Cryptography in the Wild: The Security of Cryptographic Implementations

Daniel De Almeida Braga

Ph.D. Defense - December, 14th 2022

Context and Motivations

Cryptography in the Wild...

- Cryptography has many applications
  - Confidentiality
  - Integrity
  - Authentication
  - ...

Cryptography in the Wild...

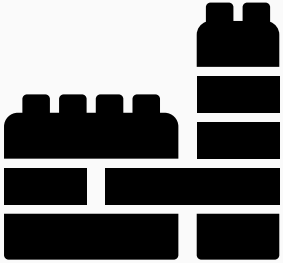
- Cryptography has many applications
  - Confidentiality
  - Integrity
  - Authentication
  - ...
- Protocols are built upon *primitives*

Cryptography in the Wild...

- Cryptography has many applications
  - Confidentiality
  - Integrity
  - Authentication
  - ...
- Protocols are built upon *primitives*

Cryptography in the Wild...

- Cryptography has many applications
  - Confidentiality
  - Integrity
  - Authentication
  - ...
- Protocols are built upon *primitives*

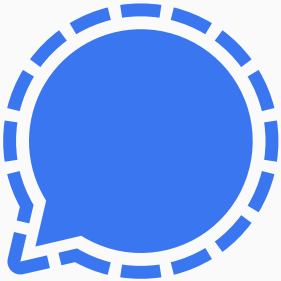


Cryptography in the Wild...

- Cryptography has many applications
  - Confidentiality
  - Integrity
  - Authentication
  - ...
- Protocols are built upon *primitives*
- They vary depending on the constraints

Cryptography in the Wild...

- Cryptography has many applications
  - Confidentiality
  - Integrity
  - Authentication
  - ...
- Protocols are built upon *primitives*
- They vary depending on the constraints



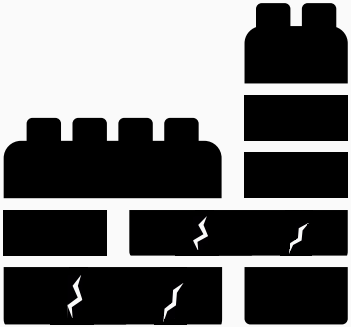
Cryptography in the Wild...

- Cryptography has many applications
  - Confidentiality
  - Integrity
  - Authentication
  - ...
- Protocols are built upon *primitives*
- They vary depending on the constraints

... What About Security?

Multiple security considerations:

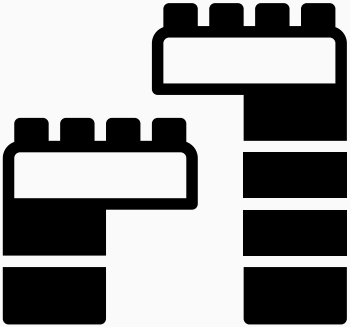
- Primitive security



... What About Security?

Multiple security considerations:

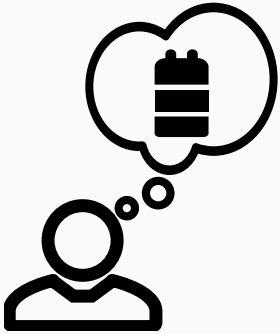
- Primitive security
- Logical security of the protocol



... What About Security?

Multiple security considerations:

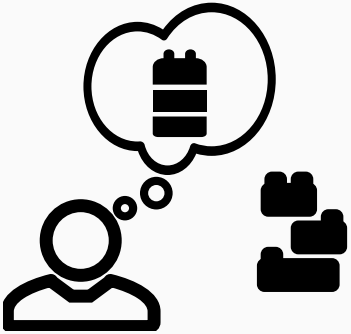
- Primitive security
- Logical security of the protocol
- Implementation security



... What About Security?

Multiple security considerations:

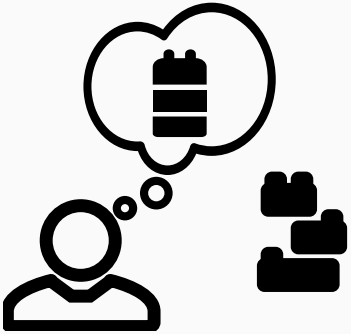
- Primitive security
- Logical security of the protocol
- Implementation security



... What About Security?

Multiple security considerations:

- Primitive security
- Logical security of the protocol
- Implementation security



Cryptographic Implementation Security

Generic bugs

- Buffer overflows
- Arithmetic errors
- Missing verification
- ...

Cryptographic Implementation Security

Generic bugs

- Buffer overflows
- Arithmetic errors
- Missing verification
- ...

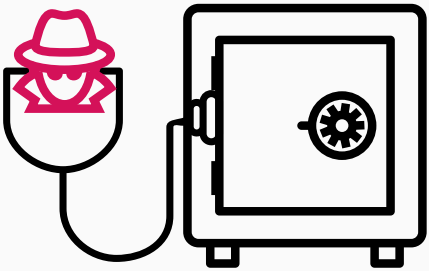
Side channel leakage

Cryptographic Implementation Security

Generic bugs

- Buffer overflows
- Arithmetic errors
- Missing verification
- ...

Side channel leakage



Secret Dependent Execution

```
def processPassword(pwd):
  if "a" in pwd:
     res = long_processing(pwd)
  else:
     res = short_processing(pwd)
   return res
```

Gain information with overall timing:

 0.5 seconds $\Rightarrow$ no a

 10 seconds $\Rightarrow$ a

Secret Dependent Execution

```
def processPassword(pwd):
  if "a" in pwd:
     res = long_processing(pwd)
  else:
     res = short_processing(pwd)
   return res
```

```
def processPassword2(pwd):
  if "a" in pwd:
    res = long_processing(pwd)
  else:
    res = long_processing2(pwd)
  return res
```

Gain information with overall timing:


0.5 seconds $\Rightarrow$ no a


10 seconds $\Rightarrow$ a

Secret Dependent Execution

```
def processPassword(pwd):
  if "a" in pwd:
     res = long_processing(pwd)
  else:
     res = short_processing(pwd)
   return res
```

Gain information with overall timing:

 0.5 seconds $\Rightarrow$ no a

 10 seconds $\Rightarrow$ a

```
def processPassword2(pwd):
  if "a" in pwd:
    res = long_processing(pwd)
  else:
    res = long_processing2(pwd)
  return res
```

Gain information with execution flow:

- Execute `long_processing` $\Rightarrow$ a
- Else, no a in pwd

Secret Independent Execution[1]

- Control flow does not depend on secret

```
if secret:
  [...]
else:
  [...]
```

[1] P. Kocher. *Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems.* In CRYPTO'96.

Secret Independent Execution[1]

- Control flow does not depend on secret

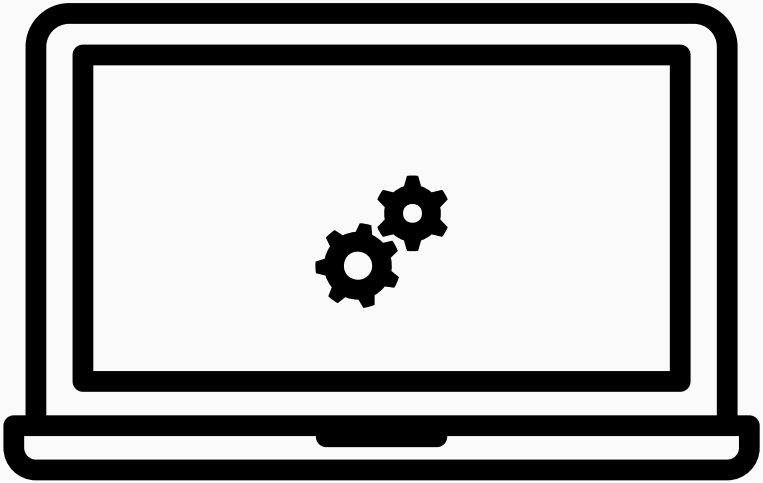
```
if secret:
  [...]
else:
  [...]
```

- Memory access does not depend on secret

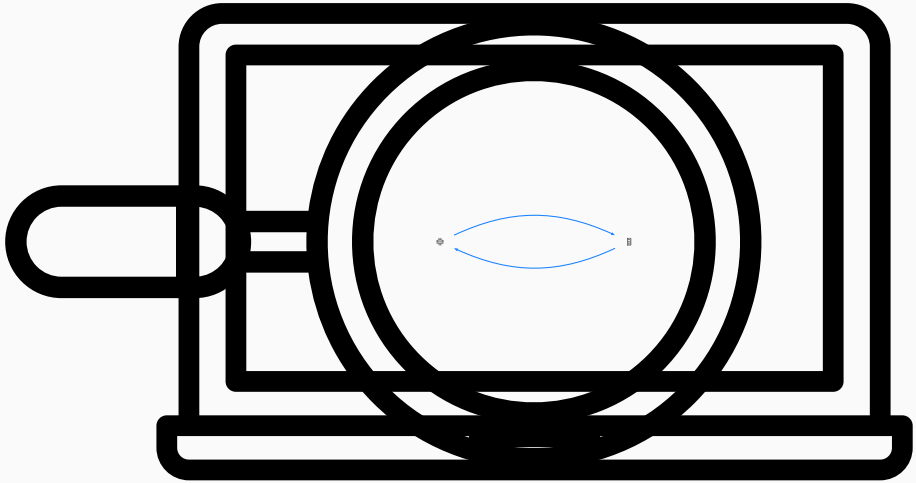
```
x = array[secret]
```

[1] P. Kocher. *Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems.* In CRYPTO'96.

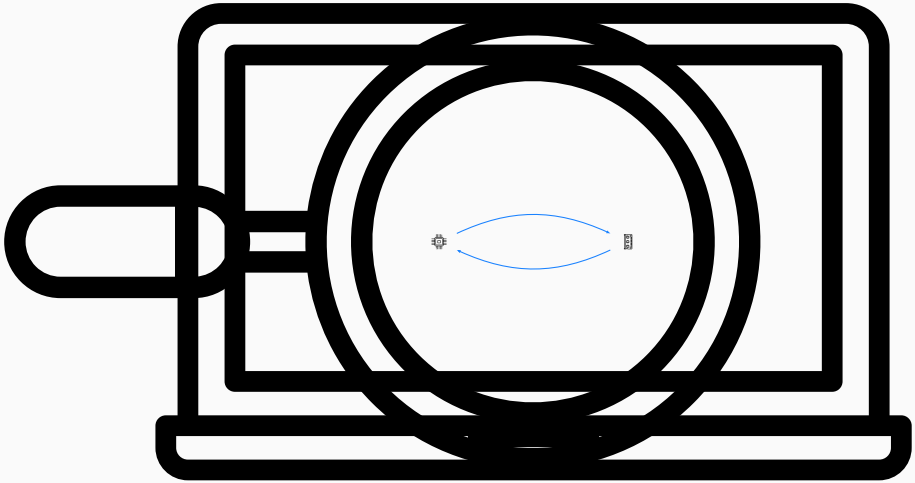
Shared Component - Memory



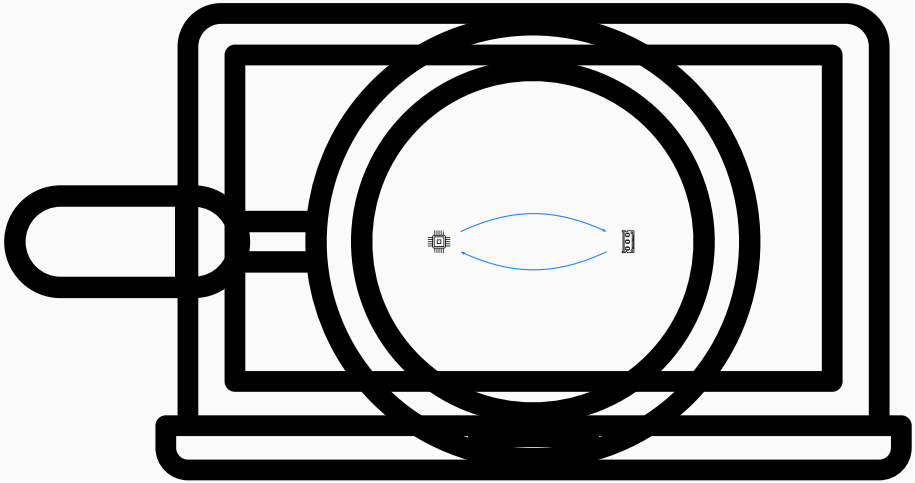
Shared Component - Memory



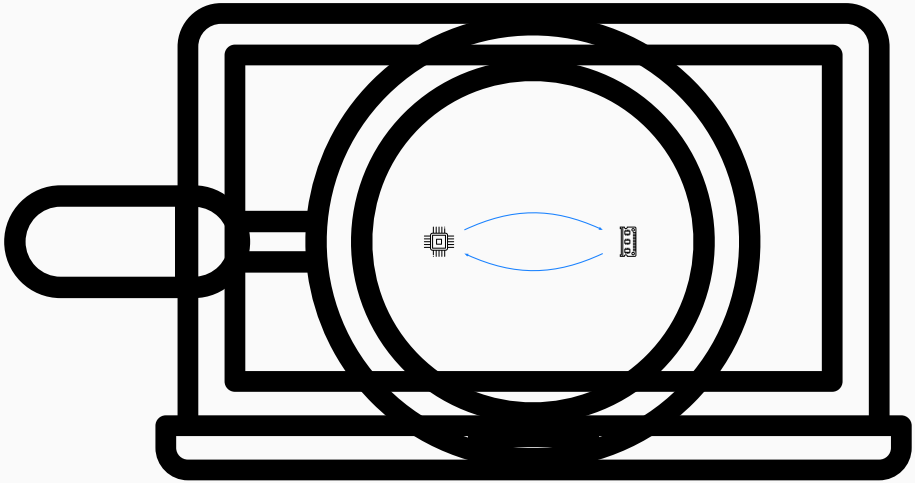
Shared Component - Memory



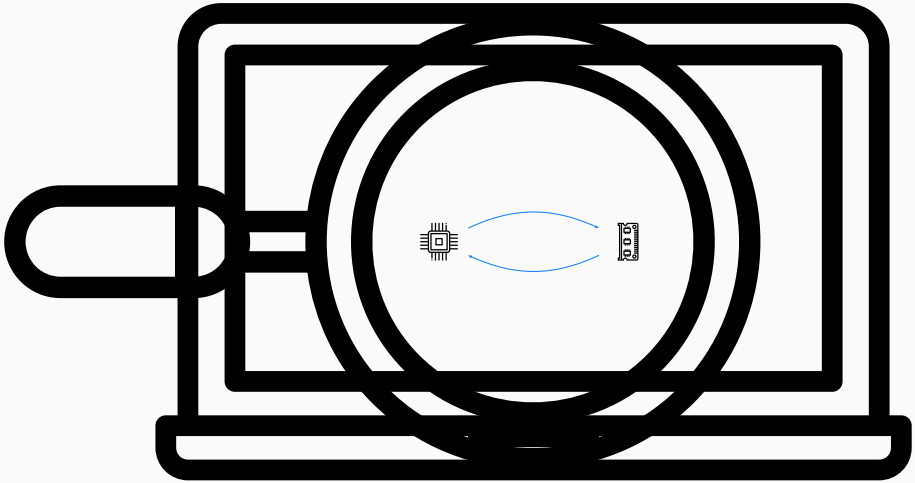
Shared Component - Memory



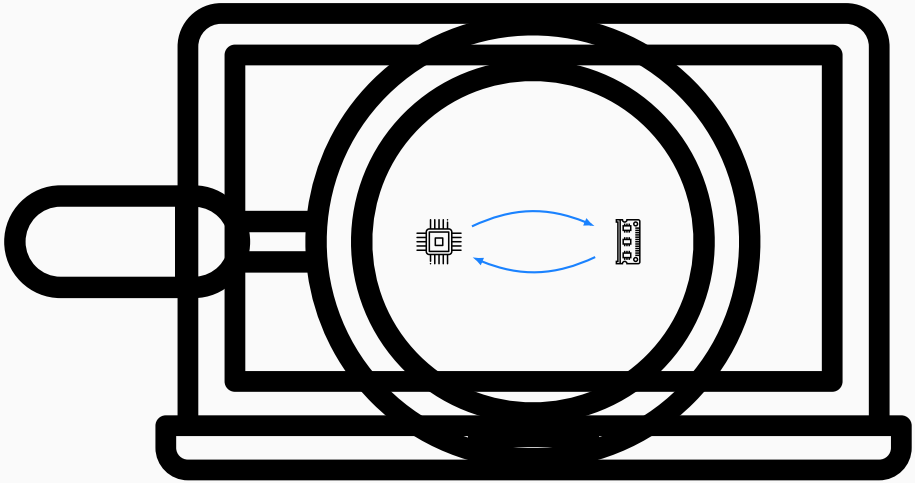
Shared Component - Memory



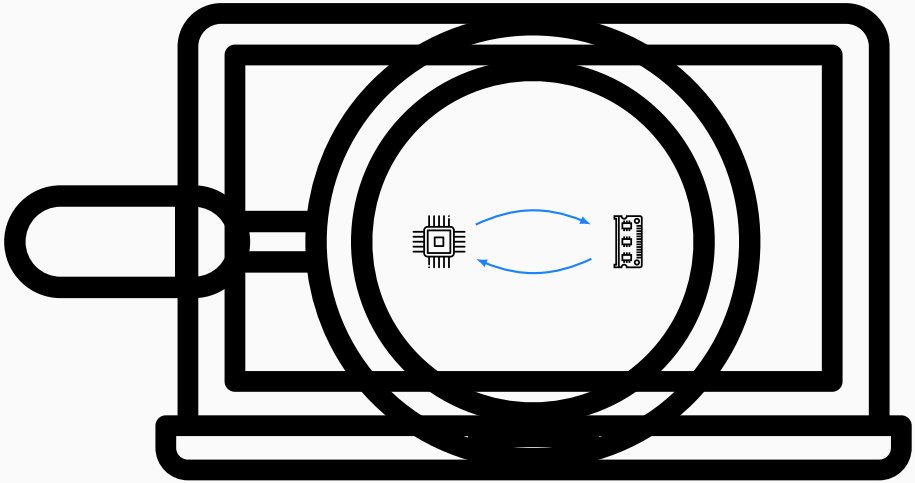
Shared Component - Memory



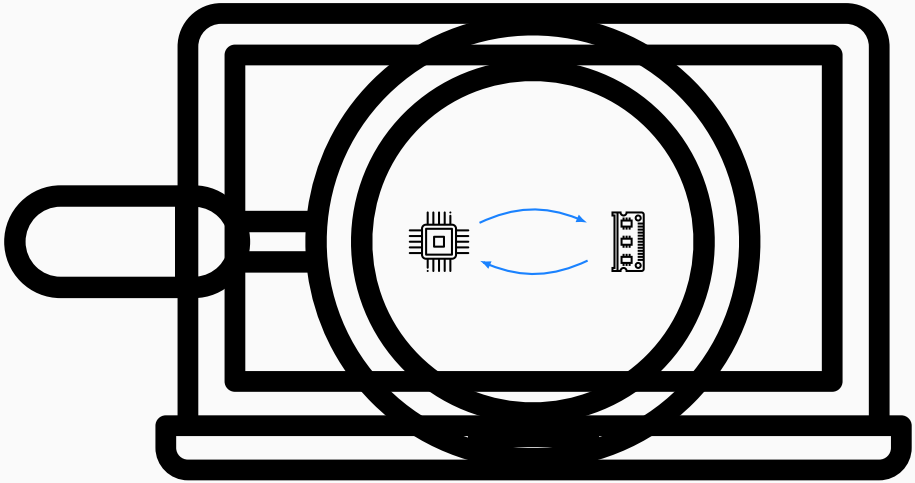
Shared Component - Memory



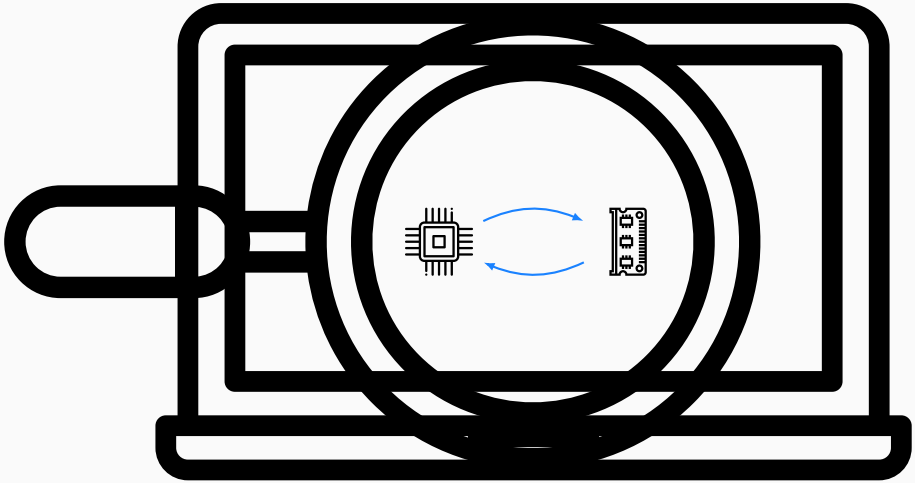
Shared Component - Memory



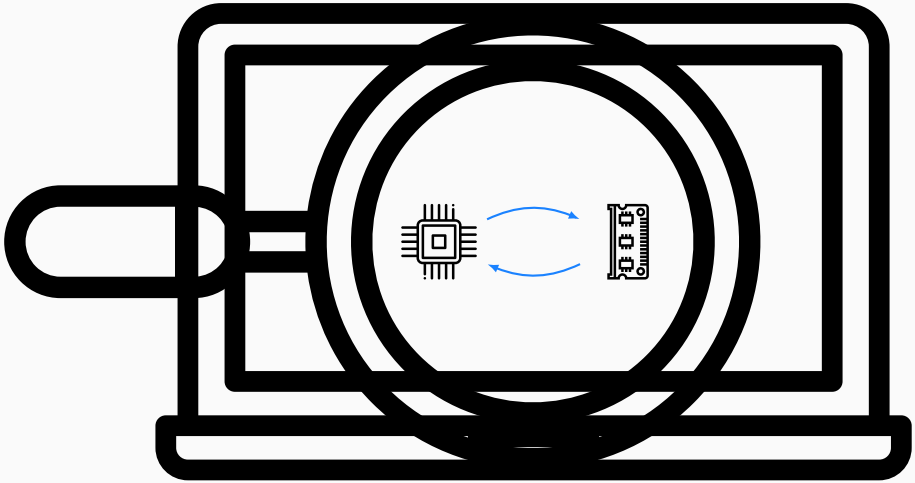
Shared Component - Memory



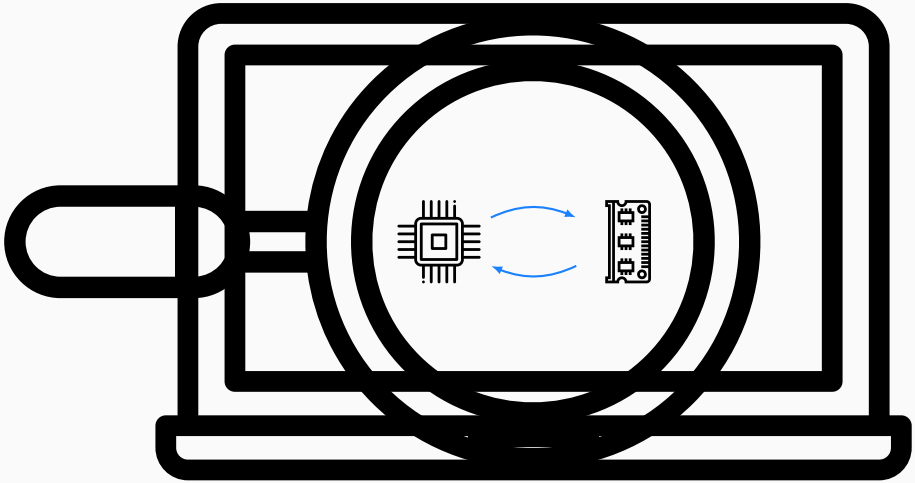
Shared Component - Memory



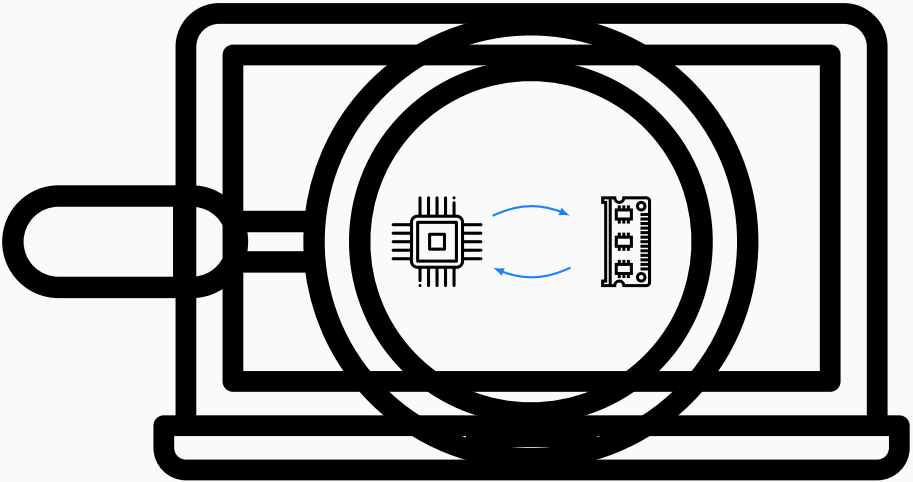
Shared Component - Memory



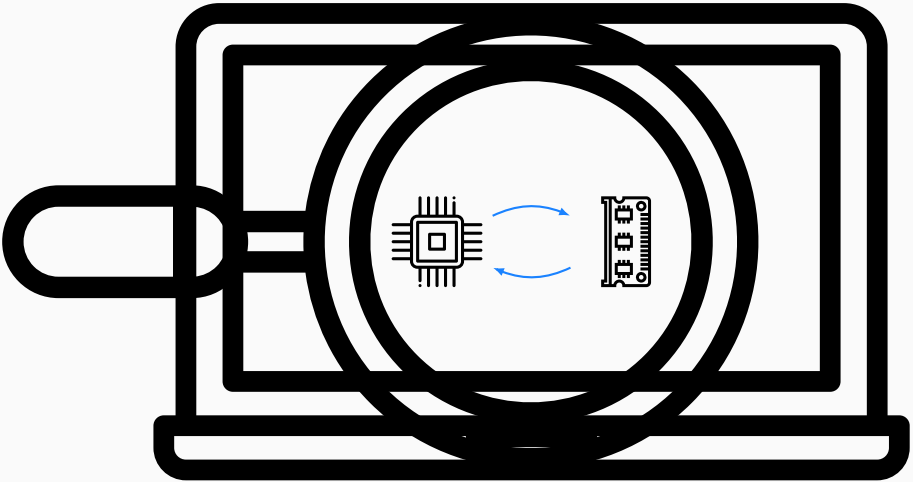
Shared Component - Memory



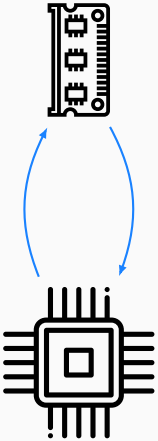
Shared Component - Memory



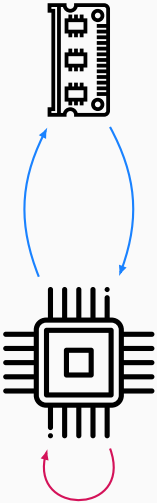
Shared Component - Memory

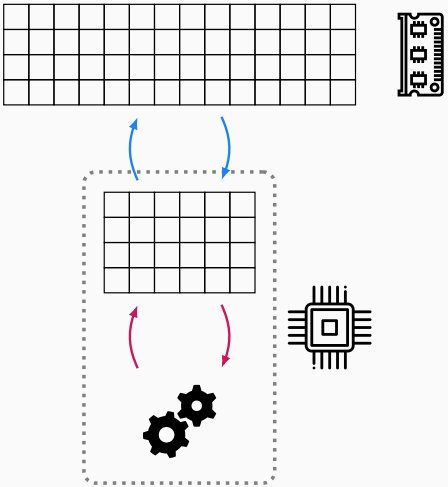


Memory Latency

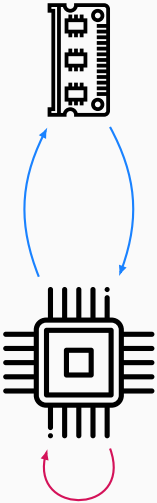


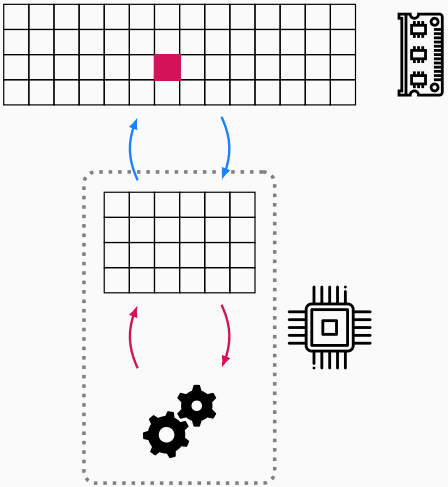
Memory Latency



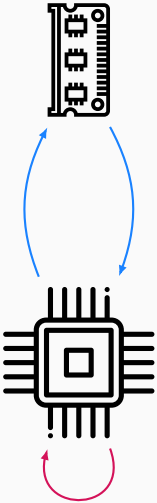


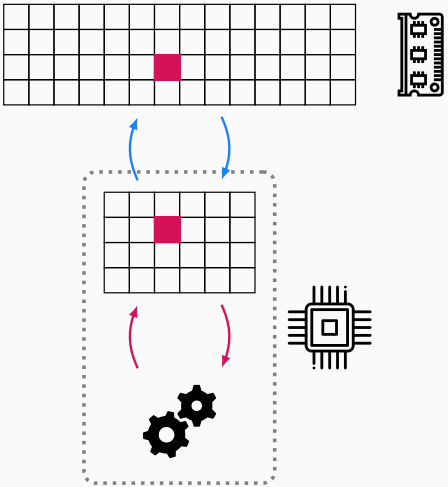
Memory Latency



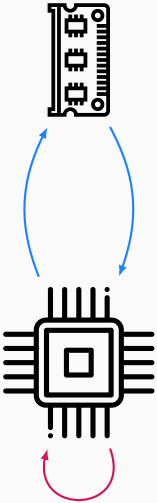


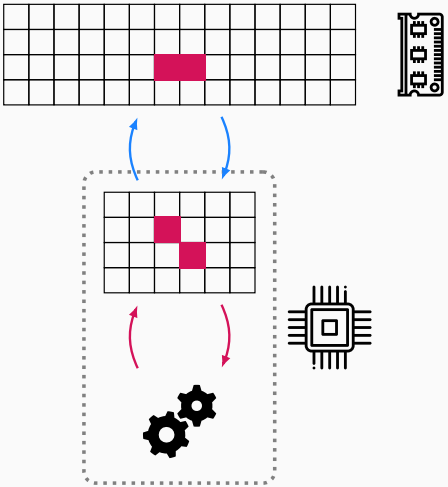
Memory Latency



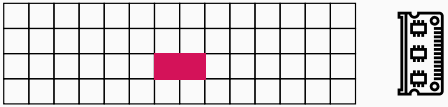


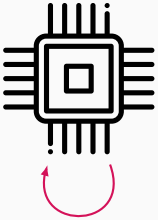
Memory Latency

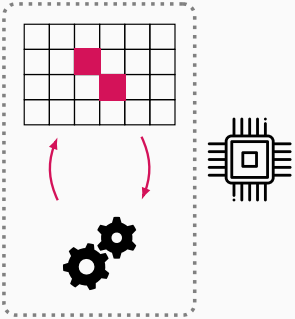




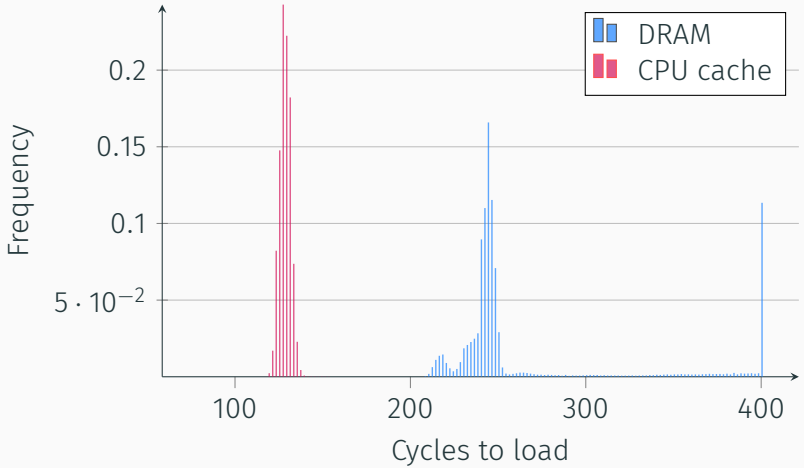
Memory Latency







Memory Latency



Flush+Reload[1]

Goal: spy on data/instructions access

[1] Y. Yarom and K. Falkner. *Flush+Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack.* In USENIX Security Symposium'14.

Flush+Reload[1]

Goal: spy on data/instructions access

Assumption: shared memory (*e.g.* spyware)

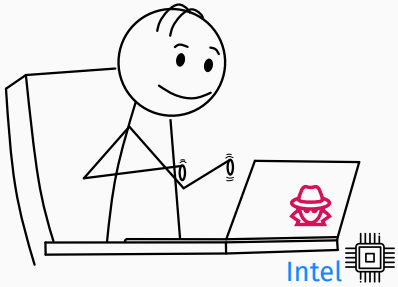
[1] Y. Yarom and K. Falkner. *Flush+Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack.* In USENIX Security Symposium'14.

Flush+Reload[1]

Goal: spy on data/instructions access

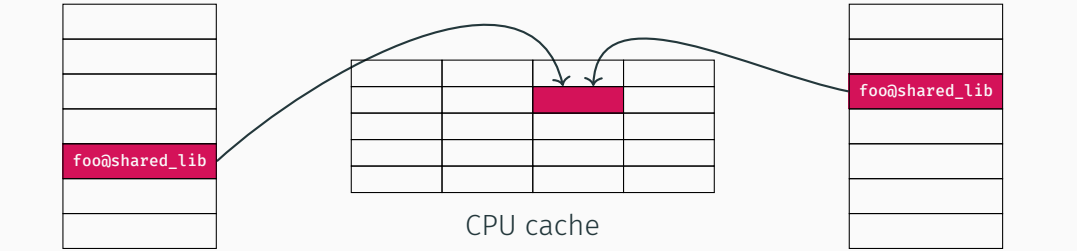
Assumption: shared memory (*e.g.* spyware)

Concept: Abuse cache contention



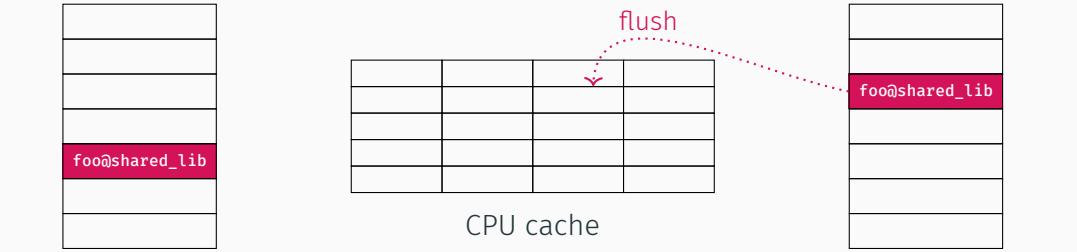
[1] Y. Yarom and K. Falkner. *Flush+Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack.* In USENIX Security Symposium'14.

Flush+Reload



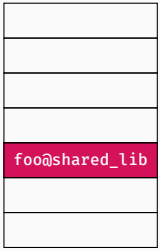
1. Maps the victim's address space

Flush+Reload

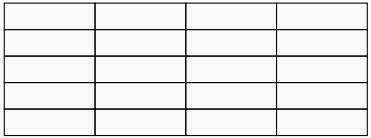


1. Maps the victim's address space
2. Flush the instruction we monitor

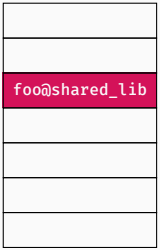
Flush+Reload



Victim address space



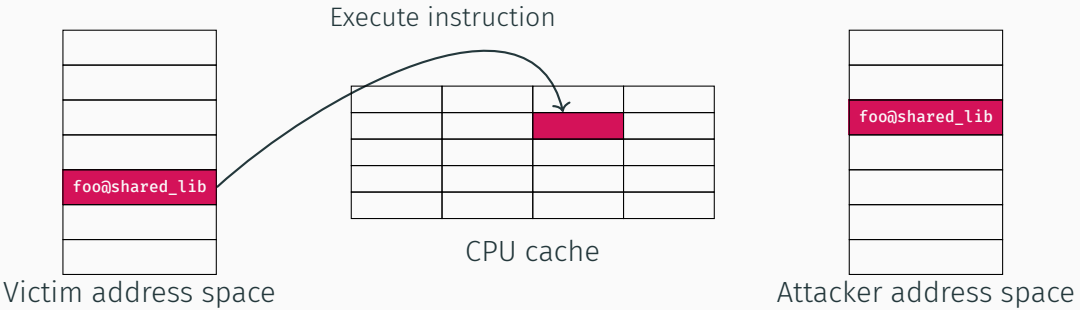
CPU cache



Attacker address space

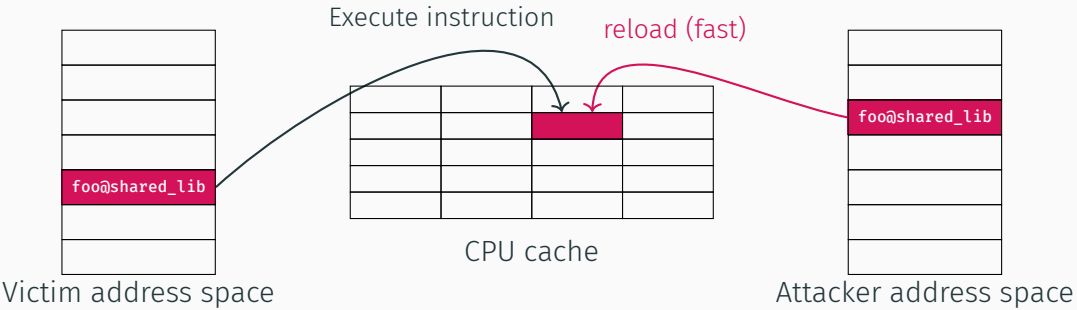
1. Maps the victim's address space
2. Flush the instruction we monitor
3. See how much time it takes to reload

Flush+Reload



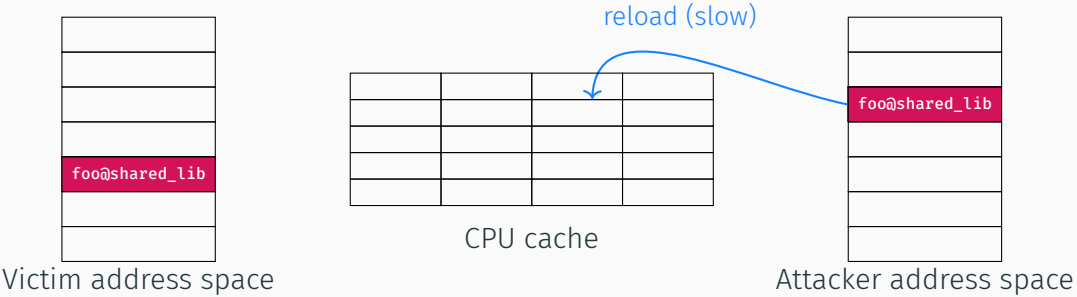
1. Maps the victim's address space
2. Flush the instruction we monitor
3. See how much time it takes to reload

Flush+Reload



1. Maps the victim's address space
2. Flush the instruction we monitor
3. See how much time it takes to reload
   - Fast ⇒ the victim already executed

Flush+Reload



1. Maps the victim's address space
2. Flush the instruction we monitor
3. See how much time it takes to reload
   - Fast ⇒ the victim already executed
   - Slow ⇒ the victim did not

Flush+Reload

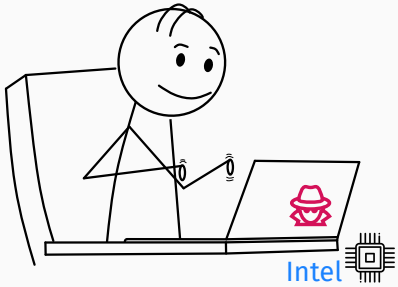
Goal: spy on data/instructions access

Assumption: shared memory (*e.g.* spyware)

Concept: Abuse cache contention

Limitations:

- Spatial resolution
- Temporal resolution
- Hardware optimizations



Flush+Reload

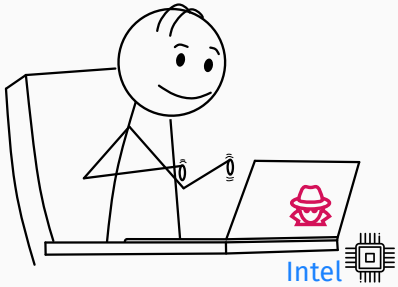
Goal: spy on data/instructions access

Assumption: shared memory (*e.g.* spyware)

Concept: Abuse cache contention

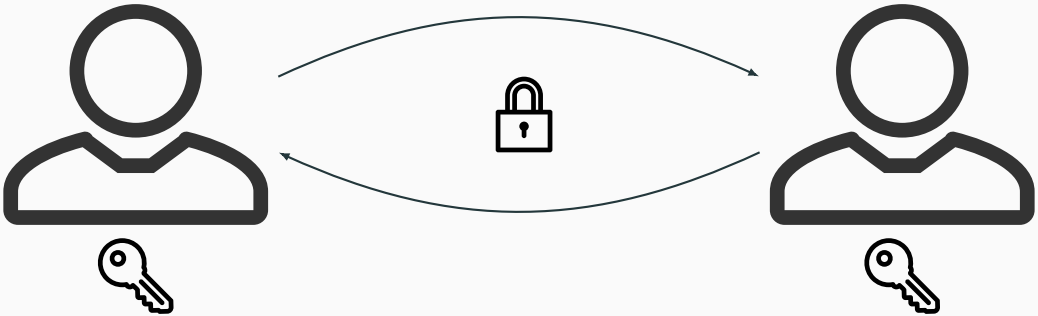
Limitations:

- Spatial resolution
- Temporal resolution
- Hardware optimizations

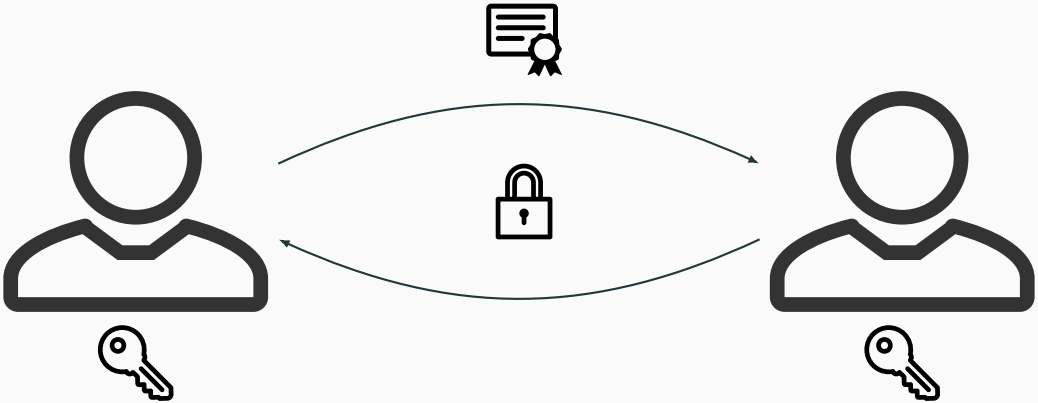


Password Authenticated Key Exchange (PAKE)

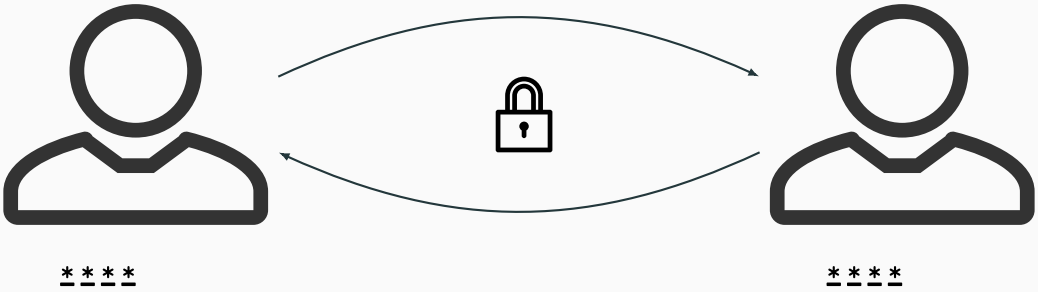
Password Authenticated Key Exchange (PAKE)



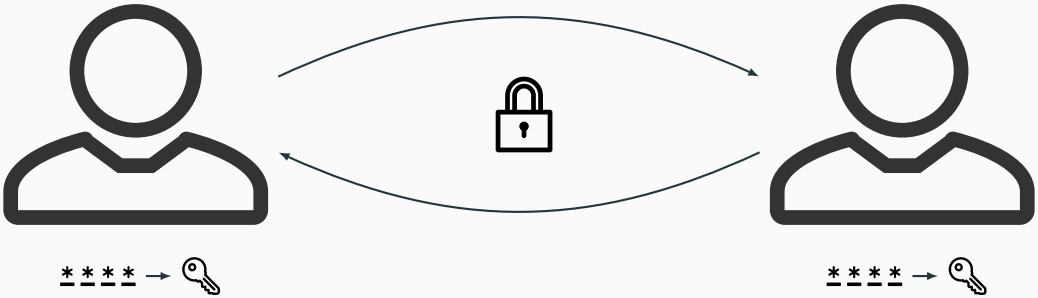
Password Authenticated Key Exchange (PAKE)



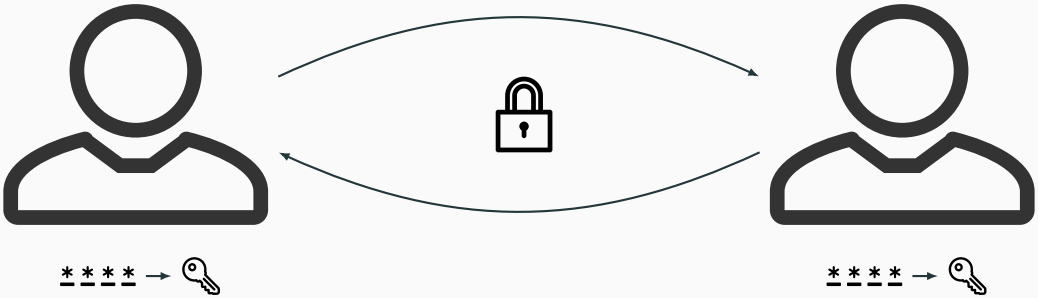
Password Authenticated Key Exchange (PAKE)



Password Authenticated Key Exchange (PAKE)



Password Authenticated Key Exchange (PAKE)



⚠ Needs to resist to (offline) dictionary attacks

Lots of different PAKEs[1]

- Balanced

[1] F. Hao and P.C van Oorschot. *SoK: Password-Authenticated Key Exchange - Theory, Practice, Standardization and Real-World Lessons.* In AsiaCCS'22.

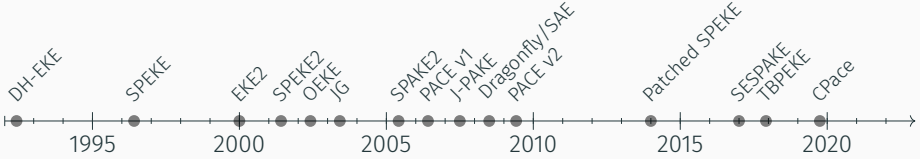
Lots of different PAKEs[1]

- Balanced

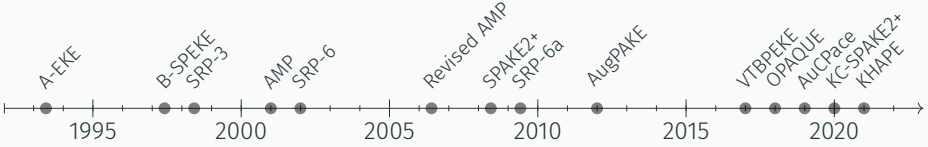
[1] F. Hao and P.C van Oorschot. *SoK: Password-Authenticated Key Exchange - Theory, Practice, Standardization and Real-World Lessons.* In AsiaCCS'22.

Lots of different PAKEs[1]

- Balanced



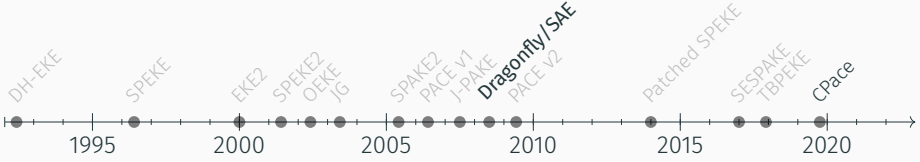
- Augmented



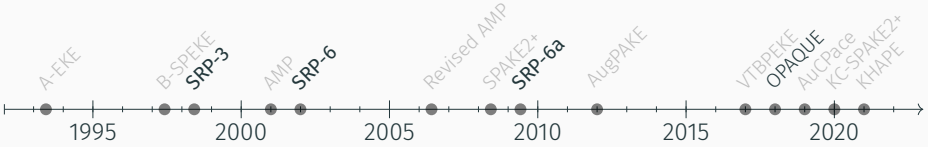
[1] F. Hao and P.C van Oorschot. *SoK: Password-Authenticated Key Exchange - Theory, Practice, Standardization and Real-World Lessons.* In AsiaCCS'22.

Lots of different PAKEs[1]

- Balanced



- Augmented



[1] F. Hao and P.C van Oorschot. *SoK: Password-Authenticated Key Exchange - Theory, Practice, Standardization and Real-World Lessons.* In AsiaCCS'22.

My Contributions

Cryptography in the Wild: The Security of Cryptographic Implementations

I. Assess the security of PAKEs implementations against microarchitectural side-channel attacks
II. Investigate the remanence of side-channel, despite their long history
III. Explore other solutions to provide secure implementations

My Contributions

Peer-reviewed:

S&P'22 User study on constant-time verification tools usage
J. Jancar, M. Fourné, D. De Almeida Braga, M. Sabt, P. Schwabe, G. Barthe, P.A. Fouque, Y. Acar

CCS'21 Password recovery attack against SRP implementation
D. De Almeida Braga, P.A. Fouque, M. Sabt

ACSAC'20 Dragonblood is Still Leaking
D. De Almeida Braga, P.A. Fouque, M. Sabt

TCHES'20 Attack on the SCP10 standard
D. De Almeida Braga, P.A. Fouque, M. Sabt

Under submission:

- Novel attack on Dragonfly, and secure implementation
  D. De Almeida Braga, M. Sabt, P.A. Fouque, N. Kulatova, K. Bhargavan

Ongoing work:

- Follow-up study on constant-time tools usability
- Prefetcher-based side-channel attack

My Contributions

Peer-reviewed:

S&P'22 User study on constant-time verification tools usage
J. Jancar, M. Fourné, D. De Almeida Braga, M. Sabt, P. Schwabe, G. Barthe, P.A. Fouque, Y. Acar

CCS'21 Password recovery attack against SRP implementation
D. De Almeida Braga, P.A. Fouque, M. Sabt

ACSAC'20 Dragonblood is Still Leaking
D. De Almeida Braga, P.A. Fouque, M. Sabt

TCHES'20 Attack on the SCP10 standard
D. De Almeida Braga, P.A. Fouque, M. Sabt

Under submission:

- Novel attack on Dragonfly, and secure implementation
  D. De Almeida Braga, M. Sabt, P.A. Fouque, N. Kulatova, K. Bhargavan

Ongoing work:

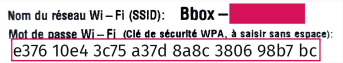
- Follow-up study on constant-time tools usability
- Prefetcher-based side-channel attack

Side Channels in Dragonfly/SAE (WPA3)

Toward Secure Wi-Fi Protocols...

Toward Secure Wi-Fi Protocols...

Offline dictionary attack

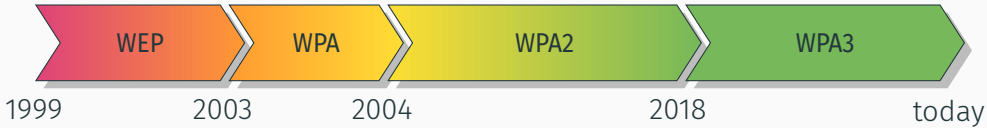


Toward Secure Wi-Fi Protocols...

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

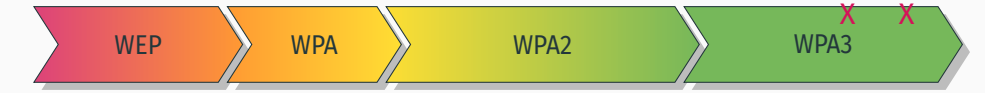
... But Still not Bulletproof

+ 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

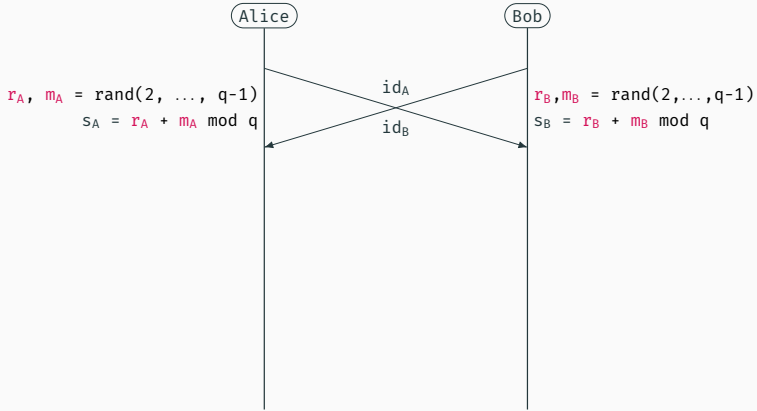


+ More secure
+ Based on a PAKE (Dragonfly)

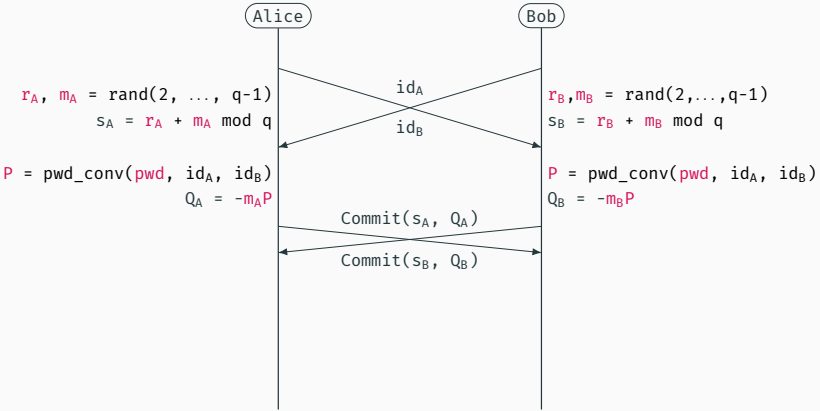
Dragonblood[1] attacks

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

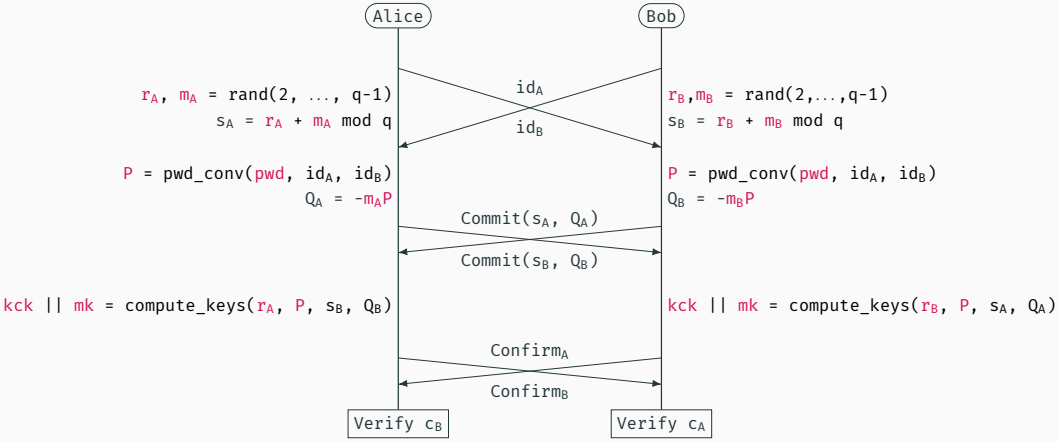
Dragonfly / SAE - A Balanced PAKE



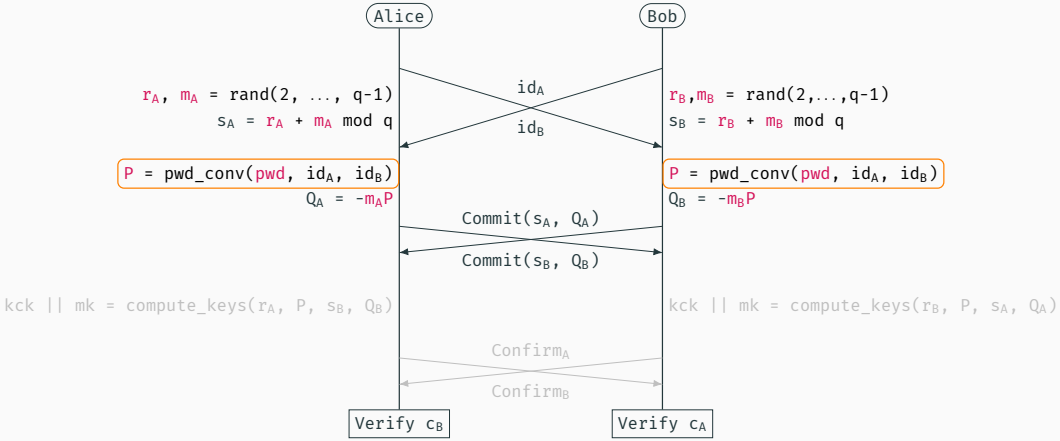
Dragonfly / SAE - A Balanced PAKE



Dragonfly / SAE - A Balanced PAKE



Dragonfly / SAE - A Balanced PAKE



Dragonblood is Still Leaking:
Practical Cache-based Side-Channel in the Wild

Daniel De Almeida Braga, Mohamed Sabt and Pierre-Alain Fouque

Presented at ACSAC 2020
🏆 2^{nd} place at CSAW Applied Research competition 2020

First attack (ACSAC 2020)

A cache-attack that lets us extract

information during the password conversion

leading to an offline dictionary attack.

First attack (ACSAC 2020)

FLUSH+RELOAD[1] and PDA[2]

A cache-attack that lets us extract

information during the password conversion

leading to an offline dictionary attack.

[1] Y. Yarom and K. Falkner. *Flush+Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack.* In USENIX Security Symposium'14.
[2] T. Allan et al. *Amplifying side channels through performance degradation.* In ACSAC'16

First attack (ACSAC 2020)

FLUSH+RELOAD[1] and PDA[2]

Password to point on an Elliptic Curve

A cache-attack that lets us extract information during the password conversion leading to an offline dictionary attack.

First attack (ACSAC 2020)

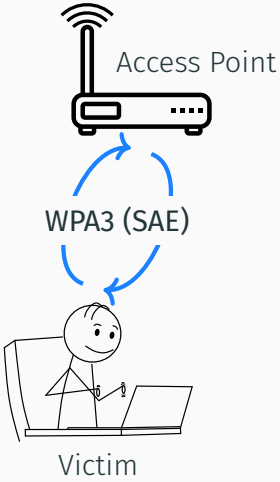
FLUSH+RELOAD[1] and PDA[2]

Password to point on an Elliptic Curve

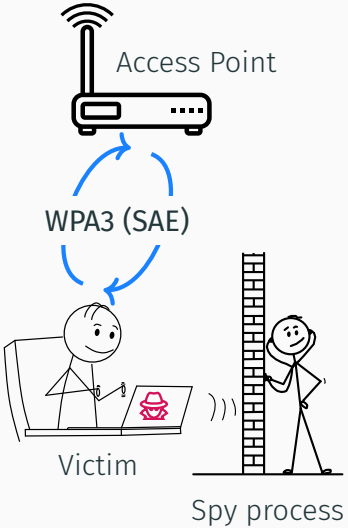
A cache-attack that lets us extract information during the password conversion leading to an offline dictionary attack.

Passive attacker can eliminate wrong passwords from a list

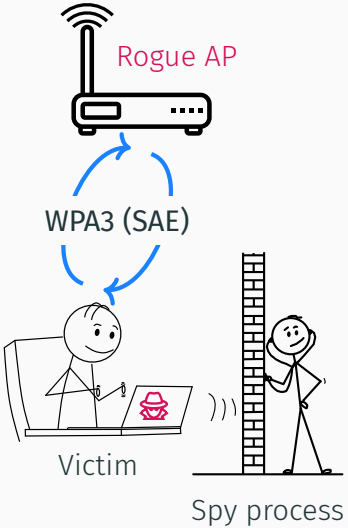
Attack Workflow



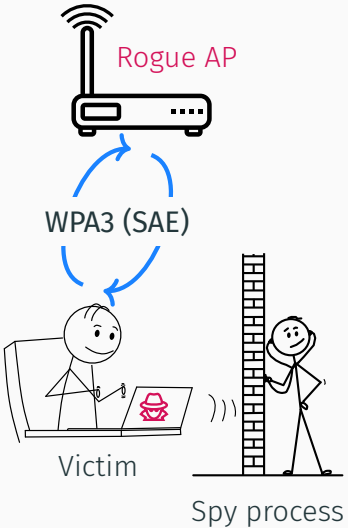
Attack Workflow



Attack Workflow



Attack Workflow



Spying/Data Acquisition

- Implementation specific
- Usually noisy measurement

Comparison metric: Signal to Noise ratio

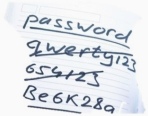
Attack Workflow

Leaked information

Trace parsing

Attack Workflow

Offline Dictionary Attack

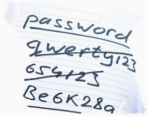


Remaining passwords

Attack Workflow

Offline Dictionary Attack

`H(secret) = 10...`

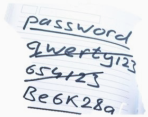


Remaining passwords

Attack Workflow

Offline Dictionary Attack

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

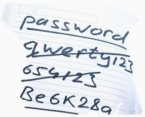


Remaining passwords

Attack Workflow

Offline Dictionary Attack

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

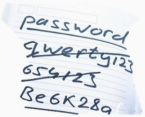


Remaining passwords

Attack Workflow

Offline Dictionary Attack

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

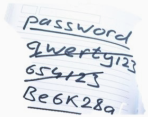


Remaining passwords

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..

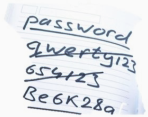


Remaining passwords

Attack Workflow

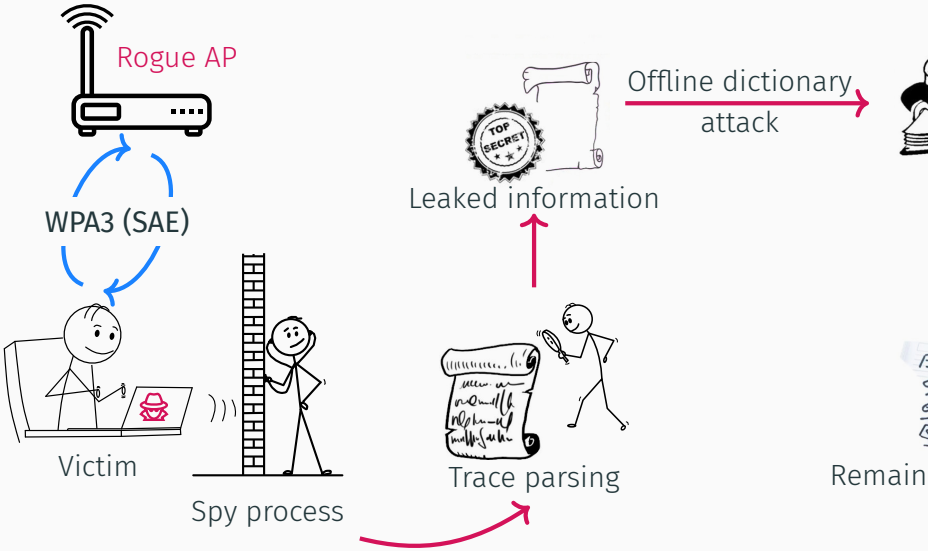
Offline Dictionary Attack

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



SAE - Probabilistic Password Conversion (EC)

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

    seed = Hash(MAC_A, MAC_B, pwd, i= 0)
    x_cand = KDF(seed, label)

    is x_cand a point's coordinate?    (1/2 chance to happen)
```

SAE - Probabilistic Password Conversion (EC)

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

    seed = Hash(MAC_A, MAC_B, pwd, i= 1)
    x_cand = KDF(seed, label)

    is x_cand a point's coordinate?  (1/2 chance to happen)
```

SAE - Probabilistic Password Conversion (EC)

```
def HuntingAndPecking(pwd, MAC_A, MAC_B, EC)

    seed = Hash(MAC_A, MAC_B, pwd, i= 1)
    x_cand = KDF(seed, label)

    is x_cand a point's coordinate?    (1/2 chance to happen)

    x, seed_x = x_cand, seed
```

SAE - Probabilistic Password Conversion (EC)

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

    seed = Hash(MAC_A, MAC_B, pwd, i=40)
    x_cand = KDF(seed, label)

    is x_cand a point's coordinate?    (1/2 chance to happen)

    x, seed_x = x_cand, seed
    pwd = get_random()
```

SAE - Probabilistic Password Conversion (EC)

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

    seed = Hash(MAC_A, MAC_B, pwd, i=40)
    x_cand = KDF(seed, label)

    is x_cand a point's coordinate?  (1/2 chance to happen)
```

SAE - Probabilistic Password Conversion (EC)

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

    seed = Hash(MAC_A, MAC_B, pwd, i=40)
    x_cand = KDF(seed, label)

    is x_cand a point's coordinate?    (1/2 chance to happen)

      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=40)
    x_cand = KDF(seed, label)

    is x_cand a point's coordinate?    (1/2 chance to happen)

      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=40)
    x_cand = KDF(seed, label)

    is x_cand a point's coordinate?    (1/2 chance to happen)

      x, seed_x = x_cand, seed
      pwd = get_random()    ← 🕵: successful conversion

  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=40)
    x_cand = KDF(seed, label)        ←  : new iteration

    is x_cand a point's coordinate?    (1/2 chance to happen)

      x, seed_x = x_cand, seed
      pwd = get_random()     ←  : successful conversion

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

Improves Upon Previous Attack

Data Leaked:

- Number of iterations to convert the password... for a set of public MAC addresses

Amount of Information:

- 2 bits on average

Practical evaluation:

- 10 measurements get reliable information

Improves Upon Previous Attack

Data Leaked:

- Number of iterations to convert the password... for a set of public MAC addresses

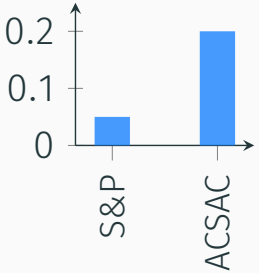
Amount of Information:

- 2 bits on average

Practical evaluation:

- 10 measurements get reliable information

Better signal to noise ratio than the original attack



Impact and Lesson Learned

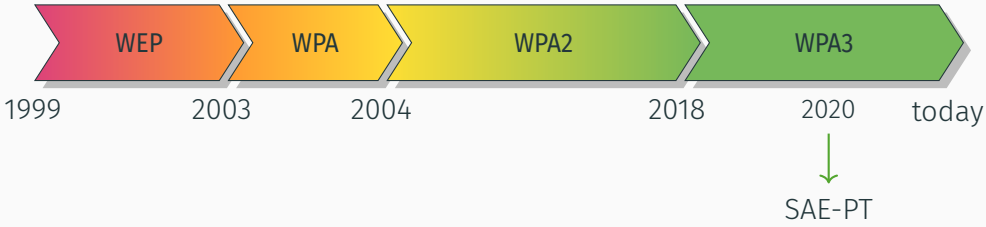
- 2 Practical attacks against iwd and FreeRadius (EAP-pwd)
  - 20 traces needed to recover a password from HaveIBeenPwned
  - ≈0.01€ on AWS instances
- 3 security patches deployed

Material available at `https://gitlab.inria.fr/ddealmei/poc-iwd-acsac2020`

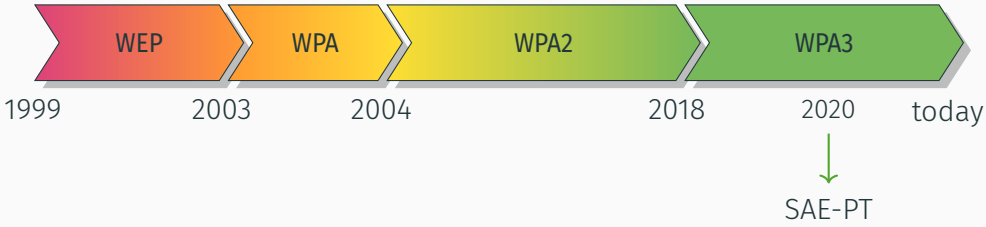
Current official recommendations do not consider microarchitectural attacks

Listen to CFRG members' warnings!

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?

A Novel Side-Channel Attack on Dragonfly Implementation and a Formally Verified Implementation

Daniel De Almeida Braga, Mohamed Sabt, Pierre-Alain Fouque,
Natalia Kulatova, Karthikeyan Bhargavan

Under submission

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

    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

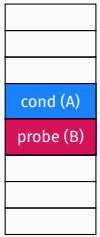
B

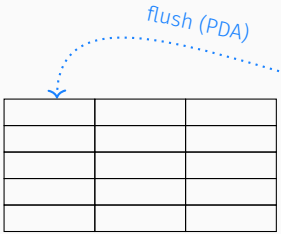
Prefetcher-based Side Channel

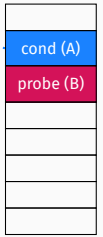
```
def set_compressed_point(x, fmt, ec):
    y = compute_y(x, ec)

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

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

    return P
```

A

B







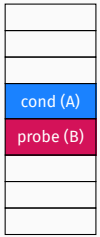
nb hits: 0

Prefetcher-based Side Channel

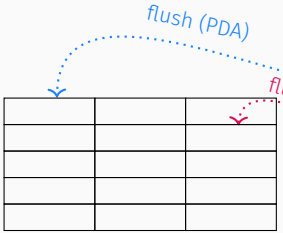
```
def set_compressed_point(x, fmt, ec):
    y = compute_y(x, ec)

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

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

    return P
```

A

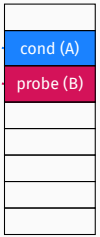
B



Victim



CPU cache



Attacker

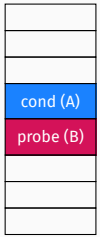
nb hits: 0

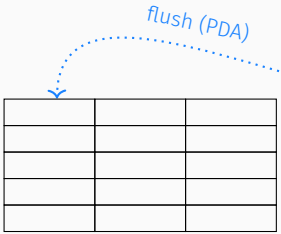
Prefetcher-based Side Channel

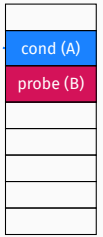
```
def set_compressed_point(x, fmt, ec):
    y = compute_y(x, ec)

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

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

    return P
```

A

B







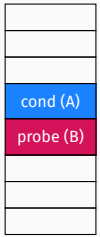
nb hits: 0

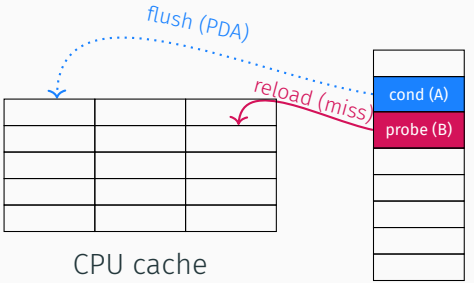
Prefetcher-based Side Channel

```
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





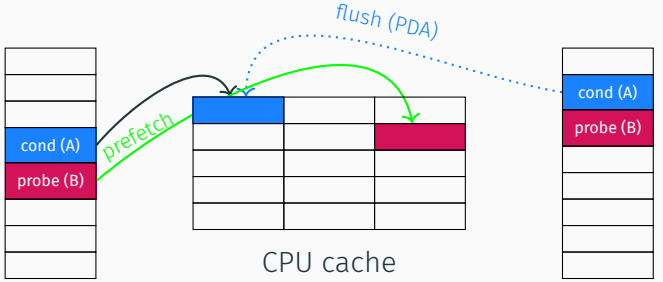
nb hits: 0

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



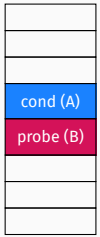
nb hits: 0

Prefetcher-based Side Channel

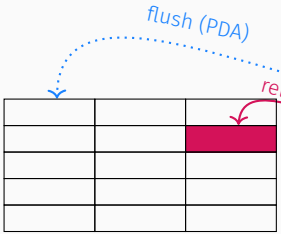
```
def set_compressed_point(x, fmt, ec):
    y = compute_y(x, ec)

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

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

    return P
```

A

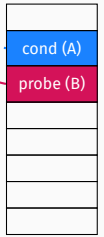
B



Victim



CPU cache



Attacker

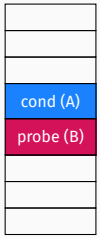
nb hits: 1

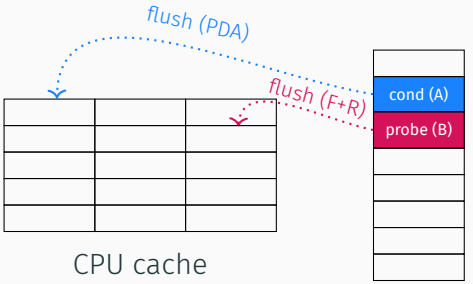
Prefetcher-based Side Channel

```
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





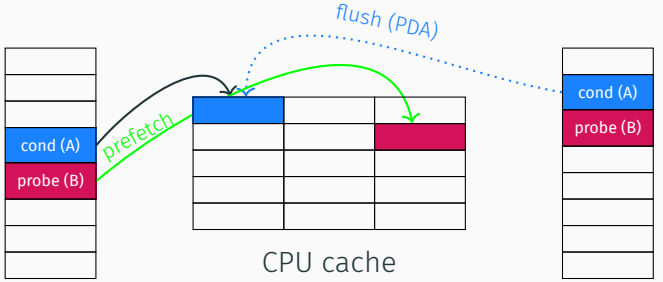
nb hits: 1

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



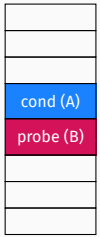
nb hits: 1

Prefetcher-based Side Channel

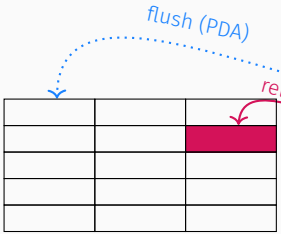
```
def set_compressed_point(x, fmt, ec):
    y = compute_y(x, ec)

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

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

    return P
```

A

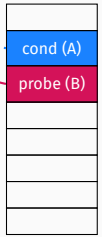
B



Victim



CPU cache



Attacker

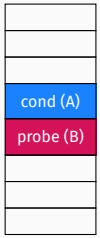
nb hits: 2

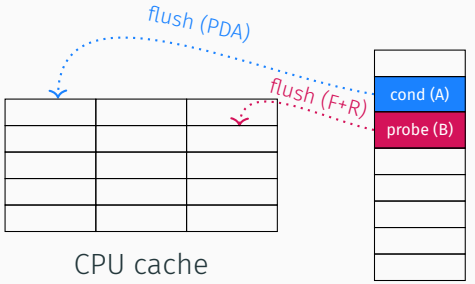
Prefetcher-based Side Channel

```
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





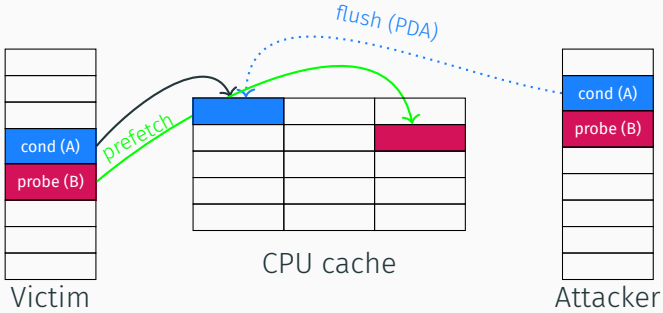
nb hits: 2

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



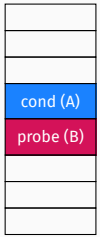
nb hits: 2

Prefetcher-based Side Channel

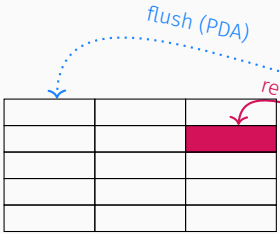
```
def set_compressed_point(x, fmt, ec):
    y = compute_y(x, ec)

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

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

    return P
```

A

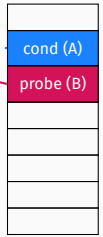
B



Victim



CPU cache



Attacker

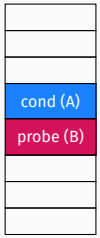
nb hits: 3

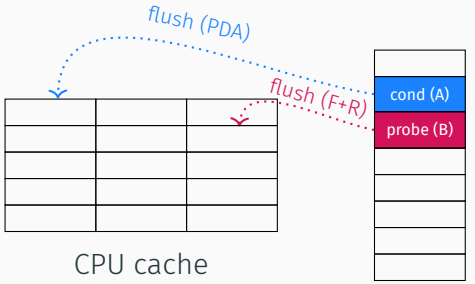
Prefetcher-based Side Channel

```
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

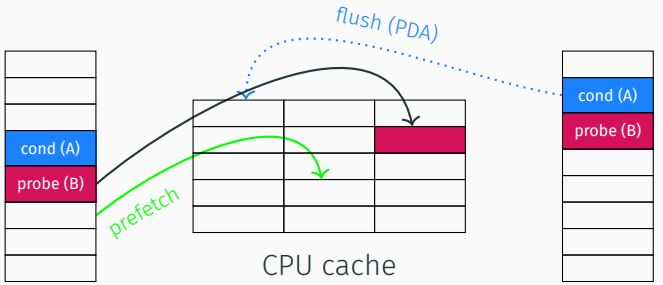




nb hits: 3

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

    return P
```

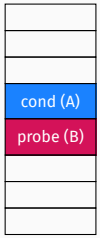


Prefetcher-based Side Channel

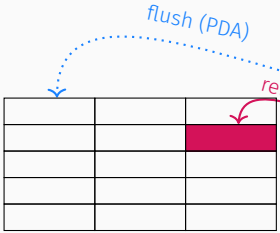
```
def set_compressed_point(x, fmt, ec):
    y = compute_y(x, ec)

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

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

    return P
```

A

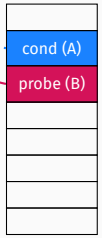
B



Victim



CPU cache



Attacker

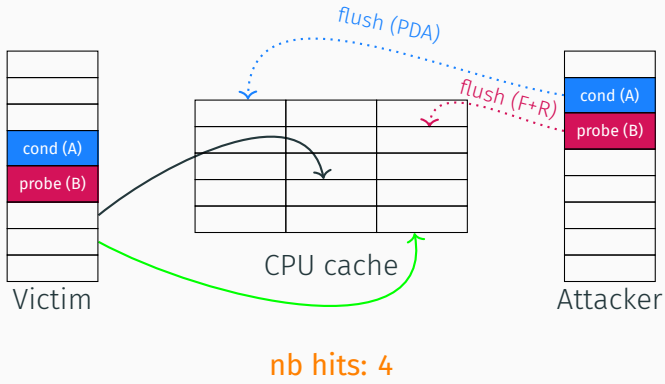
nb hits: 4

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

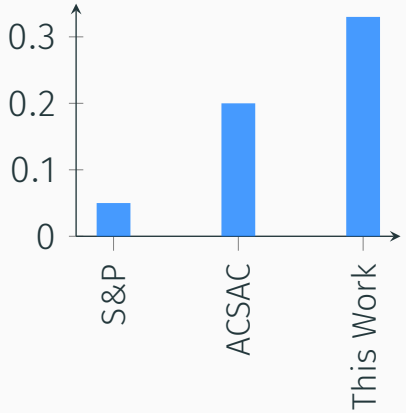
B



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!



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)

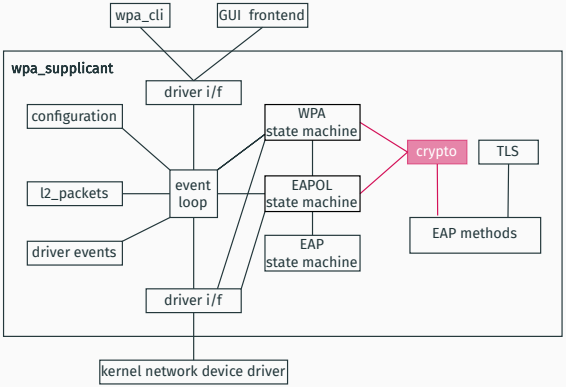
Shall we replace them?

HaCl*: A Formally Verified Cryptographic Library[1]

- Memory-safety
- Functional correctness
- Secret independence

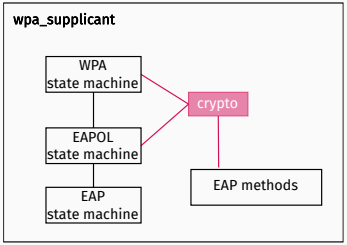
[1] J-K. Zinzindohoué et al. *HACL*: A Verified Modern Cryptographic Library.* In CCS'17

Fixing hostap[1]



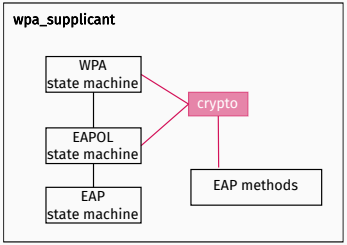
[1] Thank you Alexandre Sanchez for helping with the patch integration

Fixing hostap[1]



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

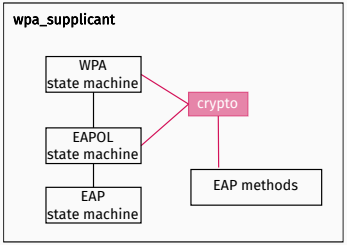
Fixing hostap[1]




```
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!
- 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!
- 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!
- 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`

Constant-time Tools & Usage

“They’re not that hard to mitigate”: What Cryptographic Library Developers Think About Timing Attacks

Jan Jancar[1], Marcel Fourné[2], Daniel De Almeida Braga[3], Mohamed Sabt[3], Peter Schwabe[2], Gilles Barthe[2], Pierre-Alain Fouque[3] and Yasemin Acar[2,4]

Published at S&P 2022

CROCS Centre for Research on Cryptography and Security [1]

MAX PLANCK INSTITUTE FOR SECURITY AND PRIVACY [2]

UMR IRISA [3]

THE GEORGE WASHINGTON UNIVERSITY WASHINGTON, DC [4]

Survey

27 librairies

OpenSSL, BearSSL, libgcrypt, s2n (Amazon), RustCrypto, ...

44 valid responses

Various roles (developers, maintainers, committers, ...)

Survey

1. Participant background

27 librairies

OpenSSL, BearSSL, libgcrypt, s2n (Amazon), RustCrypto, ...

44 valid responses

Various roles (developers, maintainers, committers, ...)

Survey

1. Participant background
↓
2. Library properties & decisions

27 librairies

OpenSSL, BearSSL, libgcrypt, s2n (Amazon), RustCrypto, ...

44 valid responses

Various roles (developers, maintainers, committers, ...)

Survey

1. Participant background
↓
2. Library properties & decisions
↓
3. Tool awareness

27 librairies

OpenSSL, BearSSL, libgcrypt, s2n (Amazon), RustCrypto, ...

44 valid responses

Various roles (developers, maintainers, committers, ...)

Survey

1. Participant background
↓
2. Library properties & decisions
↓
3. Tool awareness
↓
4. Tool use

27 librairies

OpenSSL, BearSSL, libgcrypt, s2n (Amazon), RustCrypto, ...

44 valid responses

Various roles (developers, maintainers, committers, ...)

Survey

1. Participant background
2. Library properties & decisions
3. Tool awareness
4. Tool use
5. Hypothetical tool use

27 librairies

OpenSSL, BearSSL, libgcrypt, s2n (Amazon), RustCrypto, ...

44 valid responses

Various roles (developers, maintainers, committers, ...)

Survey

1. Participant background
2. Library properties & decisions
3. Tool awareness
4. Tool use
5. Hypothetical tool use
6. Miscellaneous

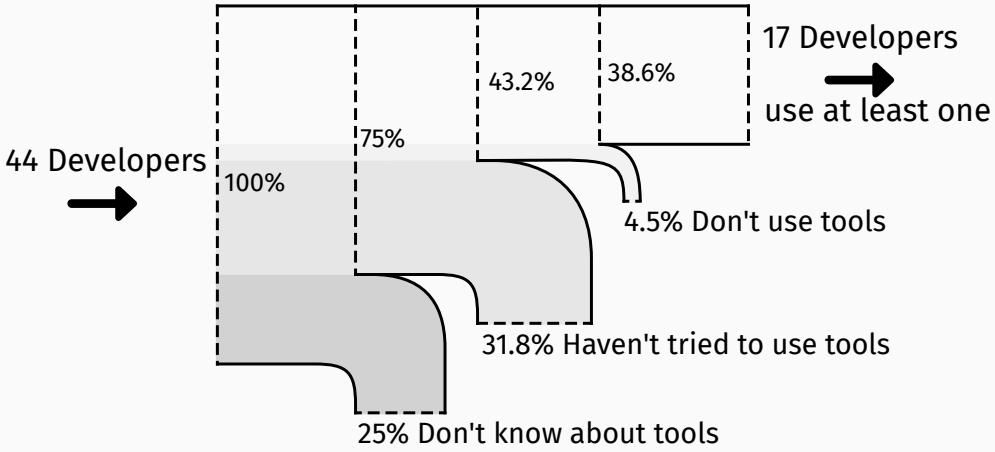
27 librairies

OpenSSL, BearSSL, libgcrypt, s2n (Amazon), RustCrypto, ...

44 valid responses

Various roles (developers, maintainers, committers, ...)

Leaky Pipeline[1]



[1] Figure by Jan Jancar

Developers' Concerns

"Who knows if the toolchain is still ***maintained*** *in a year?"*

Developers' Concerns

*"Who knows if the toolchain is still **maintained** in a year?"*

*"Static analysis tools tend to have a **high engineering overhead**: getting the tool to run, deploying it to CI systems, maintaining the installation over the years"*

Developers' Concerns

"Who knows if the toolchain is still ***maintained*** *in a year?"*

"A thing this survey might be underestimating is also ***the cost of code annotations****: it's not just about having someone annotating the code properly (which already is quite a lot of effort) but [...] they add a maintenance burden for the project."*

"Static analysis tools tend to have a ***high engineering overhead****: getting the tool to run, deploying it to CI systems, maintaining the installation over the years"*

Developers' Concerns

"Who knows if the toolchain is still ***maintained*** *in a year?"*

"A thing this survey might be underestimating is also ***the cost of code annotations****: it's not just about having someone annotating the code properly (which already is quite a lot of effort) but [...] they add a maintenance burden for the project."*

"Static analysis tools tend to have a ***high engineering overhead****: getting the tool to run, deploying it to CI systems, maintaining the installation over the years"*

"[...] so far it seems formal analysis tools (at least where we've tried to apply it to correctness) are ***not really usable by mere mortals yet****."*

Concluding Notes

Practical Impact

- **5** practical Proof of Concept attacks

[1] J. Wichelmann et al. *Microwalk-CI: Practical Side-Channel Analysis for JavaScript Applications.* CCS'22

Practical Impact

- **5** practical Proof of Concept attacks
- **13** Security patches
  - Big number libraries of **4** programming langages (C, Ruby, JS, Erlang)
  - Softwares and libraries deployed on billions of devices (OpenSSL, hostap, FreeRadius, Apple HomeKit, ...)

[1] J. Wichelmann et al. *Microwalk-CI: Practical Side-Channel Analysis for JavaScript Applications.* CCS'22

Practical Impact

- **5** practical Proof of Concept attacks
- **13** Security patches
  - Big number libraries of **4** programming langages (C, Ruby, JS, Erlang)
  - Softwares and libraries deployed on billions of devices (OpenSSL, hostap, FreeRadius, Apple HomeKit, ...)
- Shaping new constant-time tools development[1]

[1] J. Wichelmann et al. *Microwalk-CI: Practical Side-Channel Analysis for JavaScript Applications.* CCS'22

Practical Impact

- **5** practical Proof of Concept attacks
- **13** Security patches
  - Big number libraries of **4** programming langages (C, Ruby, JS, Erlang)
  - Softwares and libraries deployed on billions of devices (OpenSSL, hostap, FreeRadius, Apple HomeKit, ...)
- Shaping new constant-time tools development[1]
- New microarchitectural attack gadget (in progress)

[1] J. Wichelmann et al. *Microwalk-CI: Practical Side-Channel Analysis for JavaScript Applications.* CCS'22

Practical Impact

- **5** practical Proof of Concept attacks
- **13** Security patches
  - Big number libraries of **4** programming langages (C, Ruby, JS, Erlang)
  - Softwares and libraries deployed on billions of devices (OpenSSL, hostap, FreeRadius, Apple HomeKit, ...)
- Shaping new constant-time tools development[1]
- New microarchitectural attack gadget (in progress)
- Amendment to GlobalPlatform SCP10 standard

[1] J. Wichelmann et al. *Microwalk-CI: Practical Side-Channel Analysis for JavaScript Applications.* CCS'22

Limitations and Future Work

Characterize the new leakage gadget

- How accurate can we get?
- Does it work on other architectures?

Limitations and Future Work

Characterize the new leakage gadget

- How accurate can we get?
- Does it work on other architectures?

Other microarchitectural components

- Lots of candidates
- Evolving architectures

Limitations and Future Work

Characterize the new leakage gadget

- How accurate can we get?
- Does it work on other architectures?

Other microarchitectural components

- Lots of candidates
- Evolving architectures

Secure Standardized PAKEs

- OPAQUE and CPace
- Verifying early implementations
- Generating formally verified implementations

Limitations and Future Work

Characterize the new leakage gadget

- How accurate can we get?
- Does it work on other architectures?

Secure Standardized PAKEs

- OPAQUE and CPace
- Verifying early implementations
- Generating formally verified implementations

Other microarchitectural components

- Lots of candidates
- Evolving architectures

Academic - Real-World Gap

- What does make a tool usable?
- How to enforce formal verification for constant-time programming?
- Would another tool change anything?

Peer-reviewed:

S&P'22 "They're not that hard to mitigate": What Cryptographic Library Developers Think About Timing Attacks
J. Jancar, M. Fourné, D. De Almeida Braga, M. Sabt, P. Schwabe, G. Barthe, P.A. Fouque, Y. Acar

CCS'21 PARASITE: PAssword Recovery Attack against Srp Implementations in ThE wild
D. De Almeida Braga, P.A. Fouque, M. Sabt

ACSAC'20 Dragonblood is Still Leaking: Practical Cache-based Side-Channel in the Wild
D. De Almeida Braga, P.A. Fouque, M. Sabt

TCHES'20 The Long and Winding Path to Secure Implementation of GlobalPlatform SCP10
D. De Almeida Braga, P.A. Fouque, M. Sabt

Under submission:

- Novel attack on Dragonfly, and secure implementation
  D. De Almeida Braga, M. Sabt, P.A. Fouque, N. Kulatova, K. Bhargavan

Ongoing work:

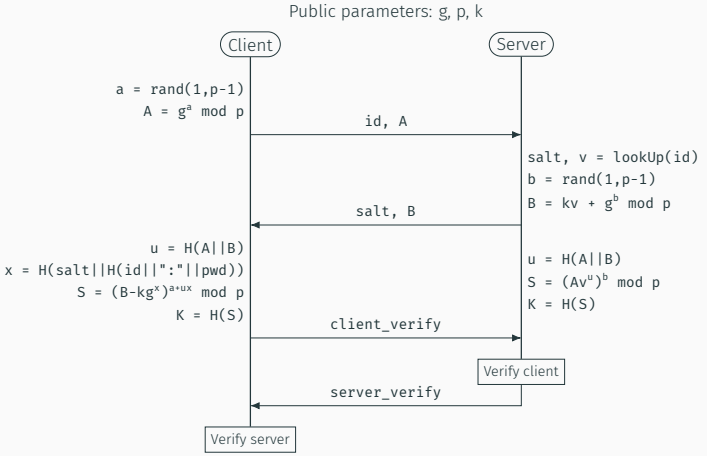
- Follow-up study on constant-time tools usability
- Prefetcher-based side-channel attack

PARASITE: PAssword Recovery Attack against Srp Implementations in ThE wild

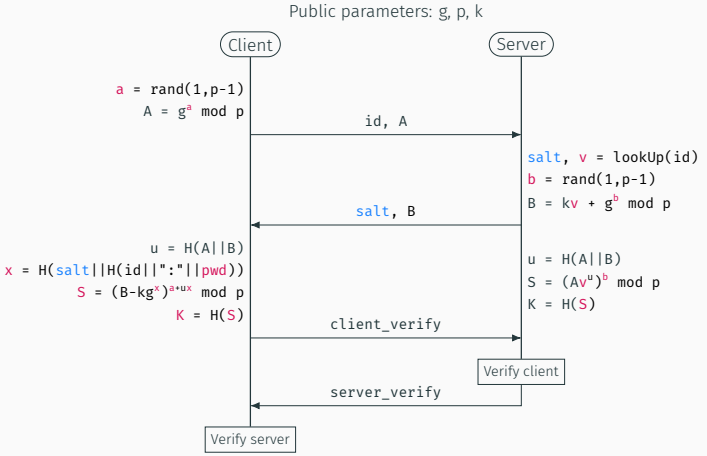
Daniel De Almeida Braga, Mohamed Sabt and Pierre-Alain Fouque

Presented at CCS 2021

SRP - A Legacy Asymmetric PAKE



SRP - A Legacy Asymmetric PAKE



Exploiting the Leakage

Attacker's Goal: Recover the password

Target: OpenSSL's modular exponentiation

Exploiting the Leakage

Attacker's Goal: Recover the password

Target: OpenSSL's modular exponentiation

Challenge: operations are *very* fast, hence tricky to reliably observe for an attacker

Solution: Identify bit patterns in the exponent, based on arithmetic overflows.

Impact and Lesson

Impact: Large impact analysis on open source projects

- 6 projects
- 10 packages/libraries
- 6 programming languages

Impact and Lesson

Impact: Large impact analysis on open source projects

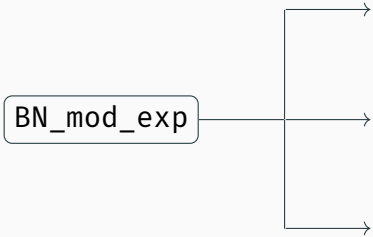
- 6 projects
- 10 packages/libraries
- 6 programming languages

Who knows how many closed-source projects?

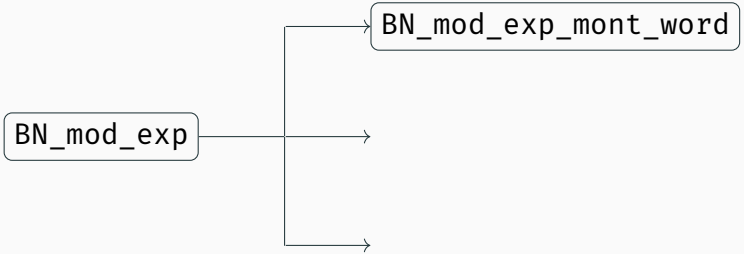
Modular exponentiation in OpenSSL

`BN_mod_exp`

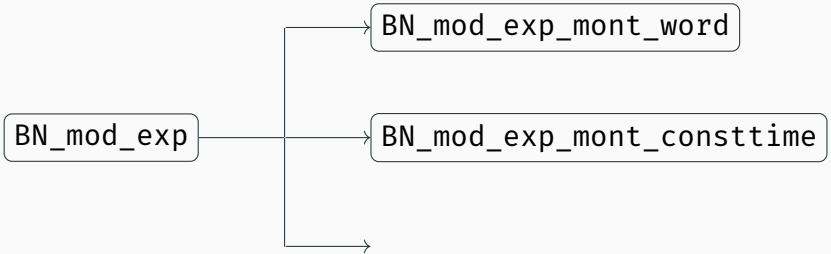
Modular exponentiation in OpenSSL



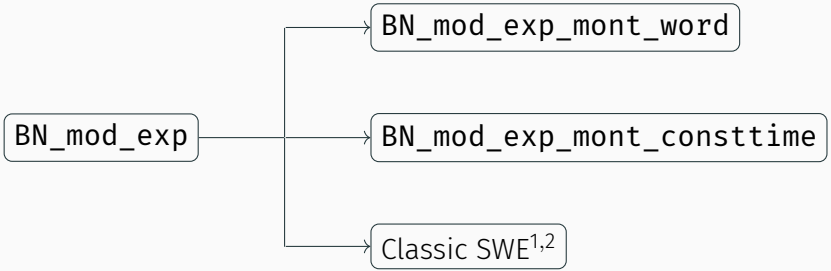
Modular exponentiation in OpenSSL



Modular exponentiation in OpenSSL

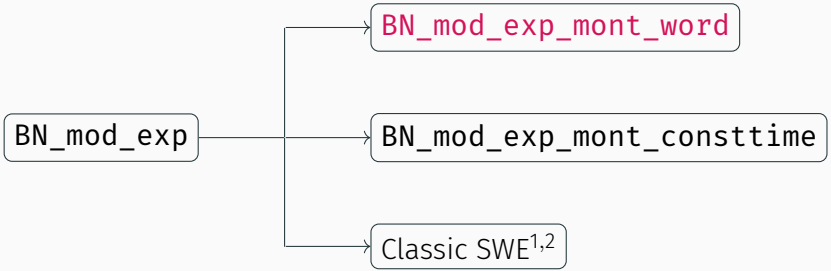


Modular exponentiation in OpenSSL



[1] C. Percival. *Cache missing for fun and profit.* 2005
[2] C. Peraida Garcia et al. *Certified Side Channels.* In USENIX Security. 2020

Modular exponentiation in OpenSSL



[1] C. Percival. *Cache missing for fun and profit.* 2005
[2] C. Peraida Garcia et al. *Certified Side Channels.* In USENIX Security. 2020

Optimized Square-and-Multiply

bin(x) = 1 1 0 1 0 ...

res= $g^x \bmod p$

w processor word (*e.g.* 64 bits)

```
def BN_mod_exp_mont_word(g, x, p):
  w = g # uint64_t
  res = BN_to_mont_word(w) # bignum
  for b in range(bitlen-2, 0, -1):
    next_w = w × w
    if (next_w / w) != w:
      res = BN_mod_mul(res, w, p)
      next_w = 1
    w = next_w;
    res = BN_mod_sqr(res, p)
    if BN_is_bit_set(x, b):
      next_w = w × g
      if (next_w / g) != w:
        res = BN_mod_mul(res, w, p)
        next_w = g
      w = next_w
```

Optimized Square-and-Multiply

bin(x) = 1 1 0 1 0 ...

res= $g^x \bmod p$

w processor word (*e.g.* 64 bits)

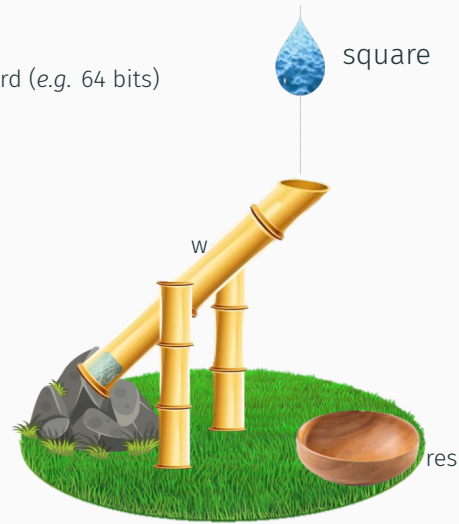

```
def BN_mod_exp_mont_word(g, x, p):
  w = g # uint64_t
  res = BN_to_mont_word(w) # bignum
→ for b in range(bitlen-2, 0, -1):
    next_w = w × w
    if (next_w / w) != w:
      res = BN_mod_mul(res, w, p)
      next_w = 1
    w = next_w;
    res = BN_mod_sqr(res, p)
    if BN_is_bit_set(x, b):
      next_w = w × g
      if (next_w / g) != w:
        res = BN_mod_mul(res, w, p)
        next_w = g
      w = next_w
```

Optimized Square-and-Multiply

bin(x) = 1 1 0 1 0 ...

res= $g^x \bmod p$

w processor word (*e.g.* 64 bits)




```
def BN_mod_exp_mont_word(g, x, p):
  w = g # uint64_t
  res = BN_to_mont_word(w) # bignum
  for b in range(bitlen-2, 0, -1):
→   next_w = w × w
    if (next_w / w) != w:
      res = BN_mod_mul(res, w, p)
      next_w = 1
    w = next_w;
    res = BN_mod_sqr(res, p)
    if BN_is_bit_set(x, b):
      next_w = w × g
      if (next_w / g) != w:
        res = BN_mod_mul(res, w, p)
        next_w = g
      w = next_w
```

Optimized Square-and-Multiply

bin(x) = 1 1 0 1 0 ...

res= $g^x \bmod p$

w processor word (*e.g.* 64 bits)

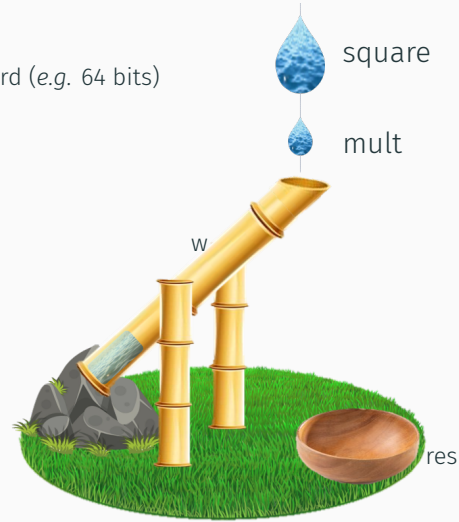

```
def BN_mod_exp_mont_word(g, x, p):
  w = g # uint64_t
  res = BN_to_mont_word(w) # bignum
  for b in range(bitlen-2, 0, -1):
    next_w = w × w
    if (next_w / w) != w:
      res = BN_mod_mul(res, w, p)
      next_w = 1
    w = next_w;
    res = BN_mod_sqr(res, p)
    if BN_is_bit_set(x, b):
→     next_w = w × g
      if (next_w / g) != w:
        res = BN_mod_mul(res, w, p)
        next_w = g
      w = next_w
```

Optimized Square-and-Multiply

bin(x) = 1 1 0 1 0 ...

res= $g^x \bmod p$

w processor word (*e.g.* 64 bits)



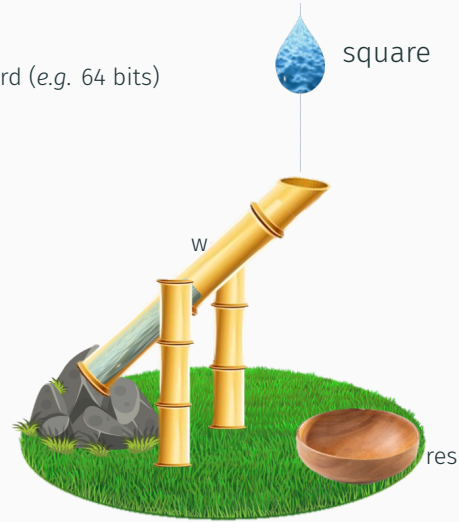

```
def BN_mod_exp_mont_word(g, x, p):
  w = g # uint64_t
  res = BN_to_mont_word(w) # bignum
  for b in range(bitlen-2, 0, -1):
→   next_w = w × w
    if (next_w / w) != w:
      res = BN_mod_mul(res, w, p)
      next_w = 1
    w = next_w;
    res = BN_mod_sqr(res, p)
    if BN_is_bit_set(x, b):
→     next_w = w × g
      if (next_w / g) != w:
        res = BN_mod_mul(res, w, p)
        next_w = g
      w = next_w
```

Optimized Square-and-Multiply

bin(x) = 1 1 0 1 0 ...

res= $g^x \bmod p$

w processor word (*e.g.* 64 bits)



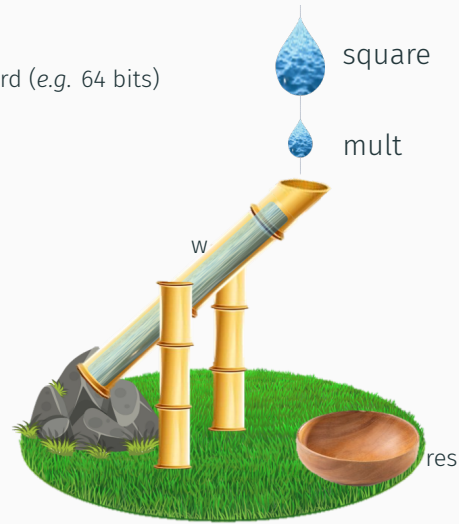

```
def BN_mod_exp_mont_word(g, x, p):
  w = g # uint64_t
  res = BN_to_mont_word(w) # bignum
  for b in range(bitlen-2, 0, -1):
→   next_w = w × w
    if (next_w / w) != w:
      res = BN_mod_mul(res, w, p)
      next_w = 1
    w = next_w;
    res = BN_mod_sqr(res, p)
    if BN_is_bit_set(x, b):
      next_w = w × g
      if (next_w / g) != w:
        res = BN_mod_mul(res, w, p)
        next_w = g
      w = next_w
```

Optimized Square-and-Multiply

bin(x) = 1 1 0 1 0 ...

res= $g^x \bmod p$

w processor word (*e.g.* 64 bits)




```
def BN_mod_exp_mont_word(g, x, p):
  w = g # uint64_t
  res = BN_to_mont_word(w) # bignum
  for b in range(bitlen-2, 0, -1):
→   next_w = w × w
    if (next_w / w) != w:
      res = BN_mod_mul(res, w, p)
      next_w = 1
    w = next_w;
    res = BN_mod_sqr(res, p)
    if BN_is_bit_set(x, b):
→     next_w = w × g
      if (next_w / g) != w:
        res = BN_mod_mul(res, w, p)
        next_w = g
      w = next_w
```

Optimized Square-and-Multiply

bin(x) = 1 1 0 1 0 ...

res= $g^x \bmod p$

w processor word (*e.g.* 64 bits)


```
def BN_mod_exp_mont_word(g, x, p):
  w = g # uint64_t
  res = BN_to_mont_word(w) # bignum
  for b in range(bitlen-2, 0, -1):
→   next_w = w × w
    if (next_w / w) != w:
      res = BN_mod_mul(res, w, p)
      next_w = 1
    w = next_w;
    res = BN_mod_sqr(res, p)
    if BN_is_bit_set(x, b):
      next_w = w × g
      if (next_w / g) != w:
        res = BN_mod_mul(res, w, p)
        next_w = g
      w = next_w
```

Optimized Square-and-Multiply

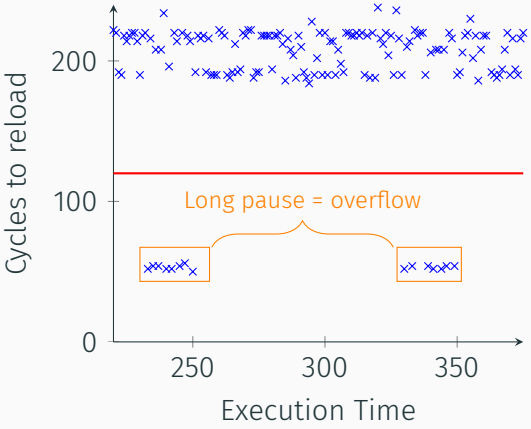
bin(x) = 1 1 0 1 0 ...

res= $g^x \bmod p$

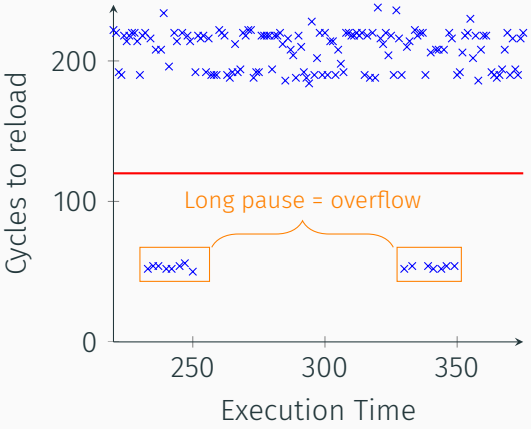
w processor word (*e.g.* 64 bits)


```
def BN_mod_exp_mont_word(g, x, p):
  w = g # uint64_t
  res = BN_to_mont_word(w) # bignum
  for b in range(bitlen-2, 0, -1):
    next_w = w × w
    if (next_w / w) != w:
  →   res = BN_mod_mul(res, w, p)
  →   next_w = 1
    w = next_w;
    res = BN_mod_sqr(res, p)
    if BN_is_bit_set(x, b):
      next_w = w × g
      if (next_w / g) != w:
        res = BN_mod_mul(res, w, p)
        next_w = g
      w = next_w
```

Trace Interpretation



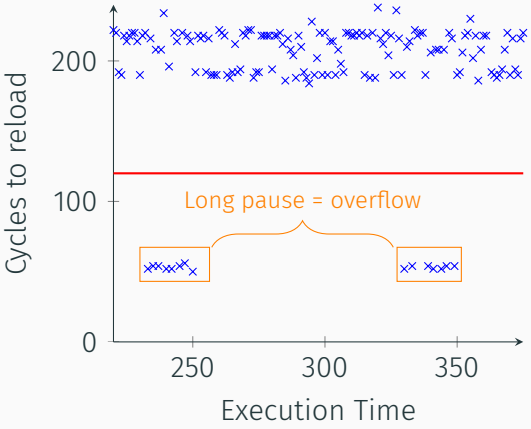
Trace Interpretation



Rules ($b \in \{0,1\}$):

- `bbbb` $\Rightarrow$ $111b$
- `bbbbb` $\Rightarrow$ $yyyyb$, $yyyy \in \{110b, 10bb, 0111\}$
- `bb...b` $\Rightarrow$ $0 \ldots 0yyyyb$

Trace Interpretation

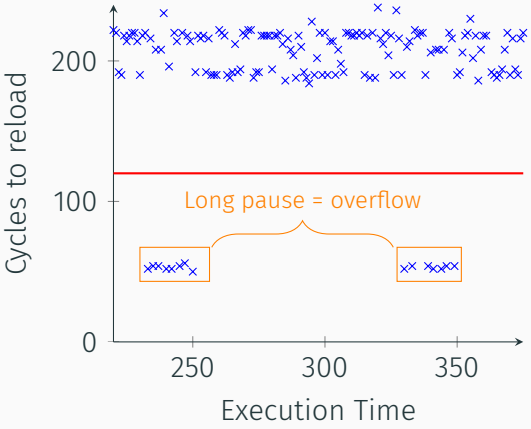


Rules ($b \in \{0,1\}$):

- `bbbb` $\Rightarrow$ $111b$
- `bbbbb` $\Rightarrow$ $yyyyb$, $yyyy \in \{110b, 10bb, 0111\}$
- `bb...b` $\Rightarrow$ $0\ldots0yyyyb$

`bbbb bbbbb bbbbbb bbbbb bbbbb bbbb`

Trace Interpretation

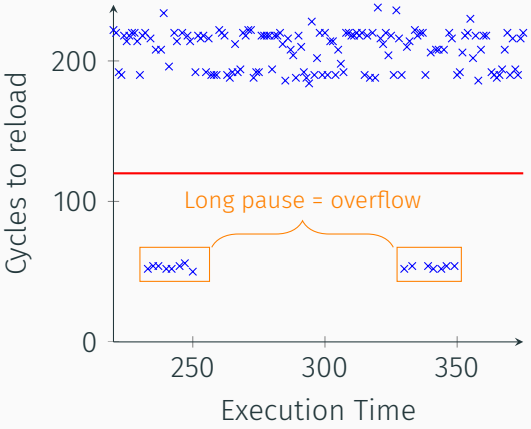


Rules ($b \in \{0,1\}$):

- `bbbb` $\Rightarrow$ $111b$
- `bbbbb` $\Rightarrow$ $yyyyb$, $yyyy \in \{110b, 10bb, 0111\}$
- `bb...b` $\Rightarrow$ $0\ldots0yyyyb$

bbbb	bbbbb	bbbbbb	bbbbb	bbbbb	bbbb
4	5	6	5	5	4

Trace Interpretation

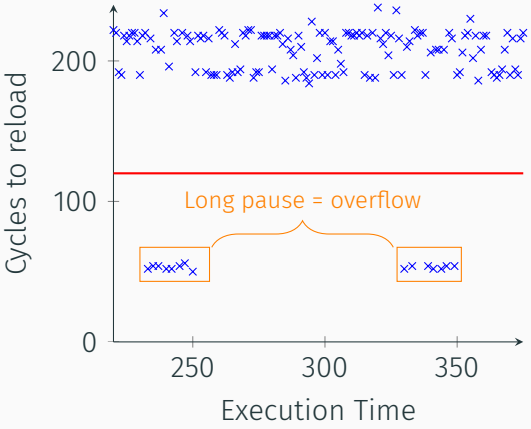


Rules ($b \in \{0,1\}$):

- bbbb $\Rightarrow$ $111b$
- bbbbb $\Rightarrow$ $yyyyb$, $yyyy \in \{110b, 10bb, 0111\}$
- bb...b $\Rightarrow$ $0\ldots0yyyyb$

```
bbbb  bbbbb  bbbbbb  bbbbb  bbbbb  bbbb
  4     5       6      5      5      4
  ⇓
111b
```

Trace Interpretation

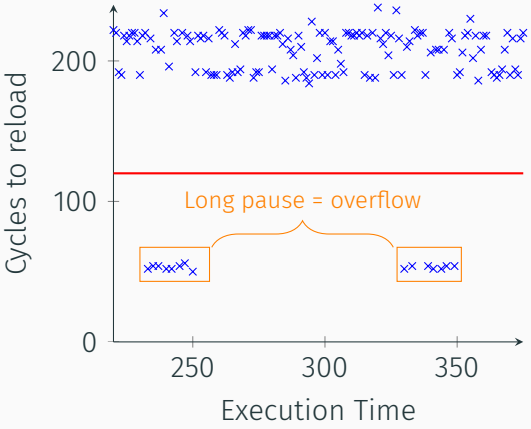


Rules ($b \in \{0,1\}$):

- `bbbb` $\Rightarrow$ $111b$
- `bbbbb` $\Rightarrow$ $yyyyb$, $yyyy \in \{110b, 10bb, 0111\}$
- `bb...b` $\Rightarrow$ $0 \ldots 0yyyyb$

bbbb	bbbbb	bbbbbb	bbbbb	bbbbb	bbbb
4	5	6	5	5	4
⇓	⇓				
111b	yyyyb				

Trace Interpretation

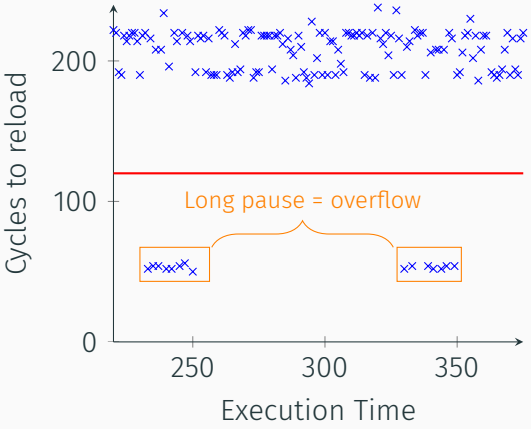


Rules ($b \in \{0,1\}$):

- `bbbb` $\Rightarrow$ $111b$
- `bbbbb` $\Rightarrow$ $yyyyb$, $yyyy \in \{110b, 10bb, 0111\}$
- `bb...b` $\Rightarrow$ $0\ldots0yyyyb$

```
bbbb  bbbbb  bbbbbb  bbbbb  bbbbb  bbbb
  4     5       6      5      5      4
  ⇓     ⇓       ⇓
111b  yyyyb  0yyyyb
```

Trace Interpretation

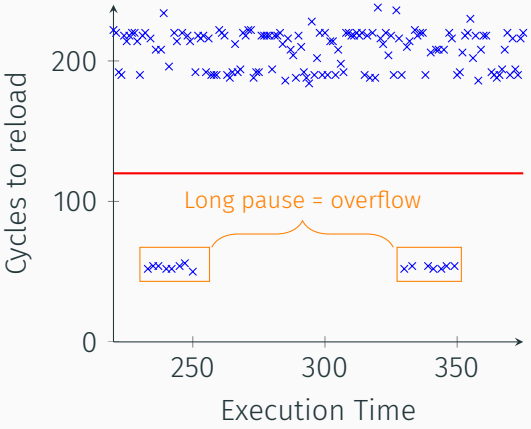


Rules ($b \in \{0,1\}$):

- `bbbb` $\Rightarrow$ $111b$
- `bbbbb` $\Rightarrow$ $yyyyb$, $yyyy \in \{110b, 10bb, 0111\}$
- `bb...b` $\Rightarrow$ $0\ldots0yyyyb$

```
bbbb  bbbbb  bbbbbb  bbbbb  bbbbb  bbbb
  4     5       6      5      5      4
  ⇓     ⇓       ⇓      ⇓
111b  yyyyb  0yyyyb  yyyyb
```

Trace Interpretation

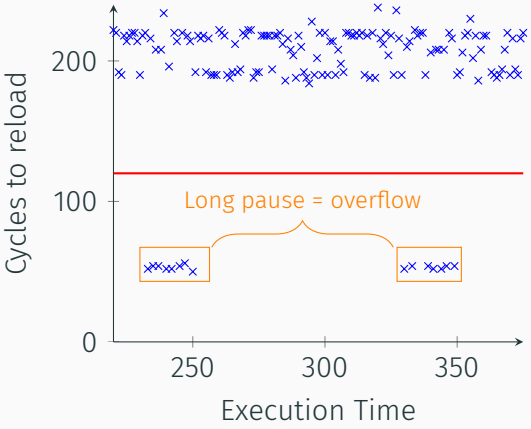


Rules ($b \in \{0,1\}$):

- `bbbb` $\Rightarrow$ $111b$
- `bbbbb` $\Rightarrow$ $yyyyb$, $yyyy \in \{110b, 10bb, 0111\}$
- `bb...b` $\Rightarrow$ $0\ldots0yyyyb$

```
bbbb bbbbb bbbbbb bbbbb bbbbb bbbb
  4    5      6     5     5     4
  ⇓    ⇓      ⇓     ⇓     ⇓
111b yyyyb 0yyyyb yyyyb yyyyb
```

Trace Interpretation



Rules ($b \in \{0,1\}$):

- `bbbb` $\Rightarrow$ $111b$
- `bbbbb` $\Rightarrow$ $yyyyb$, $yyyy \in \{110b, 10bb, 0111\}$
- `bb...b` $\Rightarrow$ $0\ldots0yyyyb$

`bbbb`	`bbbbb`	`bbbbbb`	`bbbbb`	`bbbbb`	`bbbb`
4	5	6	5	5	4
⇓	⇓	⇓	⇓	⇓	⇓
`111b`	`yyyyb`	`0yyyyb`	`yyyyb`	`yyyyb`	`bbbb`

Dictionary Attack

Client: $x = H(salt \,||\, H(user\_id : password))$

$v = g^x \bmod p$

Dictionary Attack

Client: $x = H(salt \,||\, H(user\_id : password))$

$v = g^x \bmod p$

```
trace:    1 1 1 b y y y y b 0 y y y y b 1 1 1 b 0 y y y y b
```

Dictionary Attack

Client: $x = H(salt \,||\, H(user\_id : password))$

$$v = g^x \bmod p$$

trace:	1	1	1	b	y	y	y	y	b	0	y	y	y	y	b	1	1	1	b	0	y	y	y	y	b
pwd_1	1	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	1	1
pwd_2	1	1	0	0	1	0	1	1	1	1	1	1	1	0	0	0	0	0	1	0	1	1	1	0	1
pwd_3	0	1	1	1	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	1	1	0	0	0
pwd_4	1	1	1	1	1	1	0	0	0	0	1	0	1	1	0	1	1	1	0	0	0	1	1	1	1
pwd_5	0	1	1	1	1	0	1	1	1	1	0	0	1	0	1	1	1	0	0	0	0	1	0	0	0
...																									
pwd_n	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	1	0	1	1	0	0	1	0	1

Password	x value

Dictionary Attack

Client: $x = H(salt \,||\, H(user\_id : password))$

$$v = g^x \bmod p$$

trace:	1	1	1	b	y	y	y	y	b	0	y	y	y	y	b	1	1	1	b	0	y	y	y	y	b
pwd_1	1	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	1	1
pwd_2	1	1	0	0	1	0	1	1	1	1	1	1	1	0	0	0	0	0	1	0	1	1	1	0	1
pwd_3	0	1	1	1	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	1	1	0	0	0
pwd_4	1	1	1	1	1	1	0	0	0	0	1	0	1	1	0	1	1	1	0	0	0	1	1	1	1
pwd_5	0	1	1	1	1	0	1	1	1	1	0	0	1	0	1	1	1	0	0	0	0	1	0	0	0
...																									
pwd_n	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	1	0	1	1	0	0	1	0	1

Password	x value

Dictionary Attack

Client: $x = H(salt \,||\, H(user\_id : password))$

$$v = g^x \bmod p$$

trace:	1	1	1	b	y	y	y	y	b	0	y	y	y	y	b	1	1	1	b	0	y	y	y	y	b
pwd_1	1	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	1	1
pwd_2	1	1	0	0	1	0	1	1	1	1	1	1	1	0	0	0	0	0	1	0	1	1	1	0	1
pwd_3	0	1	1	1	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	1	1	0	0	0
pwd_4	1	1	1	1	1	1	0	0	0	0	1	0	1	1	0	1	1	1	0	0	0	1	1	1	1
pwd_5	0	1	1	1	1	0	1	1	1	1	0	0	1	0	1	1	1	0	0	0	0	1	0	0	0
...																									
pwd_n	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	1	0	1	1	0	0	1	0	1

Password	x value

Dictionary Attack

Client: $x = H(salt \,||\, H(user\_id : password))$

$$v = g^x \bmod p$$

trace:	1	1	1	b	y	y	y	y	b	0	y	y	y	y	b	1	1	1	b	0	y	y	y	y	b	
pwd_1	1	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	0	1	1	15
pwd_2	1	1	0	0	1	0	1	1	1	1	1	1	1	0	0	0	0	0	1	0	1	1	1	0	1	14
pwd_3	0	1	1	1	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	1	1	0	0	0	11
pwd_4	1	1	1	1	1	1	0	0	0	0	1	0	1	1	0	1	1	1	0	0	0	1	1	1	1	0
pwd_5	0	1	1	1	1	0	1	1	1	1	0	0	1	0	1	1	1	0	0	0	0	1	0	0	0	11
...																										
pwd_n	1	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1	1	0	1	1	0	0	1	0	1	12

Password	x value	Diff score

Single Measurement Attack

- Very accurate measurement
- Each bit of information halves the number of possible passwords
  - k bits of information $\Rightarrow$ false positive/negative with probability of 2^{-k}

Single Measurement Attack

- Very accurate measurement
- Each bit of information halves the number of possible passwords
  - k bits of information $\Rightarrow$ false positive/negative with probability of 2^{-k}

For an n-bit exponent, we get $k = 0.4n + 2$ bits on average (verified empirically)

SHA-1: 66 bits of information
SHA-256: 104 bits of information

Mitigations

Two choices:

- Patch OpenSSL TLS-SRP by adding the proper flag
  - Most projects use the bignum API, not the whole SRP
  - Difficult to propagate
  - Root cause of the issue remains

- Switch to a secure by default implementation (flag for insecure/optimized)
  - No flag $\Rightarrow$ secure implementation (potential performance loss)
  - All projects are patched at once

Mitigations

Two choices:

- Patch OpenSSL TLS-SRP by adding the proper flag ← OpenSSL's choice
  - Most projects use the bignum API, not the whole SRP
  - Difficult to propagate
  - Root cause of the issue remains

- Switch to a secure by default implementation (flag for insecure/optimized)
  - No flag ⇒ secure implementation (potential performance loss)
  - All projects are patched at once