PARASITE: PAssword Recovery Attack against Srp Implementations in ThE wild

Daniel De Almeida Braga
Pierre-Alain Fouque
Mohamed Sabt

IRMAR - December, 3rd 2021

Cryptography in the Wild: The Security of Cryptographic Implementations and Standards

What I Have Been Doing

Cryptography in the Wild: The Security of Cryptographic Implementations and Standards

- Smart Cards protocol (SCP10)

Cryptography in the Wild: The Security of Cryptographic Implementations and Standards

- Smart Cards protocol (SCP10)
- Password Authenticated Key Exchange (PAKE)

What I Have Been Doing

Cryptography in the Wild: The Security of Cryptographic Implementations and Standards

- Smart Cards protocol (SCP10)
- Password Authenticated Key Exchange (PAKE)
  - Dragonfly (WPA3, EAP-pwd)

What I Have Been Doing

Cryptography in the Wild: The Security of Cryptographic Implementations and Standards

- Smart Cards protocol (SCP10)
- Password Authenticated Key Exchange (PAKE)
  - Dragonfly (WPA3, EAP-pwd)
  - SRP (deployed in a lot of projects)

What I Have Been Doing

Cryptography in the Wild: The Security of Cryptographic Implementations and Standards

- Smart Cards protocol (SCP10)
- Password Authenticated Key Exchange (PAKE)
  - Dragonfly (WPA3, EAP-pwd)
  - SRP (deployed in a lot of projects)
- Recent interest in DRM systems

What I Have Been Doing

Cryptography in the Wild: The Security of Cryptographic Implementations and Standards

- Smart Cards protocol (SCP10)
- Password Authenticated Key Exchange (PAKE)
  - Dragonfly (WPA3, EAP-pwd)
  - SRP (deployed in a lot of projects)
- Recent interest in DRM systems
- Formally verified implementations and constant-time verification tools

Cryptography in the Wild: The Security of Cryptographic Implementations and Standards

- Smart Cards protocol (SCP10)
- Password Authenticated Key Exchange (PAKE)
    - Dragonfly (WPA3, EAP-pwd)
    - SRP (deployed in a lot of projects)
- Recent interest in DRM systems
- Formally verified implementations and constant-time verification tools

PARASITE: PAssword Recovery Attack against Srp Implementations in ThE wild

Context and Motivations

A Few Words About PAKEs

What to expect from a PAKE, starting from a password:

- Authentication
- End up with strong key
- Resist to (offline) dictionary attack

Lot's of different PAKEs (two main families: balanced - asymmetric).

A Few Words About PAKEs

What to expect from a PAKE, starting from a password:

- Authentication
- End up with strong key
- **Resist to (offline) dictionary attack**

Lot's of different PAKEs (two main families: balanced - asymmetric).

Why Looking at PAKEs?

Recent interest (WPA3 and standardization) with practical security considerations

Why Looking at PAKEs?

Recent interest (WPA3 and standardization) with practical security considerations

- Dragonfly and WPA3: Dragonblood[1] and attack refinement[2]

[1] M.Vanhoef and E.Ronen *Dragonblood: Analyzing the Dragonfly Handshake of WPA3 and EAP-pwd.* In IEEE S&P. 2020
[2] D.Braga et al. *Dragonblood Is Still Leaking: Practical Cache-based Side-Channel in the Wild.* In ACSAC. 2020

Why Looking at PAKEs?

Recent interest (WPA3 and standardization) with practical security considerations

- Dragonfly and WPA3: Dragonblood[1] and attack refinement[2]
- Partitioning Oracle Attack[3] applied to some OPAQUE implementations

[1] M.Vanhoef and E.Ronen *Dragonblood: Analyzing the Dragonfly Handshake of WPA3 and EAP-pwd.* In IEEE S&P. 2020
[2] D.Braga et al. *Dragonblood Is Still Leaking: Practical Cache-based Side-Channel in the Wild.* In ACSAC. 2020
[3] J.Len et al. *Partitioning Oracle Attack.* In USENIX Security. 2021

Why Looking at PAKEs?

Recent interest (WPA3 and standardization) with practical security considerations

- Dragonfly and WPA3: Dragonblood[1] and attack refinement[2]
- Partitioning Oracle Attack[3] applied to some OPAQUE implementations

Lesson to learn: Small leakage can be devastating

[1] M.Vanhoef and E.Ronen *Dragonblood: Analyzing the Dragonfly Handshake of WPA3 and EAP-pwd.* In IEEE S&P. 2020
[2] D.Braga et al. *Dragonblood Is Still Leaking: Practical Cache-based Side-Channel in the Wild.* In ACSAC. 2020
[3] J.Len et al. *Partitioning Oracle Attack.* In USENIX Security. 2021

Why Looking at PAKEs?

Recent interest (WPA3 and standardization) with practical security considerations

- Dragonfly and WPA3: Dragonblood[1] and attack refinement[2]
- Partitioning Oracle Attack[3] applied to some OPAQUE implementations

Lesson to learn: Small leakage can be devastating

Case study: Secure Remote Password (SRP)

[1] M.Vanhoef and E.Ronen *Dragonblood: Analyzing the Dragonfly Handshake of WPA3 and EAP-pwd.* In IEEE S&P. 2020
[2] D.Braga et al. *Dragonblood Is Still Leaking: Practical Cache-based Side-Channel in the Wild.* In ACSAC. 2020
[3] J.Len et al. *Partitioning Oracle Attack.* In USENIX Security. 2021

SRP in a Few Words

Asymmetric PAKE, among the first (free) design $\Rightarrow$ de facto standard for $\approx$20 years

SRP in a Few Words

Asymmetric PAKE, among the first (free) design $\Rightarrow$ de facto standard for $\approx$20 years

What about SRP implementations in the wild?

- Still widely deployed and used
- Not much recent work on it

SRP in a Few Words

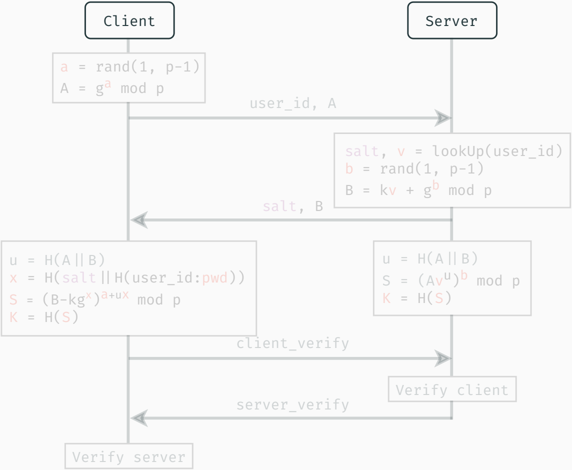
Asymmetric PAKE, among the first (free) design $\Rightarrow$ de facto standard for $\approx$20 years

What about SRP implementations in the wild?

- Still widely deployed and used
- Not much recent work on it
- Recent work on SRP at ACNS[4]

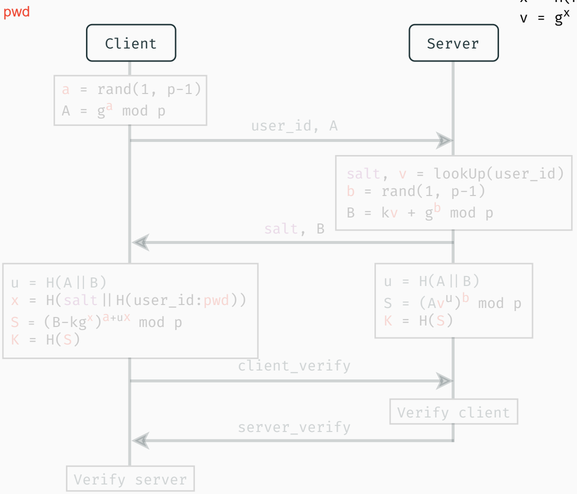
[4] A.Russon *Threat for the Secure Remote Password Protocol and a Leak in Apple's Cryptographic Library.* In ACNS. 2021

SRP Protocol Overview

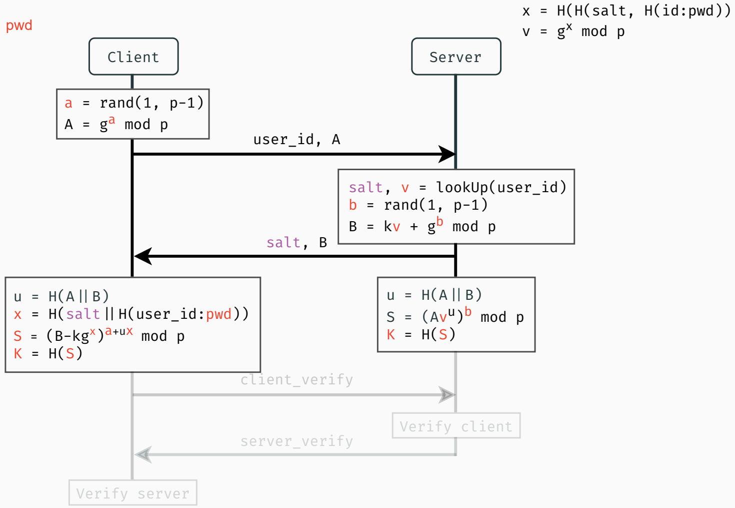


SRP Protocol Overview

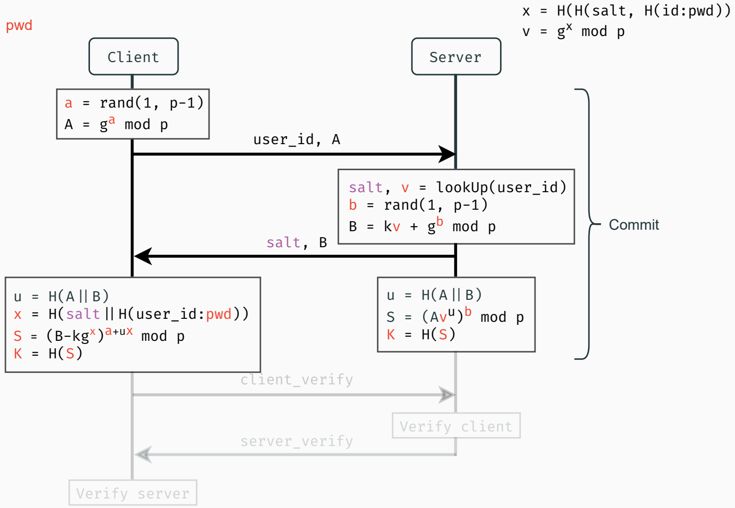
$x = H(H(salt, H(id:pwd))$
$v = g^x \bmod p$



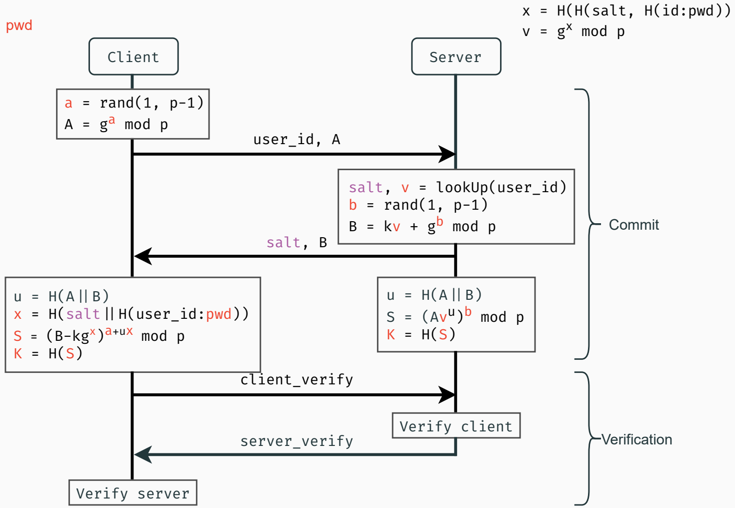
SRP Protocol Overview



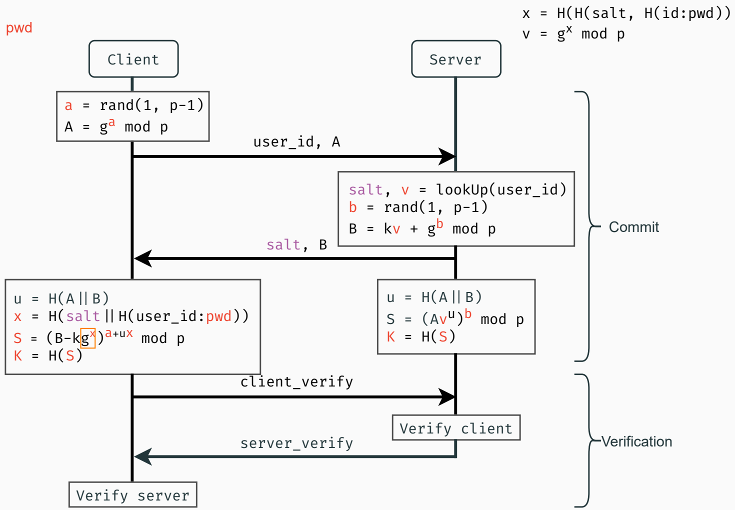
SRP Protocol Overview



SRP Protocol Overview



SRP Protocol Overview



Contributions

Contributions

1. Study various SRP implementations
2. Highlight a leakage in the root library used for big number arithmetic (OpenSSL)
3. Design PoCs[1] of an offline dictionary attack recovering the password on impacted projects
4. Outline the importance of SCA, especially for PAKEs

[1] `https://gitlab.inria.fr/ddealmei/poc-openssl-srp`

Our Main Result

A cache-attack that let us extract information

during OpenSSL modular exponentiation

allowing to recover the password in a single measure

Our Main Result

FLUSH+RELOAD[1] and PDA[2]

A cache-attack that let us extract information during OpenSSL modular exponentiation allowing to recover the password in a single measure

[1] Y. Yarom et al. *Flush+Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack.* In USENIX Security Symposium. 2014.
[2] T. Allan et al. *Amplifying side channels through performance degradation.* In ACSAC. 2016

Side Channel Attacks

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

Side Channel Attacks

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

Gain information through timing:

0.5 seconds $\Rightarrow$ no a

10 seconds $\Rightarrow$ a

Side Channel Attacks

```
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 through timing:

- 0.5 seconds ⇒ no *a*
- 10 seconds ⇒ *a*

Side Channel Attacks

```
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 through timing:

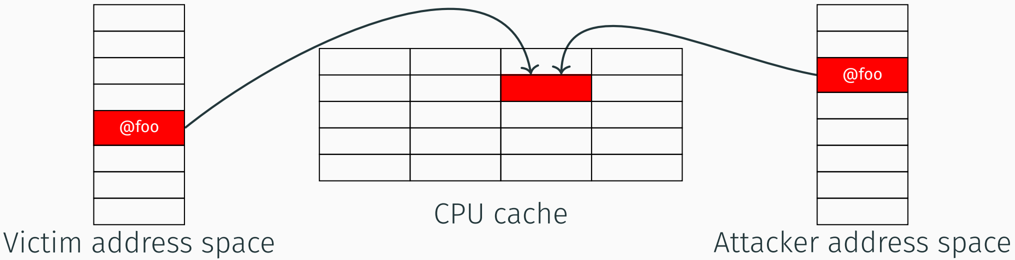
0.5 seconds ⇒ no *a*

10 seconds ⇒ *a*

Gain information execution flow:

- Execute `long_processing` ⇒ *a*
- Else, no *a* in pwd

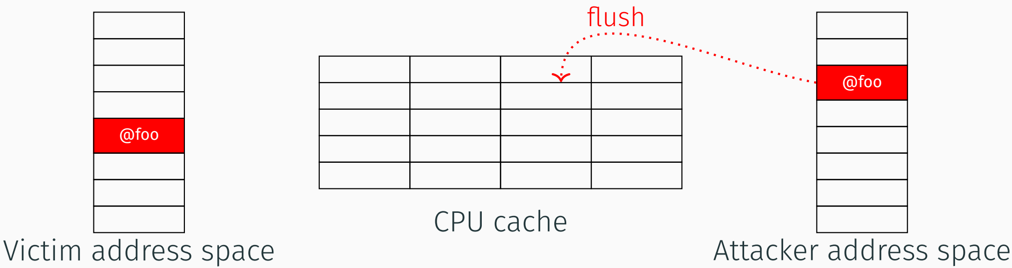
FLUSH+RELOAD[1]



1. Maps the victim's address space

[1] Y. Yarom et al. *Flush+Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack*. In USENIX Security Symposium. 2014.

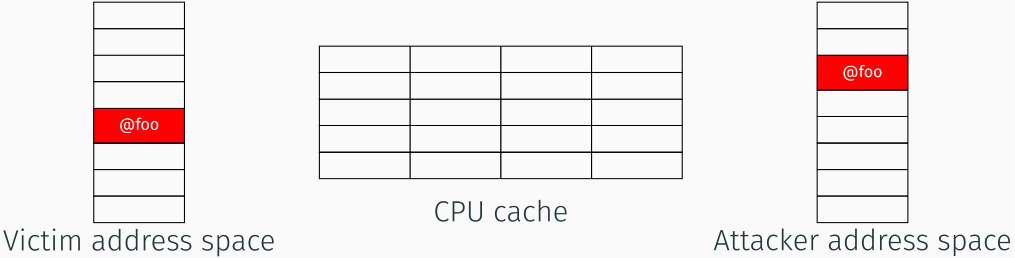
Flush+Reload[1]



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

[1] Y. Yarom et al. *Flush+Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack*. In USENIX Security Symposium. 2014.

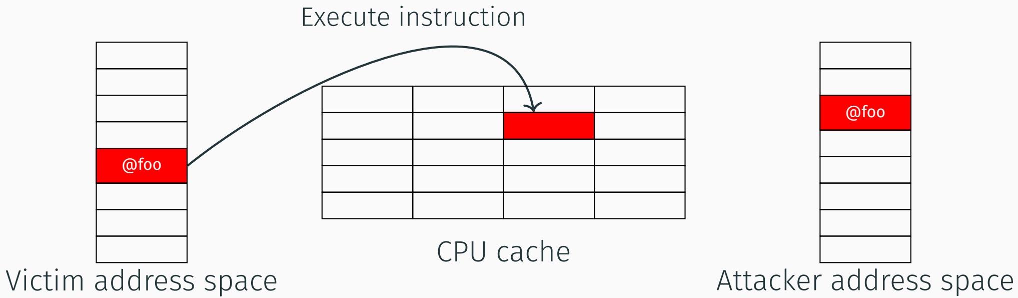
FLUSH+RELOAD[1]



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

[1] Y. Yarom et al. *Flush+Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack*. In USENIX Security Symposium. 2014.

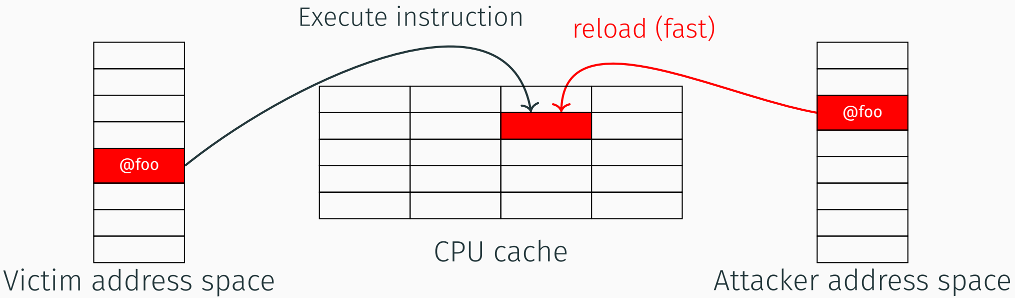
FLUSH+RELOAD[1]



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

[1] Y. Yarom et al. *Flush+Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack*. In USENIX Security Symposium. 2014.

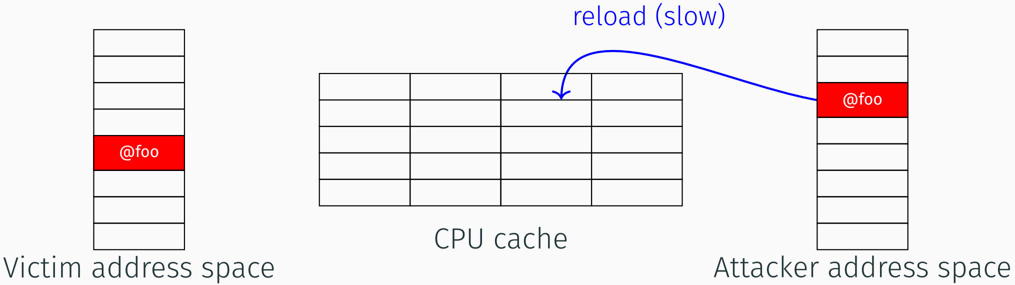
Flush+Reload[1]



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

[1] Y. Yarom et al. *Flush+Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack*. In USENIX Security Symposium. 2014.

Flush+Reload[1]



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

[1] Y. Yarom et al. *Flush+Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack*. In USENIX Security Symposium. 2014.

Our Main Result

FLUSH+RELOAD[1] and PDA[2]

A cache-attack that let us extract information

during OpenSSL modular exponentiation

allowing to recover the password in a single measure

[1] Y. Yarom et al. *Flush+Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack.* In USENIX Security Symposium. 2014.
[2] T. Allan et al. *Amplifying side channels through performance degradation.* In ACSAC. 2016

Our Main Result

FLUSH+RELOAD[1] and PDA[2]

Weak exponentiation algorithm

A cache-attack that let us extract information

during OpenSSL modular exponentiation

allowing to recover the password in a single measure

[1] Y. Yarom et al. *Flush+Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack.* In USENIX Security Symposium. 2014.
[2] T. Allan et al. *Amplifying side channels through performance degradation.* In ACSAC. 2016

Our Main Result

FLUSH+RELOAD[1] and PDA[2]

Weak exponentiation algorithm

A cache-attack that let us extract information

during OpenSSL modular exponentiation

allowing to recover the password in a single measure

Passive offline attack

[1] Y. Yarom et al. *Flush+Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack*. In USENIX Security Symposium. 2014.
[2] T. Allan et al. *Amplifying side channels through performance degradation*. In ACSAC. 2016

Our Main Result

FLUSH+RELOAD[1] and PDA[2]

Weak exponentiation algorithm

A cache-attack that let us extract information during OpenSSL modular exponentiation allowing to recover the password in a single measure

Passive offline attack

No error and lots of information

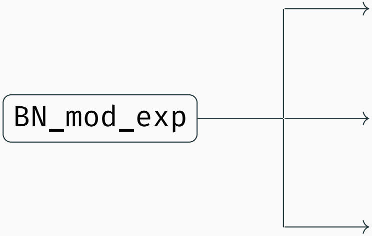
[1] Y. Yarom et al. *Flush+Reload: a High Resolution, Low Noise, L3 Cache Side-Channel Attack.* In USENIX Security Symposium. 2014.
[2] T. Allan et al. *Amplifying side channels through performance degradation.* In ACSAC. 2016

The Vulnerability

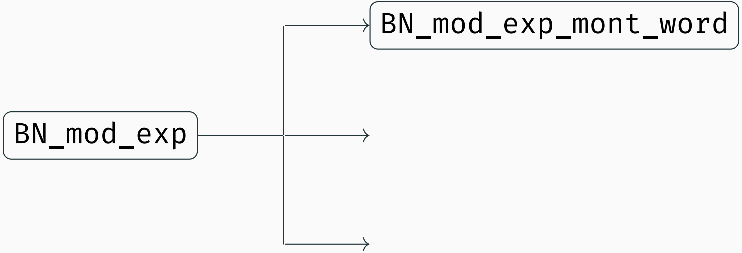
Modular exponentiation in OpenSSL

`BN_mod_exp`

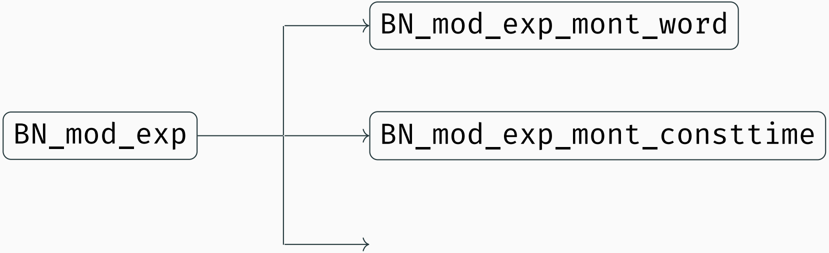
Modular exponentiation in OpenSSL



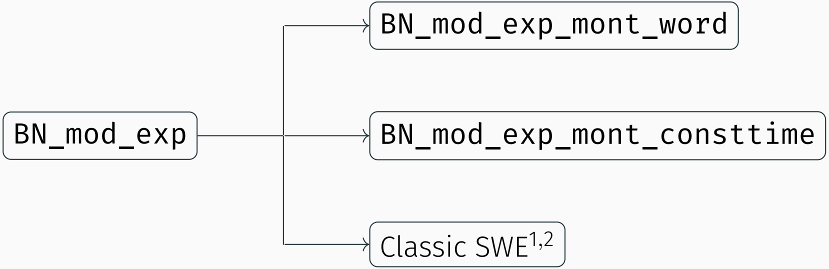
Modular exponentiation in OpenSSL



Modular exponentiation in OpenSSL



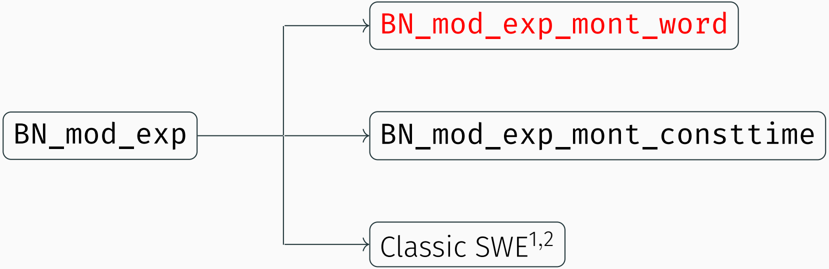
Modular exponentiation in OpenSSL



[1] C. Percival *Cache missing for fun and profit*. 2005

[2] C. Peraida Garia 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 Garia 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)

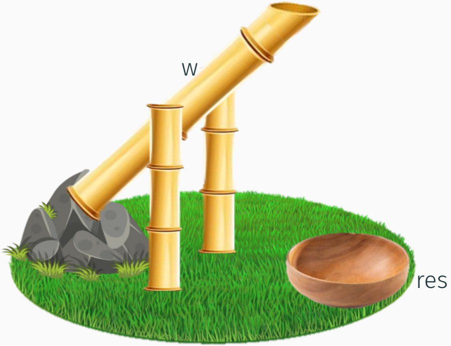
```
def BN_mod_exp_mont_word(g, x, p):
  w = g                           # uint64_t
  res = BN_to_mont_word(w)        # bignum
  for b in range(bitlen-2, 0, -1):
    next_w = w × w
    if (next_w / w) != w:
      res = BN_mod_mul(res, w, p)
      next_w = 1
    w = next_w;
    res = BN_mod_sqr(res, p)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



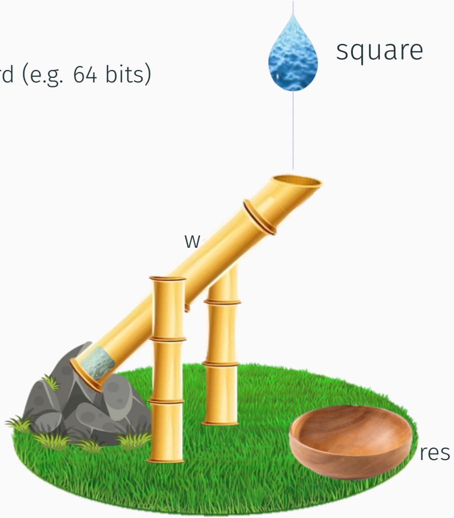

```
def BN_mod_exp_mont_word(g, x, p):
  w = g                          # uint64_t
  res = BN_to_mont_word(w)  # bignum
→ for b in range(bitlen-2, 0, -1):
    next_w = w × w
    if (next_w / w) != w:
      res = BN_mod_mul(res, w, p)
      next_w = 1
    w = next_w;
    res = BN_mod_sqr(res, p)
    if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



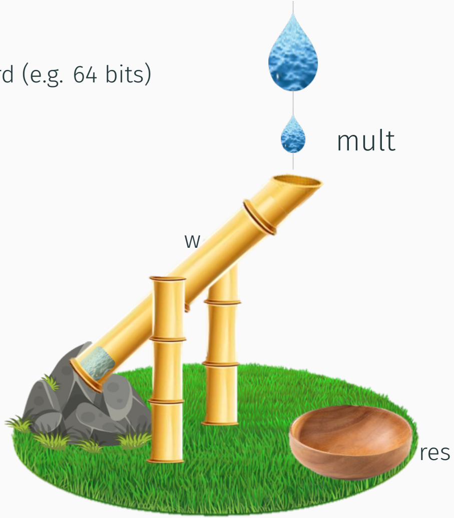

```
def BN_mod_exp_mont_word(g, x, p):
  w = g                         # uint64_t
  res = BN_to_mont_word(w)      # bignum
  for b in range(bitlen-2, 0, -1):
  → next_w = w × w
    if (next_w / w) != w:
      res = BN_mod_mul(res, w, p)
      next_w = 1
    w = next_w;
    res = BN_mod_sqr(res, p)
 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



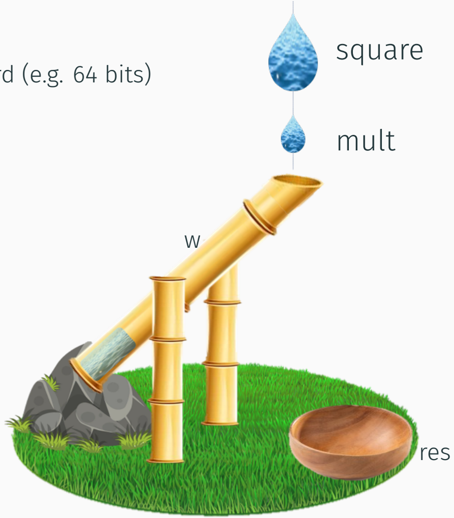

```
def BN_mod_exp_mont_word(g, x, p):
  w = g                          # uint64_t
  res = BN_to_mont_word(w)       # bignum
  for b in range(bitlen-2, 0, -1):
    next_w = w × w
    if (next_w / w) != w:
      res = BN_mod_mul(res, w, p)
      next_w = 1
    w = next_w;
    res = BN_mod_sqr(res, p)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



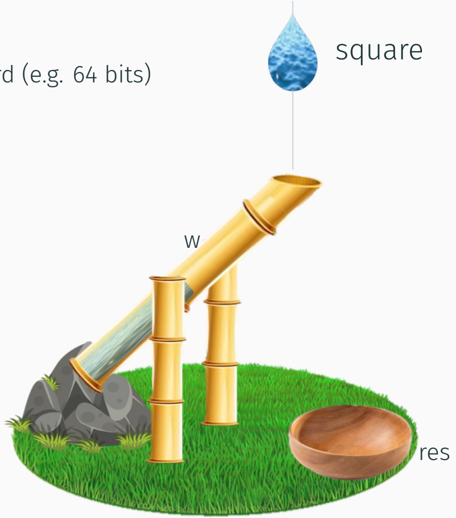

```
def BN_mod_exp_mont_word(g, x, p):
  w = g                          # uint64_t
  res = BN_to_mont_word(w)   # bignum
  for b in range(bitlen-2, 0, -1):
 →next_w = w × w
    if (next_w / w) != w:
      res = BN_mod_mul(res, w, p)
      next_w = 1
    w = next_w;
    res = BN_mod_sqr(res, p)
    if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



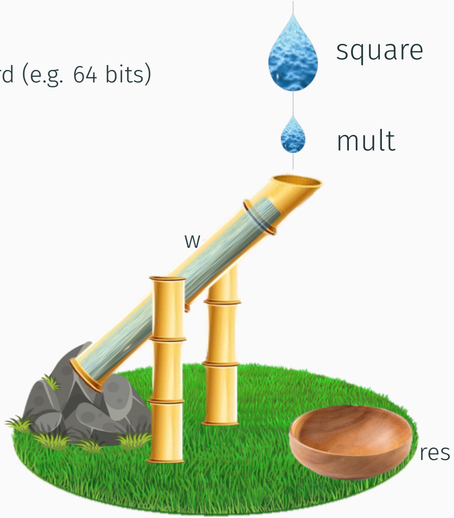

```
def BN_mod_exp_mont_word(g, x, p):
  w = g                          # uint64_t
  res = BN_to_mont_word(w)       # bignum
  for b in range(bitlen-2, 0, -1):
 →  next_w = w × w
    if (next_w / w) != w:
      res = BN_mod_mul(res, w, p)
      next_w = 1
    w = next_w;
    res = BN_mod_sqr(res, p)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



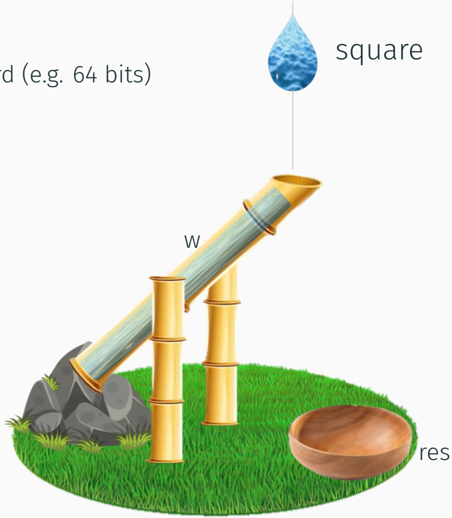

```
def BN_mod_exp_mont_word(g, x, p):
  w = g                          # uint64_t
  res = BN_to_mont_word(w)       # bignum
  for b in range(bitlen-2, 0, -1):
  → next_w = w × w
    if (next_w / w) != w:
      res = BN_mod_mul(res, w, p)
      next_w = 1
    w = next_w;
    res = BN_mod_sqr(res, p)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

w processor word (e.g. 64 bits)

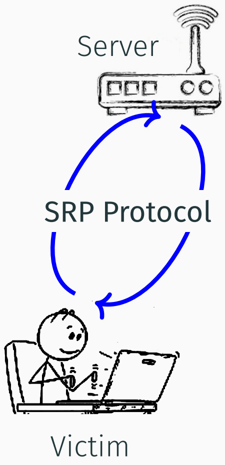

```
def BN_mod_exp_mont_word(g, x, p):
  w = g                          # uint64_t
  res = BN_to_mont_word(w)       # bignum
  for b in range(bitlen-2, 0, -1):
    next_w = w × w
    if (next_w / w) != w:
     → res = BN_mod_mul(res, w, p)
     → next_w = 1
    w = next_w;
    res = BN_mod_sqr(res, p)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

Exploiting the Leakage

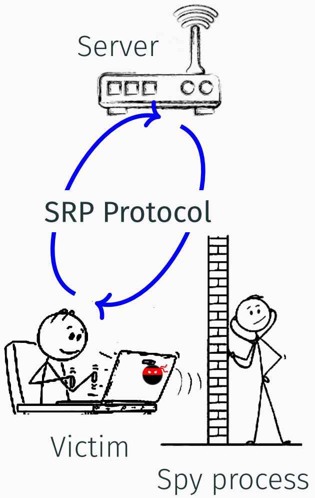
Attacker Model

- Unprivileged spyware on the victim station
- Victim tries to connect
- MitM can help to gather more information (optional)

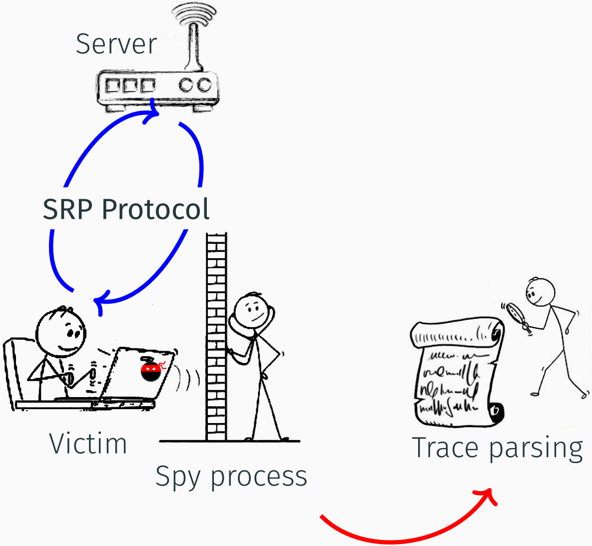
Attack Workflow



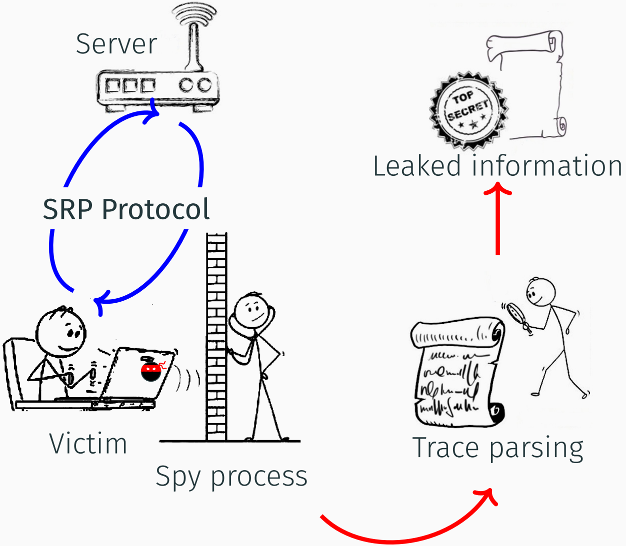
Attack Workflow



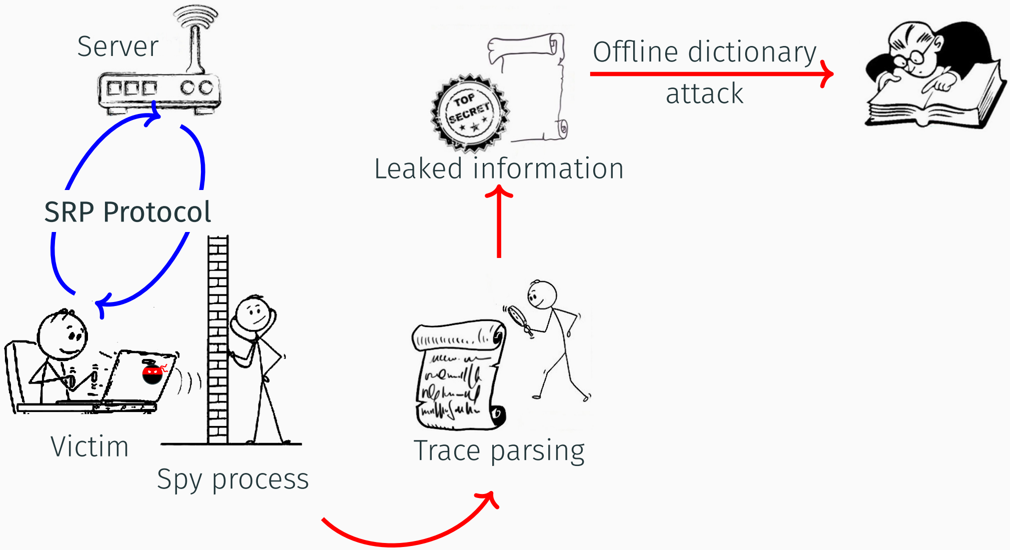
Attack Workflow



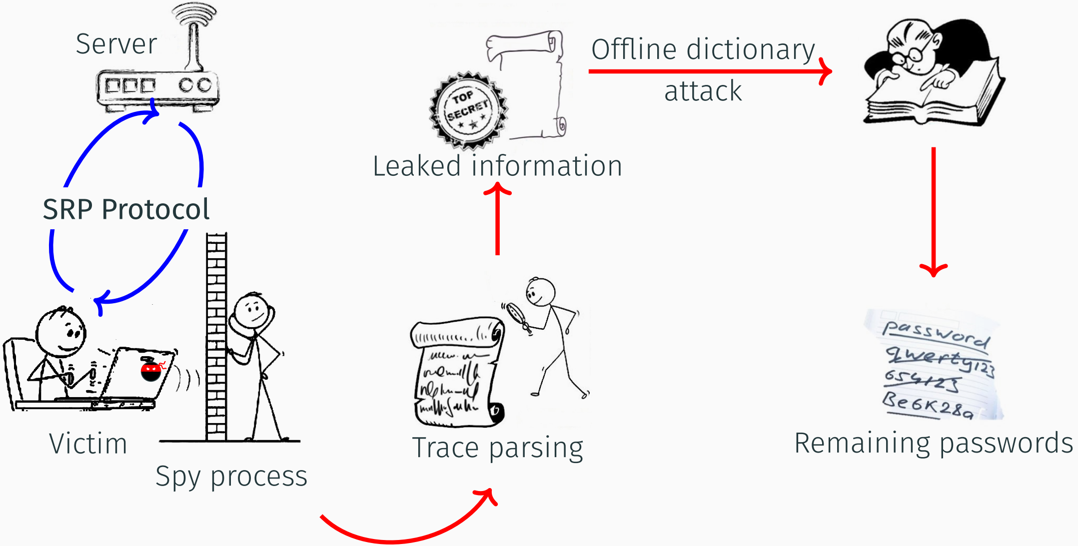
Attack Workflow



Attack Workflow



Attack Workflow



Trace Acquisition

```
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 Acquisition

```
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 Acquisition

```
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 Acquisition

```
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
```

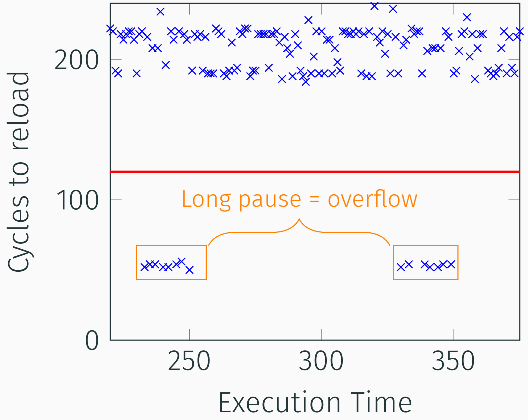
Trace Acquisition

```
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
```

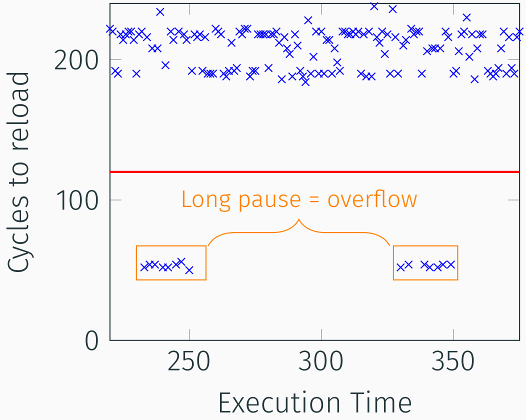
Trace Acquisition

```
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



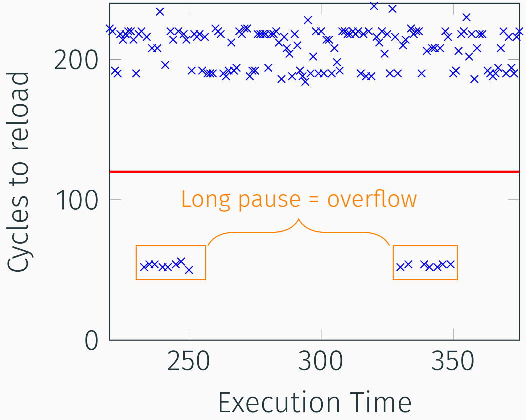
Trace Interpretation



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

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

Trace Interpretation

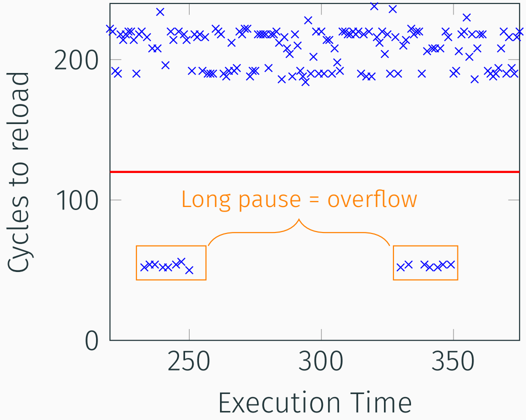


Rules ($b \in \{0,1\}$):

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

bbbb bbbbb bbbbbb bbbbb bbbbb bbbb

Trace Interpretation

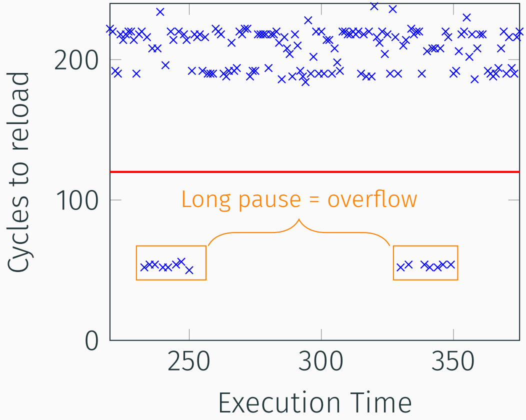


`Rules` ($b \in \{0,1\}$)`:`

- `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

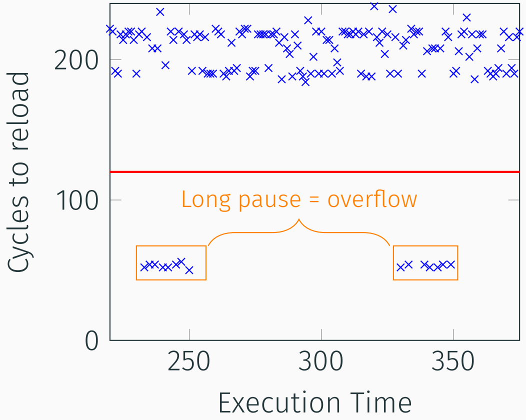


`Rules` ($b \in \{0,1\}$)`:`

- `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

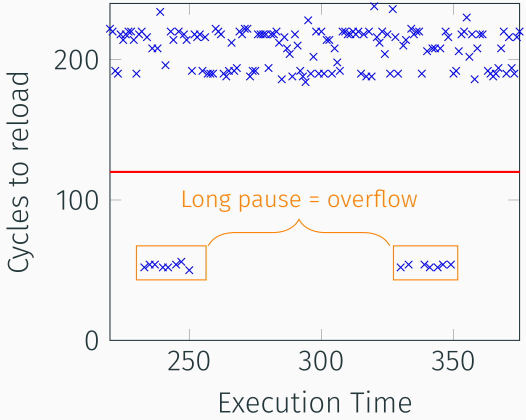


Rules ($b \in \{0,1\}$):

- 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

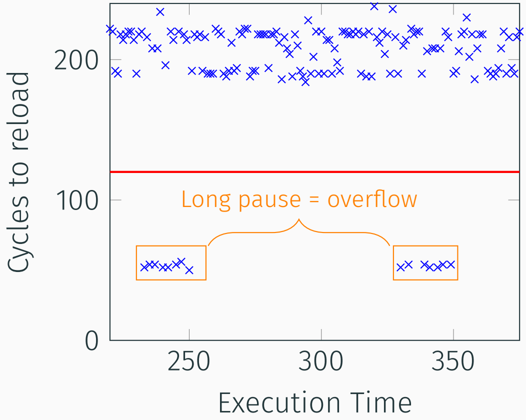


Rules ($b \in \{0,1\}$):

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

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

Trace Interpretation

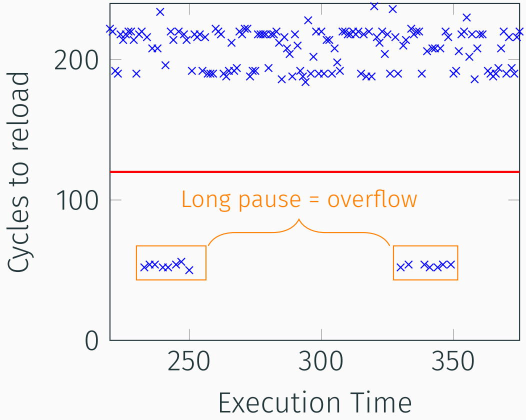


Rules ($b \in \{0, 1\}$):

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

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

Trace Interpretation

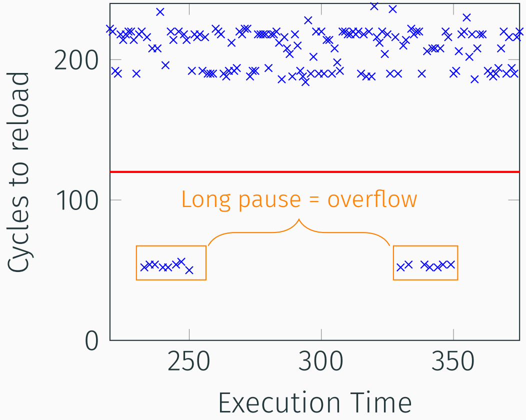


Rules ($b \in \{0, 1\}$):

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

bbbb	bbbbb	bbbbbb	bbbbb	bbbbb	bbbb
4	5	6	5	5	4
$\Downarrow$	$\Downarrow$	$\Downarrow$	$\Downarrow$	$\Downarrow$	
111b	yyyyb	0yyyyb	yyyyb	yyyyb	

Trace Interpretation



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

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

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

Dictionary Attack

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

$v = g^x \bmod p$

Dictionary Attack

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

$v = g^x \bmod p$

```
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: $x = H(salt \mid\mid H(user\_id : password))$

$v = g^x \bmod p$

```
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: $x = H(salt \mid\mid H(user\_id : password))$

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

```
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: $x = H(salt \mid\mid H(user\_id : password))$

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

```
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: $x = H(salt \mid\mid H(user\_id : password))$

$v = g^x \bmod p$

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 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

Practical Impact

Impacted Projects

- Lots of project using OpenSSL are impacted, including
  - OpenSSL TLS-SRP
  - Apple HomeKit ADK
  - Protonmail's python client
  - GoToAssist (?)

Impacted Projects

- Lots of project using OpenSSL are impacted, including
  - OpenSSL TLS-SRP
  - Apple HomeKit ADK
  - Protonmail's python client
  - GoToAssist (?)

Wait, how are big numbers managed in high level languages ?...

Impacted Langages

- Many reference libraries are based on OpenSSL to manage bignums
- They usually (never ?) manage the flag properly
  - Ruby/openssl
  - Javascript node-bignum
  - Erlang OTP
  - PySRP

All SRP implementations using these packages / libraries are affected!

Mitigations & Conclusion

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 $\leftarrow$ 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 $\Rightarrow$ secure implementation (potential performance loss)
  - All projects are patched at once

Conclusion

Practical attack against SRP implementations

- Vulnerability inherited by lots of projects
- Easy to exploit because we can use each recover bits independently

Conclusion

Practical attack against SRP implementations

- Vulnerability inherited by lots of projects
- Easy to exploit because we can use each recover bits independently

Long term lesson: be careful with SCA, especially in PAKE implementation

Conclusion

Practical attack against SRP implementations

- Vulnerability inherited by lots of projects
- Easy to exploit because we can use each recover bits independently

Long term lesson: be careful with SCA, especially in PAKE implementation

Leakage in a weak generic function

- Other protocols with small base may also use it
- Contact use if you think of one!

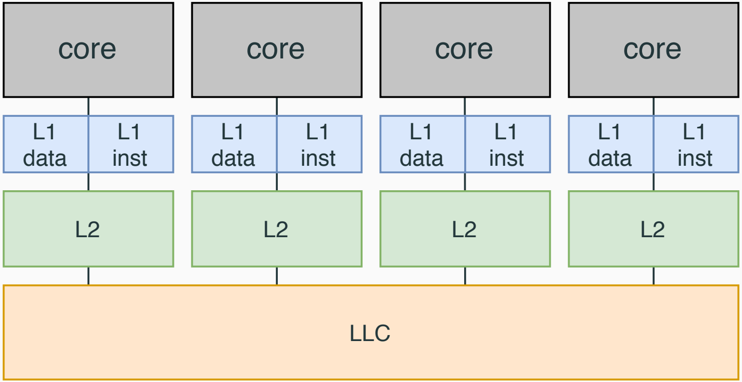
Thank you for your attention!

https://gitlab.inria.fr/ddealmei/poc-openssl-srp

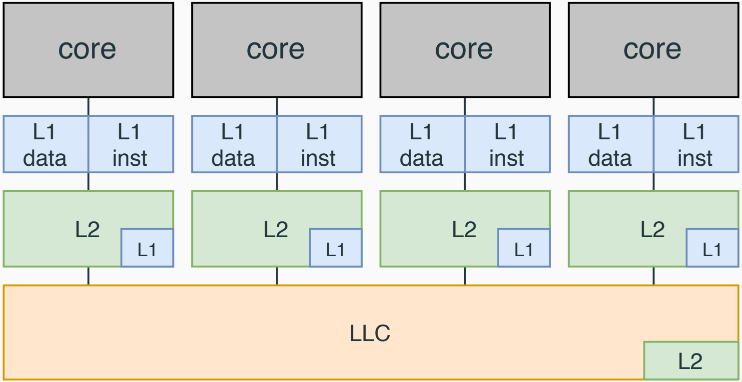
@ daniel.de-almeida-braga@irisa.fr

Backup slides

Intel CPU cache



Intel CPU cache



Inclusive cache