Hertzbleed: Turning Power Side-Channel Attacks Into Remote Timing Attacks on x86

Yingchen Wang*, Riccardo Paccagnella*, Elizabeth He,
Hovav Shacham, Christopher W. Fletcher, David Kohlbrenner

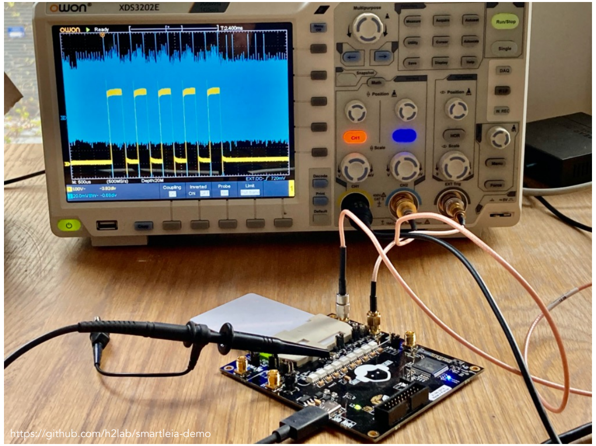
TEXAS The University of Texas at Austin
ILLINOIS
UNIVERSITY of WASHINGTON

(*co-first authors)

Power Side Channel vs Remote Timing

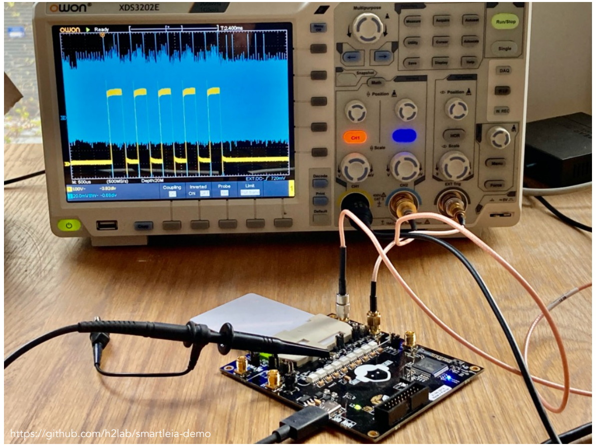
Power Side Channel vs Remote Timing

Power Side-Channel Attacks

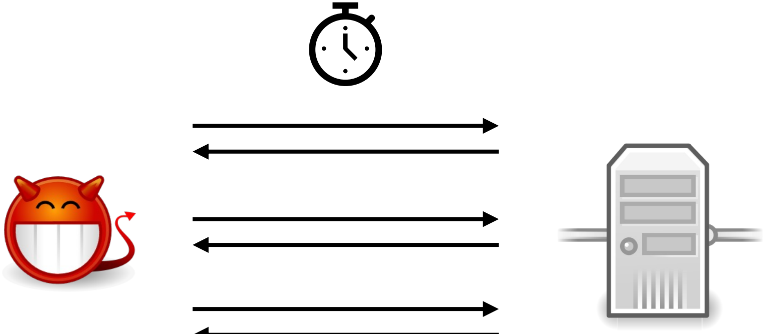


Power Side Channel vs Remote Timing

Power Side-Channel Attacks



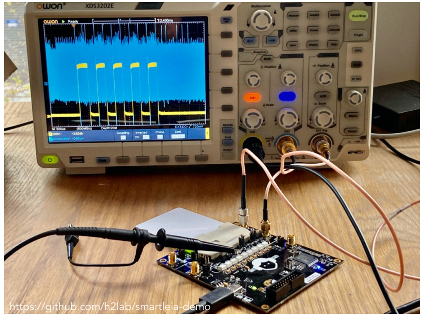
Remote Timing Attacks

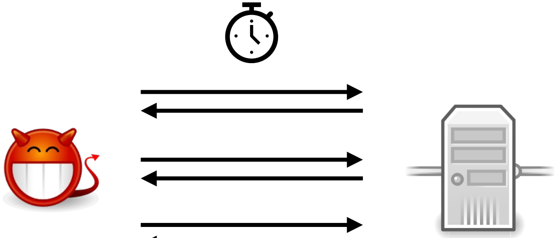


Hertzbleed: a New Class of Attacks

Power Side-Channel Attacks

Remote Timing Attacks

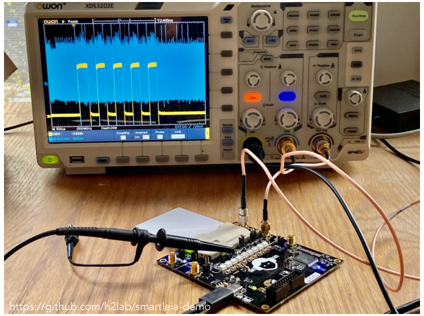


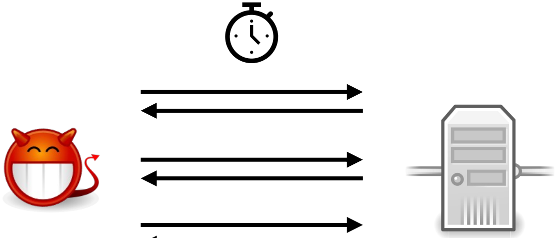


Hertzbleed: a New Class of Attacks

Power Side-Channel Attacks

Remote Timing Attacks

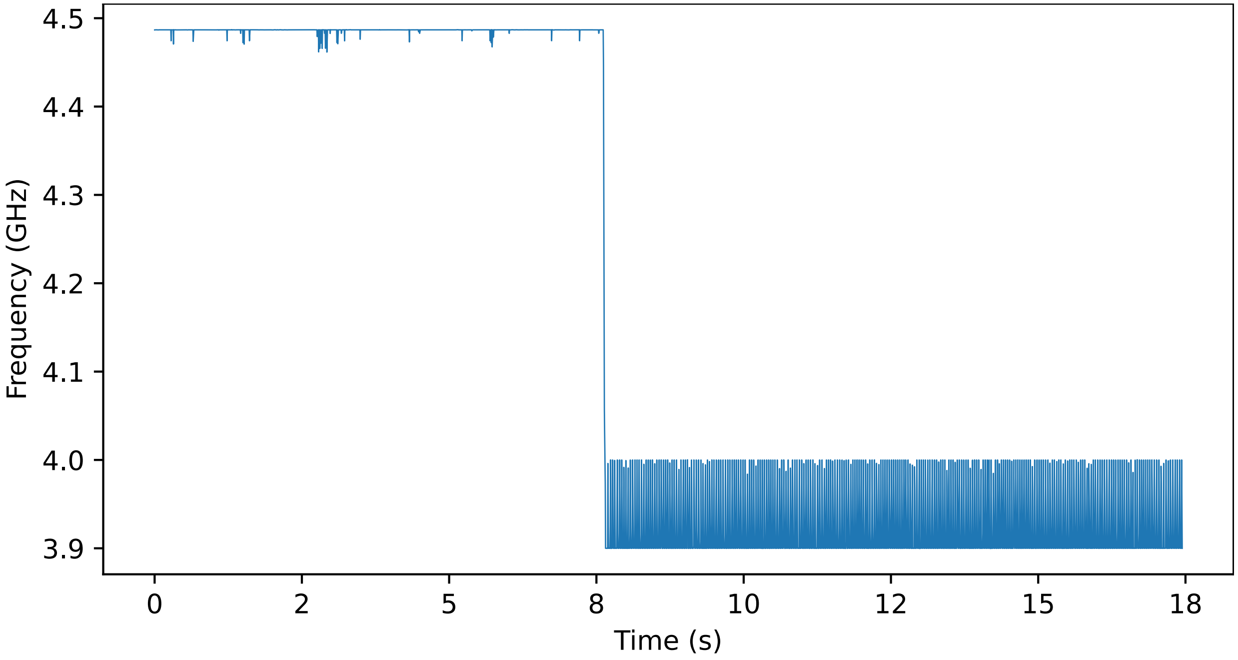




Hertzbleed: exploiting
dynamic frequency scaling (DVFS)

DVFS on a modern Intel CPU

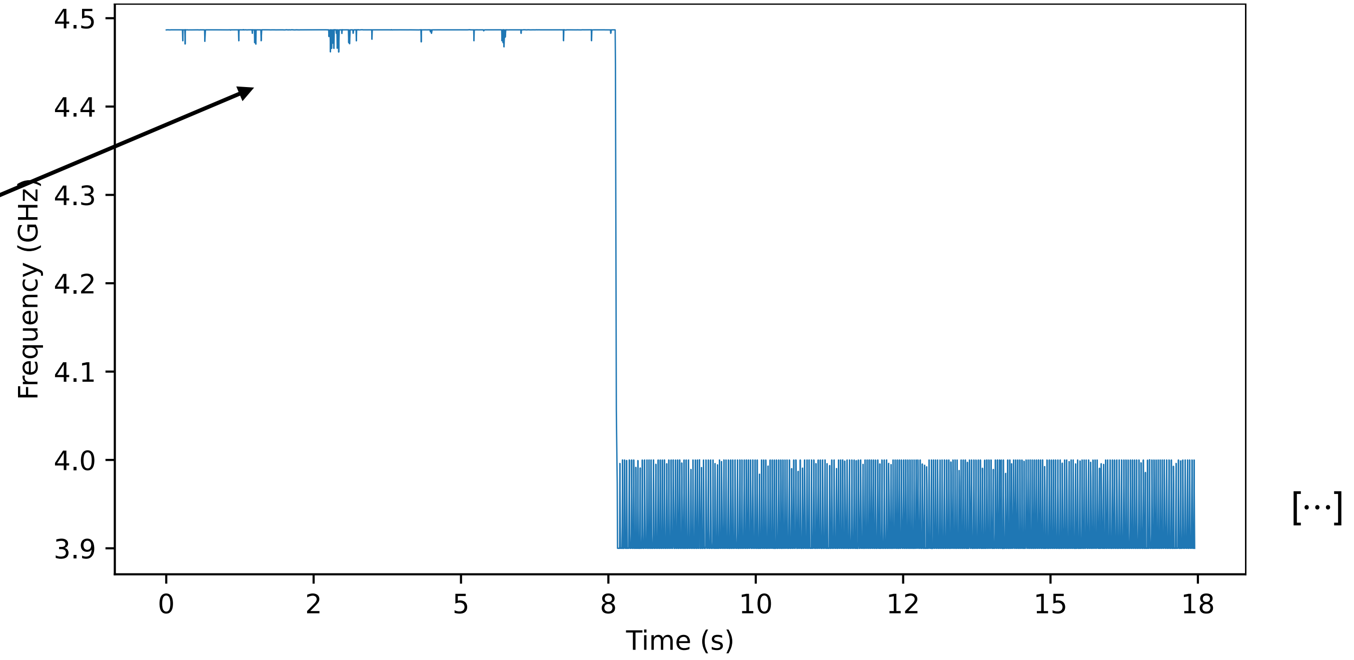
DVFS on a modern Intel CPU



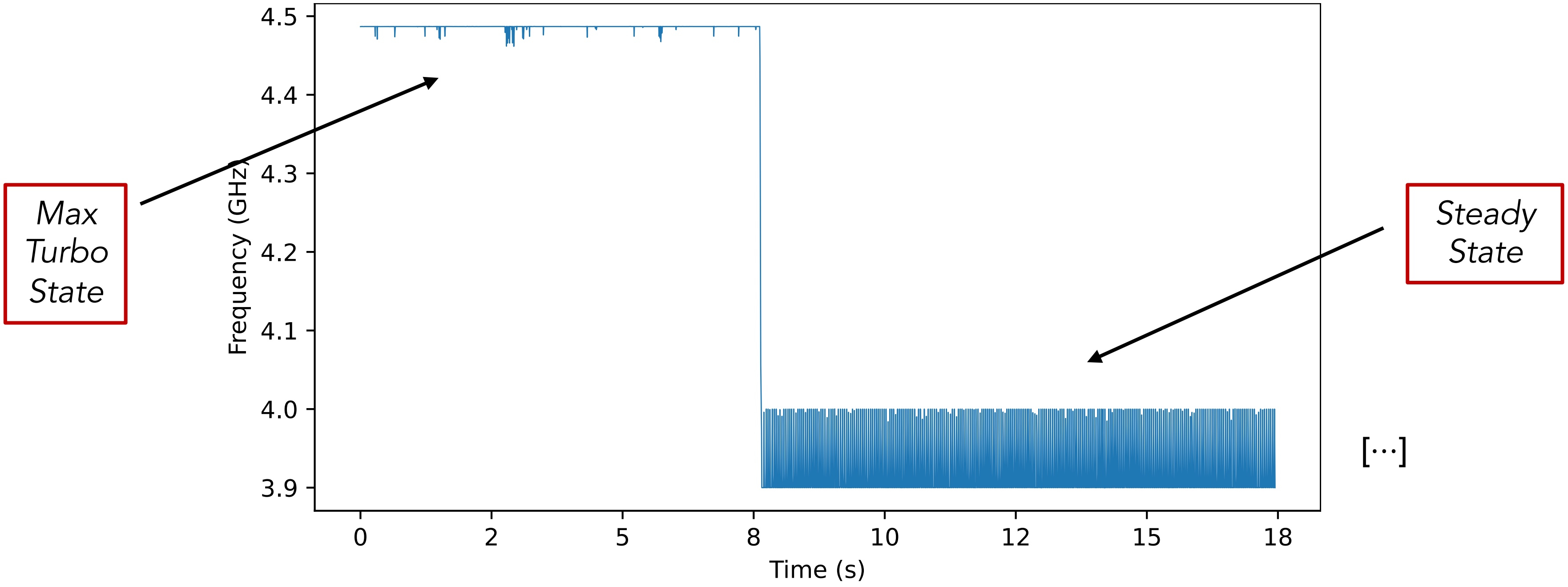
[···]

DVFS on a modern Intel CPU

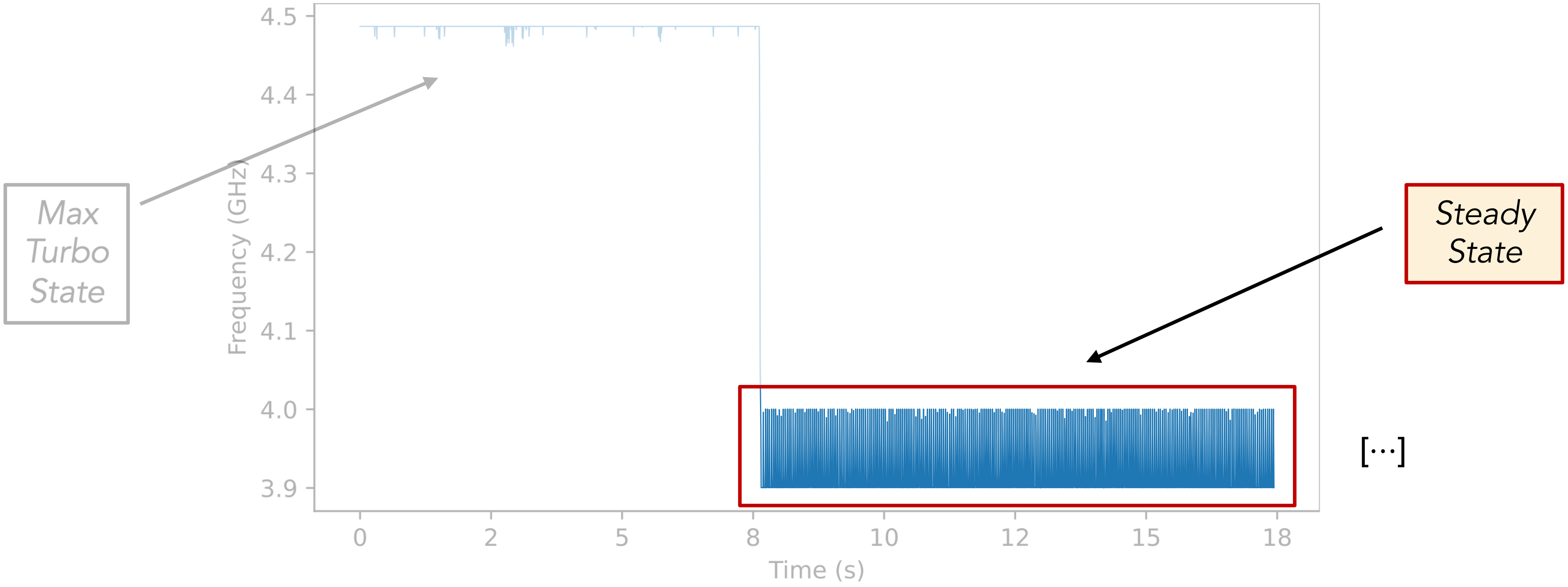
Max Turbo State



DVFS on a modern Intel CPU



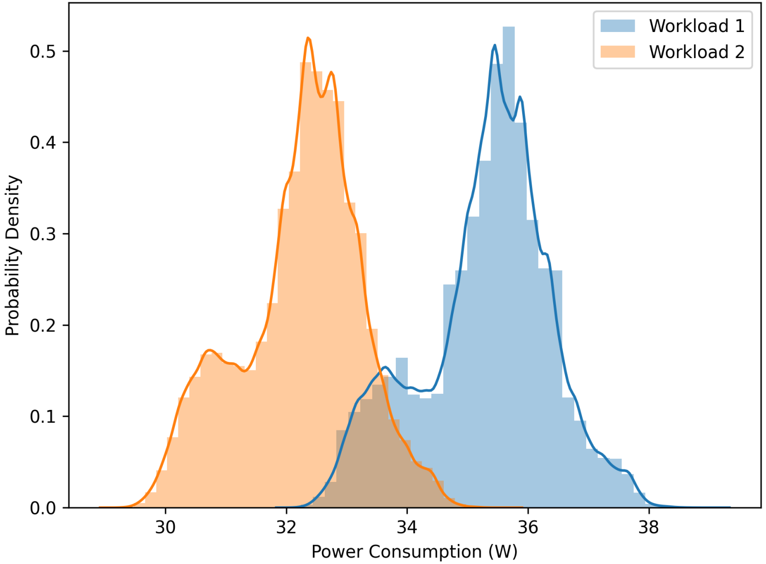
DVFS on a modern Intel CPU



Frequency Depends on Power

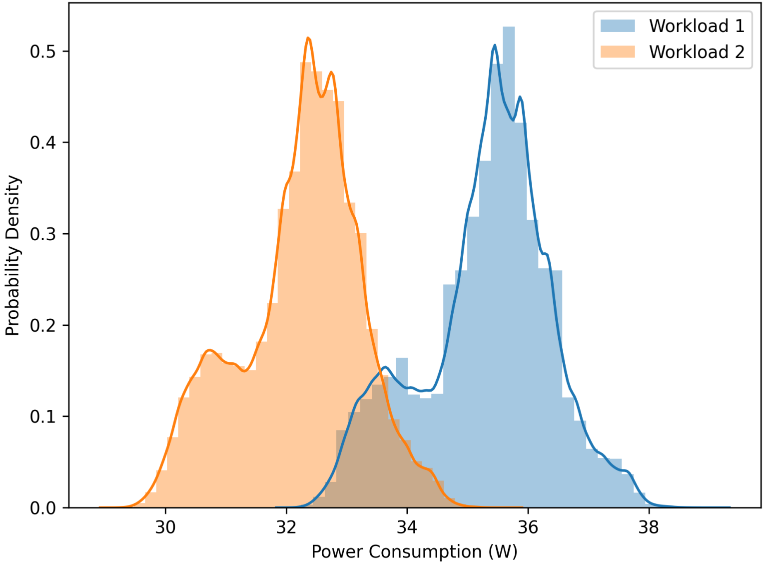
Frequency Depends on Power

Power Consumption

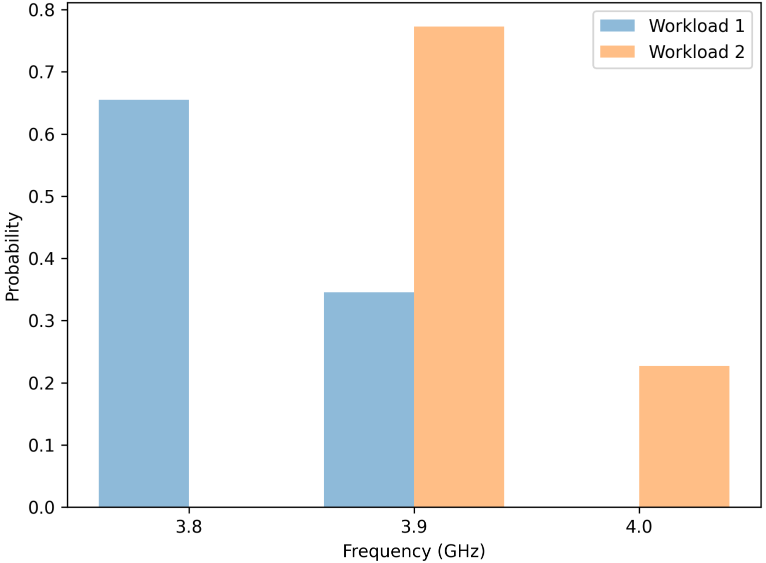


Frequency Depends on Power

Power Consumption



CPU Frequency



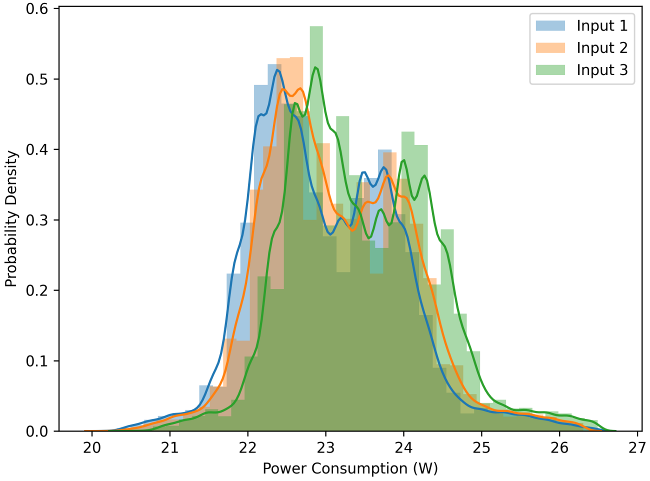
Frequency Depends on Data

- Only vary the data values being processed (“Input”).

Frequency Depends on Data

- Only vary the data values being processed ("Input").

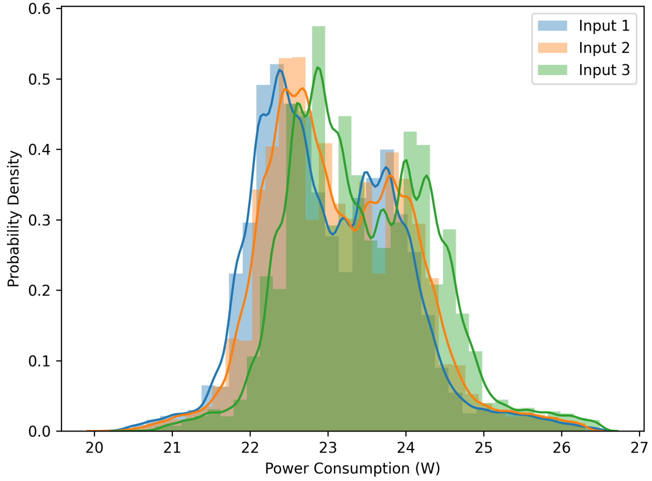
Power Consumption



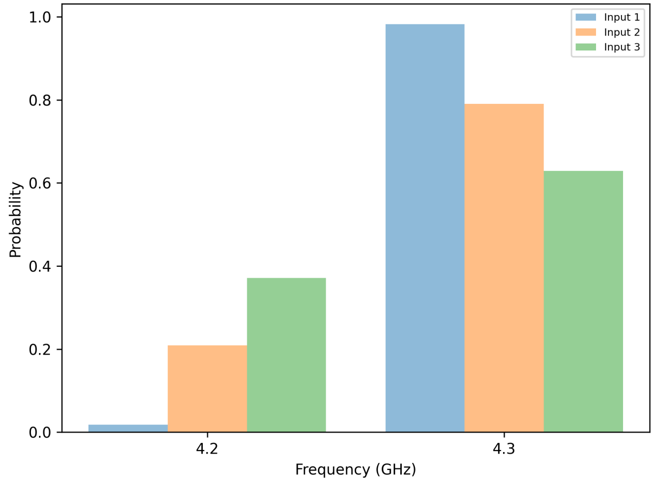
Frequency Depends on Data

- Only vary the data values being processed ("Input").

Power Consumption



CPU Frequency



Example of Data-Dependent Frequency

```
Function Sum(first, second):
    a = first
    b = second
    sum = a + b
    return sum
```

Example of Data-Dependent Frequency

Function `Sum`(*first*, *second*):
  a = first
  b = second
  sum = a + b
  return sum

Test 1 (CVE 1 number)**:**
first = 2022
second = 23823

Example of Data-Dependent Frequency

Function `Sum`(*first*, *second*):
  a = first
  b = second
  sum = a + b
  return sum

Test 1 (CVE 1 number):
first = 2022
second = 23823

Test 2 (CVE 2 number):
first = 2022
second = 24436

Example of Data-Dependent Frequency

Function `Sum`(*first*, *second*):

$$a = \text{first}$$

$$b = \text{second}$$

$$\text{sum} = a + b$$

return sum

Test 1 (CVE 1 number):

first = 2022

second = 23823

Test 2 (CVE 2 number):

first = 2022

second = 24436

Which Runs at a Higher Frequency?

Example of Data-Dependent Frequency

Function Sum(*first*, *second*):
a = first
b = second
sum = a + b
return sum

Test 1 (CVE 1 number):
first = 2022
second = 23823

Test 2 (CVE 2 number):
first = 2022
second = 24436

Which Runs at a Higher Frequency?

Example of Data-Dependent Frequency

Function `Sum`(*first*, *second*):
  a = first
  b = second
  sum = a + b
  return sum

Test 1 (CVE 1 number):
first = 2022
second = 23823

Test 2 (CVE 2 number):
first = 2022
second = 24436

Which Runs at a Higher Frequency?

We construct a *leakage model*
to answer this question.

Frequency Leakage Model

Frequency Leakage Model

Three *independent* effects:

1. Hamming distance (HD)
2. Hamming weight (HW)
3. Bit positions!

Frequency Leakage Model

Three *independent* effects:

1. Hamming distance (HD)
2. Hamming weight (HW)
3. Bit positions!

Frequency Leakage Model

Three *independent* effects:

1. Hamming distance (HD)
2. Hamming weight (HW)
3. Bit positions!

```
ax ← 0000000011111111
ax ← 0001111111100000
```

Frequency Leakage Model

Three *independent* effects:

1. Hamming distance (HD)
2. Hamming weight (HW)
3. Bit positions!

```
ax ← 0000000011111111
ax ← 0001111111100000
```

HD = 10

Frequency Leakage Model

Three *independent* effects:

1. Hamming distance (HD)
2. Hamming weight (HW)
3. Bit positions!

```
ax ← 0000000011111111
ax ← 0001111111100000
```

HD = 10

```
ax ← 0000000011111111
ax ← 0000011111111000
```

HD = 6

Frequency Leakage Model

Three *independent* effects:

1. **Hamming distance (HD)**
2. Hamming weight (HW)
3. Bit positions!

ax ← 0000000011111111
ax ← 0001111111100000

HD = 10

Consumes less power

Runs at a higher frequency!

→

ax ← 0000000011111111
ax ← 0000011111111000

HD = 6

Frequency Leakage Model

Three *independent* effects:

1. Hamming distance (HD)
2. Hamming weight (HW)
3. Bit positions!

Frequency Leakage Model

Three *independent* effects:

1. Hamming distance (HD)
2. Hamming weight (HW)
3. Bit positions!

```
ax ← 1111001111001111
ax ← ax | ax
```

Frequency Leakage Model

Three *independent* effects:

1. Hamming distance (HD)
2. **Hamming weight (HW)**
3. Bit positions!

```
ax ← 1111001111001111
ax ← ax | ax
```

HW = 12

Frequency Leakage Model

Three *independent* effects:

1. Hamming distance (HD)
2. Hamming weight (HW)
3. Bit positions!

```
ax ← 1111001111001111
ax ← ax | ax
```

HW = 12

```
ax ← 1100110011001100
ax ← ax | ax
```

HW = 8

Frequency Leakage Model

Three *independent* effects:

1. Hamming distance (HD)
2. **Hamming weight (HW)**
3. Bit positions!

```
ax ← 1111001111001111
ax ← ax | ax
```

HW = 12

Consumes less power

Runs at a higher frequency!

→

```
ax ← 1100110011001100
ax ← ax | ax
```

HW = 8

Frequency Leakage Model

Three *independent* effects:

1. Hamming distance (HD)
2. Hamming weight (HW)
3. **Bit positions!**

Frequency Leakage Model

Three *independent* effects:

1. Hamming distance (HD)
2. Hamming weight (HW)
3. **Bit positions!**

`ax ← 1111111100000000`

`ax ← ax | ax`

HW = 8

Frequency Leakage Model

Three *independent* effects:

1. Hamming distance (HD)
2. Hamming weight (HW)
3. **Bit positions!**

```
ax ← 1111111100000000
ax ← ax | ax
```

HW = 8

```
ax ← 0000000011111111
ax ← ax | ax
```

HW = 8

Frequency Leakage Model

Three *independent* effects:

1. Hamming distance (HD)
2. Hamming weight (HW)
3. **Bit positions!**

ax ← 1111111100000000

ax ← ax | ax

HW = 8

Consumes less power

Runs at a higher frequency!

→

ax ← 0000000011111111

ax ← ax | ax

HW = 8

Case Study: Bit Positions

Case Study: Bit Positions

```
rax = rcx = … = r11 = INPUT
loop:
  or %rax,%rcx     // rcx = rax | rcx
  or %rax,%rdx     // rdx = rax | rdx
  or %rax,%rsi     // rsi = rax | rsi
  or %rax,%rdi     // rdi = rax | rdi
jmp loop
```

We control `INPUT`.

Case Study: Bit Positions

```
rax = rcx = … = r11 = INPUT
loop:
  or %rax,%rcx     // rcx = rax | rcx
  or %rax,%rdx     // rdx = rax | rdx
  or %rax,%rsi     // rsi = rax | rsi
  or %rax,%rdi     // rdi = rax | rdi
jmp loop
```

We control `INPUT`.

- HD = 0

Case Study: Bit Positions

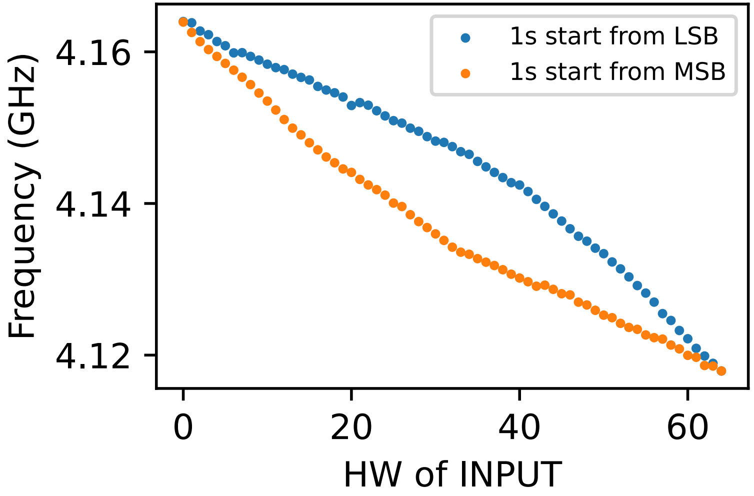
```
rax = rcx = … = r11 = INPUT
loop:
  or %rax,%rcx     // rcx = rax | rcx
  or %rax,%rdx     // rdx = rax | rdx
  or %rax,%rsi     // rsi = rax | rsi
  or %rax,%rdi     // rdi = rax | rdi
jmp loop
```

We control `INPUT`.

- HD = 0
- HW: # of 1s in `INPUT`
- Bit positions: positions of 1s in `INPUT`

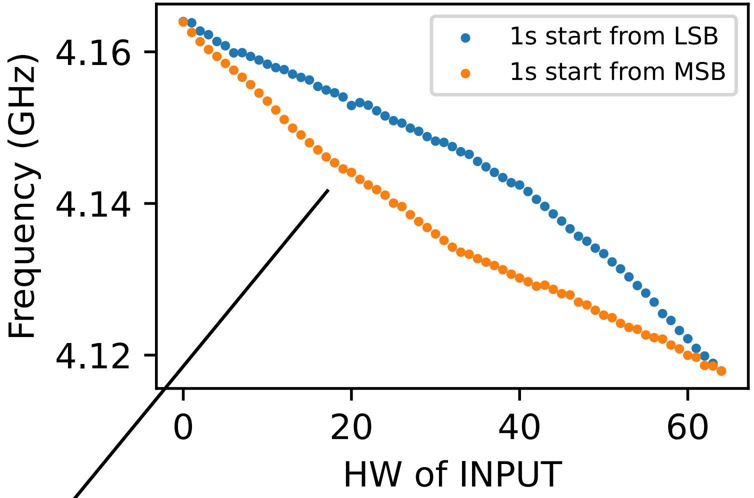
Case Study: Bit Positions

```
rax = rcx = … = r11 = INPUT
loop:
  or %rax,%rcx     // rcx = rax | rcx
  or %rax,%rdx     // rdx = rax | rdx
  or %rax,%rsi     // rsi = rax | rsi
  or %rax,%rdi     // rdi = rax | rdi
jmp loop
```



Case Study: Bit Positions

```
rax = rcx = … = r11 = INPUT
loop:
  or %rax,%rcx     // rcx = rax | rcx
  or %rax,%rdx     // rdx = rax | rdx
  or %rax,%rsi     // rsi = rax | rsi
  or %rax,%rdi     // rdi = rax | rdi
jmp loop
```



nonlinear?

Case Study: Bit Positions

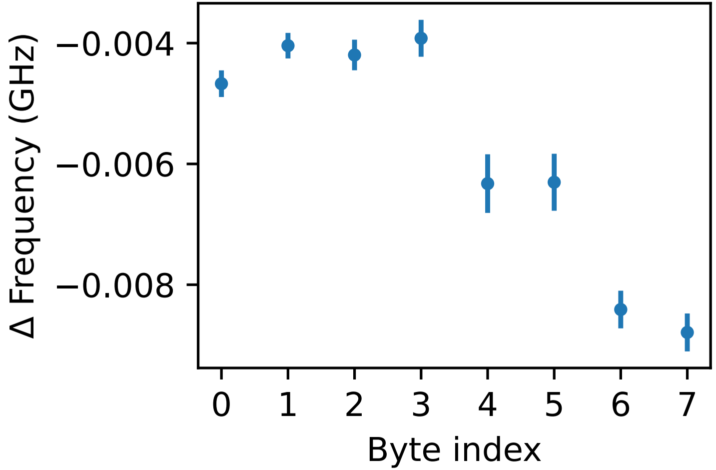
```
rax = rcx = … = r11 = INPUT
loop:
  or %rax,%rcx     // rcx = rax | rcx
  or %rax,%rdx     // rdx = rax | rdx
  or %rax,%rsi     // rsi = rax | rsi
  or %rax,%rdi     // rdi = rax | rdi
jmp loop
```

Delta freq between setting byte *i* to `0xff` (all 1s) and `0x00` (all 0s).

Case Study: Bit Positions

```
rax = rcx = … = r11 = INPUT
loop:
  or %rax,%rcx    // rcx = rax | rcx
  or %rax,%rdx    // rdx = rax | rdx
  or %rax,%rsi    // rsi = rax | rsi
  or %rax,%rdi    // rdi = rax | rdi
jmp loop
```

Delta freq between setting byte *i* to `0xff` (all 1s) and `0x00` (all 0s).

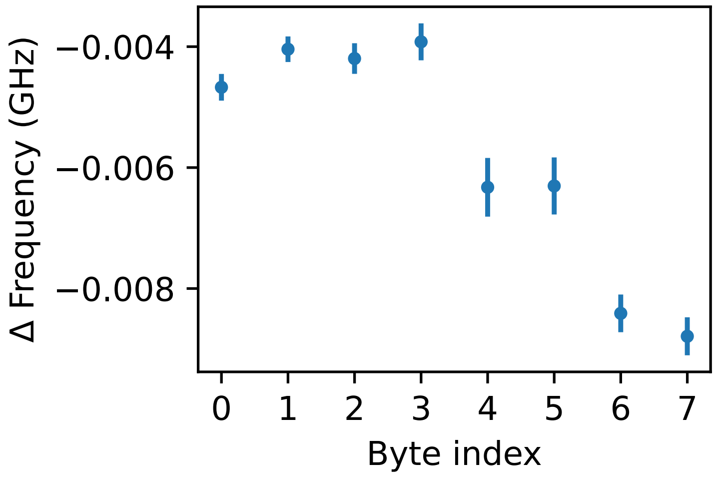


Case Study: Bit Positions

```
rax = rcx = … = r11 = INPUT
loop:
  or %rax,%rcx    // rcx = rax | rcx
  or %rax,%rdx    // rdx = rax | rdx
  or %rax,%rsi    // rsi = rax | rsi
  or %rax,%rdi    // rdi = rax | rdi
jmp loop
```

1s in the most significant bytes affect frequency (and power) more than 1s in the least significant bytes!

Delta freq between setting byte *i* to `0xff` (all 1s) and `0x00` (all 0s).

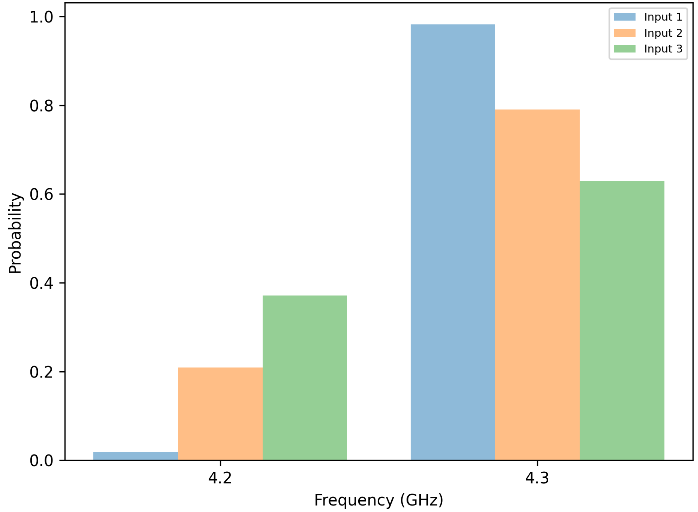


More experiments in the paper!

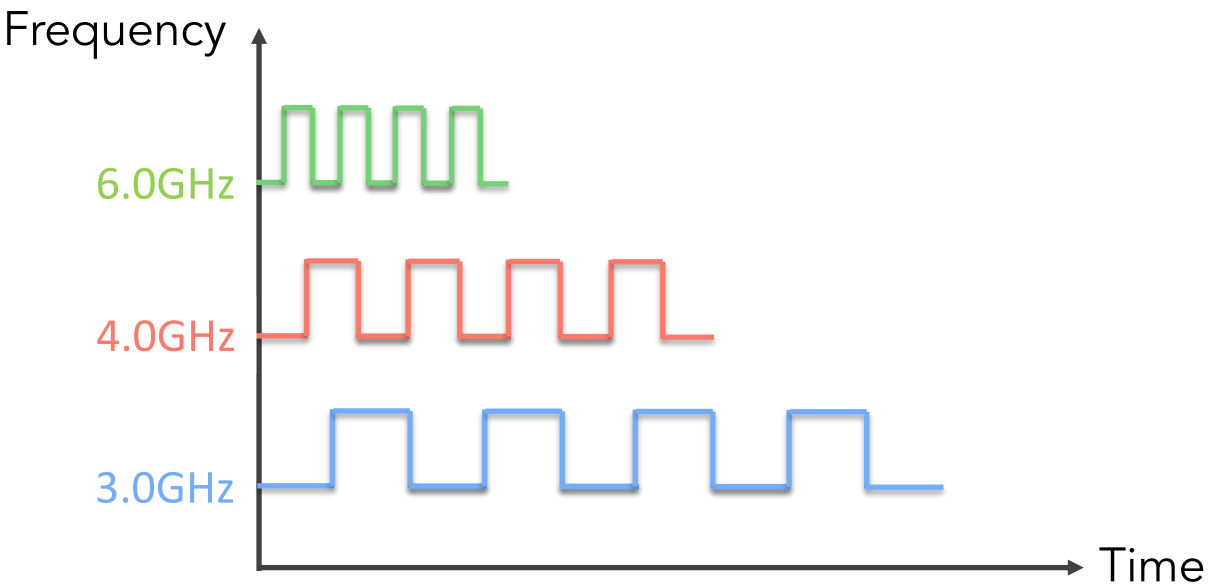
- We also show that these effects are ***independent*** and ***additive***.

More experiments in the paper!

- We also show that these effects are ***independent*** and ***additive***.
- <u>Takeaway so far</u>: computing on data with different HD, HW, or bit patterns can result in different CPU frequencies



Frequency Shows Through Timing!



Supersingular Isogeny Key Encapsulation

- SIKE: public key encryption scheme used to secure a shared key

Supersingular Isogeny Key Encapsulation

- SIKE: public key encryption scheme used to secure a shared key
  - A key generation algorithm: $(pk, sk) \leftarrow \texttt{KeyGen}()$
  - An encapsulation algorithm: $(k, c) \leftarrow \texttt{Encaps}(pk)$
  - A decapsulation algorithm: $k \leftarrow \texttt{Decaps}(sk, c)$

Supersingular Isogeny Key Encapsulation

- SIKE: public key encryption scheme used to secure a shared key
  - A key generation algorithm: $(pk, sk) \leftarrow$ `KeyGen`()
  - An encapsulation algorithm: $(k, c) \leftarrow$ `Encaps`(pk)
  - A decapsulation algorithm: $k \leftarrow$ `Decaps`(sk, c)
    - c can be anything

SIKE is a widely studied PQC scheme

CLOUDFLARE

Introducing CIRCL: An Advanced Cryptographic Library

06/20/2019

microsoft/
PQCrypto-SIDH

SIDH Library is a fast and portable software library that implements state-of-the-art supersingular isogeny cryptographic schemes. The chosen parameters aim...

NIST

PQC Standardization Process: Announcing Four Candidates to be Standardized, Plus Fourth Round Candidates

July 05, 2022

AWS KMS and ACM now support the latest hybrid post-quantum TLS ciphers

Posted On: Mar 16, 2022

aws

SIKE Attack Overview

$c' \rightarrow$ `Decapsulation(sk)`

SIKE Attack Overview

$c' \to$ `Decapsulation(sk)`

First bit = 0

SIKE Attack Overview

$c' \rightarrow$ `Decapsulation(sk)`

First bit = 0

```
[18257987050416722270 10199691716891914004 5400919966884858033 442692516914010372 12773429574585753468 6570432586462705433
[18257987050416722270 10199691716891914004 5400919966884858033 442692516914010372 12773429574585753468 6570432586462705433
[16825517838011775506 9371345100327880239 15742012313593718125 10072166590248559322 4496384122748719145 4334614431051425958
[16825517838011775506 9371345100327880239 15742012313593718125 10072166590248559322 4496384122748719145 4334614431051425958
[16740411979447774187 12800619824809723005 10853297027917591334 14982452220610001381 3800581514987461454 14564706225204848 9
[16825517838011775506 9371345100327880239 15742012313593718125 10072166590248559322 4496384122748719145 4334614431051425958
[15266560997122586383 12357889650077672103 2140980805765852605 11095593088850010277 4943695249444786644 3789592123877698465
[16825517838011775506 9371345100327880239 15742012313593718125 10072166590248559322 4496384122748719145 4334614431051425958
[17514449317981757582 13935337384110704213 3655249417699386156 3985202987060982889 6244515921924735727 5708423491580141120
[16825517838011775506 9371345100327880239 15742012313593718125 10072166590248559322 4496384122748719145 4334614431051425958
[6399333684299274457 13212325817985983642 17337857241394967822 17279086886397221 17809839742204337052 1762074623773052795 5
[16825517838011775506 9371345100327880239 15742012313593718125 10072166590248559322 4496384122748719145 4334614431051425958
[7661921139160543741 14707234213449645962 12995640548762497303 6366902398648164883 8159614558565625434 1699352089132060401
[16825517838011775506 9371345100327880239 15742012313593718125 10072166590248559322 4496384122748719145 4334614431051425958
[14502626884453394147 15471763763527174 74 16458194833336887993 11353529865322400496 18111664530210683277 14913675946160555 3
[16825517838011775506 9371345100327880239 15742012313593718125 10072166590248559322 4496384122748719145 4334614431051425958
[16292054110271644132 16402476987519138840 15295718925489978418 11943223229381717154 3939738083047782821 7326838894173812 78
[16825517838011775506 9371345100327880239 15742012313593718125 10072166590248559322 4496384122748719145 4334614431051425958
[3109004547777643616 11545393661624748910 13937898936545491063 17877233530308340276 12362683729304085816 13533668621992669 0
[16825517838011775506 9371345100327880239 15742012313593718125 10072166590248559322 4496384122748719145 4334614431051425958
[14038051704368540915 5562911119597974536 6094927895374002108 15637883509309970852 13225965834113072042 16448264607636784 36
[16825517838011775506 9371345100327880239 15742012313593718125 10072166590248559322 4496384122748719145 4334614431051425958
[18026979082695398473 4545160433979536065 8917105521003255669 8960042842321690905 14664092097322583836 17637668258660114635
[18026979082695398473 4545160433979536065 8917105521003255669 8960042842321690905 14664092097322583836 17637668258660114635
[11077978705126494 49 3973889343021377473 15397880414487620049 16447202818859113108 13000223568372994767 11226627242531480 57
[11077978705126494 49 3973889343021377473 15397880414487620049 16447202818859113108 13000223568372994767 11226627242531480 57
[4891726318645018377 8700905460331914174 16989373266006030040 12684865615969117539 7224866818255934154 7985345352476117507
[4891726318645018377 8700905460331914174 16989373266006030040 12684865615969117539 7224866818255934154 7985345352476117507
[12590157596330527957 4309529106583776359 4309400783656274695 3291966690254440795 2093371576388436535 9104435448238030533 3
[12590157596330527957 4309529106583776359 4309400783656274695 3291966690254440795 2093371576388436535 9104435448238030533 3
[14258829650860490490 3159096382619497511 1628360891341362022 188059088762649621 3407697486830288305 5337080128759733026 67
[14258829650860490490 3159096382619497511 1628360891341362022 188059088762649621 3407697486830288305 5337080128759733026 67
[3595963780190459985 717527953757986 42 1255466713598110361 7 1100338600145278727 9 1659580706669562711 0 1628175452102986565 2
[3595963780190459985 717527953757986 42 1255466713598110361 7 1100338600145278727 9 1659580706669562711 0 1628175452102986565 2
[16439123124117376047 9302501002909150857 16313838798130370411 3270836818630356446 2783122717285403776 1391244912843384146 1
[3595963780190459985 717527953757986 42 1255466713598110361 7 1100338600145278727 9 1659580706669562711 0 1628175452102986565 2
```

SIKE Attack Overview

$c' \rightarrow$ `Decapsulation(sk)`

First bit = 0

First bit = 1


```
[18257987050416722270 10199691716891914004 5400919966884858033 442692516914010372 12773429574585753468 6570432586462705433
[18257987050416722270 10199691716891914004 5400919966884858033 442692516914010372 12773429574585753468 6570432586462705433
[16825517838011775506 9371345100327880239 15742012313593718125 10072166590248559322 4496384122748719145 4334614431051425958
[16825517838011775506 9371345100327880239 15742012313593718125 10072166590248559322 4496384122748719145 4334614431051425958
[16740411979447774187 12800619824809723005 10853297027917591334 14982452220610001381 3800581514987461454 145647062252048489
[16825517838011775506 9371345100327880239 15742012313593718125 10072166590248559322 4496384122748719145 4334614431051425958
[15266560997122586383 12357889650077672103 2140980805765852605 11095593088850010277 4943695249444786644 3789592123877698465
[16825517838011775506 9371345100327880239 15742012313593718125 10072166590248559322 4496384122748719145 4334614431051425958
[17514449317981757582 13935337384110704213 3655249417699386156 3985202987060982889 6244515921924735727 5708423491580141120
[16825517838011775506 9371345100327880239 15742012313593718125 10072166590248559322 4496384122748719145 4334614431051425958
[6399333684299274457 13212325817985983642 17337857241394967822 17279086886397221 17809839742204337052 17620746237730527955
[16825517838011775506 9371345100327880239 15742012313593718125 10072166590248559322 4496384122748719145 4334614431051425958
[7661921139160543741 14707234213449645962 12995640548762497303 6366902398648164883 8159614558565625434 1699352089132060401
[16825517838011775506 9371345100327880239 15742012313593718125 10072166590248559322 4496384122748719145 4334614431051425958
[14502626884453394147 1547176376352717474 16458194833336887993 11353529865322400496 18111664530210683277 149136759461605553
[16825517838011775506 9371345100327880239 15742012313593718125 10072166590248559322 4496384122748719145 4334614431051425958
[16292054110271644132 16402476987519138840 15295718925489978418 11943223229381717154 3939738083047782821 732683889417381278
[16825517838011775506 9371345100327880239 15742012313593718125 10072166590248559322 4496384122748719145 4334614431051425958
[3109004547777643616 11545393661624748910 13937898936545491063 17877233530308340276 12362683729304085816 135336686219926690
[16825517838011775506 9371345100327880239 15742012313593718125 10072166590248559322 4496384122748719145 4334614431051425958
[14038051704368540915 5562911119597974536 6094927895374002108 15637883509309970852 13225965834113072042 1644826460763678436
[16825517838011775506 9371345100327880239 15742012313593718125 10072166590248559322 4496384122748719145 4334614431051425958
[18026979082695398473 4545160433979536065 8917105521003255669 8960042842321690905 14664092097322583836 17637668258660114635
[18026979082695398473 4545160433979536065 8917105521003255669 8960042842321690905 14664092097322583836 17637668258660114635
[1107797870512649449 3973889343021377473 15397880414487620049 16447202818859113108 13000223568372994767 1122662724253148057
[1107797870512649449 3973889343021377473 15397880414487620049 16447202818859113108 13000223568372994767 1122662724253148057
[4891726318645018377 8700905460331914174 16989373266006030040 12684865615969117539 7224866818255934154 7985345352476117507
[4891726318645018377 8700905460331914174 16989373266006030040 12684865615969117539 7224866818255934154 7985345352476117507
[12590157596330527957 4309529106583776359 4309400783656274695 3291966690254440795 2093371576388436535 9104435448238030533 3
[12590157596330527957 4309529106583776359 4309400783656274695 3291966690254440795 2093371576388436535 9104435448238030533 3
[14258829650860490490 3159096382619497511 1628360891341362022 188059088762649621 3407697486830288305 5337080128759733026 67
[14258829650860490490 3159096382619497511 1628360891341362022 188059088762649621 3407697486830288305 5337080128759733026 67
[3595963780190459985 71752795375798642 12554667135981103617 11003386001452787279 16595807066695627110 16281754521029865652
[3595963780190459985 71752795375798642 12554667135981103617 11003386001452787279 16595807066695627110 16281754521029865652
[16439123124117376047 9302501002909150857 16313838798130370411 3270836818630356446 2783122717285403776 13912449128433841461
[3595963780190459985 71752795375798642 12554667135981103617 11003386001452787279 16595807066695627110 16281754521029865652
```

SIKE Attack Overview

`c′ → Decapsulation(sk)`

First bit = 0


```
[18257987050416722270 10199691716891914004 5400919966884858033 442692516914010372 12773429574585753468 6570432586462705433
[18257987050416722270 10199691716891914004 5400919966884858033 442692516914010372 12773429574585753468 6570432586462705433
[16825517838011775506 9371345100327880239 15742012313593718125 10072166590248559322 4496384122748719145 4334614431051425958
[16825517838011775506 9371345100327880239 15742012313593718125 10072166590248559322 4496384122748719145 4334614431051425958
[16740411979447774187 12800619824809723005 10853297027917591334 14982452220610001381 3800581514987461454 145647062252048489
[16825517838011775506 9371345100327880239 15742012313593718125 10072166590248559322 4496384122748719145 4334614431051425958
[15266560997122586383 12357889650077672103 2140980805765852605 11095593088850010277 4943695249444786644 3789592123877698465
[16825517838011775506 9371345100327880239 15742012313593718125 10072166590248559322 4496384122748719145 4334614431051425958
[17514449317981757582 13935337384110704213 3655249417699386156 3985202987060982889 6244515921924735727 5708423491580141120
[16825517838011775506 9371345100327880239 15742012313593718125 10072166590248559322 4496384122748719145 4334614431051425958
[6399333684299274457 13212325817985983642 17337857241394967822 17279086886397221 17809839742204337052 17620746237730527955
[16825517838011775506 9371345100327880239 15742012313593718125 10072166590248559322 4496384122748719145 4334614431051425958
[7661921139160543741 14707234213449645962 12995640548762497303 6366902398648164883 8159614558565625434 1699352089132060401
[16825517838011775506 9371345100327880239 15742012313593718125 10072166590248559322 4496384122748719145 4334614431051425958
[14502626884453394147 1547176376352717474 16458194833336887993 11353529865322400496 18111664530210683277 149136759461605553
[16825517838011775506 9371345100327880239 15742012313593718125 10072166590248559322 4496384122748719145 4334614431051425958
[16292054110271644132 16402476987519138840 15295718925489978418 11943223229381717154 3939738083047782821 732683889417381278
[16825517838011775506 9371345100327880239 15742012313593718125 10072166590248559322 4496384122748719145 4334614431051425958
[3109004547777643616 11545393661624748910 13937898936545491063 17877233530308340276 12362683729304085816 135336686219926690
[16825517838011775506 9371345100327880239 15742012313593718125 10072166590248559322 4496384122748719145 4334614431051425958
[14038051704368540915 5562911119597974536 6094927895374002108 15637883509309970852 13225965834113072042 1644826460763678436
[16825517838011775506 9371345100327880239 15742012313593718125 10072166590248559322 4496384122748719145 4334614431051425958
[18026979082695398473 4545160433979536065 8917105521003255669 8960042842321690905 14664092097322583836 17637668258660114635
[18026979082695398473 4545160433979536065 8917105521003255669 8960042842321690905 14664092097322583836 17637668258660114635
[1107797870512649449 3973889343021377473 15397880414487620049 16447202818859113108 13000223568372994767 1122662724253148057
[1107797870512649449 3973889343021377473 15397880414487620049 16447202818859113108 13000223568372994767 1122662724253148057
[4891726318645018377 8700905460331914174 16989373266006030040 12684865615969117539 7224866818255934154 7985345352476117507
[4891726318645018377 8700905460331914174 16989373266006030040 12684865615969117539 7224866818255934154 7985345352476117507
[12590157596330527957 4309529106583776359 4309400783656274695 3291966690254440795 2093371576388436535 9104435448238030533 3
[12590157596330527957 4309529106583776359 4309400783656274695 3291966690254440795 2093371576388436535 9104435448238030533 3
[14258829650860490490 3159096382619497511 1628360891341362022 188059088762649621 3407697486830288305 5337080128759733026 67
[14258829650860490490 3159096382619497511 1628360891341362022 188059088762649621 3407697486830288305 5337080128759733026 67
[3595963780190459985 71752795375798642 12554667135981103617 11003386001452787279 16595807066695627110 16281754521029865652
[3595963780190459985 71752795375798642 12554667135981103617 11003386001452787279 16595807066695627110 16281754521029865652
[16439123124117376047 9302501002909150857 16313838798130370411 3270836818630356446 2783122717285403776 13912449128433841461
[3595963780190459985 71752795375798642 12554667135981103617 11003386001452787279 16595807066695627110 16281754521029865652
```


First bit = 1


```
[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]]} {[15072814314898230369 15028074021671655129 12525656
[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]]} {[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]]} {[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]]} {[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]]} {[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]]} {[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]]} {[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]]} {[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]]} {[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]]} {[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]]} {[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]]} {[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]]} {[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]]} {[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]]} {[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]]} {[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]]} {[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]]} {[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]]} {[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]]} {[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]]} {[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]]} {[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]]} {[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]]} {[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]]} {[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]]} {[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]]} {[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]]} {[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]]} {[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]]} {[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]]} {[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]]} {[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]]} {[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]]} {[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]
[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]]} {[0 0 0 0 0 0 0 0 0 0 0 0] [0 0 0 0 0 0 0 0 0 0 0 0]
```

SIKE Attack Overview

$c' \rightarrow$ `Decapsulation(sk)`

First bit = 0

$a = a \times R_1$
$a = a \times R_2$
$a = a \times R_3$
…
$a = a \times R_n$

First bit = 1

$a = a \times 0$
$a = a \times R_2$
$a = a \times R_3$
…
$a = a \times R_n$

SIKE Attack Overview

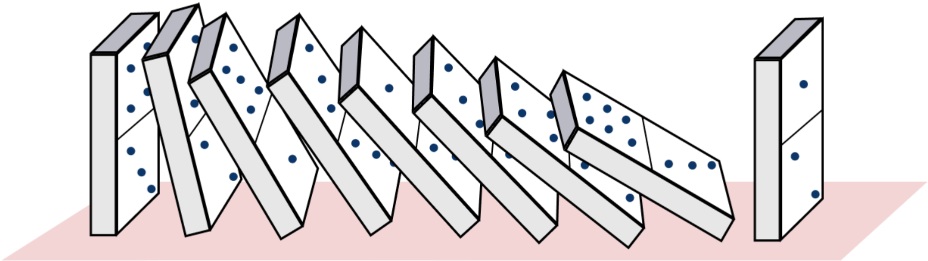
$c' \rightarrow$ `Decapsulation(sk)`

First bit = 0

$a = a \times R_1$
$a = a \times R_2$
$a = a \times R_3$
...
$a = a \times R_n$

First bit = 1

$a = a \times 0$
$a = a \times R_2$
$a = a \times R_3$
...
$a = a \times R_n$



SIKE Attack Overview

$c' \rightarrow$ `Decapsulation(sk)`

First bit = 0

$a = a \times R_1$
$a = a \times R_2$
$a = a \times R_3$
…
$a = a \times R_n$

More power
Lower frequency
Longer runtime

First bit = 1

$a = a \times 0$
$a = a \times R_2$
$a = a \times R_3$
…
$a = a \times R_n$

Less power
Higher frequency
Shorter runtime

An Important Step of SIKE's Decapsulation

Algorithm 8: Three point ladder

function `Ladder3pt`

Input: $m = (m_{\ell-1}, \ldots, m_0)_2 \in \mathbb{Z}$, (x_P, x_Q, x_{Q-P}), and $(A : 1)$

Output: $(X_{P+[m]Q} : Z_{P+[m]Q})$

$((X_0 : Z_0), (X_1 : Z_1), (X_2 : Z_2)) \leftarrow ((x_Q : 1), (x_P : 1), (x_{Q-P} : 1))$

$a_{24}^{+} \leftarrow (A + 2)/4$

for $i = 0$ **to** $\ell - 1$ **do**

  if $m_i = 1$ **then**

    $((X_0 : Z_0), (X_1 : Z_1)) \leftarrow \texttt{xDBLADD}((X_0 : Z_0), (X_1 : Z_1), (X_2 : Z_2), (a_{24}^{+} : 1))$

  else

    $((X_0 : Z_0), (X_2 : Z_2)) \leftarrow \texttt{xDBLADD}((X_0 : Z_0), (X_2 : Z_2), (X_1 : Z_1), (a_{24}^{+} : 1))$

return $(X_1 : Z_1)$

Taken from SIKE's specification
Actual implementation has **no branches**

An Important Step of SIKE's Decapsulation

Algorithm 8: Three point ladder

```
function Ladder3pt
    Input: m = (m_{ℓ-1}, ..., m_0)_2 ∈ ℤ, (x_P, x_Q, x_{Q-P}), and (A : 1)
    Output: (X_{P+[m]Q} : Z_{P+[m]Q})
((X_0 : Z_0), (X_1 : Z_1), (X_2 : Z_2)) ← ((x_Q : 1), (x_P : 1), (x_{Q-P} : 1))
a24+ ← (A + 2)/4
for i = 0 to ℓ − 1 do
    if m_i = 1 then
        ((X_0 : Z_0), (X_1 : Z_1)) ← xDBLADD((X_0 : Z_0), (X_1 : Z_1), (X_2 : Z_2), (a24+ : 1))
    else
        ((X_0 : Z_0), (X_2 : Z_2)) ← xDBLADD((X_0 : Z_0), (X_2 : Z_2), (X_1 : Z_1), (a24+ : 1))
return (X_1 : Z_1)
```

Taken from SIKE's specification
Actual implementation has no branches

m is the (static) secret key

P and *Q* are points included in the ciphertext

An Important Step of SIKE's Decapsulation

Algorithm 8: Three point ladder

```
function Ladder3pt
    Input: m = (m_{ℓ-1}, ..., m_0)_2 ∈ Z, (x_P, x_Q, x_{Q-P}), and (A : 1)
    Output: (X_{P+[m]Q} : Z_{P+[m]Q})
((X_0 : Z_0), (X_1 : Z_1), (X_2 : Z_2)) ← ((x_Q : 1), (x_P : 1), (x_{Q-P} : 1))
a^+_{24} ← (A + 2)/4
for i = 0 to ℓ − 1 do
    if m_i = 1 then
        ((X_0 : Z_0), (X_1 : Z_1)) ← xDBLADD((X_0 : Z_0), (X_1 : Z_1), (X_2 : Z_2), (a^+_{24} : 1))
    else
        ((X_0 : Z_0), (X_2 : Z_2)) ← xDBLADD((X_0 : Z_0), (X_2 : Z_2), (X_1 : Z_1), (a^+_{24} : 1))
return (X_1 : Z_1)
```

Taken from SIKE's specification
Actual implementation has no branches

m is the (static) secret key

P and *Q* are points included in the ciphertext

At each loop iteration, the data flow depends on *P*, *Q* and m_i

An Important Step of SIKE's Decapsulation

$2U,\ U{+}V \leftarrow$ `xDBLADD(` U, V, W `)` where $W = U - V$

An Important Step of SIKE's Decapsulation

$$2U,\ U{+}V \leftarrow \texttt{xDBLADD}(U, V, W) \quad \text{where} \quad W = U - V$$

$W \in \{ T, O \}$ → $2U,\ (0{:}0) \leftarrow \texttt{xDBLADD}(U, V, W)$

An Important Step of SIKE's Decapsulation

$2U,\ U+V \leftarrow$ `xDBLADD`(U, V, W) where $W = U - V$

W ∈ { T, O } → $2U,\ (0{:}0) \leftarrow$ `xDBLADD`(U, V, W)

(0:0) is not a point

An Important Step of SIKE's Decapsulation

$2U,\ U+V \leftarrow$ `xDBLADD(` U, V, W `)` where $W = U - V$

$W \in \{ T, O \}$ → $2U,\ (0{:}0) \leftarrow$ `xDBLADD(` U, V, W `)`

$(0{:}0)$ is not a point

U or V or $W = (0{:}0)$ → $2U,\ (0{:}0) \leftarrow$ `xDBLADD(` U, V, W `)`

An Important Step of SIKE's Decapsulation

Algorithm 8: Three point ladder

function Ladder3pt

Input: $m = (m_{\ell-1}, \ldots, m_0)_2 \in \mathbb{Z}$, (x_P, x_Q, x_{Q-P}), and $(A : 1)$

Output: $(X_{P+[m]Q} : Z_{P+[m]Q})$

$((X_0 : Z_0), (X_1 : Z_1), (X_2 : Z_2)) \leftarrow ((x_Q : 1), (x_P : 1), (x_{Q-P} : 1))$

$a_{24}^{+} \leftarrow (A + 2)/4$

for $i = 0$ **to** $\ell - 1$ **do**

  if $m_i = 1$ **then**

    $((X_0 : Z_0), (X_1 : Z_1)) \leftarrow \texttt{xDBLADD}((X_0 : Z_0), (X_1 : Z_1), (X_2 : Z_2), (a_{24}^{+} : 1))$

  else

    $((X_0 : Z_0), (X_2 : Z_2)) \leftarrow \texttt{xDBLADD}((X_0 : Z_0), (X_2 : Z_2), (X_1 : Z_1), (a_{24}^{+} : 1))$

return $(X_1 : Z_1)$

Taken from SIKE's specification
Actual implementation has no branches

An Important Step of SIKE's Decapsulation

Iteration *i*

for $i = 0$ **to** $\ell - 1$ **do**
  if $m_i = 1$ **then**
    $((X_0 : Z_0), (X_1 : Z_1)) \leftarrow \texttt{xDBLADD}((X_0 : Z_0), (X_1 : Z_1), (X_2 : Z_2), (a_{24}^{+} : 1))$
  else
    $((X_0 : Z_0), (X_2 : Z_2)) \leftarrow \texttt{xDBLADD}((X_0 : Z_0), (X_2 : Z_2), (X_1 : Z_1), (a_{24}^{+} : 1))$

An Important Step of SIKE's Decapsulation

Iteration i

for $i = 0$ **to** $\ell - 1$ **do**
  if $m_i = 1$ **then**
    $((X_0 : Z_0), (X_1 : Z_1)) \leftarrow \texttt{xDBLADD}((X_0 : Z_0), (X_1 : Z_1), (X_2 : Z_2), (a_{24}^{+} : 1))$
  else
    $((X_0 : Z_0), (X_2 : Z_2)) \leftarrow \texttt{xDBLADD}((X_0 : Z_0), (X_2 : Z_2), (X_1 : Z_1), (a_{24}^{+} : 1))$

T or O

An Important Step of SIKE's Decapsulation

Iteration i

for $i = 0$ **to** $\ell - 1$ **do**
  if $m_i = 1$ **then**
    $((X_0 : Z_0), (X_1 : Z_1)) \leftarrow \texttt{xDBLADD}((X_0 : Z_0), (X_1 : Z_1), (X_2 : Z_2), (a_{24}^{+} : 1))$
  else
    $((X_0 : Z_0), (X_2 : Z_2)) \leftarrow \texttt{xDBLADD}((X_0 : Z_0), (X_2 : Z_2), (X_1 : Z_1), (a_{24}^{+} : 1))$

$(0{:}0)$

T or O

An Important Step of SIKE's Decapsulation

Iteration i

for $i = 0$ **to** $\ell - 1$ **do**
  if $m_i = 1$ **then**
    $((X_0 : Z_0), (X_1 : Z_1)) \leftarrow \texttt{xDBLADD}((X_0 : Z_0), (X_1 : Z_1), (X_2 : Z_2), (a_{24}^{+} : 1))$
  else
    $((X_0 : Z_0), (X_2 : Z_2)) \leftarrow \texttt{xDBLADD}((X_0 : Z_0), (X_2 : Z_2), (X_1 : Z_1), (a_{24}^{+} : 1))$

(0:0) T or O

$2U, (0{:}0) \leftarrow \texttt{xDBLADD}(U, V, W)$ if $W \in \{T, O\}$

An Important Step of SIKE's Decapsulation

Iteration i+1

for $i = 0$ **to** $\ell - 1$ **do**
  if $m_i = 1$ **then**
    $((X_0 : Z_0), (X_1 : Z_1)) \leftarrow \texttt{xDBLADD}((X_0 : Z_0), (X_1 : Z_1), (X_2 : Z_2), (a_{24}^{+} : 1))$
  else
    $((X_0 : Z_0), (X_2 : Z_2)) \leftarrow \texttt{xDBLADD}((X_0 : Z_0), (X_2 : Z_2), (X_1 : Z_1), (a_{24}^{+} : 1))$

An Important Step of SIKE's Decapsulation

Iteration i+1

for $i = 0$ **to** $\ell - 1$ **do**
  if $m_i = 1$ **then**
    $((X_0 : Z_0), (X_1 : Z_1)) \leftarrow \texttt{xDBLADD}((X_0 : Z_0), (X_1 : Z_1), (X_2 : Z_2), (a_{24}^+ : 1))$
  else
    $((X_0 : Z_0), (X_2 : Z_2)) \leftarrow \texttt{xDBLADD}((X_0 : Z_0), (X_2 : Z_2), (X_1 : Z_1), (a_{24}^+ : 1))$

(0:0)

An Important Step of SIKE's Decapsulation

Iteration $i+1$

for $i = 0$ **to** $\ell - 1$ **do**
  if $m_i = 1$ **then**
    $((X_0 : Z_0), (X_1 : Z_1)) \leftarrow \texttt{xDBLADD}((X_0 : Z_0), (X_1 : Z_1), (X_2 : Z_2), (a_{24}^{+} : 1))$
  else
    $((X_0 : Z_0), (X_2 : Z_2)) \leftarrow \texttt{xDBLADD}((X_0 : Z_0), (X_2 : Z_2), (X_1 : Z_1), (a_{24}^{+} : 1))$

(0:0) (0:0)

An Important Step of SIKE's Decapsulation

Iteration i+1

for $i = 0$ **to** $\ell - 1$ **do**

  if $m_i = 1$ **then**

    $((X_0 : Z_0), (X_1 : Z_1)) \leftarrow \texttt{xDBLADD}((X_0 : Z_0), (X_1 : Z_1), (X_2 : Z_2), (a_{24}^{+} : 1))$

  else

    $((X_0 : Z_0), (X_2 : Z_2)) \leftarrow \texttt{xDBLADD}((X_0 : Z_0), (X_2 : Z_2), (X_1 : Z_1), (a_{24}^{+} : 1))$

(0:0)   (0:0)   (0:0)

An Important Step of SIKE's Decapsulation

Iteration $i+1$

for $i = 0$ **to** $\ell - 1$ **do**
  if $m_i = 1$ **then**
    $((X_0 : Z_0), (X_1 : Z_1)) \leftarrow \texttt{xDBLADD}((X_0 : Z_0), (X_1 : Z_1), (X_2 : Z_2), (a_{24}^{+} : 1))$
  else
    $((X_0 : Z_0), (X_2 : Z_2)) \leftarrow \texttt{xDBLADD}((X_0 : Z_0), (X_2 : Z_2), (X_1 : Z_1), (a_{24}^{+} : 1))$

(0:0) (0:0) (0:0)

$2U, (0{:}0) \leftarrow \texttt{xDBLADD}(U, V, W)$ if U or V or $W = (0{:}0)$

An Important Step of SIKE's Decapsulation

for $i = 0$ **to** $\ell - 1$ **do**

  if $m_i = 1$ **then**

    $((X_0 : Z_0), (X_1 : Z_1)) \leftarrow \texttt{xDBLADD}((X_0 : Z_0), (X_1 : Z_1), (X_2 : Z_2), (a_{24}^{+} : 1))$

  else

    $((X_0 : Z_0), (X_2 : Z_2)) \leftarrow \texttt{xDBLADD}((X_0 : Z_0), (X_2 : Z_2), (X_1 : Z_1), (a_{24}^{+} : 1))$

An Important Step of SIKE's Decapsulation

for $i = 0$ **to** $\ell - 1$ **do**

  if $m_i = 1$ **then**

    $((X_0 : Z_0), (X_1 : Z_1)) \leftarrow \texttt{xDBLADD}((X_0 : Z_0), (X_1 : Z_1), (X_2 : Z_2), (a_{24}^{+} : 1))$

  else

    $((X_0 : Z_0), (X_2 : Z_2)) \leftarrow \texttt{xDBLADD}((X_0 : Z_0), (X_2 : Z_2), (X_1 : Z_1), (a_{24}^{+} : 1))$

T or O

An Important Step of SIKE's Decapsulation

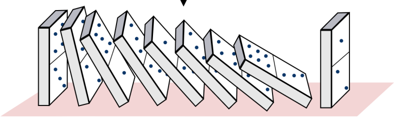
for $i = 0$ **to** $\ell - 1$ **do**

  if $m_i = 1$ **then**

    $((X_0 : Z_0), (X_1 : Z_1)) \leftarrow \texttt{xDBLADD}((X_0 : Z_0), (X_1 : Z_1), (X_2 : Z_2), (a_{24}^{+} : 1))$

  else

    $((X_0 : Z_0), (X_2 : Z_2)) \leftarrow \texttt{xDBLADD}((X_0 : Z_0), (X_2 : Z_2), (X_1 : Z_1), (a_{24}^{+} : 1))$



T or O

Adaptive Chosen-Ciphertext Attack Idea

$$\mu_i = (m_i, \ldots, m_0)_2$$

for $i = 0$ **to** $\ell - 1$ **do**
  if $m_i = 1$ **then**
    $((X_0 : Z_0), (X_1 : Z_1)) \leftarrow \texttt{xDBLADD}((X_0 : Z_0), (X_1 : Z_1), (X_2 : Z_2), (a_{24}^{+} : 1))$
  else
    $((X_0 : Z_0), (X_2 : Z_2)) \leftarrow \texttt{xDBLADD}((X_0 : Z_0), (X_2 : Z_2), (X_1 : Z_1), (a_{24}^{+} : 1))$

Adaptive Chosen-Ciphertext Attack Idea

$$\mu_i = (m_i, \dots, m_0)_2$$

```
for i = 0 to ℓ − 1 do
    if m_i = 1 then
        ((X0 : Z0), (X1 : Z1)) ← xDBLADD((X0 : Z0), (X1 : Z1), (X2 : Z2), (a24+ : 1))
    else
        ((X0 : Z0), (X2 : Z2)) ← xDBLADD((X0 : Z0), (X2 : Z2), (X1 : Z1), (a24+ : 1))
```

μ_{k-1} ✓

Adaptive Chosen-Ciphertext Attack Idea

$$\mu_i = (m_i, \dots, m_0)_2$$

for $i = 0$ **to** $\ell - 1$ **do**
  if $m_i = 1$ **then**
    $((X_0 : Z_0), (X_1 : Z_1)) \leftarrow \texttt{xDBLADD}((X_0 : Z_0), (X_1 : Z_1), (X_2 : Z_2), (a_{24}^{+} : 1))$
  else
    $((X_0 : Z_0), (X_2 : Z_2)) \leftarrow \texttt{xDBLADD}((X_0 : Z_0), (X_2 : Z_2), (X_1 : Z_1), (a_{24}^{+} : 1))$

μ_{k-1} ✓ ⟶ Ciphertext c′

Adaptive Chosen-Ciphertext Attack Idea

$$\mu_i = (m_i, \ldots, m_0)_2$$

for $i = 0$ **to** $\ell - 1$ **do**
  if $m_i = 1$ **then**
    $((X_0 : Z_0), (X_1 : Z_1)) \leftarrow \texttt{xDBLADD}((X_0 : Z_0), (X_1 : Z_1), (X_2 : Z_2), (a_{24}^{+} : 1))$
  else
    $((X_0 : Z_0), (X_2 : Z_2)) \leftarrow \texttt{xDBLADD}((X_0 : Z_0), (X_2 : Z_2), (X_1 : Z_1), (a_{24}^{+} : 1))$

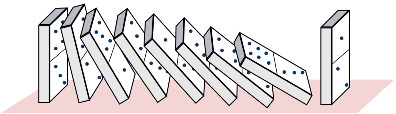
μ_{k-1} ✓ → Ciphertext c′ → $m_k \neq m_{k-1}$ → T

Adaptive Chosen-Ciphertext Attack Idea

$$\mu_i = (m_i, \dots, m_0)_2$$

for $i = 0$ **to** $\ell - 1$ **do**
    if $m_i = 1$ **then**
        $((X_0 : Z_0), (X_1 : Z_1)) \leftarrow \texttt{xDBLADD}((X_0 : Z_0), (X_1 : Z_1), (X_2 : Z_2), (a_{24}^{+} : 1))$
    else
        $((X_0 : Z_0), (X_2 : Z_2)) \leftarrow \texttt{xDBLADD}((X_0 : Z_0), (X_2 : Z_2), (X_1 : Z_1), (a_{24}^{+} : 1))$

μ_{k-1} ✓ → Ciphertext c′ → $m_k \neq m_{k-1}$ → T



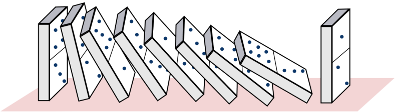
Adaptive Chosen-Ciphertext Attack Idea

$$\mu_i = (m_i, \ldots, m_0)_2$$

for $i = 0$ **to** $\ell - 1$ **do**
  if $m_i = 1$ **then**
    $((X_0 : Z_0), (X_1 : Z_1)) \leftarrow \texttt{xDBLADD}((X_0 : Z_0), (X_1 : Z_1), (X_2 : Z_2), (a_{24}^{+} : 1))$
  else
    $((X_0 : Z_0), (X_2 : Z_2)) \leftarrow \texttt{xDBLADD}((X_0 : Z_0), (X_2 : Z_2), (X_1 : Z_1), (a_{24}^{+} : 1))$

μ_{k-1} ✓ → Ciphertext c′

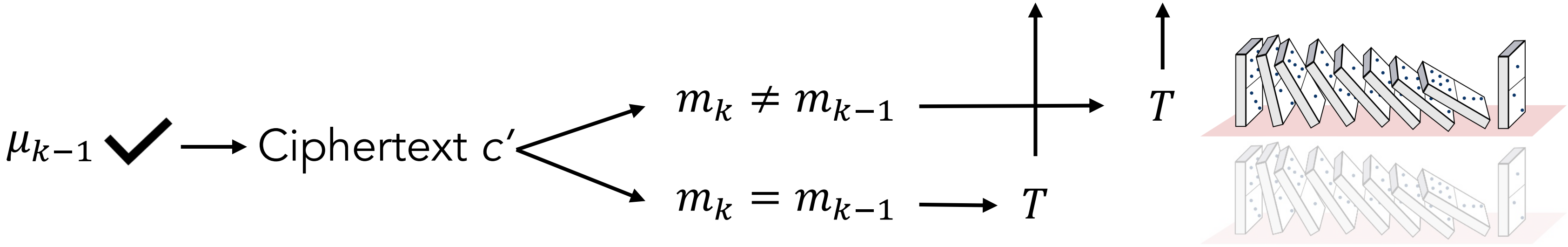
- $m_k \neq m_{k-1}$ → T
- $m_k = m_{k-1}$ → T



Adaptive Chosen-Ciphertext Attack Idea

$$\mu_i = (m_i, \ldots, m_0)_2$$

for $i = 0$ **to** $\ell - 1$ **do**
- **if** $m_i = 1$ **then**
  - $((X_0 : Z_0), (X_1 : Z_1)) \leftarrow \texttt{xDBLADD}((X_0 : Z_0), (X_1 : Z_1), (X_2 : Z_2), (a_{24}^{+} : 1))$
- **else**
  - $((X_0 : Z_0), (X_2 : Z_2)) \leftarrow \texttt{xDBLADD}((X_0 : Z_0), (X_2 : Z_2), (X_1 : Z_1), (a_{24}^{+} : 1))$



Adaptive Chosen-Ciphertext Attack Idea

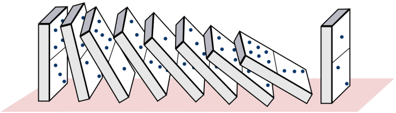
- An attacker who knows the i least significant bits of m (the key) can construct ciphertext c′ such that:

Adaptive Chosen-Ciphertext Attack Idea

- An attacker who knows the *i* least significant bits of *m* (the key) can construct ciphertext c′ such that:
  - If $m_i \neq m_{i-1}$
  - If $m_i = m_{i-1}$

Adaptive Chosen-Ciphertext Attack Idea

- An attacker who knows the *i* least significant bits of *m* (the key) can construct ciphertext c' such that:
  - If $m_i \neq m_{i-1}$

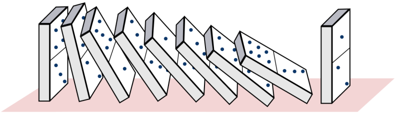


Data flow has low HW and low HD →
lower power consumption →
higher frequency →
shorter runtime!

  - If $m_i = m_{i-1}$

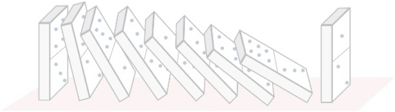
Adaptive Chosen-Ciphertext Attack Idea

- An attacker who knows the *i* least significant bits of *m* (the key) can construct ciphertext c' such that:
  - If $m_i \neq m_{i-1}$



Data flow has low HW and low HD →
lower power consumption →
higher frequency →
shorter runtime!

  - If $m_i = m_{i-1}$

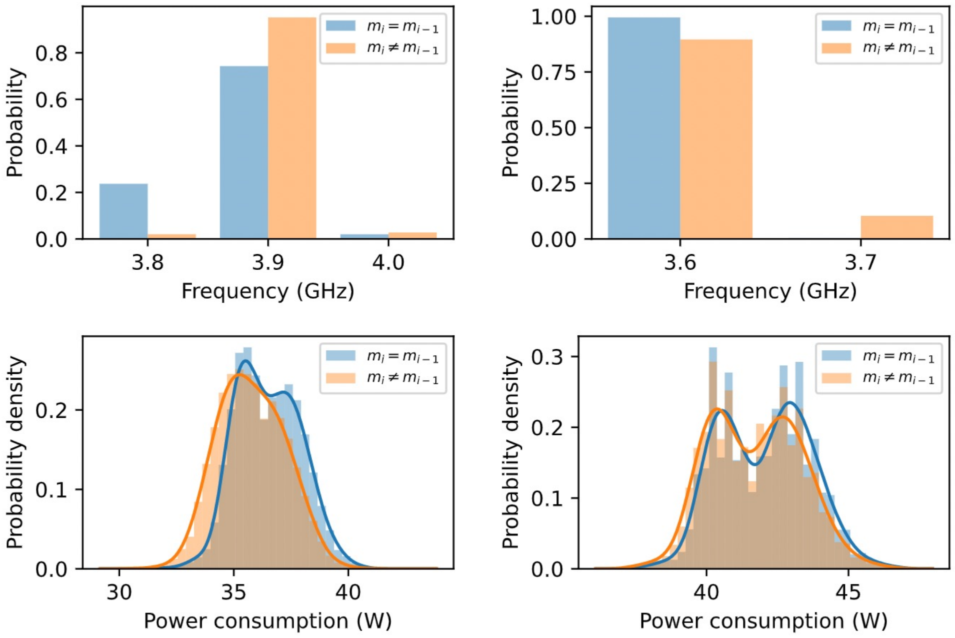


Data flow does not have low HW and low HD
→ higher power consumption →
lower frequency →
longer runtime!

Target Implementation

- Cloudflare's CIRCL (Go)
- Microsoft's PQCrypto-SIDH (C)
  - NIST Post-Quantum Cryptography competition submission

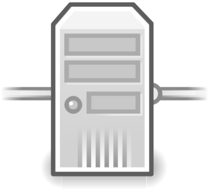
Frequency and Power Measurement



(a) CIRCL data (b) PQCrypto-SIDH data

Remote Timing Attack Model

Client



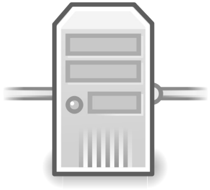
Server

Remote Timing Attack Model

`Decap`(sk, c’)

Ciphertext c’ →

Client



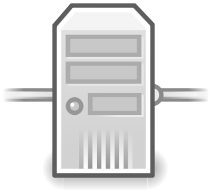
Server

CIRCL
PQCrypto-SIDH

Remote Timing Attack Model

`Decap`(sk, c′)

Ciphertext c′

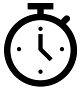


ACK!

Client

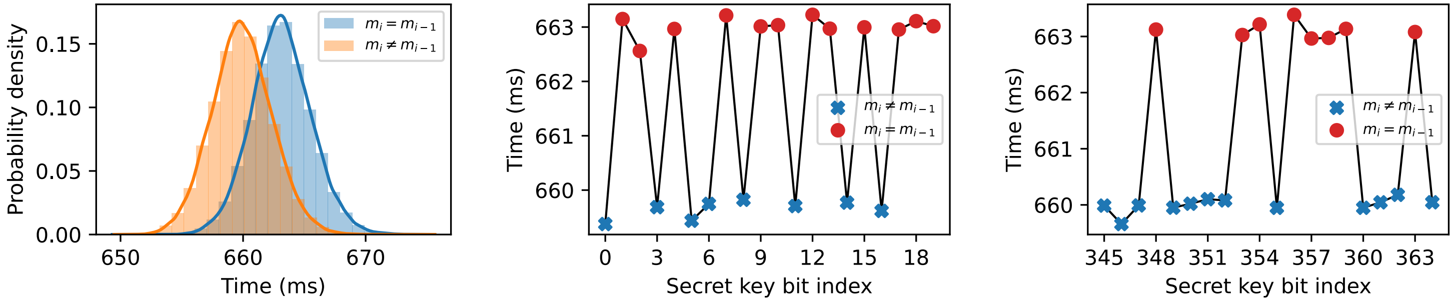
Server

How long it takes to finish *n* concurrent requests



CIRCL
PQCrypto-SIDH

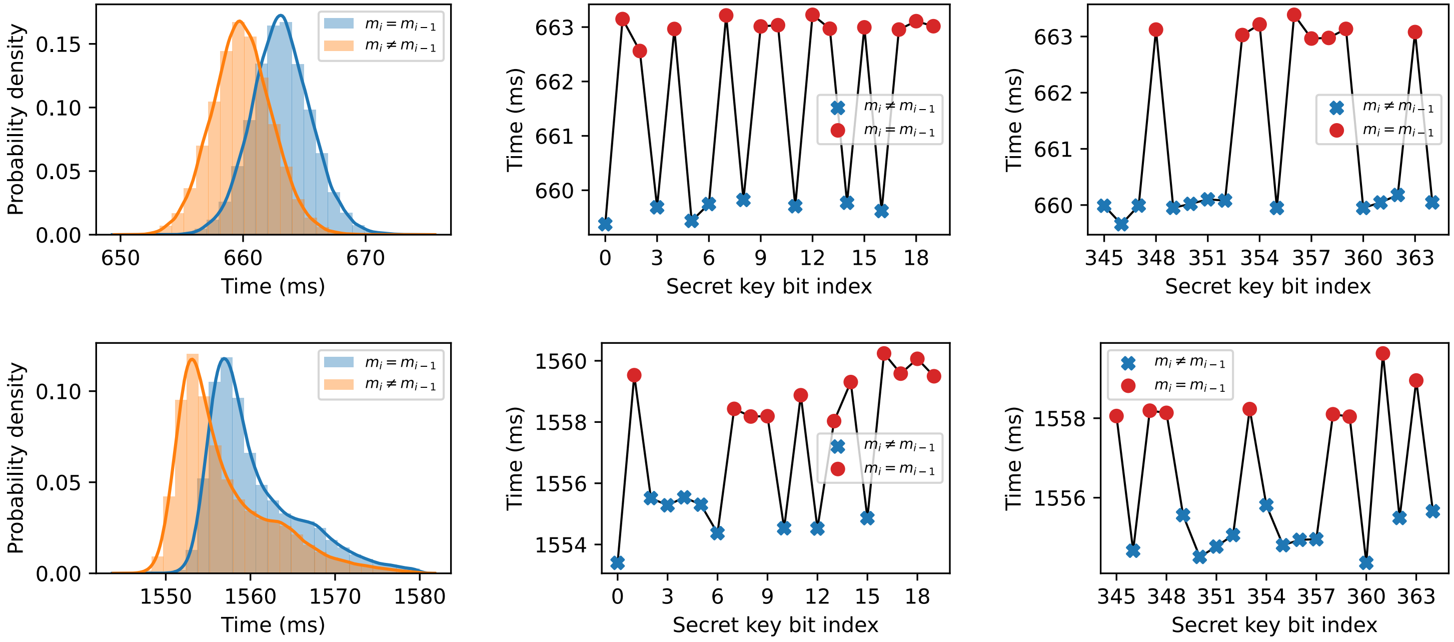
Remote Timing Attack Results



CIRCL:

Recovered full key in 36 hours

Remote Timing Attack Results



CIRCL:
Recovered full key in 36 hours

PQCrypto-SIDH:
Recovered full key in 89 hours

Discussion & Takeaway

Discussion & Takeaway

$$\text{if } secret == 1 \text{ then} \quad routine();$$

No secret-dependent branches

$$state = array[secret]$$

No secret-dependent memory accesses

$$res = x * secret \ / \ 255.0\text{f}$$

No secret inputs to variable-time instructions

Discussion & Takeaway

- Current practices for how to write constant-time code are no longer sufficient to guarantee constant-time execution.

$$\text{if } \mathit{secret} == 1 \text{ then } \mathit{routine}();$$

No secret-dependent branches

$$\mathit{state} = \mathit{array[secret]}$$

No secret-dependent memory accesses

$$\mathit{res} = x * \mathit{secret} \;/\; 255.0\text{f}$$

No secret inputs to variable-time instructions

Discussion & Takeaway

- Current practices for how to write constant-time code are no longer sufficient to guarantee constant-time execution.
- Hertzbleed turns power leakage into timing leakage.

$$\text{if } \mathit{secret} == 1 \text{ then} \quad \mathit{routine}();$$

No secret-dependent branches

$$\mathit{state} = \mathit{array}[\mathit{secret}]$$

No secret-dependent memory accesses

$$\mathit{res} = x * \mathit{secret} \; / \; 255.0\text{f}$$

No secret inputs to variable-time instructions

Discussion & Takeaway

$$\frac{seconds}{program} = \frac{instructions}{program} \times \frac{cycles}{instruction} \times \frac{seconds}{cycle}$$

Discussion & Takeaway

$$\frac{seconds}{program} = \frac{instructions}{program} \times \frac{cycles}{instruction} \times \frac{seconds}{cycle}$$

All prior timing attacks

Discussion & Takeaway

$$\frac{seconds}{program} = \frac{instructions}{program} \times \frac{cycles}{instruction} \times \frac{seconds}{cycle}$$

All prior timing attacks

Hertzbleed

NEW

Conclusion

- Frequency leaks information about the data values being processed.
- SIKE is vulnerable to a new CCA attack that can be exploited remotely using Hertzbleed.
- Current practices for how to write constant-time code are no longer sufficient to guarantee constant time execution.

www.hertzbleed.com

FPSG-UIUC hertzbleed | Stars 406

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED