

PARASITE: PAssword Recovery Attack against Srp Implementations in ThE wild

Daniel De Almeida Braga

Pierre-Alain Fouque

Mohamed Sabt

WAC'21 - August, 15th 2021



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

Lots of different PAKEs (two main families: balanced - asymmetric).

Why Looking at PAKEs?

Recent interest (WPA3 and CFRG competition after patents expiration) with practical security considerations

- Dragonfly and WPA3: Dragonblood¹ and attack refinement²
- Partitioning Oracle Attack³ applied to some OPAQUE implementations

Small leakage can be devastating

Case study: Secure Remote Password

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

What about SRP?

Available for a long time => de facto standard for more than 20 years

What about SRP implementations in the wild ?

What about SRP?

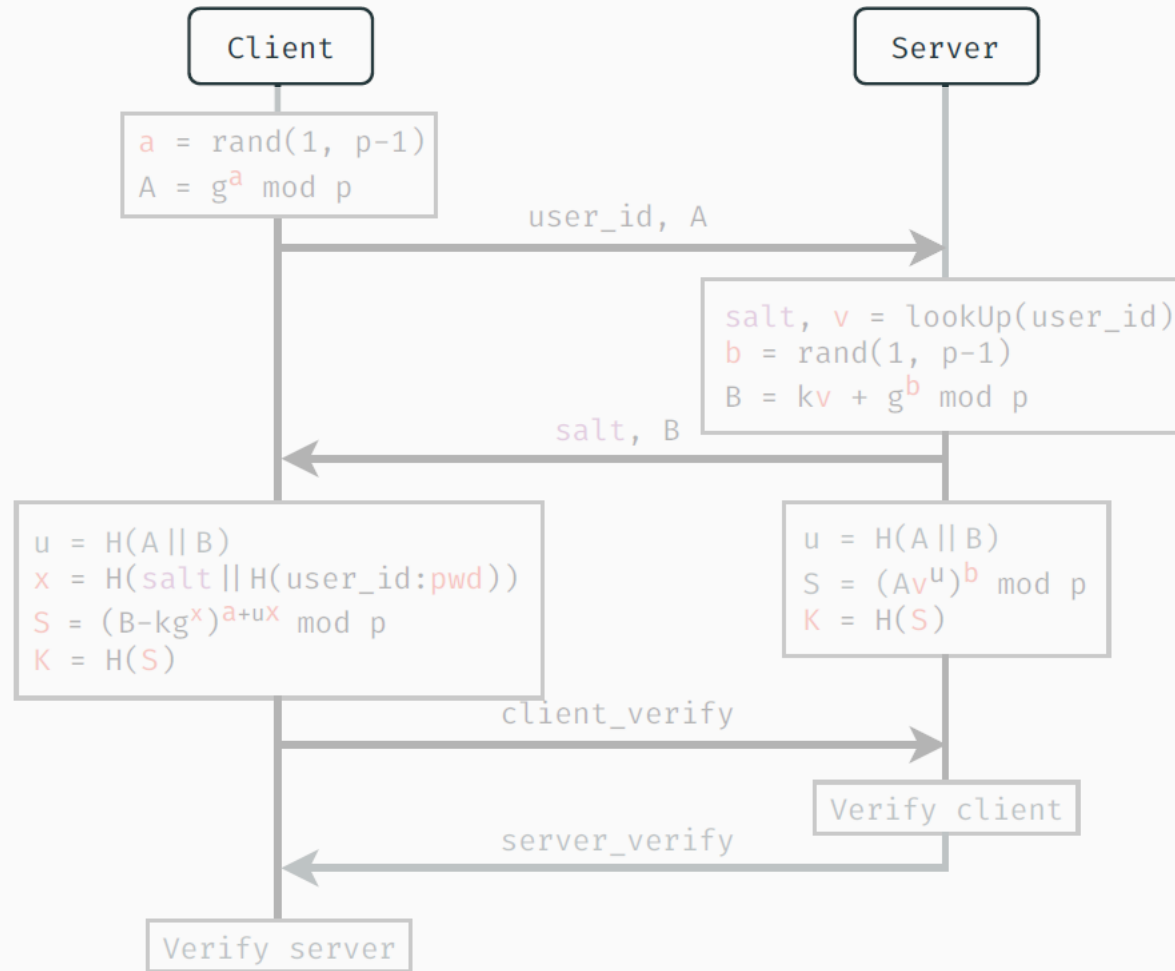
Available for a long time => de facto standard for more than 20 years

What about SRP implementations in the wild ?

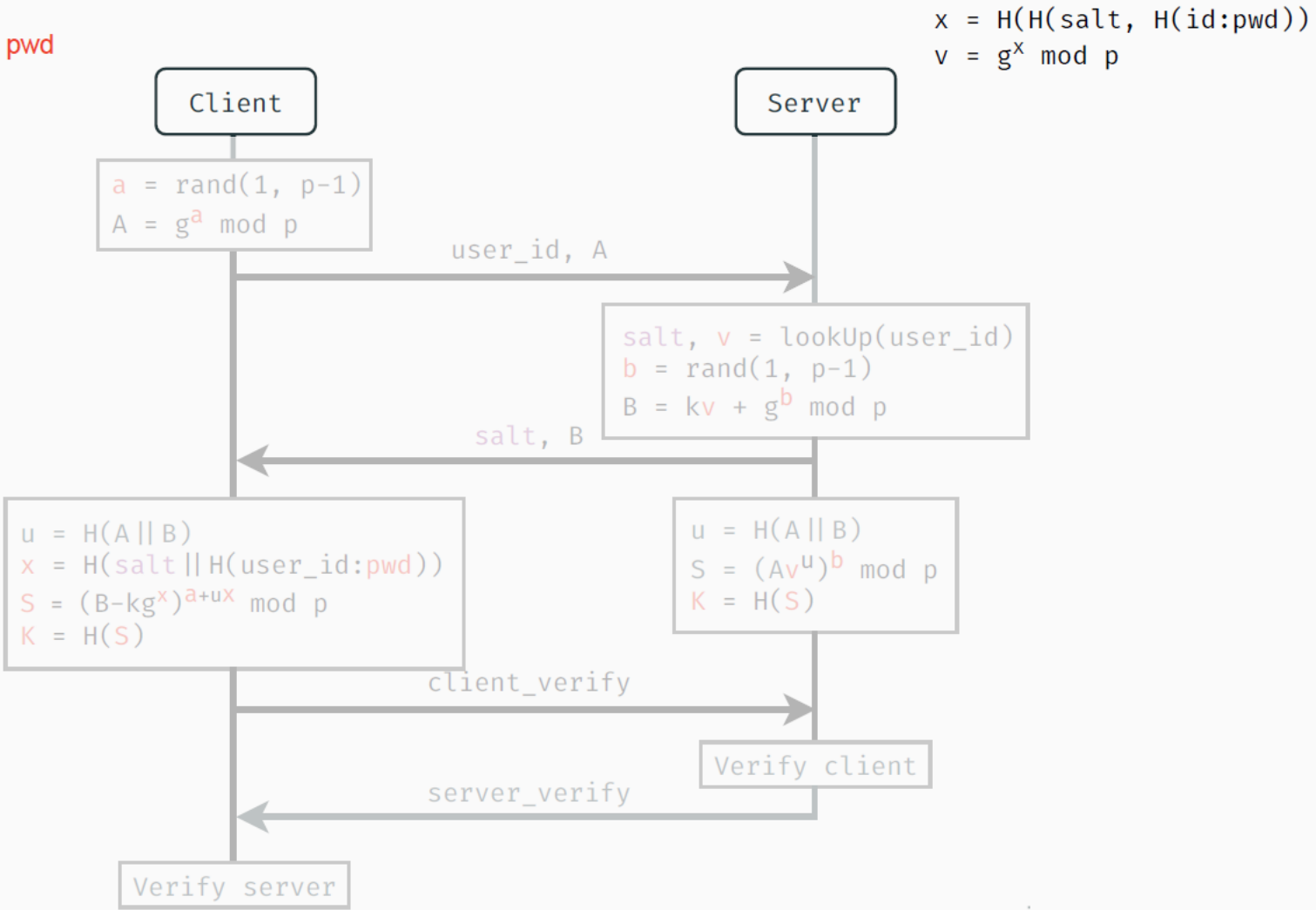
- Recent work on SRP at ACNS¹

¹ A.Russon Threat for the Secure Remote Password Protocol and a Leak in Apple's Cryptographic Library. In ACNS. 2021

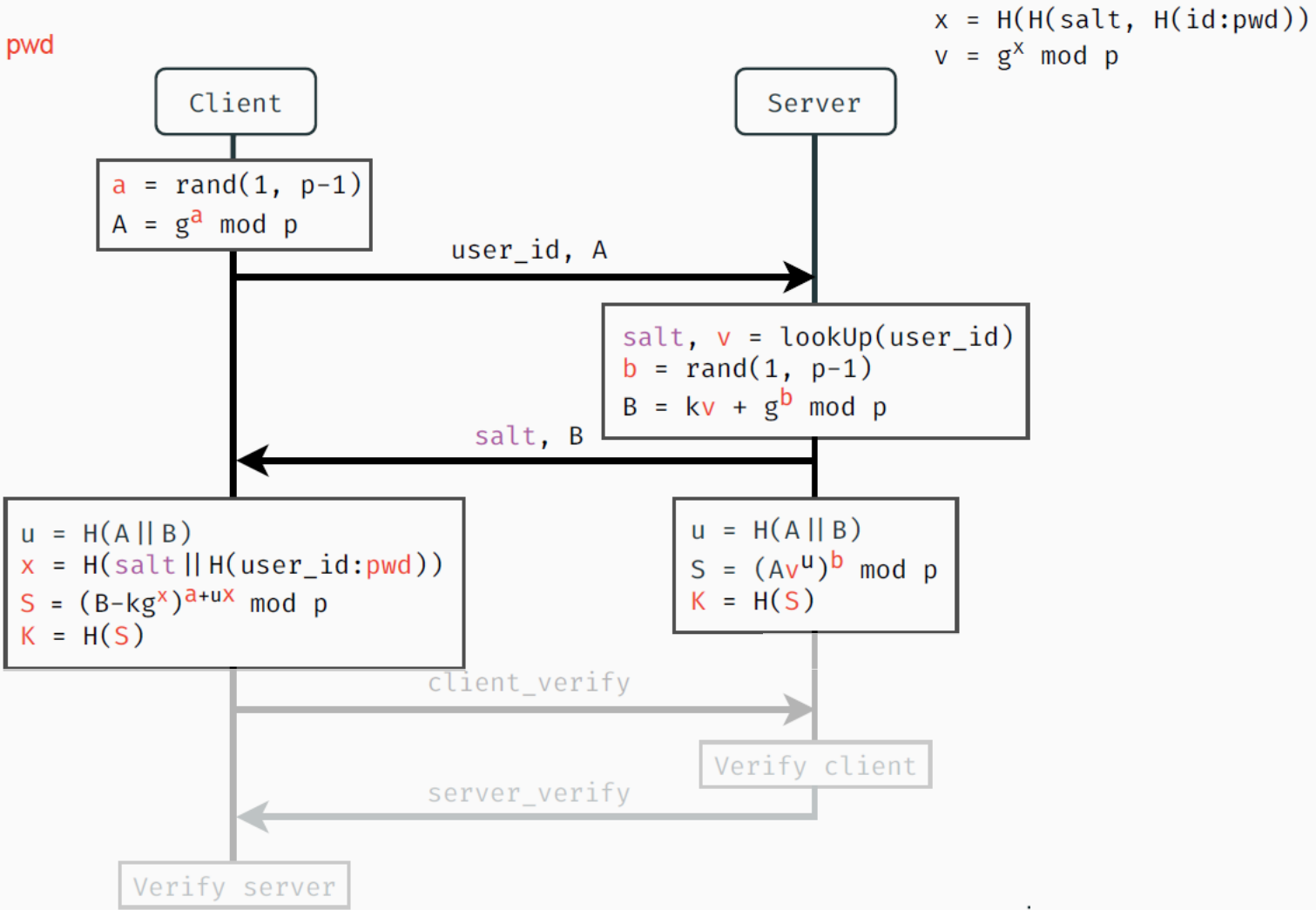
SRP Protocol Overview



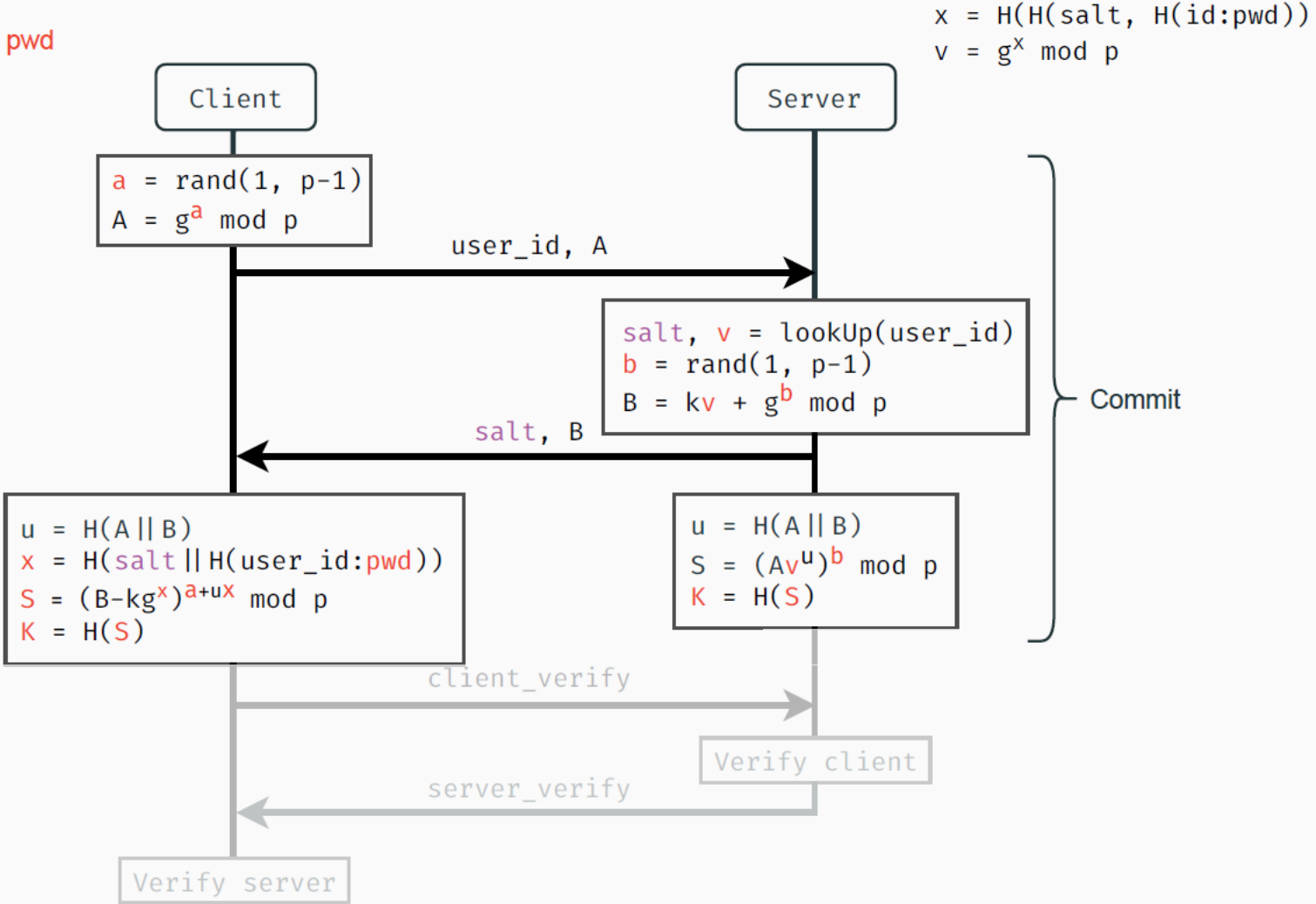
SRP Protocol Overview



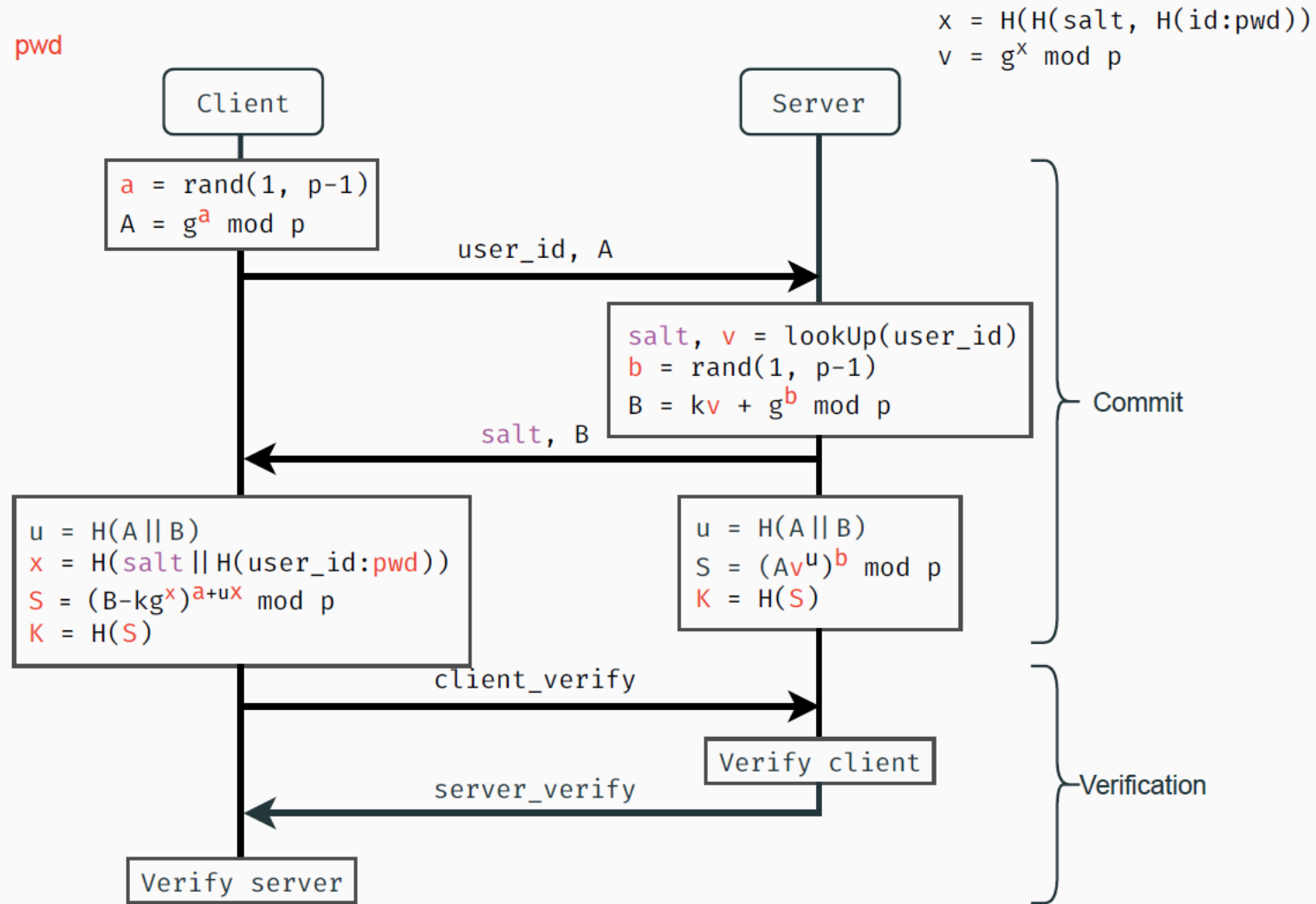
SRP Protocol Overview



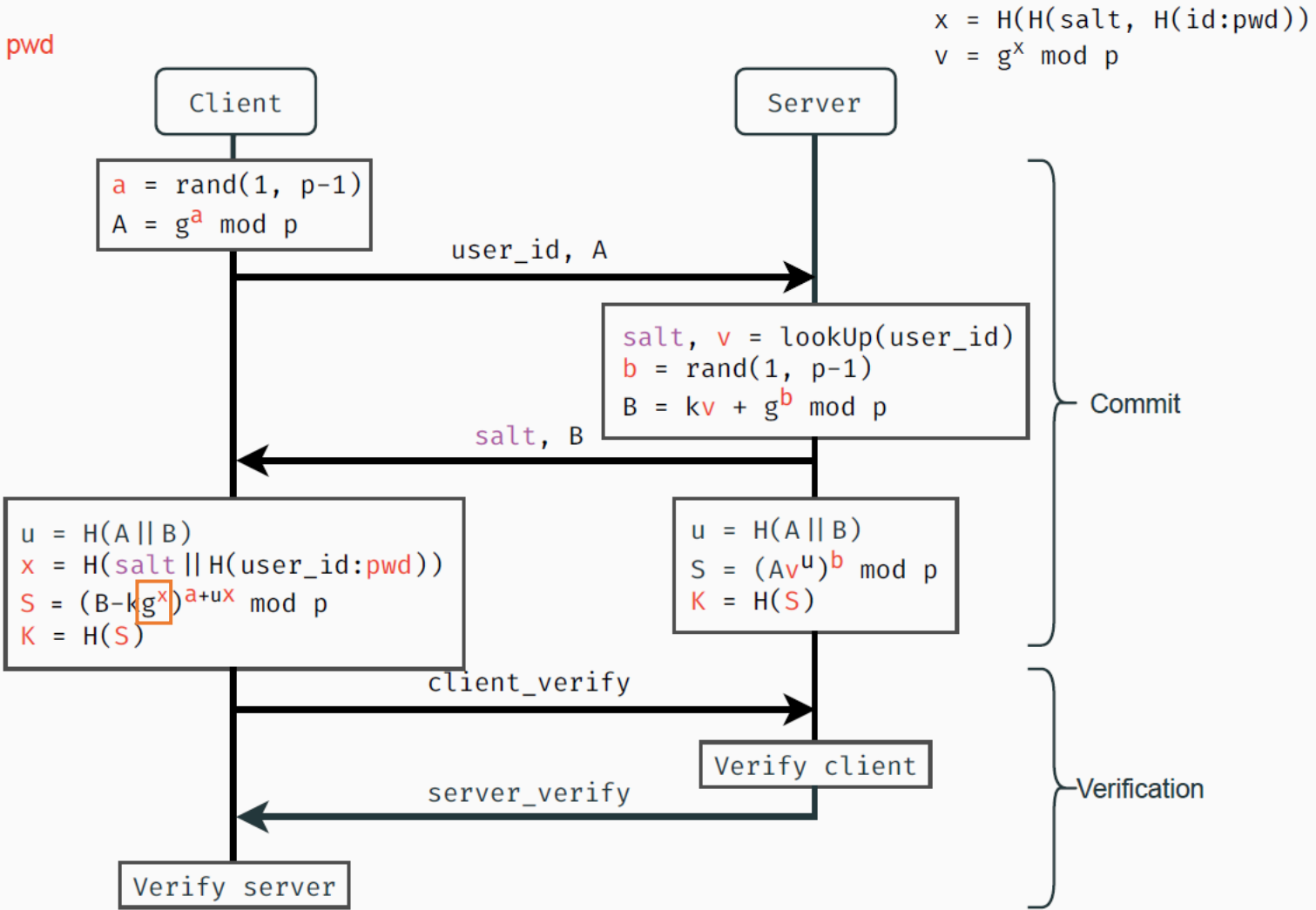
SRP Protocol Overview



SRP Protocol Overview



SRP Protocol Overview



Contributions

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

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

Our Main Result

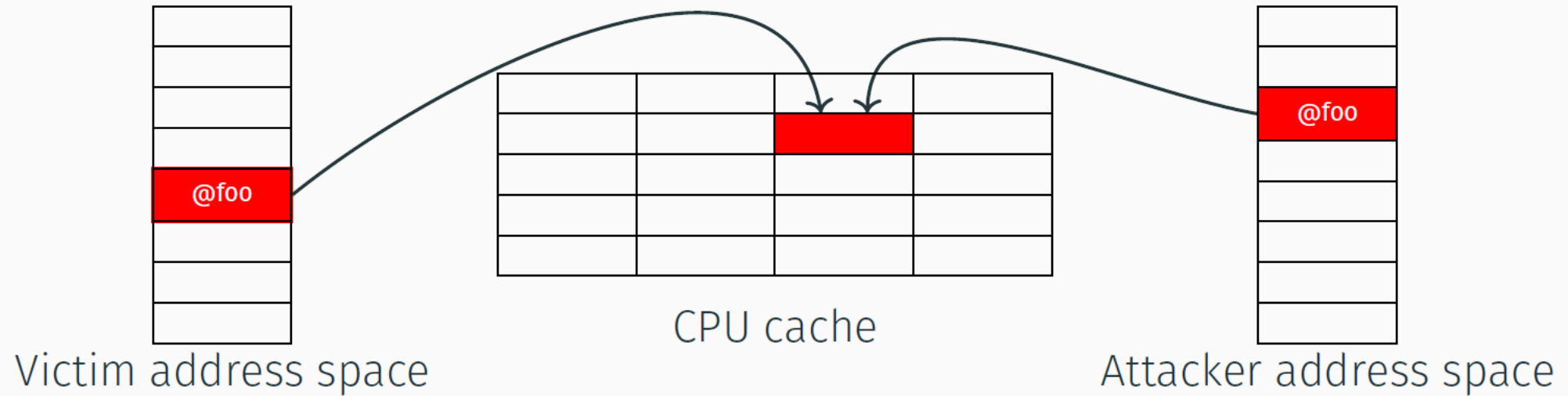
Flush+Reload¹ and PDA²

A **cache-attack** that lets 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

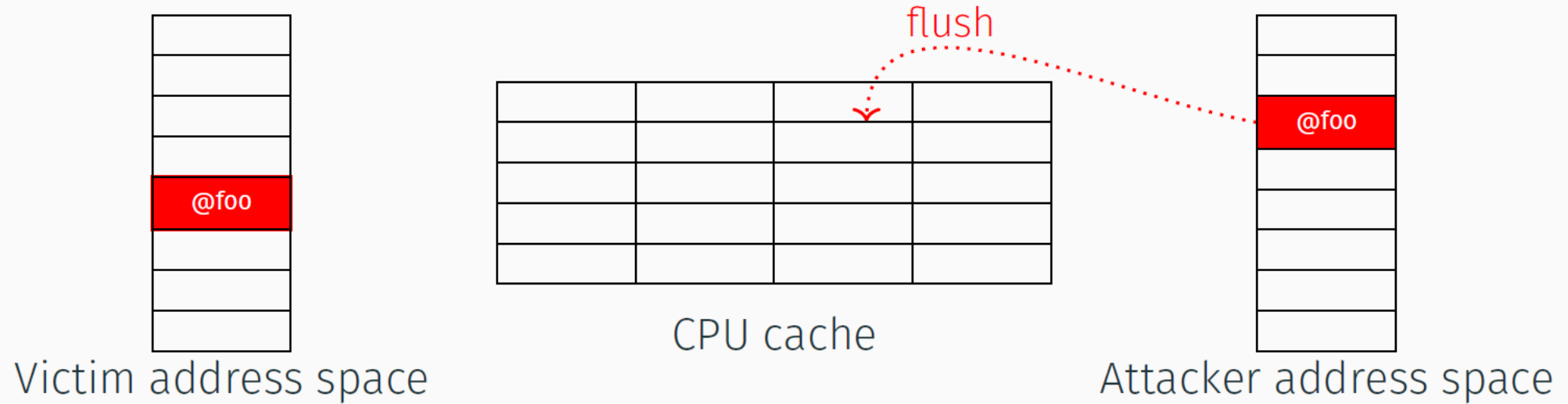
FLUSH+RELOAD¹



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

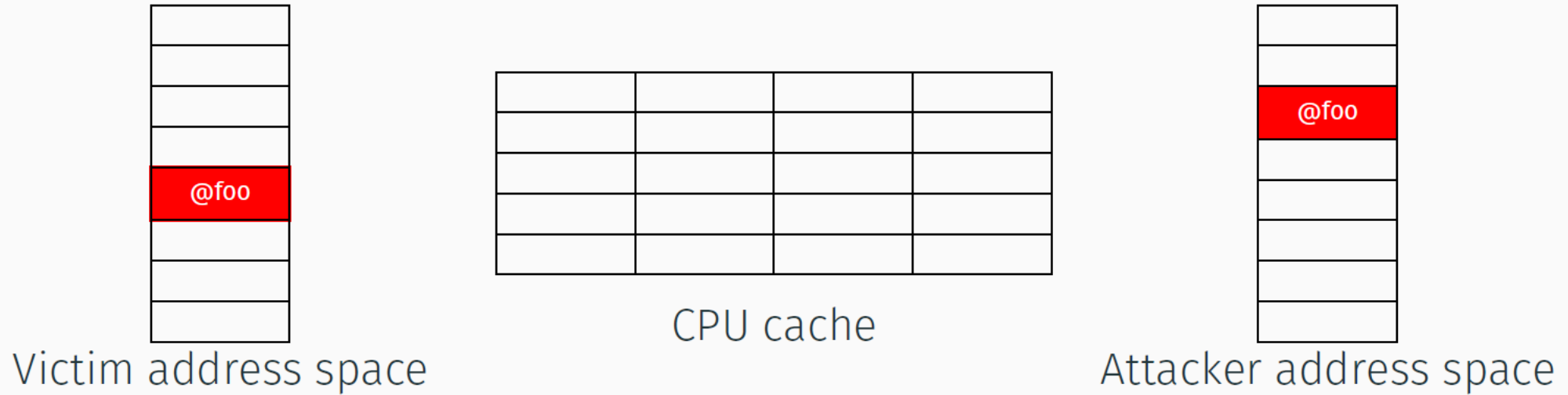
FLUSH+RELOAD¹



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

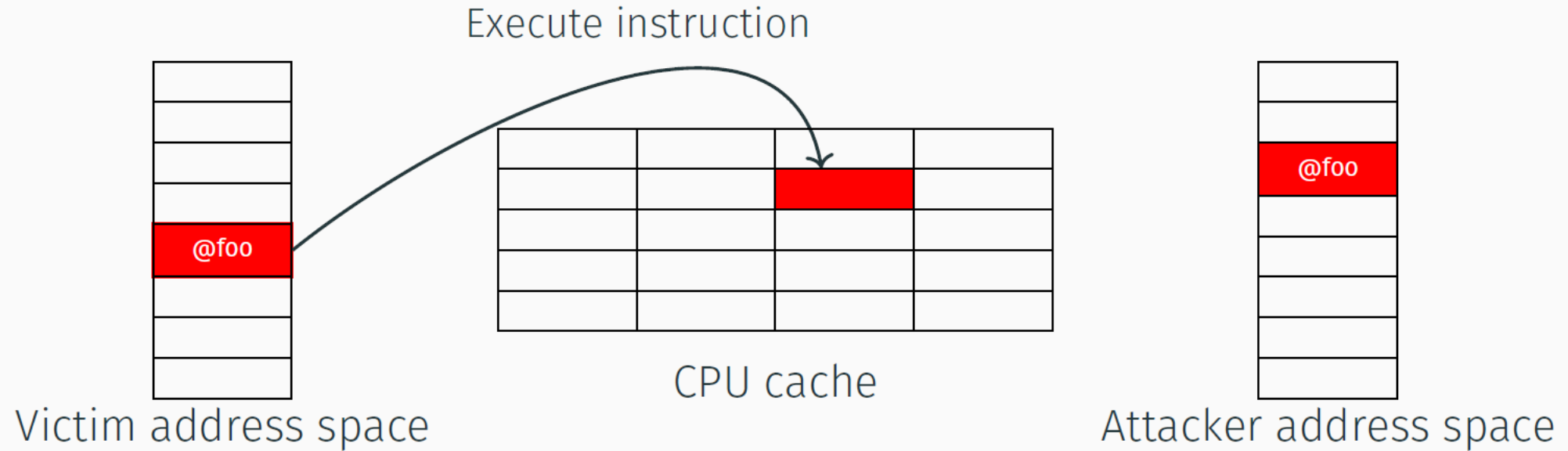
FLUSH+RELOAD¹



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

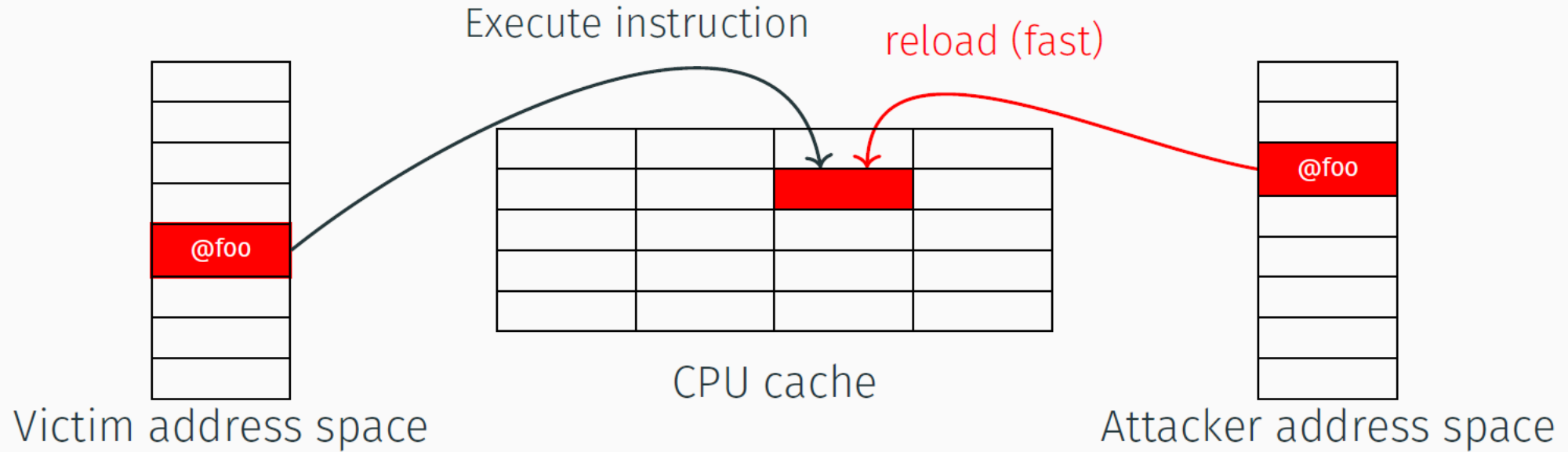
FLUSH+RELOAD¹



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

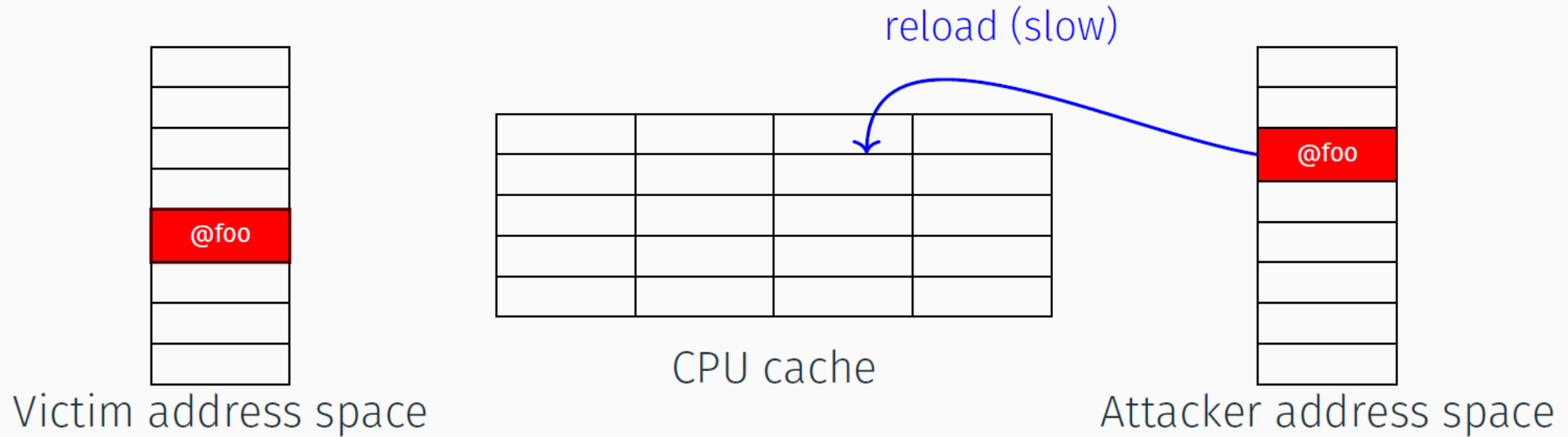
FLUSH+RELOAD¹



1. Maps the victim's address space
2. Flush the instruction we monitor
3. See how much time it takes to reload
 - Fast \Rightarrow 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+RELOAD¹



1. Maps the victim's address space
2. Flush the instruction we monitor
3. See how much time it takes to reload
 - Fast \Rightarrow the victim already executed
 - Slow \Rightarrow 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+Reload¹ and PDA²

A **cache-attack** that lets 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+Reload¹ and PDA²

Weak exponentiation algorithm

A **cache-attack** that lets 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+Reload¹ and PDA²

Weak exponentiation algorithm

A **cache-attack** that lets 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+Reload¹ and PDA²

Weak exponentiation algorithm

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

Passive offline attack

No error and enough information

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

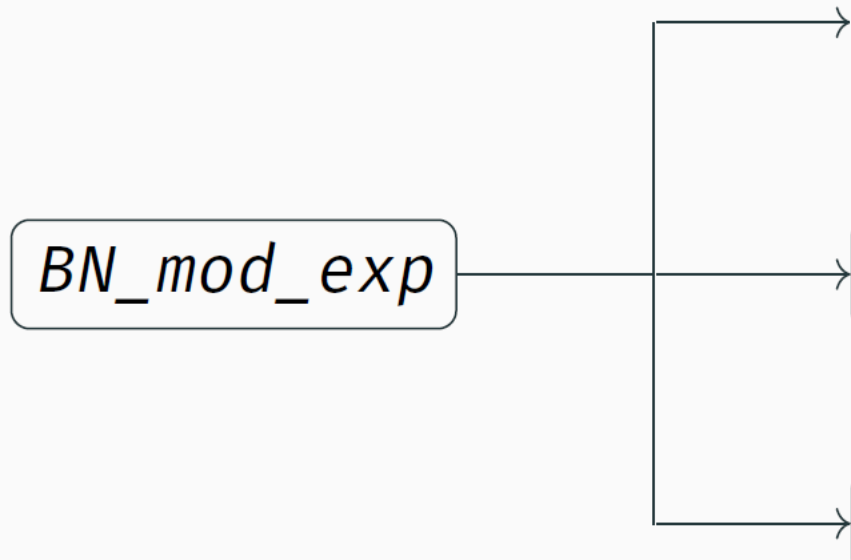
² T. Allan et al. *Amplifying side channels through performance degradation*. In ACSAC. 2016

The Vulnerability

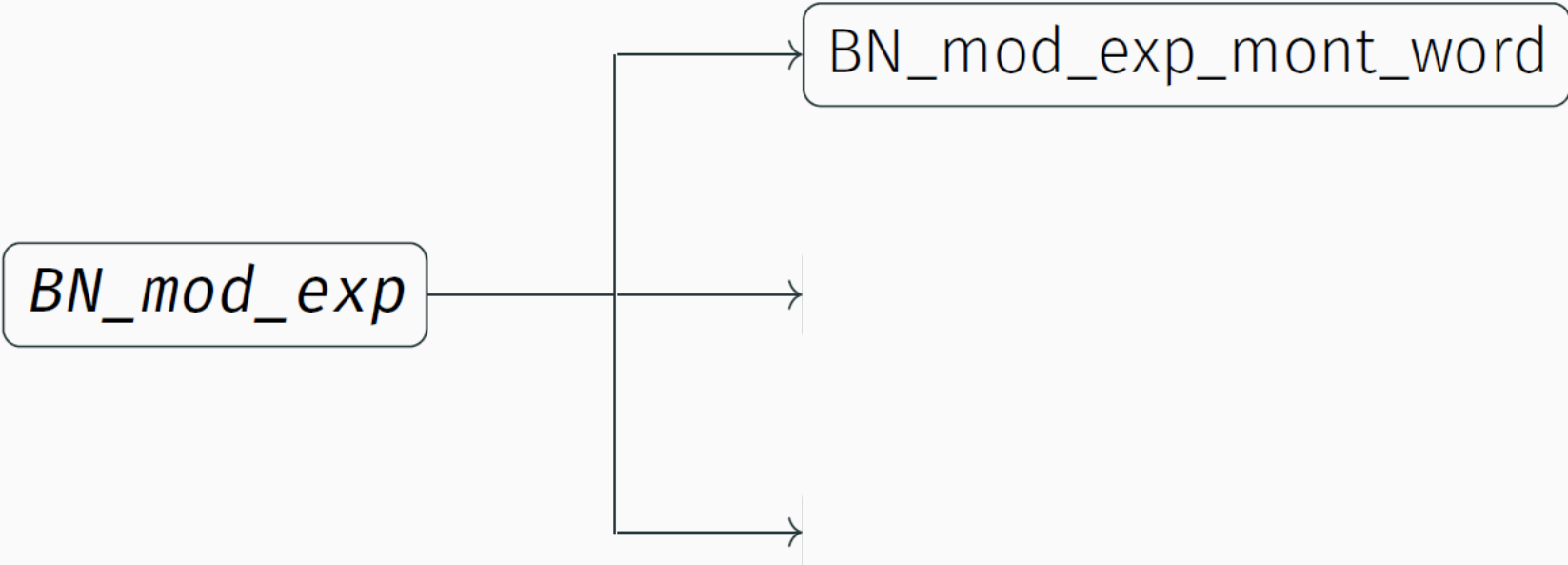
Modular exponentiation in OpenSSL

BN_mod_exp

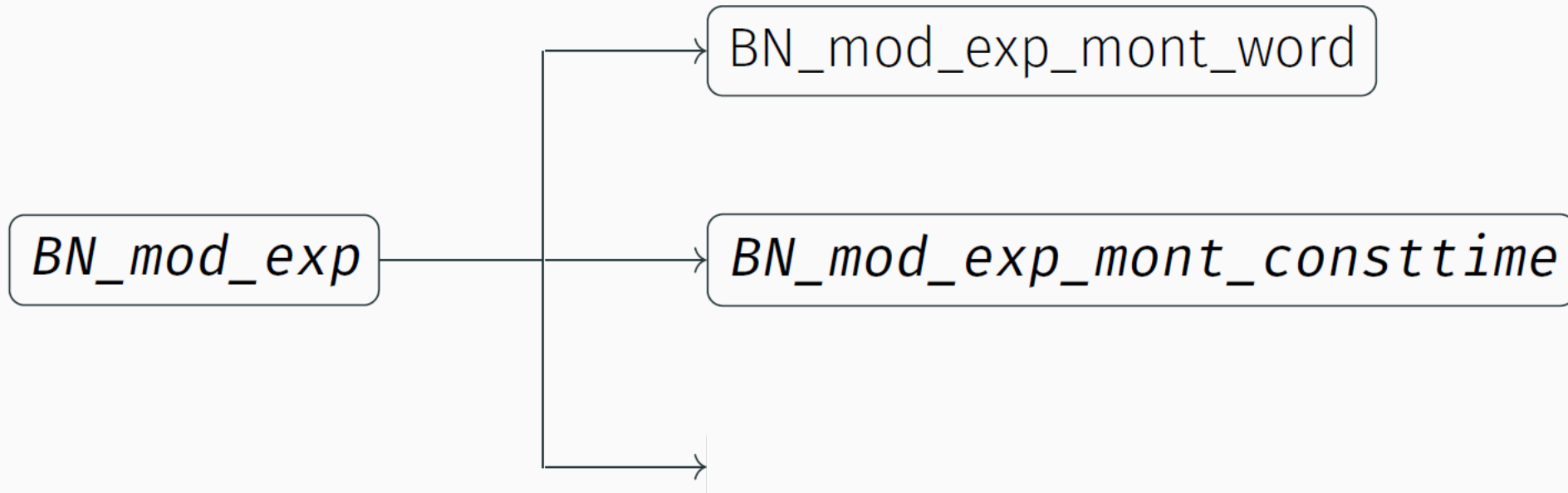
Modular exponentiation in OpenSSL



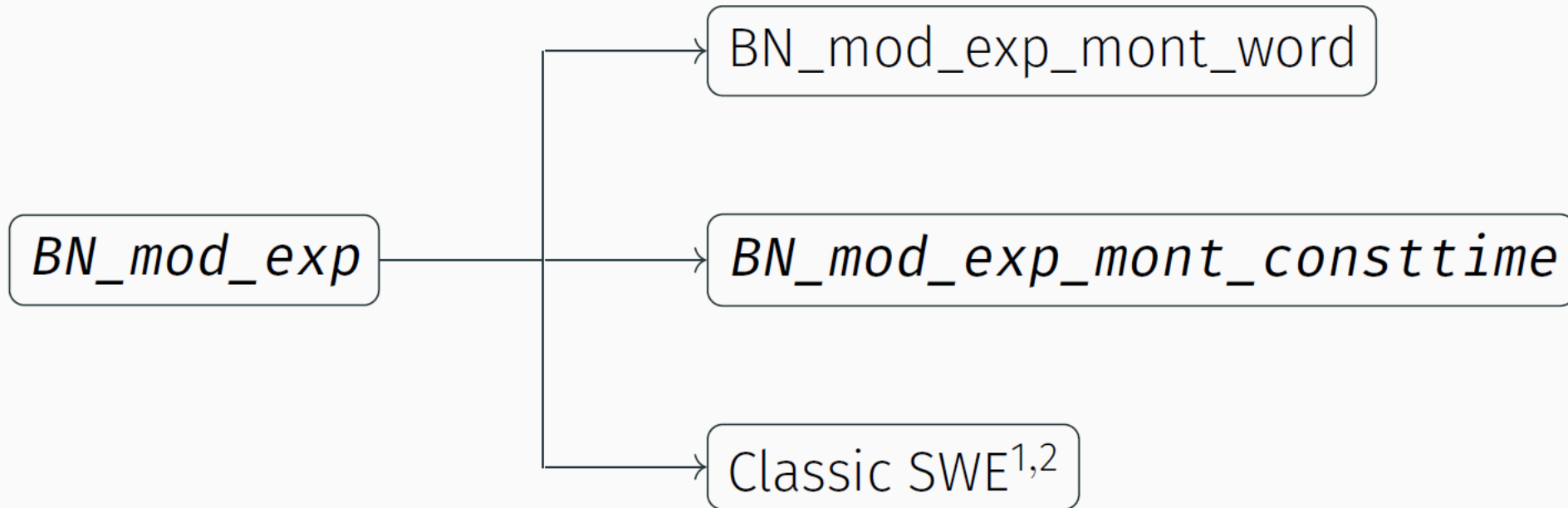
Modular exponentiation in OpenSSL



Modular exponentiation in OpenSSL



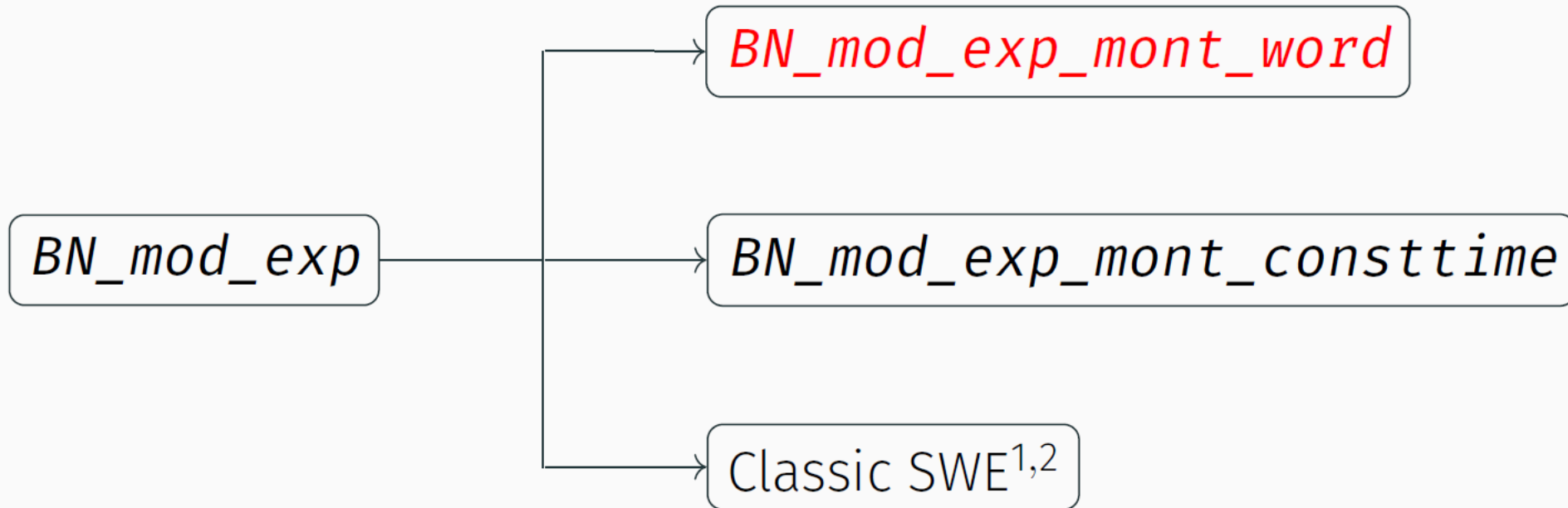
Modular exponentiation in OpenSSL



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

$$\text{bin}(e) = 1\ 1\ 0\ 1\ 0\ \dots$$

$$\text{res} = g^e \bmod p$$

w is a processor word (e.g. 64 bits)

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

Optimized Square-and-Multiply

$$\text{bin}(e) = 1\ 1\ 0\ 1\ 0\ \dots$$

$$\text{res} = g^e \bmod p$$

w is a processor word (e.g. 64 bits)



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

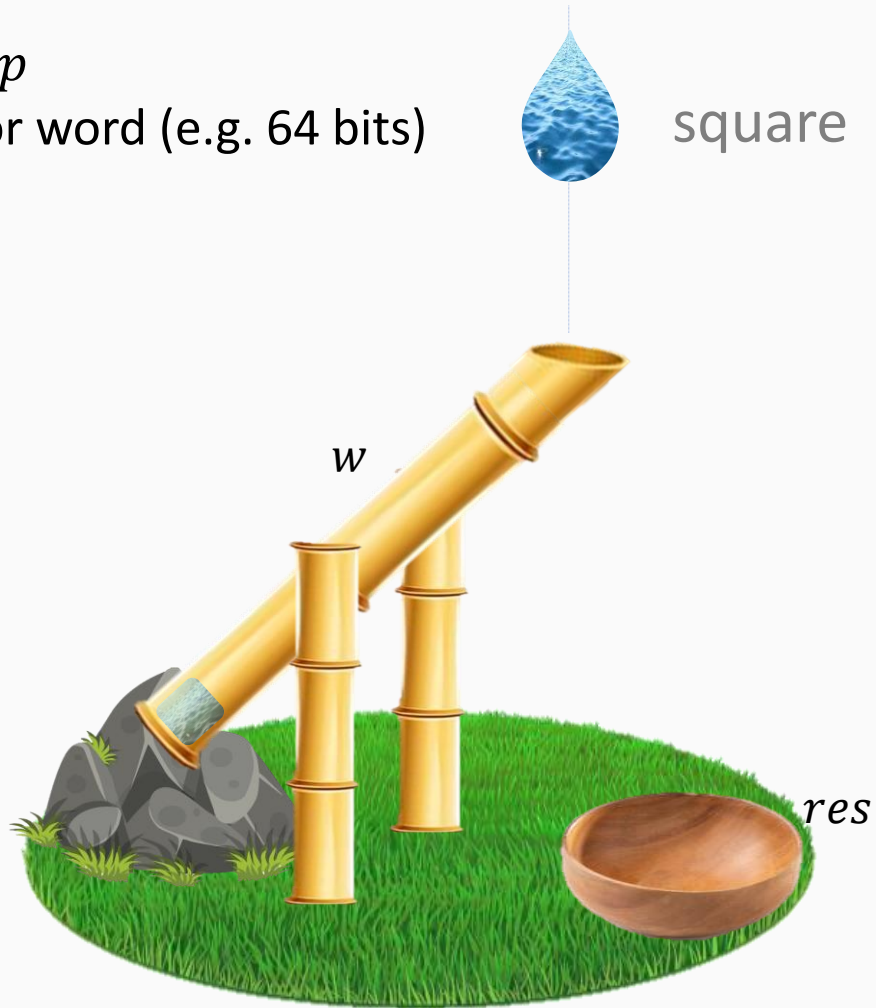
...

Optimized Square-and-Multiply

$\text{bin}(e) = 1\ 1\ 0\ 1\ 0\ \dots$

$\text{res} = g^e \bmod p$

w is a processor word (e.g. 64 bits)



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

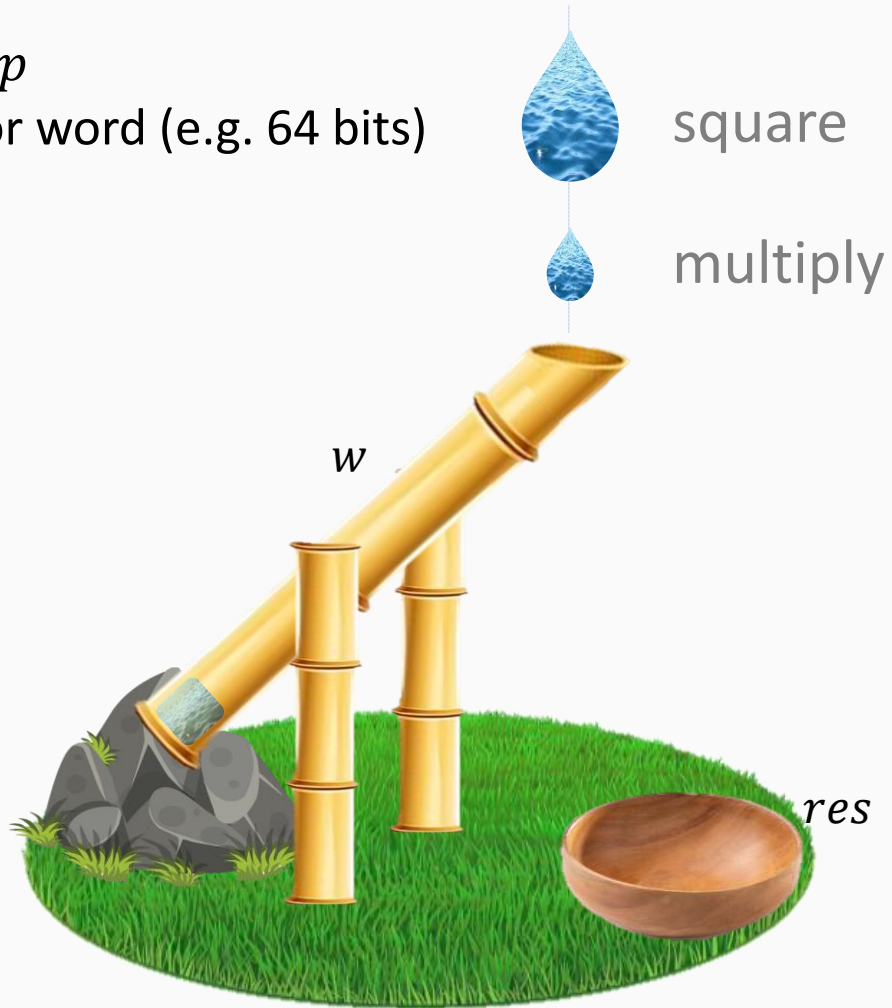
...

Optimized Square-and-Multiply

$\text{bin}(e) = 1\ 1\ 0\ 1\ 0\ \dots$

$\text{res} = g^e \bmod p$

w is a processor word (e.g. 64 bits)



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

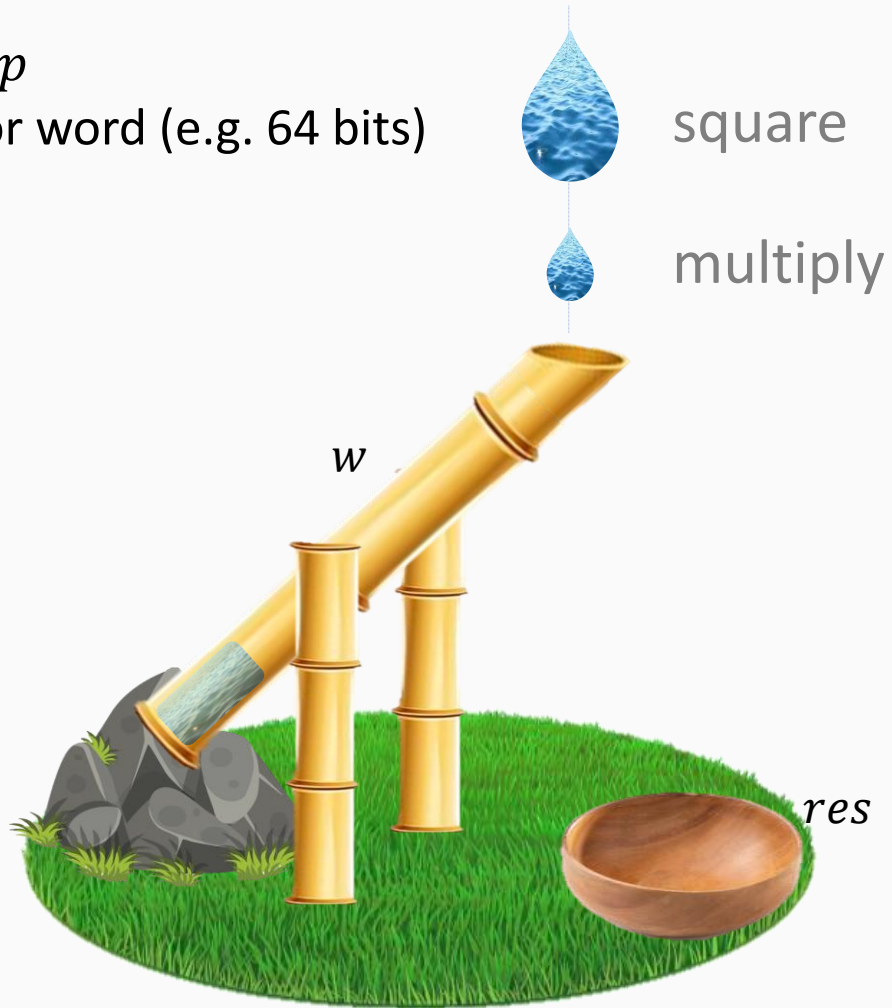
...

Optimized Square-and-Multiply

$\text{bin}(e) = 1\ 1\ 0\ 1\ 0\ \dots$

$\text{res} = g^e \bmod p$

w is a processor word (e.g. 64 bits)



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

...

Optimized Square-and-Multiply

$\text{bin}(e) = 1\ 1\ 0\ 1\ 0\ \dots$

$\text{res} = g^e \bmod p$

w is a processor word (e.g. 64 bits)



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

...

Optimized Square-and-Multiply

$\text{bin}(e) = 1\ 1\ 0\ 1\ 0\ \dots$

$\text{res} = g^e \bmod p$

w is a processor word (e.g. 64 bits)



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

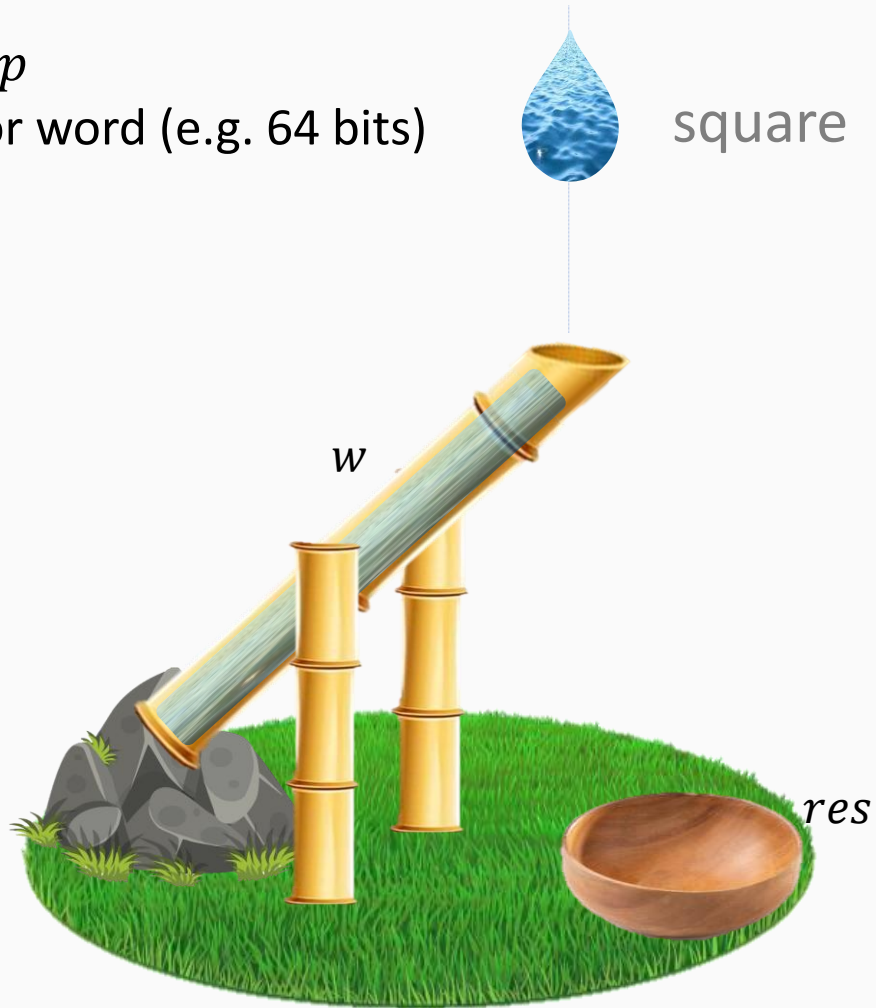
...

Optimized Square-and-Multiply

$\text{bin}(e) = 1\ 1\ 0\ 1\ 0\ \dots$

$\text{res} = g^e \bmod p$

w is a processor word (e.g. 64 bits)



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

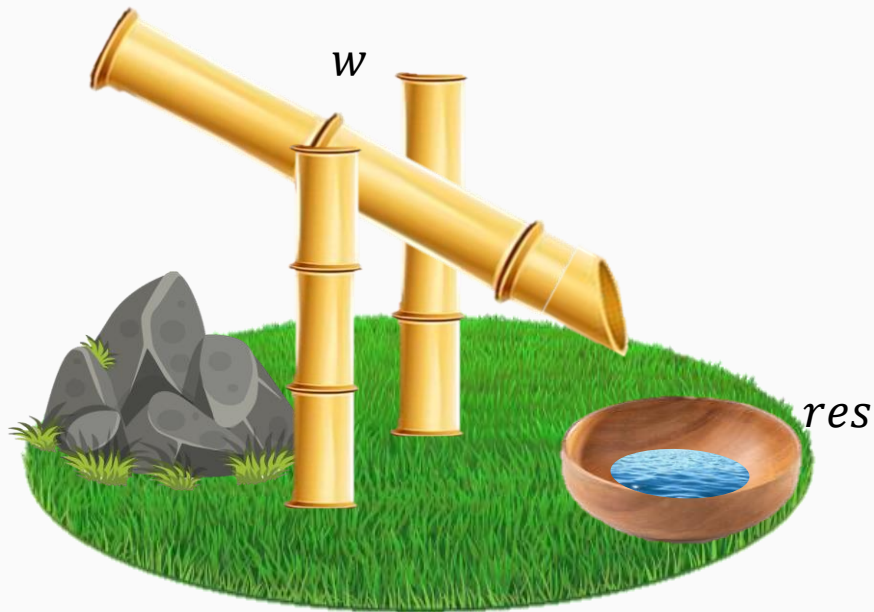
...

Optimized Square-and-Multiply

$\text{bin}(e) = 1\ 1\ 0\ 1\ 0\ \dots$

$\text{res} = g^e \bmod p$

w is a processor word (e.g. 64 bits)



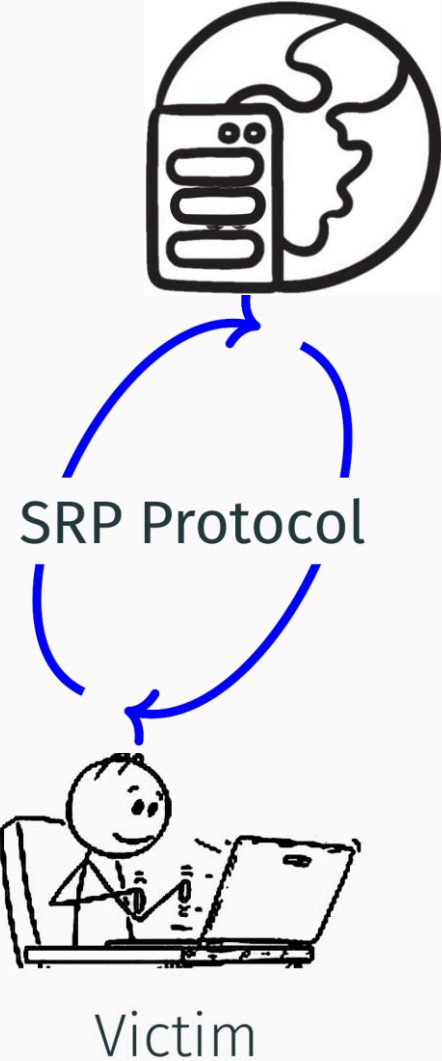
```
def BN_mod_exp_mont_word(g, w, p):  
    ...  
    w = g # uint64_t  
    res = BN_to_mont_word(w) # bigum  
    for b in range(bitlen-2, 0, -1):  
        next_w = w x w  
        if next_w/w != w:  
            → res = BN_mod_mul(res, w, p)  
            next_w = 1  
        w = next_w  
     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

...

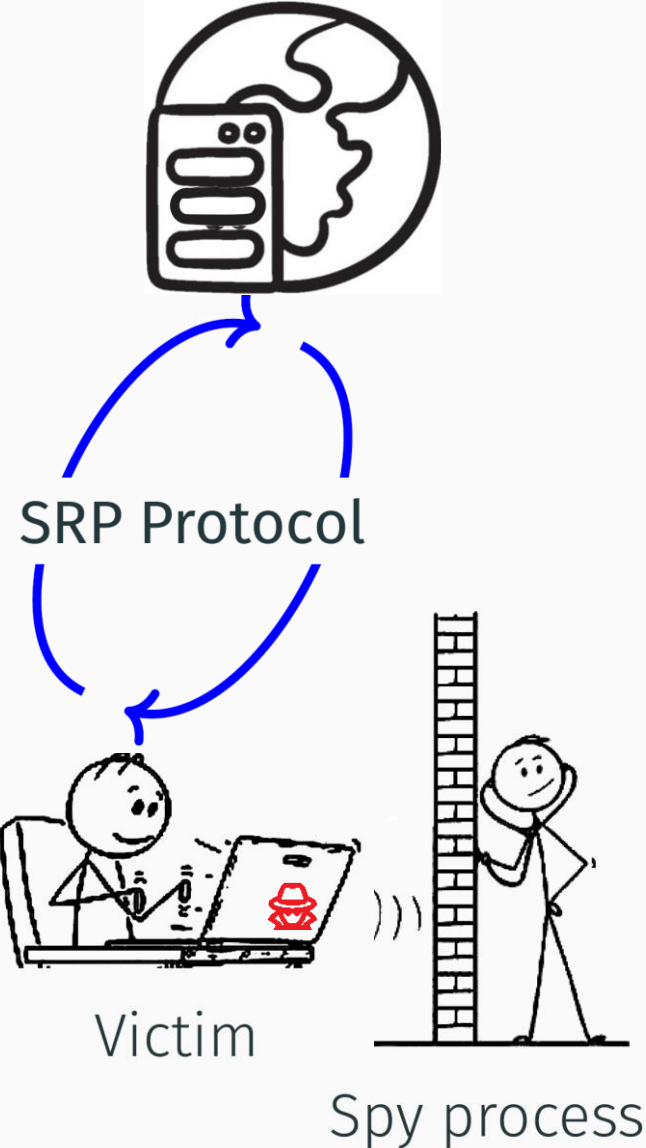
Exploiting the Leakage

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

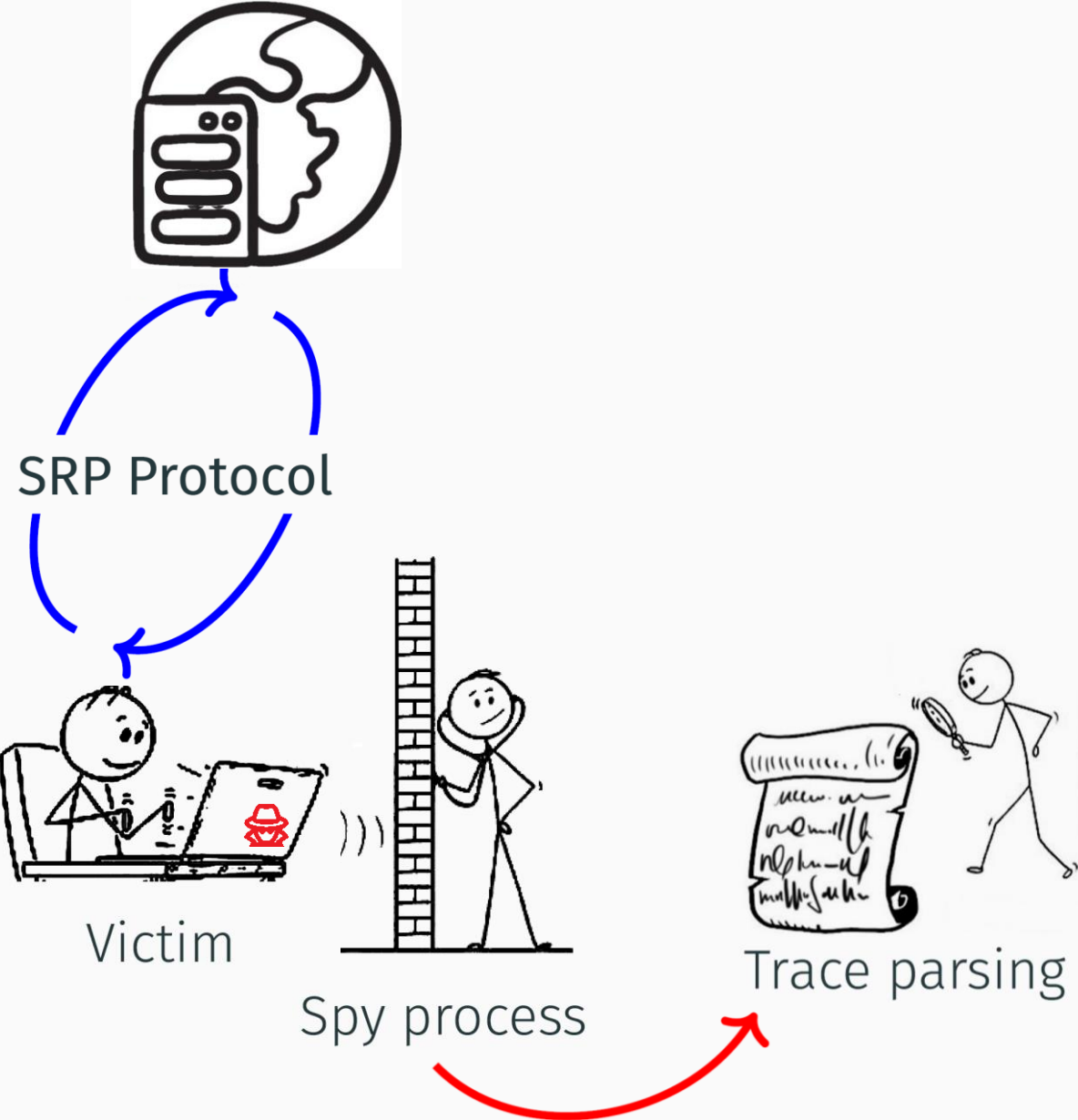
Classical Workflow



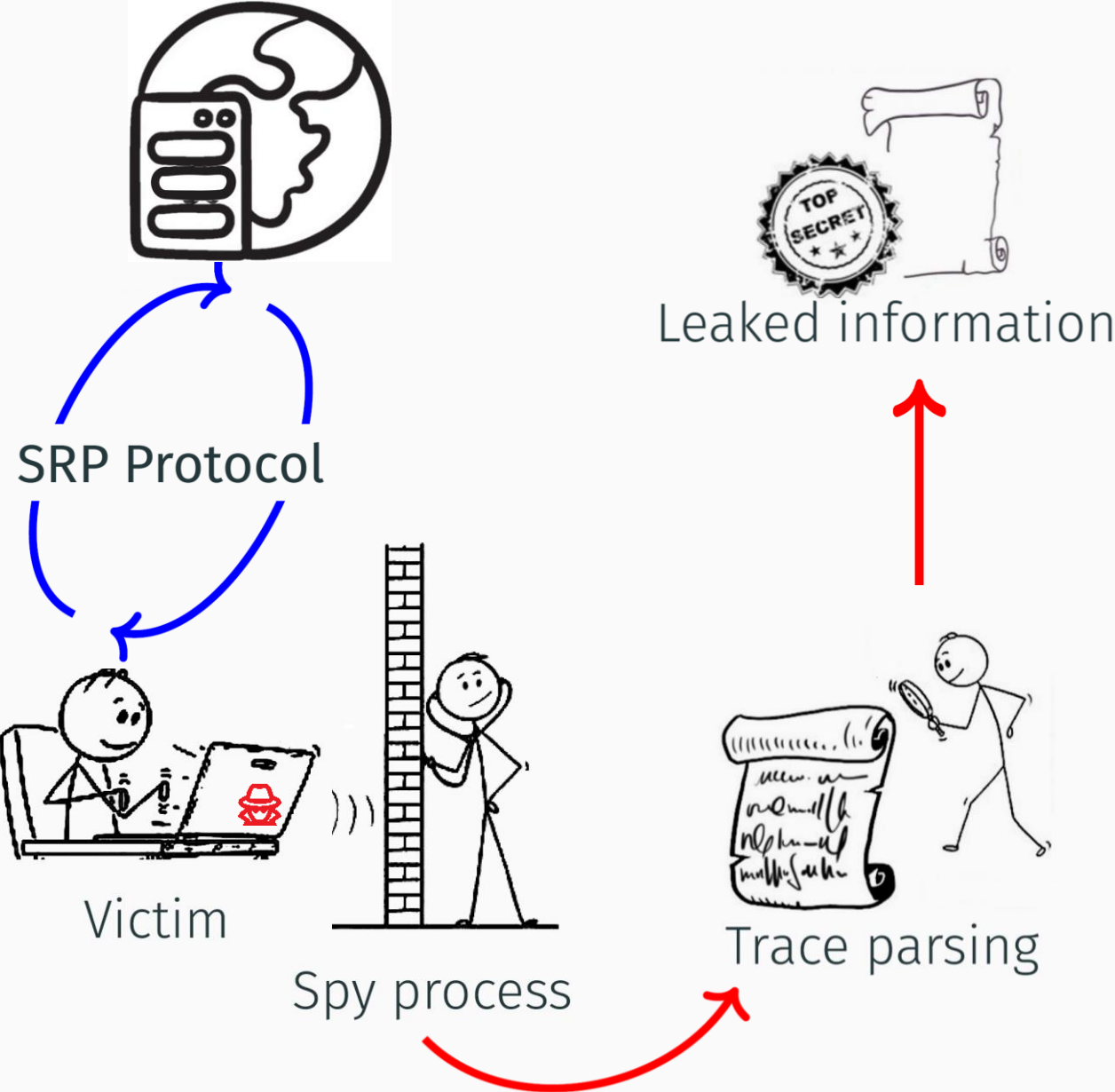
Classical Workflow



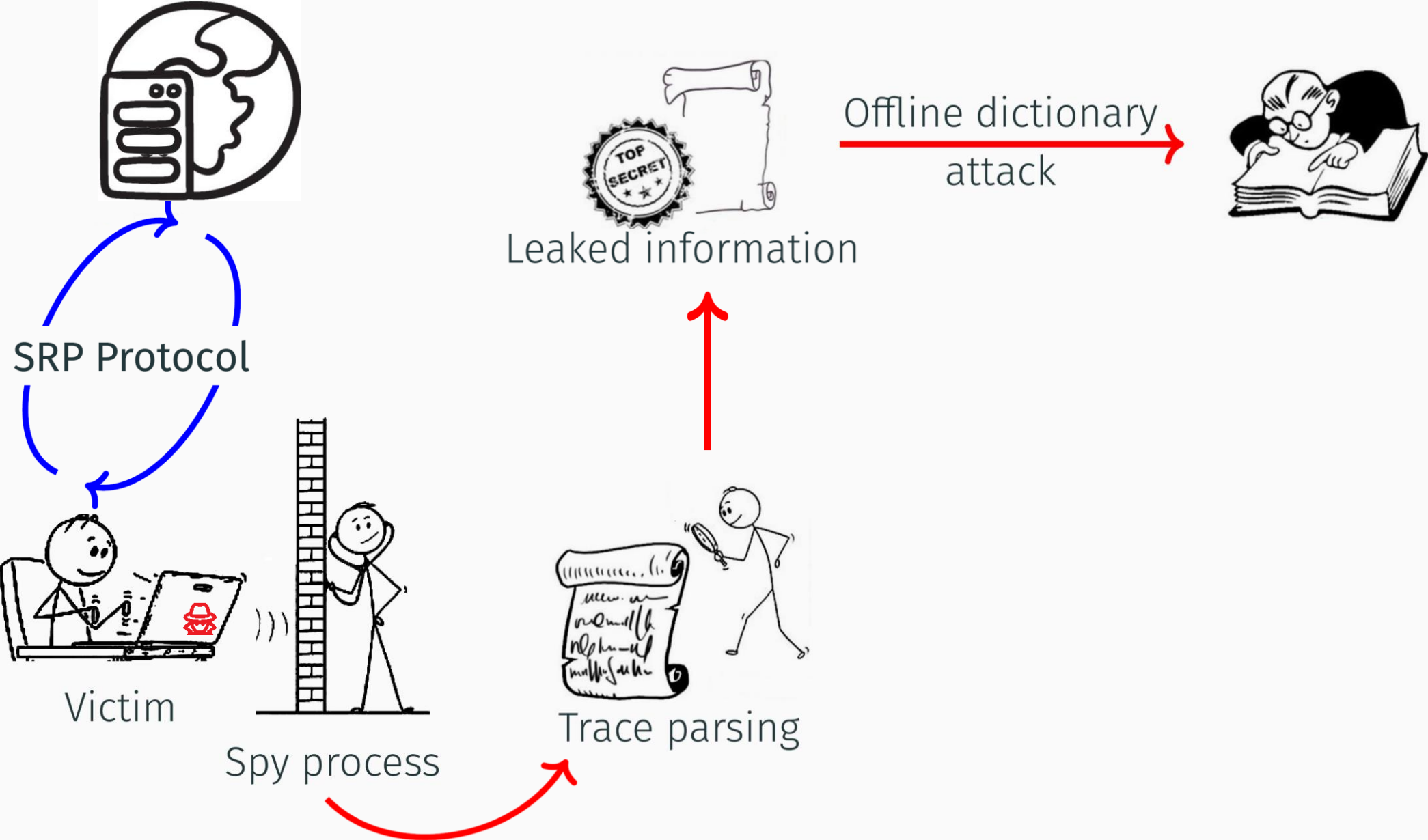
Classical Workflow



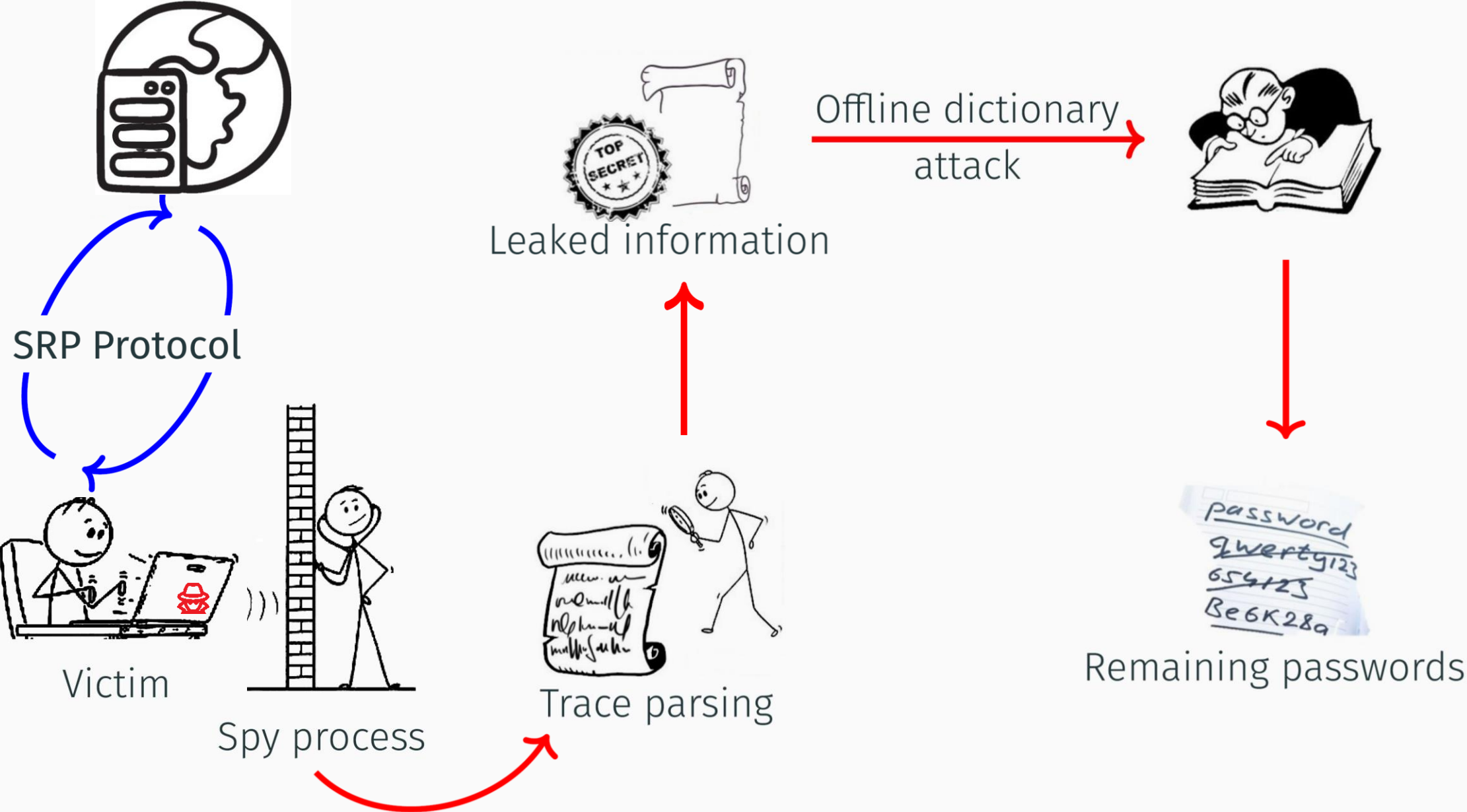
Classical Workflow



Classical Workflow



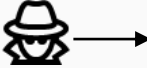
Classical Workflow



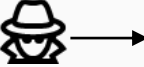
Trace Acquisition

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

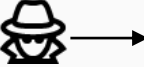
Trace Acquisition

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

Trace Acquisition

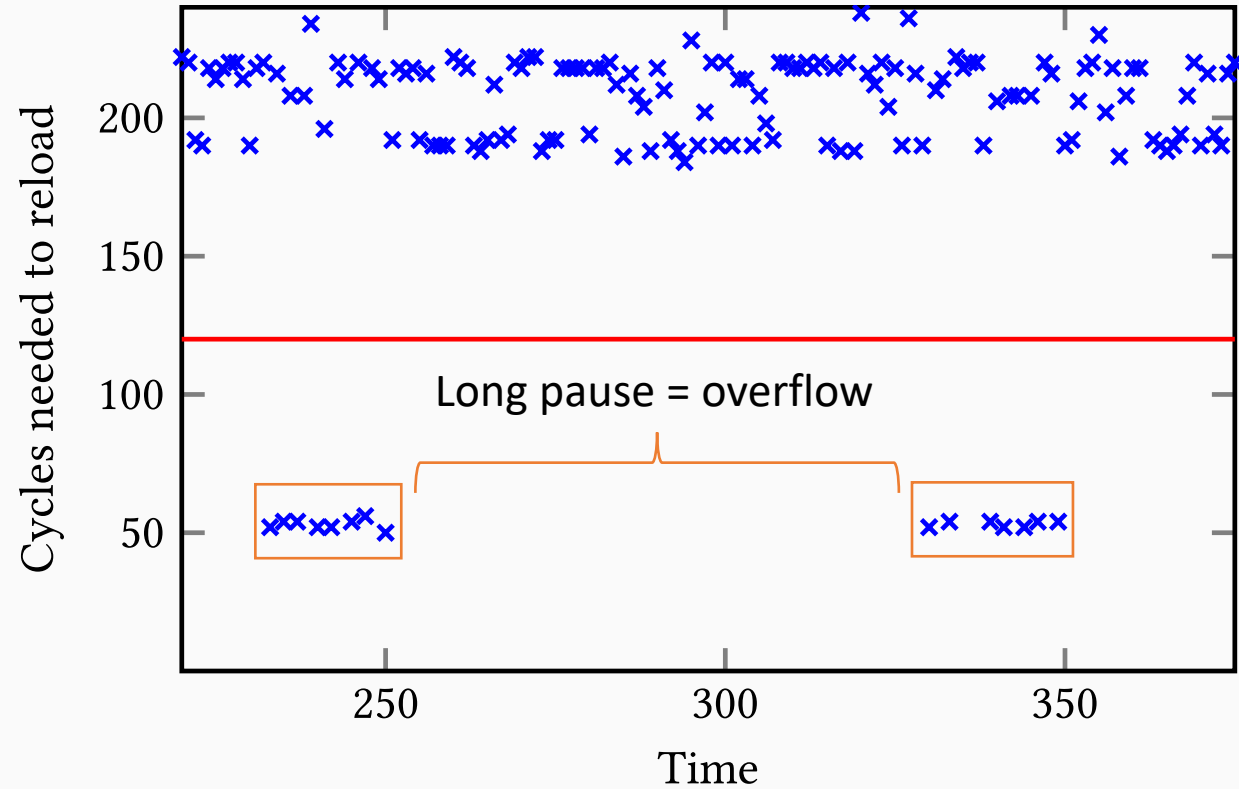
```
def BN_mod_exp_mont_word(g, w, p):  
    ...  
    w = g # uint64_t  
    res = BN_to_mont_word(w) # bigum  
    for b in range(bitlen-2, 0, -1):  
        next_w = w x w  
        if next_w/w != w:  
            res = BN_mod_mul(res, w, p)  
            next_w = 1  
        w = next_w  
     res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

Trace Acquisition

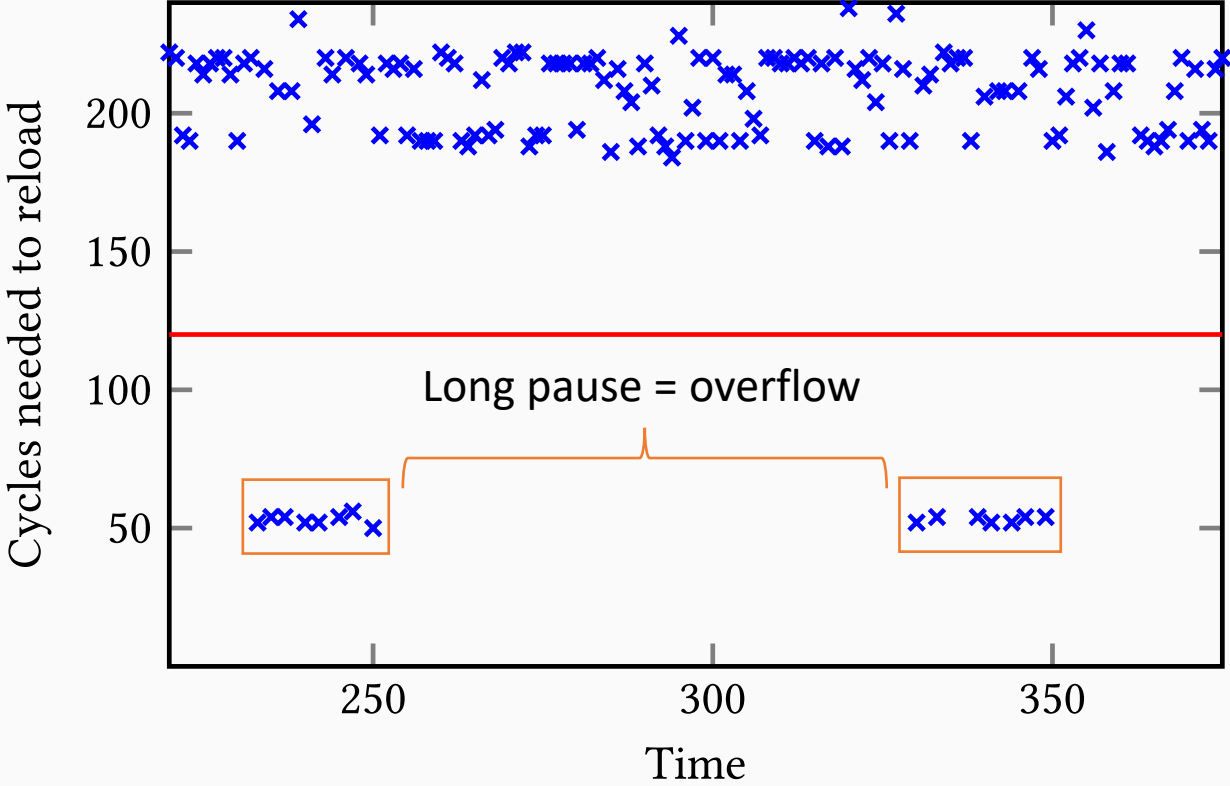
```
def BN_mod_exp_mont_word(g, w, p):  
    ...  
    w = g # uint64_t  
    res = BN_to_mont_word(w) # bigum  
    for b in range(bitlen-2, 0, -1):  
        next_w = w x w  
        if next_w/w != w:  
             → res = BN_mod_mul(res, w, p)  
            next_w = 1  
        w = next_w  
    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

Trace Acquisition

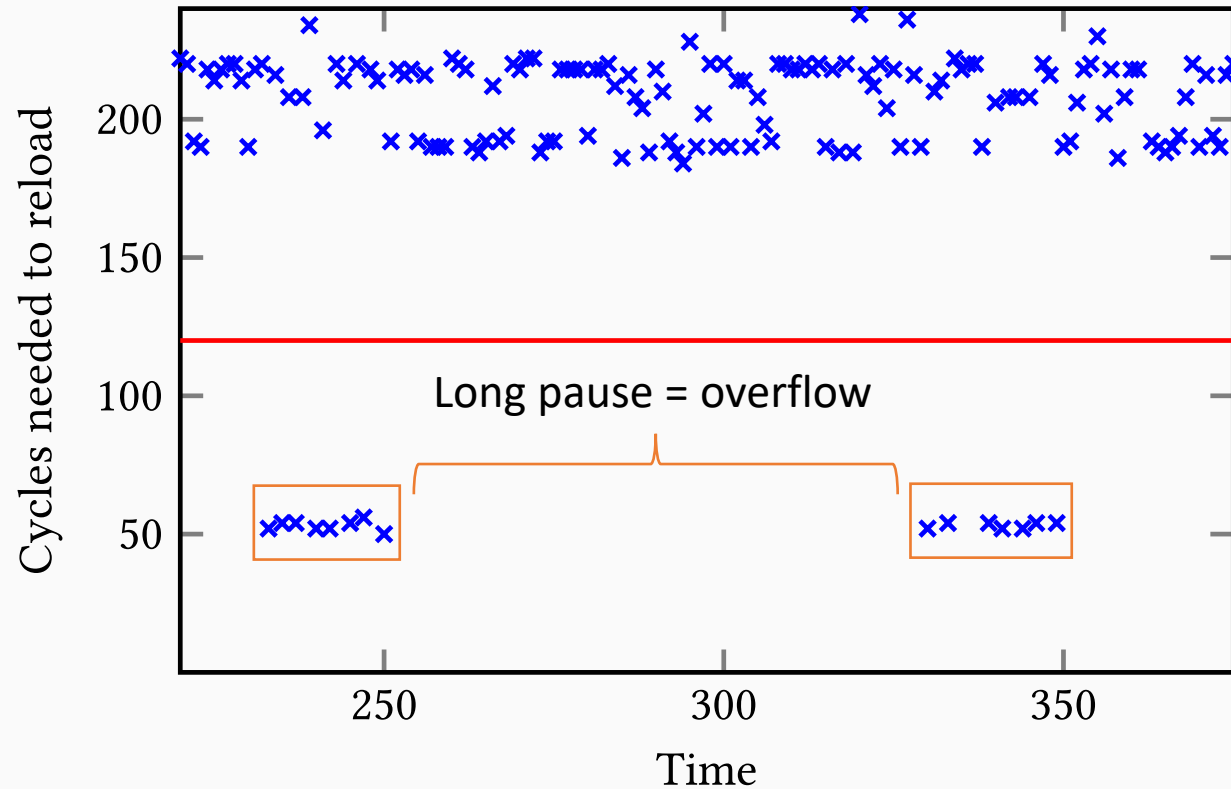
```
def BN_mod_exp_mot_word(g, w, p):  
    ...  
    w = g # uint64_t  
    res = BN_to_mont_word(w) # bigum  
    for b in range(bitlen-2, 0, -1):  
        next_w = w x w  
        if next_w/w != w:  
            ↪ res = BN_mod_mul(res, w, p) # snail  
            next_w = 1  
        w = next_w  
        ↪ res = BN_sqr(res) # hat  
        if BN_is_bit_set(x, b):  
            next_w = w x g  
            if next_w/g != w:  
                res = BN_mod_mul(res, w, p)  
            next_w = g  
        w = next_w
```



Trace Interpretation



Trace Interpretation



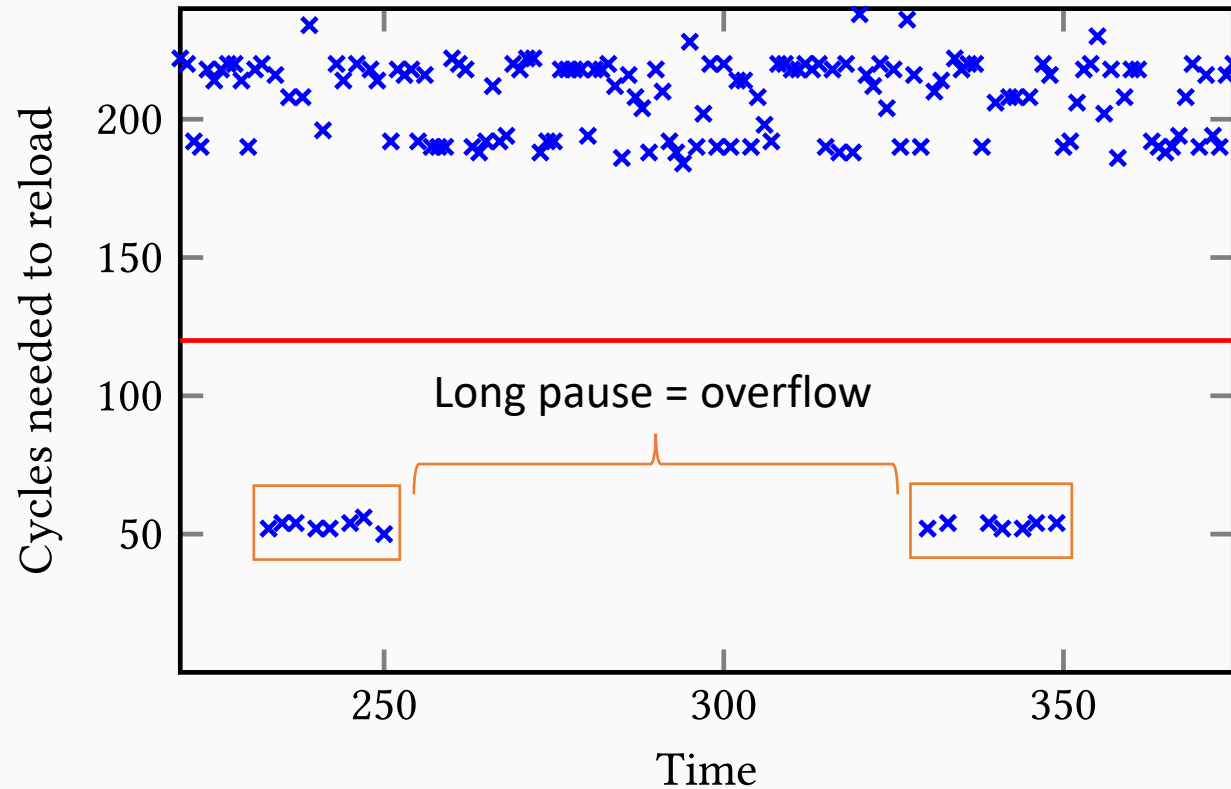
Rules ($b \in \{0,1\}$):

$Vvvv \rightarrow 111b$

$Vvvvv \rightarrow yyyyb, yyy \in \{110b, 10bb, 0111\}$

$Vv\dots v \rightarrow 0 \dots 0yyyyb$

Trace Interpretation



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

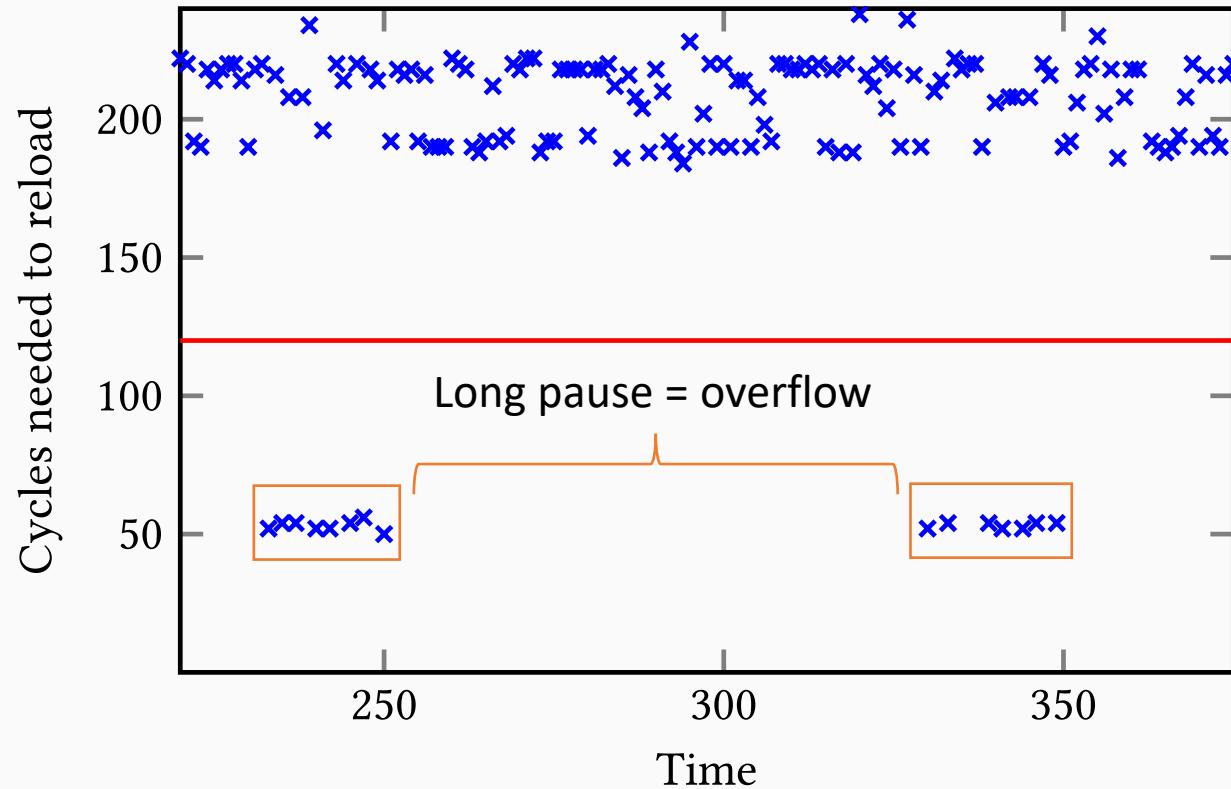
$Vvvv \rightarrow 111b$

$Vvvvv \rightarrow yyyyb, yyy \in \{110b, 10bb, 0111\}$

$Vv\dots v \rightarrow 0 \dots 0yyyyb$

$Vvvv \ Vvvvv \ Vvvvvv \ Vvvvv \ Vvvvvv \ Vvvvvv \ Vvvv$

Trace Interpretation



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

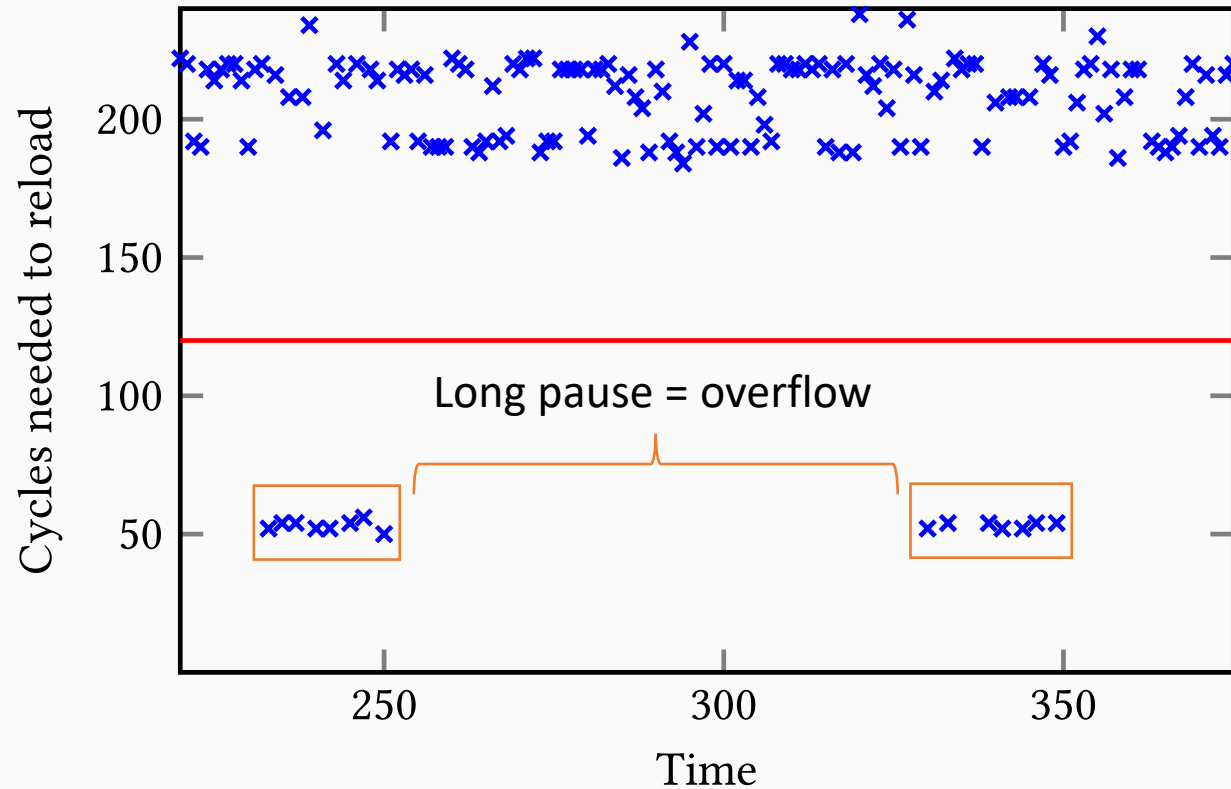
$Vvvv \rightarrow 111b$

$Vvvvv \rightarrow yyyyb, yyy \in \{110b, 10bb, 0111\}$

$Vv\dots v \rightarrow 0 \dots 0yyyb$

$Vvvv \ Vvvvv \ Vvvvvv \ Vvvvv \ Vvvvvv \ Vvvvvv \ Vvvv$
 $111b$

Trace Interpretation



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

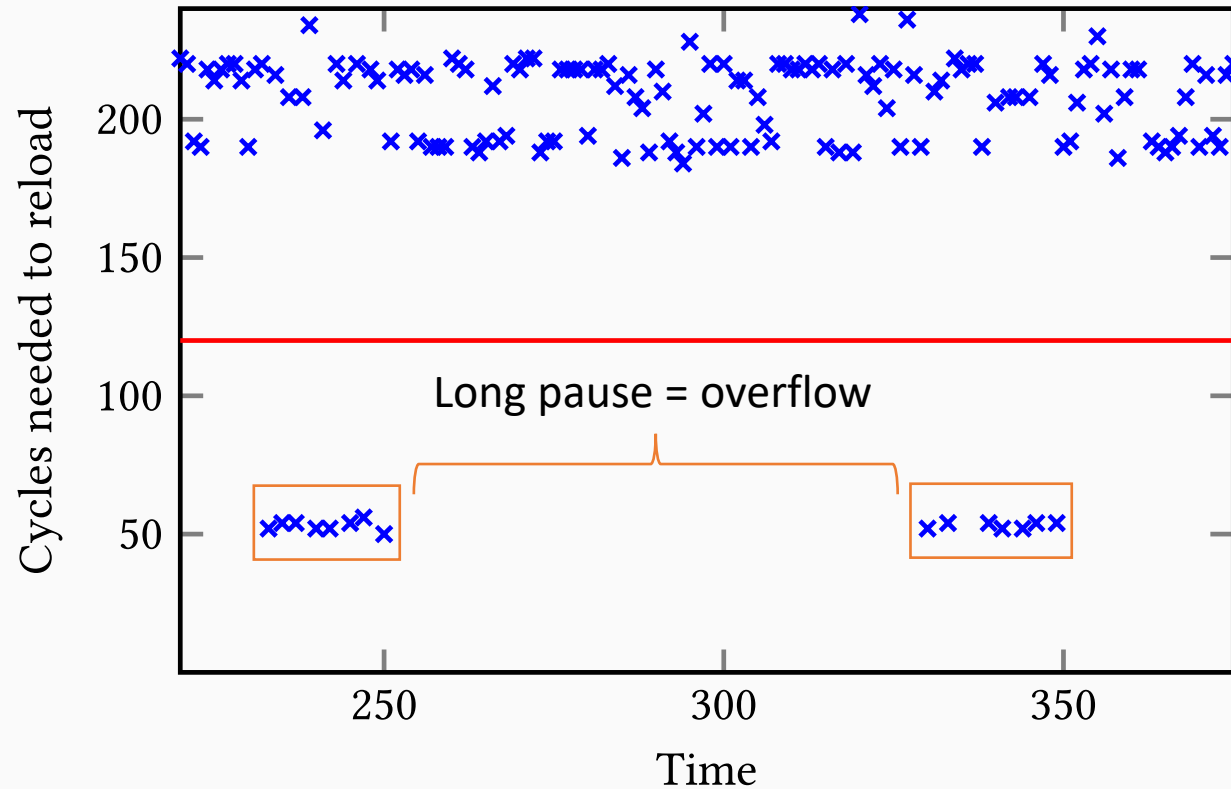
$Vvvv \rightarrow 111b$

$Vvvvv \rightarrow yyyyb, yyy \in \{110b, 10bb, 0111\}$

$Vv\dots v \rightarrow 0 \dots 0yyyyb$

$Vvvv \ Vvvvv \ Vvvvvv \ Vvvvv \ Vvvvv \ Vvvvv \ Vvvv$
 $111b \ yyyyb$

Trace Interpretation



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

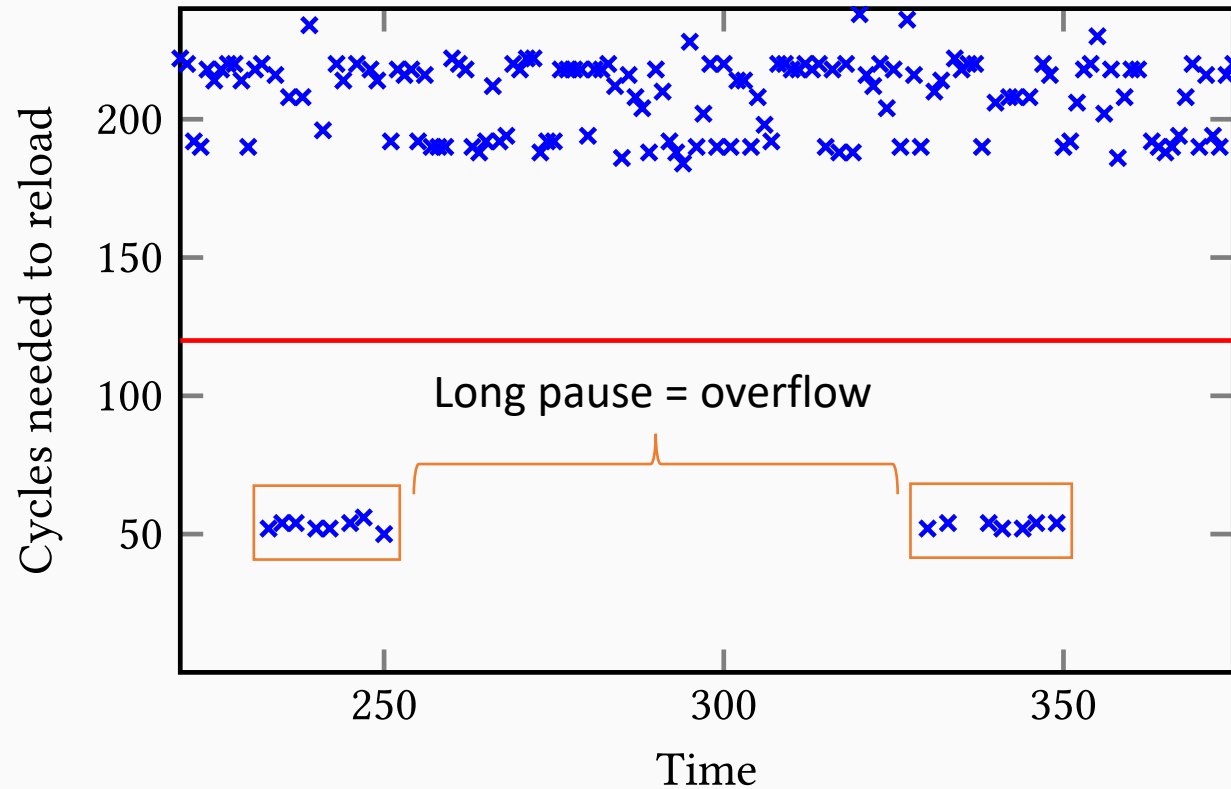
$Vvvv \rightarrow 111b$

$Vvvvv \rightarrow yyyyb, yyy \in \{110b, 10bb, 0111\}$

$Vv\dots v \rightarrow 0 \dots 0yyyyyb$

$Vvvv \ Vvvvv \ Vvvvvv \ Vvvvv \ Vvvvvv \ Vvvvvv \ Vvvv$
 $111b \ yyyyb \ 0yyyyyb$

Trace Interpretation



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

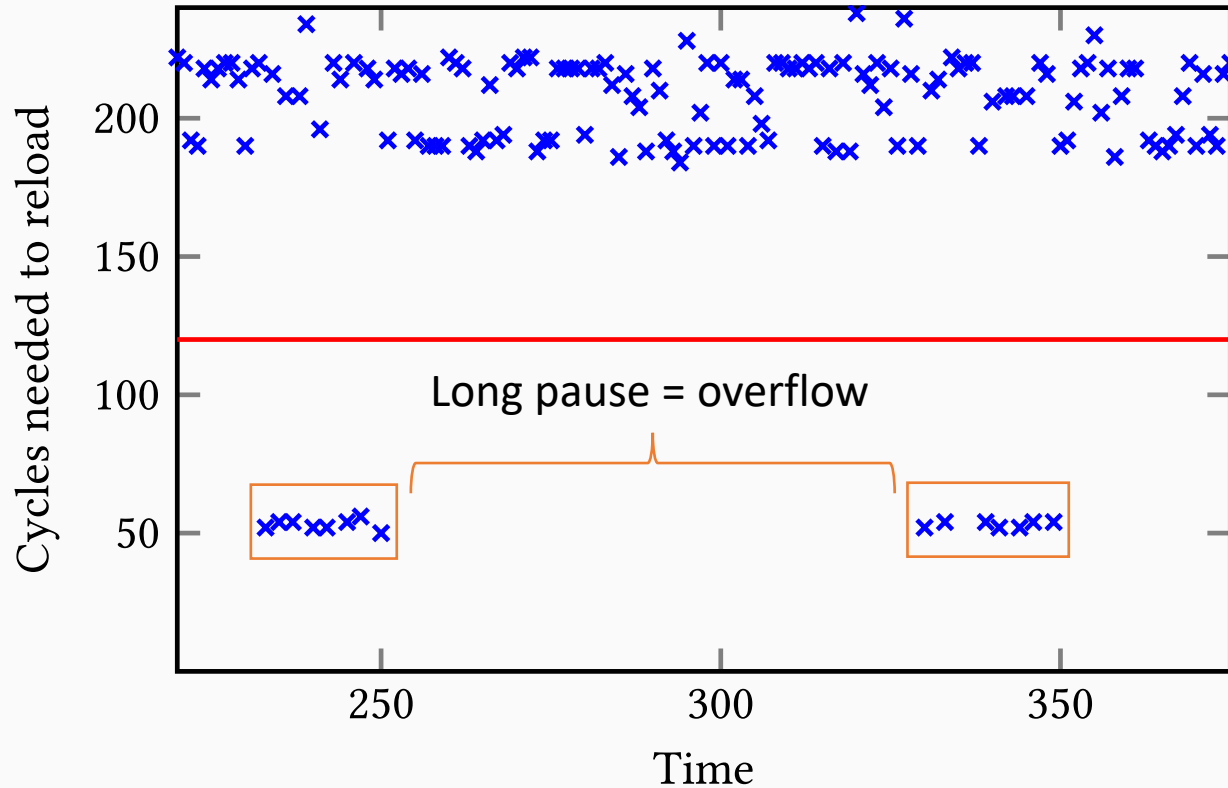
$Vvvv \rightarrow 111b$

$Vvvvv \rightarrow yyyyb, yyy \in \{110b, 10bb, 0111\}$

$Vv\dots v \rightarrow 0 \dots 0yyyyyb$

$Vvvv \ Vvvvv \ Vvvvvv \ Vvvvv \ Vvvvvv \ Vvvvvv \ Vvvv$
 $111b \ yyyyb \ 0yyyyyb \ yyyyb$

Trace Interpretation



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

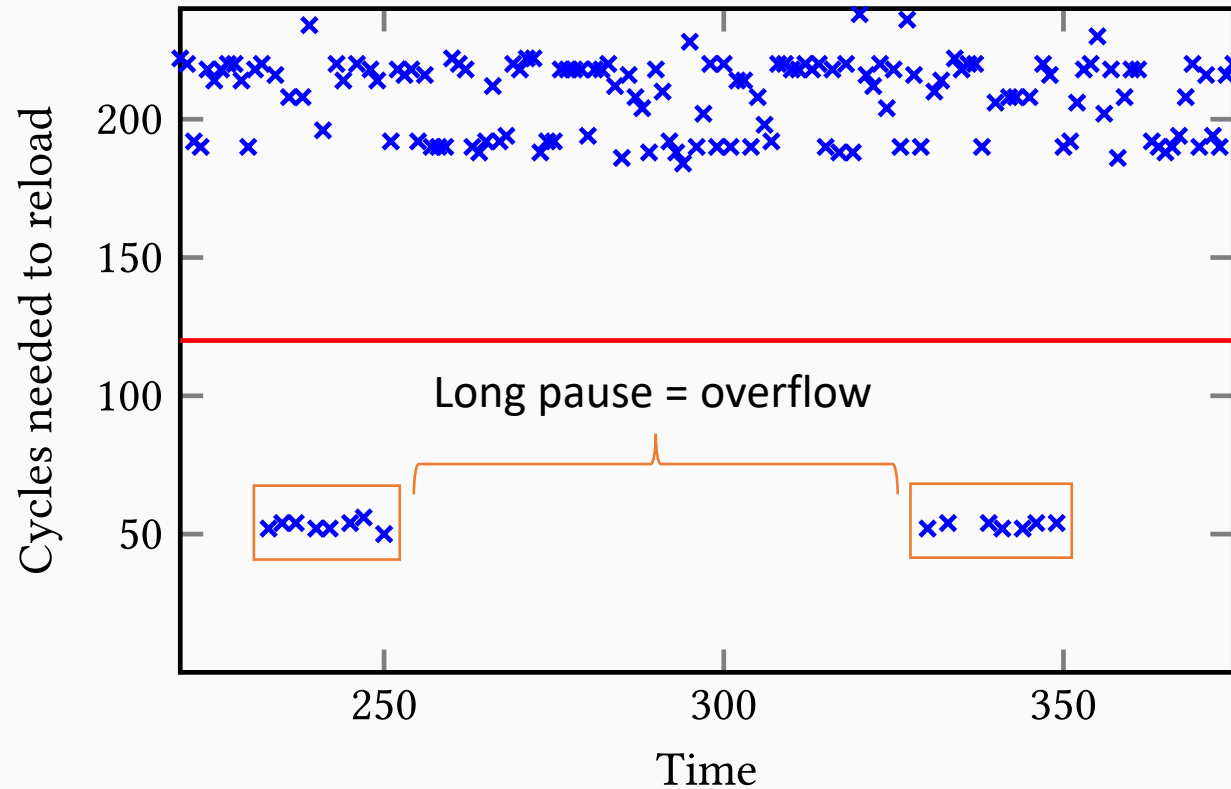
$Vvvv \rightarrow 111b$

$Vvvvv \rightarrow yyyyb, yyyy \in \{110b, 10bb, 0111\}$

$Vv\dots v \rightarrow 0 \dots 0yyyyyb$

$Vvvv \ Vvvvv \ Vvvvvv \ Vvvvv \ Vvvvvv \ Vvvvvv \ Vvvv$
 $111b \ yyyyb \ 0yyyyyb \ yyyyb \ yyyyb$

Trace Interpretation



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

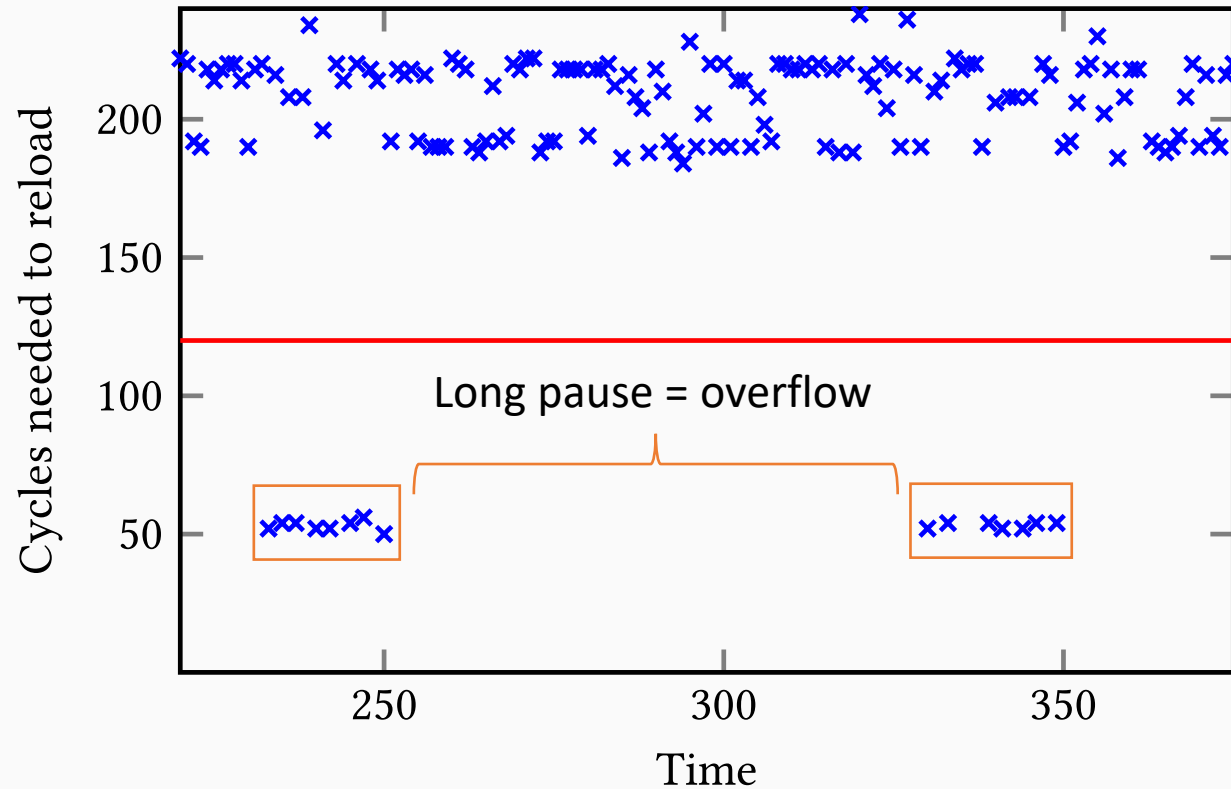
$Vvvv \rightarrow 111b$

$Vvvvv \rightarrow yyyyb, yyy \in \{110b, 10bb, 0111\}$

$Vv\dots v \rightarrow 0 \dots 0yyyyyb$

$Vvvv \ Vvvvv \ Vvvvvv \ Vvvvv \ Vvvvv \ Vvvvv \ Vvvv$
 $111b \ yyyyb \ 0yyyyyb \ yyyyb \ yyyyb \ yyyyb$

Trace Interpretation



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

$Vvvv \rightarrow 111b$

$Vvvvv \rightarrow yyyyb, yyy \in \{110b, 10bb, 0111\}$

$Vv\dots v \rightarrow 0 \dots 0yyyyb$

$Vvvv \ Vvvvv \ Vvvvvv \ Vvvvv \ Vvvvvv \ Vvvvvv \ Vvvv$
 $111b \ yyyyb \ 0yyyyb \ yyyyb \ yyyyb \ yyyyb \ bbbb$

Dictionary Attack

Client : $x = H(\text{salt} \parallel H(\text{user_id}:\text{password}))$
 $v = g^x \text{ mod } p$

Dictionary Attack

Client : $x = H(\text{salt} || H(\text{user_id}:\text{password}))$
 $v = g^x \text{ mod } p$

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

$Xxxx \rightarrow 111b$

$Xxxxx \rightarrow yyyyb, y \in \{110b, 10bb, 0111\}$

$Xx.....x \rightarrow 0 \dots 0yyyyb$

Dictionary Attack

Client : $x = H(\text{salt} || H(\text{user_id}:\text{password}))$
 $v = g^x \text{ mod } p$

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

$Xxxx \rightarrow 111b$

$Xxxxx \rightarrow yyyyb, y \in \{110b, 10bb, 0111\}$

$Xx.....x \rightarrow 0 \dots 0yyyyb$

Recovered: 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 : $x = H(\text{salt} || H(\text{user_id}:\text{password}))$
 $v = g^x \text{ mod } p$

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

$Xxxx \rightarrow 111b$

$Xxxxx \rightarrow yyyyb, y \in \{110b, 10bb, 0111\}$

$Xx.....x \rightarrow 0 \dots 0yyyyb$

| | |
|------------|---|
| Recovered: | 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 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 : $x = H(\text{salt} || H(\text{user_id}:\text{password}))$
 $v = g^x \text{ mod } p$

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

$Xxxx \rightarrow 111b$

$Xxxxx \rightarrow yyyyb, y \in \{110b, 10bb, 0111\}$

$Xx.....x \rightarrow 0 \dots 0yyyyb$

Recovered: 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 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 : $x = H(\text{salt} || H(\text{user_id}:\text{password}))$
 $v = g^x \text{ mod } p$

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

$Xxxx \rightarrow 111b$

$Xxxxx \rightarrow yyyyb, y \in \{110b, 10bb, 0111\}$

$Xx.....x \rightarrow 0 \dots 0yyyyb$

Recovered: 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 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 : $x = H(\text{salt} || H(\text{user_id}:\text{password}))$
 $v = g^x \text{ mod } p$

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

Xxxx -> 111b

Xxxxx -> yyyyb, $y \in \{110b, 10bb, 0111\}$

Xx.....x -> 0 ... 0yyyyyb

| | | |
|------------|---|----|
| Recovered: | 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 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 $\Rightarrow 2^{-k}$ probability of false positive/negative

Single Measurement Attack

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

For a n bits 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
 - PySRP (used in ProtonMail python client)
 - GoToAssit (?)

- Lots of project using OpenSSL are impacted, including
 - OpenSSL TLS-SRP
 - Apple HomeKit ADK
 - PySRP (used in ProtonMail python client)
 - GoToAssit (?)



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

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

All SRP implementations using these packages / libraries would be affected!

Mitigations & Conclusion

Two choices:

- Patch this particular issue by adding the proper flag
 - Most projects use the bignum API, not the whole SRP
 - Difficult to propagate
 - Root cause 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

Two choices:

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

After OpenSSL, we contacted impacted to help with a patch:

- Apple HomeKit ADK
- node-bignum
- Ruby/openssl
- PySRP
- protonmail-python-client
- Erlang OTP

- 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

- 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