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

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

EuroS&P - July 5th, 2023

University of Rennes, CNRS, IRISA[1]
Mozilla[2]
INRIA, Paris[3]

Toward Secure Wi-Fi Protocols...

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):

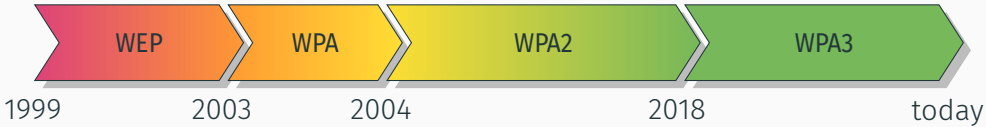
e376 10e4 3c75 a37d 8a8c 3806 98b7 bc

Toward Secure Wi-Fi Protocols...

Offline dictionary attack

KRACK attack

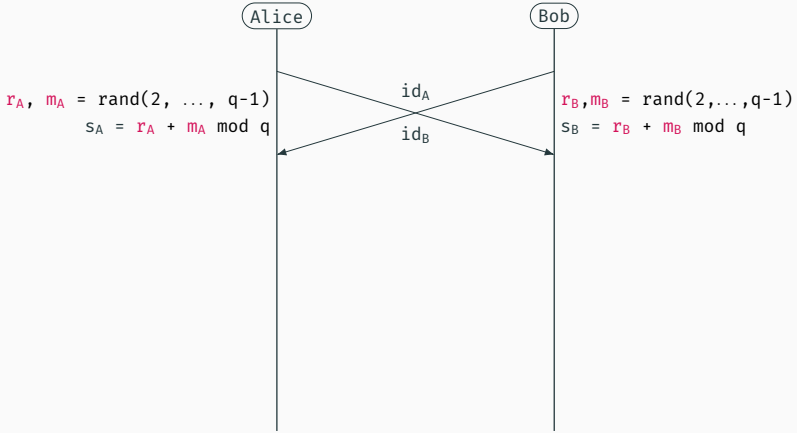
Toward Secure Wi-Fi Protocols...



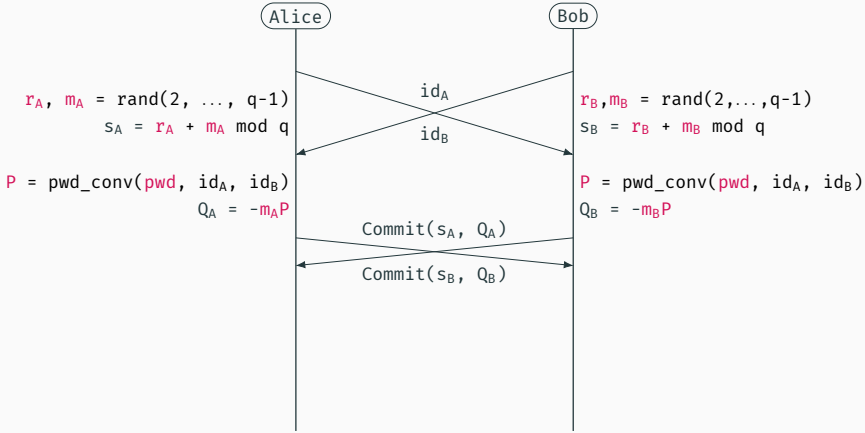
+ More secure
+ Based on a PAKE (Dragonfly[1])

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

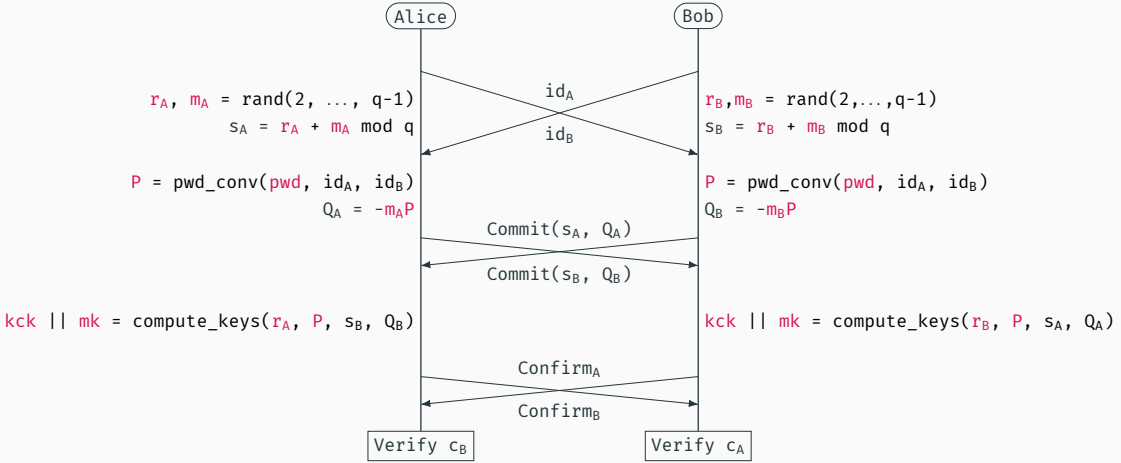
Dragonfly / SAE - A Balanced PAKE



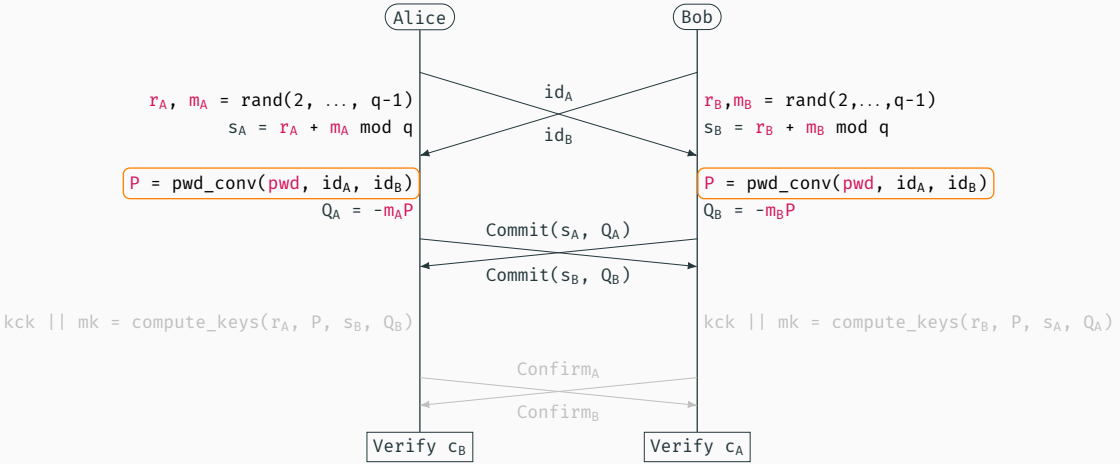
Dragonfly / SAE - A Balanced PAKE



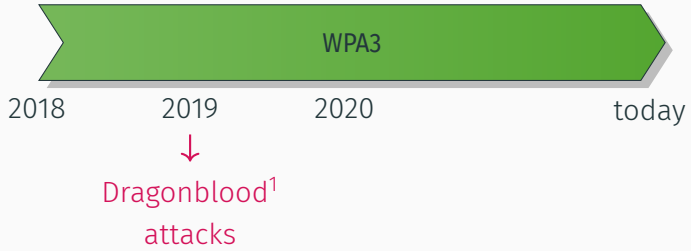
Dragonfly / SAE - A Balanced PAKE



Dragonfly / SAE - A Balanced PAKE



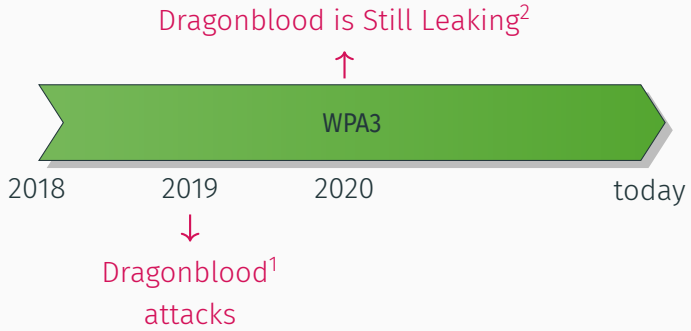
… But Still not Bulletproof



- Weird choice of password conversion method
  - Probabilistic
  - Difficult to implement securely
- Concerned were raised… and confirmed

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

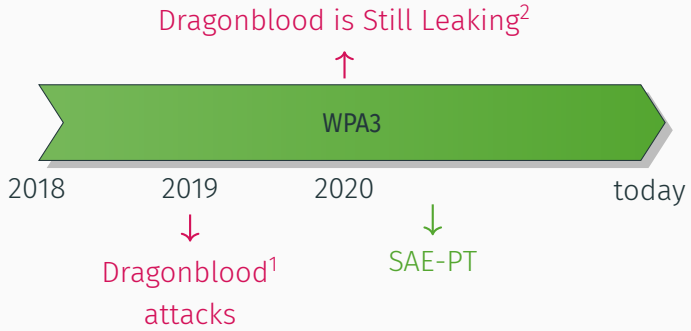
… But Still not Bulletproof



- Weird choice of password conversion method
  - Probabilistic
  - Difficult to implement securely
- Concerned were raised… and confirmed

[1] M. Vanhoef and E. Ronen. *Dragonblood: Analyzing the Dragonfly Handshake of WPA3 and EAP-pwd.* In IEEE S&P'20
[2] D. De Almeida Braga *et al Dragonblood is Still Leaking: Practical Cache-based Side-Channel in the Wild.* In ACSAC '20

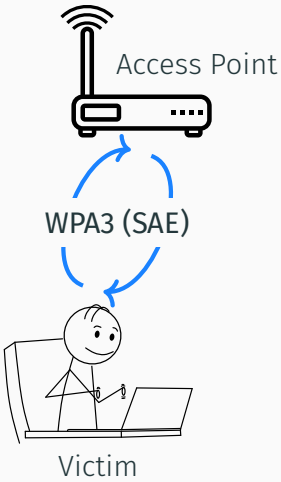
… But Still not Bulletproof



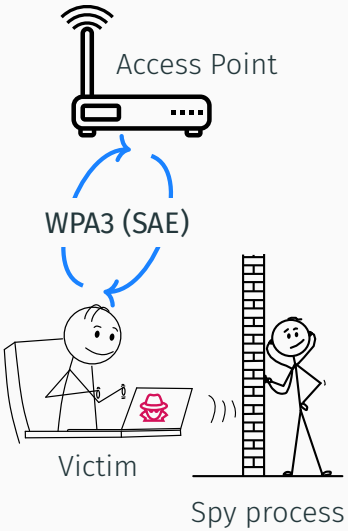
- Better password conversion (SSWU)
  - Deterministic
  - Straightforward constant-time implementation
- ⚠ **Not** backward compatible

[1] M. Vanhoef and E. Ronen. *Dragonblood: Analyzing the Dragonfly Handshake of WPA3 and EAP-pwd.* In IEEE S&P'20
[2] D. De Almeida Braga *et al Dragonblood is Still Leaking: Practical Cache-based Side-Channel in the Wild.* In ACSAC '20

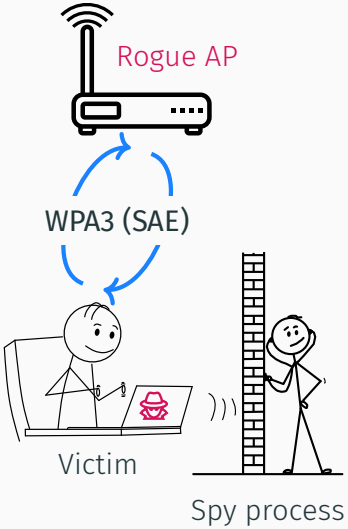
Attack Workflow

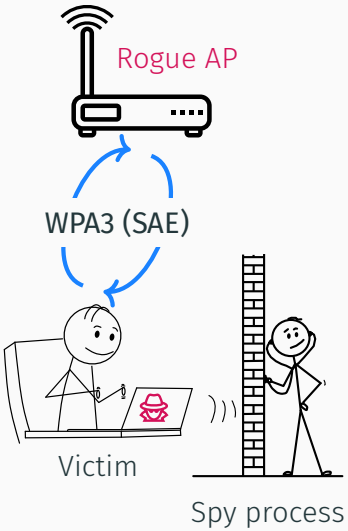


Attack Workflow



Attack Workflow





Spying/Data Acquisition

- Implementation specific
- Usually noisy measurement

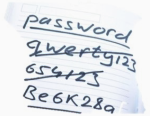
Comparison metric: Signal to Noise ratio

Attack Workflow

Leaked information

Trace parsing

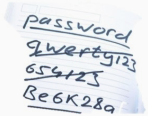
Offline Dictionary Attack



Remaining passwords

Offline Dictionary Attack

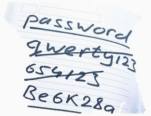
`H(secret) = 10...`



Remaining passwords

Offline Dictionary Attack

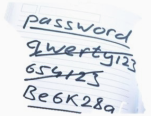
x	H(x)
secret	10..
pwd_1	
pwd_2	
pwd_3	
...	
pwd_n	



Remaining passwords

Offline Dictionary Attack

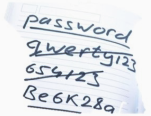
x	H(x)
secret	10..
pwd_1	01..
pwd_2	10..
pwd_3	11..
...	...
pwd_n	10..



Remaining passwords

Offline Dictionary Attack

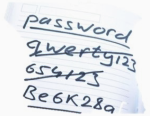
x	H(x)
secret	10..
pwd_1	01..
pwd_2	10..
pwd_3	11..
...	...
pwd_n	10..



Remaining passwords

Offline Dictionary Attack

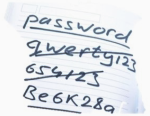
x	H(x || pub_1)	H(x || pub_2)
secret	10..	00..
pwd_1	01..	X
pwd_2	10..	00..
pwd_3	11..	X
...	...	...
pwd_n	10..	11..



Remaining passwords

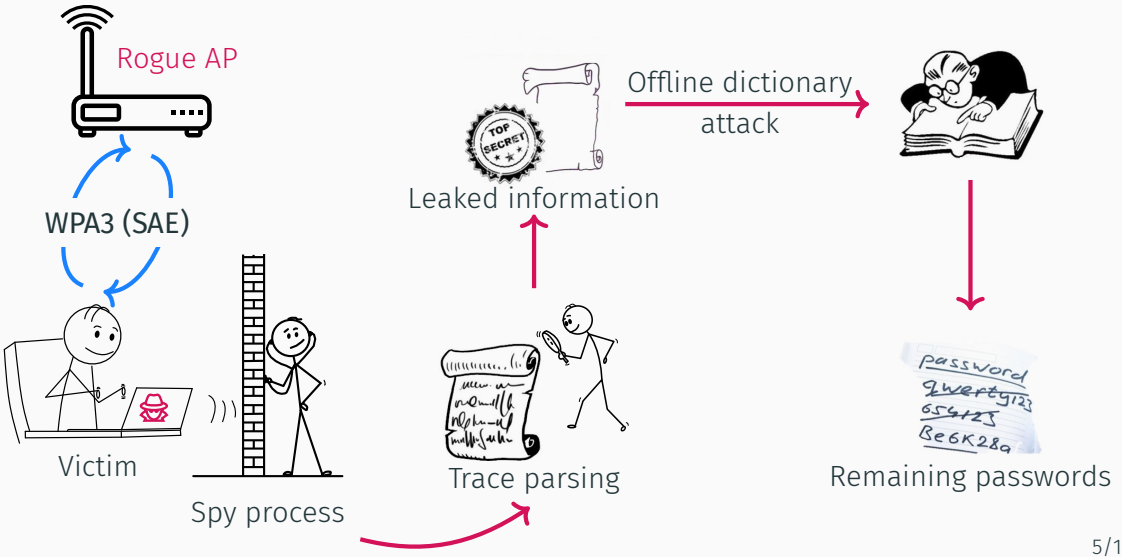
Offline Dictionary Attack

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



Looking Under the Hood

We mostly analyzed Wi-Fi daemons...

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

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

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

    return P
```

A

B

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

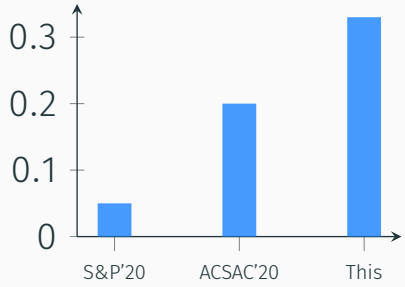
Prefetcher-based Side Channel

```
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

Very accurate distinguisher, with a better spatial resolution!



Sustainable patch for hostap

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

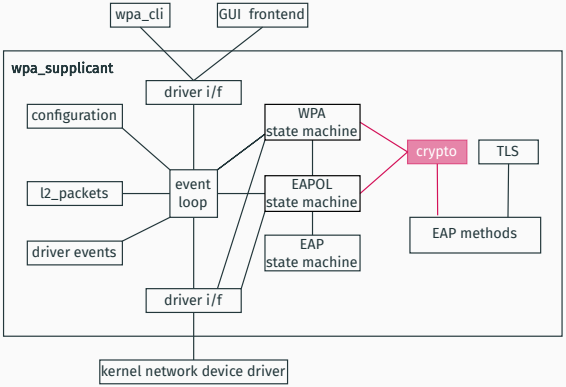
Shall we replace them?

HaCl*: A Formally Verified Cryptographic Library[1]

- Memory-safety
- Functional correctness
- Secret independence

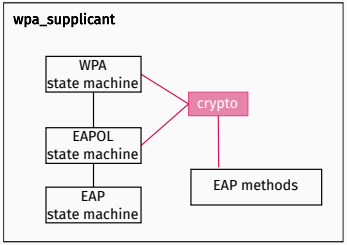
[1] J-K. Zinzindohoué et al. *HACL*: A Verified Modern Cryptographic Library.* In CCS'17

Fixing hostap[1]



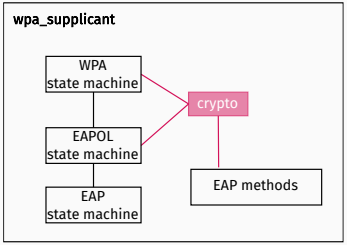
[1] Thank you Alexandre Sanchez for helping with the patch integration

Fixing hostap[1]



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

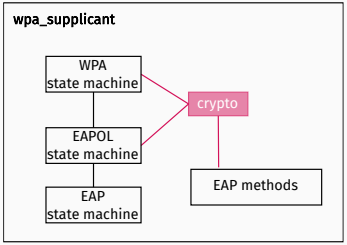
Fixing hostap[1]




```
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`

Appendix

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

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

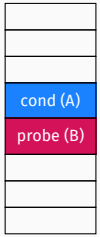
B

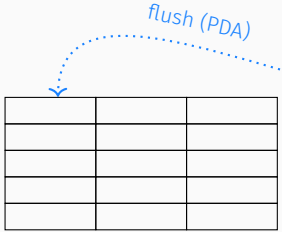
Prefetcher-based Side Channel

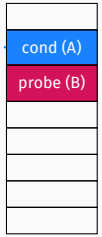
```
def set_compressed_point(x, fmt, ec):
    y = compute_y(x, ec)

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

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

    return P
```

A

B







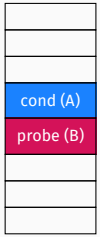
nb hits: 0

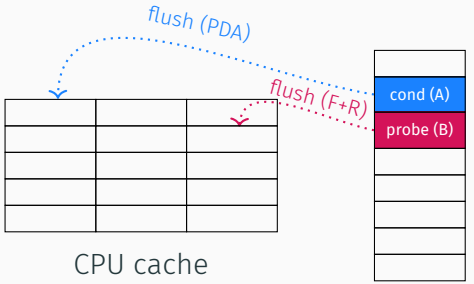
Prefetcher-based Side Channel

```
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
```

A

B





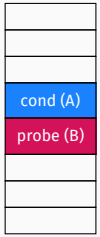
nb hits: 0

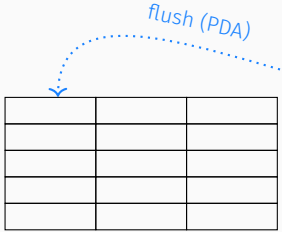
Prefetcher-based Side Channel

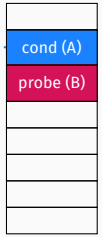
```
def set_compressed_point(x, fmt, ec):
    y = compute_y(x, ec)

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

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

    return P
```

A

B







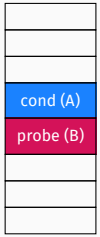
nb hits: 0

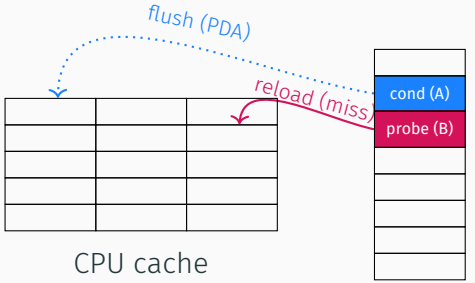
Prefetcher-based Side Channel

```
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
```

A

B





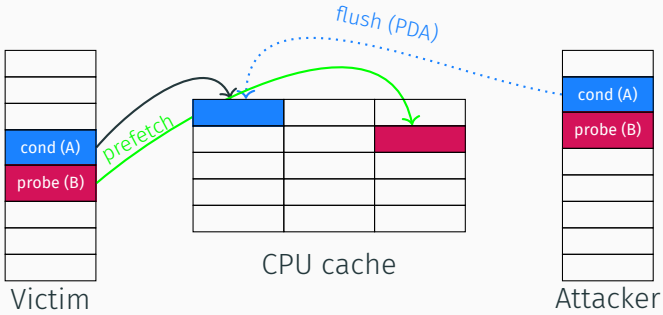
nb hits: 0

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
```

A

B



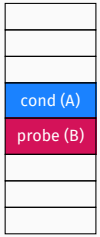
nb hits: 0

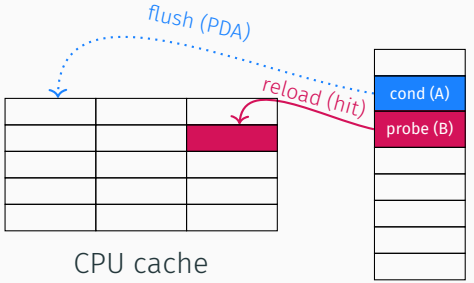
Prefetcher-based Side Channel

```
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
```

A

B





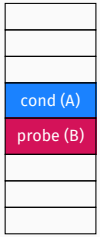
nb hits: 1

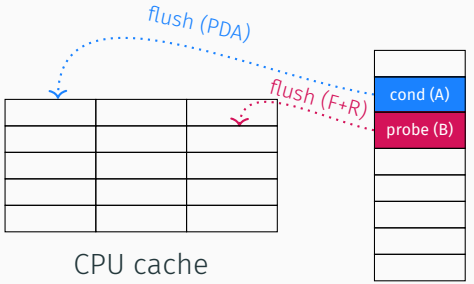
Prefetcher-based Side Channel

```
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
```

A

B





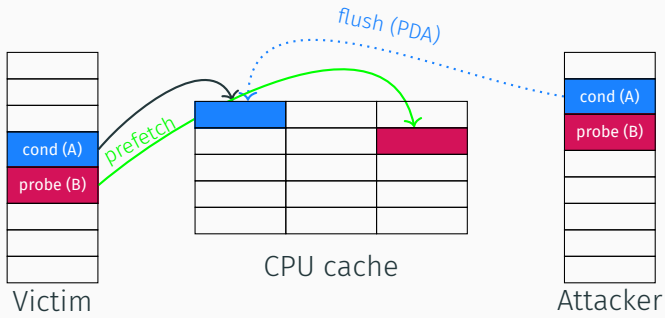
nb hits: 1

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
```

A

B



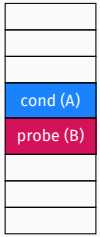
nb hits: 1

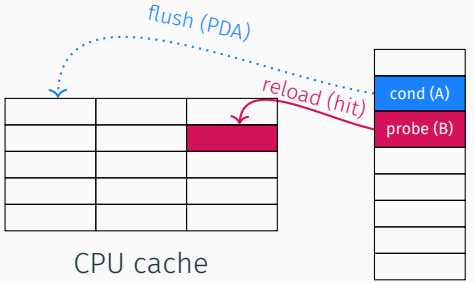
Prefetcher-based Side Channel

```
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
```

A

B





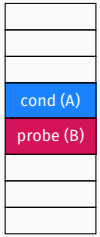
nb hits: 2

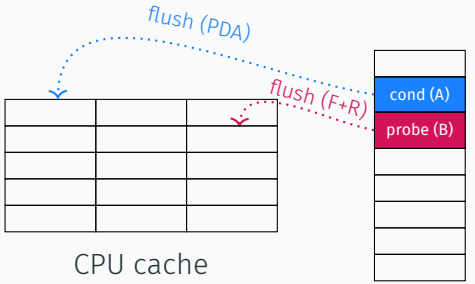
Prefetcher-based Side Channel

```
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
```

A

B





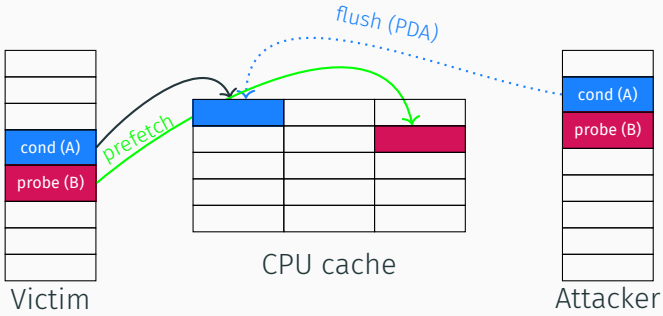
nb hits: 2

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
```

A

B



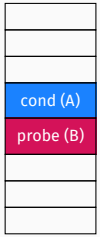
nb hits: 2

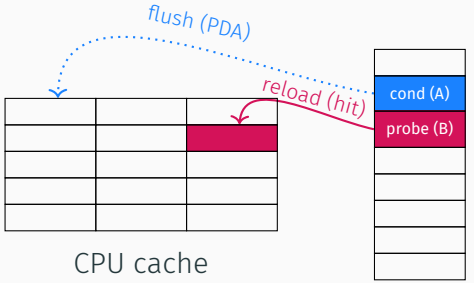
Prefetcher-based Side Channel

```
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
```

A

B





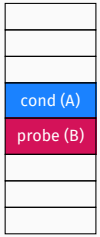
nb hits: 3

Prefetcher-based Side Channel

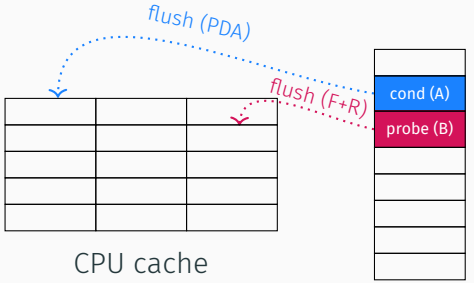
```
def set_compressed_point(x, fmt, ec):
    y = compute_y(x, ec)

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

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

    return P
```

A

B



Victim



CPU cache

Attacker

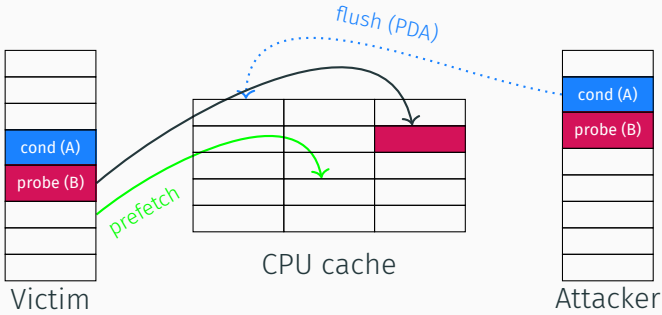
nb hits: 3

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
```

A

B



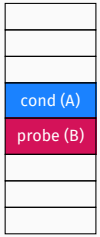
nb hits: 3

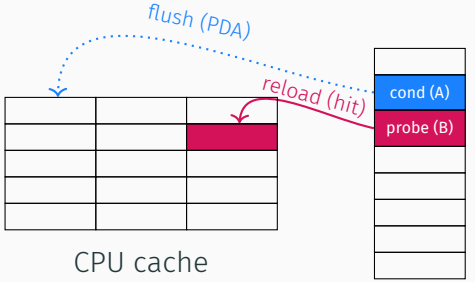
Prefetcher-based Side Channel

```
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
```

A

B





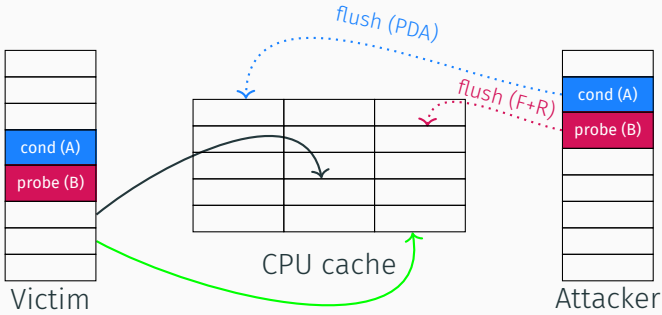
nb hits: 4

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
```

A

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!