enhancement was

due to the use of GFSK and PSK modulation. This new method of modulation is called *EDR*, or *Enhanced Data Rate*, while the older modulation is called *BR*, or *Basic Rate*. When both of the modulations are implemented together it is called *BR/EDR*.

Version 2.1 of the protocol added secured pairing named *Secure Simple Pairing (SSP)* to support *Man-in-the-Middle (MitM)* protection using authenticated Diffie-Hellman during the pairing stage.

Bluetooth 3.0 [5] introduced the support for an alternative MAC/PHY (AMP). AMP is a new feature, allowing the use of an alternative data channel. While the negotiation and establishment are still performed similarly to former versions, the data flow uses an alternative MAC PHY 802.11 (typically associated with Wi-Fi). The 802.11 standard defines different protocols for the physical layer and for the link layer. It is characterized by a high transfer-rate and a relatively high signal range. After the connection is established the 802.11 link encapsulates the data packets of the BT established connection. The result is a much higher transfer rate of up to 24 Mbit/s. This new feature was intended to allow streaming over Bluetooth, whose throughput was still poor compared to other protocols.

Bluetooth Core Specification version 4.0 [6] introduced a new modulation mode and link layer packet format called *Bluetooth Low Energy (BTLE)*. BTLE is intended for use in low power embedded devices. It was rapidly adopted by various consumer devices, such as smart phones, wearable technology, sports tracking devices and recently even health and medical equipment. BTLE PHY divides the RF spectrum into 40 channels, each of which is 2MHz in width, from 2402MHz to 2482MHz. Three of those 40 channels are labeled as advertising channels used for pairing and discovery packets. The rest are labeled as data channels, used for establishing connections and transmission of the data. The link layer was also redesigned and a new pairing protocol was added.

On December 2014, core specification 4.2 [7] was introduced, providing several new features to the BTLE protocol intended to make it the main protocol for the IoT (Internet of Things). These features include a new LE Secure Connections mode, as well as several security and privacy related features.

The latest version of Bluetooth, released on December 2016 was version 5.0 [8]. The new version added several performance features for Bluetooth Low Energy, most of them in the physical layer of the protocol. Among the new features were extended range, higher throughput and higher advertisement capacity.

In this paper we study the pairing protocols SSP used by Bluetooth BR/EDR and LE Secure Connections used by Bluetooth Low Energy. These are the only secure pairing protocols to date.