From Dragondoom to Dragonstar: Side-channel Attacks and Formally Verified Implementation of WPA3 Dragonfly Handshake

Daniel De Almeida Braga[1], Natalia Kulatova[2], Mohamed Sabt[1], Pierre-Alain Fouque[1], Karthikeyan Bhargavan[3]

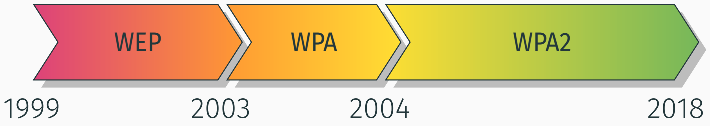
EuroS&P - July 2023

IRISA 1 Université de Rennes 1 cnrs 1 moz://a 2 Inria 3

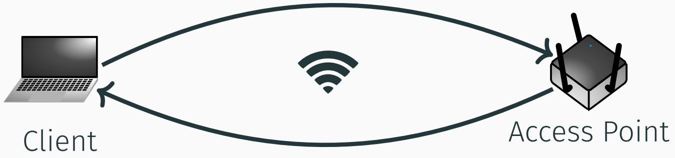
Side Channels in Dragonfly/SAE (WPA3)

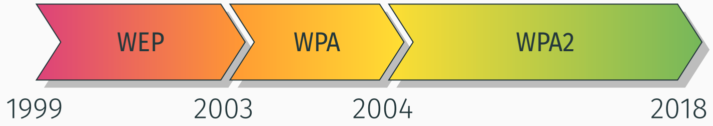
Toward Secure Wi-Fi Protocols...

Client
Access Point

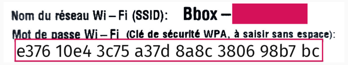


Toward Secure Wi-Fi Protocols...





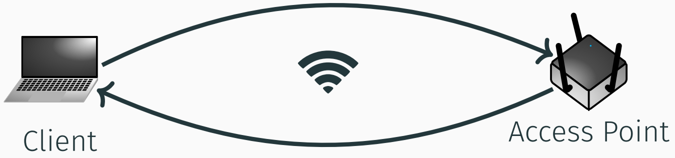
Offline dictionary attack

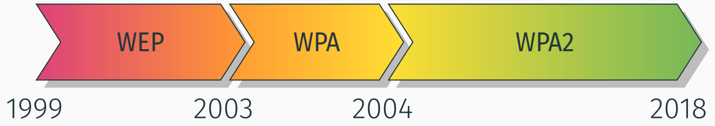

Nom du réseau Wi – Fi (SSID): **Bbox –** [redacted]

Mot de passe Wi – Fi (Clé de sécurité WPA, à saisir sans espace):

e376 10e4 3c75 a37d 8a8c 3806 98b7 bc

Toward Secure Wi-Fi Protocols...

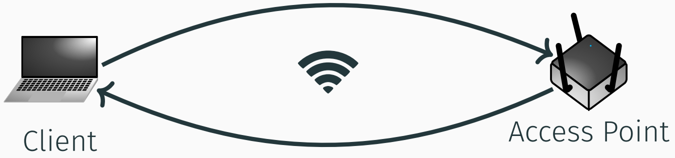


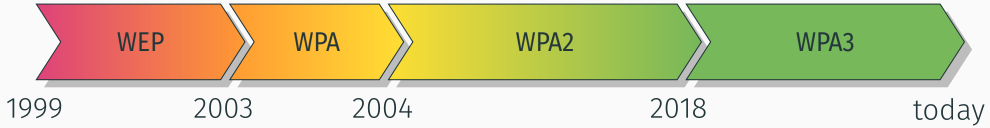


Offline dictionary attack

KRACK attack

Toward Secure Wi-Fi Protocols...



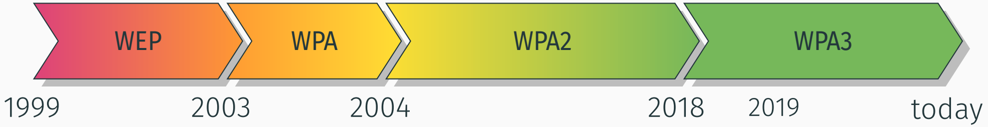


+ More secure

+ Based on a PAKE (Dragonfly[1])

[1] D. Harkins. *Dragonfly Key Exchange*. RFC 7664. 2015

... But Still not Bulletproof

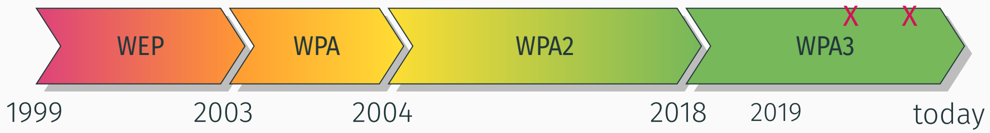


+ More secure
+ Based on a PAKE (Dragonfly)

Dragonblood[1] attacks

[1] M. Vanhoef and E. Ronen. *Dragonblood: Analyzing the Dragonfly Handshake of WPA3 and EAP-pwd.* In IEEE S&P'20

… But Still not Bulletproof

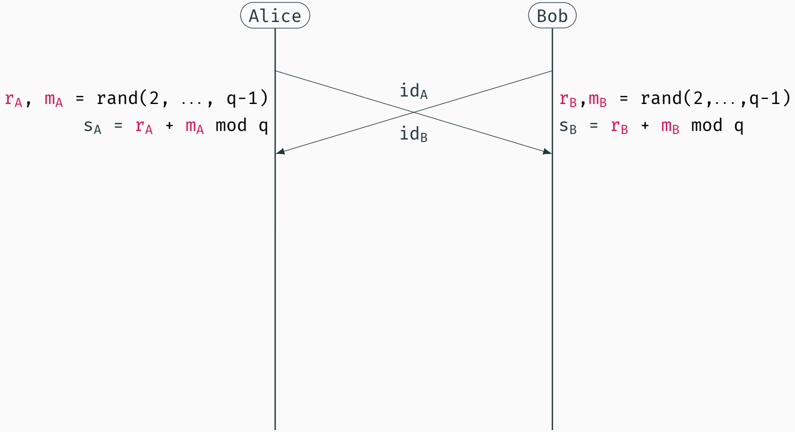


+ More secure
+ Based on a PAKE (Dragonfly)

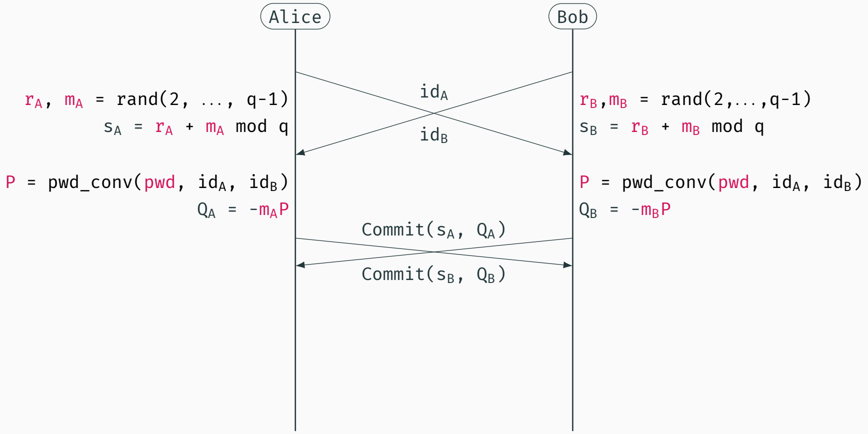
Dragonblood[1] attacks

[1] M. Vanhoef and E. Ronen. *Dragonblood: Analyzing the Dragonfly Handshake of WPA3 and EAP-pwd.* In IEEE S&P'20

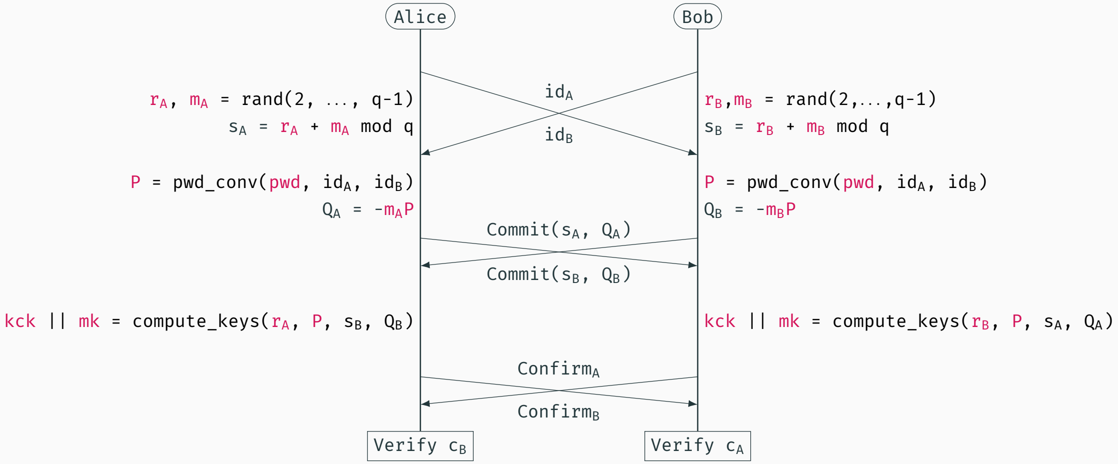
Dragonfly / SAE - A Balanced PAKE


Alice
Bob
r_A, m_A = rand(2, ..., q-1)
s_A = r_A + m_A mod q
id_A
id_B
r_B,m_B = rand(2,...,q-1)
s_B = r_B + m_B mod q

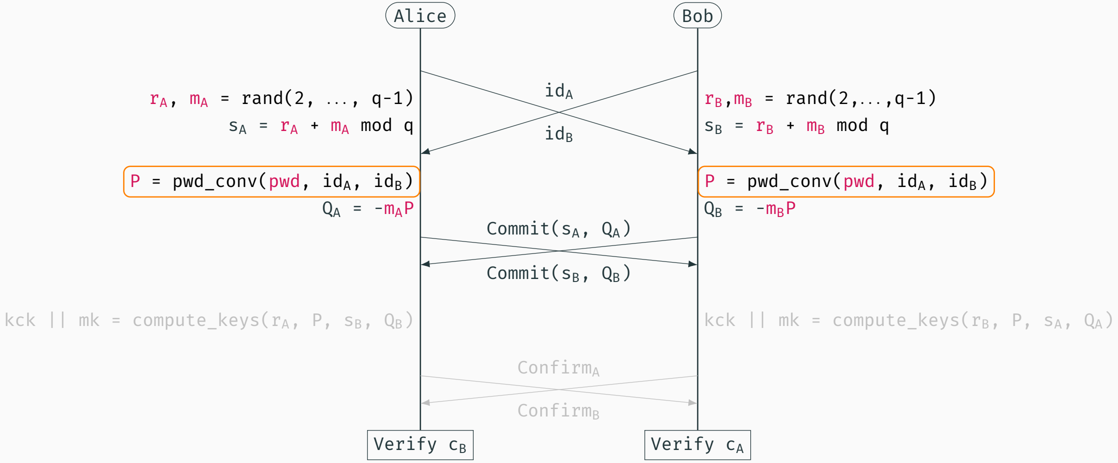
Dragonfly / SAE - A Balanced PAKE


Alice
Bob
r_A, m_A = rand(2, ..., q-1)
s_A = r_A + m_A mod q
id_A
id_B
r_B,m_B = rand(2,...,q-1)
s_B = r_B + m_B mod q
P = pwd_conv(pwd, id_A, id_B)
Q_A = $-m_A P$
P = pwd_conv(pwd, id_A, id_B)
Q_B = $-m_B P$
Commit(s_A, Q_A)
Commit(s_B, Q_B)

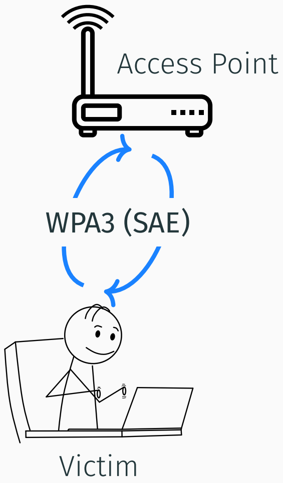
Dragonfly / SAE - A Balanced PAKE


Alice
Bob
r_A, m_A = rand(2, ..., q-1)
s_A = r_A + m_A mod q
id_A
id_B
r_B,m_B = rand(2,...,q-1)
s_B = r_B + m_B mod q
P = pwd_conv(pwd, id_A, id_B)
Q_A = $-m_AP$
P = pwd_conv(pwd, id_A, id_B)
Q_B = $-m_BP$
Commit(s_A, Q_A)
Commit(s_B, Q_B)
kck || mk = compute_keys(r_A, P, s_B, Q_B)
kck || mk = compute_keys(r_B, P, s_A, Q_A)
$Confirm_A$
$Confirm_B$
Verify c_B
Verify c_A

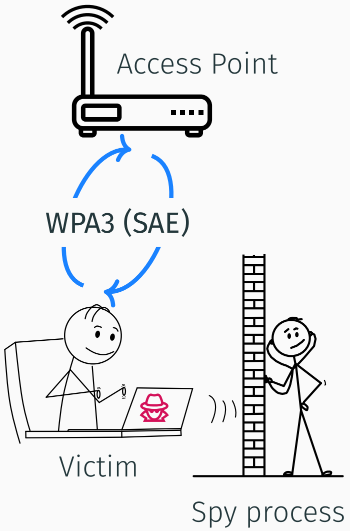
Dragonfly / SAE - A Balanced PAKE


Alice
Bob
r_A, m_A = rand(2, ..., q-1)
$s_A = r_A + m_A \bmod q$
id_A
id_B
r_B, m_B = rand(2,...,q-1)
$s_B = r_B + m_B \bmod q$
P = pwd_conv(pwd, id_A, id_B)
$Q_A = -m_A P$
P = pwd_conv(pwd, id_A, id_B)
$Q_B = -m_B P$
Commit(s_A, Q_A)
Commit(s_B, Q_B)
kck || mk = compute_keys(r_A, P, s_B, Q_B)
kck || mk = compute_keys(r_B, P, s_A, Q_A)
Confirm_A
Confirm_B
Verify c_B
Verify c_A

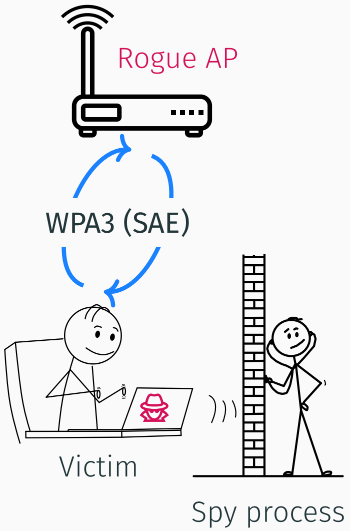
Attack Workflow



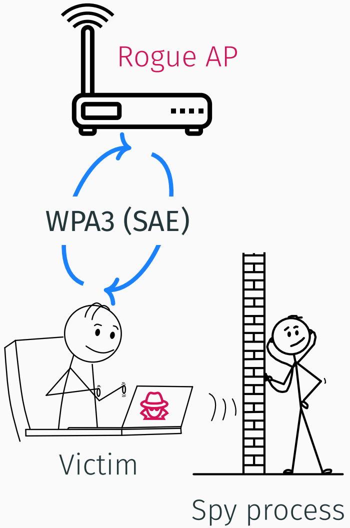
Attack Workflow



Attack Workflow



Attack Workflow



Spying/Data Acquisition

- Implementation specific
- Usually noisy measurement

Comparison metric: Signal to Noise ratio

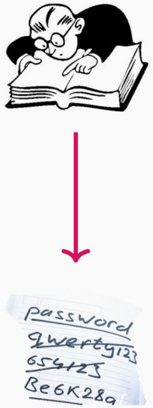
Attack Workflow

Leaked information

Trace parsing

Attack Workflow

Offline Dictionary Attack

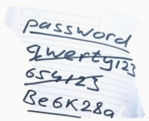


Remaining passwords

Attack Workflow

Offline Dictionary Attack

`H(secret) = 10...`



Remaining passwords

Attack Workflow

Offline Dictionary Attack

X	H(x)
secret	10..
pwd_1	
pwd_2	
pwd_3	
...	
pwd_n	

Remaining passwords

Attack Workflow

Offline Dictionary Attack

x	H(x)
secret	10..
pwd_1	01..
pwd_2	10..
pwd_3	11..
...	...
pwd_n	10..

Remaining passwords

Attack Workflow

Offline Dictionary Attack

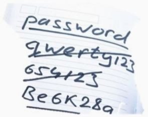
x	H(x)
secret	10..
pwd_1	01..
pwd_2	10..
pwd_3	11..
...	...
pwd_n	10..

Remaining passwords

Attack Workflow

Offline Dictionary Attack

x	H(x || pub$_1$)	H(x || pub$_2$)
secret	10..	00..
pwd$_1$	01..	X
pwd$_2$	10..	00..
pwd$_3$	11..	X
…	…	…
pwd$_n$	10..	11..



Remaining passwords

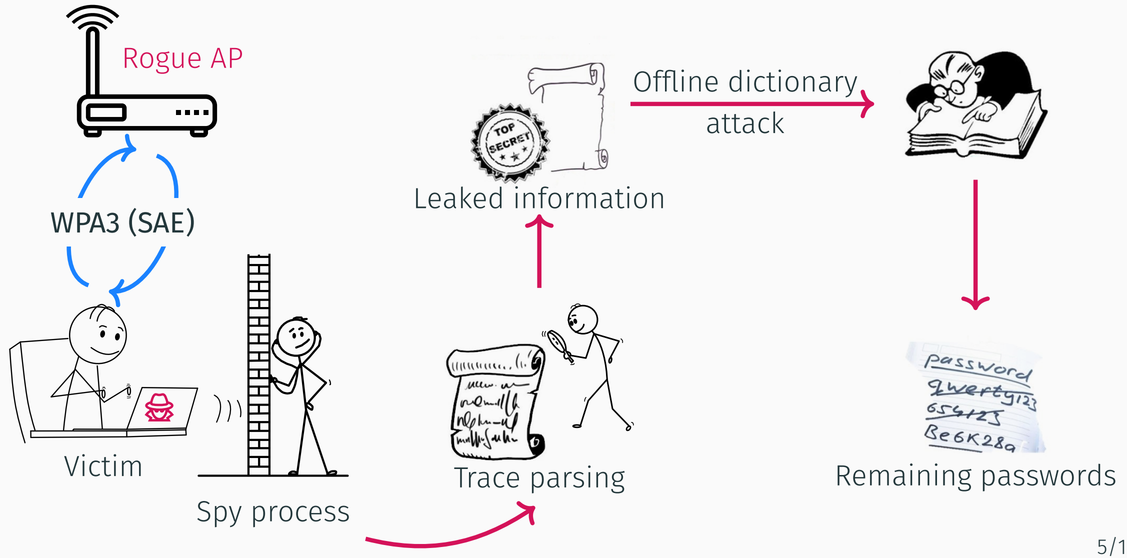
Attack Workflow

Offline Dictionary Attack

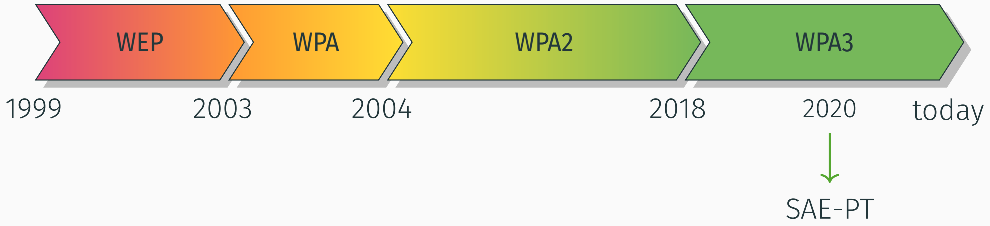
x	H(x || pub_1)	H(x || pub_2)
secret	10..	00..
pwd_1	01..	X
pwd_2	10..	00..
pwd_3	11..	X
...	...	...
pwd_n	10..	11..

Remaining passwords

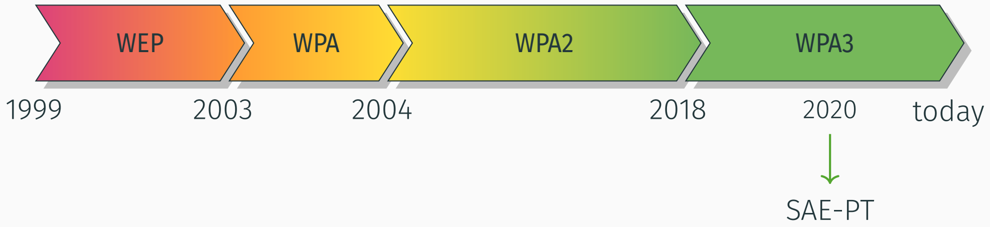
Attack Workflow



Improving the Password Conversion



Improving the Password Conversion



- Better password conversion (SSWU)
  - Deterministic
  - Straightforward constant-time implementation
- ⚠ **Not** backward compatible

Looking Under the Hood

We mostly analyzed Wi-Fi daemons...

🧐

... what about their dependencies, like crypto libraries?

SAE - Probabilistic Password Conversion (EC)

```
def HuntingAndPecking(pwd, $MAC_A$, $MAC_B$, ec)

    seed = Hash($MAC_A$, $MAC_B$, pwd, i)
    $x_{cand}$ = KDF(seed, label)

    is $x_{cand}$ a point's coordinate?

      x, $seed_x$ = $x_{cand}$, seed
      pwd = get_random()

  y = set_compressed_point(x, $seed_x$, ec)
  return (x, y)
```

SAE - Probabilistic Password Conversion (EC)

```
def HuntingAndPecking(pwd, $MAC_A$, $MAC_B$, ec)

    seed = Hash($MAC_A$, $MAC_B$, pwd, i)
    $x_{cand}$ = KDF(seed, label)

    is $x_{cand}$ a point's coordinate?

      x, $seed_x$ = $x_{cand}$, seed
      pwd = get_random()

  y = set_compressed_point(x, $seed_x$, ec)
  return (x, y)
```

SAE - Probabilistic Password Conversion (EC)

```
def HuntingAndPecking(pwd, $MAC_A$, $MAC_B$, ec)

    seed = Hash($MAC_A$, $MAC_B$, pwd, i)
    $x_{cand}$ = KDF(seed, label)

    is $x_{cand}$ a point's coordinate?

      x, $seed_x$ = $x_{cand}$, seed
      pwd = get_random()

  y = set_compressed_point(x, $seed_x$, ec)
  return (x, y)
```

Looking Under the Hood

```
def set_compressed_point(x, fmt, ec)
```

- Branching on the compression format
- Affects SAE (legacy version)
- 1-bit leakage
- Narrow scope outside of Dragonfly

Looking Under the Hood

`def set_compressed_point(x, fmt, ec)`

- Branching on the compression format
- Affects SAE (legacy version)
- 1-bit leakage
- Narrow scope outside of Dragonfly

`def bin2bn(buf, buf_length)`

- Skipping leading 0 bytes
- Affects both SAE and SAE-PT
- 8-bit leakage with proba 1/256
- Wide scope (targets utility function)

Looking Under the Hood

`def set_compressed_point(x, fmt, ec)`

- Branching on the compression format
- Affects SAE (legacy version)
- 1-bit leakage
- Narrow scope outside of Dragonfly

`def bin2bn(buf, buf_length)`

- Skipping leading 0 bytes
- Affects both SAE and SAE-PT
- 8-bit leakage with proba 1/256
- Wide scope (targets utility function)

Affected projects:

- hostap/wpa_supplicant with OpenSSL/WolfSSL
- iwd with ell
- FreeRadius with OpenSSL

”Obviously” Vulnerable, yet Difficult to Exploit

- Very few conditional instructions (one cache line or less)
- Many false positives with ”vanilla” Flush+Reload
- Using existing attack to create a new distinguisher

Abuse prefetching behaviors to create a new distinguisher!

Prefetcher-based Side Channel

```
def set_compressed_point(x, fmt, ec):
     y = compute_y(x, ec)

    if y = fmt mod 2:               }
       y = ec.p - y                 } A
                                    }
    P = init_point(x, y, ec)        }
    [...]                           } B

    return P
```

Prefetcher-based Side Channel

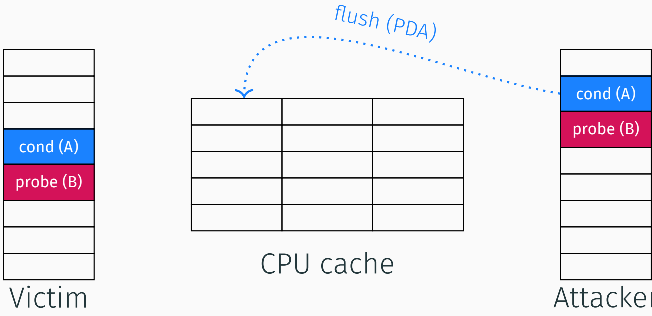
```
def set_compressed_point(x, fmt, ec):
     y = compute_y(x, ec)

    if y = fmt mod 2:
      y = ec.p - y                 } A

    P = init_point(x, y, ec)
    [...]                          } B

    return P
```

Prefetcher-based Side Channel

```
def set_compressed_point(x, fmt, ec):
     y = compute_y(x, ec)

    if y = fmt mod 2:
      y = ec.p - y

    P = init_point(x, y, ec)
    [...]

    return P
```

A

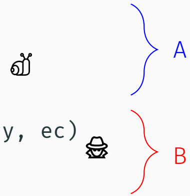
B

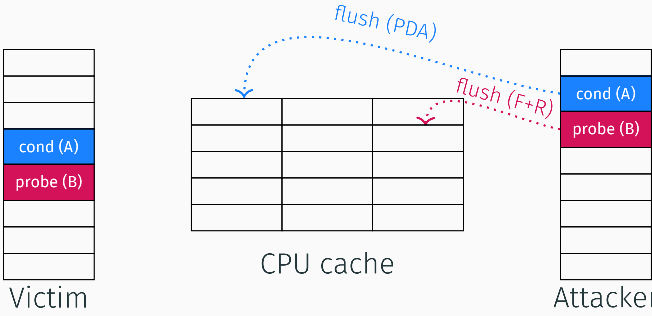


nb hits: 0

Prefetcher-based Side Channel

```
def set_compressed_point(x, fmt, ec):
     y = compute_y(x, ec)

    if y = fmt mod 2:
      y = ec.p - y

    P = init_point(x, y, ec)
    [...]

    return P
```

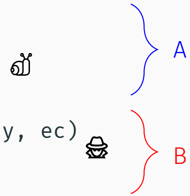


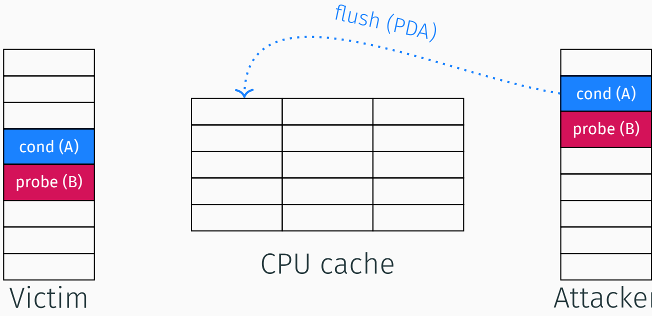


nb hits: 0

Prefetcher-based Side Channel

```
def set_compressed_point(x, fmt, ec):
     y = compute_y(x, ec)

    if y = fmt mod 2:
      y = ec.p - y

    P = init_point(x, y, ec)
    [...]

    return P
```

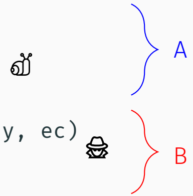


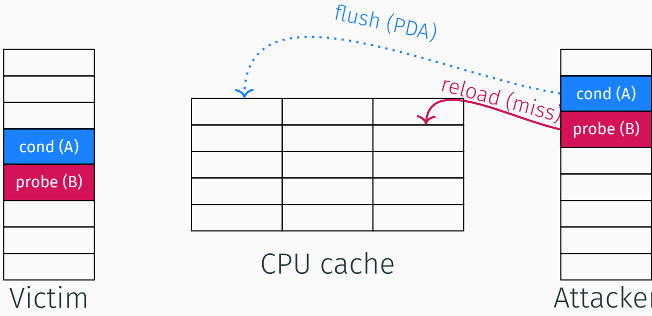


nb hits: 0

Prefetcher-based Side Channel

```
def set_compressed_point(x, fmt, ec):
     y = compute_y(x, ec)

    if y = fmt mod 2:
      y = ec.p - y

    P = init_point(x, y, ec)
    [...]

    return P
```

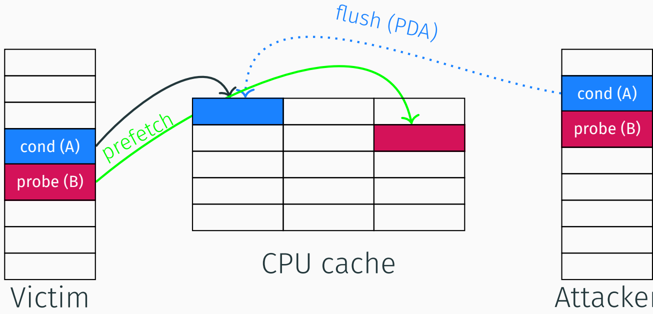




nb hits: 0

Prefetcher-based Side Channel

```
def set_compressed_point(x, fmt, ec):
     y = compute_y(x, ec)

→   if y = fmt mod 2:
      y = ec.p - y

    P = init_point(x, y, ec)
    [...]

    return P
```



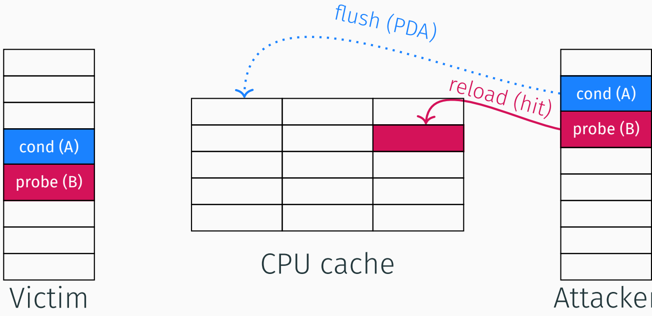
nb hits: 0

Prefetcher-based Side Channel

```
def set_compressed_point(x, fmt, ec):
     y = compute_y(x, ec)

→   if y = fmt mod 2:
      y = ec.p - y

    P = init_point(x, y, ec)
    [...]

    return P
```

$\left.\right\}$ A

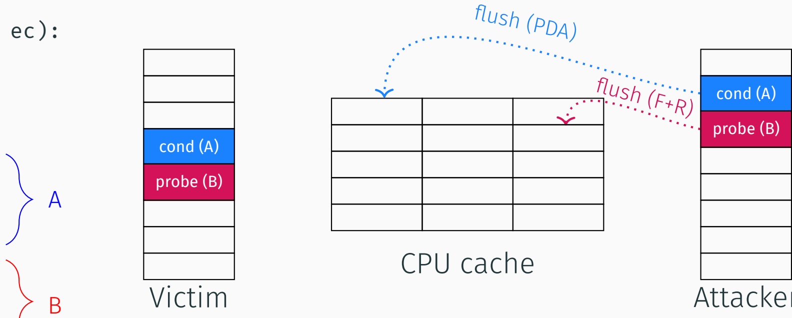
$\left.\right\}$ B



nb hits: 1

Prefetcher-based Side Channel

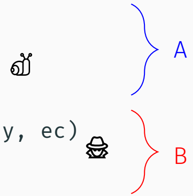
```
def set_compressed_point(x, fmt, ec):
     y = compute_y(x, ec)

→   if y = fmt mod 2:
      y = ec.p - y

    P = init_point(x, y, ec)
    [...]

    return P
```

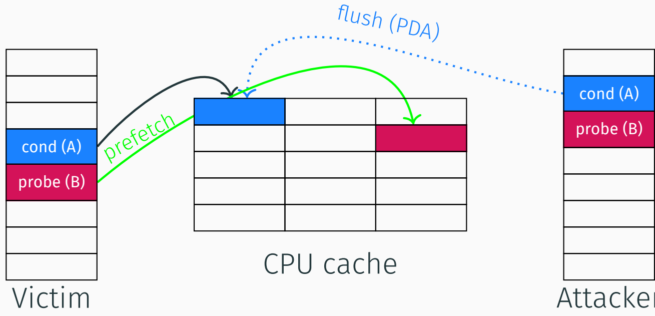


nb hits: 1

Prefetcher-based Side Channel

```
def set_compressed_point(x, fmt, ec):
     y = compute_y(x, ec)

    if y = fmt mod 2:
→     y = ec.p - y

    P = init_point(x, y, ec)
    [...]

    return P
```

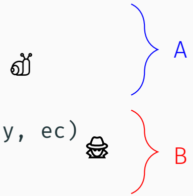


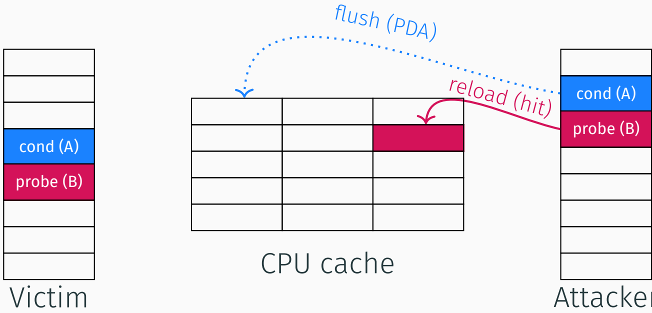


nb hits: 1

Prefetcher-based Side Channel

```
def set_compressed_point(x, fmt, ec):
     y = compute_y(x, ec)

    if y = fmt mod 2:
→     y = ec.p - y

    P = init_point(x, y, ec)
    [...]

    return P
```





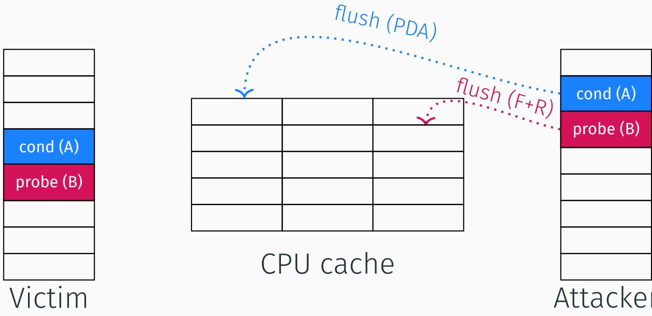
nb hits: 2

Prefetcher-based Side Channel

```
def set_compressed_point(x, fmt, ec):
     y = compute_y(x, ec)

    if y = fmt mod 2:
→     y = ec.p - y

    P = init_point(x, y, ec)
    [...]

    return P
```

$\left.\right\}$ A

$\left.\right\}$ B



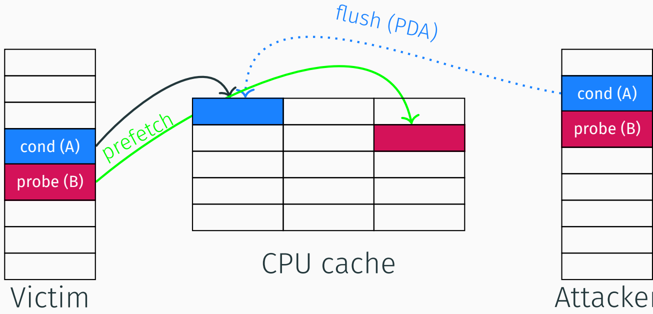
nb hits: 2

Prefetcher-based Side Channel

```
def set_compressed_point(x, fmt, ec):
     y = compute_y(x, ec)

    if y = fmt mod 2:
→     y = ec.p - y

    P = init_point(x, y, ec)
    [...]

    return P
```

$\left.\right\}$ A

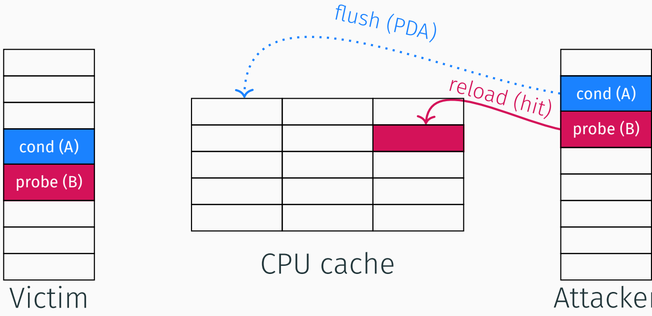
$\left.\right\}$ B



nb hits: 2

Prefetcher-based Side Channel

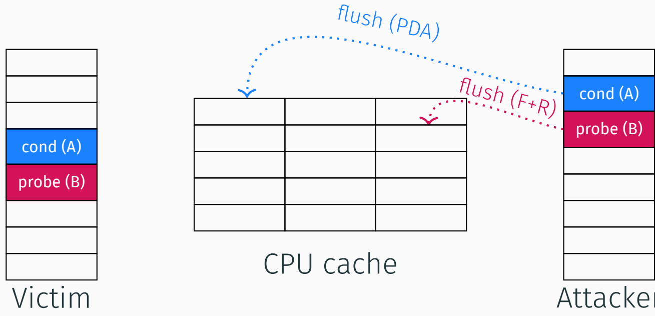
```
def set_compressed_point(x, fmt, ec):
     y = compute_y(x, ec)

    if y = fmt mod 2:
→     y = ec.p - y
                                         } A
    P = init_point(x, y, ec)
    [...]                                } B

    return P
```



nb hits: 3

Prefetcher-based Side Channel

```
def set_compressed_point(x, fmt, ec):
     y = compute_y(x, ec)

    if y = fmt mod 2:
→     y = ec.p - y

    P = init_point(x, y, ec)
    [...]

    return P
```

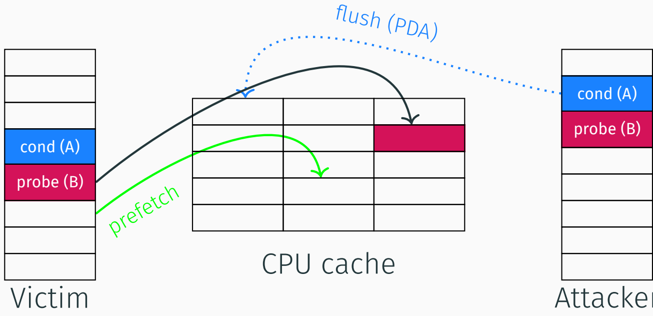


nb hits: 3

Prefetcher-based Side Channel

```
def set_compressed_point(x, fmt, ec):
     y = compute_y(x, ec)

    if y = fmt mod 2:
      y = ec.p - y

→   P = init_point(x, y, ec)
    [...]

    return P
```

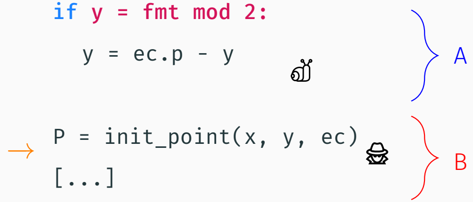
$$\left.\begin{matrix}\text{if y = fmt mod 2:}\\ \text{y = ec.p - y}\end{matrix}\right\} A \qquad \left.\begin{matrix}\text{P = init\_point(x, y, ec)}\\ [...]\end{matrix}\right\} B$$

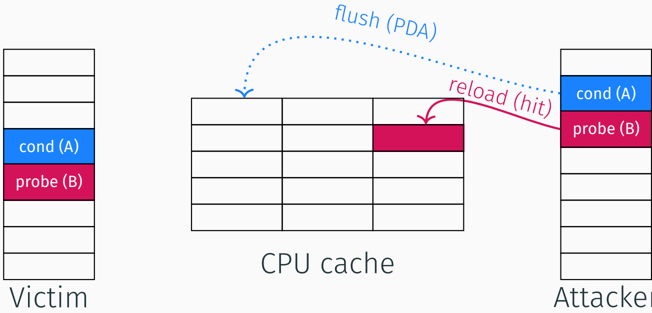


nb hits: 3

Prefetcher-based Side Channel

```
def set_compressed_point(x, fmt, ec):
     y = compute_y(x, ec)

    if y = fmt mod 2:
      y = ec.p - y

    P = init_point(x, y, ec)
    [...]

    return P
```





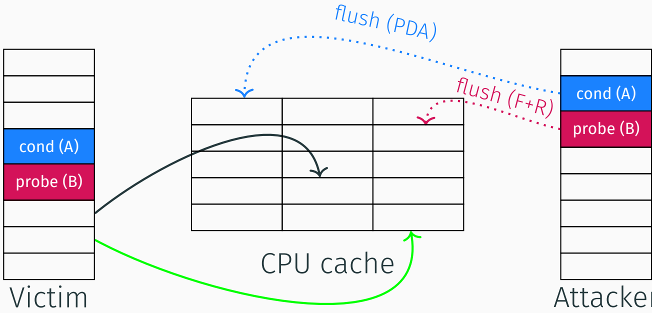
nb hits: 4

Prefetcher-based Side Channel

```
def set_compressed_point(x, fmt, ec):
     y = compute_y(x, ec)

    if y = fmt mod 2:
      y = ec.p - y

    P = init_point(x, y, ec)
    [...]

→   return P
```

$$\left.\begin{array}{l}\texttt{if y = fmt mod 2:}\\ \texttt{y = ec.p - y}\end{array}\right\} A$$

$$\left.\begin{array}{l}\texttt{P = init\_point(x, y, ec)}\\ \texttt{[...]}\end{array}\right\} B$$

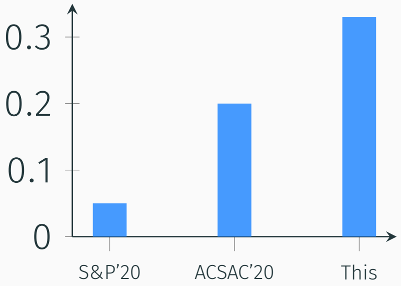


nb hits: 4

Prefetcher-based Side Channel

```
def set_compressed_point(x, fmt, ec):
     y = compute_y(x, ec)

    if y = fmt mod 2:
      y = ec.p - y

    P = init_point(x, y, ec)
    [...]

    return P
```

Very accurate distinguisher, with a better spatial resolution!



Improves Upon Previous Attack

Data Leaked:

- TODO

for a set of public MAC addresses

Amount of Information:

- 1 bit

Practical evaluation:

- 3 measurements get reliable

information

Improves Upon Previous Attack

Data Leaked:

- TODO

for a set of public MAC addresses

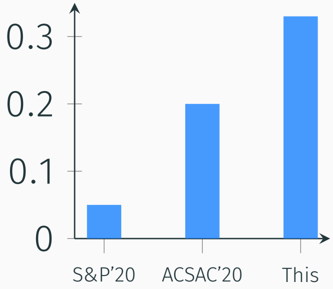
Amount of Information:

- 1 bit

Practical evaluation:

- 3 measurements get reliable information

Very accurate distinguisher, with a better spatial resolution!



Sustainable patch for hostap

- Cryptographic libraries refused to patch
- Many other potential vulnerabilities ($\approx$ 400)

Shall we replace them?

[1] J-K. Zinzindohoué et al. *HACL*: A Verified Modern Cryptographic Library*. In CCS'17

Sustainable patch for hostap

- Cryptographic libraries refused to patch
- Many other potential vulnerabilities (≈ 400)

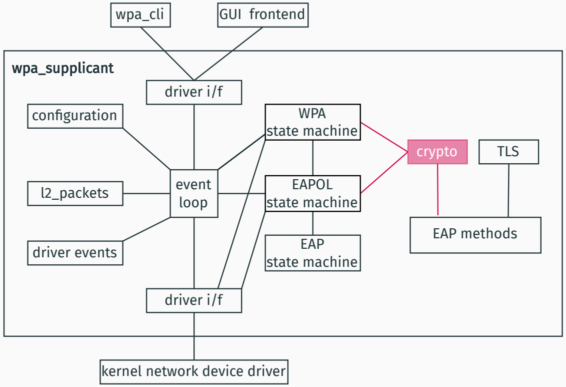
Shall we replace them?

HaCl*: A Formally Verified Cryptographic Library[1]

- Memory-safety
- Functional correctness
- Secret independence

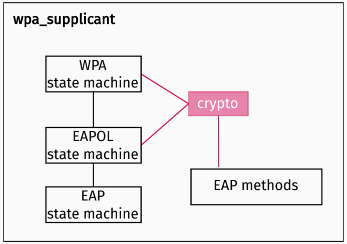
[1] J-K. Zinzindohoué et al. *HACL*: A Verified Modern Cryptographic Library*. In CCS'17

Fixing hostap[1]



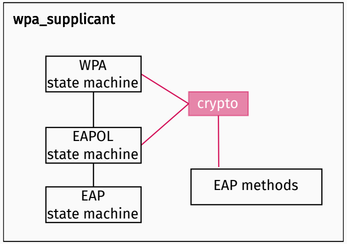
[1] Thank you Alexandre Sanchez for helping with the patch integration

Fixing hostap[1]



[1] Thank you Alexandre Sanchez for helping with the patch integration

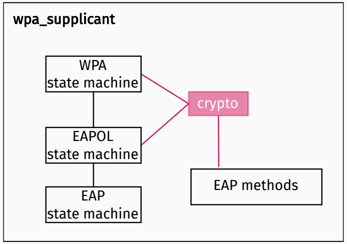
Fixing hostap[1]



```
crypto/
  ...
  crypto.h
  crypto_mbedtls.c
  crypto_openssl.c
  crypto_wolfssl.c
  ...
```

[1] Thank you Alexandre Sanchez for helping with the patch integration

Fixing hostap[1]



```
crypto/
  ...
  crypto.h
  crypto_hacl.c
  crypto_mbedtls.c
  crypto_openssl.c
  crypto_wolfssl.c
  ...
```

[1] Thank you Alexandre Sanchez for helping with the patch integration

Impact

A New Attack

- Dictionary attack (SAE/SAE-PT)
    - Improved signal-to-noise ratio!
    - First side-channel in SAE-PT (supposed to be ct by design)
- New generic gadget
    - Potential impact on many low-level arithmetic functions

Impact

A New Attack

- Dictionary attack (SAE/SAE-PT)
    - Improved signal-to-noise ratio!
    - First side-channel in SAE-PT (supposed to be ct by design)
- New generic gadget
    - Potential impact on many low-level arithmetic functions

A Better Defense

- 3 Security patches (hostap, iwd, FreeRadius)
- Formally verified crypto implementation (HaCl*)
- Benefit from HaCl*'s team support

Impact

A New Attack

- Dictionary attack (SAE/SAE-PT)
  - Improved signal-to-noise ratio!
  - First side-channel in SAE-PT (supposed to be ct by design)
- New generic gadget
  - Potential impact on many low-level arithmetic functions

A Better Defense

- 3 Security patches (hostap, iwd, FreeRadius)
- Formally verified crypto implementation (HaCl*)
- Benefit from HaCl*'s team support

Material available at

- `https://gitlab.inria.fr/ddealmei/artifact_dragondoom`
- `https://gitlab.inria.fr/ddealmei/artifact_dragonstar`