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

Mathy Vanhoef and Eyal Ronen

ANRW. Montreal, Canada, 22 July 2019.

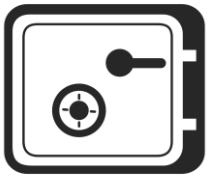
NEW YORK UNIVERSITY

Background: Dragonfly in WPA3

= Password Authenticated Key Exchange (PAKE)

Provide mutual authentication

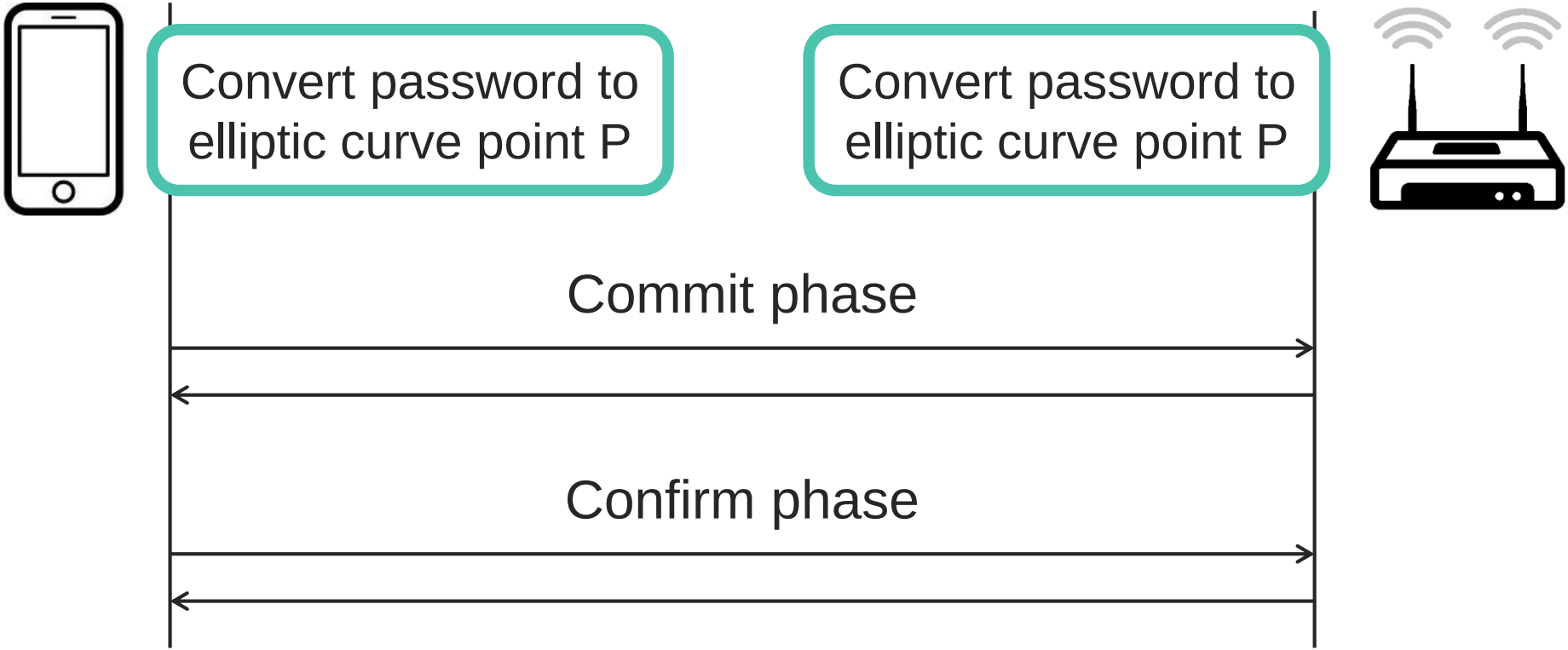
Negotiate session key



Forward secrecy & prevent offline dictionary attacks

Protect against server compromise

Dragonfly


Convert password to elliptic curve point P
Convert password to elliptic curve point P
Commit phase
Confirm phase

With MODP groups: hash-to-group

```
for (counter = 1; counter < 256; counter++)
    value = hash(pw, counter, addr1, addr2)
    if value >= p: continue
    P = value^((p-1)/q)
    if P > 1: return P
```

With MODP groups: hash-to-group

```
for (counter = 1; counter < 256; counter++)
    value = hash(pw, counter, addr1, addr2)
    if value >= p: continue
    P = value^((p-1)/q)
    if P > 1  return P
```

In practice always true

With MODP groups: hash-to-group

```
for (counter = 1; counter < 256; counter++)
    value = hash(pw, counter, addr1, addr2)
    if value
    P = value^((p-1)/q)
    if P > 1  return P
```

Problem: value >= p

In practice always true

With MODP groups: hash-to-group

```
for (counter = 1; counter < 256; counter++)
    value = hash(pw, counter, addr1, addr2)
    if value >= p: continue
    P = value^((p-1)/q)
    if P > 1: return P
```

With MODP groups: hash-to-group

```
for (counter = 1; counter < 256; counter++)
    value = hash(pw, counter, addr1, addr2)
    if value >= p: continue
    P = value^((p−1)/q)
    if P > 1: return P
```

With MODP groups: hash-to-group

```
for (counter = 1; counter < 256; counter++)
    value = hash(pw, counter, addr1, addr2)
    if value >= p: continue
    P = value^((p-1)/q)
    if P > 1: return P
```

No timing leak countermeasures
despite warnings by IETF & CFRG!

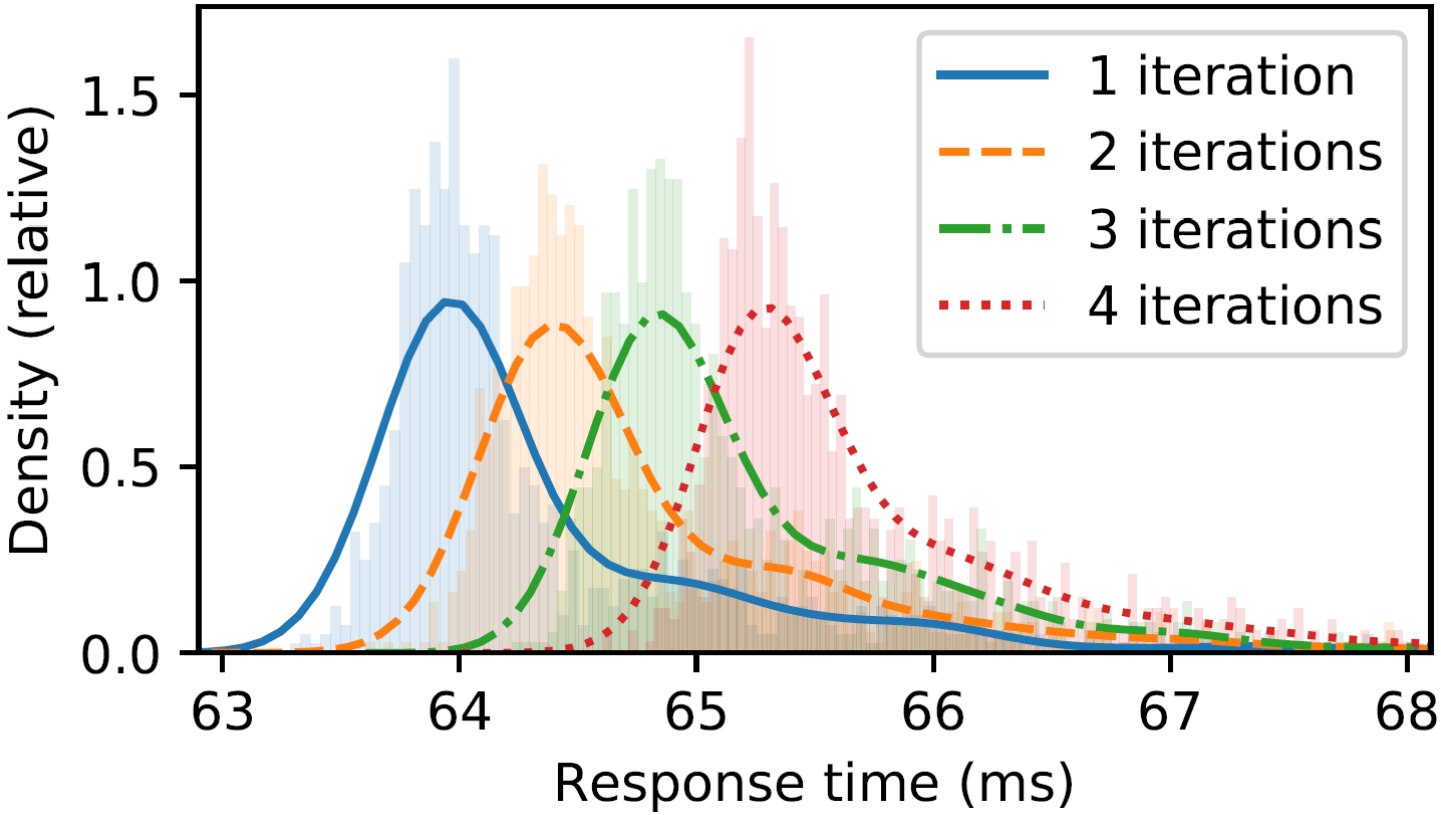
With MODP groups: hash-to-group

```
for (counter = 1; counter < 256; counter++)
    value = hash(pw, counter, addr1, addr2)
    if value >= p: continue
    P = value^((p-1)/q)
    if P > 1: return P
```

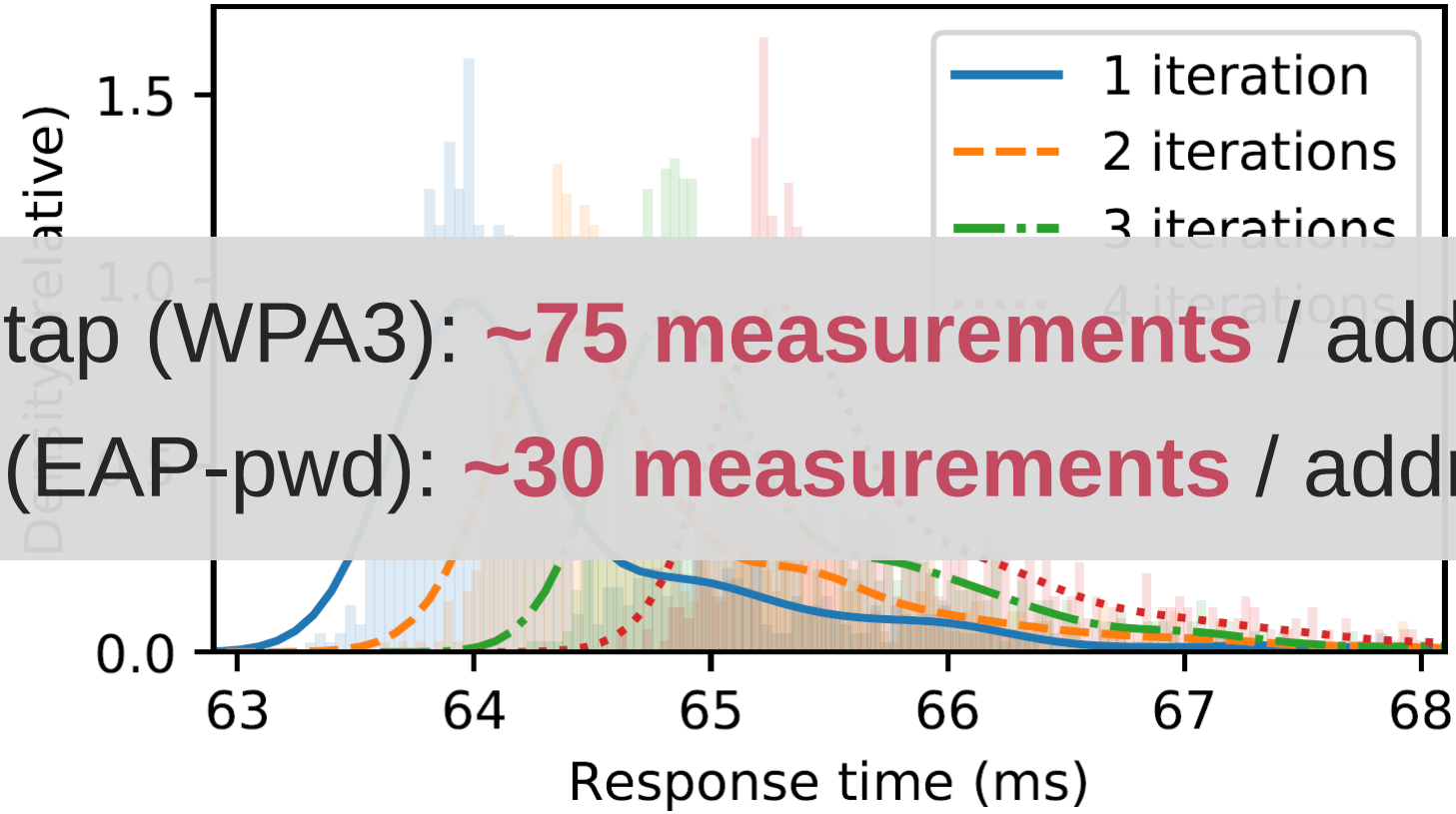
WPA3: **spoof client address** to obtain different executions

No timing leak countermeasures
despite warnings by IETF & CFRG!

Raspberry Pi 1 B+: differences are measurable



Raspberry Pi 1 B+: differences are measurable



Hostap (WPA3): **~75 measurements** / address

iwd (EAP-pwd): **~30 measurements** / address

Leaked information: #iterations needed

Client address	addrA
Measured	

Leaked information: #iterations needed

Client address	addrA
Measured	
Password 1	
Password 2	
Password 3	

Leaked information: #iterations needed

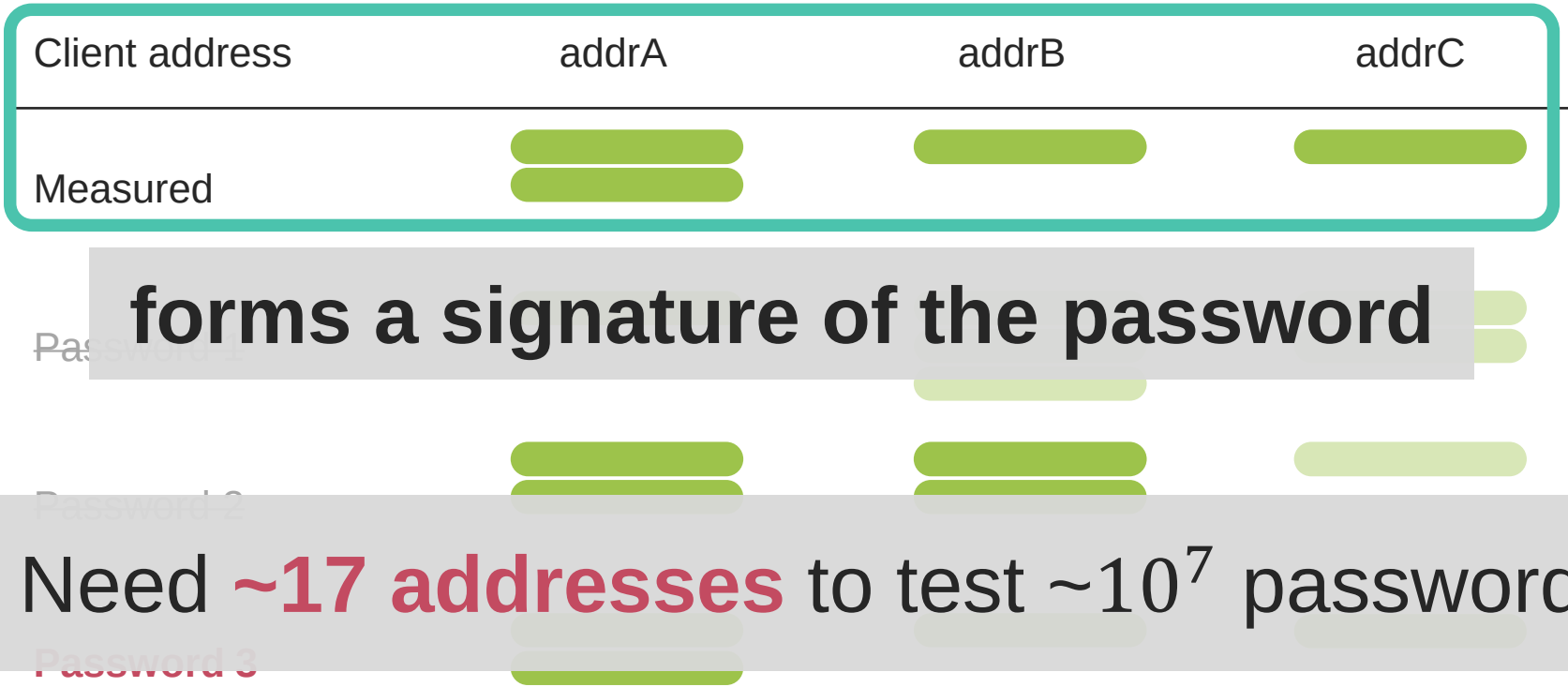
Client address	addrA
Measured	
~~Password 1~~	
Password 2	
Password 3	

Leaked information: #iterations needed

Leaked information: #iterations needed

Leaked information: #iterations needed

Leaked information: #iterations needed



Need **~17 addresses** to test ~10^7 passwords

What about elliptic curves?

Hash-to-group with elliptic curves also affected?

- By default Dragonfly uses NIST curves
- **Timing leaks for NIST curves are mitigated**

Dragonfly also supports Brainpool curves

- After our initial disclosure, the Wi-Fi Alliace private created guidelines that mention these are secure to use
- Bad news: **Brainpool curves in Dragonfly are insecure**

Hash-to-curve

```
for (counter = 1; counter < k or not x; counter++)
    value = hash(pw, counter, addr1, addr2)
    if value >= p: continue
    y_sqr = value^3 + a * value + b
    if is_quadratic_residue(y_sqr) and not x:
        x = value
        pw = random()
y = sqrt(x^3 + a * x + b)
return (x, y)
```

Hash-to-curve

```
for (counter = 1; counter < k or not x; counter++)
    value = hash(pw, counter, addr1, addr2)
    if value >= p: continue
    y_sq
    if is_quadratic_residue(y_sqr) and not x:
        x = value
        pw = random()
y = sqrt(x^3 + a * x + b)
return (x, y)
```

Problem: no solution for y

Hash-to-curve

```
for (counter = 1; counter < k or not x; counter++)
    value = hash(pw, counter, addr1, addr2)
    if value >= p: continue
    y_sqr = value^3 + a * value + b
    if is_quadratic_residue(y_sqr) and not x:
        x = value
        pw = random()
y = sqrt(x^3 + a * x + b)
return (x, y)
```

Hash-to-curve

```
for (counter = 1; counter < k or not x; counter++)
    value = hash(pw, counter, addr1, addr2)
    if value >= p: continue
    y_sqr = value^3 + a * value + b
    if is_quadratic_residue(y_sqr) and not x:
        x = value
        pw = random()
y = sqrt(x^3 + a * x + b)
return (x, y)
```

Hash-to-curve

```
for (counter = 1; counter < k or not x; counter++)
    value = hash(pw, counter, addr1, addr2)
    if value >= p: continue
    y_sq = value^3 + a * value + b
    if is_quadratic_residue(y_sq):
        x = value
        pw = random()
y = sqrt(x^3 + a * x + b)
return (x, y)
```

Problem: different passwords have different execution time

Hash-to-curve

```
for (counter = 1; counter < k or not x; counter++)
    value = hash(pw, counter, addr1, addr2)
    if value >= p: continue
    y_sqr = value^3 + a * value + b
    if is_quadratic_residue(y_sqr) and not x:
        x = value
        pw =
y = sqrt(x^3
return (x, y
```

→ Always execute at least k iterations

Hash-to-curve

```
for (counter = 1; counter < k or not x; counter++)
    value = hash(pw, counter, addr1, addr2)
    if value >= p: continue
    y_sqr = value^3 + a * value + b
    if is_quadratic_residue(y_sqr) and not x:
        x = value
        pw = random()
y = sqrt(x^3 + a * x + b)
return (x, y)
```

In case quadratic test is not constant time

Hash-to-curve

```
for (counter = 1; counter < k or not x; counter++)
    value = hash(pw, counter, addr1, addr2)
    if value >= p: continue
    y_sqr = value^3 + a * value + b
    if is_quadratic_residue(y_sqr) and not x:
        x = value
        pw = random()
y = sqrt(x^3 + a * x + b)
return (x, y)
```

Problem: value >= p

Hash-to-curve

```
for (counter = 1; counter < 2^len; counter++)
    value = hash(pw, counter, addr1, addr2)
    if value >= p: continue
    y_sqr = value^3 + a * value + b
    if is_quadratic_residue(y_sqr) and not x:
        x = value
        pw = random()
y = sqrt(x^3 + a * x + b)
return (x, y)
```

May be true for Brainpool curves!

Hash-to-curve

```
for (counter = 1; counter < 2^n; counter++)
    value = hash(pw, counter, addr1, addr2)
    if value >= p: continue
    y_sqr = value^3 + a * value + b
    if is_quadratic_residue(y_sqr) and not x:
        x = value
        pw = random()
y = sqrt(x^3 + a * x + b)
return (x, y)
```

May be true for Brainpool curves!

Quadratic test may be skipped

Hash-to-curve

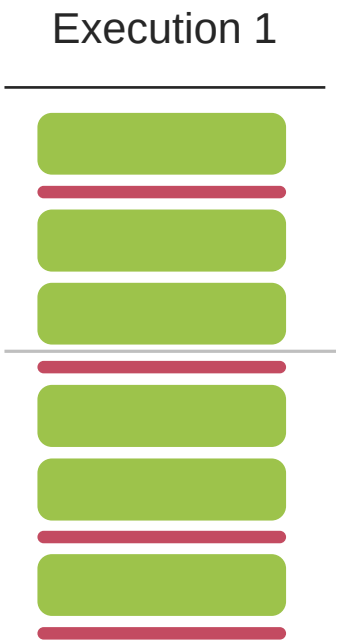
```
for (counter = 1; counter < 256; counter++)
    value = hash(pw, counter, addr1, addr2)
    if value >= p: continue
    y_sqr = value^3 + a * value + b
    if is_quadratic_residue(y_sqr) and not x:
        x = value
        pw = random()
y = sqrt(x^3 + a * x + b)
return (x, y)
```

May be true for Brainpool curves!

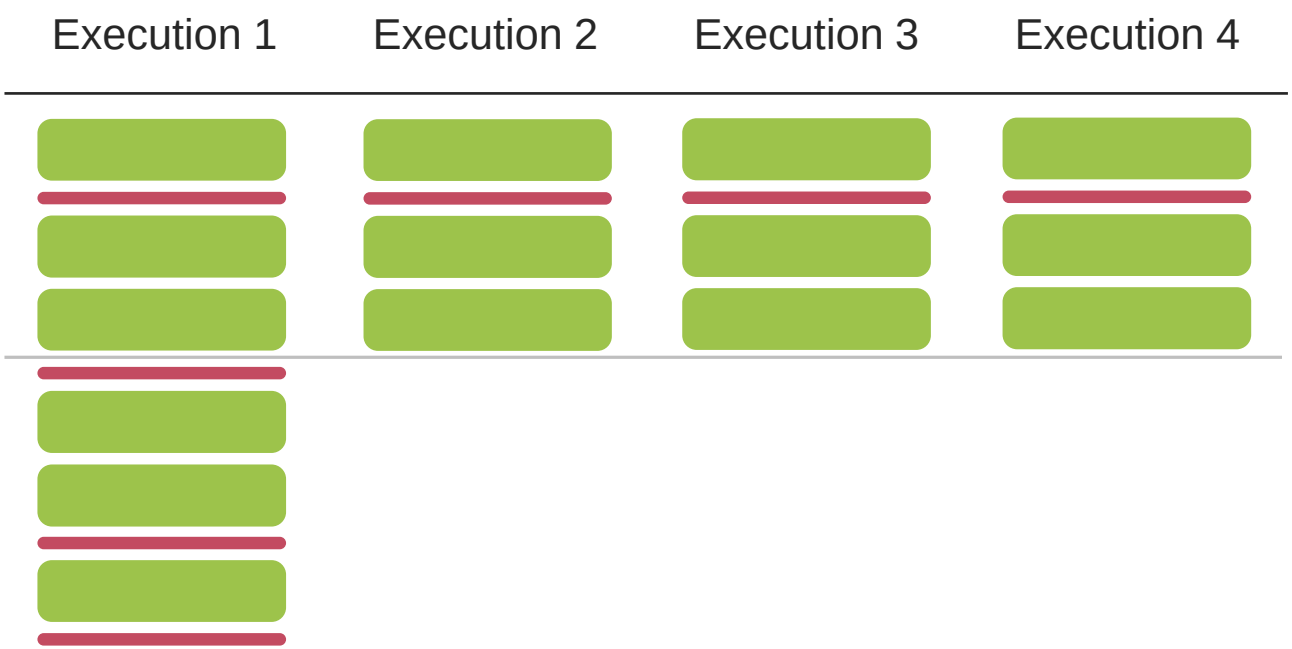
Quadratic test may be skipped

A random #(extra iterations) have a too big hash output

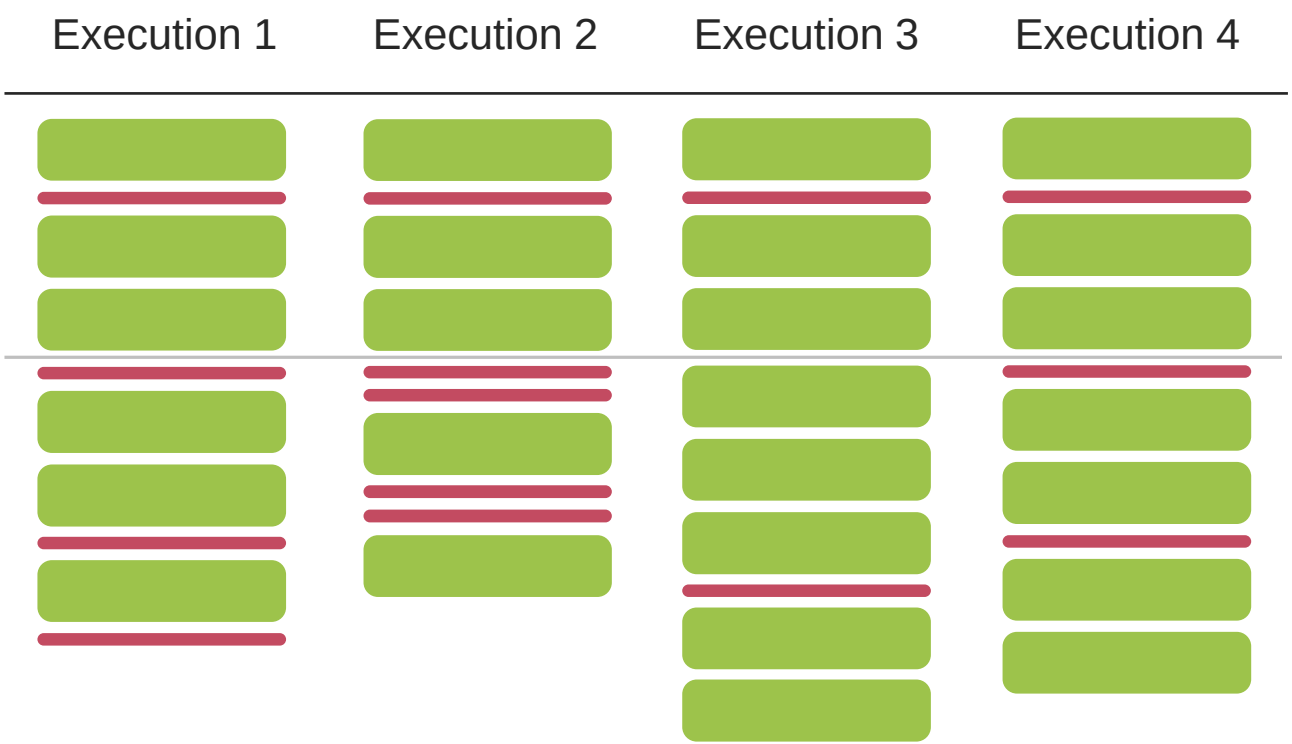
Influence of extra iterations



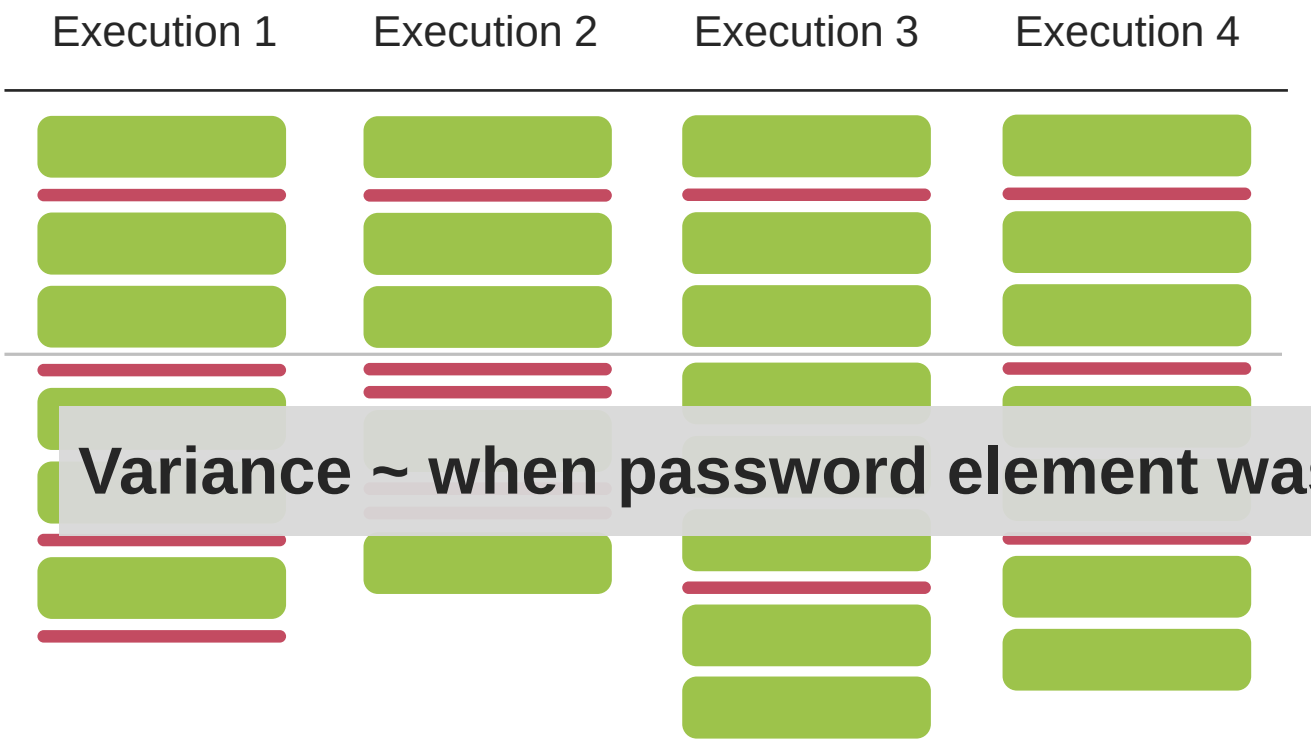
Influence of extra iterations



Influence of extra iterations

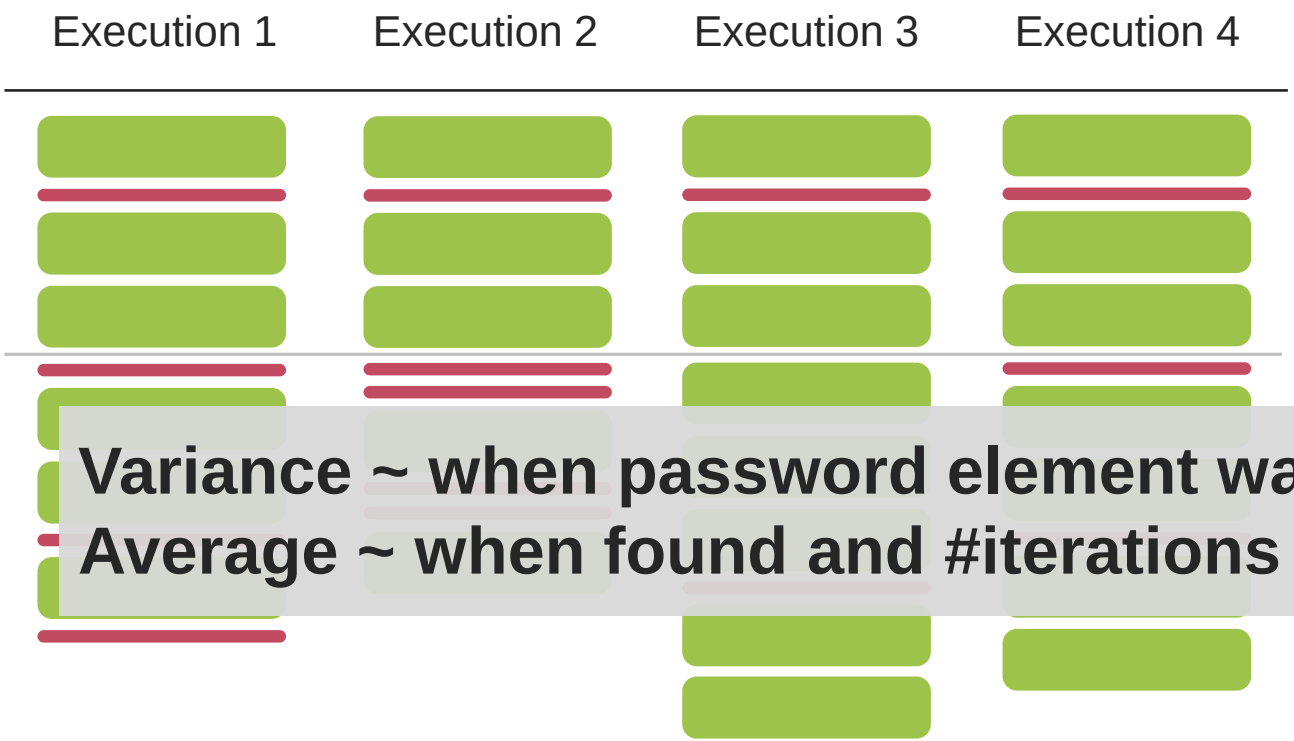


Influence of extra iterations



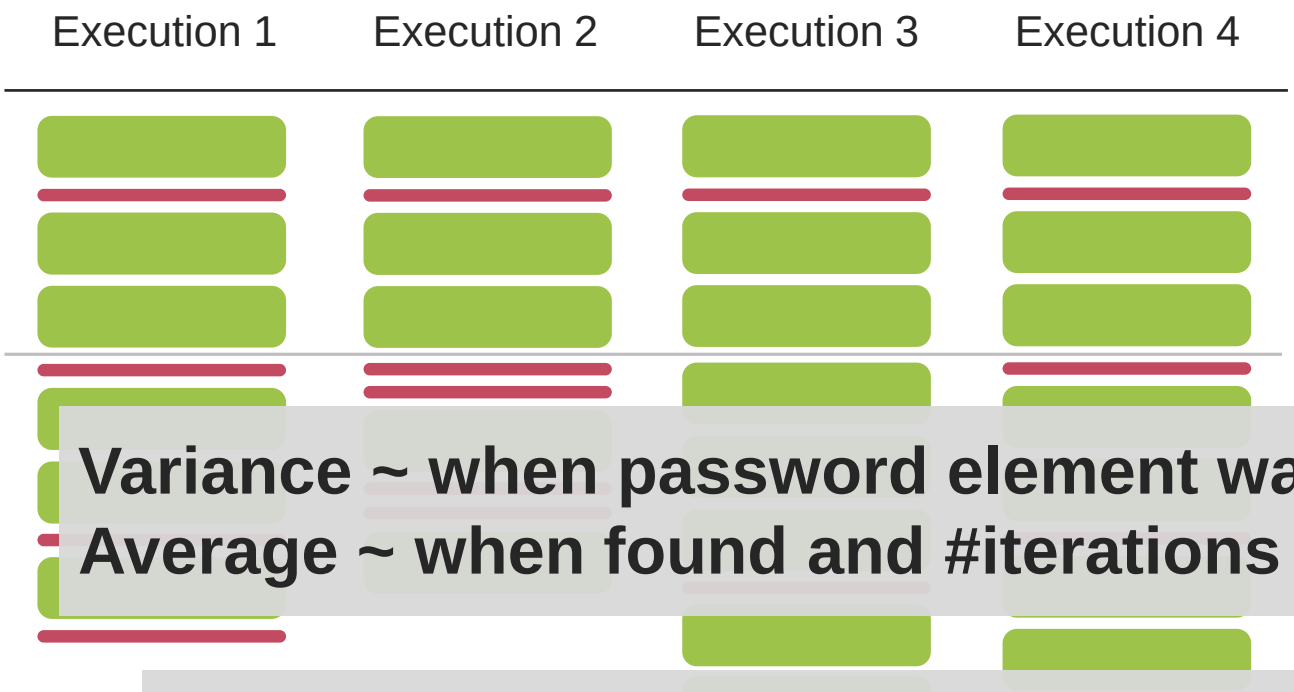
Variance ~ when password element was found

Influence of extra iterations



Variance ~ when password element was found
Average ~ when found and #iterations with big hash

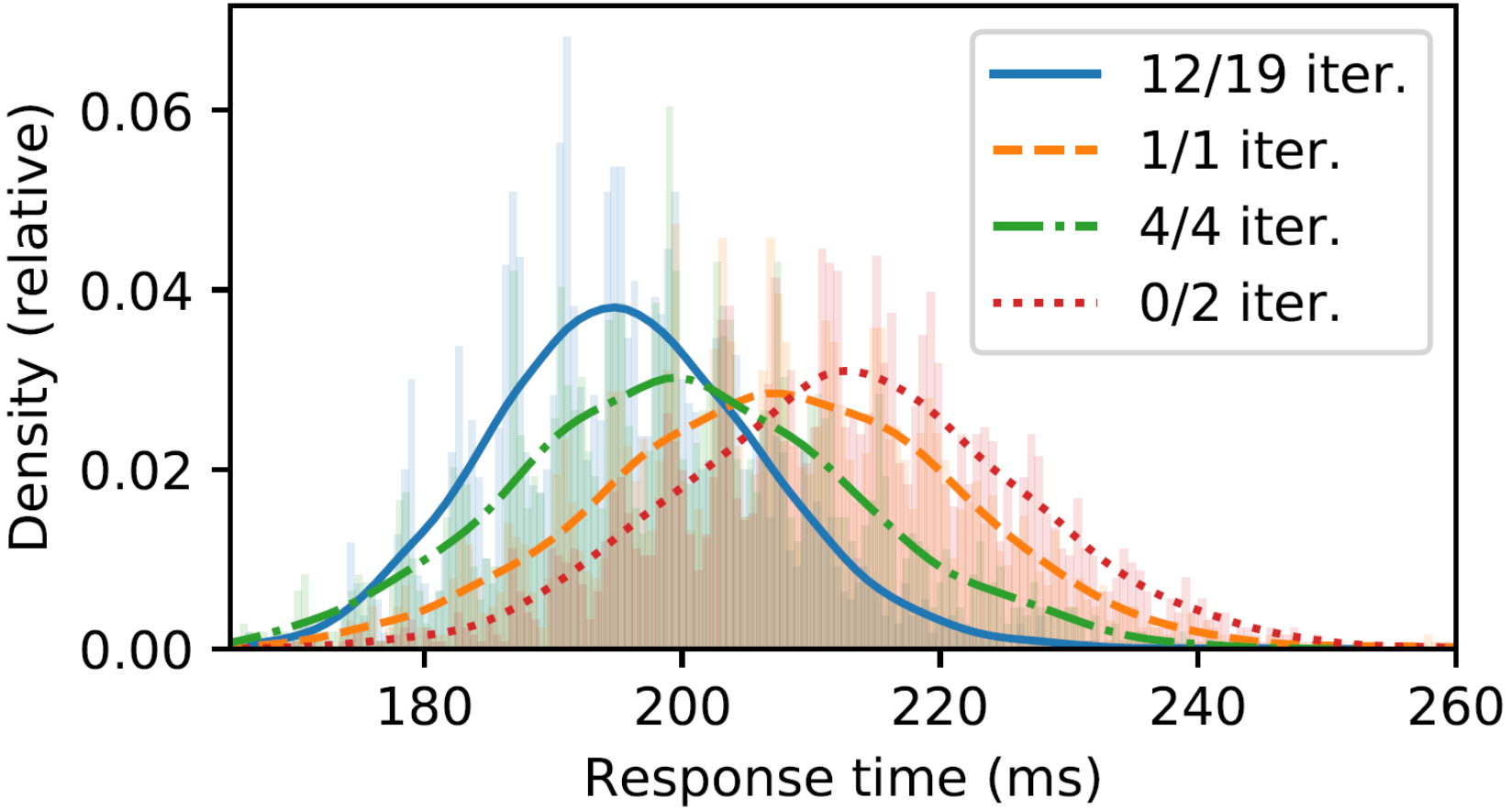
Influence of extra iterations



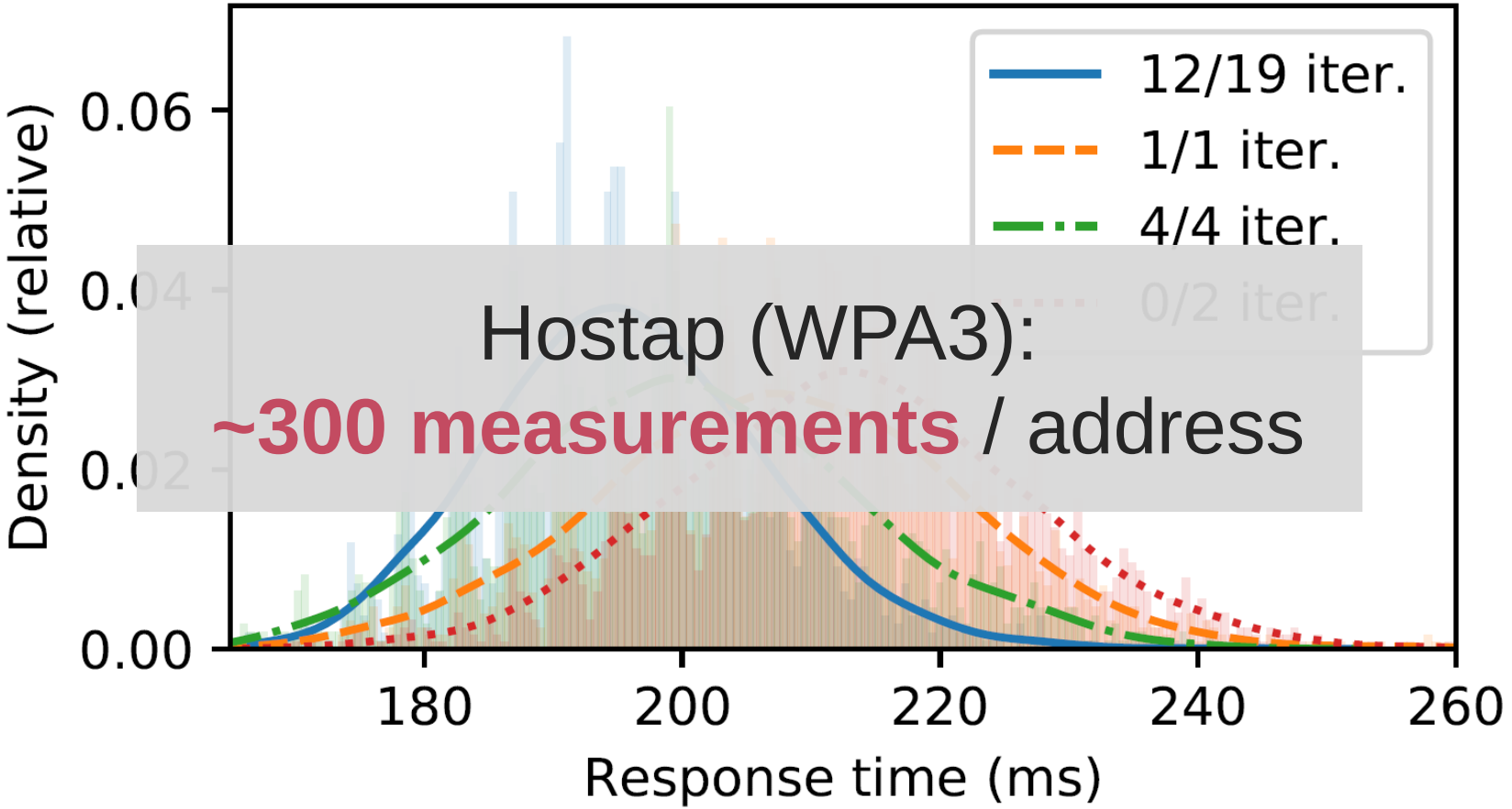
Variance ~ when password element was found
Average ~ when found and #iterations with big hash

→ Again forms a signature of the password

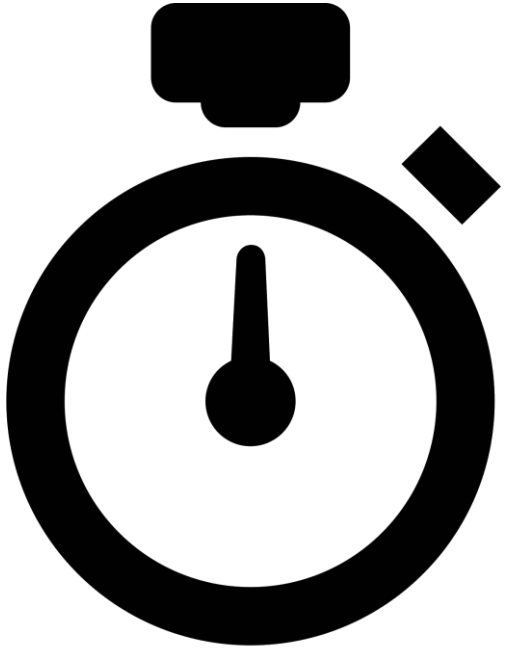
Raspberry Pi 1 B+



Raspberry Pi 1 B+



Hostap (WPA3):
~300 measurements / address



Cache Attacks

Hash-to-curve: Quadratic Residue

```
for (counter = 1; counter < k or not x; counter++)
    value = hash(pw, counter, addr1, addr2)
    if value >= p: continue
    y_sqr = value^3 + a * value + b
    if is_quadratic_residue(y_sqr) and not x:
        x = value
```

NIST curves: use Flush+Reload to detect if code is executed in 1st iteration

Hash-to-curve: Qu

Use as clock to detect in which iteration we are

```
for (counter = 1;
    value = hash(pw, counter, addr1, addr2)
    if value >= p: continue
    y_sqr = value^3 + a * value + b
    if is_quadratic_residue(y_sqr) and not x:
        x = value
```

NIST curves: use Flush+Reload to detect if code is executed in 1st iteration

Hash-to-curve: Bra

Use as clock to detect in which iteration we are

```
for (counter = 1;                                    )
    value = hash(pw, counter, addr1, addr2)
    if value >= p: continue
    y_sqr = value^3 + a * value + b
    if is_quadratic_residue(y_sqr) and not x:
        x = value
        pw = random()
y = sqrt(x^3 + a * x + b)
return (x, y)
```

Brainpool: use Flush+Reload to detect if code is executed in 1st iteration

There’s a lot more!

Implementation-specific vulnerabilities

- Invalid curve attacks, reflection attacks, **bad randomness**

Wi-Fi specific attacks

- Downgrades to WPA2 & denial-of-service

Practical impact

- Brute-force attacks on GPUs: $1 for RockYou database
- 802.11 being updated to use **Shallue-Woestijne-Ulas**

Thank you! Questions?

Lessons learned:

- Must be constant-time and efficient
- Allow offline computation of P
- Discuss impact of bad randomness
- Limit number of parameters (e.g. curves)
- **Dragonfly is hard to implement securely**

https://wpa3.mathyvanhoef.com