# **Dragonblood**: Analyzing the Dragonfly Handshake of WPA3 and EAP-pwd

Mathy Vanhoef and **Eyal Ronen**

KATHOLIEKE UNIVERSITEIT LEUVEN

TEL AVIV UNIVERSITY

NEW YORK UNIVERSITY

# Background: Wi-Fi Security

- 1999: Wired Equivalent Privacy (WEP)
  - RC4 with 40 (!) or 104 bits key
  - **Broken** in 2001 [FMS01]
  - Deprecated 2004

# Background: Wi-Fi Security

- 1999: Wired Equivalent Privacy (WEP)
  - RC4 with 40 (!) or 104 bits key
  - **Broken** in 2001 [FMS01]
  - Deprecated 2004
- 2003: Wi-Fi Protected Access (WPA)

# Background: Wi-Fi Security

- 1999: Wired Equivalent Privacy (WEP)
  - RC4 with 40 (!) or 104 bits key
  - **Broken** in 2001 [FMS01]
  - Deprecated 2004
- 2003: Wi-Fi Protected Access (WPA)
- 2004: Wi-Fi Protected Access 2 (WPA2)
  - Allows **offline password brute-force**
  - KRACK and Kraken attack [VP][2017-8]

# Background: Dragonfly in WPA3 and EAP-pwd

**= Password Authenticated Key Exchange (PAKE)**

# Background: Dragonfly in WPA3 and EAP-pwd

## = Password Authenticated Key Exchange (PAKE)

Provide mutual authentication

# Background: Dragonfly in WPA3 and EAP-pwd

**= Password Authenticated Key Exchange (PAKE)**

Provide mutual authentication

Negotiate session key

# Background: Dragonfly in WPA3 and EAP-pwd

**= Password Authenticated Key Exchange (PAKE)**

Provide mutual authentication

Negotiate session key

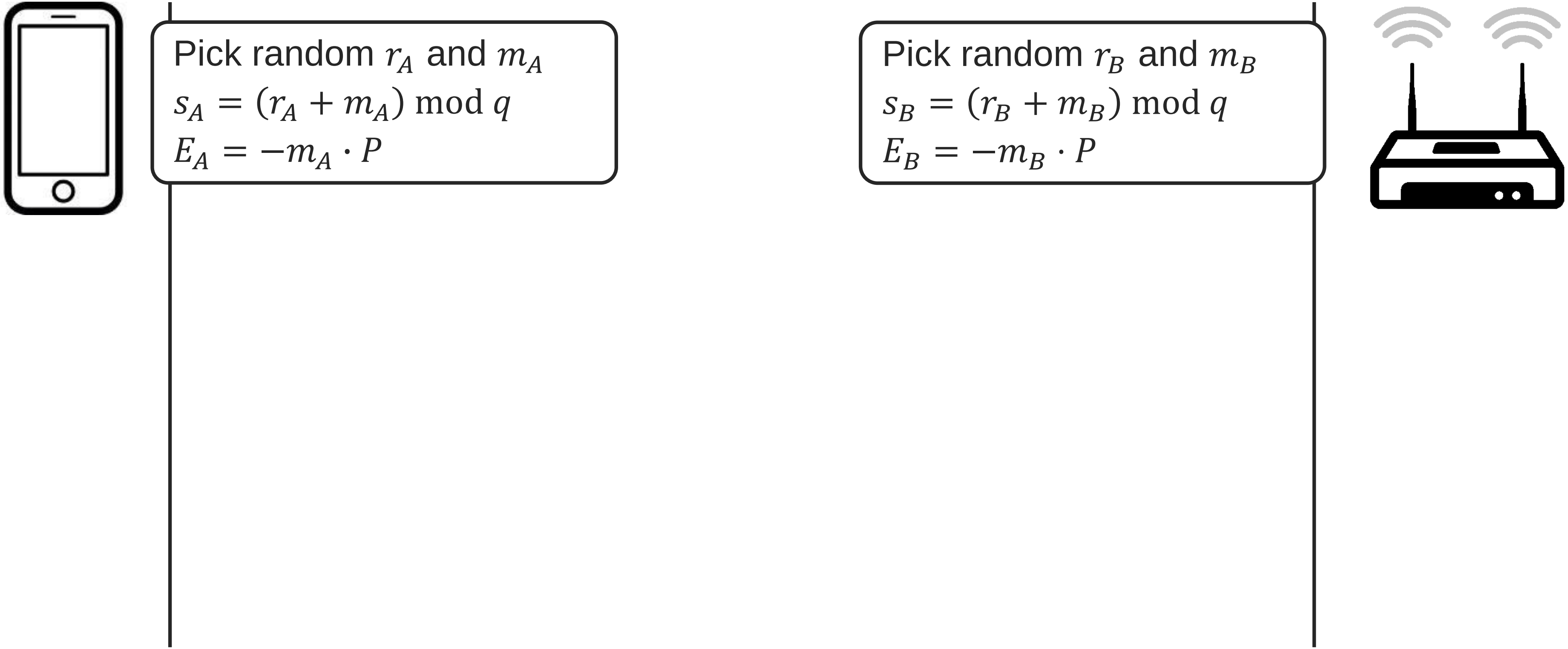
Prevent offline dictionary attacks

# Our Results [V**R** 20]

- Comprehensive analysis of WPA3
  - First attacks against the new protocol
  - Break most of the security guarantees
  - Provide PoC for attacks
- Recommendations for fixing the crypto design
  - Resulting in draft for new protocol version

# The Dragonfly Protocol

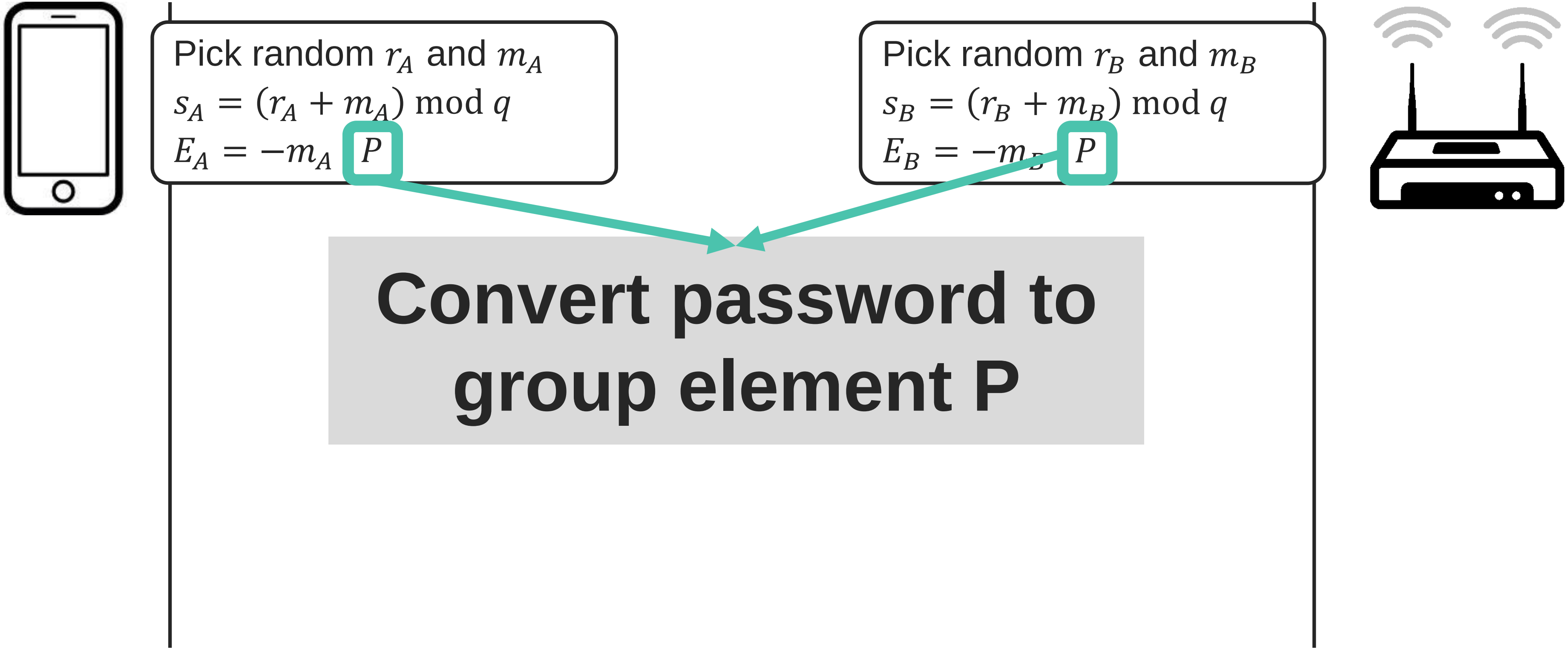
# Dragonfly



Pick random $r_A$ and $m_A$
$s_A = (r_A + m_A) \bmod q$
$E_A = -m_A \cdot P$

Pick random $r_B$ and $m_B$
$s_B = (r_B + m_B) \bmod q$
$E_B = -m_B \cdot P$

# Dragonfly



Pick random $r_A$ and $m_A$

$s_A = (r_A + m_A) \bmod q$

$E_A = -m_A \boxed{P}$

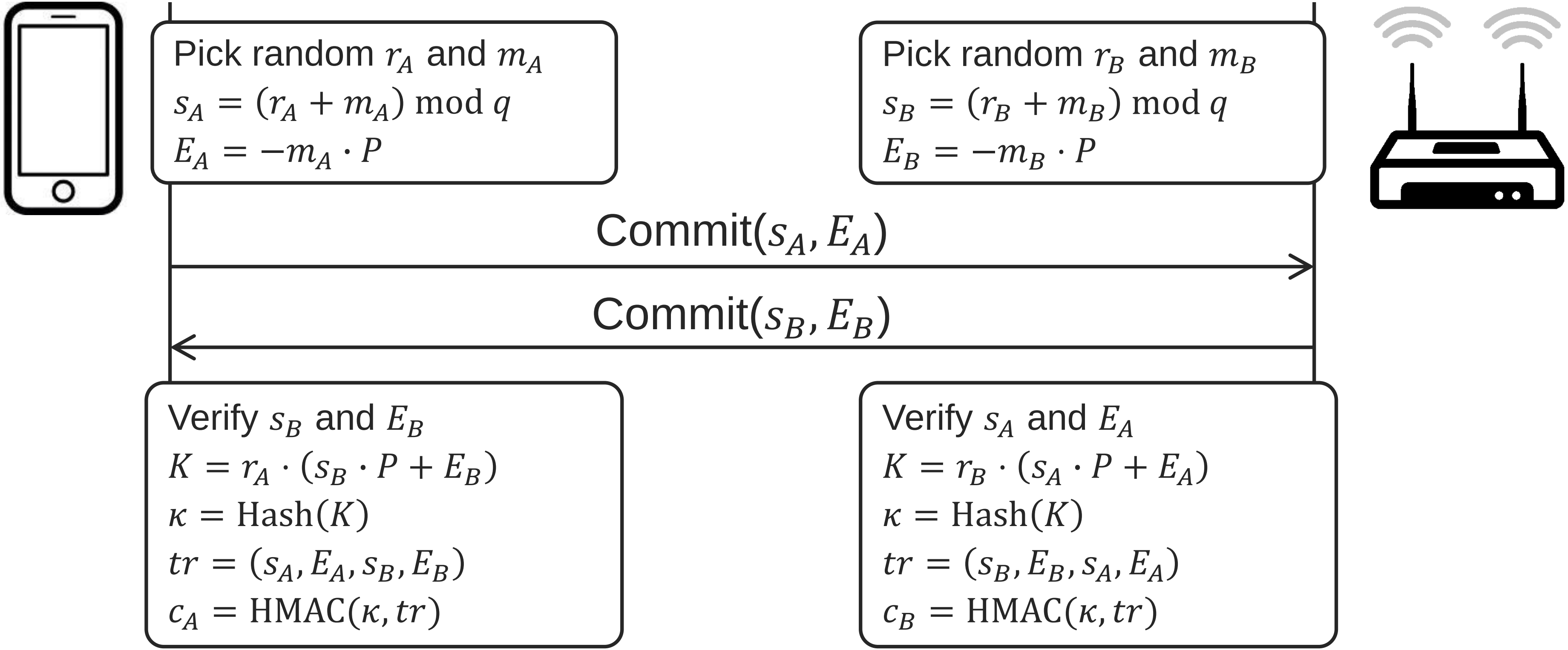
Pick random $r_B$ and $m_B$

$s_B = (r_B + m_B) \bmod q$

$E_B = -m_B \boxed{P}$

**Convert password to group element P**

# Dragonfly



Pick random $r_A$ and $m_A$

$s_A = (r_A + m_A) \bmod q$

$E_A = -m_A \cdot P$

Pick random $r_B$ and $m_B$

$s_B = (r_B + m_B) \bmod q$

$E_B = -m_B \cdot P$

Commit($s_A, E_A$)

Commit($s_B, E_B$)

Verify $s_B$ and $E_B$

$K = r_A \cdot (s_B \cdot P + E_B)$

$\kappa = \text{Hash}(K)$

$tr = (s_A, E_A, s_B, E_B)$

$c_A = \text{HMAC}(\kappa, tr)$

Verify $s_A$ and $E_A$

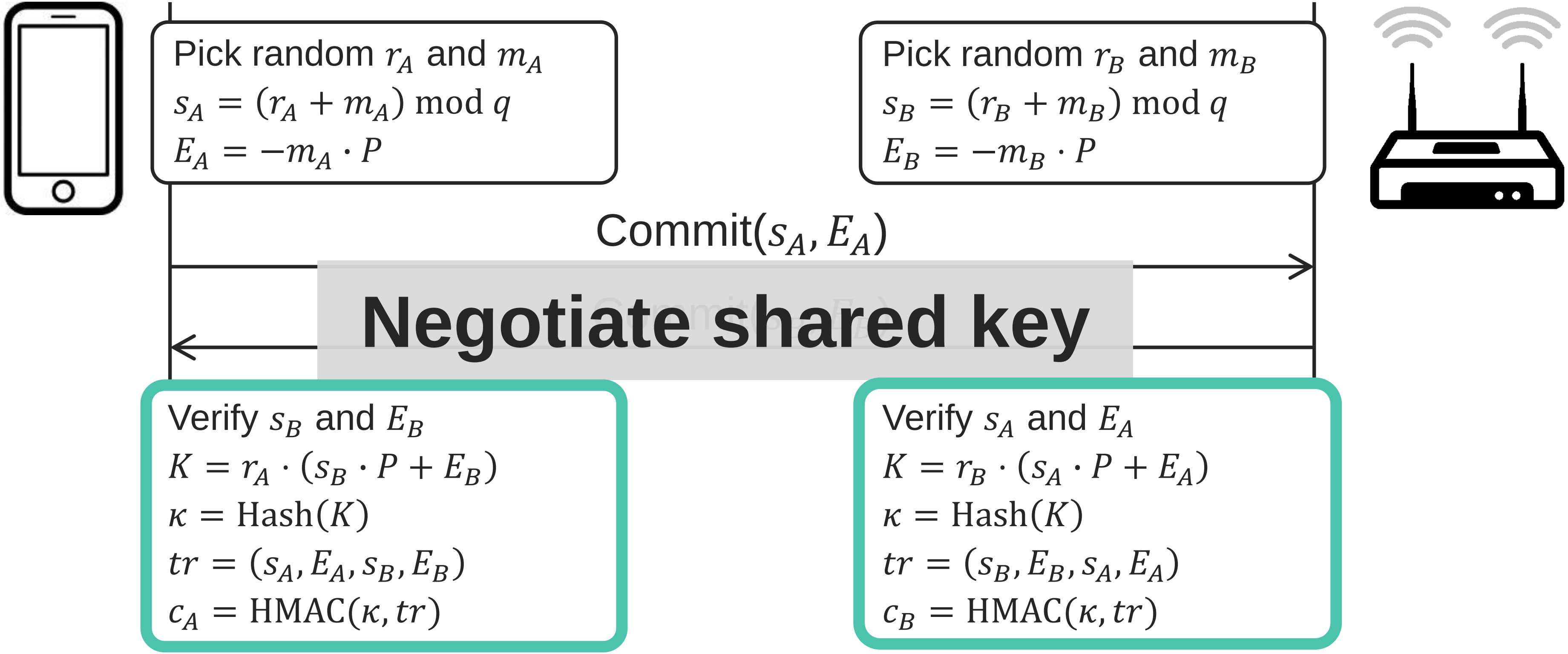
$K = r_B \cdot (s_A \cdot P + E_A)$

$\kappa = \text{Hash}(K)$

$tr = (s_B, E_B, s_A, E_A)$

$c_B = \text{HMAC}(\kappa, tr)$

# Dragonfly



Pick random $r_A$ and $m_A$
$s_A = (r_A + m_A) \bmod q$
$E_A = -m_A \cdot P$

Pick random $r_B$ and $m_B$
$s_B = (r_B + m_B) \bmod q$
$E_B = -m_B \cdot P$

Commit($s_A, E_A$)

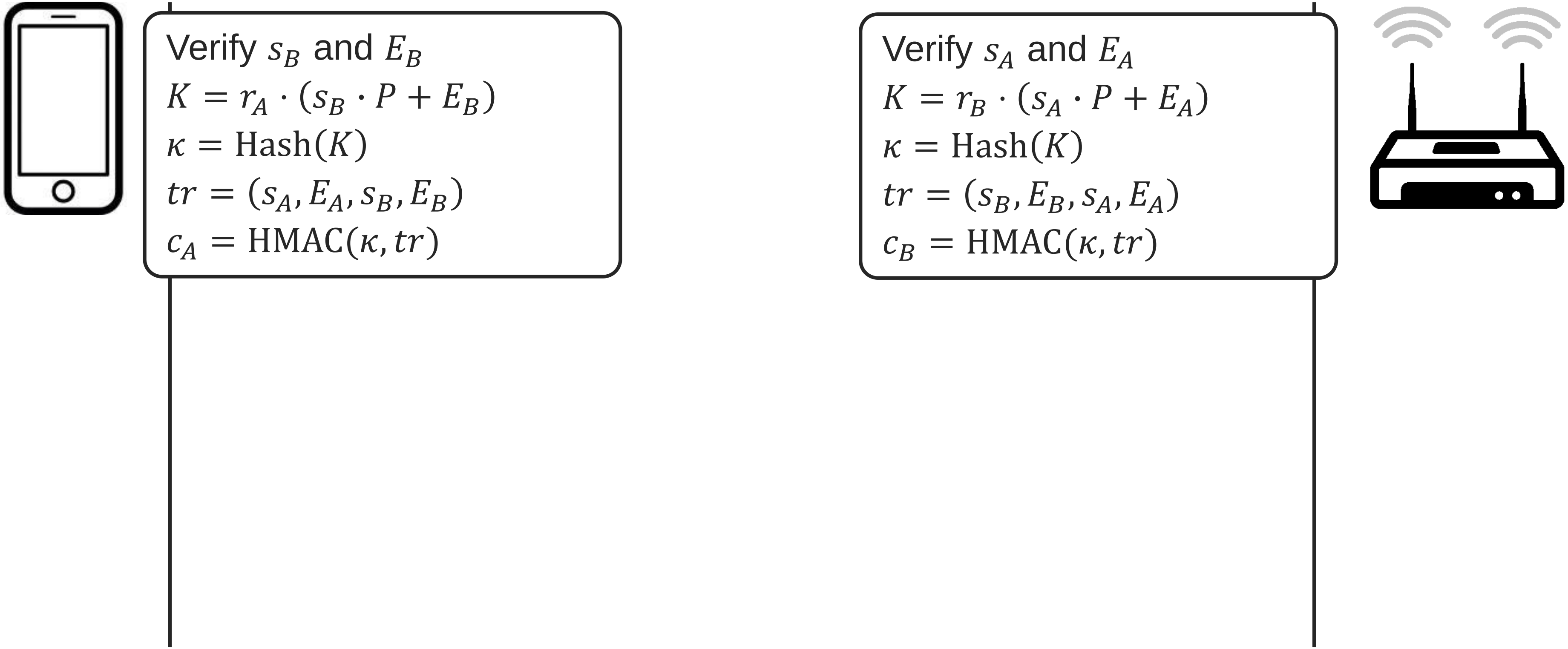
**Negotiate shared key**

Commit($s_B, E_B$)

Verify $s_B$ and $E_B$
$K = r_A \cdot (s_B \cdot P + E_B)$
$\kappa = \mathrm{Hash}(K)$
$tr = (s_A, E_A, s_B, E_B)$
$c_A = \mathrm{HMAC}(\kappa, tr)$

Verify $s_A$ and $E_A$
$K = r_B \cdot (s_A \cdot P + E_A)$
$\kappa = \mathrm{Hash}(K)$
$tr = (s_B, E_B, s_A, E_A)$
$c_B = \mathrm{HMAC}(\kappa, tr)$

# Dragonfly



Verify $s_B$ and $E_B$

$K = r_A \cdot (s_B \cdot P + E_B)$

$\kappa = \mathrm{Hash}(K)$

$tr = (s_A, E_A, s_B, E_B)$

$c_A = \mathrm{HMAC}(\kappa, tr)$

Verify $s_A$ and $E_A$

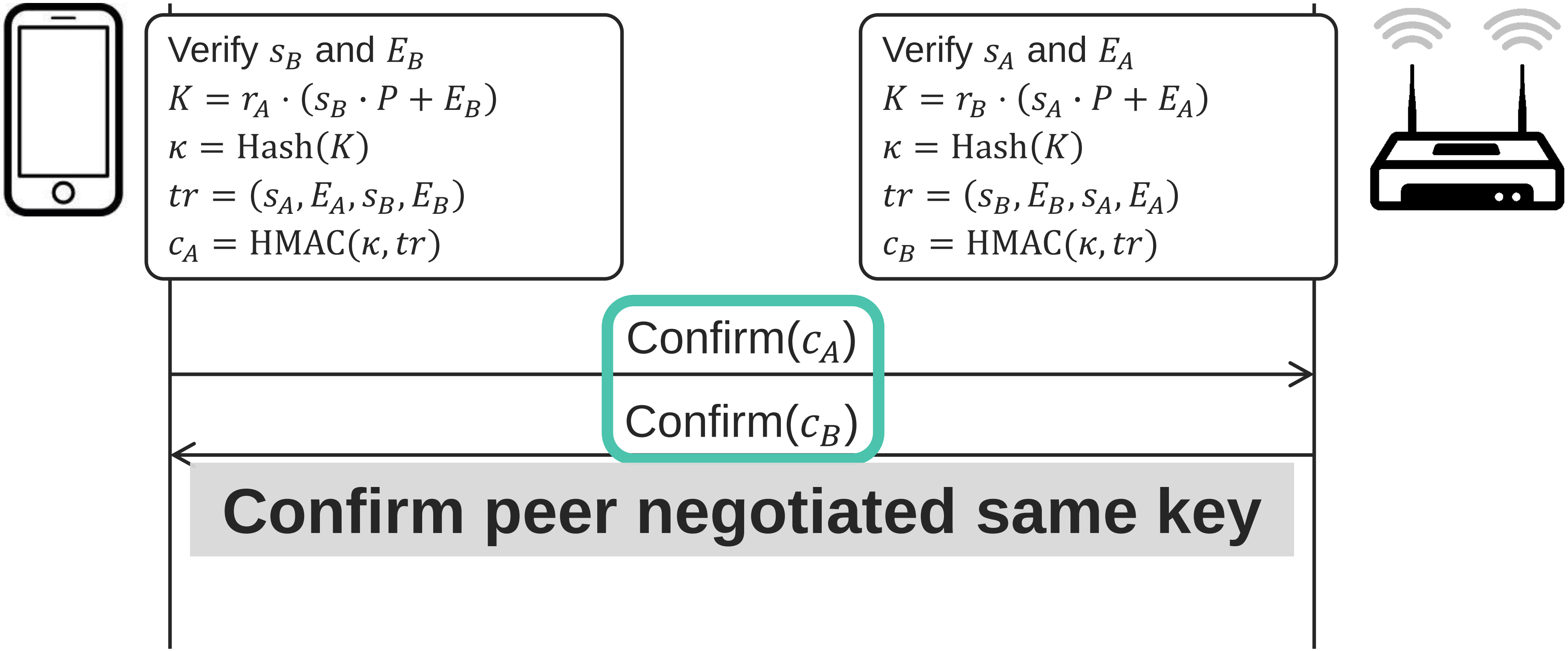
$K = r_B \cdot (s_A \cdot P + E_A)$

$\kappa = \mathrm{Hash}(K)$

$tr = (s_B, E_B, s_A, E_A)$

$c_B = \mathrm{HMAC}(\kappa, tr)$

# Dragonfly



Verify $s_B$ and $E_B$

$K = r_A \cdot (s_B \cdot P + E_B)$

$\kappa = \mathrm{Hash}(K)$

$tr = (s_A, E_A, s_B, E_B)$

$c_A = \mathrm{HMAC}(\kappa, tr)$

Verify $s_A$ and $E_A$

$K = r_B \cdot (s_A \cdot P + E_A)$

$\kappa = \mathrm{Hash}(K)$

$tr = (s_B, E_B, s_A, E_A)$

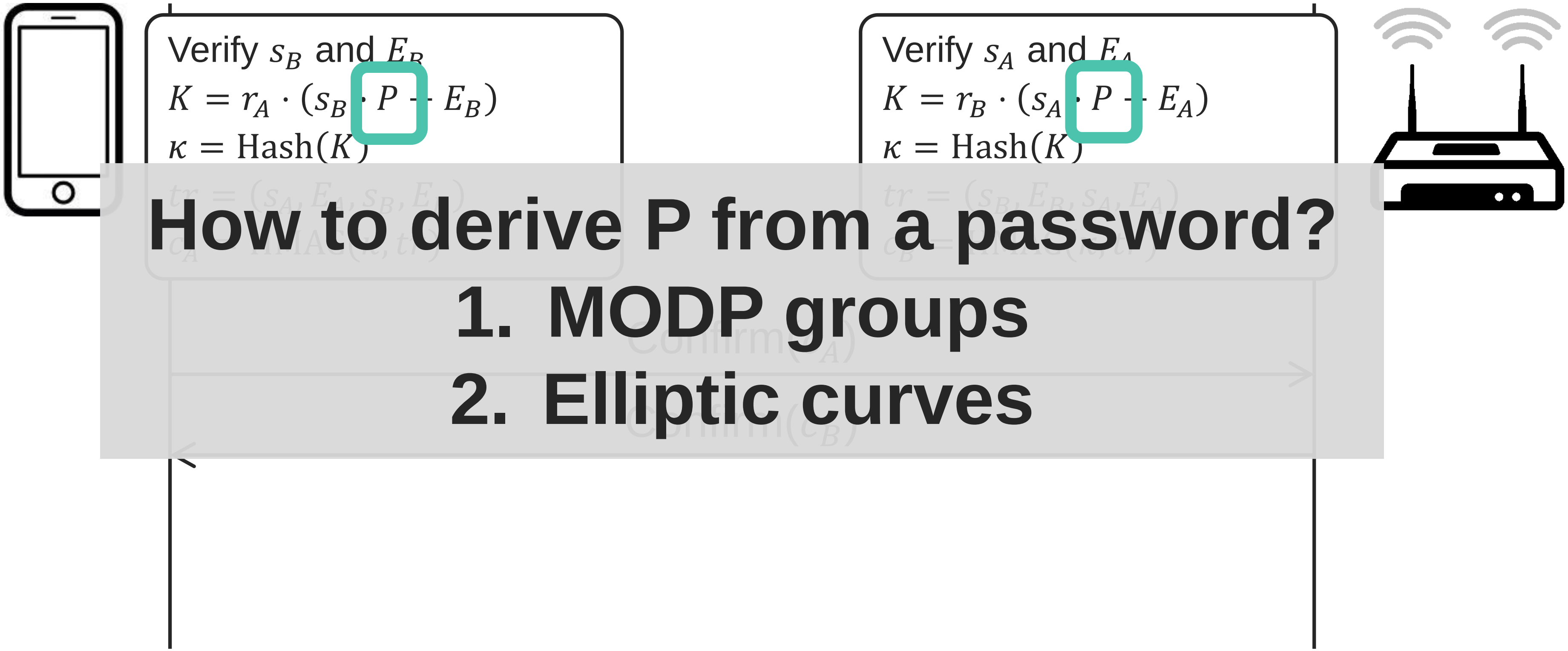
$c_B = \mathrm{HMAC}(\kappa, tr)$

Confirm($c_A$)

Confirm($c_B$)

**Confirm peer negotiated same key**

# Dragonfly



Verify $s_B$ and $E_B$

$K = r_A \cdot (s_B \cdot \boxed{P} + E_B)$

$\kappa = \text{Hash}(K)$

$tr = (s_A, E_A, s_B, E_B)$

$c_A = \text{HMAC}(\kappa, tr)$

Verify $s_A$ and $E_A$

$K = r_B \cdot (s_A \cdot \boxed{P} + E_A)$

$\kappa = \text{Hash}(K)$

$tr = (s_B, E_B, s_A, E_A)$

$c_B = \text{HMAC}(\kappa, tr)$

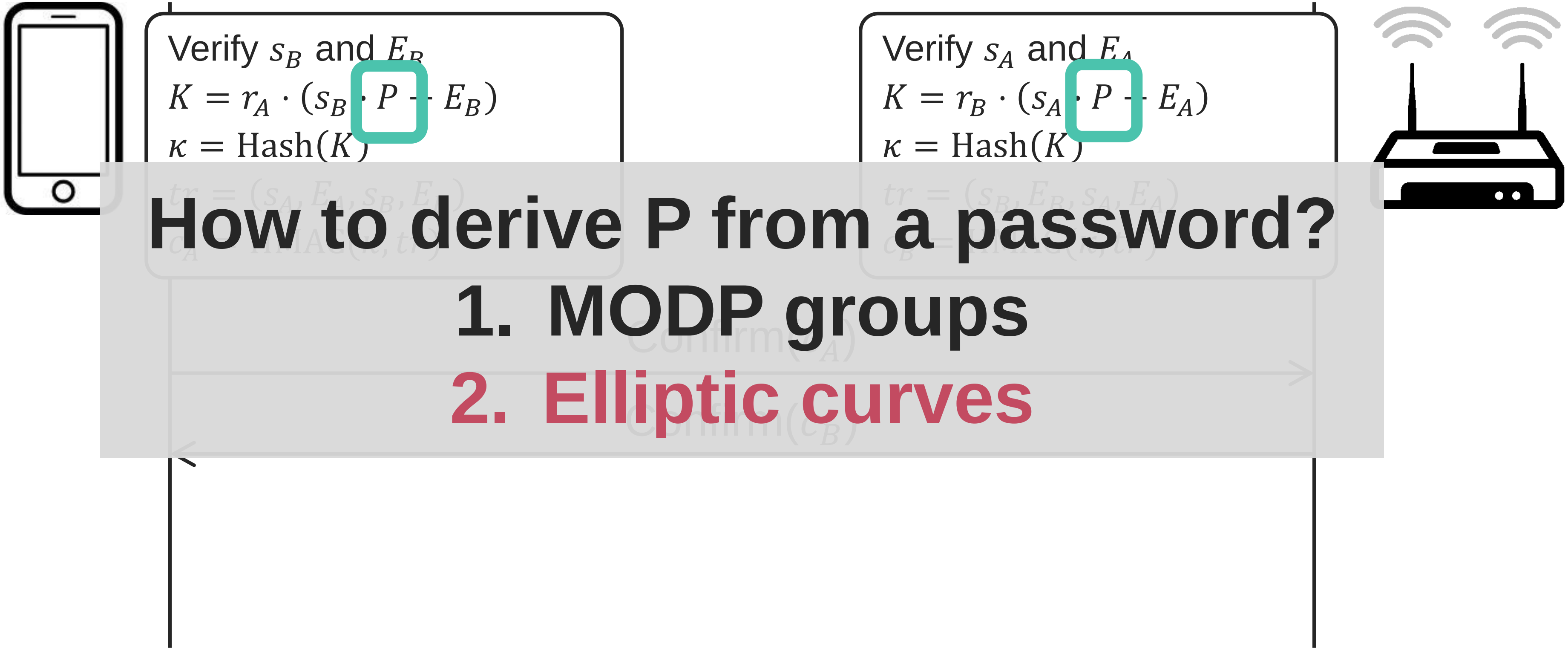
Confirm($c_A$)

Confirm($c_B$)

**How to derive P from a password?**

1. **MODP groups**
2. **Elliptic curves**

# Dragonfly



Verify $s_B$ and $E_B$

$K = r_A \cdot (s_B \cdot \boxed{P} + E_B)$

$\kappa = \text{Hash}(K)$

$tr = (s_A, E_A, s_B, E_B)$

$c_A = \text{HMAC}(\kappa, tr)$

Verify $s_A$ and $E_A$

$K = r_B \cdot (s_A \cdot \boxed{P} + E_A)$

$\kappa = \text{Hash}(K)$

$tr = (s_B, E_B, s_A, E_A)$

$c_B = \text{HMAC}(\kappa, tr)$

Confirm($c_A$)

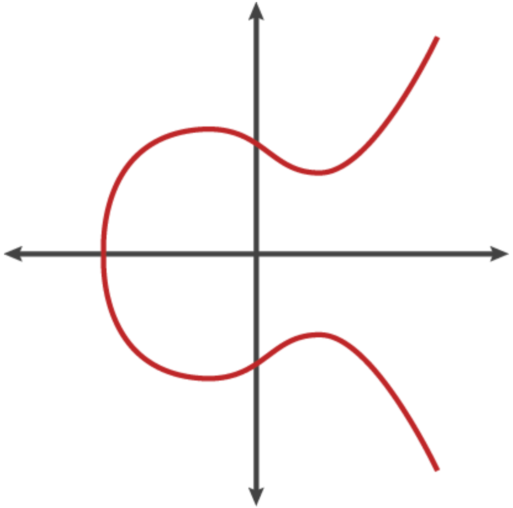
Confirm($c_B$)

**How to derive P from a password?**

1. **MODP groups**
2. **Elliptic curves**

# Elliptic Curves

- Operations performed on points (x, y) where:
  - $x < p$ and $y < p$ with $p$ a prime
  - $y^2 = x^3 + ax + b \bmod p$ must hold



- Need to convert password pw to point P (x,y) on the curve

# Hash2Curve

- Hash2Curve is a hash function *H* such that:
  - H is a RO mapping from arbitrary strings into the full group domain:

$$H : \{0,1\}^* \rightarrow \mathbb{G}$$

$$g \in \mathbb{G}, \{x, y\} \in \mathbb{Z}_q, X = H(x), Y = H(y)$$

$$X = g^?, X = Y^?$$

# Hash2Curve

- Hash2Curve is a hash function *H* such that:
  - H is a RO mapping from arbitrary strings into the full group domain:

$$H : \{0,1\}^* \rightarrow \mathbb{G}$$

$$g \in \mathbb{G}, \{x, y\} \in \mathbb{Z}_q, X = H(x), Y = H(y)$$

$$X = g^?, X = Y^?$$

- For WPA3 it was decided that point P is

$$P = H(pw, (MACaddress)_1, (MACaddress)_2)$$

# Hash-to-curve: EAP-pwd

```
for (counter = 1; counter < 40; counter++)
    x = hash(pw, addr1, addr2, counter)
    if x >= p: continue
    if square_root_exists(x) and not P:
        return (x, $\sqrt{x^3 + ax + b}$)
```

# Hash-to-curve: EAP-pwd

```
for (counter = 1; counter < 40; counter++)
    x = hash(pw, addr1, addr2, counter)
    if x >= p: continue
    if square_root_exists(x) and not P:
        return (x, $\boxed{\sqrt{x^3 + ax + b}}$)
```

**Half of x values aren’t on the curve**

# Hash-to-curve: EAP-pwd

```
for (counter = 1; counter < 40; counter++)
    x = hash(pw, addr1, addr2, counter)
    if x >= p: continue
    if square_root_exists(x) and not P:
        return (x, $\sqrt{x^3 + ax + b}$)
```

# Hash-to-curve: EAP-pwd

```
for (counter = 1; counter < 40; counter++)
    x = hash(pw, addr1, addr2, counter)
    if x >= p: continue

    if square_root_exists(x) and not P:
        return (x, $\sqrt{x^3 + ax + b}$)
```

**#iterations depends on password**

**(and public MAC addresses)**

# Hash-to-curve: EAP-pwd

```
for (counter = 1; counter < 40; counter++)
    x = hash(pw, addr1, addr2, counter)
    if x >= p: continue
    if square_root_exists(x) and not P:
        return (x, $\sqrt{x^3 + ax + b}$)
```

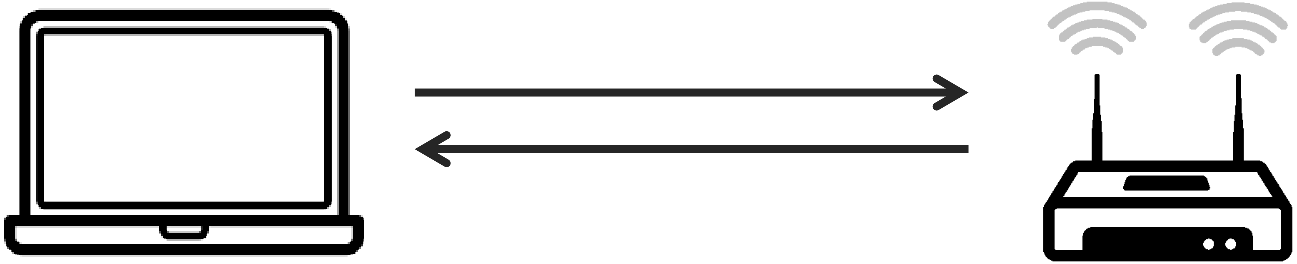
**#iterations depends on password**

**(and public MAC addresses)**

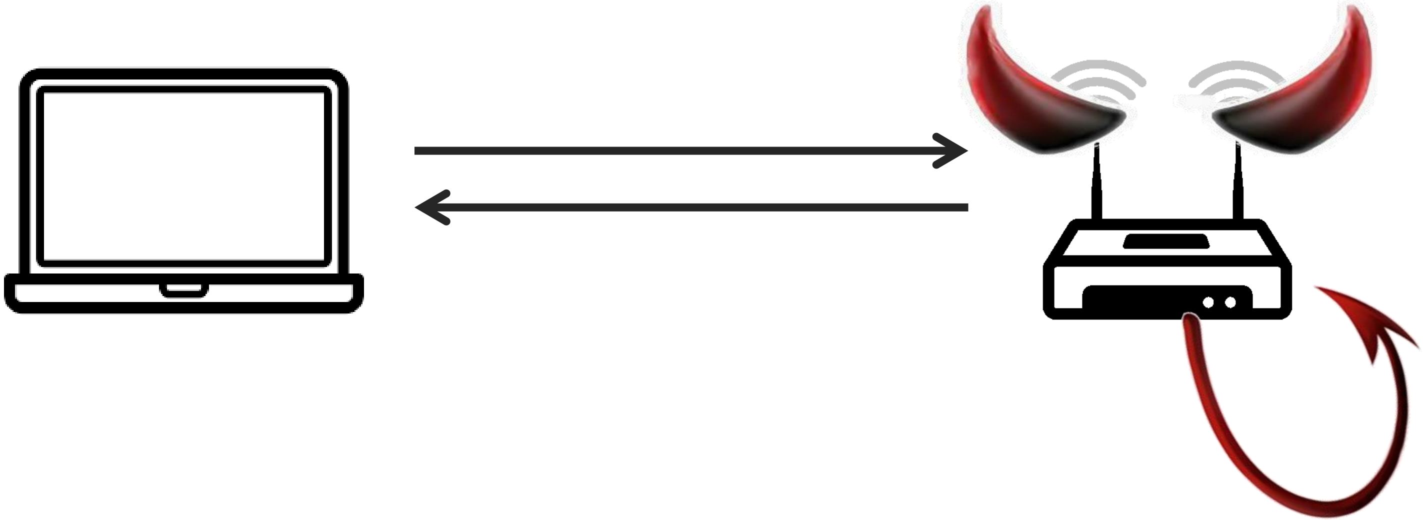
**No timing leak countermeasures,**

**despite warnings by IETF & CFRG!**

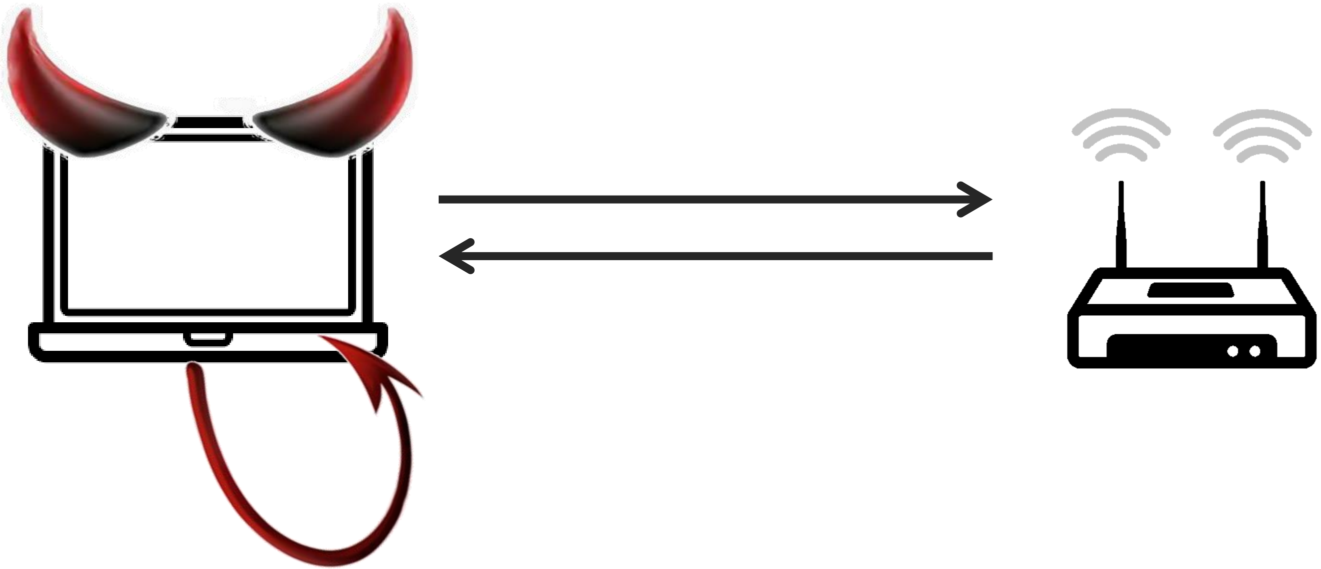
# Attacking Clients



# Attacking Clients



# Attacking Access Points



# Leaked information: #iterations needed

| Client address | addrA |
|---|---|
| Measured | 🟩<br>🟩 |

# Leaked information: #iterations needed

| Client address | addrA |
|---|---|
| Measured | 🟩<br>🟩 |
| Password 1 | 🟩 |
| Password 2 | 🟩<br>🟩 |
| Password 3 | 🟩<br>🟩 |

# Leaked information: #iterations needed

| Client address | addrA |
|---|---|
| Measured | 🟩<br>🟩 |
| ~~Password 1~~ | 🟩 |
| Password 2 | 🟩<br>🟩 |
| Password 3 | 🟩<br>🟩 |

# What information is leaked?

```
for (counter = 1; counter < 40; counter++)
    x = hash(pw, addr1, addr2, counter)
    if x >= p: continue
    if square_root_exists(x) and not P:
        return (x, $\sqrt{x^3 + ax + b}$)
```

# What information is leaked?

```
for (counter = 1; counter < 40; counter++)
    x = hash(pw, addr1, addr2, counter)
    if x >= p: continue
    if square_root_exists(x) and not P:
        return (x, $\sqrt{x^3 + ax + b}$)
```

**Spoof client address to obtain different execution & leak new data**

# Leaked information: #iterations needed

| Client address | addrA | addrB |
|---|---|---|
| Measured | 🟩<br>🟩 | 🟩 |
| ~~Password 1~~ | 🟩 | |
| Password 2 | 🟩<br>🟩 | |
| Password 3 | 🟩<br>🟩 | |

# Leaked information: #iterations needed

| Client address | addrA | addrB |
|---|---|---|
| Measured | 🟩<br>🟩 | 🟩 |
| ~~Password 1~~ | 🟩 | 🟩<br>🟩<br>🟩 |
| Password 2 | 🟩<br>🟩 | 🟩<br>🟩 |
| Password 3 | 🟩<br>🟩 | 🟩 |

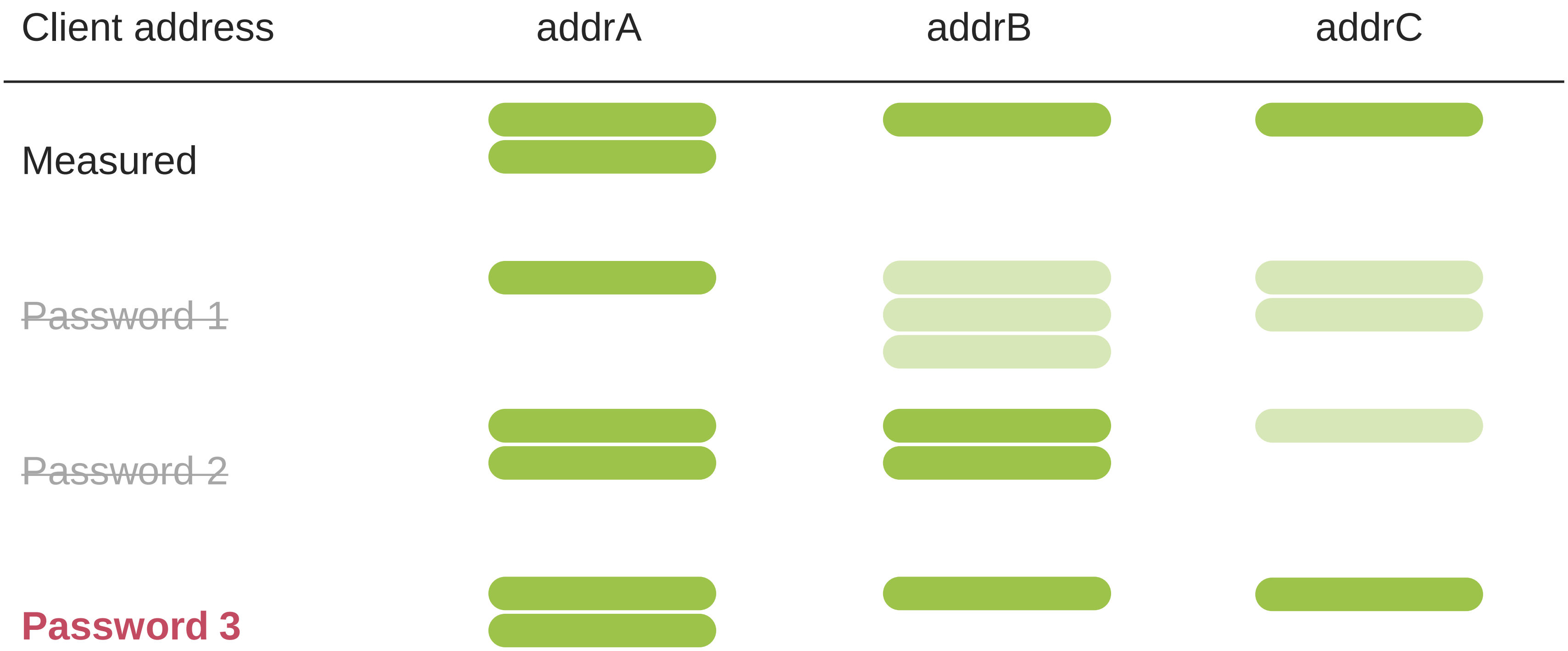
# Leaked information: #iterations needed

| Client address | addrA | addrB |
|---|---|---|
| Measured | 🟩<br>🟩 | 🟩 |
| ~~Password 1~~ | 🟩 | 🟩<br>🟩<br>🟩 |
| ~~Password 2~~ | 🟩<br>🟩 | 🟩<br>🟩 |
| Password 3 | 🟩<br>🟩 | 🟩 |

# Leaked information: #iterations needed

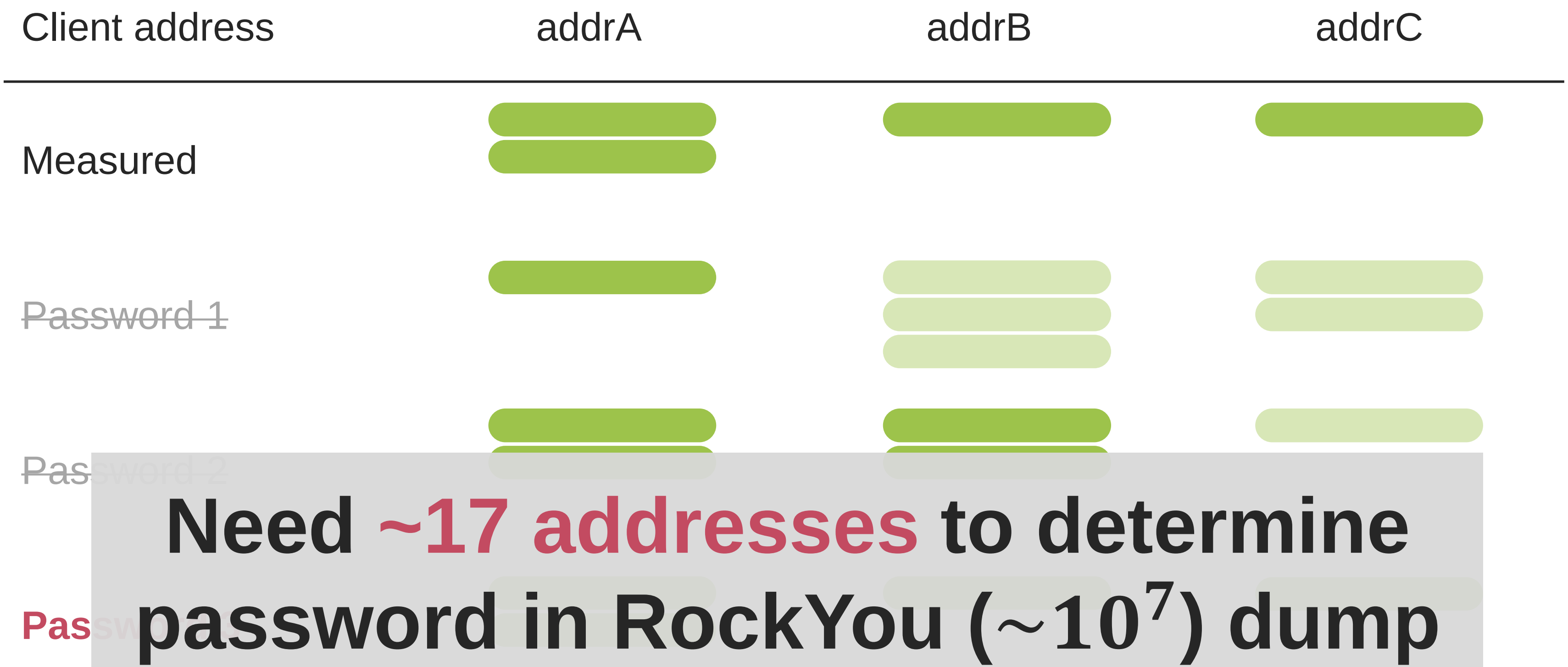
| Client address | addrA | addrB | addrC |
|---|---|---|---|
| Measured | 🟩<br>🟩 | 🟩 | 🟩 |
| ~~Password 1~~ | 🟩 | ▨<br>▨<br>▨ | ▨<br>▨ |
| ~~Password 2~~ | 🟩<br>🟩 | 🟩<br>🟩 | ▨ |
| Password 3 | 🟩<br>🟩 | 🟩 | 🟩 |

# Leaked information: #iterations needed



| Client address | addrA | addrB | addrC |
|---|---|---|---|
| Measured | 🟩<br>🟩 | 🟩 | 🟩 |
| ~~Password 1~~ | 🟩 | □<br>□<br>□ | □<br>□ |
| ~~Password 2~~ | 🟩<br>🟩 | 🟩<br>🟩 | □ |
| **Password 3** | 🟩<br>🟩 | 🟩 | 🟩 |

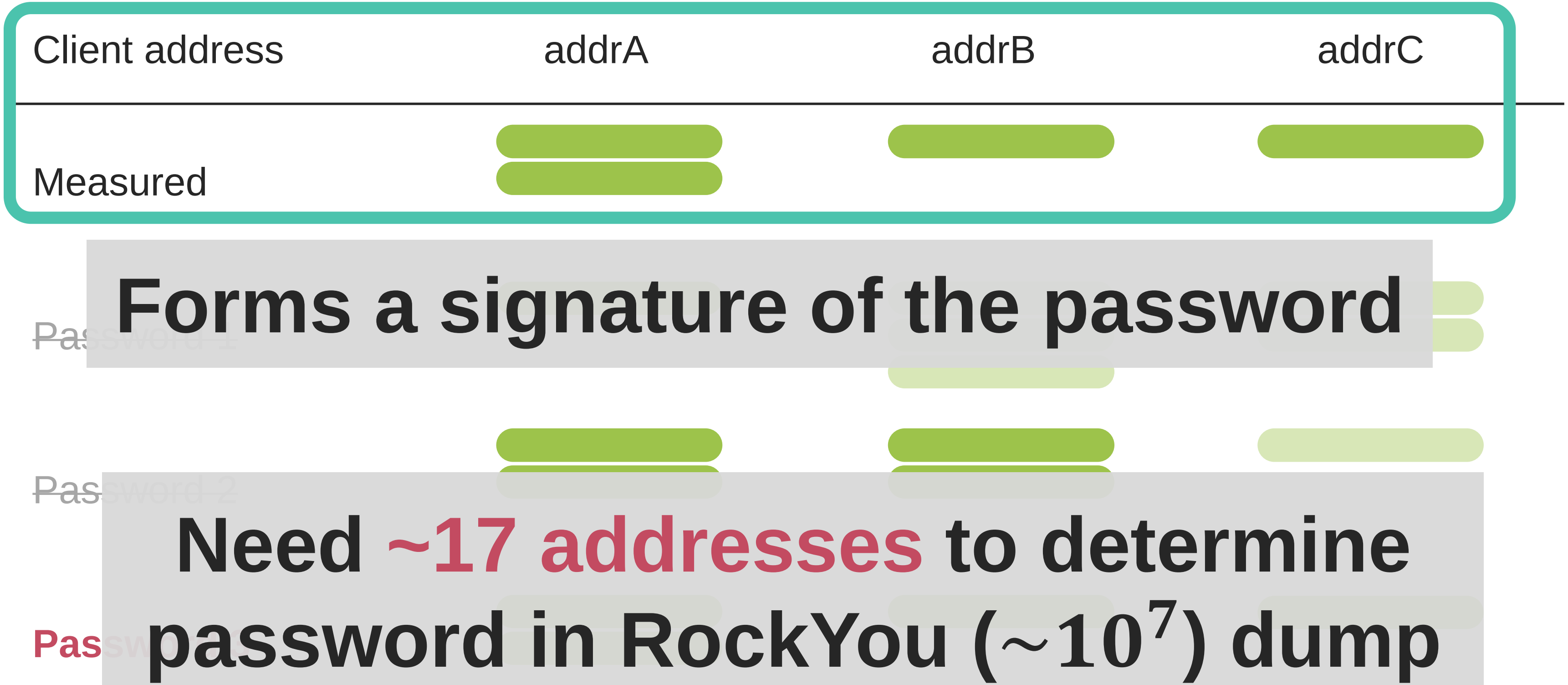
# Leaked information: #iterations needed



| Client address | addrA | addrB | addrC |
|---|---|---|---|
| Measured | 🟩<br>🟩 | 🟩 | 🟩 |
| ~~Password 1~~ | 🟩 | ▨<br>▨<br>▨ | ▨<br>▨ |
| ~~Password 2~~ | 🟩<br>🟩 | 🟩<br>🟩 | ▨ |
| Pas | | | |

**Need ~17 addresses to determine password in RockYou ($\sim 10^7$) dump**

# Leaked information: #iterations needed



Client address addrA addrB addrC

Measured

~~Password 1~~

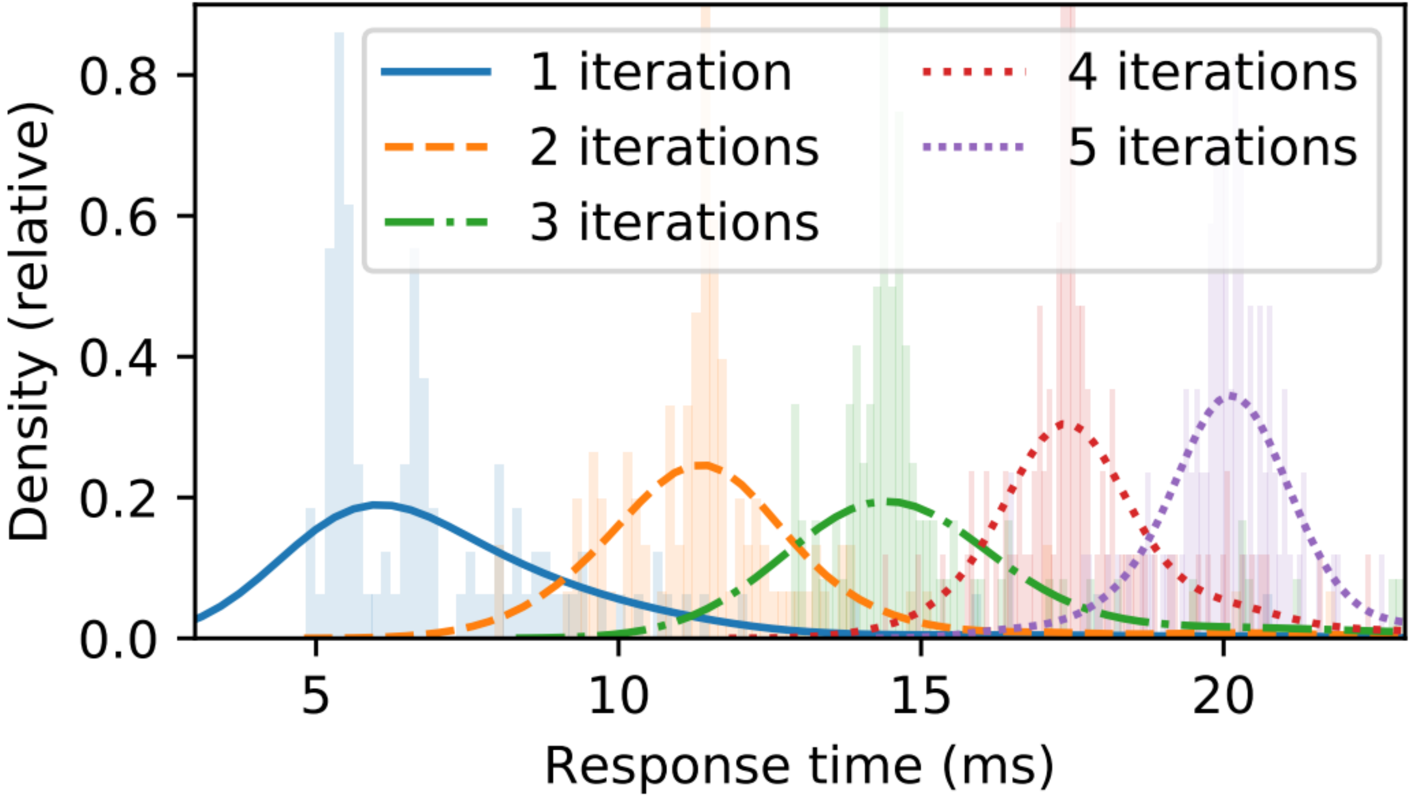
~~Password 2~~

Passw

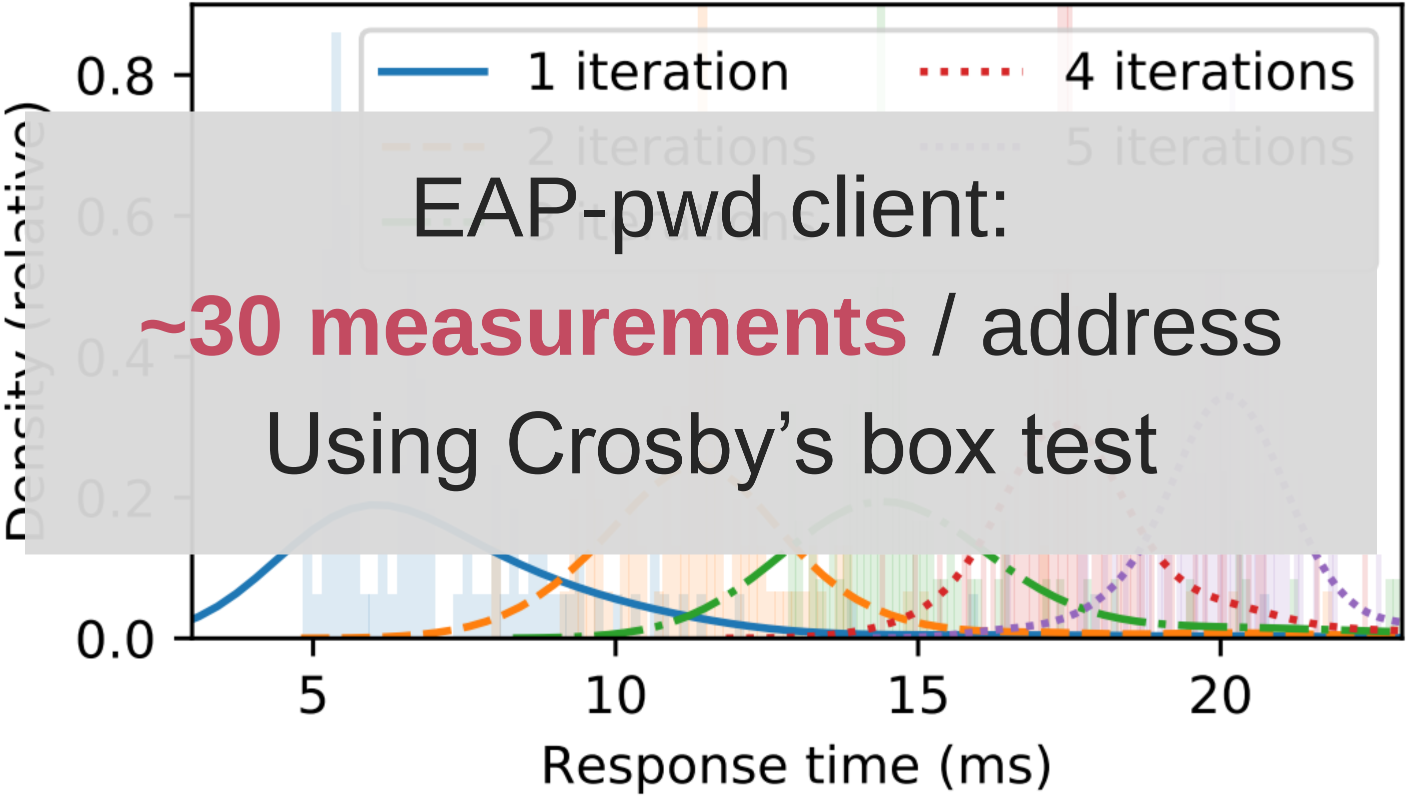
**Forms a signature of the password**

**Need ~17 addresses to determine password in RockYou ($\sim10^7$) dump**

# Raspberry Pi 1 B+: differences are measurable



# Raspberry Pi 1 B+: differences are measurable



EAP-pwd client:

**~30 measurements** / address

Using Crosby’s box test

# Hash-to-curve: WPA3

```
for (counter = 1; counter < 40; counter++)
    x = hash(pw, counter, addr1, addr2)
    if x >= p: continue
    if square_root_exists(x) and not P:
        P = (x, $\sqrt{x^3 + ax + b}$)
        pw = rand()
return P
```

# Hash-to-curve: WPA3

```
for (counter = 1; counter < 40; counter++)
    x = hash(pw, counter, addr1, addr2)
    if x >= p: continue
    if square_root_exists(x) and not P:
        P = (x, $\sqrt{x^3 + ax + b}$)
        pw = rand()
return P
```

**WPA3: always do 40 loops & return first P**

# Hash-to-curve: WPA3

```
for (counter = 1; counter < 40; counter++)
    x = hash(pw, counter, addr1, addr2)
    if x >= p: continue
    if square_root_exists(x) and not P:
        P = (x, $\sqrt{x^3 + ax + b}$)
return P
```

**Blinded constant time**

**square root test**

# Hash-to-curve: WPA3

```
for (counter = 1; counter < 40; counter++)
    x = hash(pw, counter, addr1, addr2)
    if x >= p: continue
    if square_root_exists(x) and not P:
        P = (x, $\sqrt{x^3 + ax + b}$)
        pw = rand()
return P
```

**Extra iterations based on random password**

# Hash-to-curve: WPA3

```
for (counter = 1; counter < 40; counter++)
    x = hash(pw, counter, addr1, addr2)
    if x >= p: continue
    if square_root_exists(x) and not P:
        P = (x, $\sqrt{x^3 + ax + b}$)
        pw = rand()
return P
```

**Are we Safe?**

# Hash-to-curve: WPA3

```
for (counter = 1; counter < 40; counter++)
    x = hash(pw, counter, addr1, addr2)
    if x >= p: continue
    if square_root_exists(x) and not P:
        P = (x, $\sqrt{x^3 + ax + b}$)
        pw = rand()
return P
```

**Truncate to size of prime p**

# Hash-to-curve: WPA3

```
for (counter = 1; counter < 40; counter++)
    x = hash(pw, counter, addr1, addr2)
    if x >= p: continue
    if square_root_exists(x) and not P:
        P = (x, $\sqrt{x^3 + ax + b}$)
        pw = rand()
return P
```

**Brainpool:** $p =$ 0xA9FB57DBA1EEA9BC…

→ **High chance that x >= p**

# Hash-to-curve: WPA3

```
for (counter = 1; counter < 40; counter++)
    x = hash(pw, counter, addr1, addr2)
    if x >= p: continue
    if square_root_exists(x) and not P:
        P = (x, $\sqrt{x^3 + ax + b}$)
        pw = rand()
return P
```

**= rejection sampling**

# Hash-to-curve: WPA3

```
for (counter = 1; counter < 40; counter++)
    x = hash(pw, counter, addr1, addr2)
    if x >= p: continue
    if square_root_exists(x) and not P:
        P = (x, $\sqrt{x^3 + ax + b}$)
        pw = rand()
return P
```

**Code may be skipped**

# Hash-to-curve: WPA3

```
for (counter = 1; counter < 40; counter++)
    x = hash(pw, counter, addr1, addr2)
    if x >= p: continue
    if square_root_exists(x) and not P:
        P = (x, $\sqrt{x^3 + ax + b}$)
        pw = rand()
return P
```

**#Times skipped depends on password**

# Hash-to-curve: WPA3

```
for (counter = 1; counter < 40; counter++)
    x = hash(pw, counter, addr1, addr2)
    if x >= p: continue
    if square_root_exists(x) and not P:
        P = (x, $\sqrt{x^3 + ax + b}$)
        pw = rand()
return P
```

**#Times skipped depends on password**

**& random password in extra itreations**

# Hash-to-curve: WPA3

```
for (counter = 1; counter < 40; counter++)
    x = hash(pw, counter, addr1, addr2)
    if x >= p: continue
    if square_root_exists(x) and not P:
        P = (x, $\sqrt{x^3 + ax + b}$)
        pw = rand()
return P
```

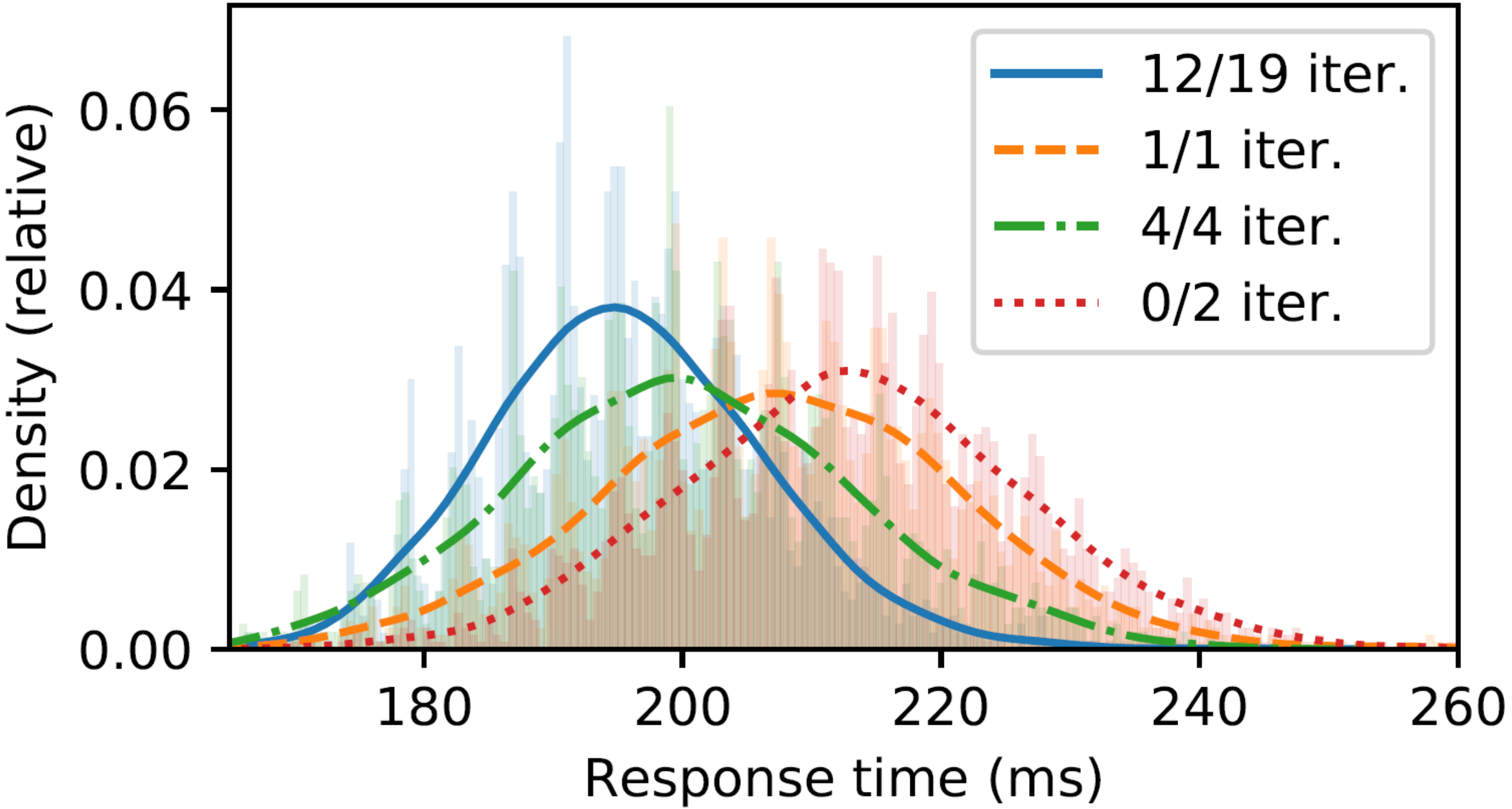
**Variance ~ when password element was found**

# Hash-to-curve: WPA3

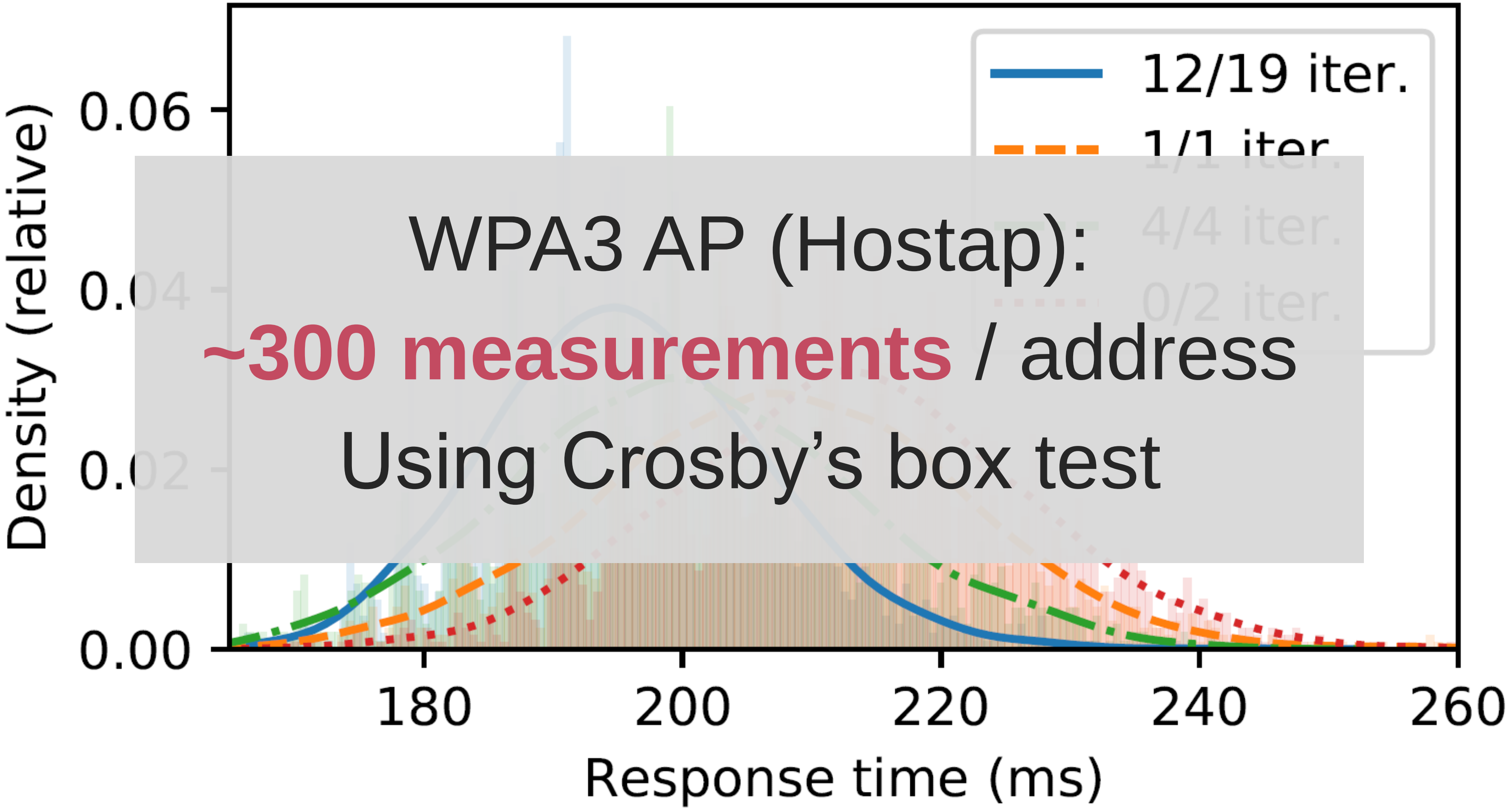
```
for (counter = 1; counter < 40; counter++)
    x = hash(pw, counter, addr1, addr2)
    if x >= p: continue
    if square_root_exists(x) and not P:
        P = (x, $\sqrt{x^3 + ax + b}$)
        pw = rand()
return P
```

**Variance ~ when password element was found**
**Average ~ when found & #iterations code skipped**

# Raspberry Pi 1 B+



# Raspberry Pi 1 B+



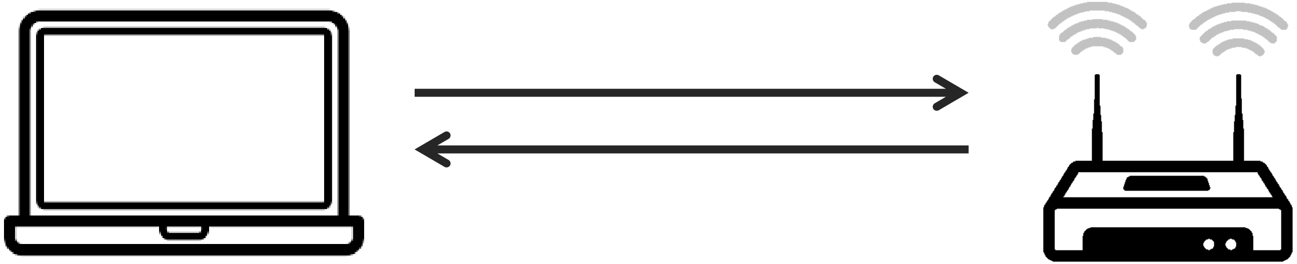
WPA3 AP (Hostap):

**~300 measurements** / address

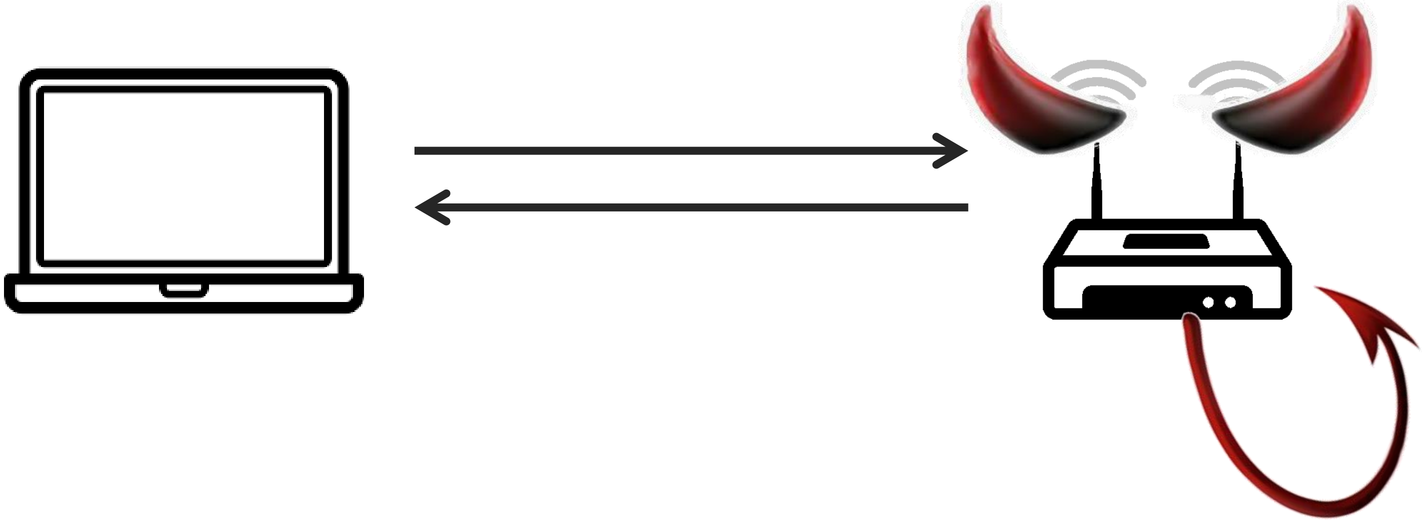
Using Crosby's box test

# Cache Attacks

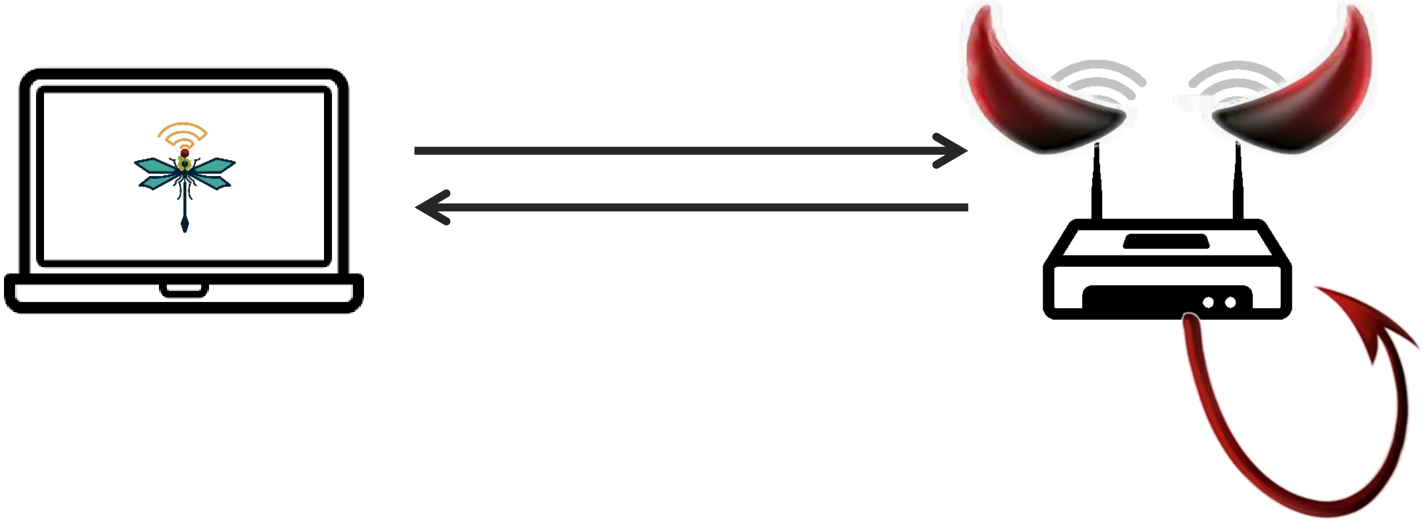
# Threat Model



# Threat Model



# Threat Model



# Cache attack on NIST curves

```
for (counter = 1; counter < 40; counter++)
    x = hash(pw, counter, addr1, addr2)
    if x >= p: continue
    if square_root_exists(x) and not P:
        P = (x, $\sqrt{x^3 + ax + b}$)
        pw = rand()
return P
```

**NIST:** $p = \text{0x0xFFFFFFFF00000001000}\ldots$

→ **Negligible chance that x >= p**

# Cache attack on NIST curves

```
for (counter = 1; counter < 40; counter++)
    x = hash(pw, counter, addr1, addr2)
    if x >= p: continue
    if square_root_exists(x) and not P:
        P = (x, $\sqrt{x^3 + ax + b}$)
        pw = rand()
return P
```

**NIST curves: use Flush+Reload to detect when code is executed**

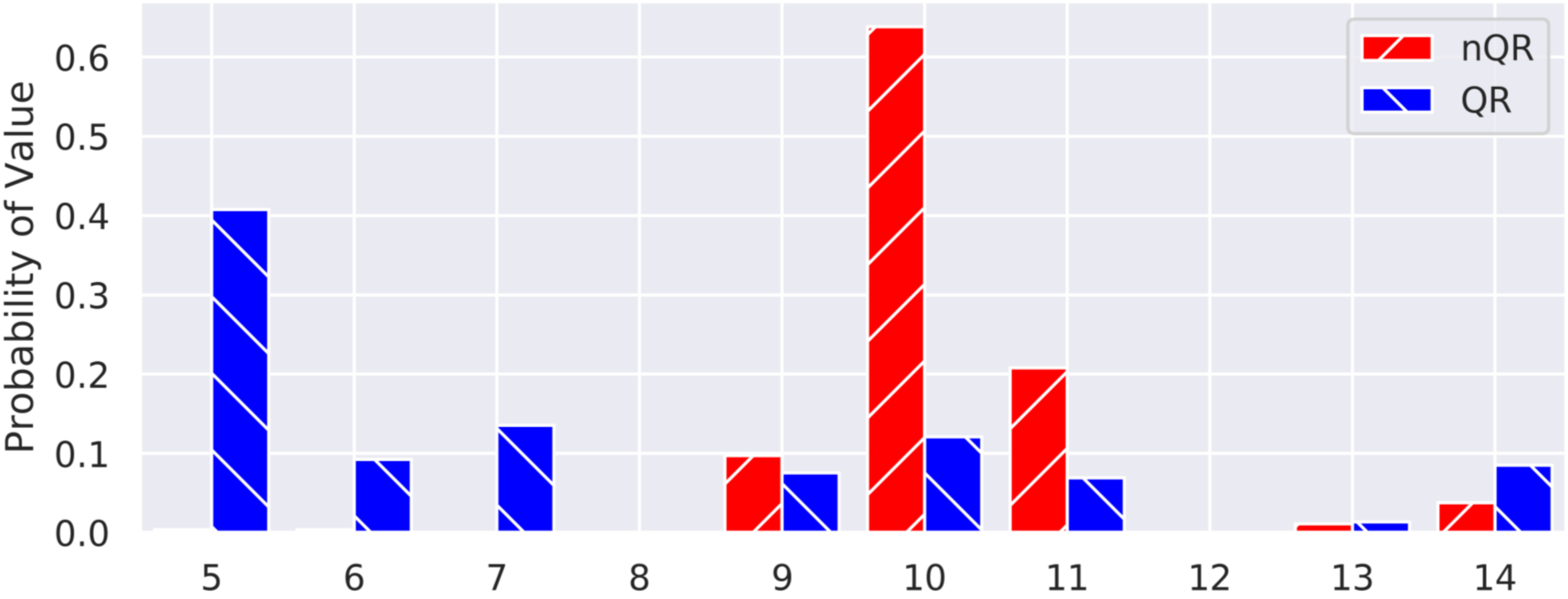
# Cache attack on NIST curves

**Monitor using Flush+Reload to know in which iteration we are**

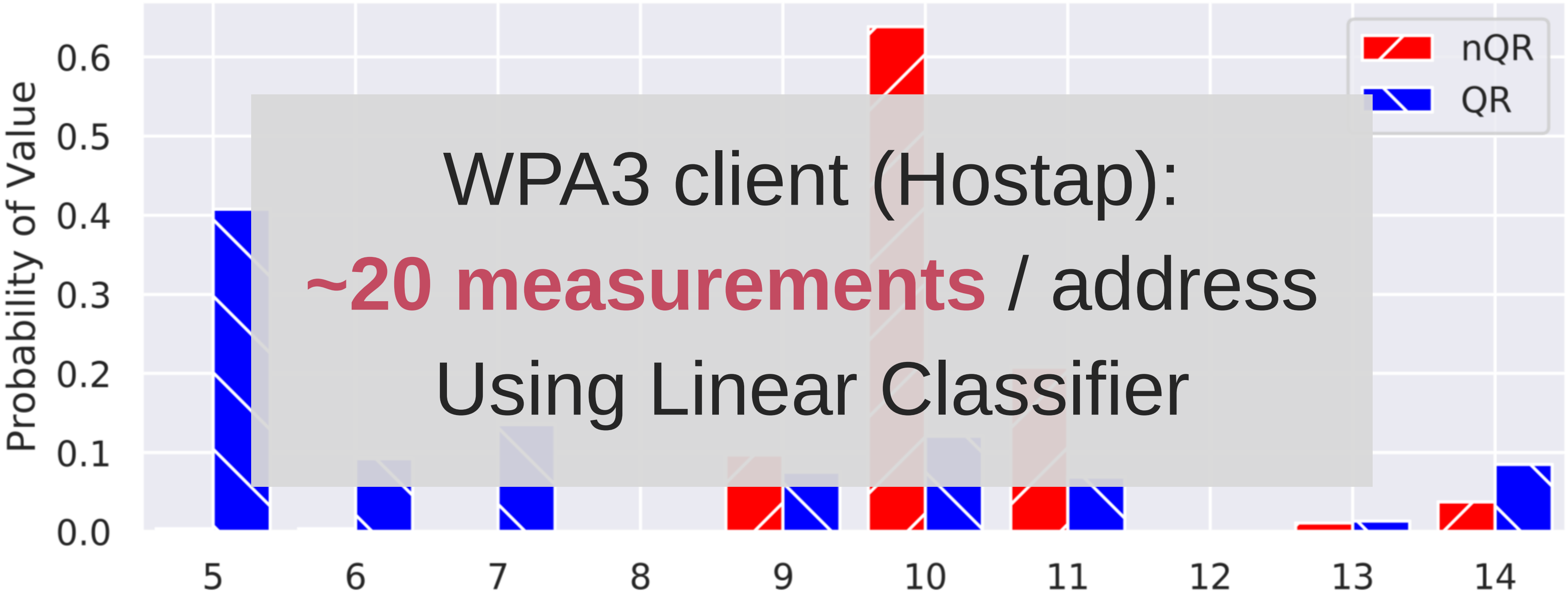
```
for (counter = 1; counter < 40; counter++)
    x = hash(pw, counter, addr1, addr2)
    if x >= p: continue
    if square_root_exists(x) and not P:
        P = (x, $\sqrt{x^3 + ax + b}$)
        pw = rand()
return P
```

**NIST curves: use Flush+Reload to detect when code is executed**

# Attacking client: Intel Core i7-7500



# Attacking client: Intel Core i7-7500



WPA3 client (Hostap):

**~20 measurements** / address

Using Linear Classifier

# Detailed Analysis: See Paper

- Estimate required #(spoofed MAC addresses):

$$\ell = \sum_{i=1}^{\infty} i \cdot \Pr[Z_d = i] = \sum_{i=1}^{\infty} i \cdot (\Pr[Z_d \leq i] - \Pr[Z_d \leq i - 1])$$

# Detailed Analysis: See Paper

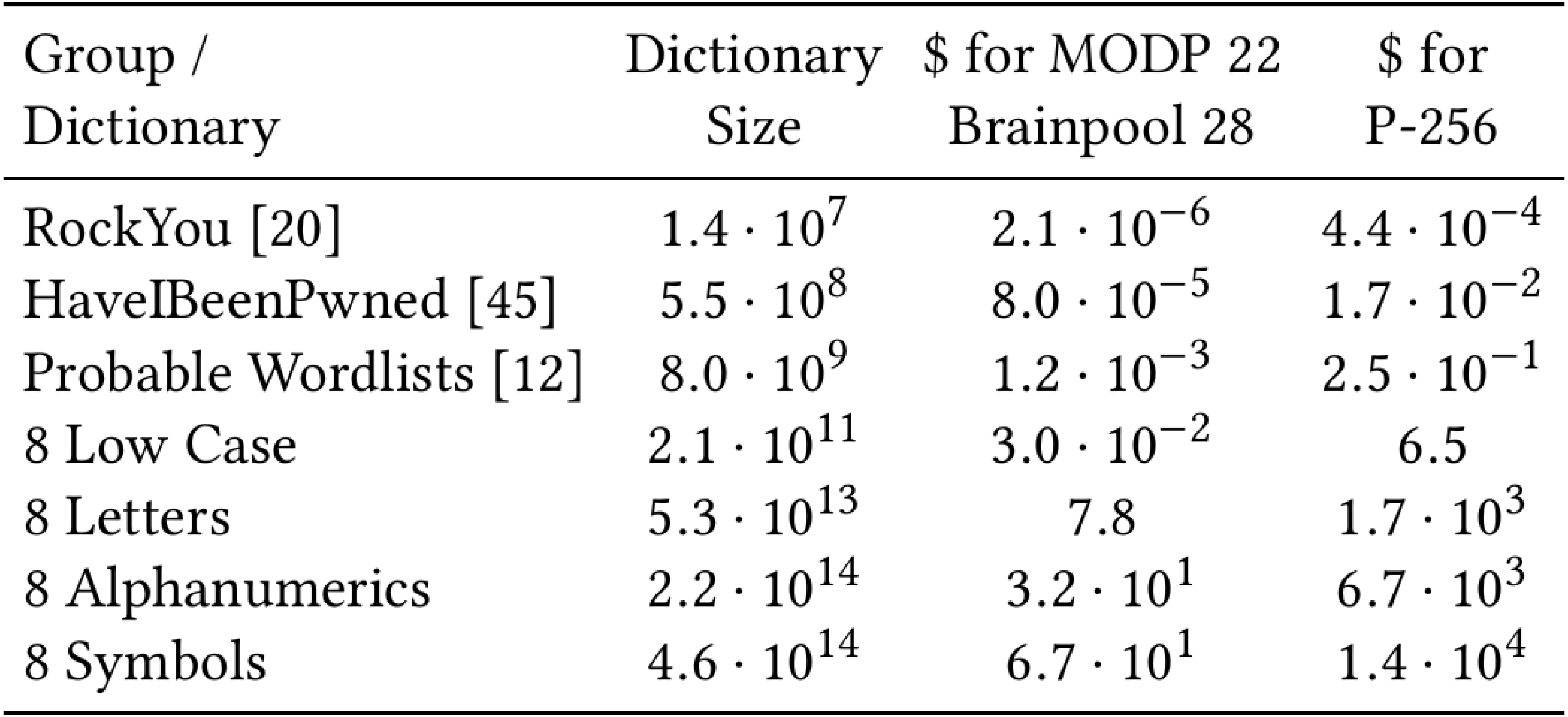
- Estimate required #(spoofed MAC addresses):

$$\ell = \sum_{i=1}^{\infty} i \cdot \Pr[Z_d = i] = \sum_{i=1}^{\infty} i \cdot (\Pr[Z_d \leq i] - \Pr[Z_d \leq i - 1])$$

- Offline brute-force cost:

$$\sum_{n=1}^{k'} n \cdot p_e^{n-1} \cdot (1 - p_e) + p_e^{k'} \cdot \sum_{n=1}^{\infty} (k' + n) \cdot (1 - p_e)^{n-1} \cdot p_e$$

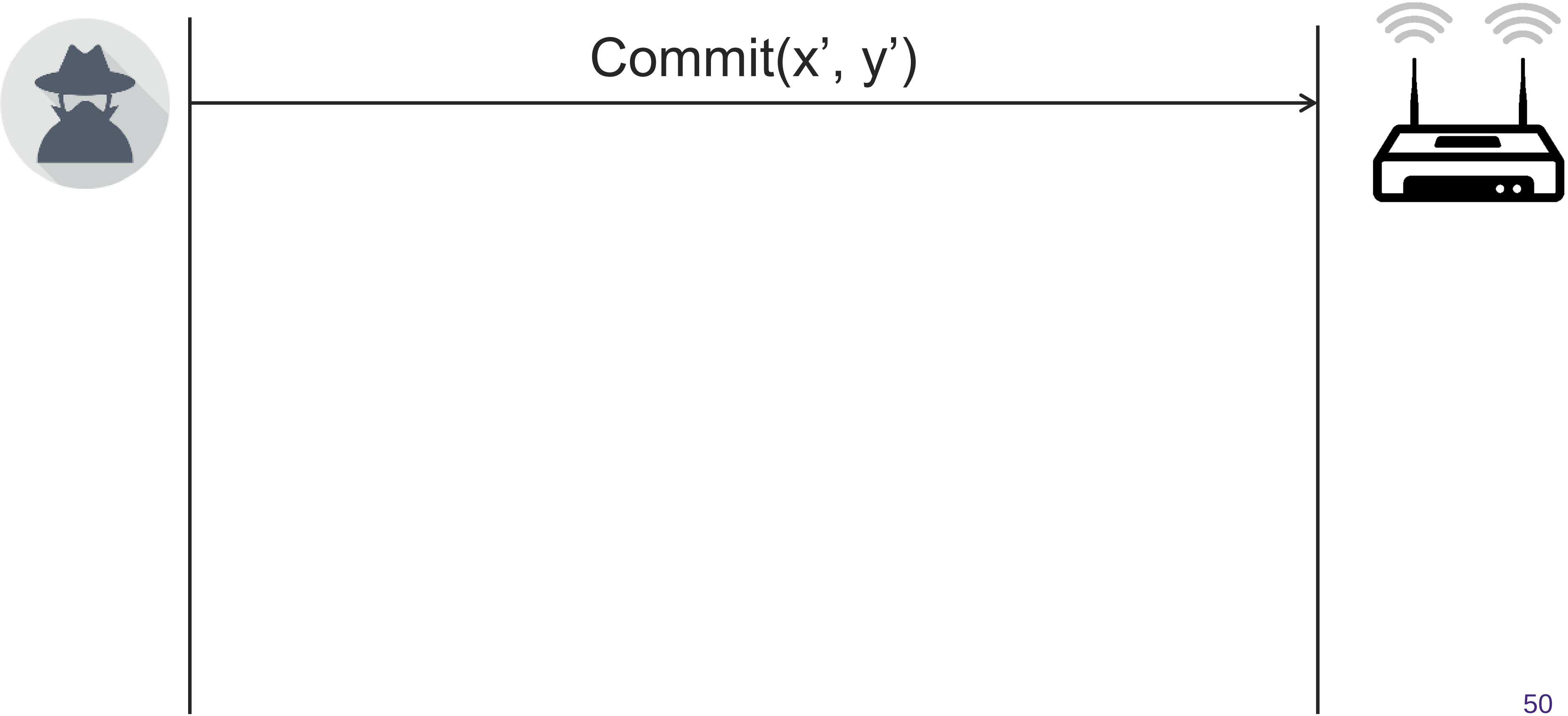
# Password Brute-force Cost



| Group / Dictionary | Dictionary Size | $ for MODP 22 Brainpool 28 | $ for P-256 |
|---|---|---|---|
| RockYou [20] | $1.4 \cdot 10^{7}$ | $2.1 \cdot 10^{-6}$ | $4.4 \cdot 10^{-4}$ |
| HaveIBeenPwned [45] | $5.5 \cdot 10^{8}$ | $8.0 \cdot 10^{-5}$ | $1.7 \cdot 10^{-2}$ |
| Probable Wordlists [12] | $8.0 \cdot 10^{9}$ | $1.2 \cdot 10^{-3}$ | $2.5 \cdot 10^{-1}$ |
| 8 Low Case | $2.1 \cdot 10^{11}$ | $3.0 \cdot 10^{-2}$ | 6.5 |
| 8 Letters | $5.3 \cdot 10^{13}$ | 7.8 | $1.7 \cdot 10^{3}$ |
| 8 Alphanumerics | $2.2 \cdot 10^{14}$ | $3.2 \cdot 10^{1}$ | $6.7 \cdot 10^{3}$ |
| 8 Symbols | $4.6 \cdot 10^{14}$ | $6.7 \cdot 10^{1}$ | $1.4 \cdot 10^{4}$ |

# Implementation Inspection

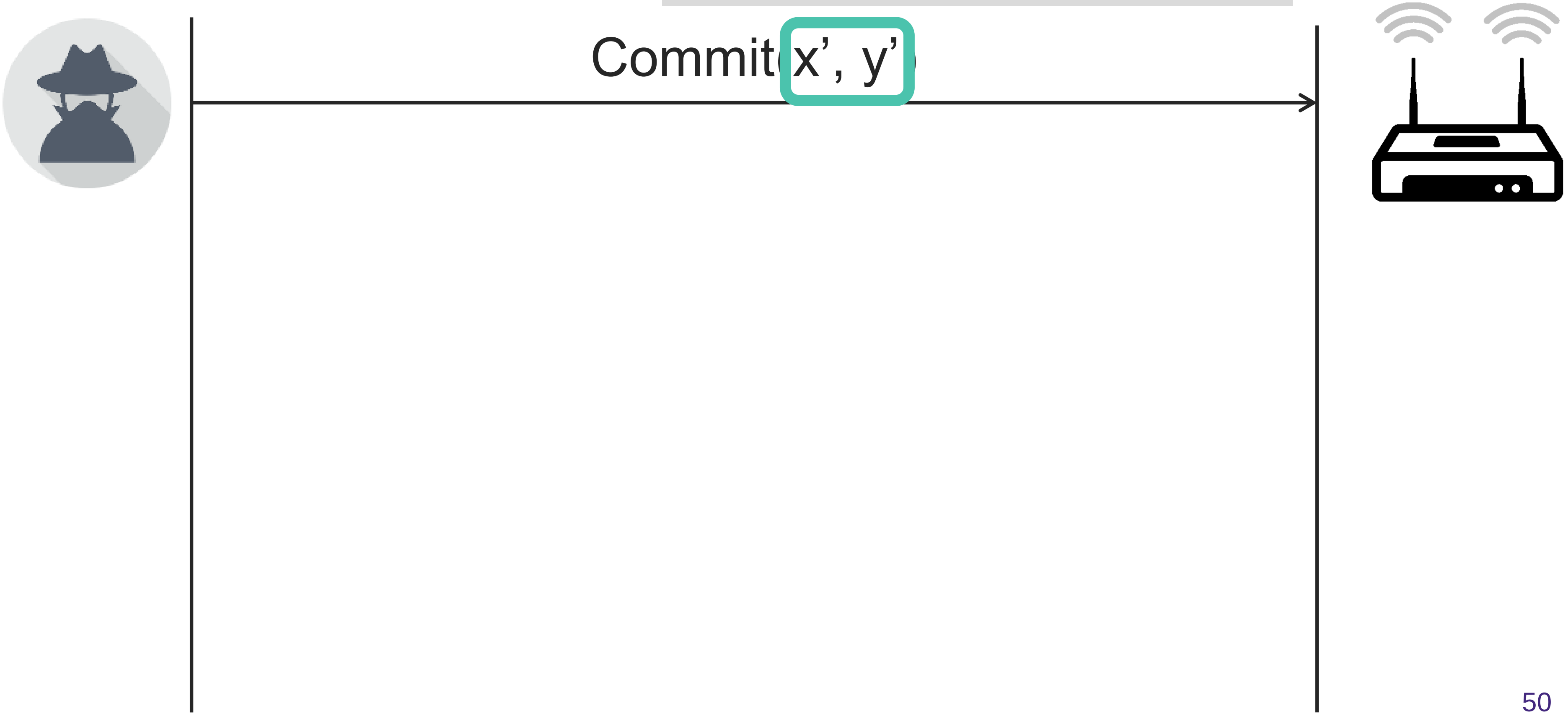
# Invalid Curve Attack



Commit(x’, y’)

# Invalid Curve Attack

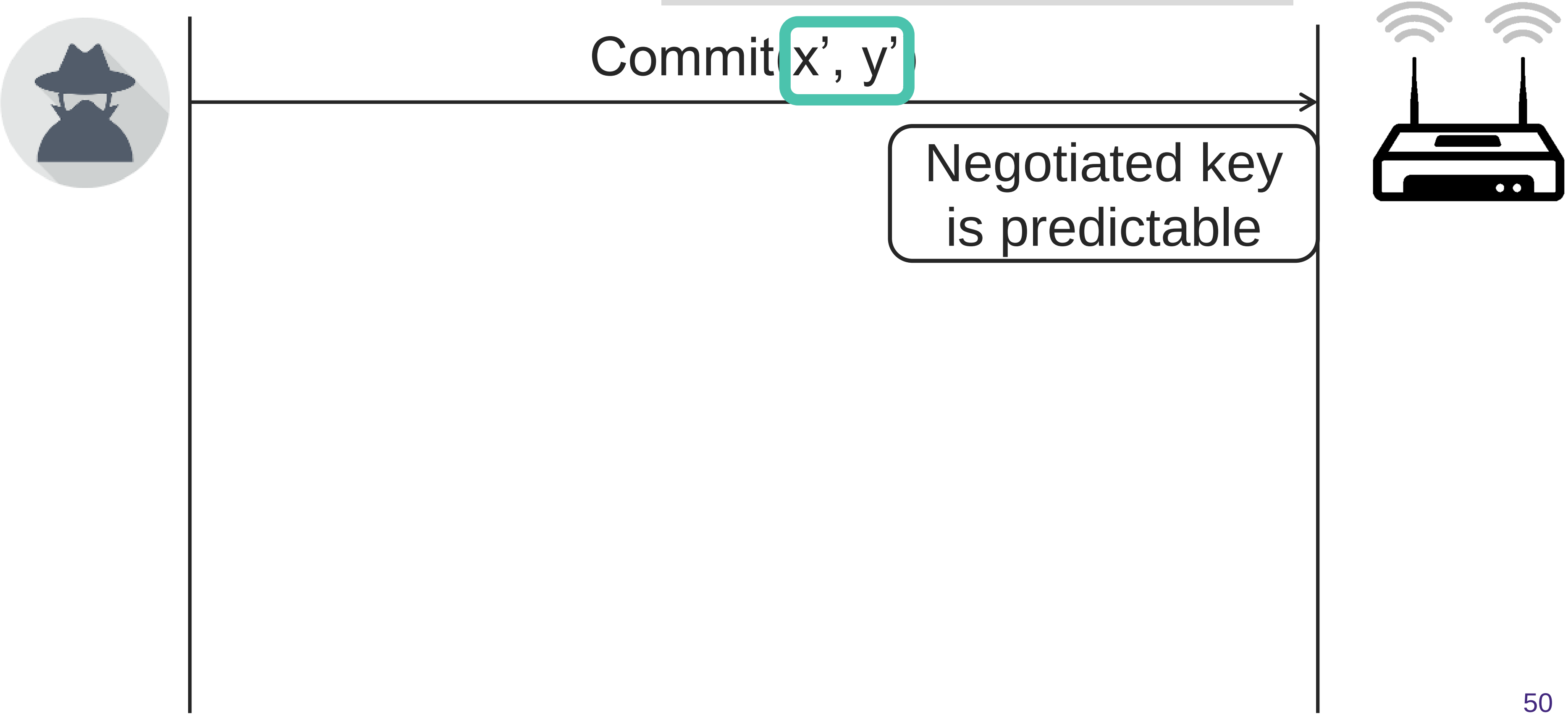
**Point isn’t on curve**



Commit(x’, y’)

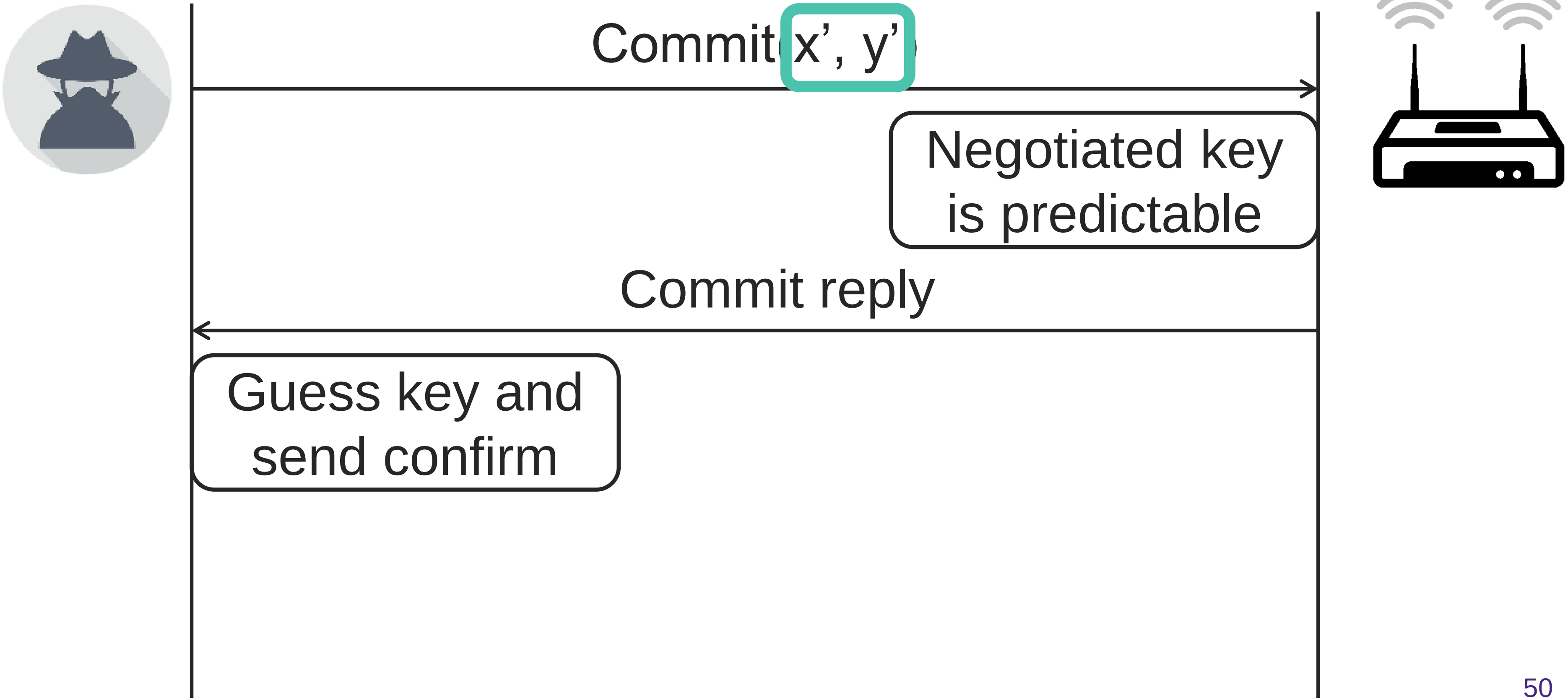
# Invalid Curve Attack

**Point isn’t on curve**



# Invalid Curve Attack

**Point isn’t on curve**



Commit(x’, y’)

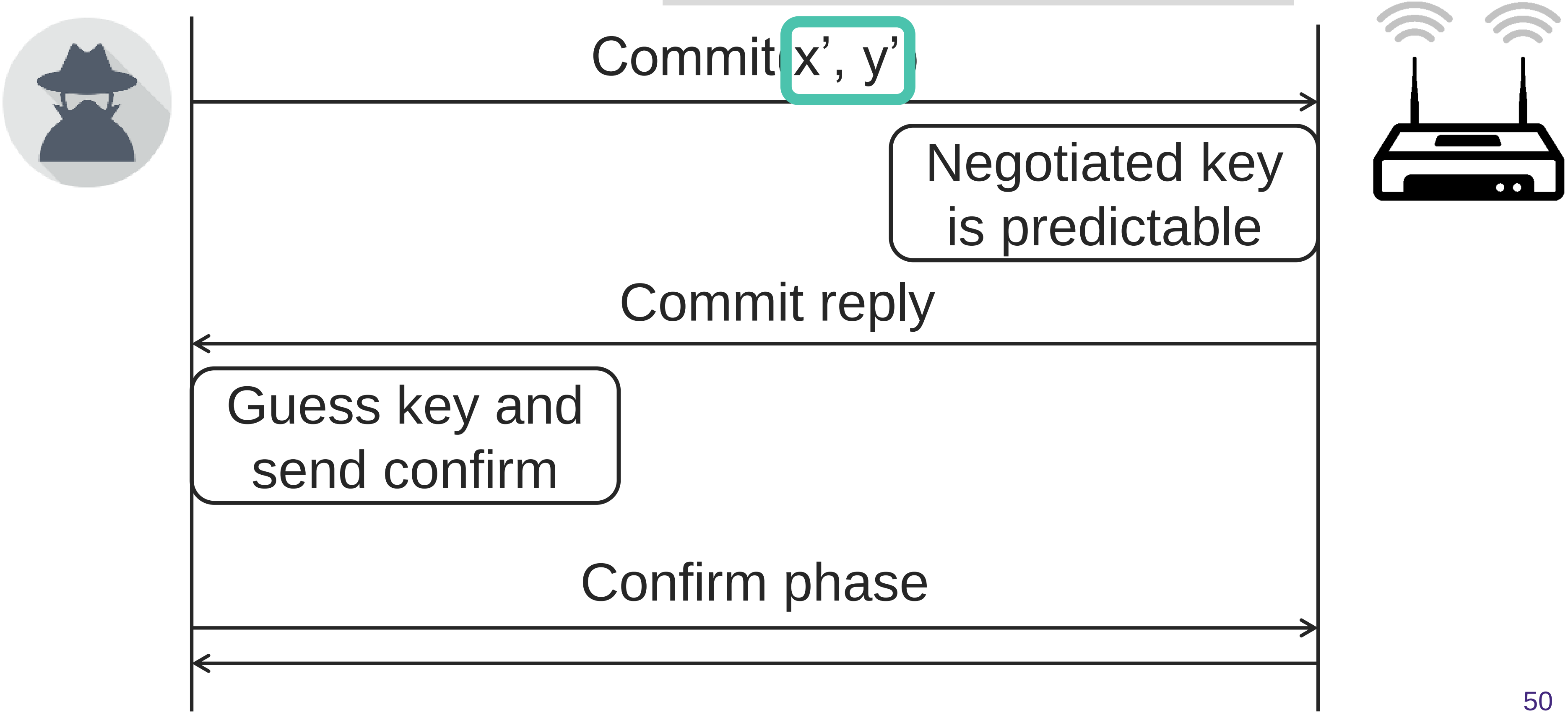
Negotiated key is predictable

Commit reply

Guess key and send confirm

# Invalid Curve Attack

**Point isn’t on curve**



Commit(x’, y’)

Negotiated key is predictable

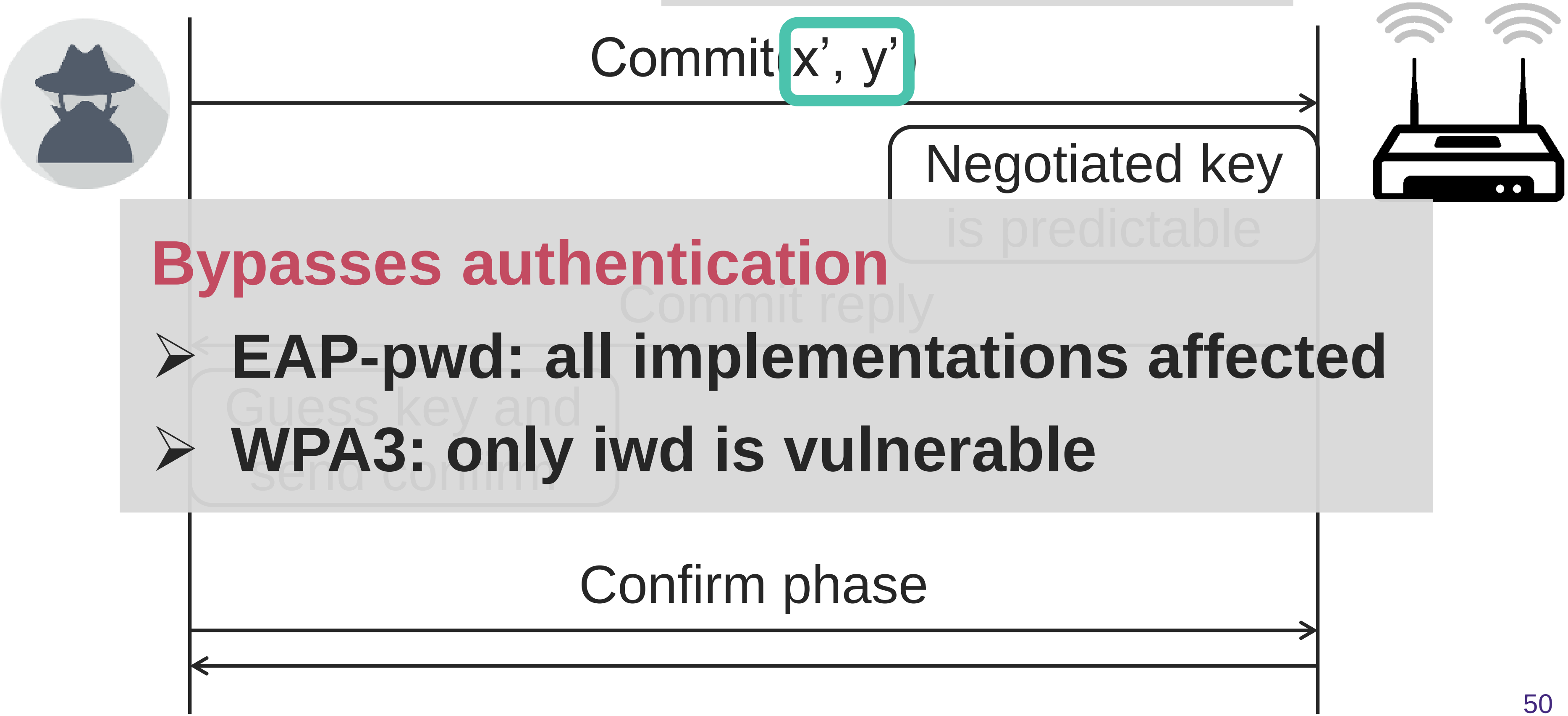
Commit reply

Guess key and send confirm

Confirm phase

# Invalid Curve Attack

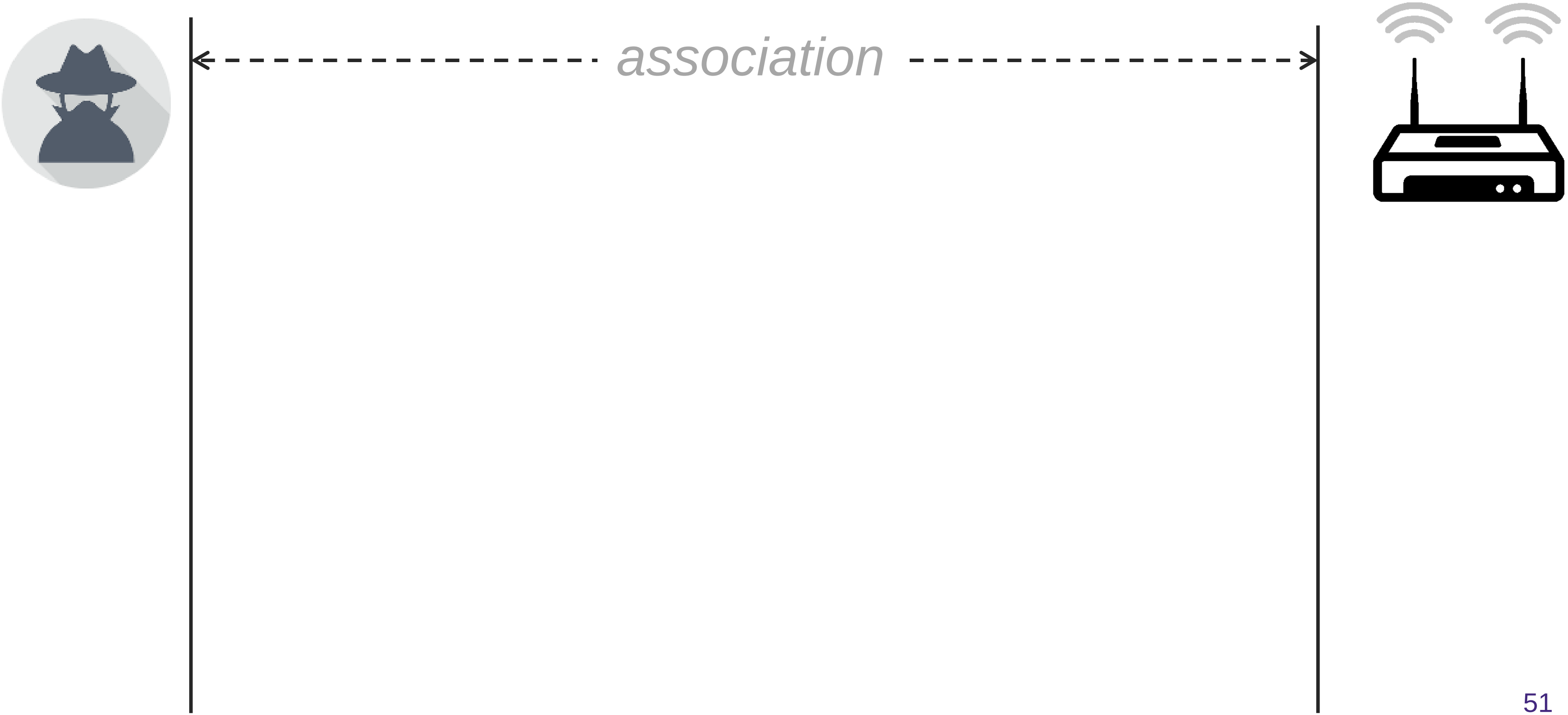
**Point isn't on curve**



**Bypasses authentication**

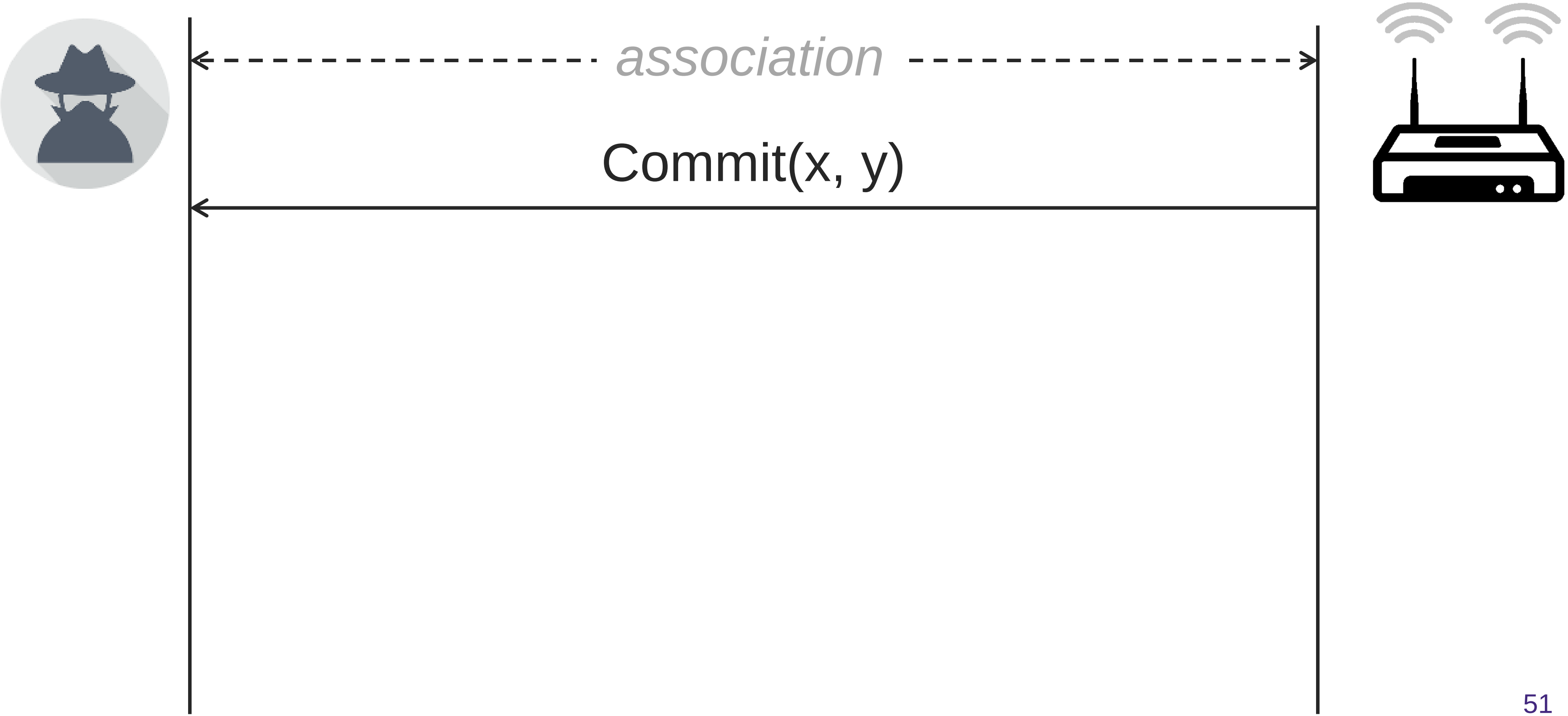
- ➢ **EAP-pwd: all implementations affected**
- ➢ **WPA3: only iwd is vulnerable**

# Reflection Attack: EAP-pwd example



*association*

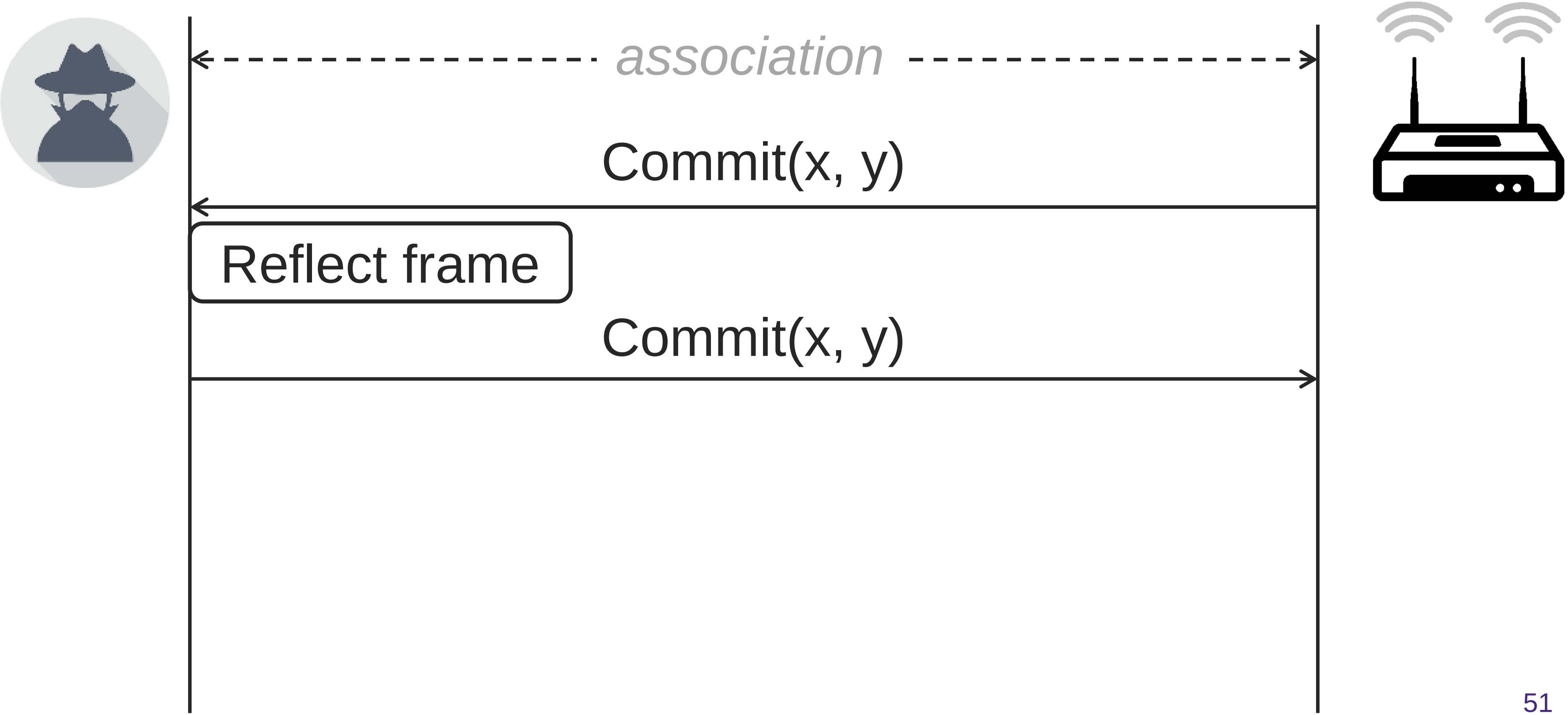
# Reflection Attack: EAP-pwd example



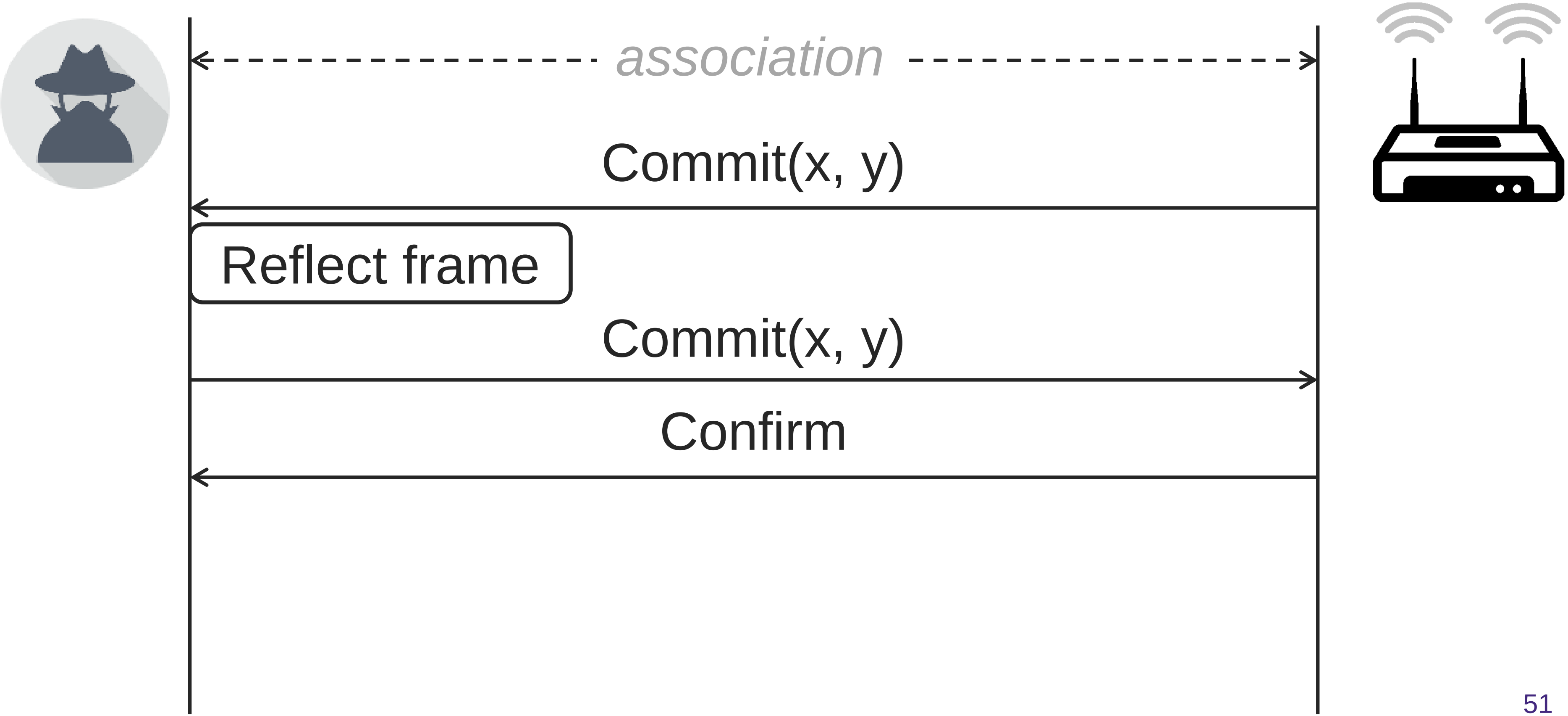
*association*

Commit(x, y)

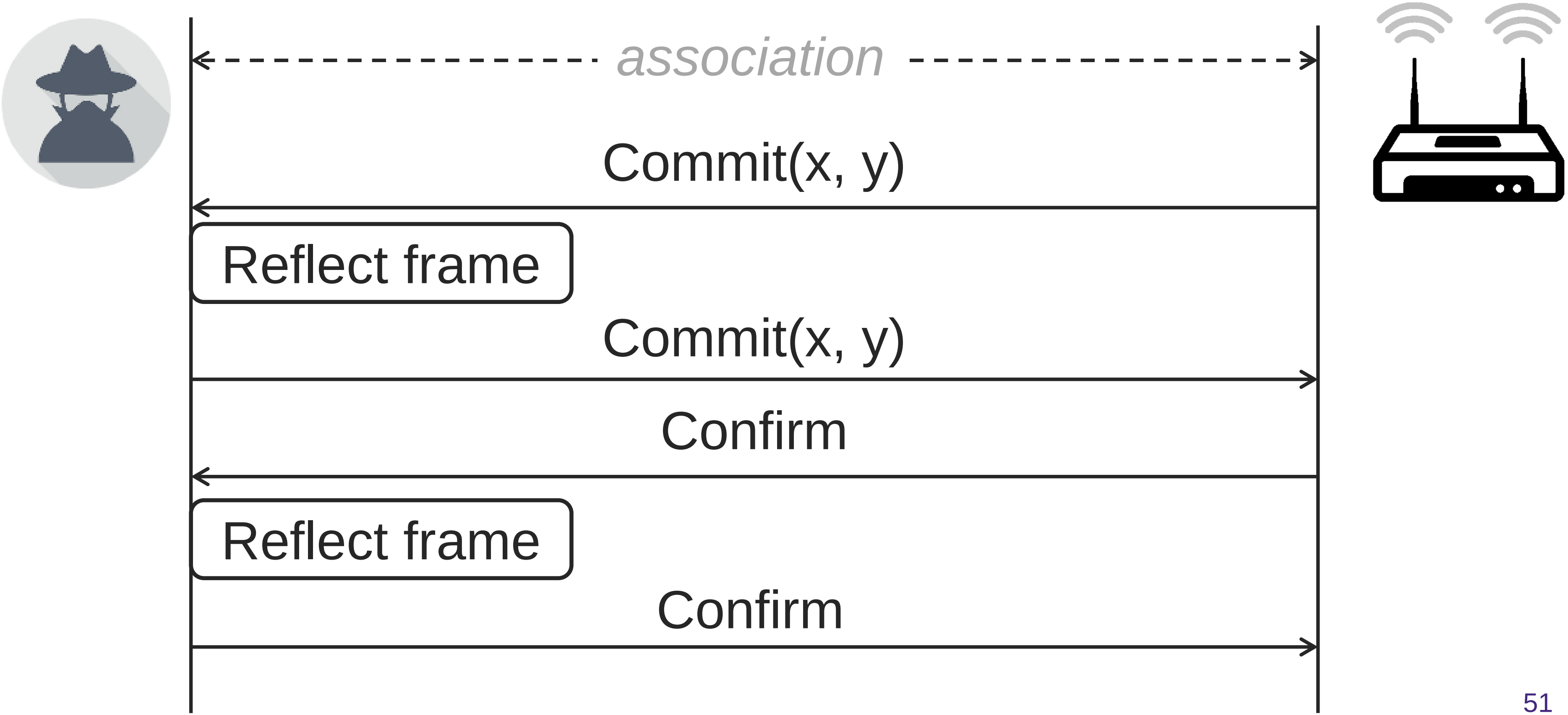
# Reflection Attack: EAP-pwd example



# Reflection Attack: EAP-pwd example



# Reflection Attack: EAP-pwd example



*association*

Commit(x, y)

Reflect frame

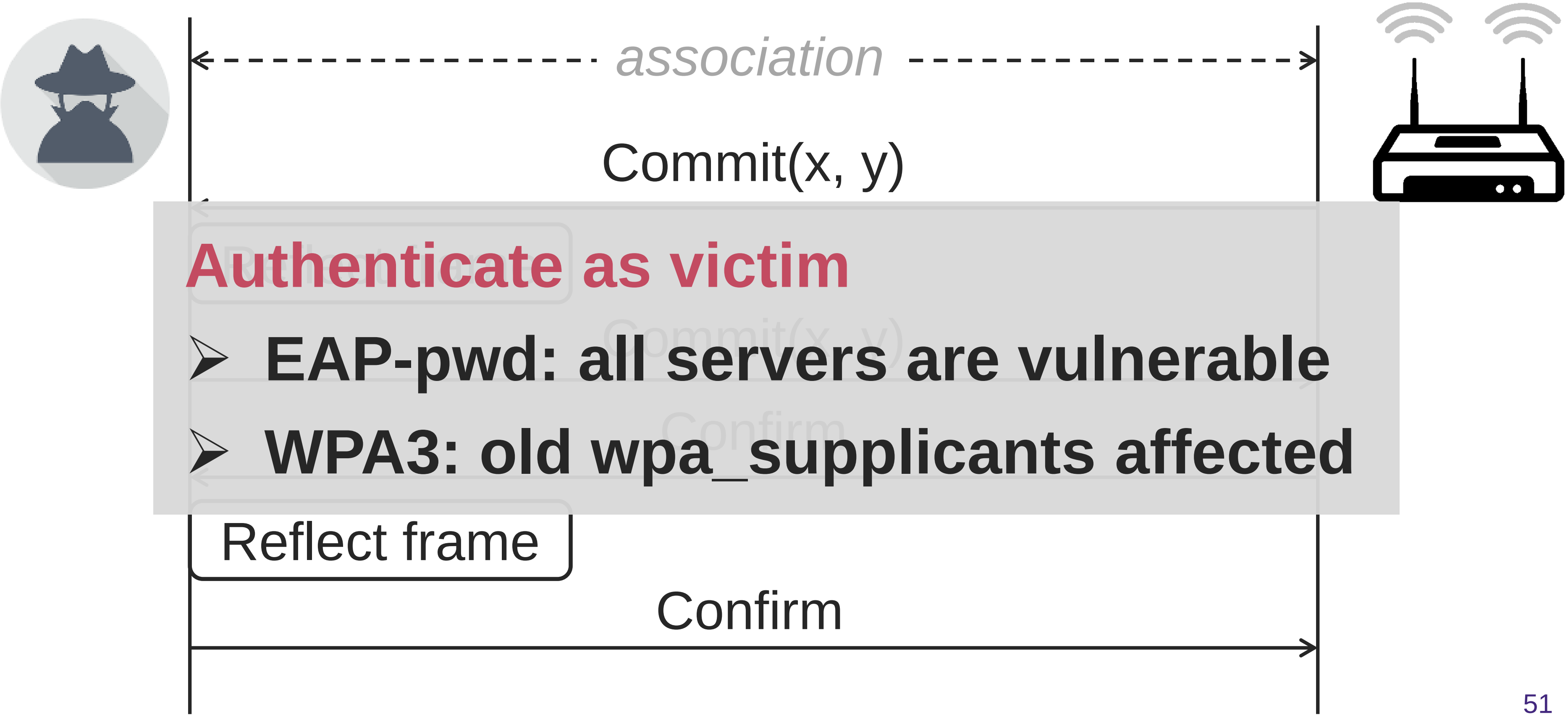
Commit(x, y)

Confirm

Reflect frame

Confirm

# Reflection Attack: EAP-pwd example



**Authenticate as victim**

- ➢ **EAP-pwd: all servers are vulnerable**
- ➢ **WPA3: old wpa_supplicants affected**

# Other Implementation Vulnerabilities

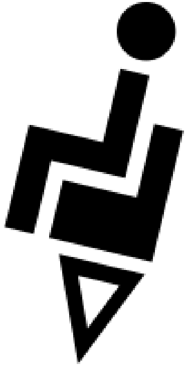
**Bad randomness**:

- Can recover password element P
- Aruba’s EAP-pwd client for Windows is affected
- With WPA2 bad randomness has lower impact!

# Other Implementation Vulnerabilities

**Bad randomness**:

- Can recover password element P
- Aruba’s EAP-pwd client for Windows is affected
- With WPA2 bad randomness has lower impact!

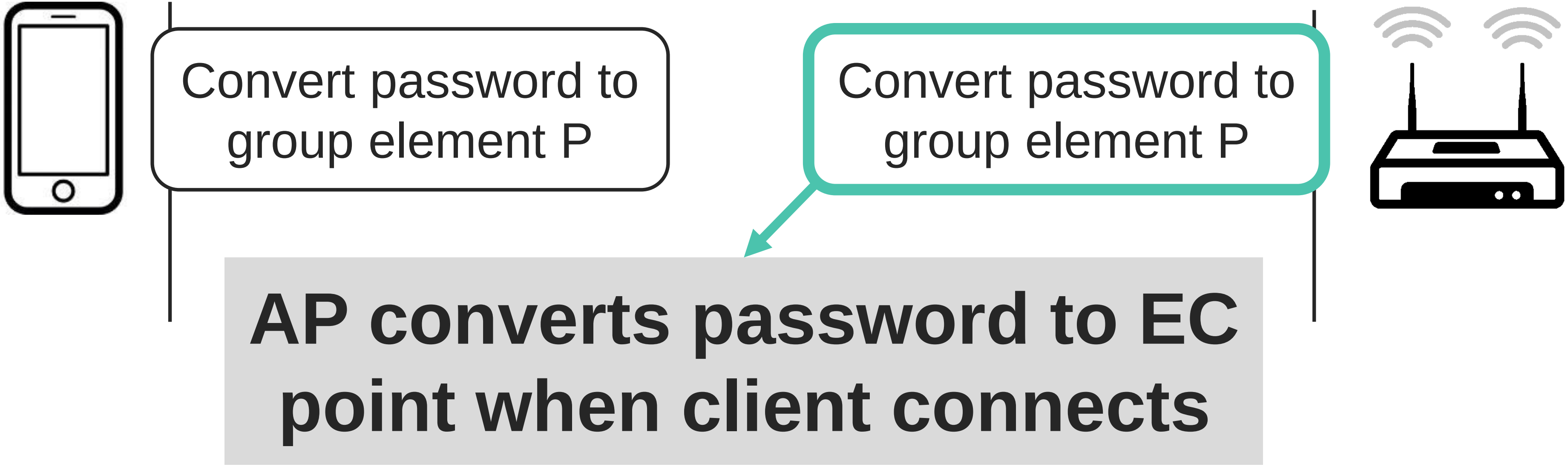


**Side-channels**:

- FreeRADIUS aborts if >10 iterations are needed
- Aruba’s EAP-pwd aborts if >30 are needed
- Can use leaked info to recover password

# Wi-Fi Specific Attacks

# Denial-of-Service Attack



Convert password to group element P

Convert password to group element P

**AP converts password to EC point when client connects**

- Conversion is computationally expensive (**40 iterations**)
- Forging **8 connections/sec** saturates AP’s CPU

# Downgrade Attacks

Transition mode: **WPA2/3 use the same password**

- WPA2’s handshake detects downgrades

# Downgrade Attacks

Transition mode: **WPA2/3 use the same password**

- WPA2’s handshake detects downgrades
- Performing partial WPA2 handshake → **dictionary attacks**

# Downgrade Attacks

Transition mode: **WPA2/3 use the same password**

- WPA2’s handshake detects downgrades
- Performing partial WPA2 handshake → **dictionary attacks**

Handshake can be performed with multiple curves

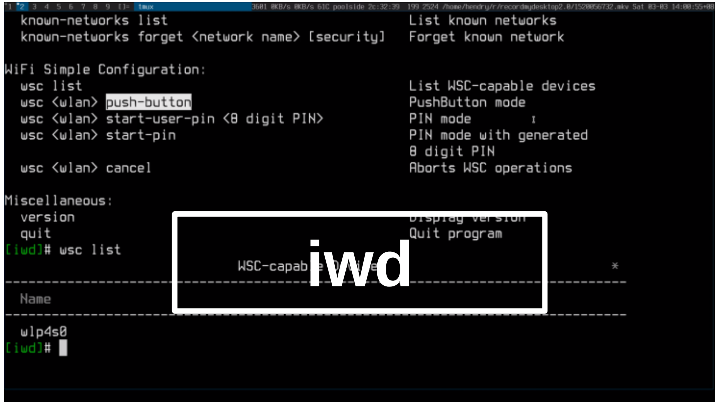
- Initiator proposes curve & responder accepts/rejects
- **Spoof reject messages to downgrade** used curve

# Implementation-specific downgrades

- Clone WPA3-only network & advertise it only supports WPA2

# Implementation-specific downgrades

- Clone WPA3-only network & advertise it only supports WPA2
- Galaxy S10 & iwd connected using the WPA3-only password
- Results in trivial dictionary attack



```
1 2 3 4 5 6 7 8 9 []= tmux     3601 0KB/s 0KB/s 61C poolside 2c:32:39  199 2524 /home/hendry/r/recordmydesktop2.0/1520056732.mkv Sat 03-03 14:00:55+08
  known-networks list                               List known networks
  known-networks forget <network name> [security]   Forget known network

WiFi Simple Configuration:
  wsc list                                          List WSC-capable devices
  wsc <wlan> push-button                            PushButton mode
  wsc <wlan> start-user-pin <8 digit PIN>           PIN mode
  wsc <wlan> start-pin                              PIN mode with generated
                                                    8 digit PIN
  wsc <wlan> cancel                                 Aborts WSC operations

Miscellaneous:
  version                                           Display version
  quit                                              Quit program
[iwd]# wsc list
                             WSC-capable Devices                          *
--------------------------------------------------------------------------------
  Name
--------------------------------------------------------------------------------
  wlp4s0
[iwd]# █
```

# Disclosure

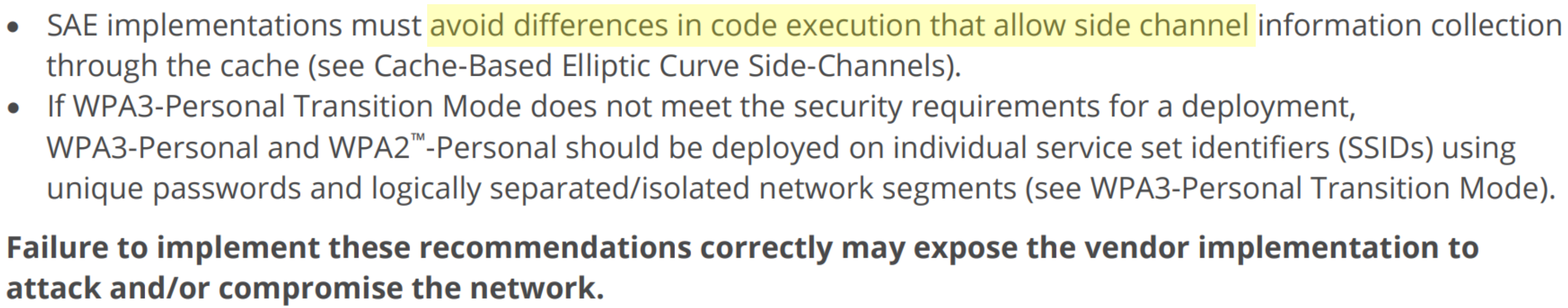
# Disclosure process

Notified parties early with **hope to influence WPA3**

Reaction of the Wi-Fi Alliance

- **Privately created** backwards-compatible security guidelines
- **2nd disclosure** round to address Brainpool side-channels
- **Nov 2019**: Updated guidelines now prohibit Brainpool curves

# Latest Wi-Fi Alliance guidelines (Nov 2019)

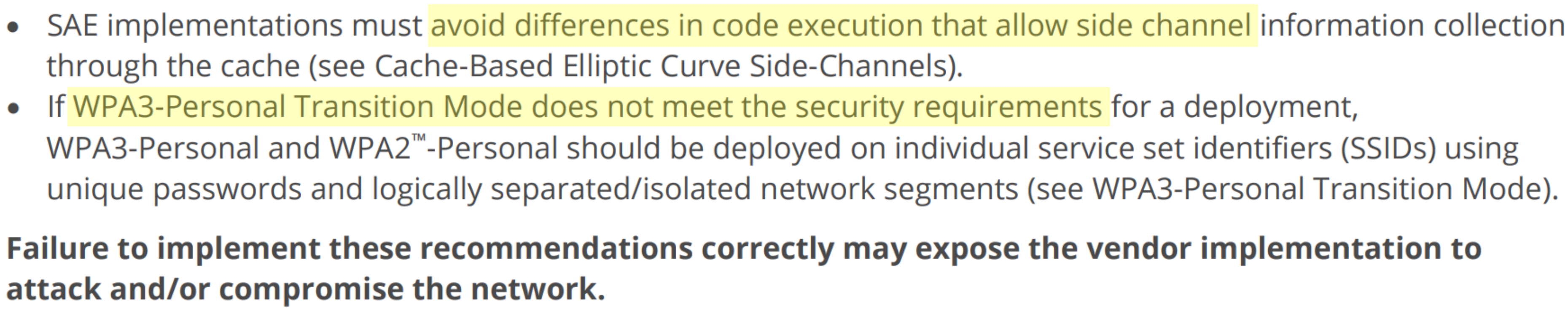


- SAE implementations must avoid differences in code execution that allow side channel information collection through the cache (see Cache-Based Elliptic Curve Side-Channels).
- If WPA3-Personal Transition Mode does not meet the security requirements for a deployment, WPA3-Personal and WPA2™-Personal should be deployed on individual service set identifiers (SSIDs) using unique passwords and logically separated/isolated network segments (see WPA3-Personal Transition Mode).

**Failure to implement these recommendations correctly may expose the vendor implementation to attack and/or compromise the network.**

- “implementations must avoid [..] side-channels”

# Latest Wi-Fi Alliance guidelines (Nov 2019)

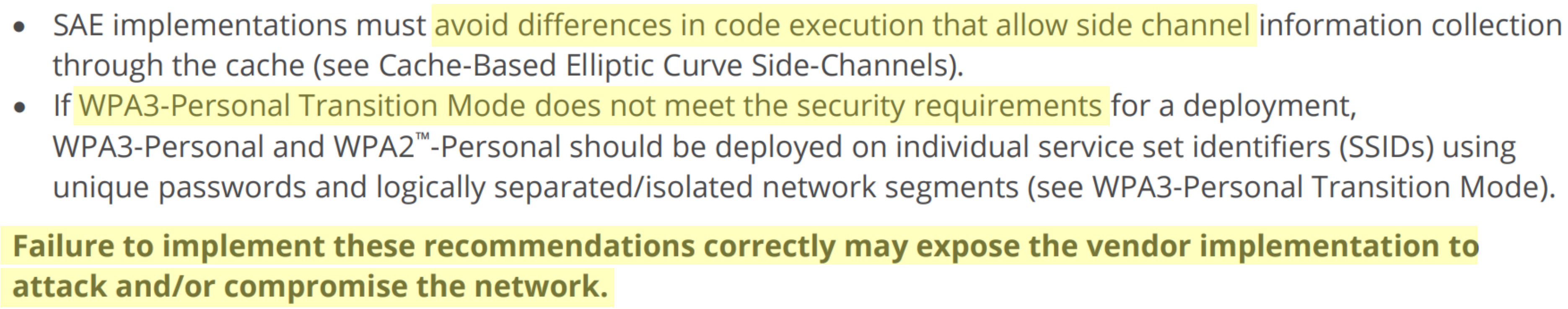


- SAE implementations must avoid differences in code execution that allow side channel information collection through the cache (see Cache-Based Elliptic Curve Side-Channels).
- If WPA3-Personal Transition Mode does not meet the security requirements for a deployment, WPA3-Personal and WPA2™-Personal should be deployed on individual service set identifiers (SSIDs) using unique passwords and logically separated/isolated network segments (see WPA3-Personal Transition Mode).

**Failure to implement these recommendations correctly may expose the vendor implementation to attack and/or compromise the network.**

- “implementations must avoid [..] side-channels”
- If WPA3-Transition *“doesn’t meet security requirements”*, then seperate passwords

# Latest Wi-Fi Alliance guidelines (Nov 2019)



- SAE implementations must avoid differences in code execution that allow side channel information collection through the cache (see Cache-Based Elliptic Curve Side-Channels).
- If WPA3-Personal Transition Mode does not meet the security requirements for a deployment, WPA3-Personal and WPA2™-Personal should be deployed on individual service set identifiers (SSIDs) using unique passwords and logically separated/isolated network segments (see WPA3-Personal Transition Mode).

**Failure to implement these recommendations correctly may expose the vendor implementation to attack and/or compromise the network.**

- “implementations must avoid [..] side-channels”
- If WPA3-Transition *“doesn’t meet security requirements”*, then seperate passwords
- “Failure to implement...” → how can it be checked?

# Fundamental issue still unsolved

- Hard to implement in constant time
- On lightweight devices, doing **40 iterations is too costly**

# Fundamental issue still unsolved

- Hard to implement in constant time
- On lightweight devices, doing **40 iterations is too costly**

**Draft IEEE 802.11 standard has been updated**

- Exclude MAC addresses from hash2curve
  - Allows offline computation of password element
- Now uses constant-time hash2curve
- Explicitly prohibit use of weak EC & MODP groups
- Prevent crypto group downgrade attack

# Remaining issues

Message **transcript is not included in key derivation**

- Prevents formal proof of protocol
- High risk of implementation issues
  - E.g. prevention of crypto group downgrade attack

# Remaining issues

Message **transcript is not included in key derivation**

- Prevents formal proof of protocol
- High risk of implementation issues
  - E.g. prevention of crypto group downgrade attack

Downgrade to WPA2

- **Not addressed in the standard**
- Up to vendor whether to implement trust-on-first-use
  - Done by Android & NetworkManager of Linux

# Issue 2: not backwards-compatible

Might lead to WPA3.1?

- Not yet clear how Wi-Fi Alliance will handle this
- **Risk of downgrade attacks** to original WPA3

# Issue 2: not backwards-compatible

Might lead to WPA3.1?

- Not yet clear how Wi-Fi Alliance will handle this
- **Risk of downgrade attacks** to original WPA3

Should you switch to WPA3?

- WPA2 is trivial to attack... so yes.

# Conclusion

- WPA3 vulnerable to side-channels
- Countermeasures are costly
- **Draft 802.11 standard updated**
- Issues could have been avoided!

https://wpa3.mathyvanhoef.com

# Thank you! Questions?

- WPA3 vulnerable to side-channels
- Countermeasures are costly
- **Draft 802.11 standard updated**
- Issues could have been avoided!

https://wpa3.mathyvanhoef.com