Simple Power Analysis assisted Chosen Ciphertext Attack on ML-KEM

CASCADE 2025

Alexandre Berzati, Andersson Calle Viera, **Maya Chartouny**, David Vigilant

April 2, 2025

Outline

Outline

Introduction

PQC: Several algorithms are now standardized through various international initiatives

Kyber is a PQC key encapsulation mechanism selected by the NIST

ML-KEM standard variant derived from Kyber

Icon made by Freepik from www.flaticon.com

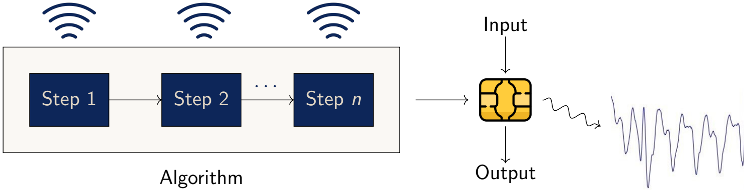
Introduction

PQC: Several algorithms are now standardized through various international initiatives

Kyber is a PQC key encapsulation mechanism selected by the NIST

ML-KEM standard variant derived from Kyber

Problem: Algorithms run on physical devices



Icon made by Freepik from www.flaticon.com

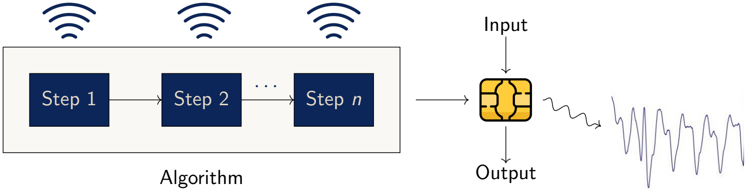
Introduction

PQC: Several algorithms are now standardized through various international initiatives

Kyber is a PQC key encapsulation mechanism selected by the NIST

ML-KEM standard variant derived from Kyber

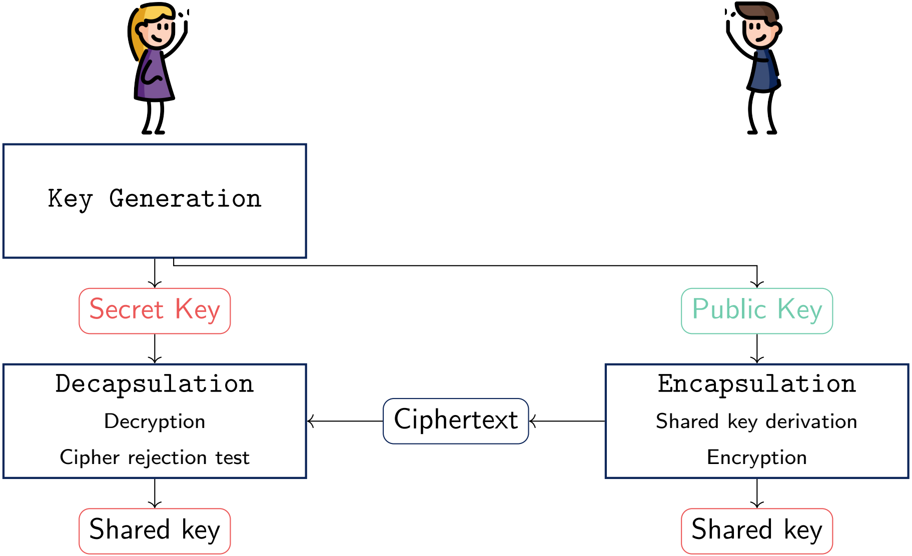
Problem: Algorithms run on physical devices



Our Contribution: SPA assisted CCA on Kyber

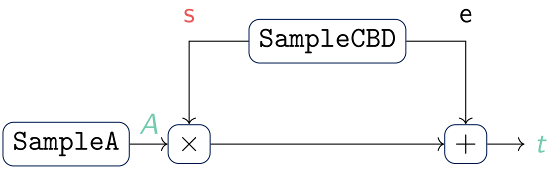
Icon made by Freepik from www.flaticon.com

Kyber structure



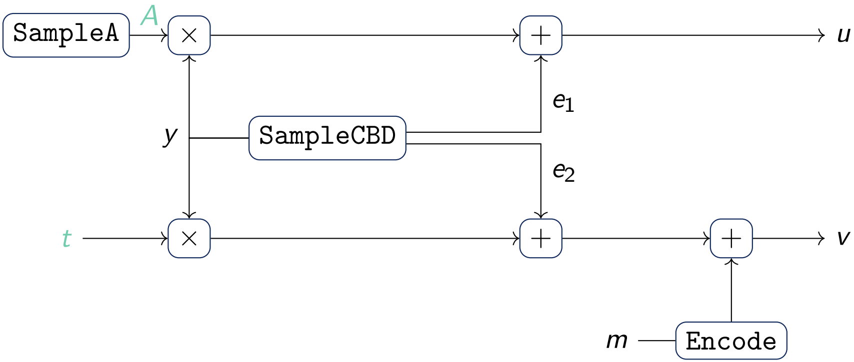
Key Generation

$$\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$$
$$n = 256 \text{ et } q = 3\,329$$



Public key: A, t
Secret key: s

Encryption

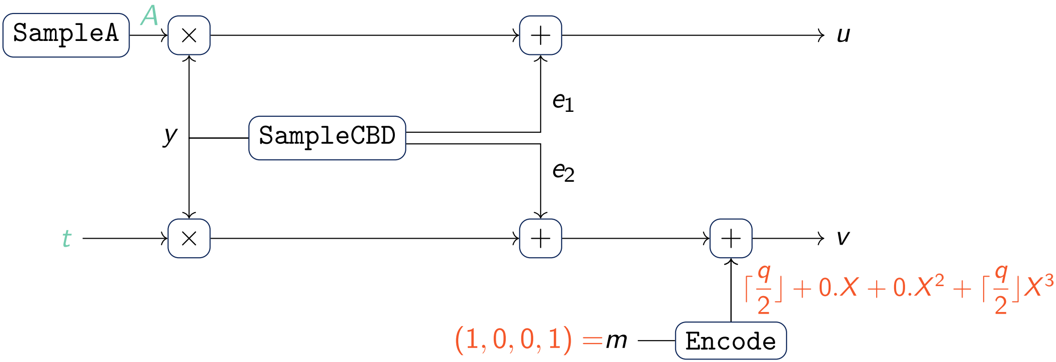


Ciphertext:

$u = Ay + e_1$

$v = ty + e_2 + \texttt{Encode}(m)$

Encryption

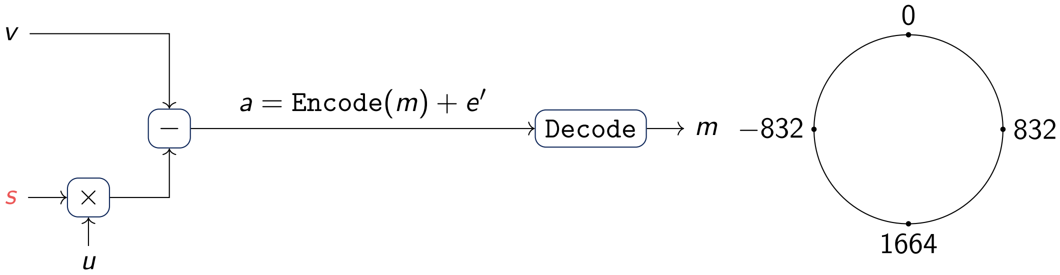


Ciphertext:

$u = Ay + e_1$

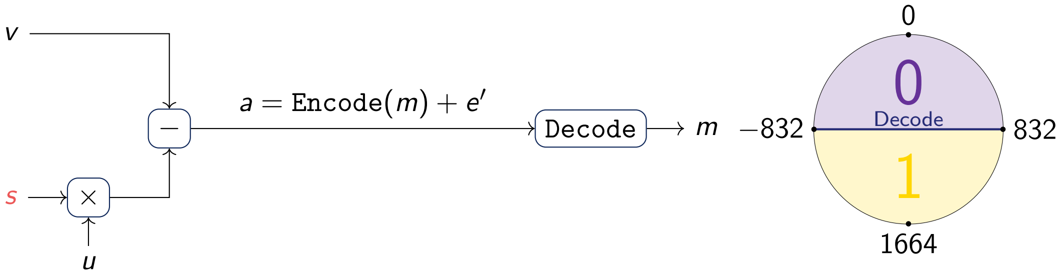
$v = ty + e_2 + \texttt{Encode}(m)$

Decryption



$m = v - su$

Decryption



$m = v - su$ is well recovered if the error is not too big

Outline

Zoom on decapsulation

Decapsulation

Zoom on decapsulation

Decapsulation

$\vdots$

Zoom on decapsulation

Decapsulation

$\vdots$

Decryption

$\vdots$

Zoom on decapsulation

Decapsulation

⋮

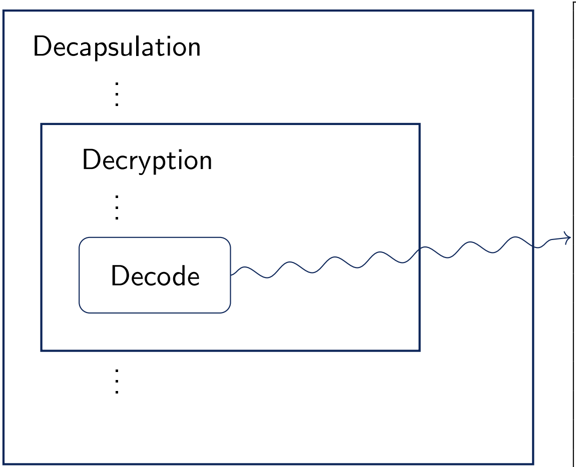
Decryption

⋮

Decode

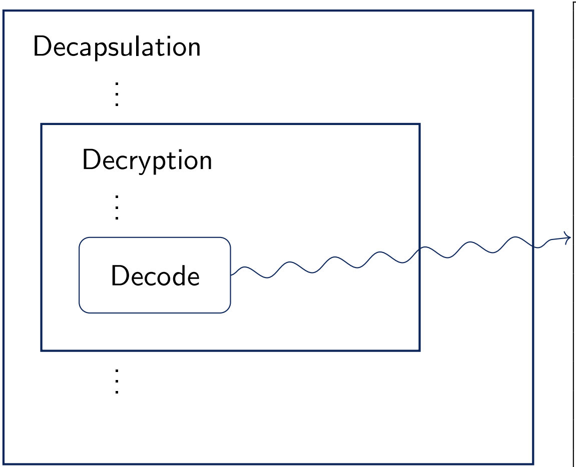
⋮

Zoom on decapsulation




```c
// Q = 3329
void poly_tomsg(uint8_t msg[32],
                    const poly *a){
  unsigned int i,j;
  uint32_t t;

  for(i=0;i<N/8;i++) {
    msg[i] = 0;
    for(j=0;j<8;j++) {
      t = a[8*i+j];
      t += ((int16_t)t >> 15) & Q;
      t = (((t << 1) + Q/2)/Q) & 1;
      msg[i] |= t << j;
    }
  }
 return msg;
}
```

Zoom on decapsulation




```c
// Q = 3329
void poly_tomsg(uint8_t msg[32],
                          const poly *a){
  unsigned int i,j;
  uint32_t t;

  for(i=0;i<N/8;i++) {
    msg[i] = 0;
    for(j=0;j<8;j++) {
      t = a[8*i+j];
      t += ((int16_t)t >> 15) & Q;
      t = (((t << 1) + Q/2)/Q) & 1;
      msg[i] |= t << j;
   }
 }
 return msg;
}
```

Reference code submitted to NIST
Considered to have constant time

KyberSlash1 [Ber+25]

- Non-constant division
  - Some platforms
  - Some compilation flags

KyberSlash1 [Ber+25]

- Non-constant division
  - Some platforms
  - Some compilation flags

```c
// Q = 3329
void poly_tomsg(uint8_t msg[32],
                  const poly *a){
  unsigned int i,j;
  uint32_t t;

  for(i=0;i<N/8;i++) {
    msg[i] = 0;
    for(j=0;j<8;j++) {
      t = a[8*i+j];
      t += ((int16_t)t >> 15) & Q;
      t = (((t << 1) + Q/2)/Q) & 1;
      msg[i] |= t << j;
    }
  }
  return msg;
}
```

KyberSlash1 [Ber+25]

- Non-constant division
  - Some platforms
  - Some compilation flags
- Timing attack: Difference between coefficients rounded to 0 and those rounded to 1

```
// Q = 3329
void poly_tomsg(uint8_t msg[32],
                    const poly *a){
  unsigned int i,j;
  uint32_t t;

  for(i=0;i<N/8;i++) {
    msg[i] = 0;
    for(j=0;j<8;j++) {
      t = a[8*i+j];
      t += ((int16_t)t >> 15) & Q;
      t = (((t << 1) + Q/2)/Q) & 1;
      msg[i] |= t << j;
    }
  }
 return msg;
}
```

KyberSlash1 [Ber+25]

- Non-constant division
  - Some platforms
  - Some compilation flags
- Timing attack: Difference between coefficients rounded to 0 and those rounded to 1
- Allows key recovery

```c
// Q = 3329
void poly_tomsg(uint8_t msg[32],
                        const poly *a){
  unsigned int i,j;
  uint32_t t;

  for(i=0;i<N/8;i++) {
    msg[i] = 0;
    for(j=0;j<8;j++) {
      t = a[8*i+j];
      t += ((int16_t)t >> 15) & Q;
      t = (((t << 1) + Q/2)/Q) & 1;
      msg[i] |= t << j;
    }
  }
 return msg;
}
```

KyberSlash1 [Ber+25]

- Non-constant division
  - Some platforms
  - Some compilation flags
- Timing attack: Difference between coefficients rounded to 0 and those rounded to 1
- Allows key recovery

Post-KyberSlash1:

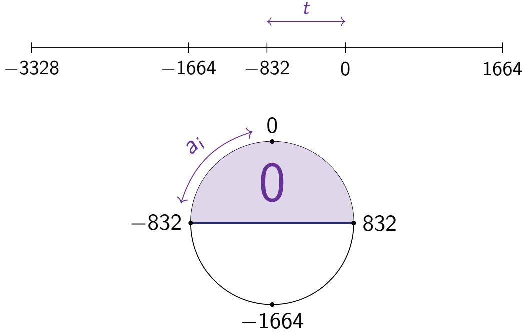
```
t <<= 1;
t += 1665;
t *= 80635;
t >>= 28;
t &= 1;
```

```
// Q = 3329
void poly_tomsg(uint8_t msg[32],
                const poly *a){
  unsigned int i,j;
  uint32_t t;

  for(i=0;i<N/8;i++) {
    msg[i] = 0;
    for(j=0;j<8;j++) {
      t = a[8*i+j];
      t += ((int16_t)t >> 15) & Q;
      t = (((t << 1) + Q/2)/Q) & 1;
      msg[i] |= t << j;
    }
  }
  return msg;
}
```

Analysis of the post-KyberSlash1 implementation

```c
// Q = 3329
void poly_tomsg(uint8_t msg[32],
                const poly *a){
   unsigned int i,j;
   uint32_t t;

   for(i=0;i<N/8;i++) {
     msg[i] = 0;
     for(j=0;j<8;j++) {
       t = a[8*i+j];
       t <<= 1;
       t += 1665;
       t *= 80635;
       t >>= 28;
       t &= 1;
       msg[i] |= t << j;
     }
   return msg;
  }
}
```

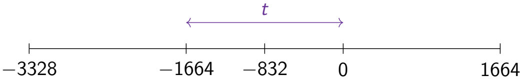
Case 0: If $a_i \in [-832, -1]$, then $m_i = 0$

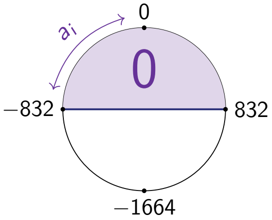


Analysis of the post-KyberSlash1 implementation

```c
// Q = 3329
void poly_tomsg(uint8_t msg[32],
                        const poly *a){
   unsigned int i,j;
   uint32_t t;

   for(i=0;i<N/8;i++) {
     msg[i] = 0;
     for(j=0;j<8;j++) {
       t = a[8*i+j];
       t <<= 1;
       t += 1665;
       t *= 80635;
       t >>= 28;
       t &= 1;
       msg[i] |= t << j;
     }
  return msg;
  }
}
```

Case 0: If $a_i \in [-832, -1]$, then $m_i = 0$

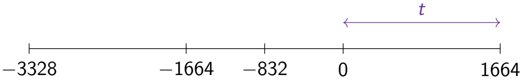


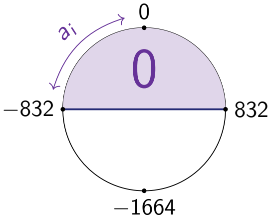


Analysis of the post-KyberSlash1 implementation

```c
// Q = 3329
void poly_tomsg(uint8_t msg[32],
                const poly *a){
  unsigned int i,j;
  uint32_t t;

  for(i=0;i<N/8;i++) {
    msg[i] = 0;
    for(j=0;j<8;j++) {
      t = a[8*i+j];
      t <<= 1;
      t += 1665;
      t *= 80635;
      t >>= 28;
      t &= 1;
      msg[i] |= t << j;
    }
  return msg;
  }
}
```

Case 0: If $a_i \in [-832, -1]$, then $m_i = 0$

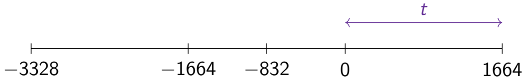


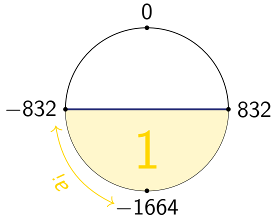


Analysis of the post-KyberSlash1 implementation

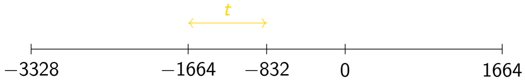
```c
// Q = 3329
void poly_tomsg(uint8_t msg[32],
                const poly *a){
   unsigned int i,j;
   uint32_t t;

   for(i=0;i<N/8;i++) {
     msg[i] = 0;
     for(j=0;j<8;j++) {
       t = a[8*i+j];
       t <<= 1;
       t += 1665;
       t *= 80635;
       t >>= 28;
       t &= 1;
       msg[i] |= t << j;
     }
   return msg;
   }
}
```

Case 0: If $a_i \in [-832, -1]$, then $m_i = 0$





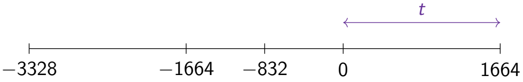
Case 1: If $a_i \in [-1664, -833]$, then $m_i = 1$

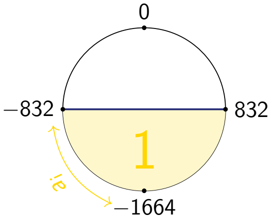


Analysis of the post-KyberSlash1 implementation

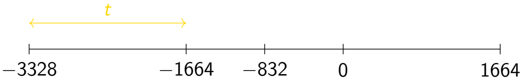
```c
// Q = 3329
void poly_tomsg(uint8_t msg[32],
                const poly *a){
  unsigned int i,j;
  uint32_t t;

  for(i=0;i<N/8;i++) {
    msg[i] = 0;
    for(j=0;j<8;j++) {
      t = a[8*i+j];
      t <<= 1;
      t += 1665;
      t *= 80635;
      t >>= 28;
      t &= 1;
      msg[i] |= t << j;
    }
  return msg;
  }
}
```

Case 0: If $a_i \in [-832, -1]$, then $m_i = 0$





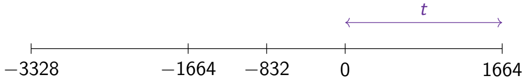
Case 1: If $a_i \in [-1664, -833]$, then $m_i = 1$

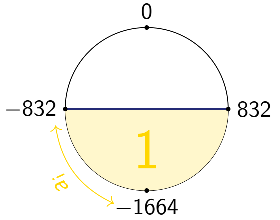


Analysis of the post-KyberSlash1 implementation

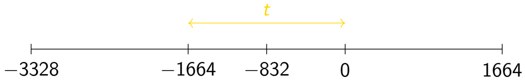
```
// Q = 3329
void poly_tomsg(uint8_t msg[32],
                const poly *a){
   unsigned int i,j;
   uint32_t t;

   for(i=0;i<N/8;i++) {
     msg[i] = 0;
     for(j=0;j<8;j++) {
       t = a[8*i+j];
       t <<= 1;
       t += 1665;
       t *= 80635;
       t >>= 28;
       t &= 1;
       msg[i] |= t << j;
     }
  return msg;
  }
}
```

Case 0: If $a_i \in [-832, -1]$, then $m_i = 0$



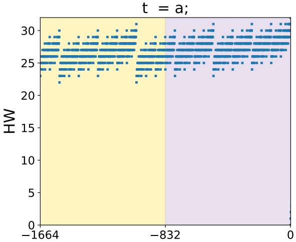


Case 1: If $a_i \in [-1664, -833]$, then $m_i = 1$



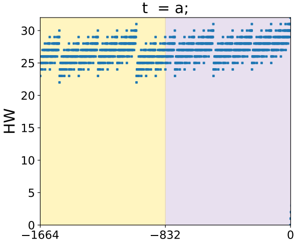
Confirmation of the differences

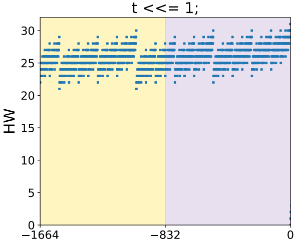
HW for all possible values in $[-1664, -833]$ and $[-832, -1]$



Confirmation of the differences

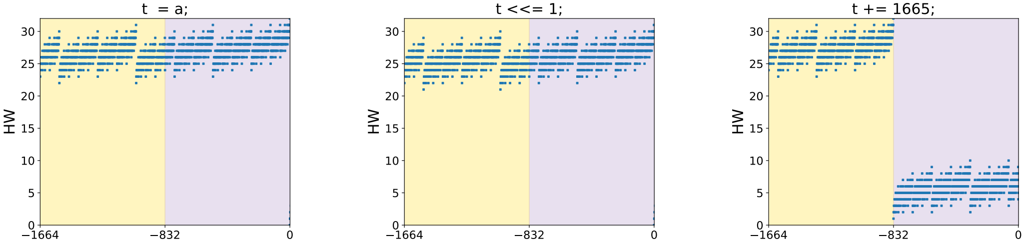
HW for all possible values in $[-1664, -833]$ and $[-832, -1]$





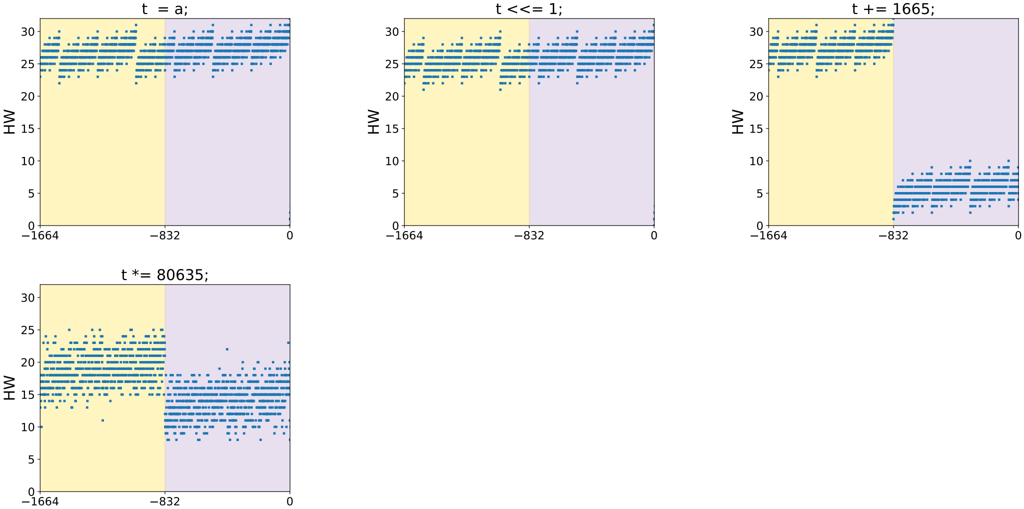
Confirmation of the differences

HW for all possible values in $[-1664, -833]$ and $[-832, -1]$



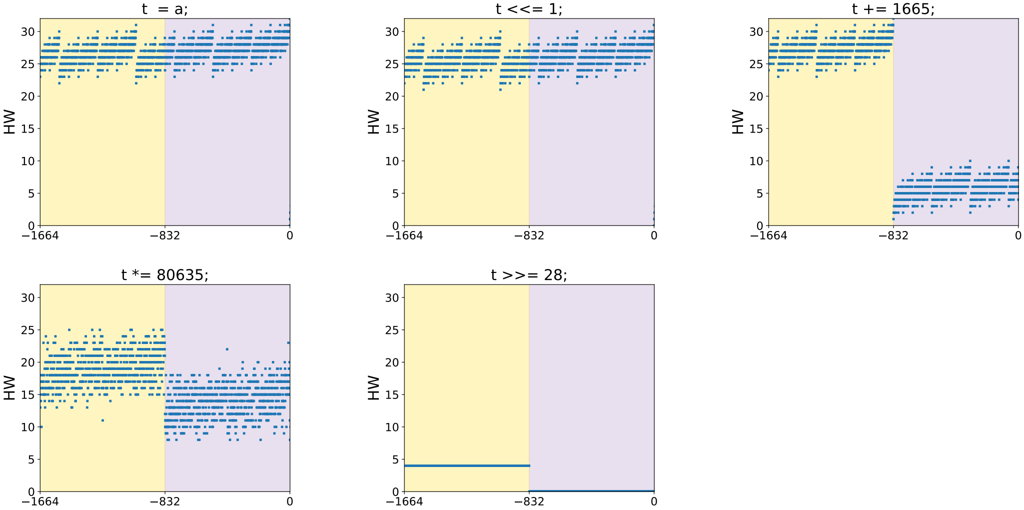
Confirmation of the differences

HW for all possible values in $[-1664, -833]$ and $[-832, -1]$



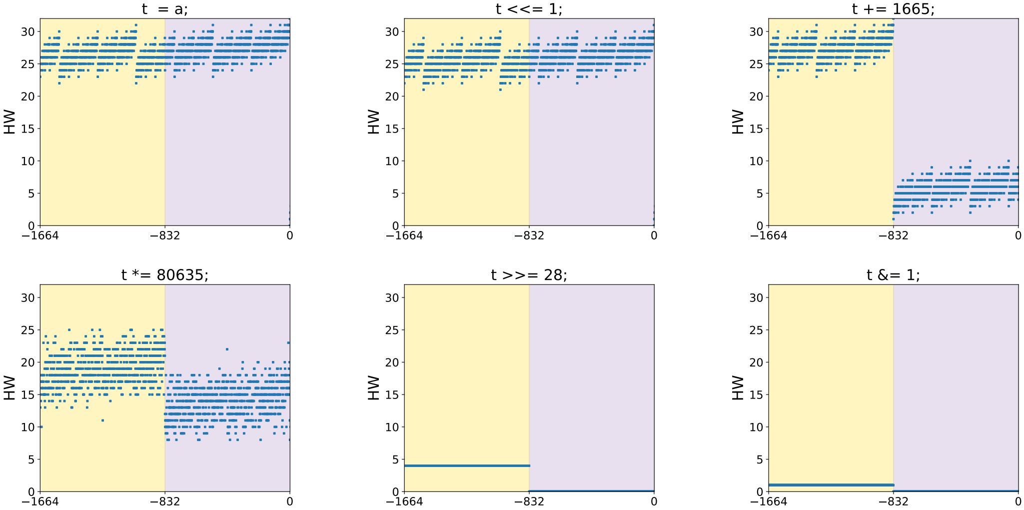
Confirmation of the differences

HW for all possible values in $[-1664, -833]$ and $[-832, -1]$



Confirmation of the differences

HW for all possible values in $[-1664, -833]$ and $[-832, -1]$



Attack strategy

- **Step 1:** Send several well-chosen (u, v) pairs to the oracle in order to:
  - Collect traces where we end up in case 0
  - Collect traces where we end up in case 1

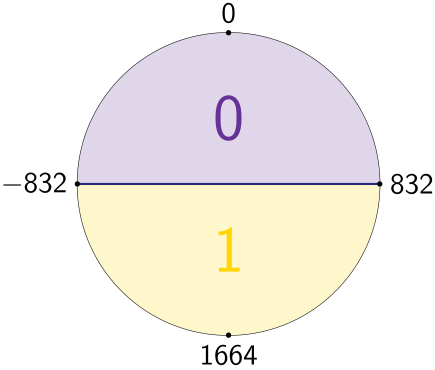
  Without knowing the **secret**

  Compute the averages $\mathcal{M}_0$ and $\mathcal{M}_1$ for each set
- **Step 2:** Send malicious ciphertexts to recover the secret key

Step 1: Dataset construction

$$m = v - su, \quad -\eta_1 \leq s \leq \eta_1$$

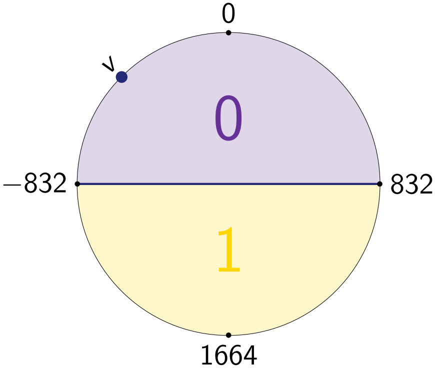
Choose u and v such that a is located in the semicircle desired



Step 1: Dataset construction

$$m = v - s u, \quad -\eta_1 \leq s \leq \eta_1$$

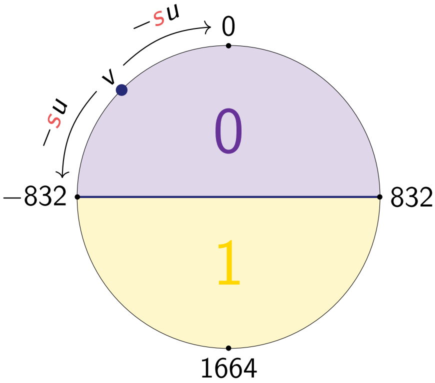
Choose u and v such that a is located in the semicircle of case 0



Step 1: Dataset construction

$$m = v - su, \quad -\eta_1 \leq s \leq \eta_1$$

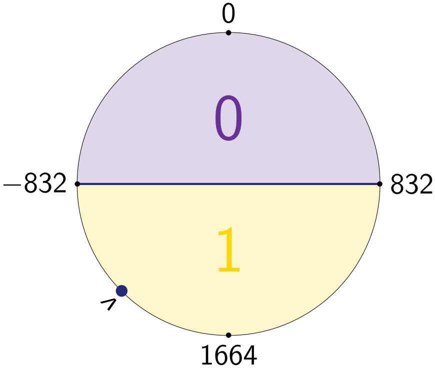
Choose u and v such that a is located in the semicircle of case 0



Step 1: Dataset construction

$$m = v - su, \quad -\eta_1 \leq s \leq \eta_1$$

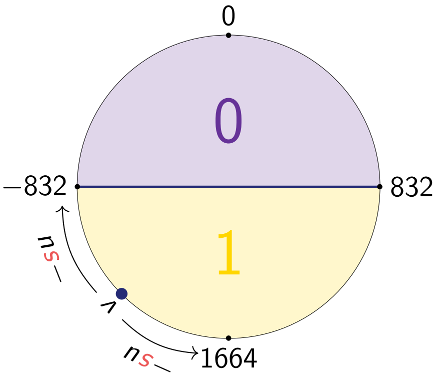
Choose u and v such that a is located in the semicircle of case 1



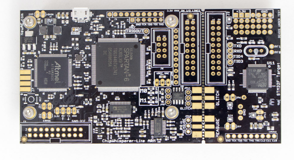
Step 1: Dataset construction

$$m = v - {\color{red}s}u, \quad -\eta_1 \leq {\color{red}s} \leq \eta_1$$

Choose u and v such that a is located in the semicircle of case 1



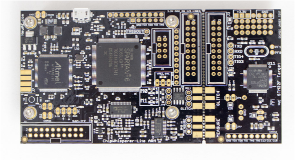
Trace acquisition



- Arm Cortex M4
- CPU: 32 bits
- RAM: 48kB
- 4 samples/cycle

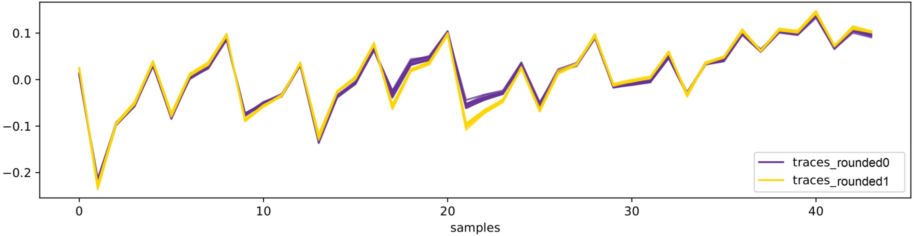
Trace acquisition

$\xrightarrow{(u,v)}$



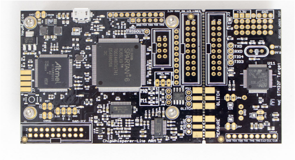
- Arm Cortex M4
- CPU: 32 bits
- RAM: 48kB
- 4 samples/cycle

$\downarrow$ *traces*



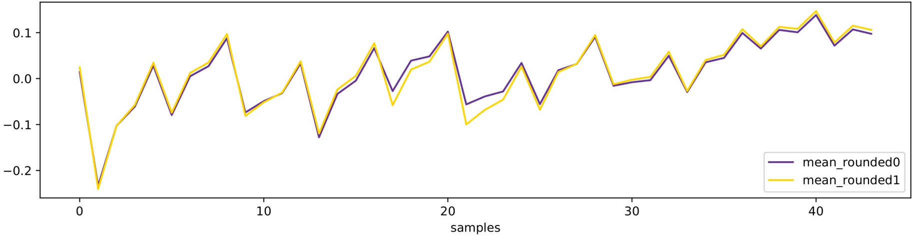
Trace acquisition

$\xrightarrow{(u,v)}$



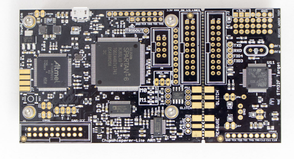
- Arm Cortex M4
- CPU: 32 bits
- RAM: 48kB
- 4 samples/cycle

$\Big\downarrow$ *means*



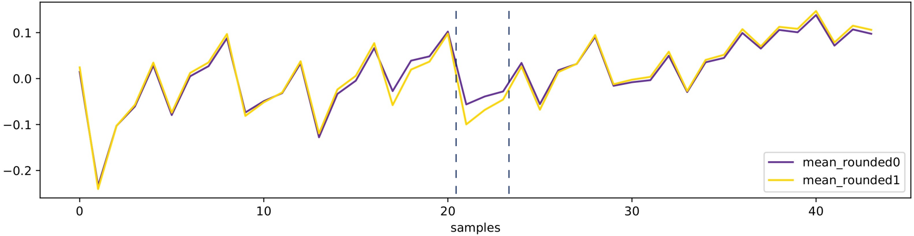
Trace acquisition

$\xrightarrow{(u,v)}$



- Arm Cortex M4
- CPU: 32 bits
- RAM: 48kB
- 4 samples/cycle

$\downarrow$ *means*



Significant distance between the two averages
$\rightarrow$ Distinguisher

Step 2: Recover the secret key

Objective: Recover the secret key s

$$\boxed{v} - \boxed{s[0] \;\big|\; s[1]} \times \boxed{\begin{array}{c} \mathrm{u}[0] \\ \hline \mathrm{u}[1] \end{array}}$$

$$= \boxed{v - su}$$

Step 2: Recover the secret key

Objective: Recover the secret key s

$$\boxed{v_0 X^0 + \cdots + v_{255} X^{255}} - \boxed{s[0]_0 X^0 + \cdots + s[0]_{255} X^{255} \;\Big|\; s[1]_0 X^0 + \cdots + s[1]_{255} X^{255}} \times \boxed{\begin{array}{c} u[0]_0 X^0 + \cdots + u[0]_{255} X^{255} \\ \hline u[1]_0 X^0 + \cdots + u[1]_{255} X^{255} \end{array}}$$

$$= \boxed{\cdots}$$

Step 2: Recover the secret key

Objective: Recover the secret key s

$$\boxed{-832-\cdots-832X^{255}} \quad - \quad \boxed{s[0]_0X^0+\cdots+s[0]_{255}X^{255}} \boxed{s[1]_0X^0+\cdots+s[1]_{255}X^{255}} \quad \times \quad \begin{array}{|c|}\hline 208 \\ \hline 0 \\ \hline\end{array}$$

$$= \boxed{(-832-208s[0]_0) \quad + \quad \cdots \quad + \quad (-832-208s[0]_{255})X^{255}}$$

Step 2: Recover the secret key

Objective: Recover the secret key s

$$\boxed{-832 - \cdots - 832X^{255}} - \boxed{s[0]_0X^0 + \cdots + s[0]_{255}X^{255} \;\Big|\; s[1]_0X^0 + \cdots + s[1]_{255}X^{255}} \times \boxed{\begin{array}{c} 208 \\ \hline 0 \end{array}}$$

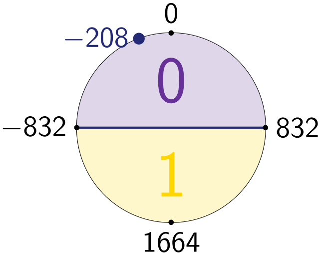
$$= \boxed{(-832 - 208s[0]_0) \quad + \quad \cdots}$$

Step 2: Recover the secret key

Objective: Recover the secret key s

$$\boxed{-832 - \cdots - 832X^{255}} - \begin{bmatrix} s[0]_0X^0 + \cdots + s[0]_{255}X^{255} & s[1]_0X^0 + \cdots + s[1]_{255}X^{255} \end{bmatrix} \times \begin{bmatrix} 208 \\ 0 \end{bmatrix}$$

$$= \boxed{(-832 - 208(-3)) \quad + \quad \cdots}$$

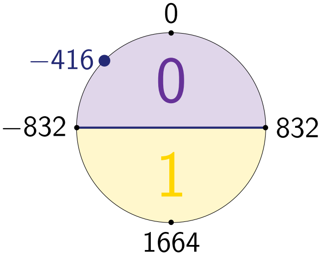


Step 2: Recover the secret key

Objective: Recover the secret key s

$$\boxed{-832 - \cdots - 832X^{255}} \;-\; \boxed{s[0]_0X^0 + \cdots + s[0]_{255}X^{255}}\,\boxed{s[1]_0X^0 + \cdots + s[1]_{255}X^{255}} \;\times\; \boxed{\begin{array}{c} 208 \\ \hline 0 \end{array}}$$

$$= \boxed{(-832 - 208(-2)) \quad + \quad \cdots}$$

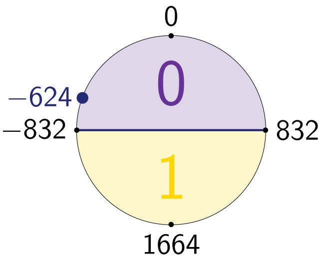


Step 2: Recover the secret key

Objective: Recover the secret key s

$$\boxed{-832 - \cdots - 832X^{255}} - \boxed{s[0]_0 X^0 + \cdots + s[0]_{255} X^{255}} \boxed{s[1]_0 X^0 + \cdots + s[1]_{255} X^{255}} \times \begin{array}{|c|} \hline 208 \\ \hline 0 \\ \hline \end{array}$$

$$= \boxed{(-832 - 208(-1)) \quad + \quad \cdots}$$

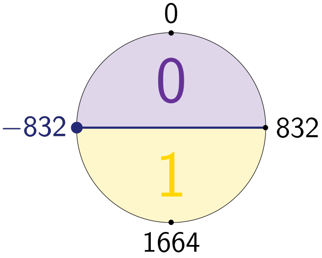


Step 2: Recover the secret key

Objective: Recover the secret key s

$$\boxed{-832 - \cdots - 832X^{255}} \quad - \quad \boxed{s[0]_0X^0 + \cdots + s[0]_{255}X^{255}} \boxed{s[1]_0X^0 + \cdots + s[1]_{255}X^{255}} \quad \times \quad \begin{array}{|c|}\hline 208 \\ \hline 0 \\ \hline\end{array}$$

$$= \boxed{(-832 - 208(0)) \quad + \quad \cdots}$$

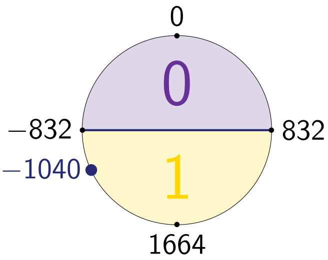


Step 2: Recover the secret key

Objective: Recover the secret key s

$$\boxed{-832 - \cdots - 832X^{255}} \;-\; \boxed{s[0]_0X^0 + \cdots + s[0]_{255}X^{255}} \boxed{s[1]_0X^0 + \cdots + s[1]_{255}X^{255}} \;\times\; \begin{array}{|c|}\hline 208 \\ \hline 0 \\ \hline \end{array}$$

$$= \boxed{(-832 - 208(1)) \quad + \quad \cdots}$$

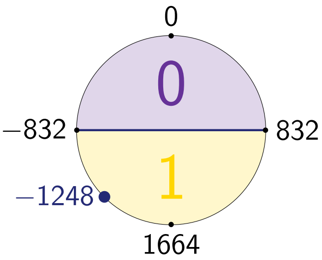


Step 2: Recover the secret key

Objective: Recover the secret key s

$$\boxed{-832 - \cdots - 832X^{255}} \quad - \quad \boxed{s[0]_0X^0 + \cdots + s[0]_{255}X^{255} \,\Big|\, s[1]_0X^0 + \cdots + s[1]_{255}X^{255}} \quad \times \quad \boxed{\begin{array}{c} 208 \\ \hline 0 \end{array}}$$

$$= \boxed{(-832 - 208(2)) \quad + \quad \cdots}$$

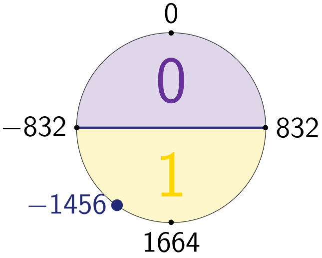


Step 2: Recover the secret key

Objective: Recover the secret key s

$$\boxed{-832-\cdots-832X^{255}} \;-\; \boxed{s[0]_0X^0+\cdots+s[0]_{255}X^{255} \;\Big|\; s[1]_0X^0+\cdots+s[1]_{255}X^{255}} \;\times\; \begin{bmatrix}208\\ 0\end{bmatrix}$$

$$= \boxed{(-832-208(3)) \quad + \quad \cdots}$$



Step 2: Recover the secret key

For all coefficients at once:

Start

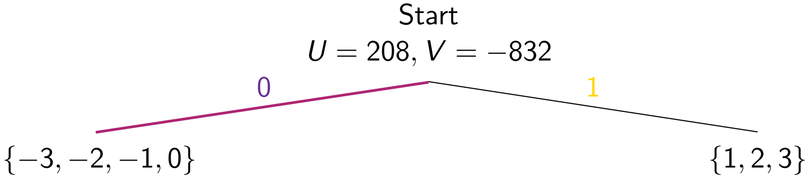
$U = 208, V = -832$

0   1

$\{-3, -2, -1, 0\}$ $\{1, 2, 3\}$

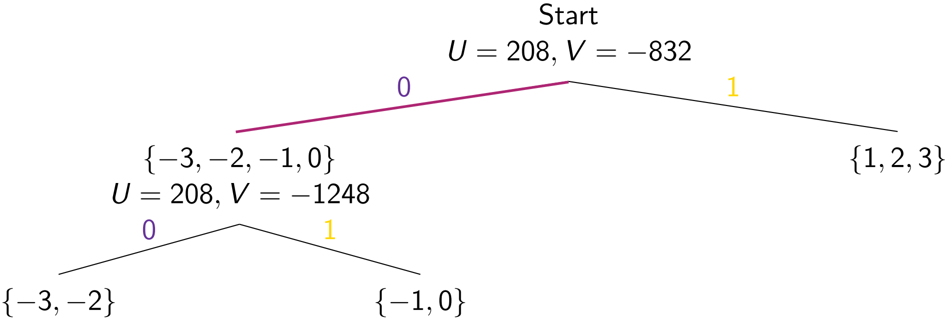
Step 2: Recover the secret key

For all coefficients at once:



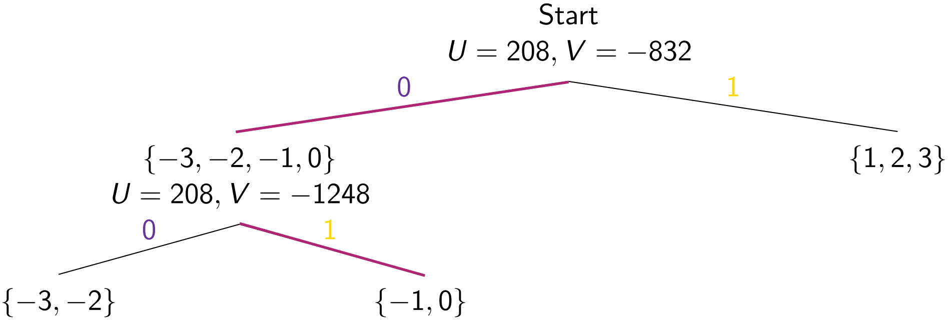
Step 2: Recover the secret key

For all coefficients at once:



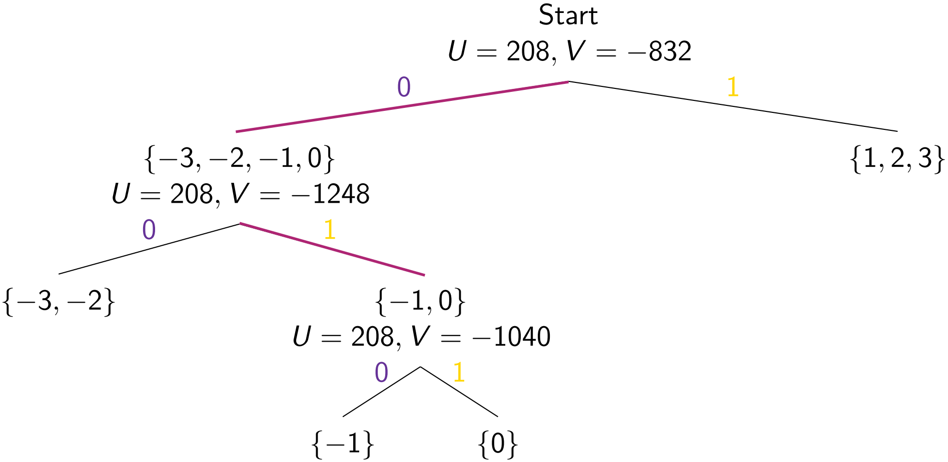
Step 2: Recover the secret key

For all coefficients at once:



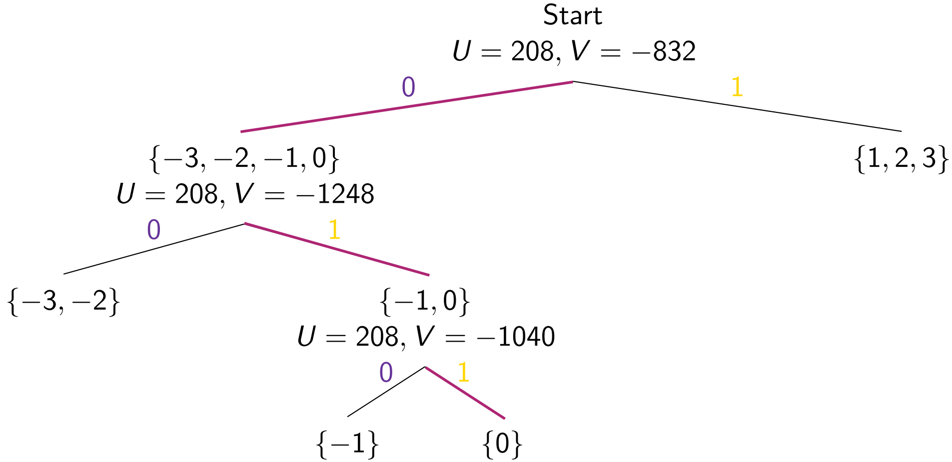
Step 2: Recover the secret key

For all coefficients at once:



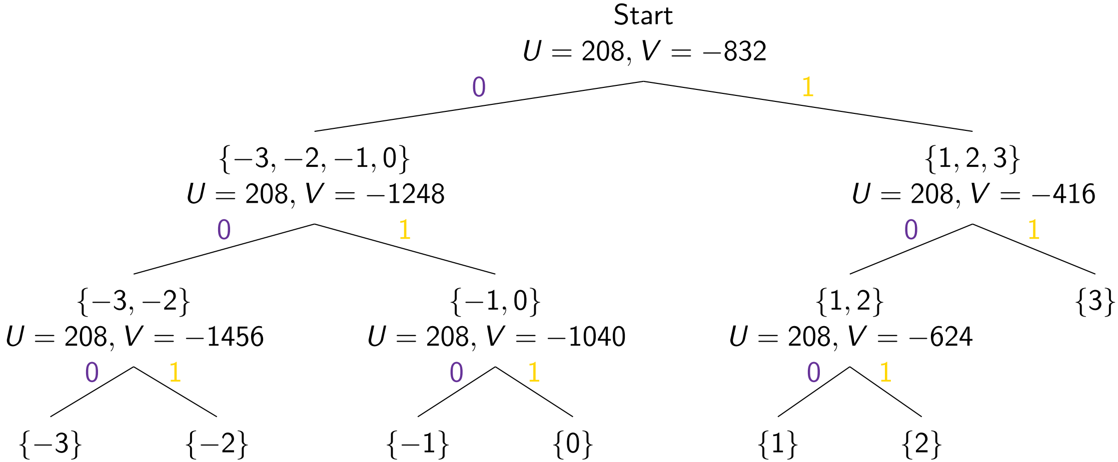
Step 2: Recover the secret key

For all coefficients at once:



Step 2: Recover the secret key

For all coefficients at once:



Attack performance

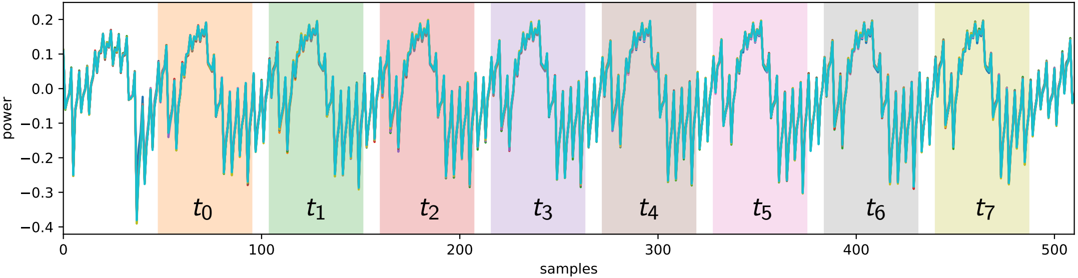
- **Step 1**: Construction of the averages
  - Number of traces: 42 per average ($\mathcal{M}_0$ and $\mathcal{M}_1$)
  - Time: $\approx$ 3 min
  - Advantage: Can be performed directly on the victim
- **Step 2**: Chosen ciphertext assisted by parallel power analysis
  - Number of traces: 3 traces per polynomial for all security levels
  - Time: $\approx$ 30 sec

Performance: On the 100 keys from the KAT files

Security level	**Kyber-512**	**Kyber-768**	**Kyber-1024**
Success rates	100%	100%	100%

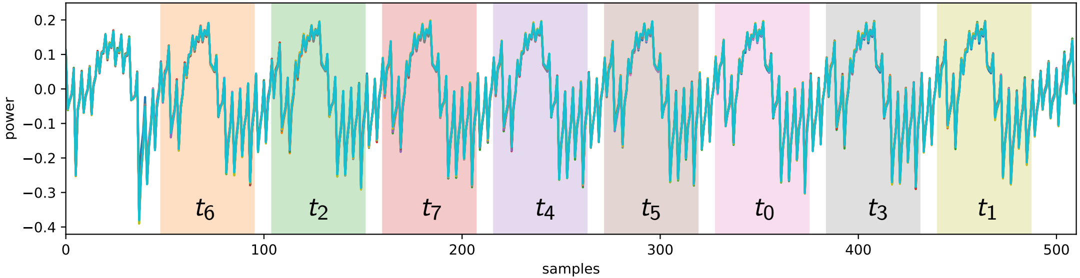
Attack in the presence of shuffling

Without shuffling:



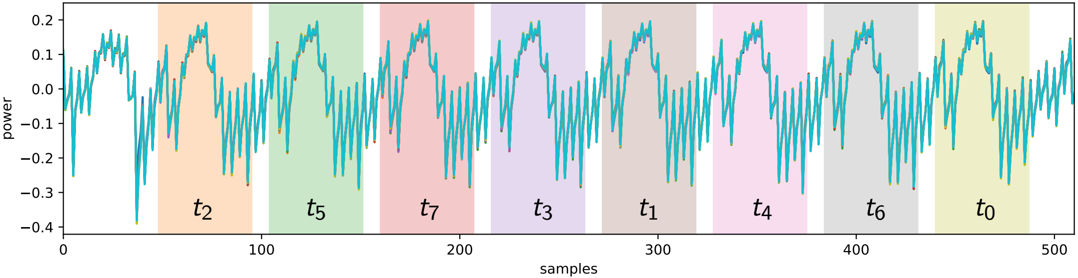
Attack in the presence of shuffling

With shuffling:



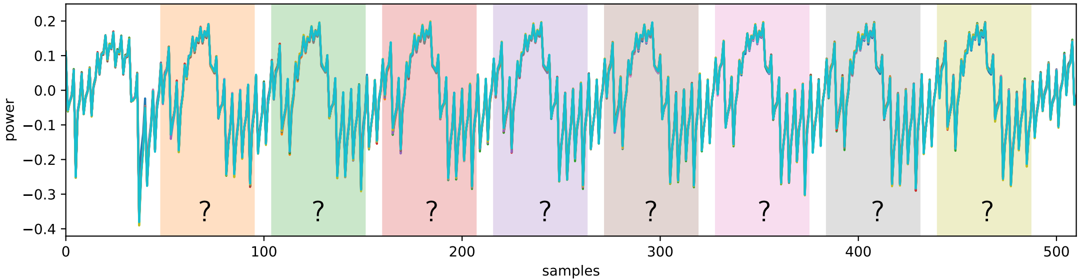
Attack in the presence of shuffling

With shuffling:



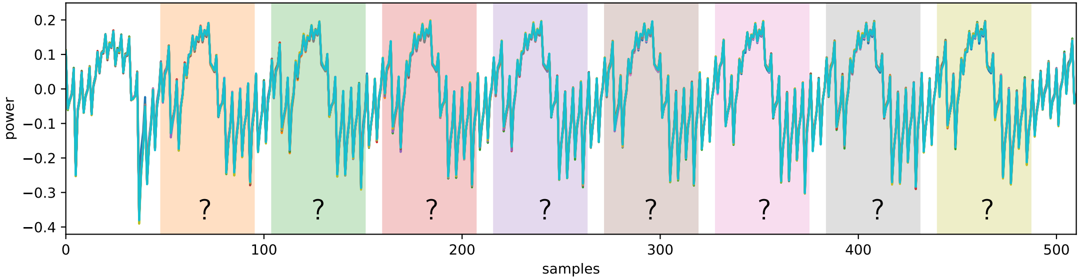
Attack in the presence of shuffling

With shuffling:



Attack in the presence of shuffling

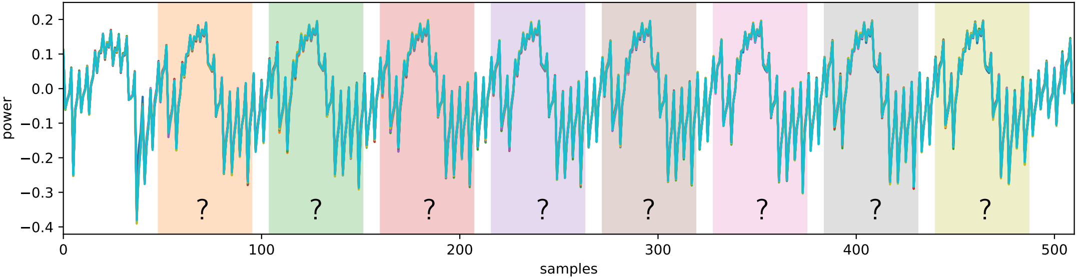
With shuffling:



- **Step 1**: Construction of averages as before, but focusing only on the first coefficient

Attack in the presence of shuffling

With shuffling:



- **Step 1**: Construction of averages as before, but focusing only on the first coefficient
- **Step 2**: New strategy to find the secret key
  - Only one coefficient can be varied at a time, parallel attack is no longer possible
  - Count the total 1 obtained at each step and compare

Step 2: Secret key recovery with shuffling

For each coefficient:

Base-count

$U = 208,\ V = -832, \mathcal{N}_0$

Step 2: Secret key recovery with shuffling

For each coefficient:

Base-count

$U = 208,\ V = -832,\ \mathcal{N}_0$

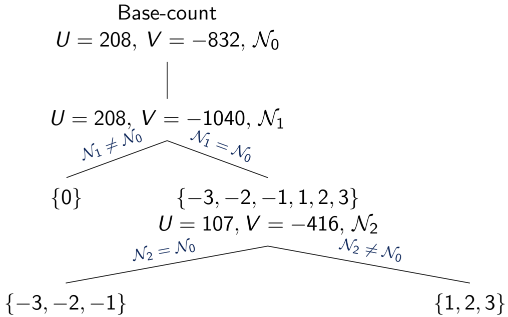
$U = 208,\ V = -1040,\ \mathcal{N}_1$

$\mathcal{N}_1 \neq \mathcal{N}_0$: $\{0\}$

$\mathcal{N}_1 = \mathcal{N}_0$: $\{-3, -2, -1, 1, 2, 3\}$

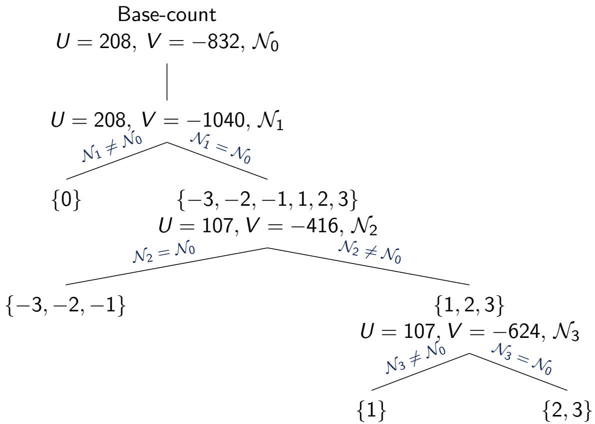
Step 2: Secret key recovery with shuffling

For each coefficient:



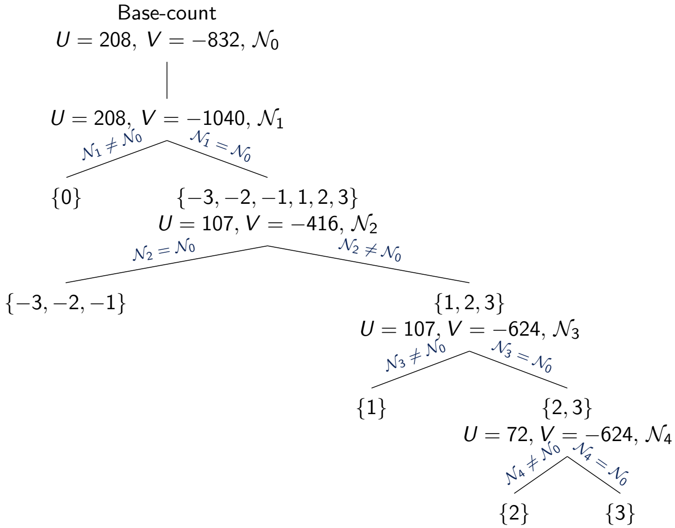
Step 2: Secret key recovery with shuffling

For each coefficient:



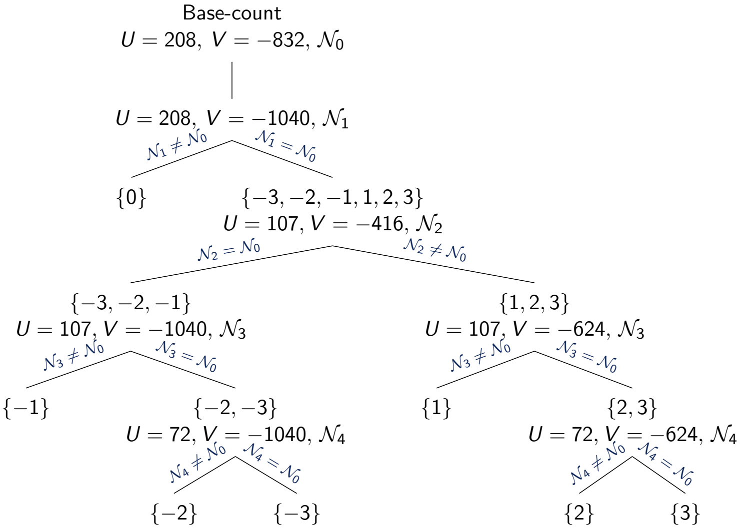
Step 2: Secret key recovery with shuffling

For each coefficient:



Step 2: Secret key recovery with shuffling

For each coefficient:



Attack performance with shuffling

- **Step 1**: Construction of the averages
  - Number of traces: 42 per average ($\mathcal{M}_0$ and $\mathcal{M}_1$)
  - Time: $\approx$ 3 min
  - Advantage: Can be performed directly on the victim
- **Step 2**: Chosen ciphertext assisted by power analysis
  - Number of traces: $\approx 1844/2494/3326$ traces to recover the secret depending on the security level
  - Time: $\approx$ 2h 30 min

Performance: On 100 keys from the KAT files

Security level	**Kyber-512**	**Kyber-768**	**Kyber-1024**
Success rate	100%	100%	100%

Outline

Conclusion

- Timing attacks transposed into power leakage
- Attack applicable also to shuffling implementation
- Attack can be done directly on the victim and without profiling
- Inverting addition and multiplication reduces leakage, but residual bias remains
- To be truly protected, masking must be used

Thank you

Questions?

References I

[Ber+25] Daniel J. Bernstein, Karthikeyan Bhargavan, Shivam Bhasin, Anupam Chattopadhyay, Tee Kiah Chia, Matthias J. Kannwischer, Franziskus Kiefer, Thales B. Paiva, Prasanna Ravi, and Goutam Tamvada. "KyberSlash: Exploiting secret-dependent division timings in Kyber implementations". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2025) (see sildes 20–24).

[Kan+] Matthias J. Kannwischer, Peter Schwabe, Douglas Stebila, and Thom Wiggers. *PQClean*. `https://github.com/PQClean/PQClean`. Accessed: 2022-12-15.

References II

[NIS23] NIST. *FIPS 203: Module-Lattice-Based Key-Encapsulation Mechanism Standard*. Federal Inf. Process. Stds. (NIST FIPS), National Institute of Standards and Technology, Gaithersburg, MD. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.203.pdf. 2023. DOI: 10.6028/NIST.FIPS.203. URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.203.pdf.