Hertzbleed: The Claim of Constant-time is Frequently Wrong

Yingchen Wang, Riccardo Paccagnella, Alan Wandke, Zhao Gang, Grant Garrett-Grossman, Christopher W. Fletcher, David Kohlbrenner, Hovav Shacham

TEXAS
The University of Texas at Austin

ILLINOIS

UNIVERSITY of WASHINGTON

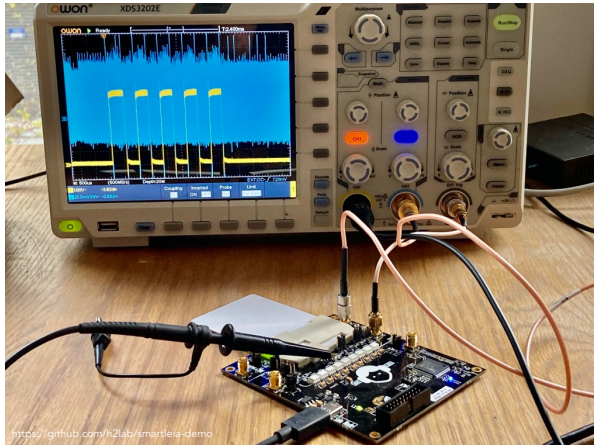
Berkeley
UNIVERSITY OF CALIFORNIA

Carnegie Mellon University

Power Side Channel vs Remote Timing

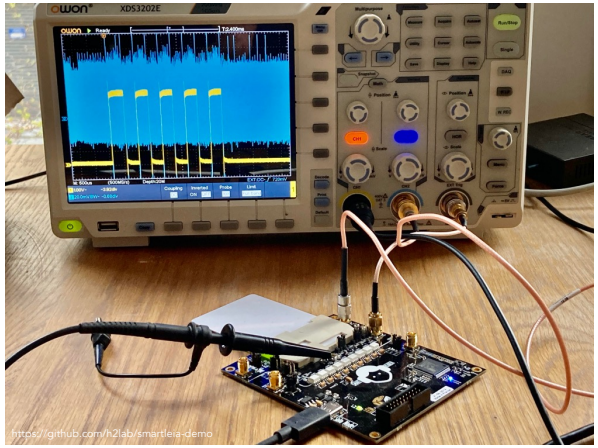
Power Side Channel vs Remote Timing

Power Side-Channel Attacks

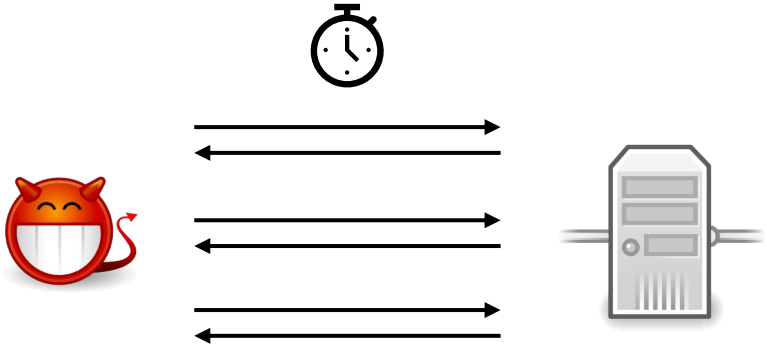


Power Side Channel vs Remote Timing

Power Side-Channel Attacks

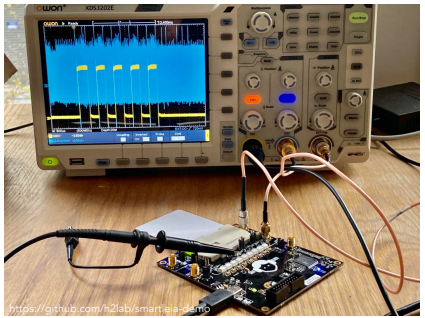


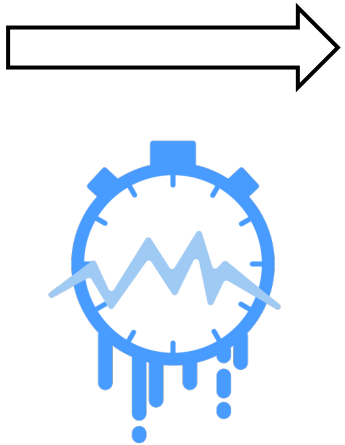
Remote Timing Attacks



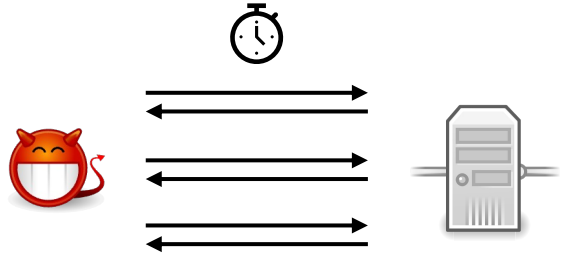
Hertzbleed: a New Class of Attacks

Power Side-Channel Attacks





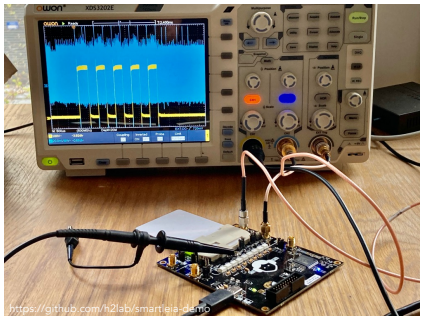
Remote Timing Attacks

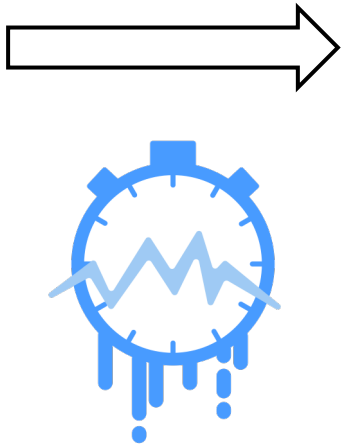


Hertzbleed: enable remote key extraction from constant-time cryptography implementation.

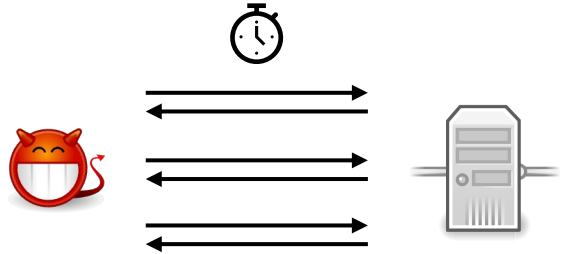
Hertzbleed: a New Class of Attacks

Power Side-Channel Attacks





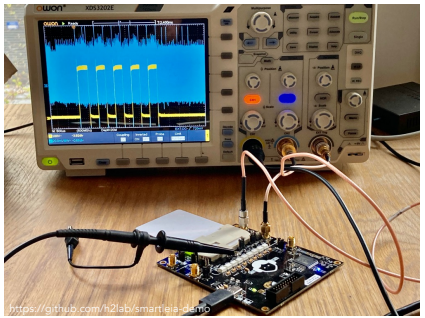
Remote Timing Attacks

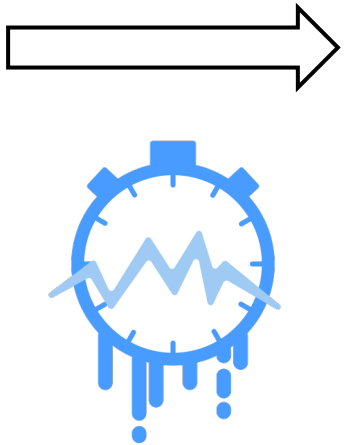


Hertzbleed: exploiting
dynamic voltage and frequency scaling (DVFS)

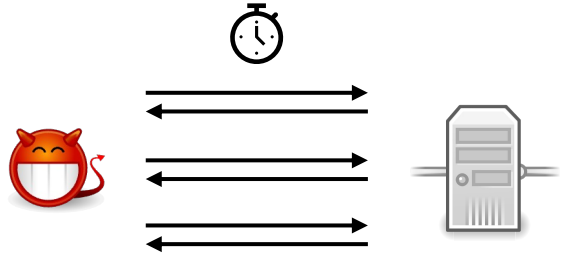
Hertzbleed: a New Class of Attacks

Power Side-Channel Attacks



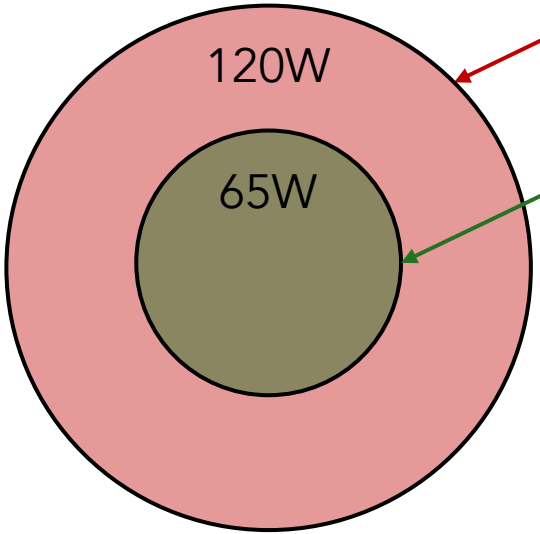


Remote Timing Attacks



Hertzbleed: Re-think the definition of constant-time programming

DVFS on a modern Intel CPU



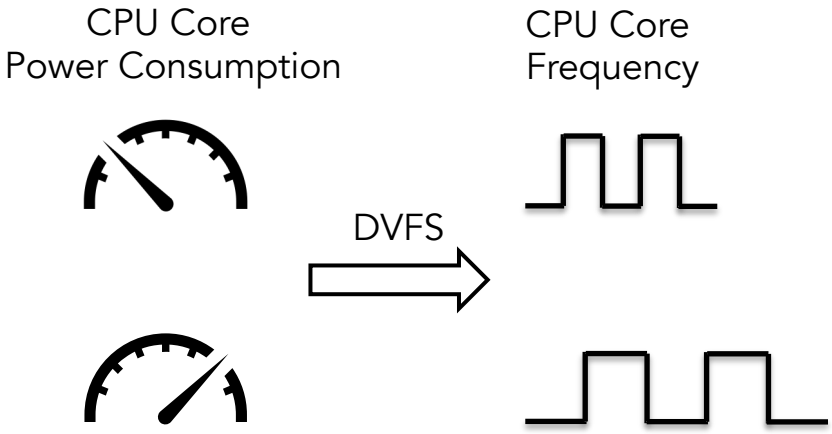
Maximum Power Draw

Thermal Design Point (TDP)

DVFS is the manager that ensures the CPU stays below the TDP.

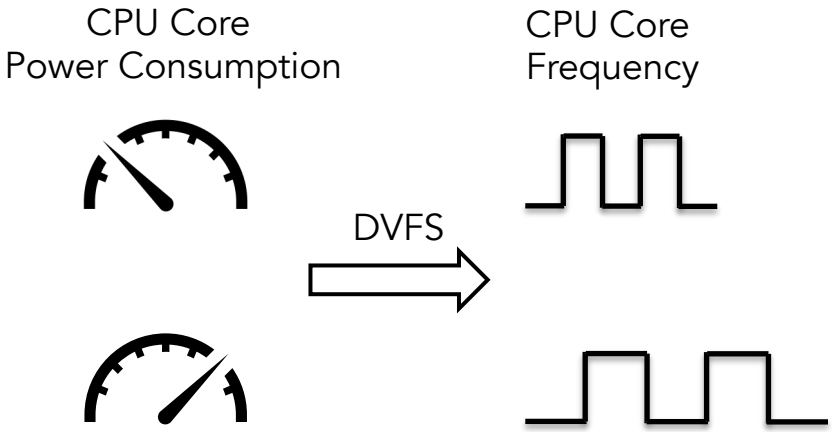
Intel i7-8700

DVFS on a modern Intel CPU



DVFS is the manager that ensures the CPU stays below the TDP.

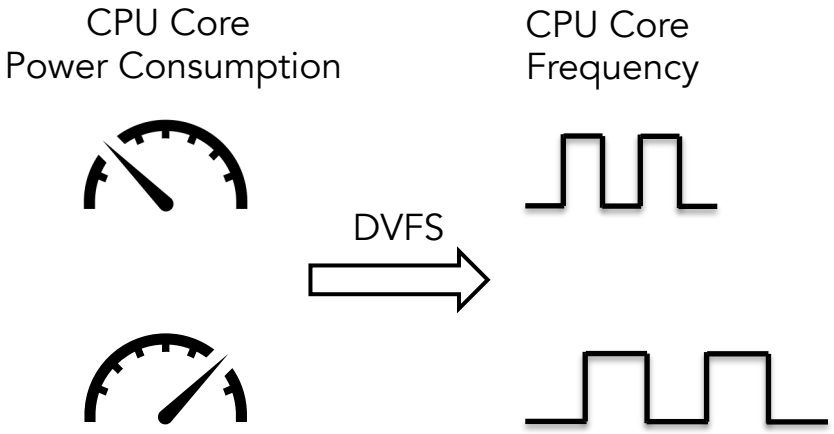
DVFS on a modern Intel CPU



DVFS is the manager that ensures the CPU stays below the TDP.

Why DVFS has anything to do with constant-time cryptography?

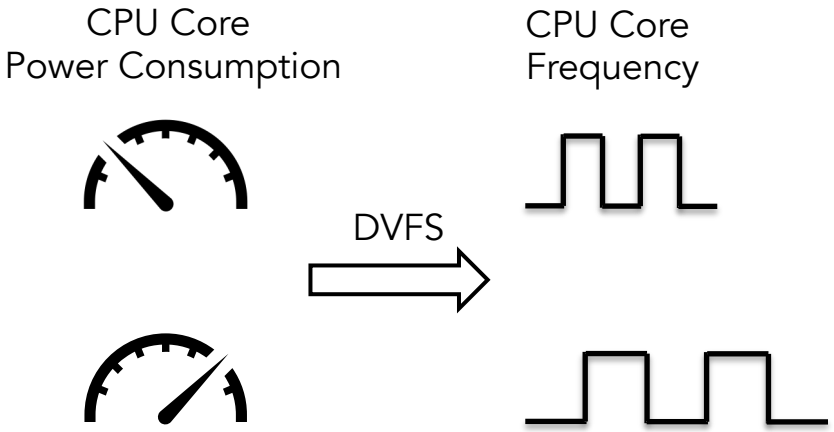
DVFS on a modern Intel CPU



Why DVFS has anything to do with constant-time cryptography?

Power leaks data!

DVFS on a modern Intel CPU



Why DVFS has anything to do with constant-time cryptography?

Power leaks data!

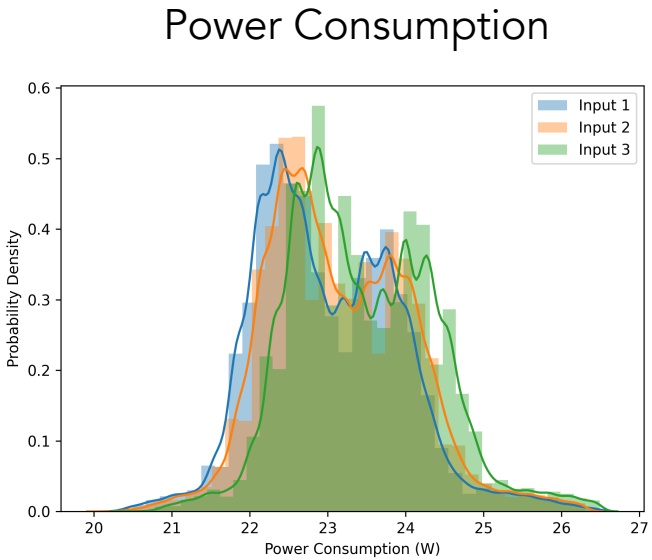
Does frequency also leak data?

Frequency Depends on Data

- Vary the data values (Input) being processed in a “constant-time” workload.

Frequency Depends on Data

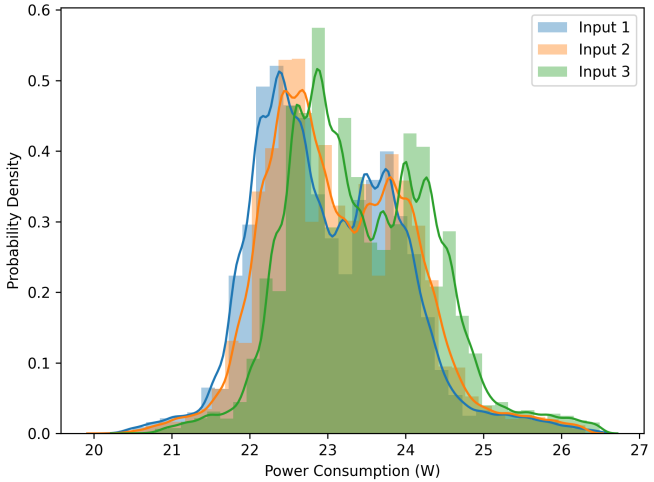
- Vary the data values (Input) being processed in a “constant-time” workload.



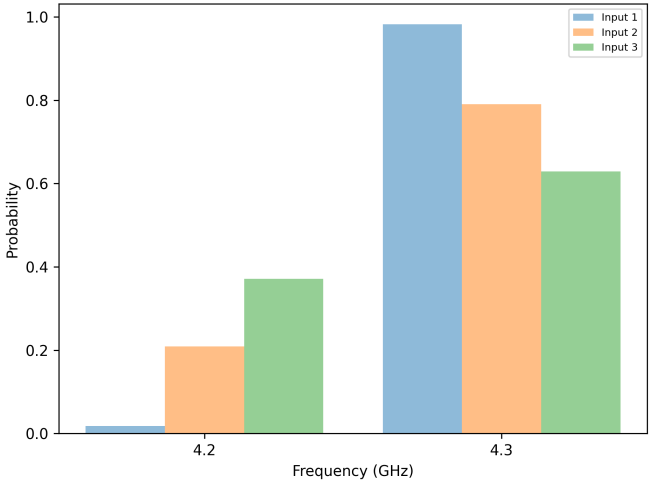
Frequency Depends on Data

- Vary the data values (Input) being processed in a “constant-time” workload.

Power Consumption



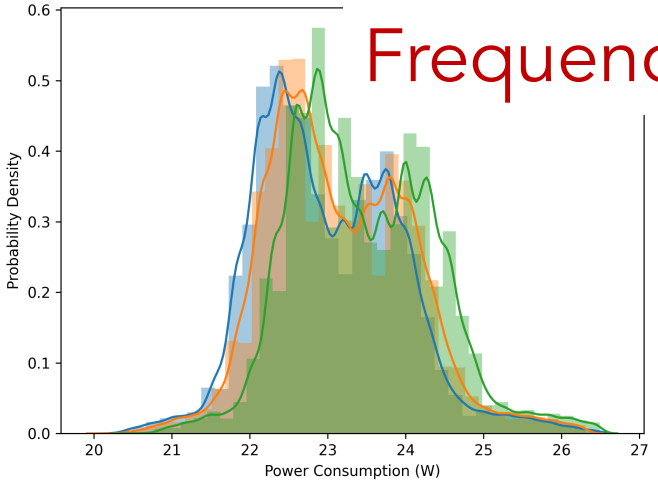
CPU Frequency



Frequency Depends on Data

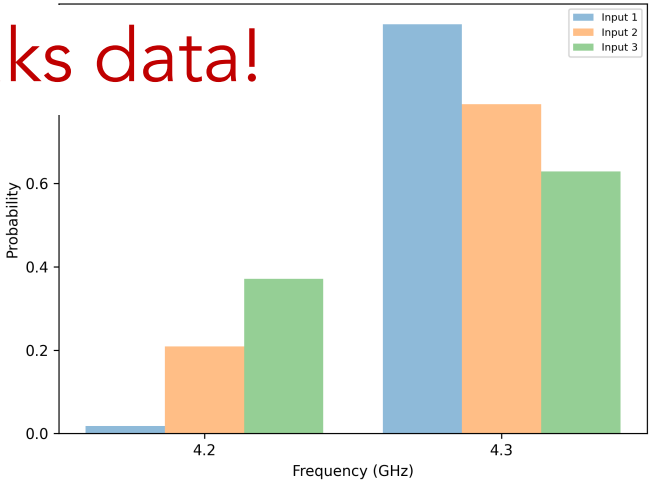
- Vary the data values (Input) being processed in a “constant-time” workload.

Power Consumption



Frequency leaks data!

CPU 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

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

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

```
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!

```
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

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!**

Consumes different amount of CPU power

Runs at a different CPU frequency!

```
rax ← 0x00000000000000ff
rax ← rax | rax
```

HW = 8, HD = 0

Frequency Leakage Model

Three *independent* effects:

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

Consumes different amount of CPU power

Runs at a different CPU frequency!

```
rax ← 0x000000000000ff00
rax ← rax | rax
```

HW = 8, HD = 0

Frequency Leakage Model

Three *independent* effects:

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

Consumes different amount of CPU power

Runs at a different CPU frequency!

```
rax ← 0x00000000ff0000
rax ← rax | rax
```

HW = 8, HD = 0

Frequency Leakage Model

Three *independent* effects:

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

Consumes different amount of CPU power

Runs at a different CPU frequency!

```
rax ← 0x00000000ff000000
rax ← rax | rax
```

HW = 8, HD = 0

Frequency Leakage Model

Three *independent* effects:

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

Consumes different amount of CPU power

Runs at a different CPU frequency!

```
rax ← 0x000000ff00000000
rax ← rax | rax
```

HW = 8, HD = 0

Frequency Leakage Model

Three *independent* effects:

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

Consumes different amount of CPU power

Runs at a different CPU frequency!

```
rax ← 0x0000ff0000000000
rax ← rax | rax
```

HW = 8, HD = 0

Frequency Leakage Model

Three *independent* effects:

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

Consumes different amount of CPU power

Runs at a different CPU frequency!

```
rax ← 0x00ff000000000000
rax ← rax | rax
```

HW = 8, HD = 0

Frequency Leakage Model

Three *independent* effects:

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

Consumes different amount of CPU power

Runs at a different CPU frequency!

```
rax ← 0xff00000000000000
rax ← rax | rax
```

HW = 8, HD = 0

Frequency Leakage Model

Three *independent* effects:

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

CPU frequency can leak information about data even with a fixed HD and HW.

Consumes different amount of CPU power

Runs at a different CPU frequency!

```
rax ← 0xffffffffffffffff
rax ← rax | rax
```

HW = 8, HD = 0

Hertzbleed: CPU Frequency Depends on CPU Power

A constant-time program with different secret inputs

CPU Power Consumption

Higher Power

CPU Frequency

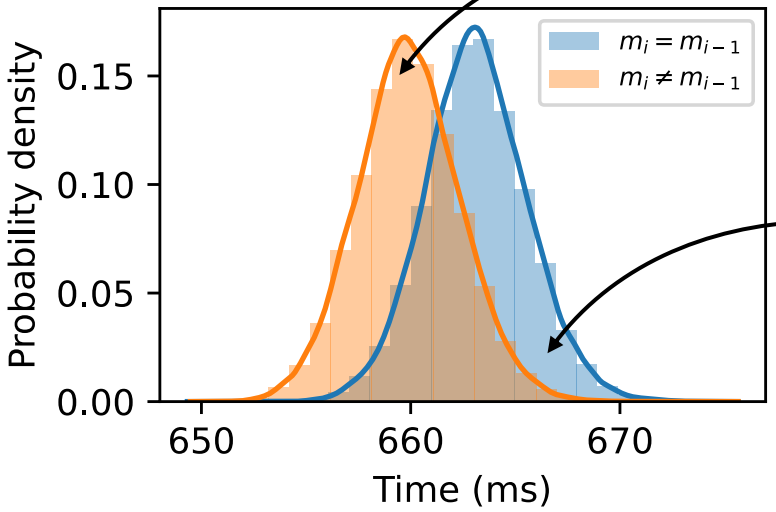
Lower Frequency

Program Execution Time

Longer Time

Hertzbleed: Remote Key Extraction

Constant-time cryptography implementation

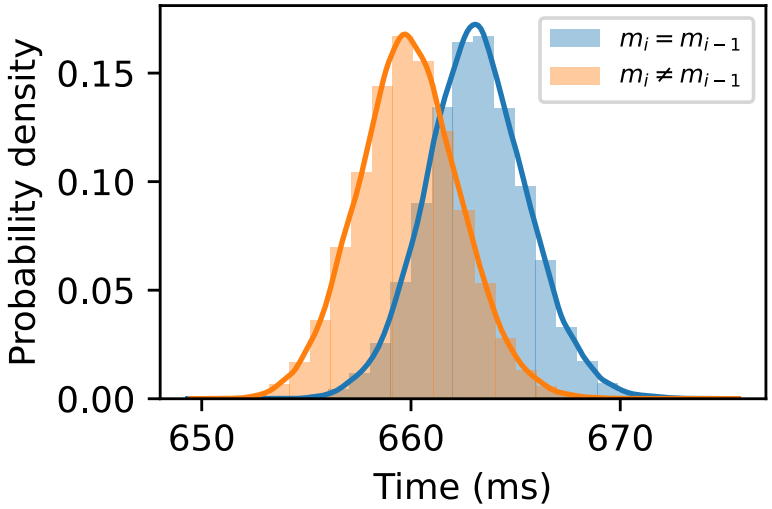


Attacker made a correct guess of a secret key bit.

Attacker made an incorrect guess of a secret key bit.

[1]Supersingular Isogeny Key Encapsulation

Hertzbleed: Remote Key Extraction

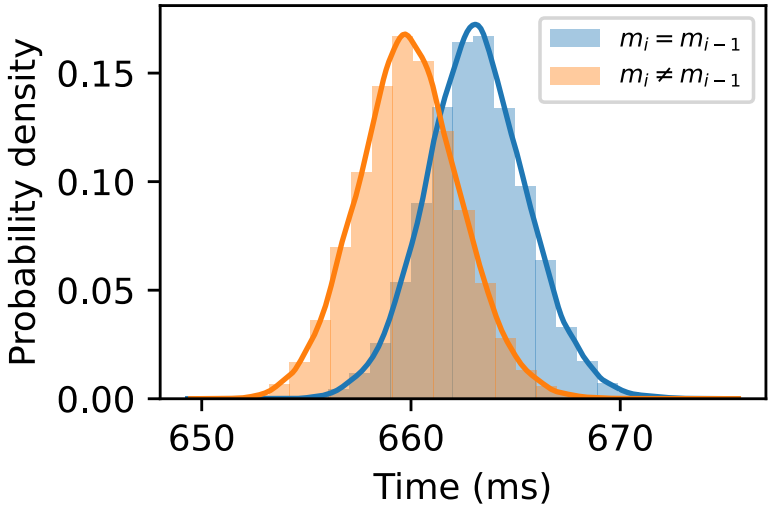


SIKE[1] has been deprecated

Is SIKE the only cryptosystem vulnerable to Hertzbleed?

[1]Supersingular Isogeny Key Encapsulation

Hertzbleed: Remote Key Extraction



SIKE[1] has been deprecated

Is every cryptosystem vulnerable to Hertzbleed?

[1]Supersingular Isogeny Key Encapsulation

Which universe do we live in?

SIKE[1] is the only cryptosystem vulnerable to Hertzbleed.

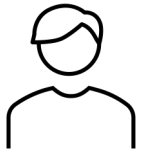
Every cryptosystem is vulnerable to Hertzbleed.

[1]Supersingular Isogeny Key Encapsulation

Which universe do we live in?

SIKE[1] is the only cryptosystem vulnerable to Hertzbleed.

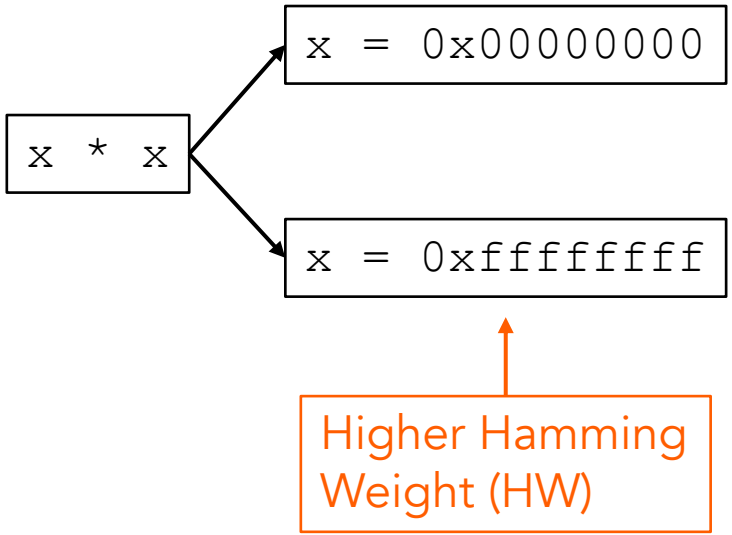
Every cryptosystem is vulnerable to Hertzbleed.



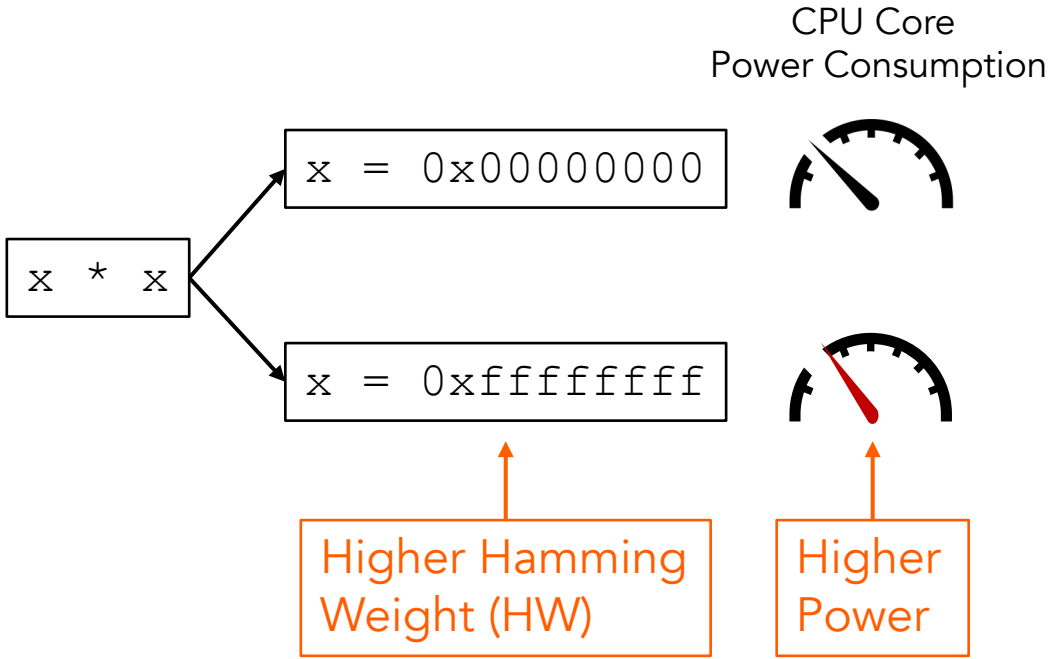
[1]Supersingular Isogeny Key Encapsulation

Hertzbleed: Amplification Gadget

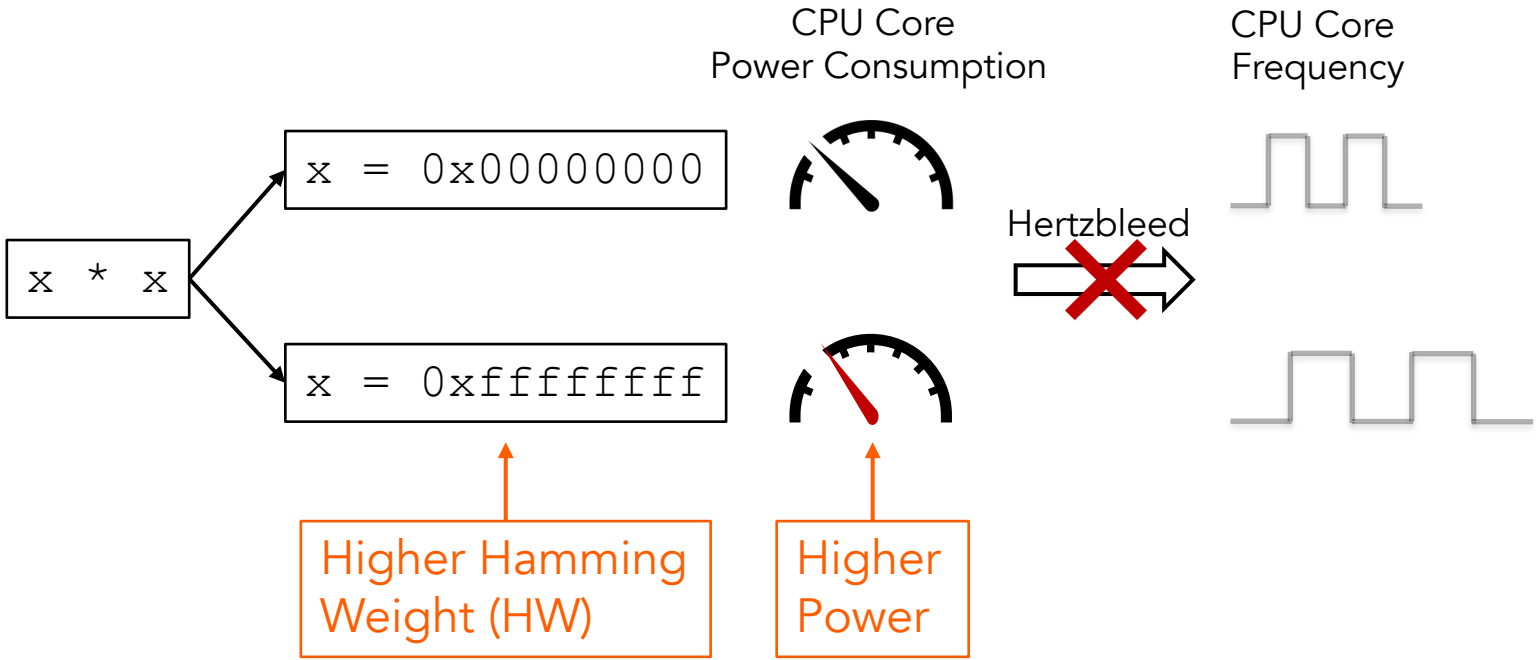
Power leakage and frequency leakage are not equivalent.



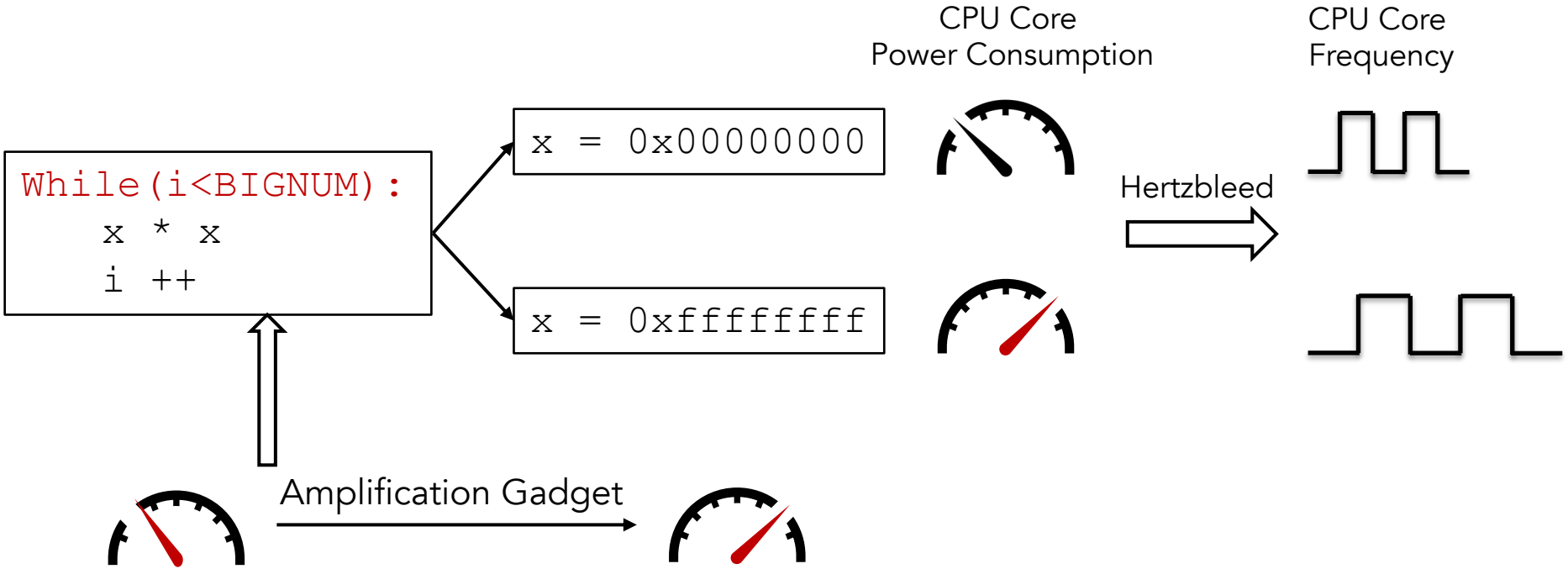
Hertzbleed: Amplification Gadget



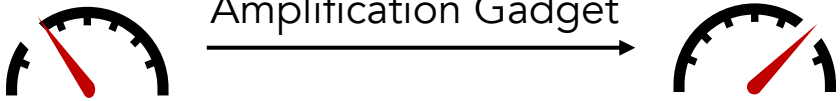
Hertzbleed: Amplification Gadget



Hertzbleed: Amplification Gadget



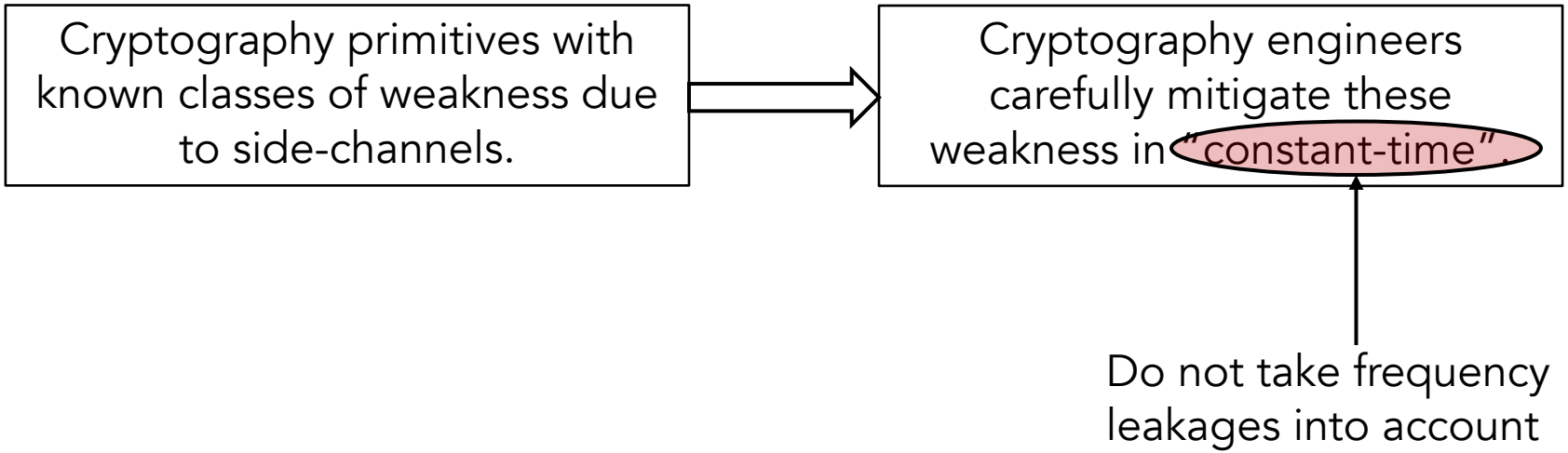
Hertzbleed: Amplification Gadget



No systematic amplification gadget discovery methodology

WPH+2022: Hertzbleed: Turning Power Side-Channel Attacks Into Remote Timing Attacks on x86

Hertzbleed: Amplification Gadget



Hertzbleed: Amplification Gadget

Cryptography primitives with known classes of weakness due to side-channels.

⇒

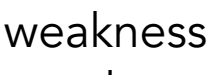
Cryptography engineers carefully mitigate these weakness in "constant-time".

Will mitigated side-channel leakage reappear if examined through a Hertzbleed lens?

WPW+2023: DVFS Frequently Leaks Secrets: Hertzbleed Attacks Beyond SIKE, Cryptography, and CPU-Only Data

Hertzbleed: Amplification Gadget

Cryptography primitives with known classes of weakness due to side-channels.

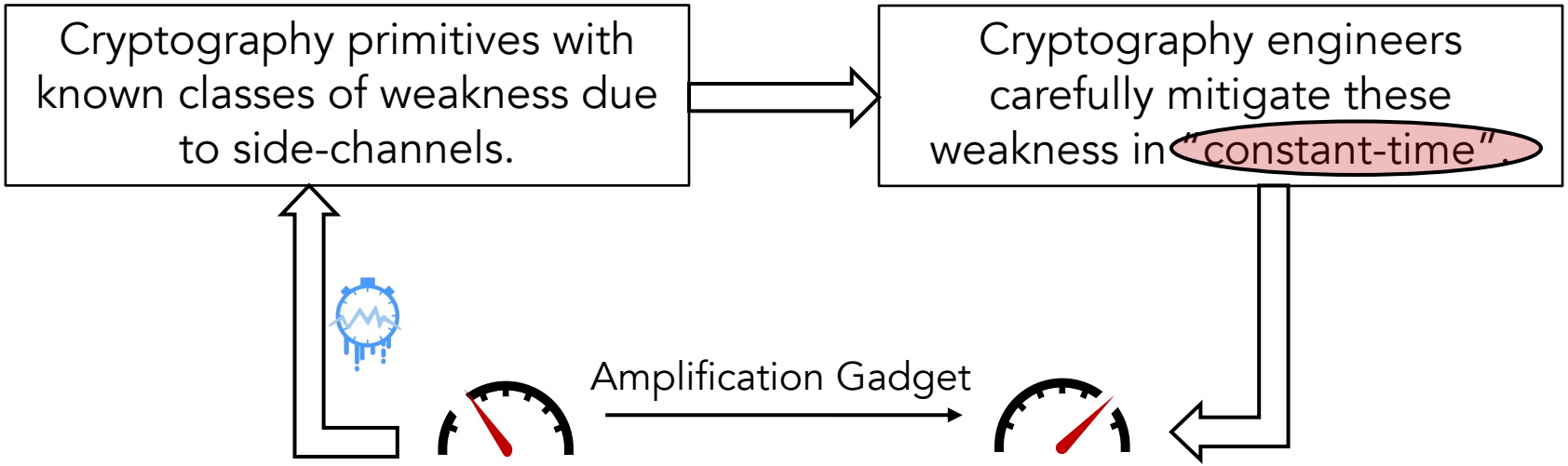


Cryptography engineers carefully mitigate these weakness in "constant-time".

Mitigated side-channel weakness do reappear when looking through a Hertzbleed lens.

WPW+2023: DVFS Frequently Leaks Secrets: Hertzbleed Attacks Beyond SIKE, Cryptography, and CPU-Only Data

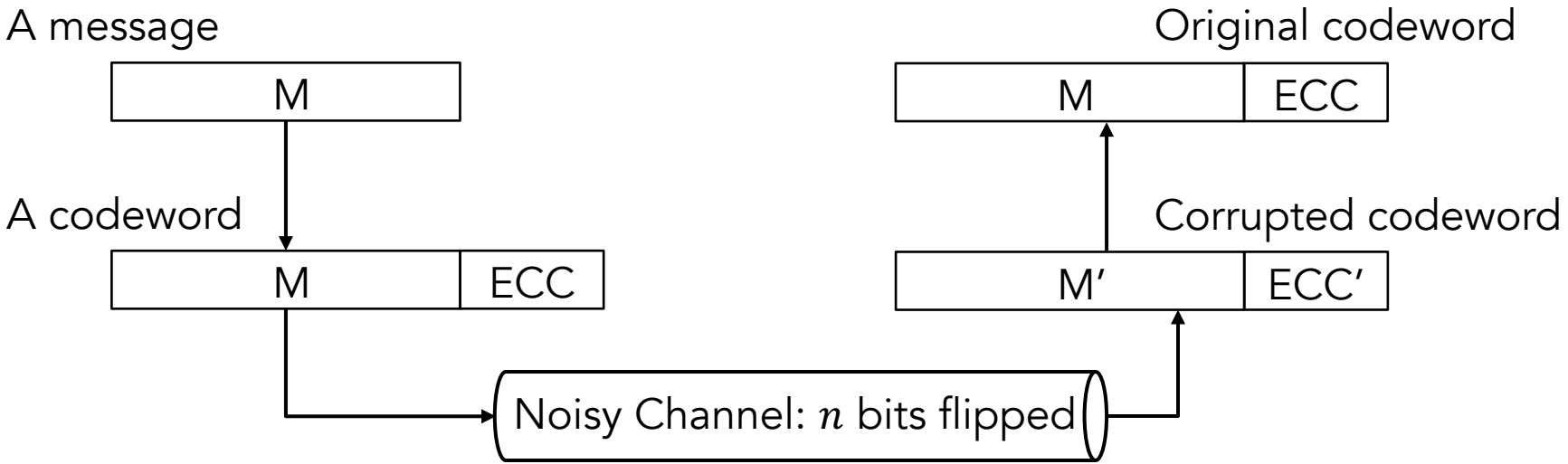
Hertzbleed: Amplification Gadget



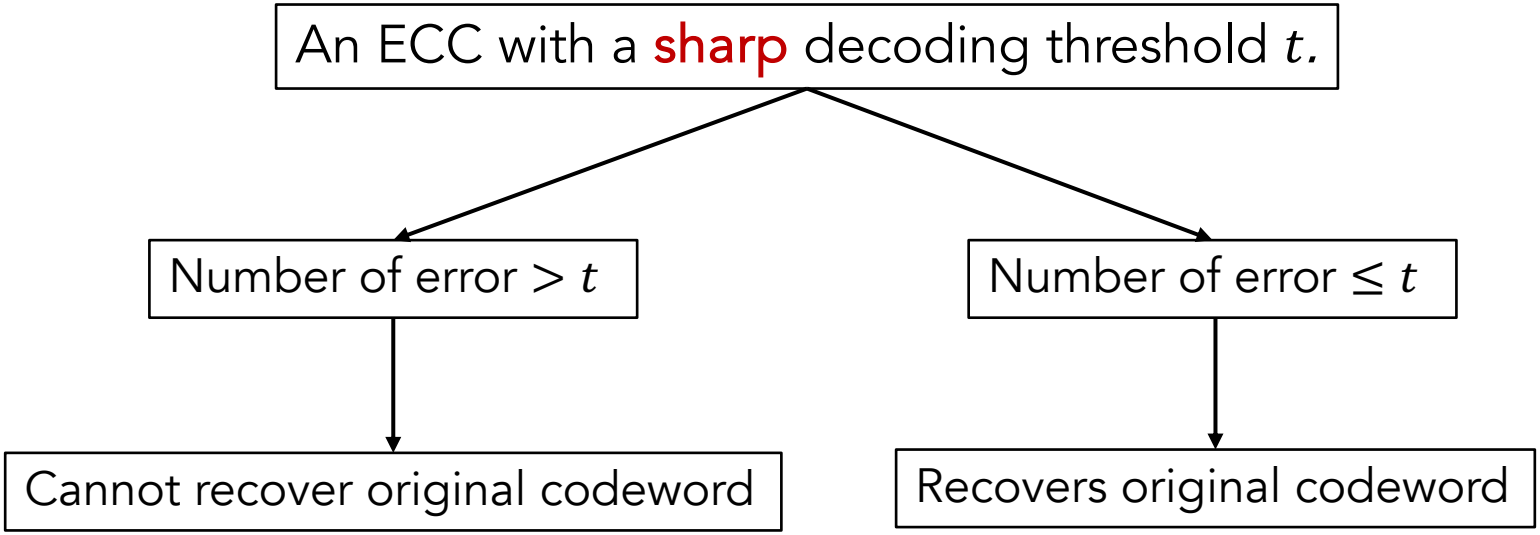
WPW+2023: DVFS Frequently Leaks Secrets: Hertzbleed Attacks Beyond SIKE, Cryptography, and CPU-Only Data

Background: Error correcting code

Systematic error correcting code (ECC): Encoding a message with redundancy for detecting and recovering errors.

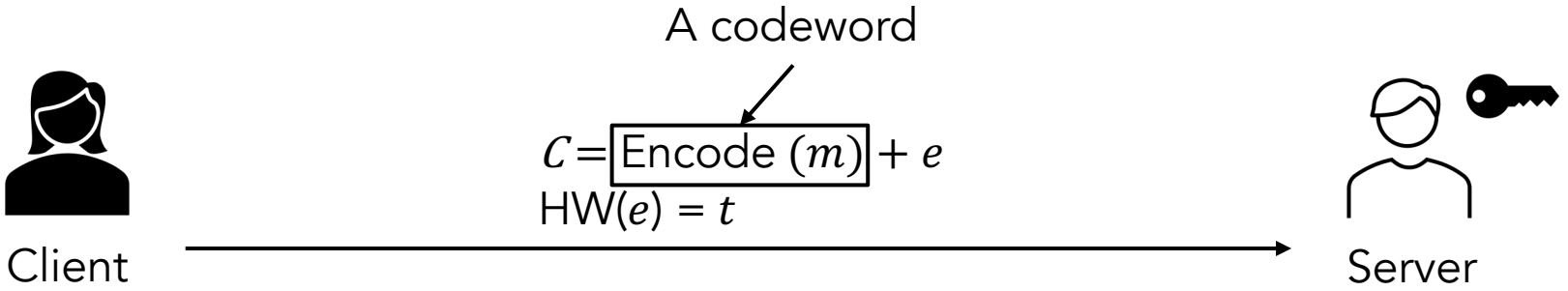


Background: Binary Goppa Code



Background: McEliece Public-Key Cryptosystem

Client: pick a codeword m, and secret error vector e with HW(e) = t.

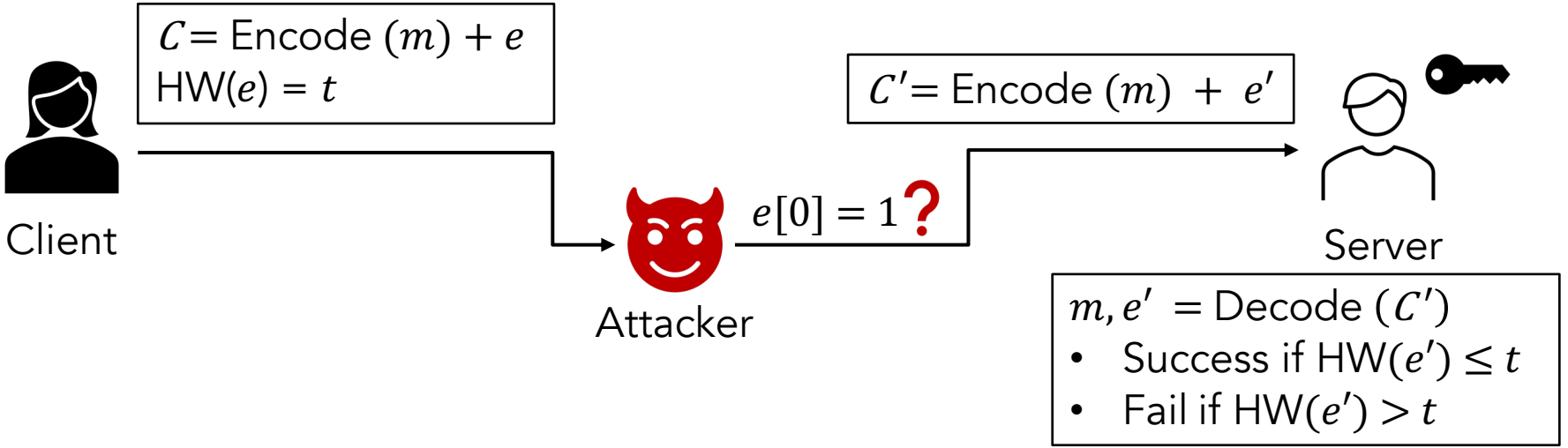


Anyone without the secret key
cannot decode C to m = cannot recover e from C.

m, e = Decode (C)

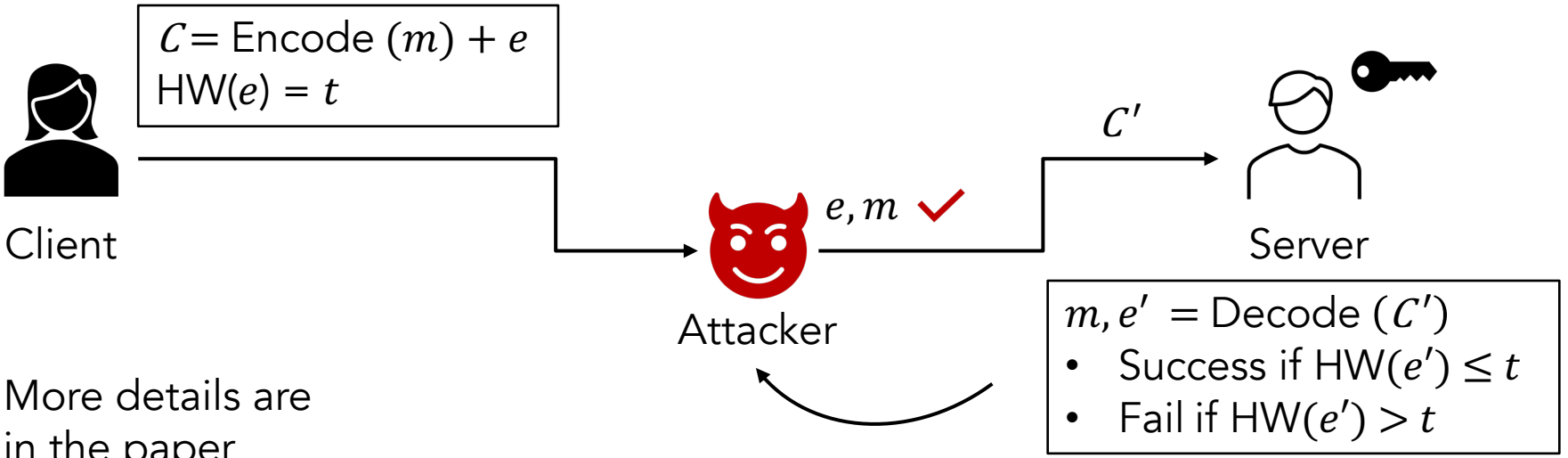
Sloppy Alice Attack on McEliece Public Key Cryptosystem

Threat model: MITM attacker attempts to recover e, and then computes m.



Sloppy Alice attacks! Adaptive chosen ciphertext attacks on the McEliece Public-Key Cryptosystem

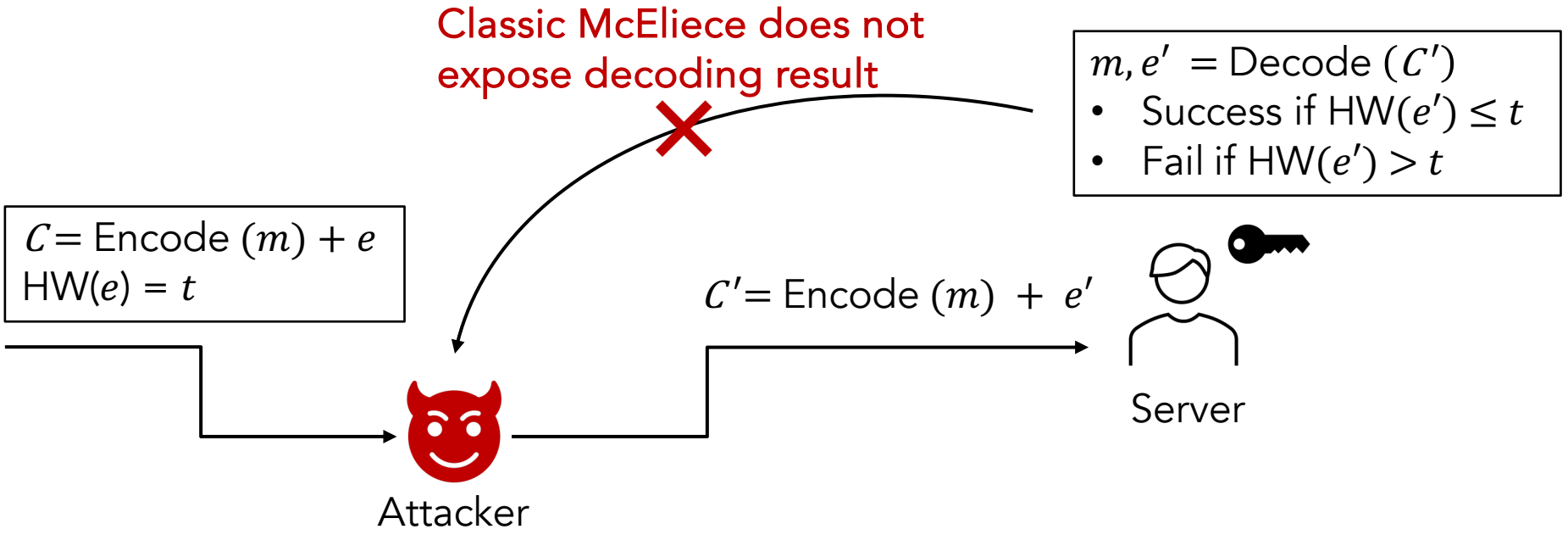
Sloppy Alice Attack on McEliece Public Key Cryptosystem



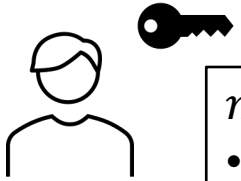
More details are in the paper

Sloppy Alice attacks! Adaptive chosen ciphertext attacks on the McEliece Public-Key Cryptosystem

Classic McElice: A KEM on top of the McEliece Public Key Cryptosystem



Hertzbleed Attack on Classic McEliece

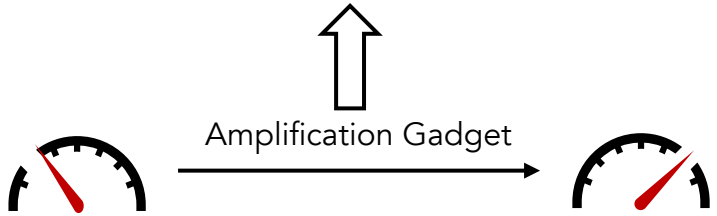


Server

$m, e' = \text{Decode}(C')$
- Success if $\text{HW}(e') \leq t$
- Fail if $\text{HW}(e') > t$

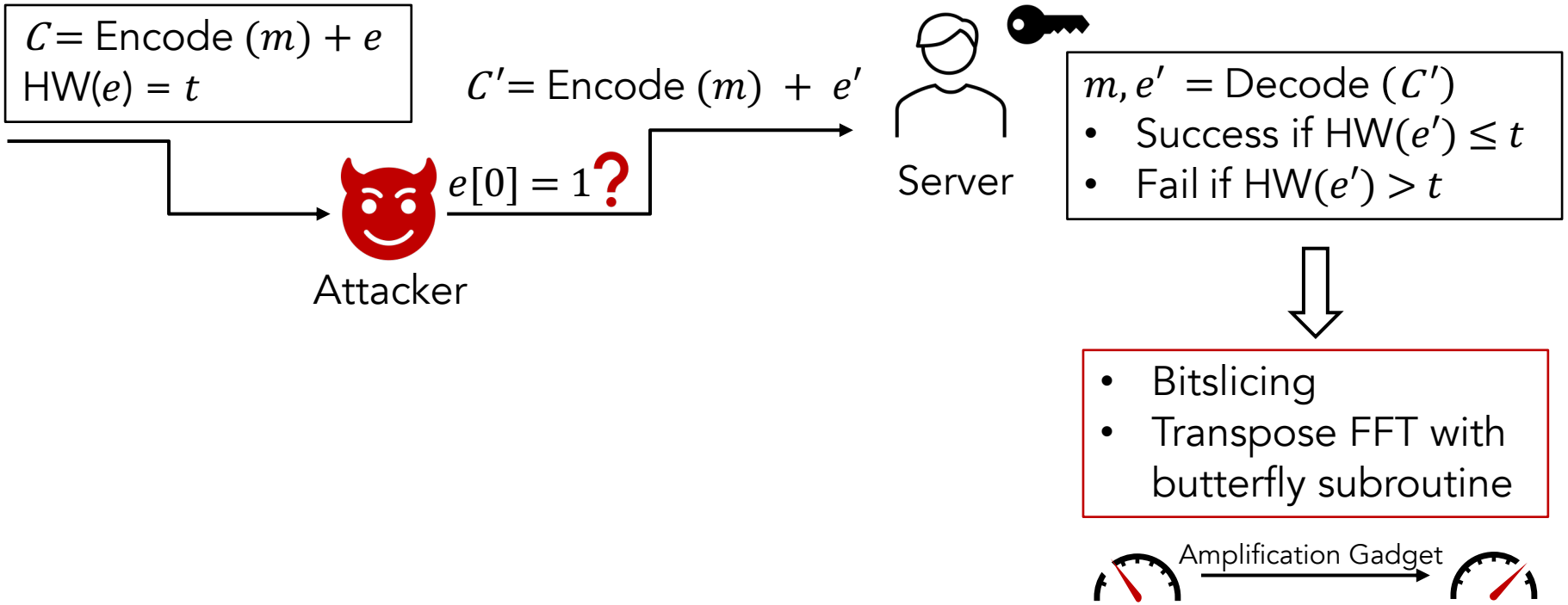
- Valid C
- Malformed C': $\text{HW}(e') \leq t$
- Malformed C': $\text{HW}(e') > t$

Classic McEliece does a hamming weight checking and re-encryption to reject any malformed ciphertext.



A decode oracle via Hertzbleed

Hertzbleed Attack on Classic McEliece

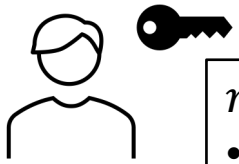


Hertzbleed Attack on Classic McEliece

C = Encode (m) + e
HW(e) = t

C' = Encode (m) + e'

$e[0] = 1$?



Server

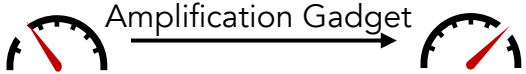
m, e' = Decode (C')
- Success if HW$(e') \leq t$
- Fail if HW$(e') > t$

Attacker

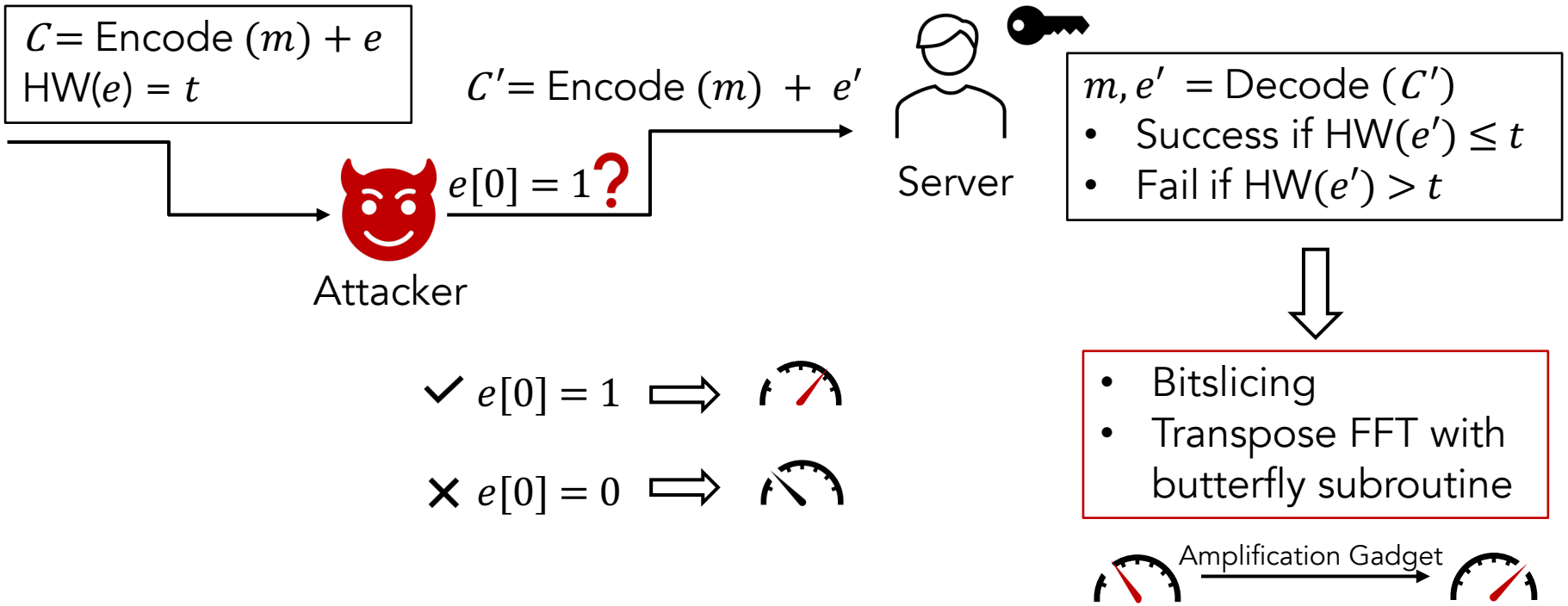
✓ $e[0] = 1$ ⟹ High HW&HD

✗ $e[0] = 0$ ⟹ Low HW&HD

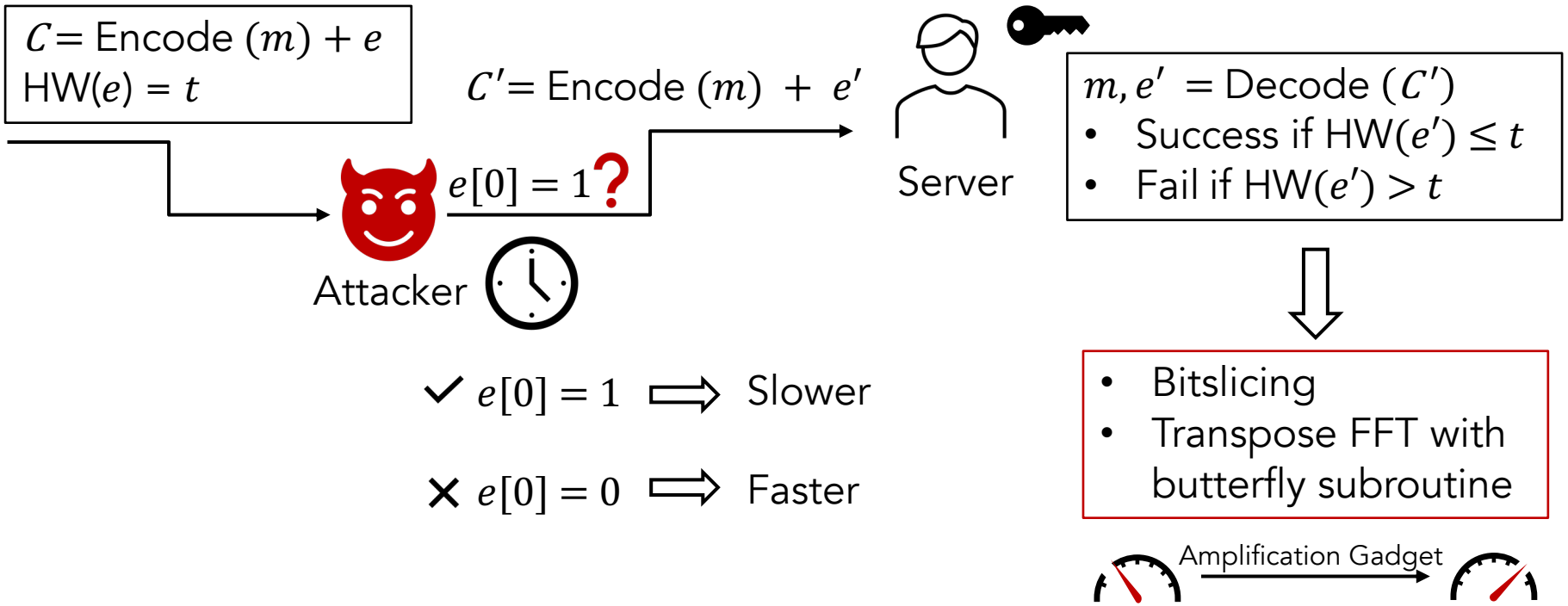
- Bitslicing
- Transpose FFT with butterfly subroutine



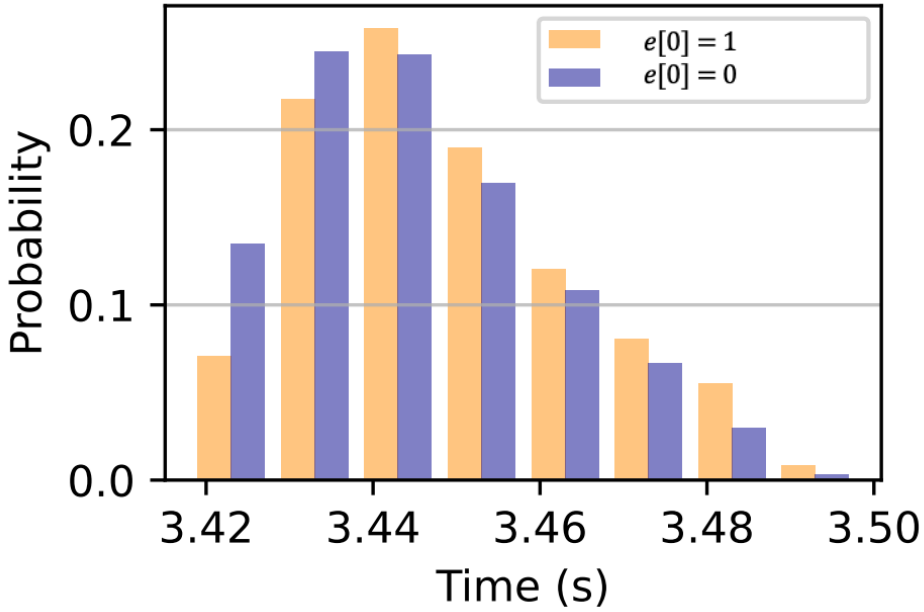
Hertzbleed Attack on Classic McEliece



Hertzbleed Attack on Classic McEliece



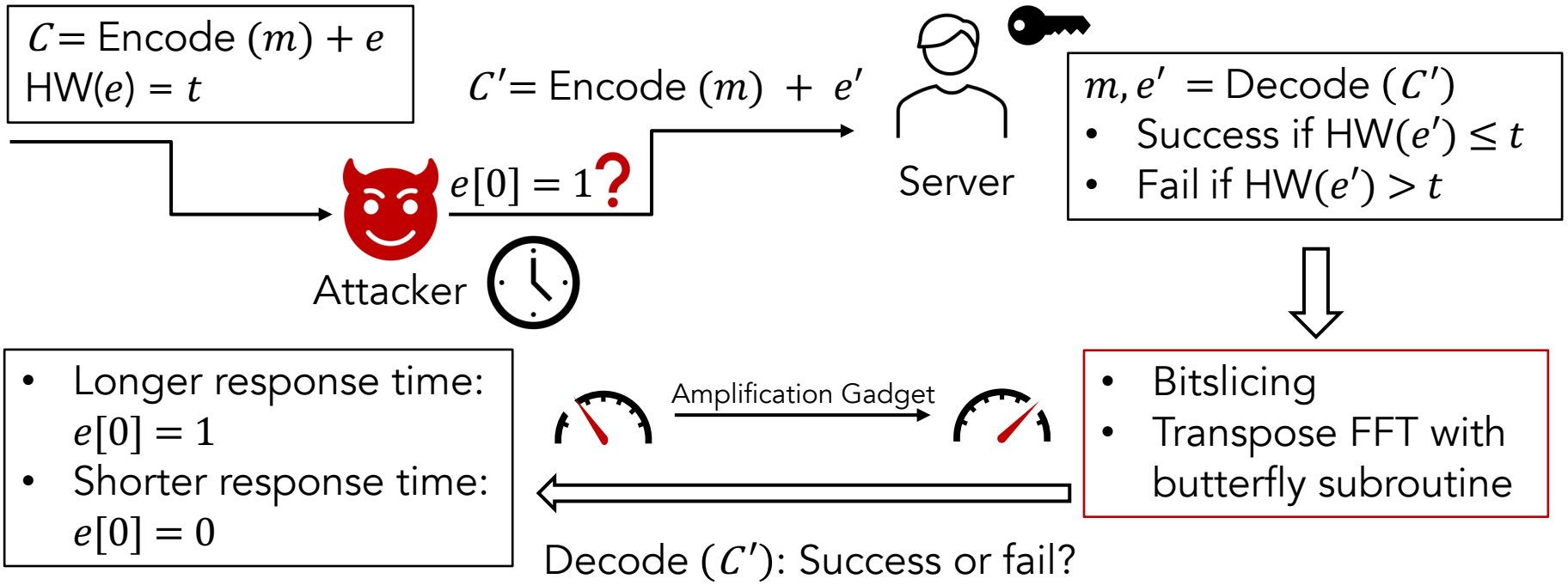
Hertzbleed Attack on Classic McEliece



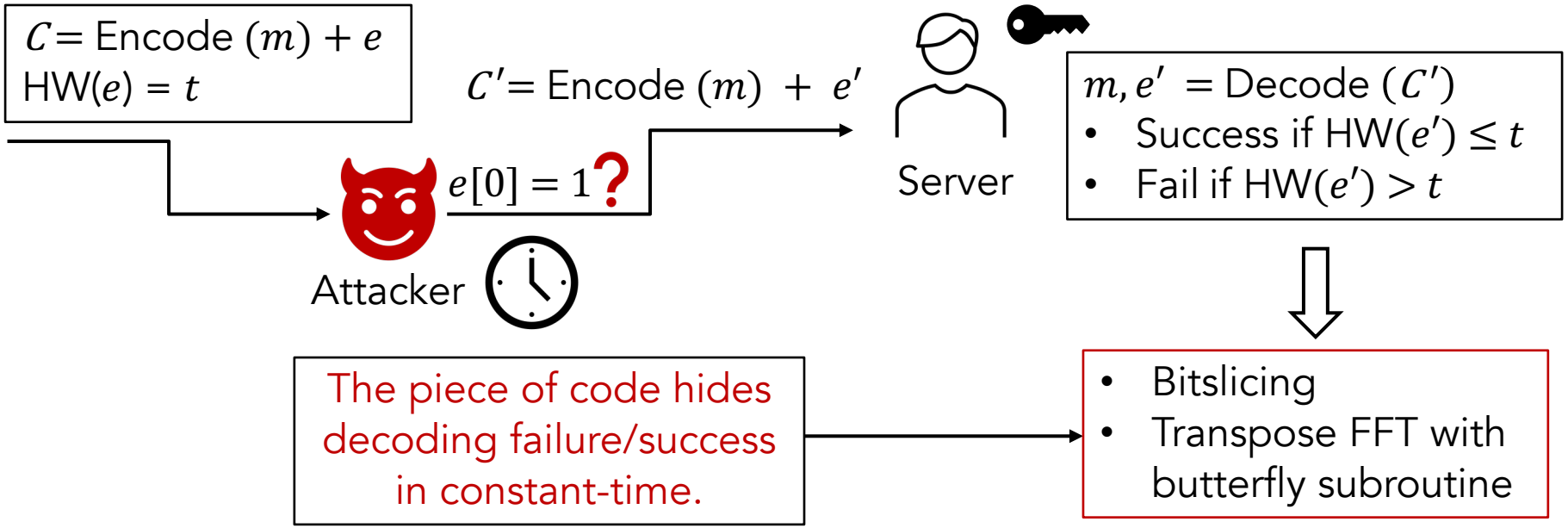
Classic McEliece decapsulation timing distribution:

- Orange: $e[0] = 1$
- Blue: $e[0] = 0$

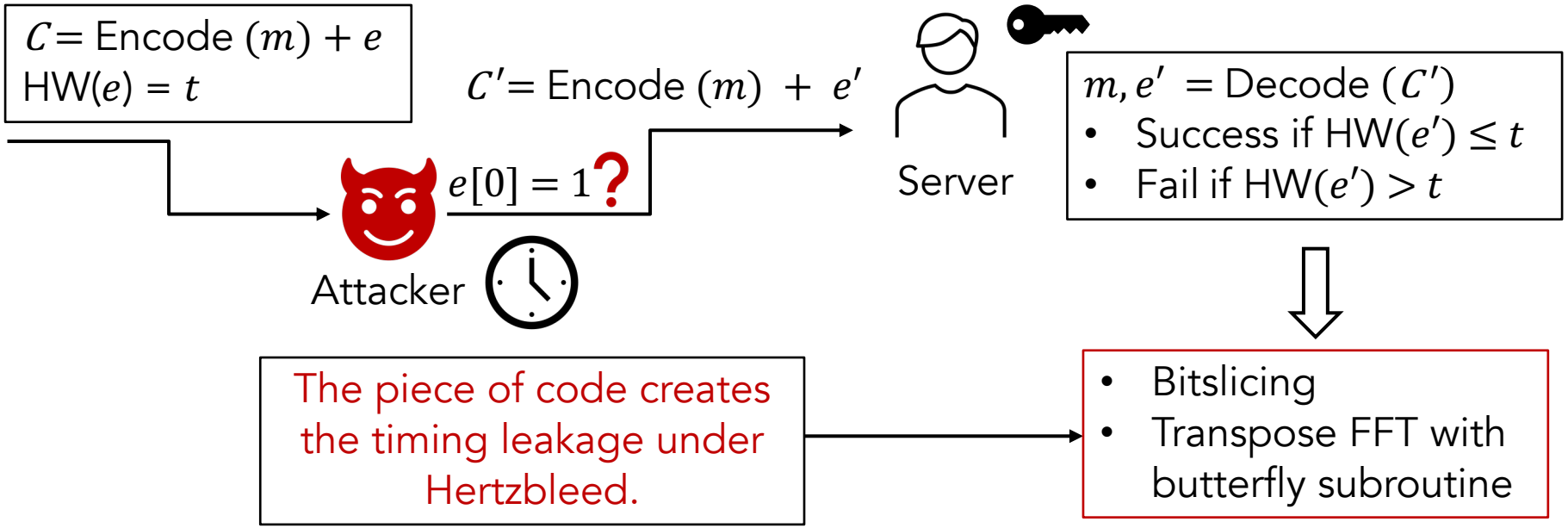
Hertzbleed Attack on Classic McEliece



Hertzbleed Attack on Classic McEliece



Hertzbleed Attack on Classic McEliece



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.
- No systematic way of achieving constant-power without masking.

```
if secret == 1 then
    routine();
```

No secret-dependent branches

```
state = array[secret]
```

No secret-dependent memory accesses

```
res = x * secret / 255.0f
```

No secret inputs to variable-time instructions

Discussion & Takeaway

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

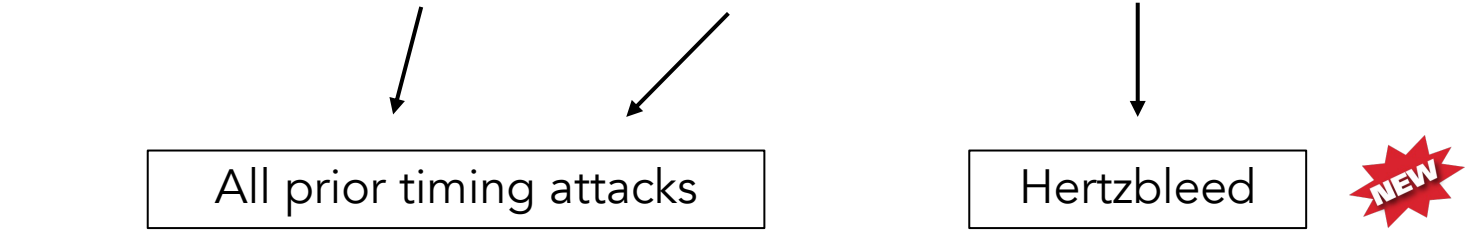
Discussion & Takeaway

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

All prior timing attacks

Discussion & Takeaway

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



References

- Hertzbleed: Turning Power Side-Channel Attacks Into Remote Timing Attacks on x86 (USENIX Security 2022) www.hertzbleed.com
  - **Yingchen Wang*,** Riccardo Paccagnella*, Elizabeth He, Hovav Shacham, Christopher Fletcher, David Kohlbrenner.
  - IEEE Micro Top Picks 2023, Black Hat Pwnie Award 2022 for Best Cryptographic Attack

- DVFS Frequently Leaks Secrets: Hertzbleed Attacks Beyond SIKE, Cryptography, and CPU-Only Data (IEEE Security & Privacy 2023)
  - **Yingchen Wang**, Riccardo Paccagnella, Alan Wandke, Zhao Gang, Grant Garrett-Grossman, Christopher Fletcher, David Kohlbrenner, Hovav Shacham.