Breaking the Bluetooth Pairing – Fixed Coordinate Invalid Curve Attack

Eli Biham   Lior Neumann

Department of Computer Science
Technion – Israel Institute of Technology

Cryptoday 2018

Overview

- Bluetooth is a widely deployed platform for wireless communication between mobile devices.
- Examples:
  - Mobile computers – mobile-phones and laptops.
  - Computer peripherals – mouses and keyboards.
  - Wearable smart devices – fitness tracker and smart watches.
  - Audio equipments – wireless headphones and speakers.
  - IoT – smart door locks and smart lights.

Overview

- The Bluetooth standard is comprised of two main protocols
    - Bluetooth BR/EDR, and
    - Bluetooth Low Energy (aka. Bluetooth Smart)
- Both protocols promise to provide confidentiality and MitM protection.
- In this talk we show that none of these protocols provided the promised protections.

Bluetooth Pairing

- The Bluetooth pairing establishes connection between two devices.
- The latest pairing protocols are
    - Bluetooth BR/EDR – Secure Simple Pairing (SSP)
    - Bluetooth Low Energy – Low Energy Secure Connections (LE SC)
- Both LE SC and SSP are variants of authenticated Elliptic-Curve Diffie-Hellman protocol for key-exchange.

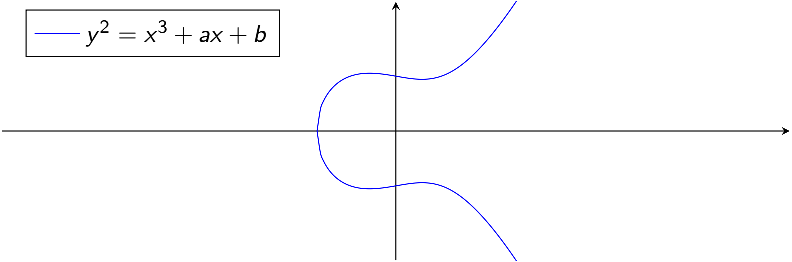
Legacy Pairing Eavesdropping Attack

- A paper published in 2013 by Mike Ryan pointed out that BTLE "Legacy Pairing" is vulnerable to an eavesdropping attack.
    - Legacy Pairing is protected by a 6-digit decimal mutual temporary key.
    - The attack recovers the session key by exhaustively searching through all million possible temporary keys.
    - This vulnerability was mitigated by LE SC using ECDH.
- Mike Ryan also published CrackLE, an open-source software that recovers the session key from captured Legacy Pairing traffic.

CRACKLE

Introduction to Elliptic Curves

- Elliptic curves over finite fields are defined by group equation and the underlying field $\mathbb{F}_q$.
- Consider curves in Weierstrass form $y^2 = x^3 + ax + b$.



- The elements of the group are:
  - All pairs $(x, y) \in \mathbb{F}_q^2$ that satisfy the curve equation.
  - An identity element called *point-at-infinity* denoted by ∞.
  - We denote points that satisfy the equation as $P = (Px, Py)$.
- The figures are drawn over $\mathbb{R}$ for intuition, while the formulae are defined over $\mathbb{F}_q$ as used in cryptography.

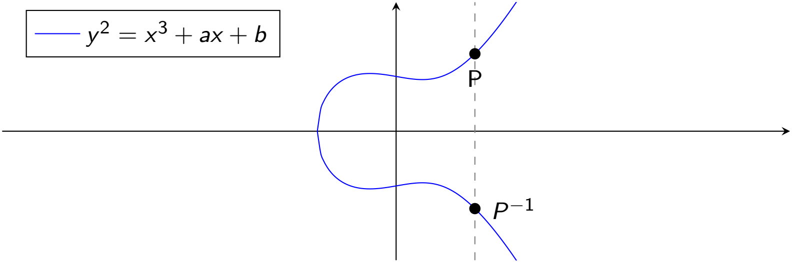
Introduction to Elliptic Curves

- The group operation is point addition.
- The use the following notations:
  - Point Addition – Adding two group elements $P, Q \in E$, st. $P \neq Q$.
  - Point Doubling – Adding a group element $P \in E$ to itself.
  - Repeated Addition – Denote $[\alpha]P$ to be the sum of α times repeated additions of P to itself.

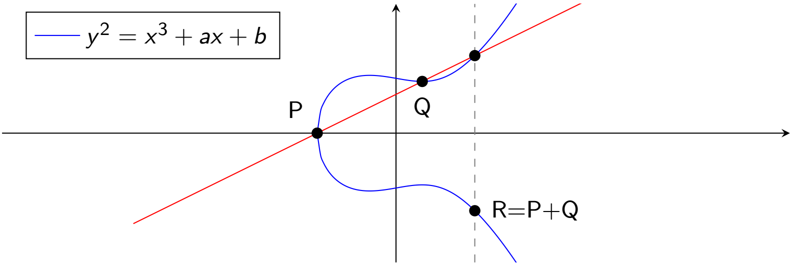
Point Inversion

- Given a point $P = (Px, Py)$ the inverse of P is computed by reflecting it across the x-axis

$$P^{-1} = (Px, -Py).$$



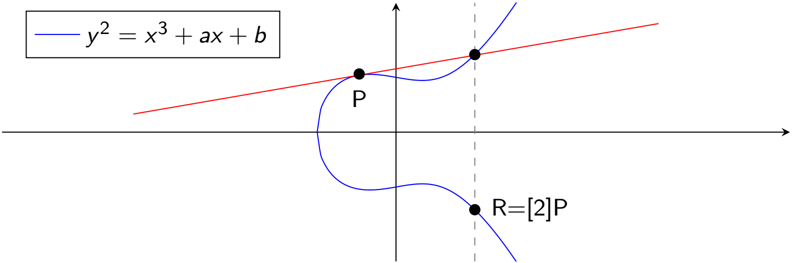
Point Addition



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

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

Point Doubling



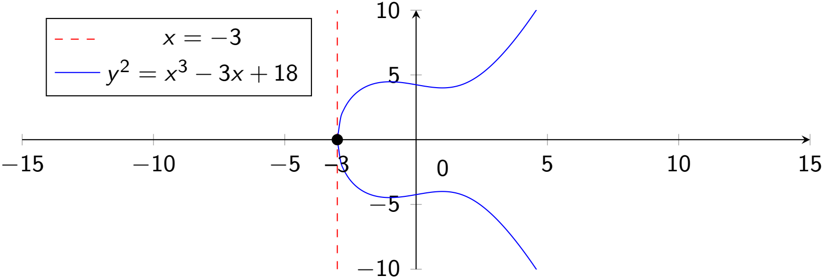
$$s \equiv (3Px^2 + a)(2Py)^{-1} \pmod{q}$$
$$Rx \equiv s^2 - 2Px \pmod{q}$$
$$Ry \equiv Py - s(Rx - Px) \pmod{q}$$

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

Order Two Points

- An important observation is that every point of the form $P = (Px, 0)$ equals its own inverse, thus has order two

$$[2]P = \infty.$$

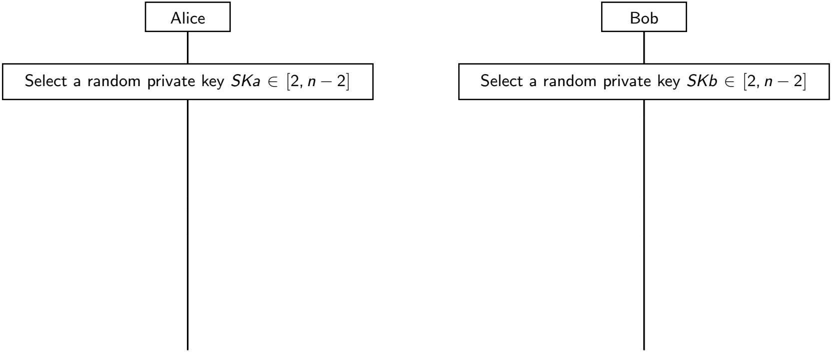


Elliptic Curve Diffie-Hellman

- The *Elliptic Curve Diffie-Hellman* (*ECDH*) protocol is a variant of the Diffie-Hellman key exchange protocol.
- Both parties agree on an Elliptic Curve E and a generator point $P \in E$.
- Then they communicate as follows:

Elliptic Curve Diffie-Hellman

- The *Elliptic Curve Diffie-Hellman* (*ECDH*) protocol is a variant of the Diffie-Hellman key exchange protocol.
- Both parties agree on an Elliptic Curve E and a generator point $P \in E$.
- Then they communicate as follows:



Alice

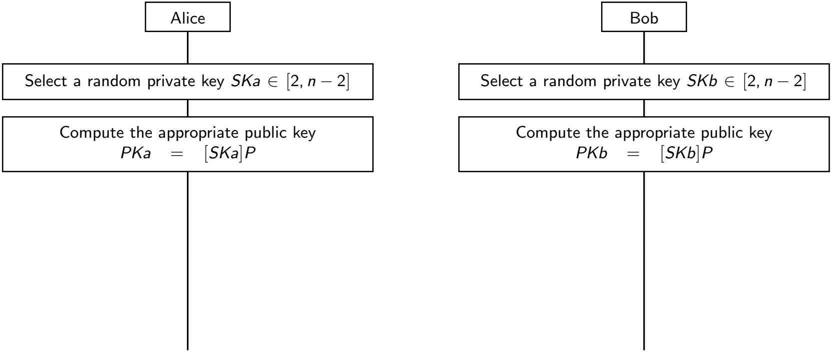
Select a random private key $SKa \in [2, n-2]$

Bob

Select a random private key $SKb \in [2, n-2]$

Elliptic Curve Diffie-Hellman

- The *Elliptic Curve Diffie-Hellman* (*ECDH*) protocol is a variant of the Diffie-Hellman key exchange protocol.
- Both parties agree on an Elliptic Curve E and a generator point $P \in E$.
- Then they communicate as follows:



Alice

Select a random private key $SKa \in [2, n-2]$

Compute the appropriate public key
$PKa = [SKa]P$

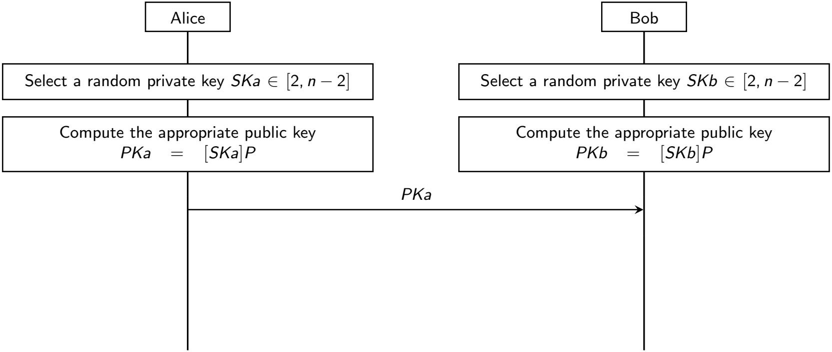
Bob

Select a random private key $SKb \in [2, n-2]$

Compute the appropriate public key
$PKb = [SKb]P$

Elliptic Curve Diffie-Hellman

- The *Elliptic Curve Diffie-Hellman* (*ECDH*) protocol is a variant of the Diffie-Hellman key exchange protocol.
- Both parties agree on an Elliptic Curve E and a generator point $P \in E$.
- Then they communicate as follows:



Alice

Bob

Select a random private key $SKa \in [2, n-2]$

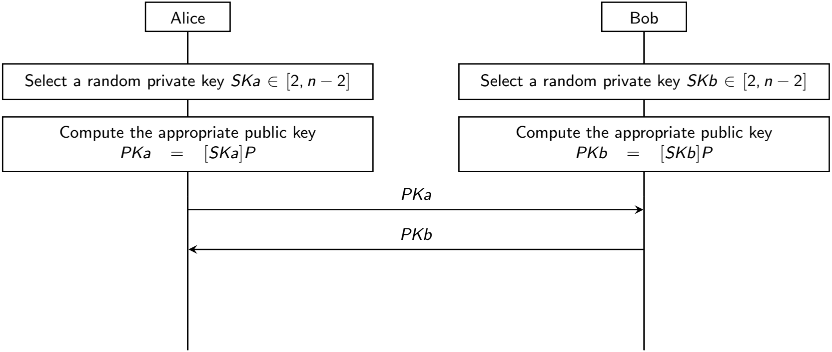
Select a random private key $SKb \in [2, n-2]$

Compute the appropriate public key $PKa = [SKa]P$

Compute the appropriate public key $PKb = [SKb]P$

Elliptic Curve Diffie-Hellman

- The *Elliptic Curve Diffie-Hellman* (*ECDH*) protocol is a variant of the Diffie-Hellman key exchange protocol.
- Both parties agree on an Elliptic Curve E and a generator point $P \in E$.
- Then they communicate as follows:



Alice

Bob

Select a random private key $SKa \in [2, n-2]$

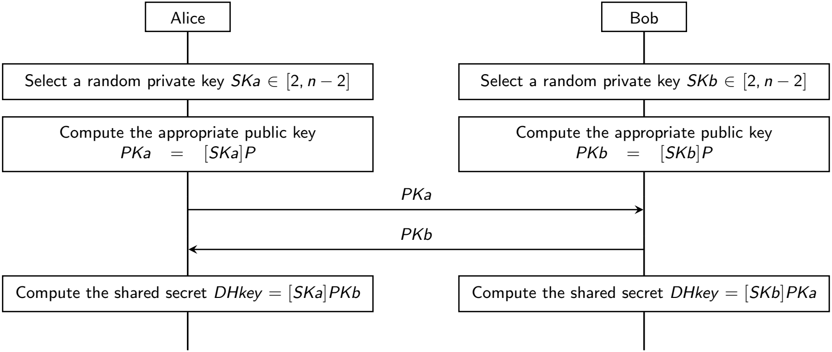
Select a random private key $SKb \in [2, n-2]$

Compute the appropriate public key $PKa = [SKa]P$

Compute the appropriate public key $PKb = [SKb]P$

Elliptic Curve Diffie-Hellman

- The *Elliptic Curve Diffie-Hellman* (*ECDH*) protocol is a variant of the Diffie-Hellman key exchange protocol.
- Both parties agree on an Elliptic Curve E and a generator point $P \in E$.
- Then they communicate as follows:



Alice

Bob

Select a random private key $SKa \in [2, n-2]$

Select a random private key $SKb \in [2, n-2]$

Compute the appropriate public key $PKa = [SKa]P$

Compute the appropriate public key $PKb = [SKb]P$

PKa

PKb

Compute the shared secret $DHkey = [SKa]PKb$

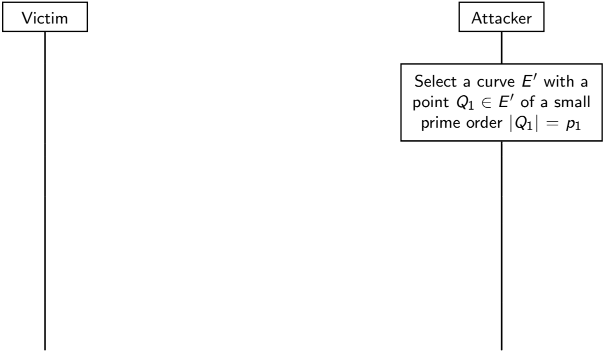
Compute the shared secret $DHkey = [SKb]PKa$

Invalid Curve Attack

- The Invalid Curve Attack, introduced by Biehl et al., is a cryptographic attack where invalid group elements (points) are used in order to manipulate the group operations to reveal secret information.

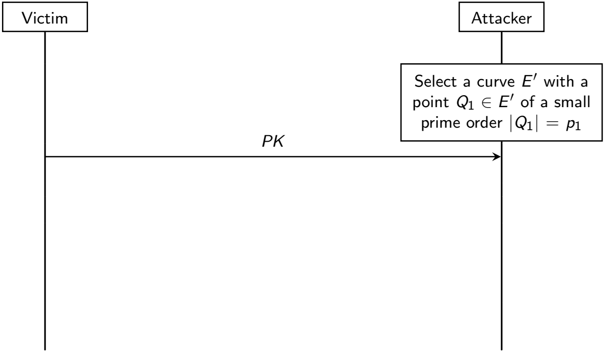
Invalid Curve Attack

- Let SK be the secret key of the victim device and let $PK = [SK]P$ its public key.
- Let E' be a different group defined by the curve equation $y^2 = x^3 + ax + b'$ with the same a and a different b' parameter.



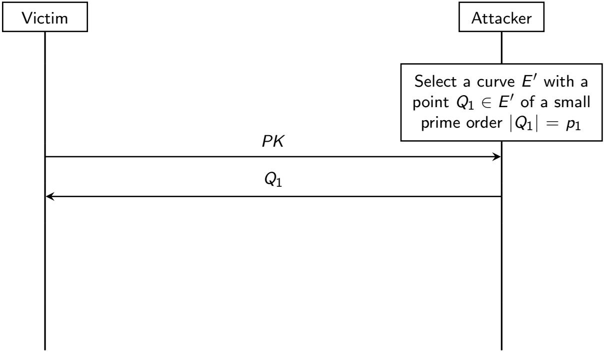
Invalid Curve Attack

- Let SK be the secret key of the victim device and let $PK = [SK]P$ its public key.
- Let E' be a different group defined by the curve equation $y^2 = x^3 + ax + b'$ with the same a and a different b' parameter.



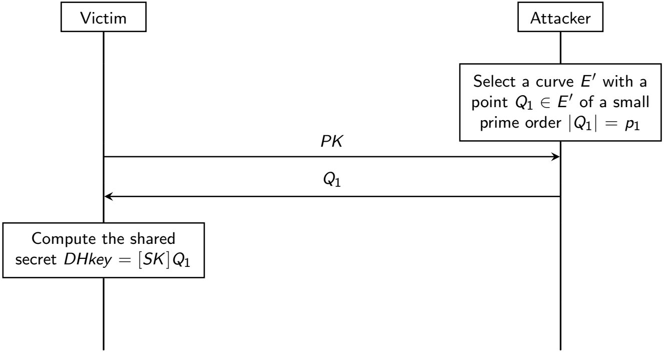
Invalid Curve Attack

- Let SK be the secret key of the victim device and let $PK = [SK]P$ its public key.
- Let E' be a different group defined by the curve equation $y^2 = x^3 + ax + b'$ with the same a and a different b' parameter.



Invalid Curve Attack

- Let SK be the secret key of the victim device and let $PK = [SK]P$ its public key.
- Let E' be a different group defined by the curve equation $y^2 = x^3 + ax + b'$ with the same a and a different b' parameter.



Victim

Attacker

Select a curve E' with a point $Q_1 \in E'$ of a small prime order $|Q_1| = p_1$

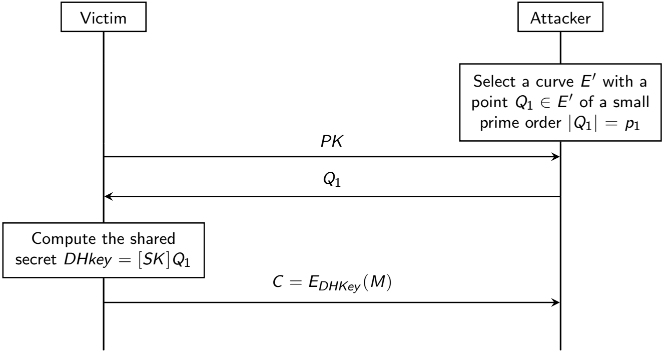
PK

Q_1

Compute the shared secret $DHkey = [SK]Q_1$

Invalid Curve Attack

- Let SK be the secret key of the victim device and let $PK = [SK]P$ its public key.
- Let E' be a different group defined by the curve equation $y^2 = x^3 + ax + b'$ with the same a and a different b' parameter.



Victim

Attacker

Select a curve E' with a point $Q_1 \in E'$ of a small prime order $|Q_1| = p_1$

PK

Q_1

Compute the shared secret $DHkey = [SK]Q_1$

$C = E_{DHKey}(M)$

Invalid Curve Attack

- For simplicity lets assume that M is a message known to the attacker.
- The attacker wishes to find the discrete log of $DHKey$ in the small subgroup generated by Q_1.
- Let a_1 be the discrete log of $DHkey$:

$$a_1 \equiv SK \pmod{p_1}.$$

- The attacker finds a_1 by iterating over all $a_1 \in [0, p_1 - 1]$ and checking whether $E_{[a_1]Q_1}(M) = C$.
- This exchange repeats with a different subgroup orders p_i 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.

Invalid Curve Attack

- A patent assigned by Peter Landrock and Jan Ulrik Kjaersgaard in 2008 describes how attacker could reveal the private key of the victim in SSP using the Invalid Curve Attack.
  - As a mitigation the BT specification suggests refreshing the ECDH key-pair on every pairing attempt.
  - Most implementors follow this suggestion.

Bluetooth Pairing

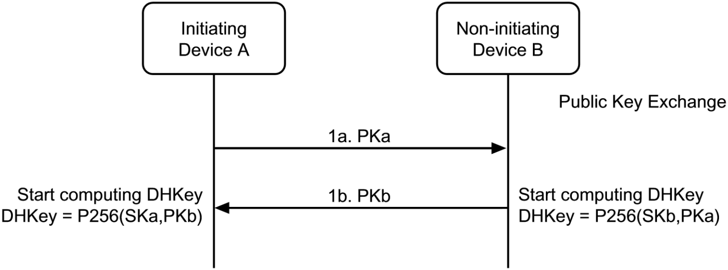
- The pairing protocol is part of the Bluetooth link layer protocol.
  - It generates the encryption keys for the rest of the protocol.
- Due to the similarity of SSP and LE SC, our attack applies to both protocols.
  - For this presentation we arbitrarily chose to concentrate on LE SC.

Bluetooth LE Secure Connections

The protocol comprises of four phases:

- Phase 1 – Feature exchange (irrelevant for this talk).
- Phase 2 – Key exchange.
- Phase 3 – Authentication.
- Phase 4 – Key derivation.

Bluetooth LE SC Phase 2 – Key Exchange



Initiating Device A

Non-initiating Device B

Public Key Exchange

1a. PKa

1b. PKb

Start computing DHKey
DHKey = P256(SKa,PKb)

Start computing DHKey
DHKey = P256(SKb,PKa)

Cryptographic Functions

Function f4 – Commitment Value Generation Function

$$\text{f4}(U, V, X, Y) = \text{AES-CMAC}_X(U \parallel V \parallel Y)$$

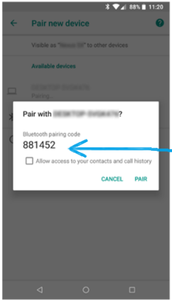
Function g2 – User Confirm Value Generation Function

The six least decimal digits of the following function:

$$g2(U, V, X, Y) = \text{AES-CMAC}_X(U \parallel V \parallel Y) \pmod{2^{32}}$$

Bluetooth LE SC Phase 3 – Authentication

Note that unintuitively *PKa* and *PKb* in this diagram refers to the **x-coordinate** of each public-key, later in the specification defined as *PKax* and *PKbx*.



Pair new device

Visible as "[redacted]" to other devices

Available devices

[redacted]

Pairing...

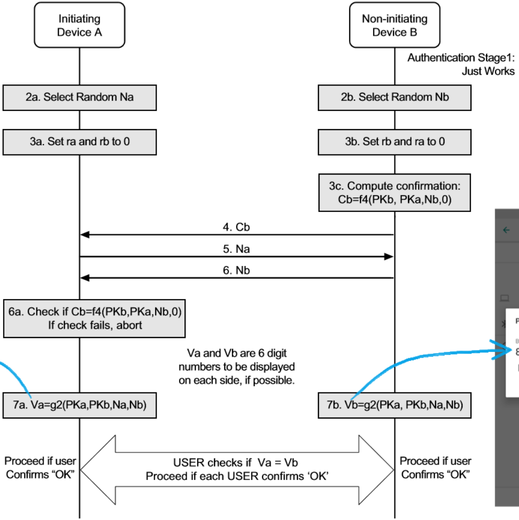
Pair with [redacted]?

Bluetooth pairing code

881452

☐ Allow access to your contacts and call history

CANCEL PAIR



Initiating Device A

Non-initiating Device B

Authentication Stage1: Just Works

2a. Select Random Na

2b. Select Random Nb

3a. Set ra and rb to 0

3b. Set rb and ra to 0

3c. Compute confirmation: Cb=f4(PKb, PKa,Nb,0)

4. Cb

5. Na

6. Nb

6a. Check if Cb=f4(PKb,PKa,Nb,0) If check fails, abort

Va and Vb are 6 digit numbers to be displayed on each side, if possible.

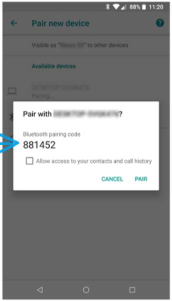
7a. Va=g2(PKa,PKb,Na,Nb)

7b. Vb=g2(PKa, PKb,Na,Nb)

Proceed if user Confirms "OK"

USER checks if Va = Vb
Proceed if each USER confirms 'OK'

Proceed if user Confirms "OK"



Pair new device

Visible as "[redacted]" to other devices

Available devices

[redacted]

Pairing...

Pair with [redacted]?

Bluetooth pairing code

881452

☐ Allow access to your contacts and call history

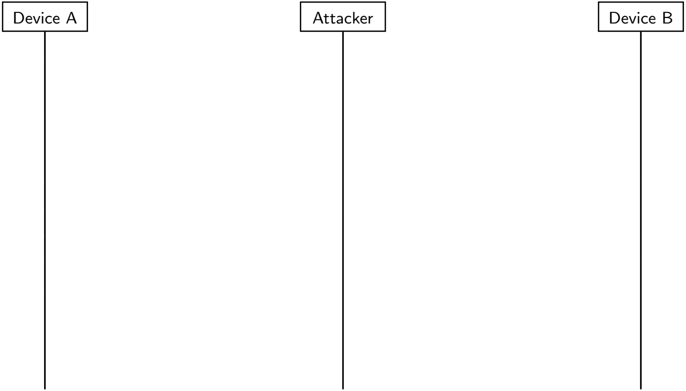
CANCEL PAIR

Our Fixed Coordinate Invalid Curve Attack

- The Fixed Coordinate Invalid Curve Attack is a new variant of the Invalid Curve Attack in which we exploit the ability to forge low order ECDH public keys that preserve the x-coordinate of the original public-keys.
- It is based on the following observations:
  - Only the x-coordinate of each party is authenticated during the Bluetooth pairing protocol.
  - The protocol does not require its implementations to validate whether a given public-key satisfies the curve equation.
- We describe two versions of our attack:
  - Semi-Passive.
  - Fully-Active.

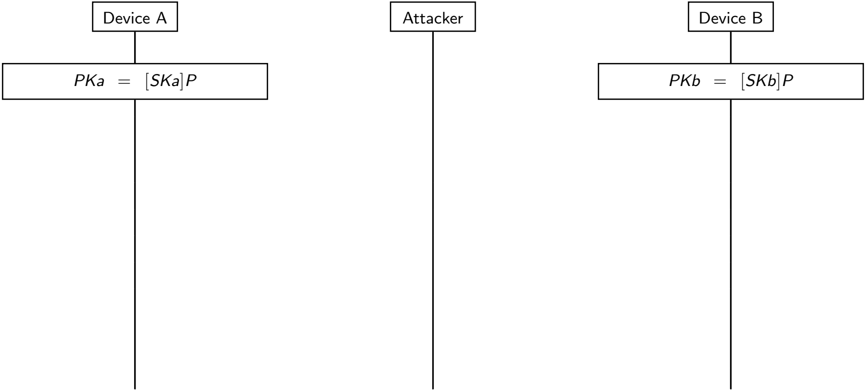
The Semi-Passive Attack

- The Semi-Passive attack requires a message interception during the second phase of the pairing.
- It replaces the y-coordinate of each public key with 0.



The Semi-Passive Attack

- The Semi-Passive attack requires a message interception during the second phase of the pairing.
- It replaces the y-coordinate of each public key with 0.



Device A

Attacker

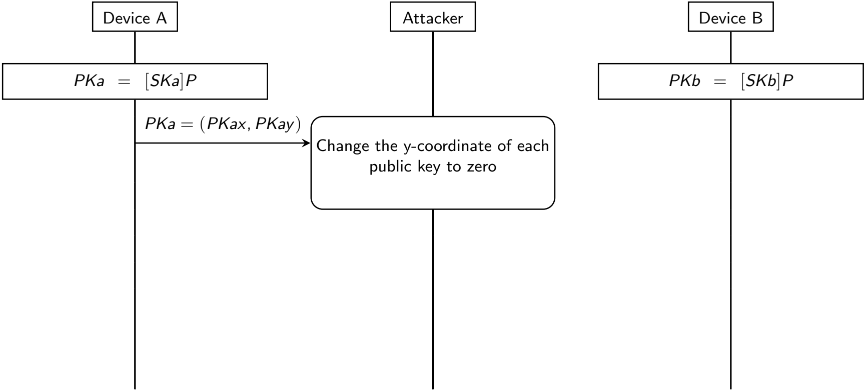
Device B

$PKa = [SKa]P$

$PKb = [SKb]P$

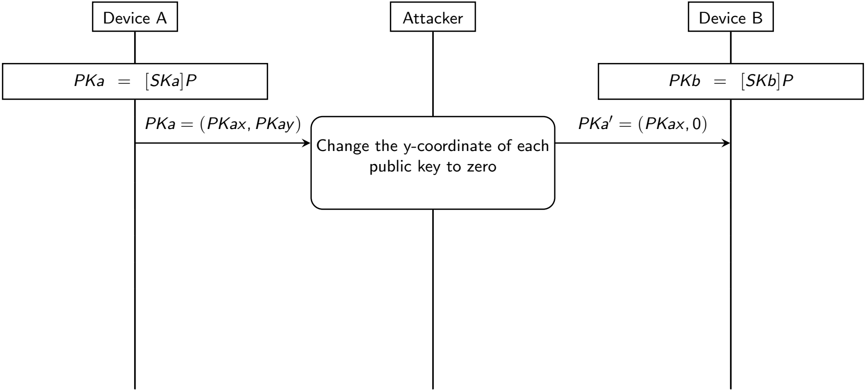
The Semi-Passive Attack

- The Semi-Passive attack requires a message interception during the second phase of the pairing.
- It replaces the y-coordinate of each public key with 0.



The Semi-Passive Attack

- The Semi-Passive attack requires a message interception during the second phase of the pairing.
- It replaces the y-coordinate of each public key with 0.



Device A

Attacker

Device B

$PKa = [SKa]P$

$PKb = [SKb]P$

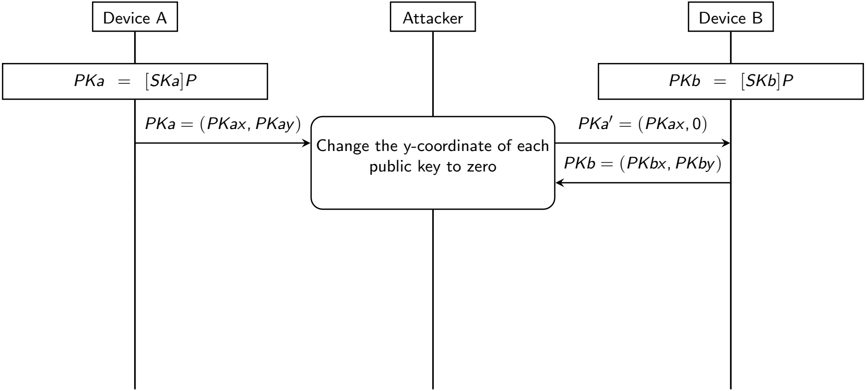
$PKa = (PKax, PKay)$

Change the y-coordinate of each public key to zero

$PKa' = (PKax, 0)$

The Semi-Passive Attack

- The Semi-Passive attack requires a message interception during the second phase of the pairing.
- It replaces the y-coordinate of each public key with 0.



Device A

Attacker

Device B

$PKa = [SKa]P$

$PKb = [SKb]P$

$PKa = (PKax, PKay)$

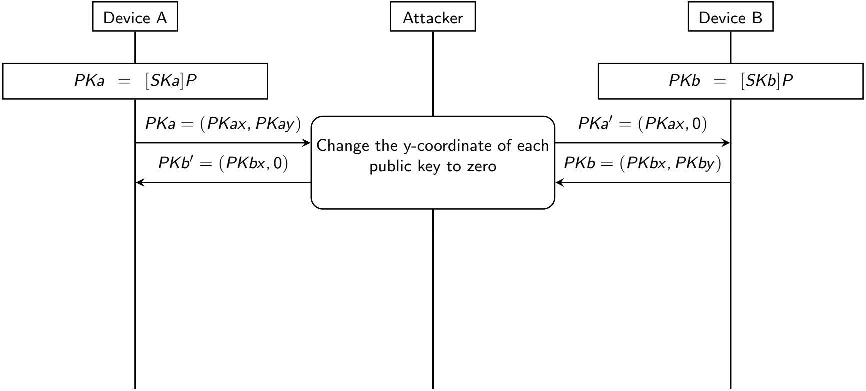
Change the y-coordinate of each public key to zero

$PKa' = (PKax, 0)$

$PKb = (PKbx, PKby)$

The Semi-Passive Attack

- The Semi-Passive attack requires a message interception during the second phase of the pairing.
- It replaces the y-coordinate of each public key with 0.



Device A

Attacker

Device B

$PKa = [SKa]P$

$PKb = [SKb]P$

$PKa = (PKax, PKay)$

$PKb' = (PKbx, 0)$

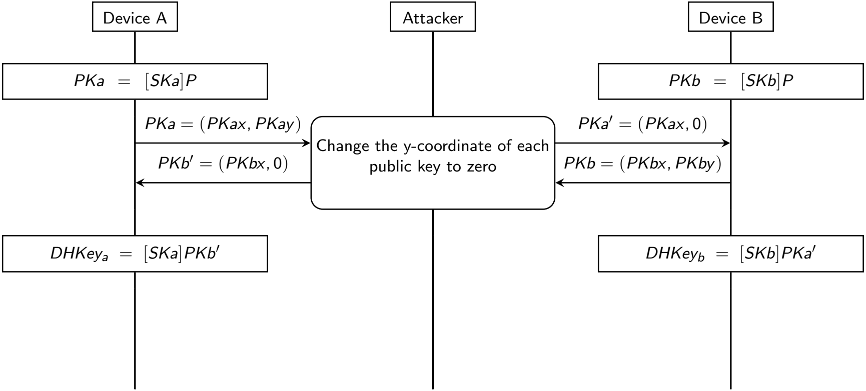
Change the y-coordinate of each public key to zero

$PKa' = (PKax, 0)$

$PKb = (PKbx, PKby)$

The Semi-Passive Attack

- The Semi-Passive attack requires a message interception during the second phase of the pairing.
- It replaces the y-coordinate of each public key with 0.



Device A

Attacker

Device B

$PKa = [SKa]P$

$PKb = [SKb]P$

$PKa = (PKax, PKay)$

Change the y-coordinate of each public key to zero

$PKa' = (PKax, 0)$

$PKb' = (PKbx, 0)$

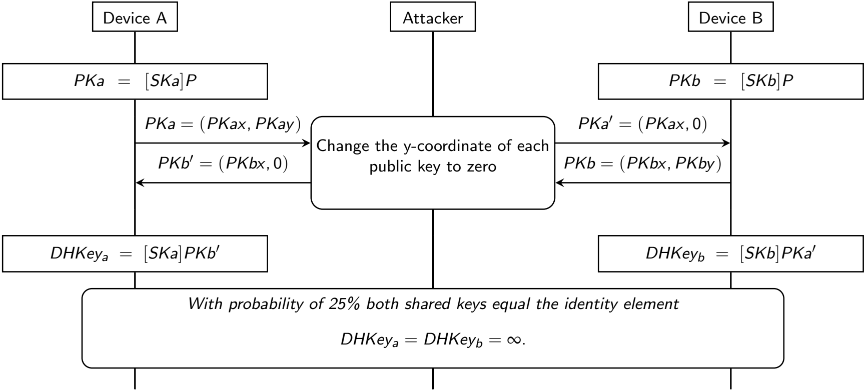
$PKb = (PKbx, PKby)$

$DHKey_a = [SKa]PKb'$

$DHKey_b = [SKb]PKa'$

The Semi-Passive Attack

- The Semi-Passive attack requires a message interception during the second phase of the pairing.
- It replaces the y-coordinate of each public key with 0.



Device A

Attacker

Device B

$PKa = [SKa]P$

$PKb = [SKb]P$

$PKa = (PKax, PKay)$

$PKb' = (PKbx, 0)$

Change the y-coordinate of each public key to zero

$PKa' = (PKax, 0)$

$PKb = (PKbx, PKby)$

$DHKey_a = [SKa]PKb'$

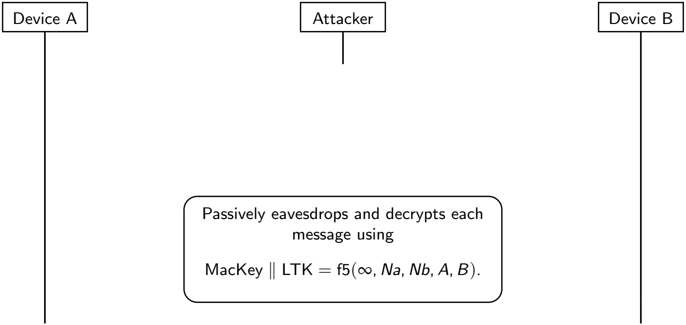
$DHKey_b = [SKb]PKa'$

With probability of 25% both shared keys equal the identity element

$DHKey_a = DHKey_b = \infty.$

The Semi-Passive Attack – Passive Message Eavesdropping

- In case both shared keys equal the identity element
    - the attack is undetected,
    - the attacker knows the shared key, and
    - the rest of the communication can be passively eavesdropped.

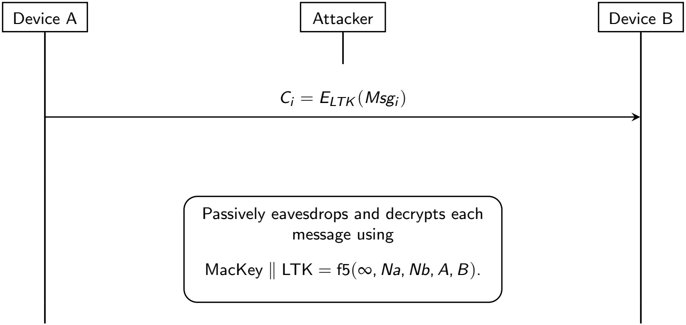


Passively eavesdrops and decrypts each message using

MacKey $\|$ LTK $= \mathrm{f5}(\infty, Na, Nb, A, B)$.

The Semi-Passive Attack – Passive Message Eavesdropping

- In case both shared keys equal the identity element
  - the attack is undetected,
  - the attacker knows the shared key, and
  - the rest of the communication can be passively eavesdropped.

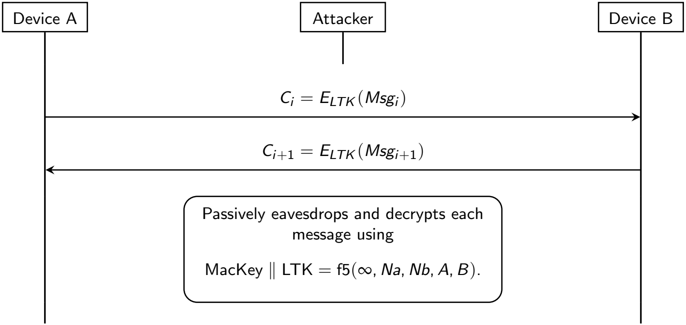


Passively eavesdrops and decrypts each message using

MacKey $\|$ LTK $=$ f5(∞, Na, Nb, A, B).

The Semi-Passive Attack – Passive Message Eavesdropping

- In case both shared keys equal the identity element
  - the attack is undetected,
  - the attacker knows the shared key, and
  - the rest of the communication can be passively eavesdropped.



Passively eavesdrops and decrypts each message using

MacKey $\|$ LTK $=$ f5(∞, Na, Nb, A, B).

Cryptographic Functions

Function f5 – Key Derivation Function

$$SALT = \text{0x6C888391AAF5A53860370BDB5A6083BE}$$

$$T = \text{AES-CMAC}_{SALT}(DHKey)$$

$$\text{f5}(DHKey, N1, N2, A1, B2) =$$

$$\text{AES-CMAC}_T(0 \,\|\, \text{`}btle' \,\|\, N1 \,\|\, N2 \,\|\, A1 \,\|\, A2 \,\|\, 256) \,\|$$

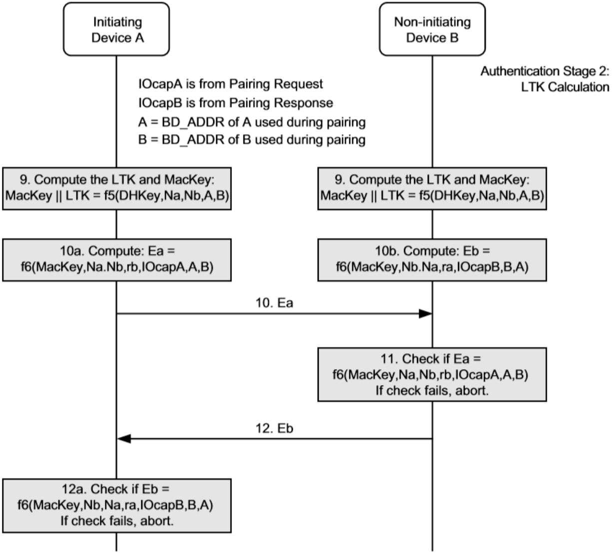
$$\text{AES-CMAC}_T(1 \,\|\, \text{`}btle' \,\|\, N1 \,\|\, N2 \,\|\, A1 \,\|\, A2 \,\|\, 256)$$

Function f6 – Check Value Generation Function

$$\text{f6}(W, N1, N2, R, IOcap, A1, A2) =$$

$$\text{AES-CMAC}_W(N1 \,\|\, N2 \,\|\, R \,\|\, IOcap \,\|\, A1 \,\|\, A2)$$

Bluetooth LE SC Phase 4 – Key Derivation



Initiating Device A

Non-initiating Device B

Authentication Stage 2:
LTK Calculation

IOcapA is from Pairing Request
IOcapB is from Pairing Response
A = BD_ADDR of A used during pairing
B = BD_ADDR of B used during pairing

9. Compute the LTK and MacKey:
MacKey || LTK = f5(DHKey,Na,Nb,A,B)

9. Compute the LTK and MacKey:
MacKey || LTK = f5(DHKey,Na,Nb,A,B)

10a. Compute: Ea =
f6(MacKey,Na.Nb,rb,IOcapA,A,B)

10b. Compute: Eb =
f6(MacKey,Nb.Na,ra,IOcapB,B,A)

10. Ea

11. Check if Ea =
f6(MacKey,Na,Nb,rb,IOcapA,A,B)
If check fails, abort.

12. Eb

12a. Check if Eb =
f6(MacKey,Nb,Na,ra,IOcapB,B,A)
If check fails, abort.

The Fully-Active Attack

- By also intercepting messages sent during the fourth phase we can further improve the attack success probability to 50%.
- $DHKey_b$ never equals PKb'

  $\Longrightarrow$ the Semi-Passive attack fails when $DHKey_a = PKb'$.

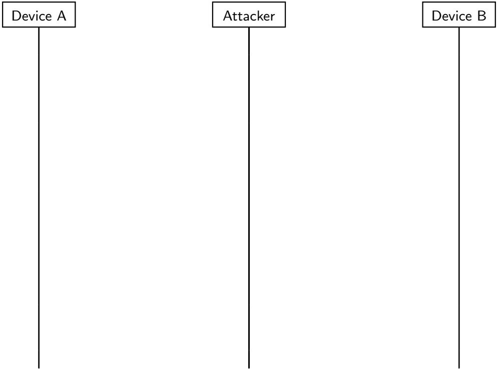
$DHKey_a$	$DHKey_b$
∞	∞
∞	PKa'
PKb'	∞
PKb'	PKa'

The Fully-Active Attack

- In the beginning of the fourth phase Device A commits to the mutual key by transmitting Ea.
- The attacker can use the value of Ea in order to determine the value of $DHKey_a \in \{PKb', \infty\}$.
- If $DHKey_a = \infty$ the attacker continues as described in the Semi-Passive Attack without further interception.

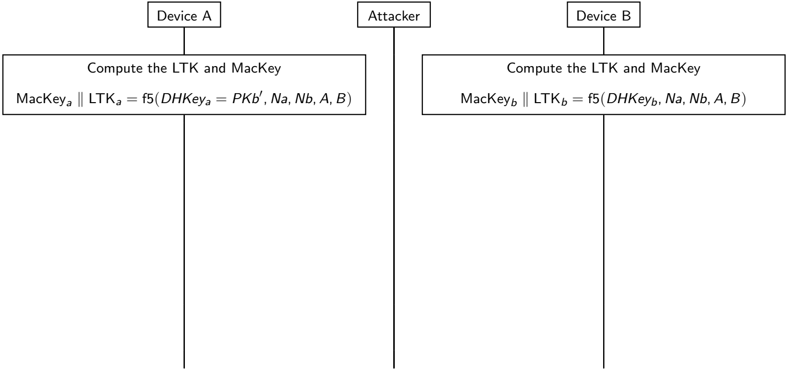
The Fully-Active Attack – Phase 4

- The following diagram describes the attack considering $DHKey_a = PKb'$.



The Fully-Active Attack – Phase 4

- The following diagram describes the attack considering $DHKey_a = PKb'$.



Device A

Attacker

Device B

Compute the LTK and MacKey

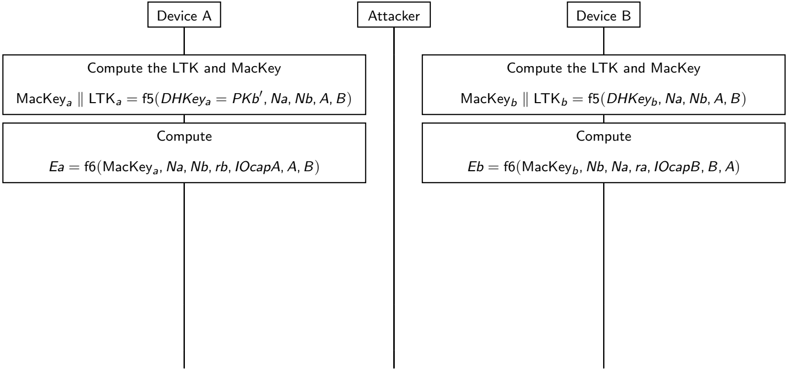
$\text{MacKey}_a \parallel \text{LTK}_a = \text{f5}(DHKey_a = PKb', Na, Nb, A, B)$

Compute the LTK and MacKey

$\text{MacKey}_b \parallel \text{LTK}_b = \text{f5}(DHKey_b, Na, Nb, A, B)$

The Fully-Active Attack – Phase 4

- The following diagram describes the attack considering $DHKey_a = PKb'$.



Compute the LTK and MacKey

$\mathrm{MacKey}_a \parallel \mathrm{LTK}_a = \mathrm{f5}(DHKey_a = PKb', Na, Nb, A, B)$

Compute

$Ea = \mathrm{f6}(\mathrm{MacKey}_a, Na, Nb, rb, IOcapA, A, B)$

Compute the LTK and MacKey

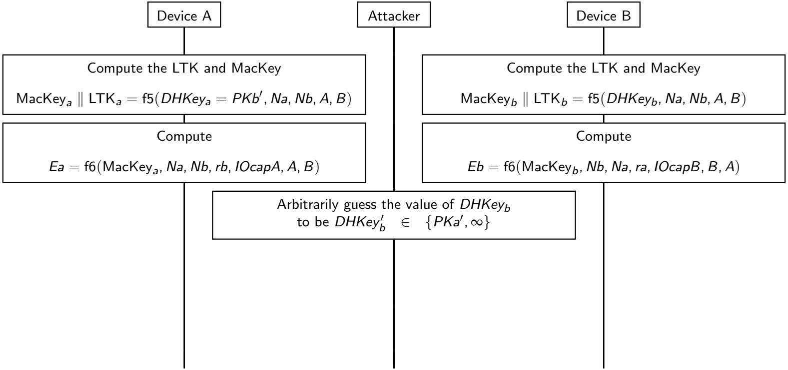
$\mathrm{MacKey}_b \parallel \mathrm{LTK}_b = \mathrm{f5}(DHKey_b, Na, Nb, A, B)$

Compute

$Eb = \mathrm{f6}(\mathrm{MacKey}_b, Nb, Na, ra, IOcapB, B, A)$

The Fully-Active Attack – Phase 4

- The following diagram describes the attack considering $DHKey_a = PKb'$.



Device A

Attacker

Device B

Compute the LTK and MacKey

$\text{MacKey}_a \parallel \text{LTK}_a = \text{f5}(DHKey_a = PKb', Na, Nb, A, B)$

Compute the LTK and MacKey

$\text{MacKey}_b \parallel \text{LTK}_b = \text{f5}(DHKey_b, Na, Nb, A, B)$

Compute

$Ea = \text{f6}(\text{MacKey}_a, Na, Nb, rb, IOcapA, A, B)$

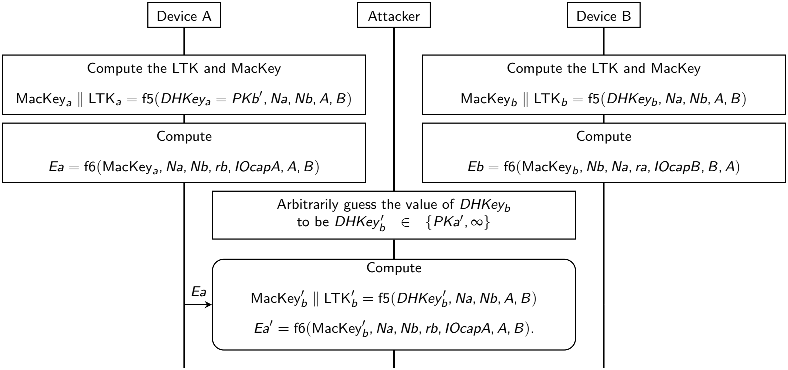
Compute

$Eb = \text{f6}(\text{MacKey}_b, Nb, Na, ra, IOcapB, B, A)$

Arbitrarily guess the value of $DHKey_b$
to be $DHKey'_b \in \{PKa', \infty\}$

The Fully-Active Attack – Phase 4

- The following diagram describes the attack considering $DHKey_a = PKb'$.



Device A

Attacker

Device B

Compute the LTK and MacKey

$\mathrm{MacKey}_a \parallel \mathrm{LTK}_a = \mathrm{f5}(DHKey_a = PKb', Na, Nb, A, B)$

Compute the LTK and MacKey

$\mathrm{MacKey}_b \parallel \mathrm{LTK}_b = \mathrm{f5}(DHKey_b, Na, Nb, A, B)$

Compute

$Ea = \mathrm{f6}(\mathrm{MacKey}_a, Na, Nb, rb, IOcapA, A, B)$

Compute

$Eb = \mathrm{f6}(\mathrm{MacKey}_b, Nb, Na, ra, IOcapB, B, A)$

Arbitrarily guess the value of $DHKey_b$ to be $DHKey'_b \in \{PKa', \infty\}$

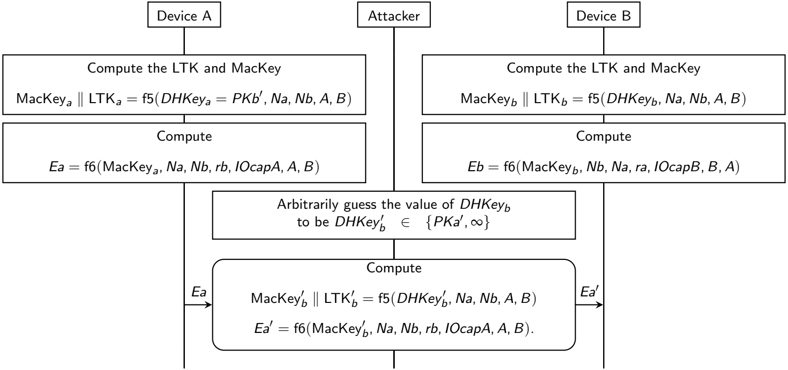
Compute

$\mathrm{MacKey}'_b \parallel \mathrm{LTK}'_b = \mathrm{f5}(DHKey'_b, Na, Nb, A, B)$

$Ea' = \mathrm{f6}(\mathrm{MacKey}'_b, Na, Nb, rb, IOcapA, A, B)$.

The Fully-Active Attack – Phase 4

- The following diagram describes the attack considering $DHKey_a = PKb'$.



Compute the LTK and MacKey

$\text{MacKey}_a \parallel \text{LTK}_a = \text{f5}(DHKey_a = PKb', Na, Nb, A, B)$

Compute the LTK and MacKey

$\text{MacKey}_b \parallel \text{LTK}_b = \text{f5}(DHKey_b, Na, Nb, A, B)$

Compute

$Ea = \text{f6}(\text{MacKey}_a, Na, Nb, rb, IOcapA, A, B)$

Compute

$Eb = \text{f6}(\text{MacKey}_b, Nb, Na, ra, IOcapB, B, A)$

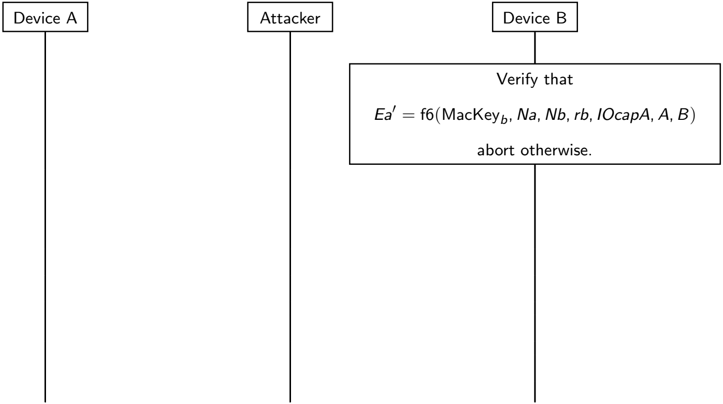
Arbitrarily guess the value of $DHKey_b$ to be $DHKey'_b \in \{PKa', \infty\}$

Compute

$\text{MacKey}'_b \parallel \text{LTK}'_b = \text{f5}(DHKey'_b, Na, Nb, A, B)$

$Ea' = \text{f6}(\text{MacKey}'_b, Na, Nb, rb, IOcapA, A, B)$.

The Fully-Active Attack – Phase 4 (Cont.)



Device A

Attacker

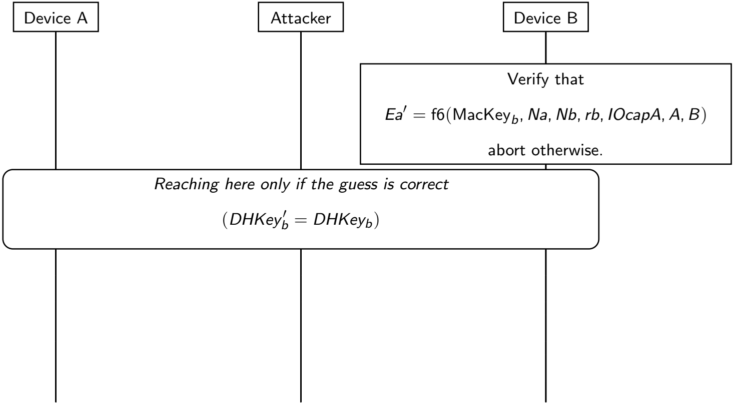
Device B

Verify that

$Ea' = \mathrm{f6}(\mathrm{MacKey}_b, Na, Nb, rb, IOcapA, A, B)$

abort otherwise.

The Fully-Active Attack – Phase 4 (Cont.)



Device A

Attacker

Device B

Verify that

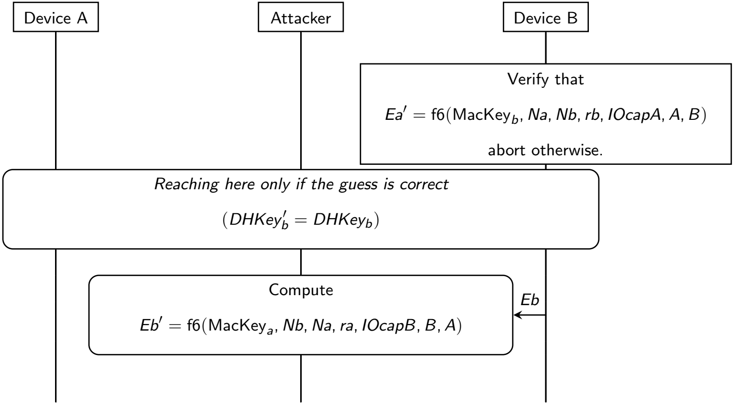
$Ea' = \mathrm{f6}(\mathrm{MacKey}_b, Na, Nb, rb, IOcapA, A, B)$

abort otherwise.

Reaching here only if the guess is correct

$(DHKey'_b = DHKey_b)$

The Fully-Active Attack – Phase 4 (Cont.)



Verify that

$Ea' = \mathrm{f6}(\mathrm{MacKey}_b, Na, Nb, rb, IOcapA, A, B)$

abort otherwise.

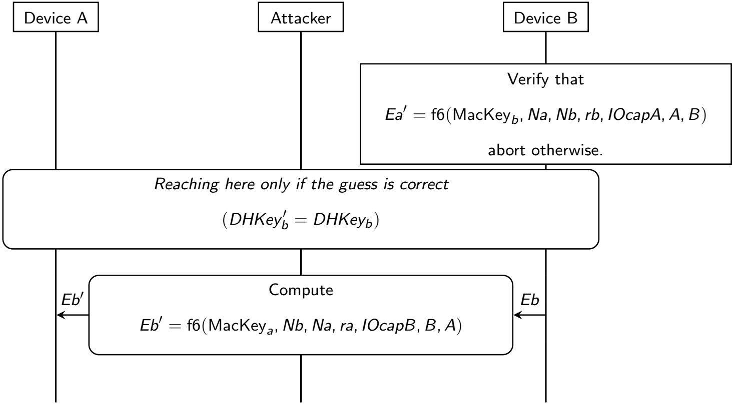
Reaching here only if the guess is correct

$(DHKey'_b = DHKey_b)$

Compute

$Eb' = \mathrm{f6}(\mathrm{MacKey}_a, Nb, Na, ra, IOcapB, B, A)$

The Fully-Active Attack – Phase 4 (Cont.)



Device A

Attacker

Device B

Verify that

$Ea' = \mathrm{f6}(\mathrm{MacKey}_b, Na, Nb, rb, IOcapA, A, B)$

abort otherwise.

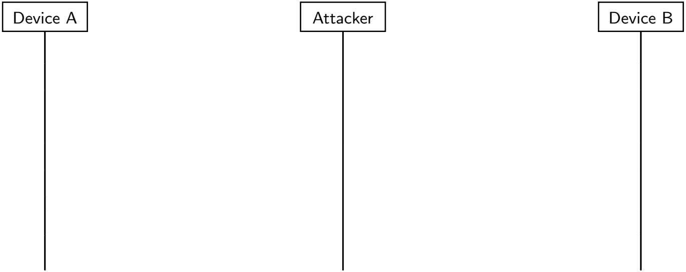
Reaching here only if the guess is correct

$(DHKey'_b = DHKey_b)$

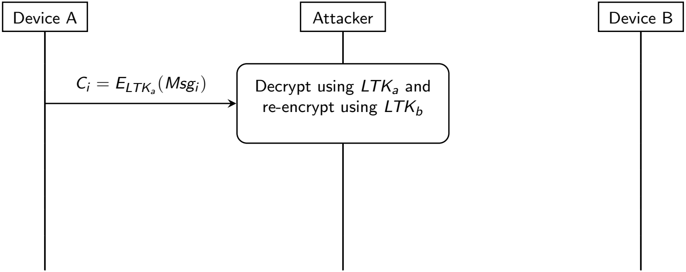
Compute

$Eb' = \mathrm{f6}(\mathrm{MacKey}_a, Nb, Na, ra, IOcapB, B, A)$

The Fully-Active Attack – Active Message Relaying



The Fully-Active Attack – Active Message Relaying



Device A

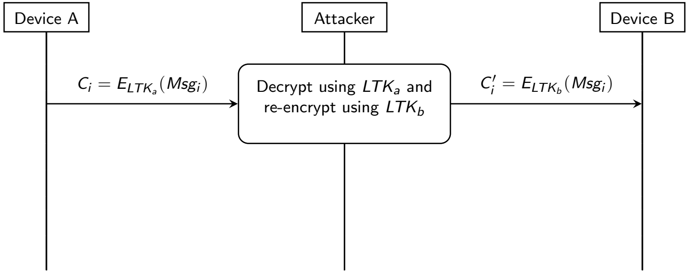
Attacker

Device B

$C_i = E_{LTK_a}(Msg_i)$

Decrypt using LTK_a and re-encrypt using LTK_b

The Fully-Active Attack – Active Message Relaying



Device A

Attacker

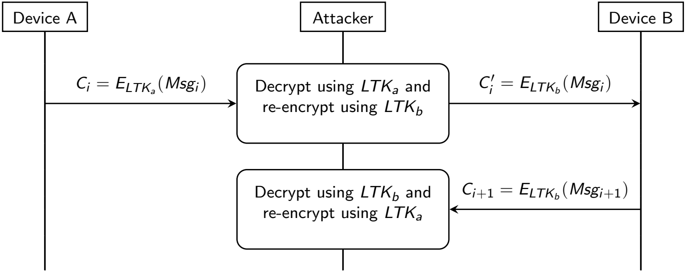
Device B

$C_i = E_{LTK_a}(Msg_i)$

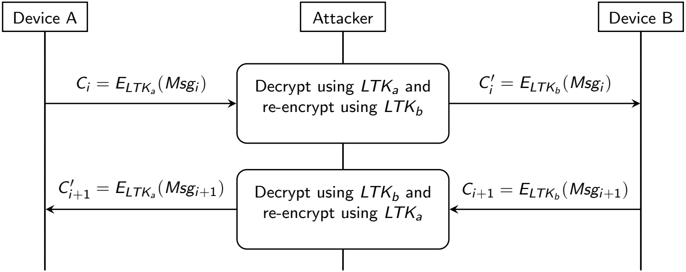
Decrypt using LTK_a and re-encrypt using LTK_b

$C_i' = E_{LTK_b}(Msg_i)$

The Fully-Active Attack – Active Message Relaying



The Fully-Active Attack – Active Message Relaying


Device A

Attacker

Device B

$C_i = E_{LTK_a}(Msg_i)$

Decrypt using LTK_a and re-encrypt using LTK_b

$C'_i = E_{LTK_b}(Msg_i)$

$C'_{i+1} = E_{LTK_a}(Msg_{i+1})$

Decrypt using LTK_b and re-encrypt using LTK_a

$C_{i+1} = E_{LTK_b}(Msg_{i+1})$

Success Rate of Our Attack

Success Rate – Semi-Passive Attack

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

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

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

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

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

Success Rate of Our Attack

Success Rate – Semi-Passive Attack

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

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

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

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

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

Frequency Hopping

- Bluetooth uses frequency hopping.
    - It has been shown that this frequency hopping could be predicted by an attacker and therefore does not provide security.
    - More sophisticated equipment can listen/transmit to all of the channels used by Bluetooth thus avoiding this issue entirely.

Over the Air Packet Manipulation

- MitM attacks requires over the air packets manipulation.
  - There are several projects that provide over the air packet manipulation capability on Bluetooth, such as GATTack.
  - Unfortunately, all of the solutions we found are limited to Bluetooth 4.0 and do not support Bluetooth 4.2 (with LE SC) due to its larger packet size.
  - It is safe to assume that products supporting Bluetooth 4.2 packet manipulation will be released in the near future as it becomes more popular.
- At the moment, only Bluetooth LE equipment is available for these attacks, since it is far simpler than Bluetooth BR/EDR.

Design Flaws

- Both the x-coordinate and the y-coordinate are sent during the public key exchange.

  ⟹ This is unnecessary and highly inadvisable.

- The protocol authenticates only the x-coordinate.

  ⟹ The y-coordinate remains unauthenticated.

Mitigations

- In order to protect against the classical Invalid Curve Attack the specification suggests refreshing the ECDH key-pair every pairing attempt.

  $\Longrightarrow$ Our attack still works when this mitigation is applied.

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

Vulnerable Platforms

- Our new attack was applicable to most available Bluetooth devices.
- We informed the Bluetooth SIG and the vendors.
- CVE-2018-5383 was assigned to this vulnerability in the Bluetooth protocol.

Vulnerable Platforms – Bluetooth LE SC

- LE SC pairing is implemented in the host.
- The vulnerability is found in the host's operating system
  - Regardless of the Bluetooth controller.
- The Android Bluetooth stack, "Bluedroid" is vulnerable.
  - Tested on Nexus 5X devices with Android version 8.1.
- Apple iOS and MacOS was found to be vulnerable.
  - This includes all of the latest Apple products (both laptops, phones and tablets).
- At the time of our publication Microsoft Windows did not yet support LE SC.
  - This made all Windows versions vulnerable to the simpler Legacy Pairing Eavesdropping Attack.

Vulnerable Platforms – Bluetooth BR/EDR SSP

- The key exchange in SSP is performed by the Bluetooth controller.
- The vulnerability depends on the Bluetooth controller's firmware implementation.
  - Independent of the operating-system.
- Controllers of most major vendors are vulnerable:
  - Qualcomm – Tested on Qualcomm's QCA6174A.
  - Broadcom – Tested on Broadcom's BCM4358 and BCM4339.
  - Intel – Tested on Intel 8265.

Industry Reaction

- Google rated this vulnerability as High-Severity.
  - A patch was released for the Android OS on June 4th 2018.
- Apple released a formal statement explaining the vulnerability to its users.
  - A patch for iOS and MacOS was released on July 23rd 2018.
- Intel rated this vulnerability as High Severity as well.
  - A patch, referred by INTEL-SA-00128, was released to dozens of Intel's products on July 23rd 2018.
- Qualcomm and Broadcom had also released patches to their vendor partners.

Bluetooth Protocol Fix

- On July 23rd the Bluetooth SIG released a statement to addressing our findings.
  - “To remedy the vulnerability, the Bluetooth SIG has now updated the Bluetooth specification to require products to validate any public key received as part of public key-based security procedures. In addition, the Bluetooth SIG has added testing for this vulnerability within our Bluetooth Qualification Program.”
  - The included specification change, released under the name “Erratum 10734”, implements our recommended mitigation.

Summary

- We introduced the *Fixed Coordinate Invalid Curve Attack* which provides
  - A new tool for attacking the ECDH protocols.
  - Presented the application of our new attack to the Bluetooth pairing protocol.
- As a result of our attack all of the variants of Bluetooth were proven insecure.
- We discovered multiple design flaws in the Bluetooth specification.
- We found that all of the major vendors are vulnerable.
- The Bluetooth protocol was modified according to our findings.

Thanks

- Special thanks to the CERT/CC for helping us managing the responsible disclosure to the vendors, and to the vendors for the cooperation on patching their systems.

The End