Physical security of code-based cryptosystems

Séminaire Stéphanois de Maths Accessibles – Institut Camille Jordan

Pierre-Louis Cayrel and Brice Colombier

07/07/2025

Laboratoire Hubert Curien
UMR ▪ CNRS ▪ 5516 ▪ Saint-Étienne

Acknowledgment

Joint work with:

- Vlad-Florin Drăgoi (Univ. Arad, Romania)
- Vincent Grosso (SESAM team, LabHC, Saint-Étienne)
- Nicolas Vallet (SESAM team, LabHC, Saint-Étienne)
- Alexandre Menu (SAS team, EMSE, Gardanne)

Agenda

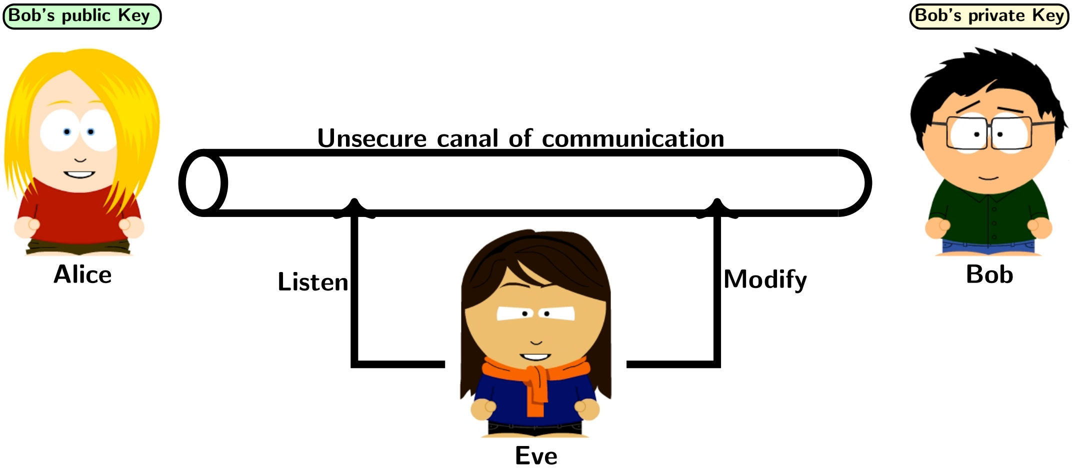
1. Introduction
   - public-key & post-quantum cryptography
   - the *Classic McEliece* cryptosystem
2. Hard problems in code-based cryptography
   - some easier variants
3. Physical attacks
   - side-channel analysis
   - laser fault injection
4. Physical attacks on code-based cryptosystems
   - how to make 2 happen thanks to 3
5. Conclusion & perspectives
   - algebraic structure of side-channel leakage
   - countermeasures

Introduction

Introduction

$\rightarrow$ Public-key and post-quantum cryptography

Cryptography



RSA Algorithm Overview

- **Key Generation:**
  - Choose two large prime numbers p and q.
  - Compute $n = p \times q$.
  - Compute Euler's totient $\varphi(n) = (p-1)(q-1)$.
  - Choose e such that $1 < e < \varphi(n)$ and $\gcd(e, \varphi(n)) = 1$.
  - Compute d as the modular inverse of e:

$$d \equiv e^{-1} \mod \varphi(n).$$

- **Public key:** (e, n)
- **Private key:** (d, n)
- **Encryption:**

$$C = M^e \mod n$$

  where M is the plaintext message.
- **Decryption:**

$$M = C^d \mod n$$

Why is RSA at Risk?

- Quantum computers can run **Shor's algorithm**.
- Shor's algorithm factors large integers efficiently.
- This breaks RSA which rely on factoring.
- Once large quantum computers exist, RSA is no longer secure.

- **Post-Quantum Alternatives :** designed to resist quantum attacks
  - Lattice-based encryption
  - Hash-based signatures
  - **Code-based cryptography**

Introduction

$\rightarrow$ The *Classic McEliece* cryptosystem

Classic McEliece

Classic McEliece is a **Key Encapsulation Mechanism**.

- `KeyGen`(m, n, t) `->` $(\mathrm{k}_{\mathrm{pub}}, \mathrm{k}_{\mathrm{priv}})$

  Generate a **private/public key pair**

- `Encap`$(\mathrm{k}_{\mathrm{pub}})$ `->` $(\mathbf{c}, \mathrm{k}_{\mathrm{session}})$

  Generate a **session key** and encapsulate it into a **ciphertext** using the **public key**

- `Decap`$(\mathbf{c}, \mathrm{k}_{\mathrm{priv}})$ `->` $(\mathrm{k}_{\mathrm{session}})$

  Derive the same **session key** from the **ciphertext** using the **private key**

Classic McEliece key generation

The key generation procedure generates a **private/public key pair**

- `KeyGen`(m, n, t) `->` $(\mathrm{k}_{\mathrm{pub}}, \mathrm{k}_{\mathrm{priv}})$

  Generate a set $\mathcal{L} = \{\alpha_0, \ldots, \alpha_{n-1}\}$ of random elements of $\mathbb{F}_{2^m}$ with $\#\mathcal{L} = n$

  Generate an irreducible monic polynomial $g \in \mathbb{F}_{2^m}[x]$ of degree t

  ...

Classic McEliece encapsulation

The Encapsulation procedure generates a **session key** and encapsulates it.

- `Encap`($\mathsf{k_{pub}}$) `->` ($\mathbf{c}$, $\mathsf{k_{session}}$)

  Generate a random vector $\mathbf{e} \in \mathbb{F}_2^n$ of Hamming weight t
  Compute $\mathbf{c} = \mathbf{H}_{\text{pub}}\mathbf{e}$
  ...

Classic McEliece encapsulation

The Encapsulation procedure generates a **session key** and encapsulates it.

- `Encap`(k_{pub}) `->` ($\mathbf{c}$, $k_{session}$)

  Generate a random vector $\mathbf{e} \in \mathbb{F}_2^n$ of Hamming weight t

  Compute $\mathbf{c} = \mathbf{H}_{pub}\mathbf{e}$

  ...

Classic McEliece decapsulation

The Decapsulation procedure derives the same **session key** from the **ciphertext**.

- `Decap`($\mathbf{c}$, $\mathrm{k_{priv}}$) `->` ($\mathrm{k_{session}}$)

  Compute $\mathbf{v}$ by padding $\mathbf{c}$ with $n - mt$ zeros

  Compute the $2t \times n$ parity-check matrix $\boldsymbol{H}_{priv_{g^2}}$

$$\boldsymbol{H}_{priv_{g^2}} = \begin{pmatrix} g^{-2}(\alpha_0) & \dots & g^{-2}(\alpha_{n-1}) \\ \vdots & \ddots & \vdots \\ \alpha_0^{2t-1} g^{-2}(\alpha_0) & \dots & \alpha_{n-1}^{2t-1} g^{-2}(\alpha_{n-1}) \end{pmatrix}$$

  Compute the syndrome $\mathbf{s} = \boldsymbol{H}_{priv_{g^2}} \mathbf{v}^T$

  ...

Classic McEliece decapsulation

The Decapsulation procedure derives the same **session key** from the **ciphertext**.

- `Decap`($\mathbf{c}$, $\mathrm{k}_{\mathrm{priv}}$) `->` ($\mathrm{k}_{\mathrm{session}}$)

  Compute $\mathbf{v}$ by padding $\mathbf{c}$ with $n - mt$ zeros

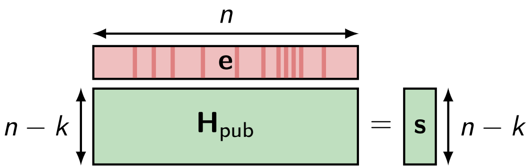
  Compute the $2t \times n$ parity-check matrix $\boldsymbol{H}_{priv_{g^2}}$

$$\boldsymbol{H}_{priv_{g^2}} = \begin{pmatrix} g^{-2}(\alpha_0) & \dots & g^{-2}(\alpha_{n-1}) \\ \vdots & \ddots & \vdots \\ \alpha_0^{2t-1} g^{-2}(\alpha_0) & \dots & \alpha_{n-1}^{2t-1} g^{-2}(\alpha_{n-1}) \end{pmatrix}$$

  Compute the syndrome $\mathbf{s} = \boldsymbol{H}_{priv_{g^2}} \mathbf{v}^T$

  ...

Classic McEliece parameters



n	k	$(n-k)$	t
3488	2720	768	64
4608	3360	1248	96
6688	5024	1664	128
6960	5413	1547	119
8192	6528	1664	128

The public key $\mathbf{H}_{\text{pub}}$ is **very large** (up to 1.7 MB).

Hard problems in code-based cryptography

Hard problems in code-based cryptography

$\rightarrow$ Syndrome decoding problem

Syndrome decoding problem

Syndrome decoding problem

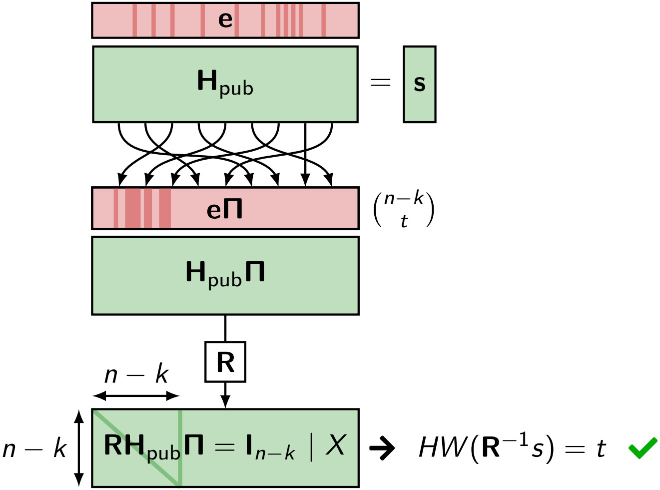
Input: a binary parity-check matrix $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}$
a binary vector $\mathbf{s} \in \mathbb{F}_2^{n-k}$
a scalar $t \in \mathbb{N}^+$

Output: a binary vector $\mathbf{x} \in \mathbb{F}_2^n$ with a Hamming weight $\mathrm{HW}(\mathbf{x}) \leq t$ such that: $\mathbf{Hx} = \mathbf{s}$

Known to be an NP-complete problem.

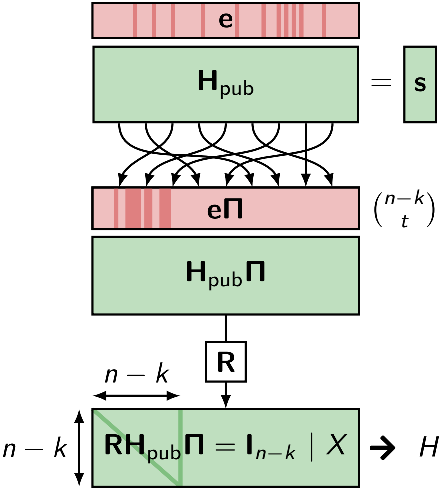
Information-set decoding-based strategies

Information Set Decoding *à la* Prange



Information-set decoding-based strategies

Information Set Decoding *à la* Prange



$\rightarrow$ $HW(\mathbf{R}^{-1}s) = t$ ✓

Can be improved by allowing δ ones in the last k positions of $e\mathbf{\Pi}$ and use more advanced ISD variants.

Hard problems in code-based cryptography

$\rightarrow$ Syndrome decoding problem over $\mathbb{N}$

Syndrome decoding problem

Binary syndrome decoding problem (Binary SDP)

Input: a binary parity-check matrix $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}$
a binary vector $\mathbf{s} \in \mathbb{F}_2^{n-k}$
a scalar $t \in \mathbb{N}^+$

Output: a binary vector $\mathbf{x} \in \mathbb{F}_2^n$ with a Hamming weight $\mathrm{HW}(\mathbf{x}) \leq t$ such that: $\mathbf{Hx} = \mathbf{s}$

Integer syndrome decoding problem ($\mathbb{N}$-SDP)

Input: a binary parity-check matrix $\mathbf{H} \in \{0,1\}^{(n-k)\times n}$
a ~~binary~~ vector $\mathbf{s} \in \mathbb{N}^{n-k}$
a scalar $t \in \mathbb{N}^+$

Output: a binary vector $\mathbf{x} \in \{0,1\}^n$ with a Hamming weight $\mathrm{HW}(\mathbf{x}) \leq t$ such that:
$\mathbf{Hx} = \mathbf{s}$

$\mathbb{N}$-SDP as an optimisation problem

Option 1: Consider $\mathbf{H}_{\text{pub}}\mathbf{e} = \mathbf{s}$ as an **optimization problem** and solve it.

Integer syndrome decoding problem ($\mathbb{N}$-SDP)

Input: a matrix $\mathbf{H}_{\text{pub}} \in \mathcal{M}_{n-k,n}(\mathbb{N})$ with $h_{i,j} \in \{0,1\}$ for all i, j
a vector $\mathbf{s} \in \mathbb{N}^{n-k}$
a scalar $t \in \mathbb{N}^{+}$

Output: a vector $\mathbf{e}$ in $\mathbb{N}^n$ with $x_i \in \{0,1\}$ for all i
and with a Hamming weight $\mathrm{HW}(\mathbf{x}) \leq t$ such that: $\mathbf{H}_{\text{pub}}\mathbf{e} = \mathbf{s}$

ILP problem

Let $\mathbf{b} \in \mathbb{N}^n$, $\mathbf{c} \in \mathbb{N}^m$ and $\mathbf{A} \in \mathcal{M}_{n-k,n}(\mathbb{N})$ then:

$$\min\{\mathbf{b}^T\mathbf{x} \mid \mathbf{A}\mathbf{x} = \mathbf{c}, \mathbf{x} \in \mathbb{N}^n, \mathbf{x} \geq 0\}$$

with $\mathbf{b} = (1, 1, ..., 1)$ and $\mathbf{x} \in \{0,1\}^n$

$\mathbb{N}$-SDP as an optimisation problem

Option 1: Consider $\mathbf{H}_{\text{pub}}\mathbf{e} = \mathbf{s}$ as an **optimization problem** and solve it.

Integer syndrome decoding problem ($\mathbb{N}$-SDP)

Input: a matrix $\mathbf{H}_{\text{pub}} \in \mathcal{M}_{n-k,n}(\mathbb{N})$ with $h_{i,j} \in \{0,1\}$ for all i, j
a vector $\mathbf{s} \in \mathbb{N}^{n-k}$
a scalar $t \in \mathbb{N}^{+}$

Output: a vector $\mathbf{e}$ in $\mathbb{N}^{n}$ with $x_i \in \{0,1\}$ for all i
and with a Hamming weight $\text{HW}(\mathbf{x}) \leq t$ such that: $\mathbf{H}_{\text{pub}}\mathbf{e} = \mathbf{s}$

ILP problem

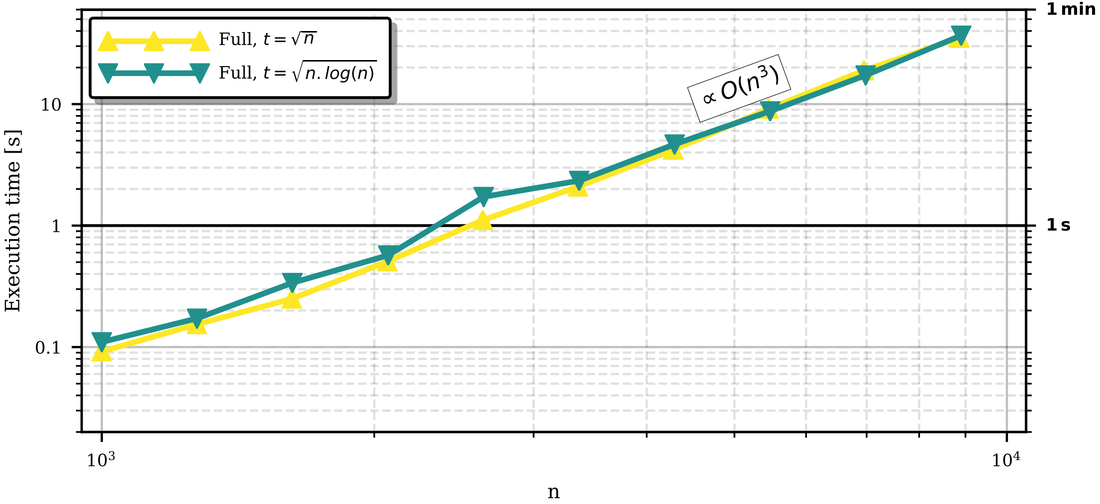
Let $\mathbf{b} \in \mathbb{N}^{n}$, $\mathbf{c} \in \mathbb{N}^{m}$ and $\mathbf{A} \in \mathcal{M}_{n-k,n}(\mathbb{N})$ then:

$$\min\{\mathbf{b}^{T}\mathbf{x} \mid \mathbf{A}\mathbf{x} = \mathbf{c}, \mathbf{x} \in \mathbb{N}^{n}, \mathbf{x} \geq 0\}$$

with $\mathbf{b} = (1, 1, ..., 1)$ and $\mathbf{x} \in \{0,1\}^{n}$

Solved by **integer linear programming** (using `Scipy.optimize.linprog` for example)

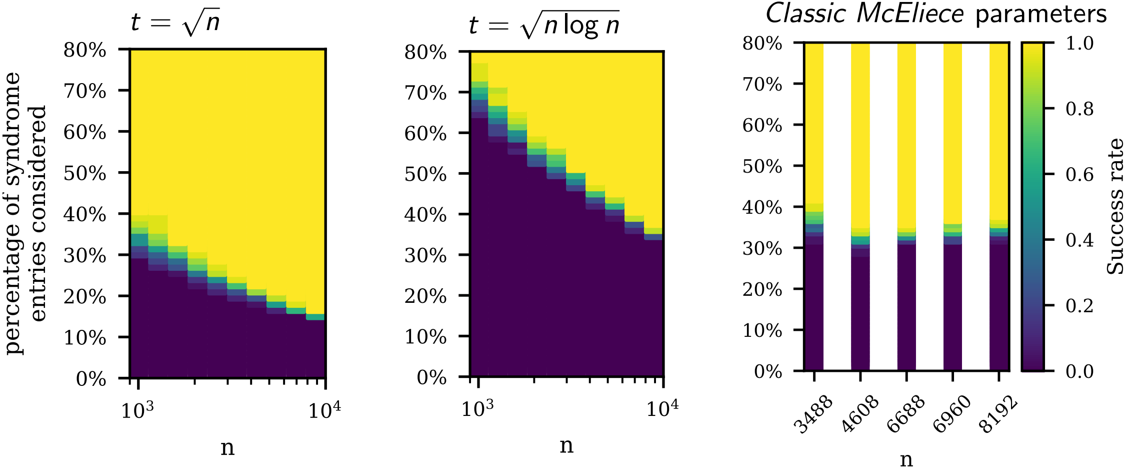
Integer Linear Programming



For *Classic McEliece*: $3488 < n < 8192$

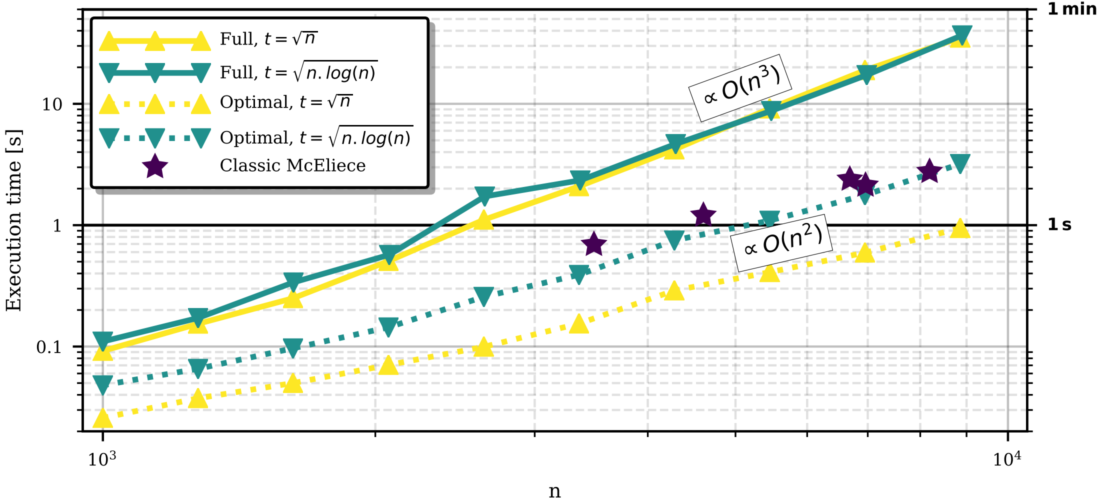
Required fraction of faulty syndrome entries

We observed that only a **fraction** of the faulty syndrome entries is enough to solve the problem.



For *Classic McEliece*, **less than** 40 % faulty syndrome entries is enough.

Experimental results



When considering the **optimal fraction**, time complexity drops from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2)$.
The largest parameters can be attacked in **a few seconds** on a desktop computer.

$\mathbb{N}$-SDP as an optimization problem: summary

Considering the $\mathbb{N}$-SDP as an optimization problem [1] [2]

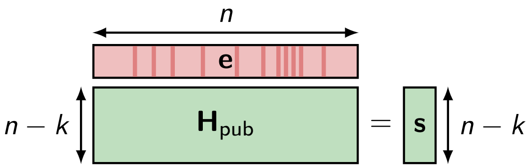
- 👍 easy to **express**,
- 👍 allows to use a **generic ILP solver**,
- 👍 is reasonably **efficient**,
- 👎 does not tolerate **any error** in the integer syndrome.

[1] Vlad-Florin Dragoi et al. "Solving a Modified Syndrome Decoding Problem Using Integer Programming". In: **International Journal of Computers Communications & Control** (2020).

[2] Pierre-Louis Cayrel et al. "Message-Recovery Laser Fault Injection Attack on the Classic McEliece Cryptosystem". In: **EUROCRYPT**. 2021.

How to solve the $\mathbb{N}$-SDP efficiently ?

Option 2: Reframe $\mathbf{H}_{\text{pub}}\mathbf{e} = \mathbf{s}$

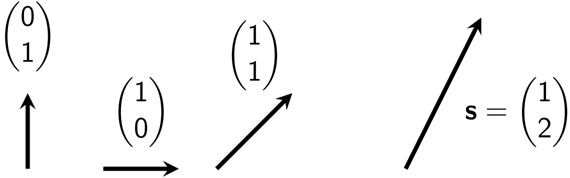


We want to find **which columns** of $\mathbf{H}_{\text{pub}}$ **contributed** to $\mathbf{s}$.

The score function

Example: $t = 2 = \mathrm{HW}(\mathbf{e})$

$$\mathbf{H}_{\mathrm{pub}}\mathbf{e} = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} .\mathbf{e} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$$



The dot product can be used to compute a **score** for a column:

Score function

$$\psi_i(\mathbf{s}) = \mathbf{H}_{pub[,i]} \cdot \mathbf{s} + \bar{\mathbf{H}}_{pub[,i]} \cdot \bar{\mathbf{s}} \qquad \text{with } \bar{\mathbf{H}} = 1 - \mathbf{H} \qquad \text{and } \bar{\mathbf{s}} = t - \mathbf{s}$$

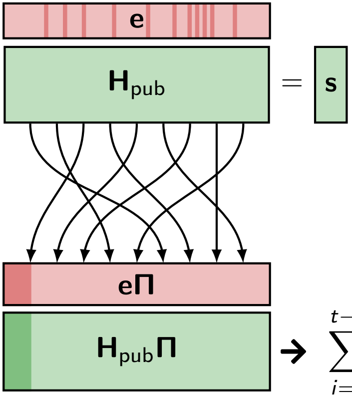
$$\psi_0(\mathbf{s}) = 3 \qquad \psi_1(\mathbf{s}) = 1 \qquad \psi_2(\mathbf{s}) = 3$$

From the score to the support

Algorithm 1 Permutation from score

for `i` $\leftarrow 0$ to $n-1$ **do**
  Compute $\psi_i(\mathbf{s})$
$\mathbf{\Pi} \leftarrow$ sort $\psi(\mathbf{s})$ in descending order
Return $\mathbf{\Pi}$

Best-case scenario: t-threshold decoder



$$\rightarrow \sum_{i=0}^{t-1} (\mathbf{H}_{\text{pub}}\mathbf{\Pi})_{[,:t]} = \mathbf{s}$$ ✓

Solving $\mathbb{N}$-SDP with the score function

Solving $\mathbb{N}$-SDP with the score function [3]

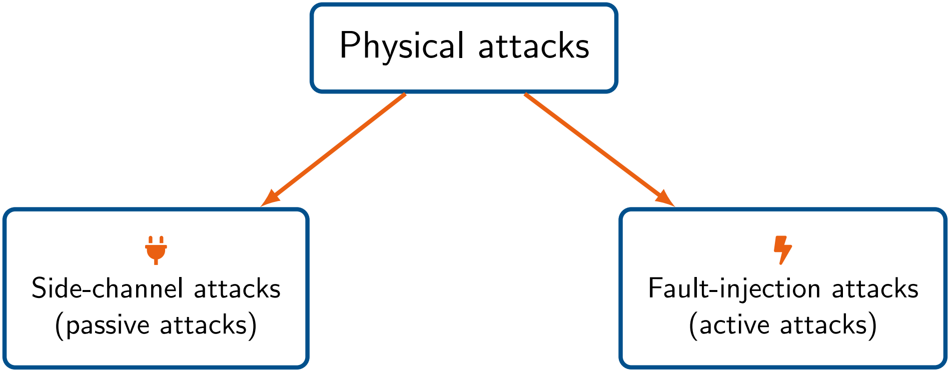
- is **computationally efficient**
- tolerates **some errors** in the integer syndrome
- gets **more efficient** with **larger** cryptographic parameters
- does **not** cope so well with **high noise levels**

[3] Brice Colombier et al. "Profiled Side-Channel Attack on Cryptosystems Based on the Binary Syndrome Decoding Problem". In: **IEEE Transactions on Information Forensics and Security** (2022).

Physical attacks

Physical attacks

In a physical attack, an attacker has a **physical access** to the device.



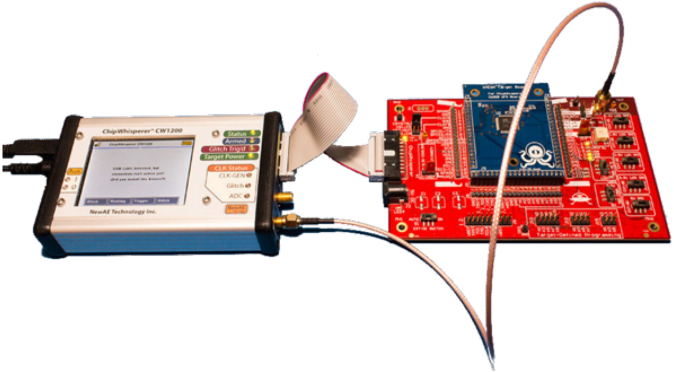
Physical attacks

$\rightarrow$ Side-channel attacks

Side-channel attacks: attacker model

Attacker model: an attacker can **measure a physical quantity** while **the device is running**.

- execution time [4]
- power consumption [5]
- electromagnetic radiation [6]



[4] Paul C. Kocher. "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems". In: **CRYPTO**. 1996

[5] Paul C. Kocher et al. "Differential Power Analysis". In: **CRYPTO**. 1999

[6] Karine Gandolfi et al. "Electromagnetic Analysis: Concrete Results". In: **CHES**. 2001

Side-channel attacks: leakage model

Leakage model: relation between the physical quantity and the (secret) data being processed.

$$\mathcal{L}(d) = f(d) + \mathcal{N}(\mu, \sigma)$$

Identity $\mathcal{L}(d) = d + \mathcal{N}(\mu, \sigma)$

Hamming weight $\mathcal{L}(d) = \mathrm{HW}(d) + \mathcal{N}(\mu, \sigma)$ with $\mathrm{HW}(a) = \{i \in [0; n-1] \mid a_i \neq 0\}$

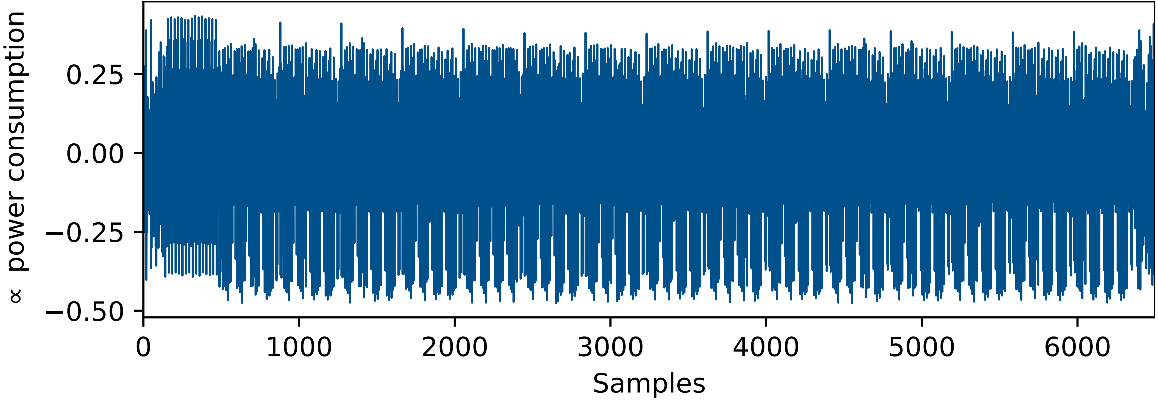
→ number of bits equal to 1

Hamming distance $\mathcal{L}(d) = \mathrm{HD}(d, d^-) + \mathcal{N}(\mu, \sigma)$ with $\mathrm{HD(a, b)} = \mathrm{HW(a \oplus b)}$

→ number of bits that are different

Side-channel trace

Single execution of the target program:

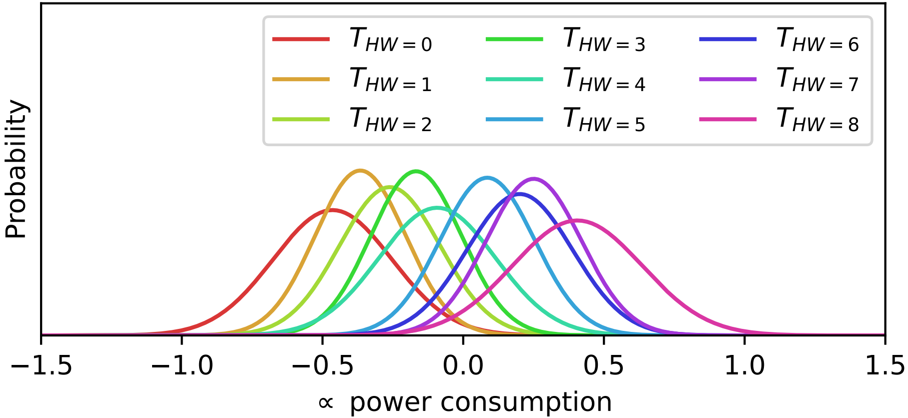


Repeated many times to build a sufficiently large dataset.

Profiled side-channel attack: profiling phase

Step 1/2: profiling phase

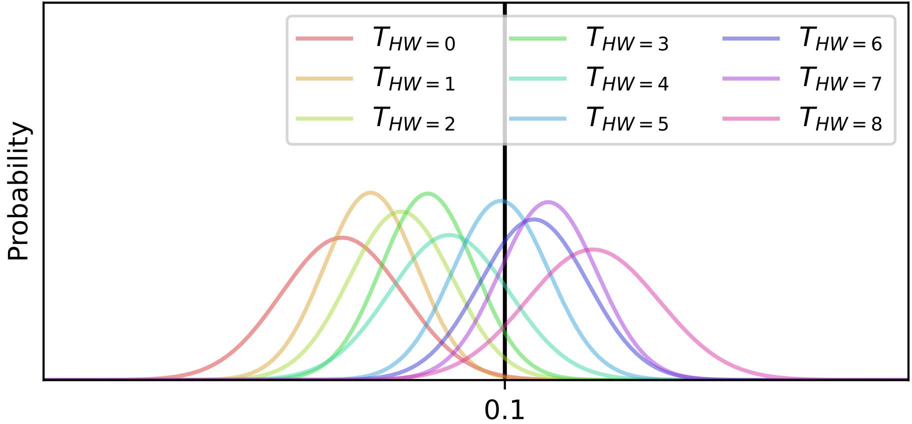
Build Gaussian templates (μ, Σ) for every possible leakage value (w.r.t the leakage model)



Profiled side-channel attack: matching phase

Step 2/2: matching phase

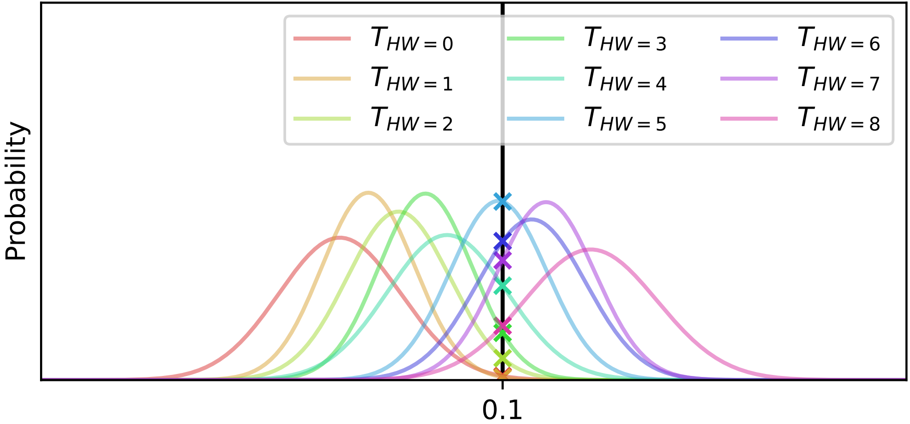
Match an observation against the previously built templates.



Profiled side-channel attack: matching phase

Step 2/2: matching phase

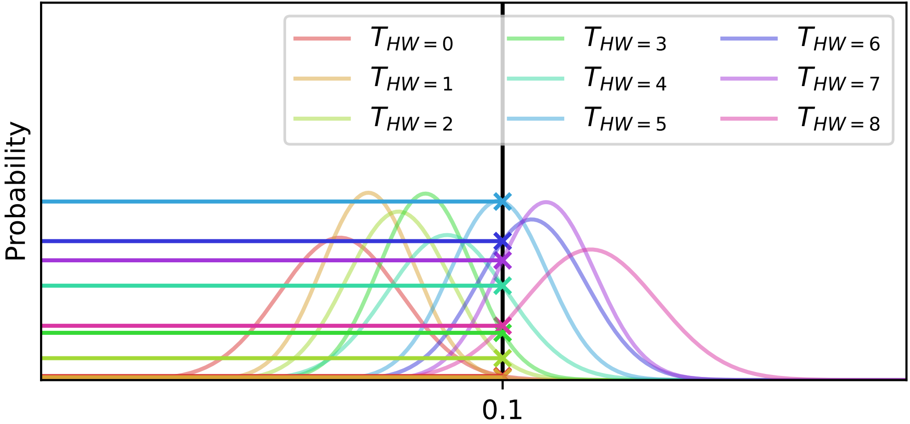
Match an observation against the previously built templates.



Profiled side-channel attack: matching phase

Step 2/2: matching phase

Match an observation against the previously built templates.



Physical attacks

$\rightarrow$ Fault injection attacks

Fault injection attacks: attacker model

Attacker model: an attacker can **inject faults inside the device** while **it is operating**.

- Heat
- Clock glitches
- Power glitches
- Electromagnetic pulses
- Laser pulses
- X-rays

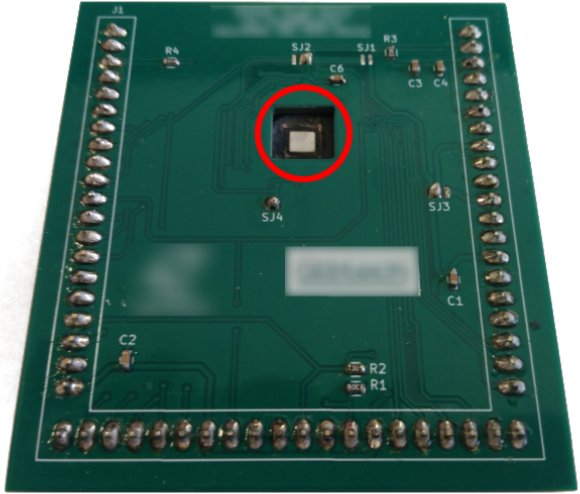
Fault injection attacks: attacker model

Attacker model: an attacker can **inject faults inside the device** while **it is operating**.

- Heat
- Clock glitches
- Power glitches
- Electromagnetic pulses
- Laser pulses
- X-rays

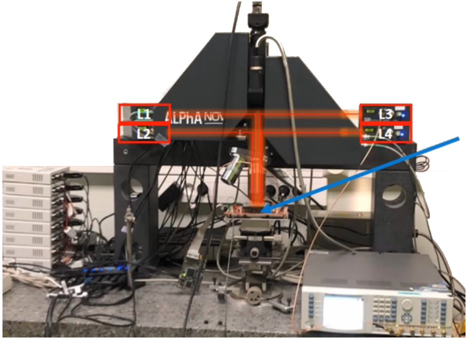
Laser fault injection attack: sample preparation

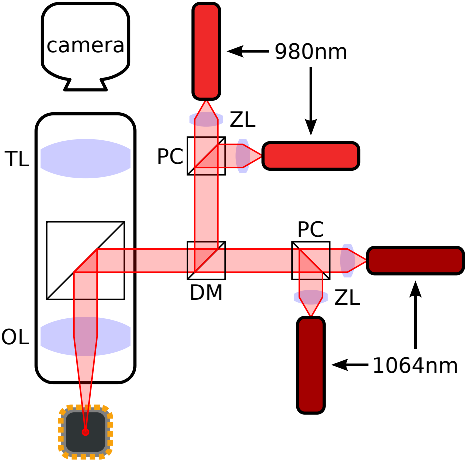
Backside access to the chip:



Laser fault injection attack: setup

Infrared 4-spot laser fault injection setup [7]





[7] Brice Colombier et al. "Multi-Spot Laser Fault Injection Setup: New Possibilities for Fault Injection Attacks". In: **CARDIS**. Nov. 2021.

Laser fault injection attack: fault model

Fault model: Concise description of **the effects of the fault** happening inside the device.

Fault model for laser fault injection in Flash memory [8] [9]:

- down to single-bit
- bit-set ($0 \rightarrow 1$)
- temporary (stored data is not affected)

[8] Brice Colombier et al. "Laser-Induced Single-bit Faults in Flash Memory: Instructions Corruption on a 32-Bit Microcontroller". In: **IEEE HOST**. May 2019.
[9] Alexandre Menu et al. "Single-Bit Laser Fault Model in NOR Flash Memories: Analysis and Exploitation". In: **FDTC**. Sept. 2020.

Laser fault injection: instruction corruption fault model

Instruction corruption: changing the **opcode**.

bits	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
`EORS`: Rd = Rm $\oplus$ Rn	0	1	0	0	0	0	0	0	0	1	Rm			Rdn		
`EORS`: R1 = R0 $\oplus$ R1	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	1
`ADCS`: R1 = R0 + R1	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	1

Physical attacks on code-based cryptosystems

Physical attacks on code-based cryptosystems

$\rightarrow$ Message-recovery attacks on the encapsulation step

Classic McEliece encapsulation

The Encapsulation procedure generates a **session key** and encapsulates it.

- `Encap`(k_{pub}) `->` ($\mathbf{c}$, $\text{k}_{\text{session}}$)

  Generate a random vector $\mathbf{e} \in \mathbb{F}_2^n$ of Hamming weight t

  Compute $\mathbf{c} = \mathbf{H}_{\text{pub}}\mathbf{e}$

  ...

Classic McEliece encapsulation

The Encapsulation procedure generates a **session key** and encapsulates it.

- `Encap`(k_{pub}) `->` ($\mathbf{c}$, $k_{session}$)

  Generate a random vector $\mathbf{e} \in \mathbb{F}_2^n$ of Hamming weight t

  Compute $\mathbf{c} = \mathbf{H}_{pub}\mathbf{e}$

  ...

Syndrome computation: matrix-vector multiplication over $\mathbb{F}_2$

Perform the matrix-vector multiplication over $\mathbb{N}$ instead of $\mathbb{F}_2$ [10].

Algorithm 2 Schoolbook matrix-vector multiplication over $\mathbb{F}_2$

```
function MAT_VEC_MULT_SCHOOLBOOK(mat, vec)
  for row ← 0 to n − k − 1 do
    syn[row] = 0                                  ▷ Initialization
  for row ← 0 to n − k − 1 do
    for col ← 0 to n − 1 do
      syn[row] ^= mat[row][col] & vec[col]        ▷ multiply-accumulate
  return syn
```

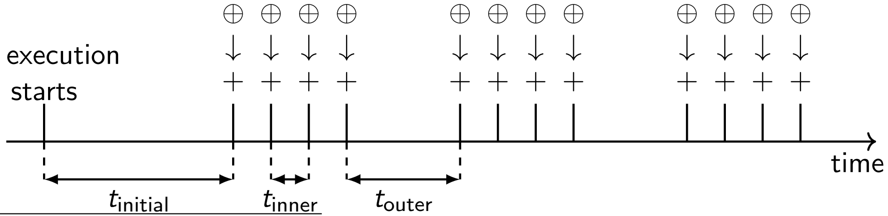
[10]Pierre-Louis Cayrel et al. “Message-Recovery Laser Fault Injection Attack on the Classic McEliece

Syndrome computation: matrix-vector multiplication over $\mathbb{F}_2$

Perform the matrix-vector multiplication over $\mathbb{N}$ instead of $\mathbb{F}_2$ [10].

Algorithm 2 Schoolbook matrix-vector multiplication over $\mathbb{F}_2$

```
function MAT_VEC_MULT_SCHOOLBOOK(mat, vec)
  for row ← 0 to n − k − 1 do
    syn[row] = 0                                  ▷ Initialization
  for row ← 0 to n − k − 1 do
    for col ← 0 to n − 1 do
      syn[row] ^= mat[row][col] & vec[col]        ▷ multiply-accumulate
  return syn
```



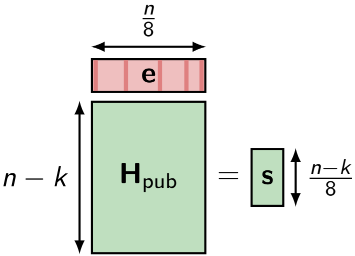
[10]Pierre-Louis Cayrel et al. “Message-Recovery Laser Fault Injection Attack on the Classic McEliece

Packed matrix-vector multiplication

Objection: the schoolbook matrix-vector multiplication algorithm is **highly inefficient**!
Each **machine word** stores only **one bit**: a **lot** of memory is wasted.

Algorithm 3 Packed matrix-vector multiplication

function MAT_VEC_MULT_PACKED(`mat, vec`)
  for `row` $\leftarrow$ 0 to $((n-k)/8-1)$ **do**
    `syn[row] = 0` ▷ Initialisation
  for `row` $\leftarrow$ 0 to $(n-k-1)$ **do**
    $b = 0$
    for `col` $\leftarrow$ 0 to $(n/8-1)$ **do**
      `b ^= mat[row][col] & vec[col]`
    b ^= b >> 4
    b ^= b >> 2 ▷ Exclusive-OR folding
    b ^= b >> 1
    b &= 1 ▷ LSB extraction
    `syn[row/8] |=` b `<< (row % 8)` ▷ Packing
  return `syn`



Packed matrix-vector multiplication: side-channel analysis

Algorithm 4 Packed matrix-vector multiplication

```
1: ...
2: for col ← 0 to (n/8 − 1) do
3:   b ^= mat[row][col] & vec[col]
4: ...
```

```
b = 00000000

b = 00000000

b = 00001000

b = 00001000

b = 00001010
```

Packed matrix-vector multiplication: side-channel analysis

Algorithm 4 Packed matrix-vector multiplication

...
for `col` $\leftarrow 0$ to $(n/8 - 1)$ **do**
    `b ^= mat[row][col] & vec[col]`
...

`b = 00000000` HW=0
HD = 0
`b = 00000000` HW=0
HD = 1
`b = 00001000` HW=1
HD = 0
`b = 00001000` HW=1
HD = 1
`b = 00001010` HW=2

Packed matrix-vector multiplication: side-channel analysis

Algorithm 4 Packed matrix-vector multiplication

```
...
for col ← 0 to (n/8 − 1) do
  b ^= mat[row][col] & vec[col]
...
```

	b = 00000000	HW=0
HD = 0		
	b = 00000000	HW=0
HD = 1		
	b = 00001000	HW=1
HD = 0		
	b = 00001000	HW=1
HD = 1		
	b = 00001010	HW=2

Integer syndrome from Hamming distances or Hamming weights

$$s_j = \sum_{i=1}^{\frac{n}{8}-1} \mathrm{HD}(\mathbf{b}_{j,i}, \mathbf{b}_{j,i-1})$$

$$= \sum_{i=1}^{\frac{n}{8}-1} \left| \mathrm{HW}(\mathbf{b}_{j,i}) - \mathrm{HW}(\mathbf{b}_{j,i-1}) \right| \quad \text{if } \mathrm{HD}(\mathbf{b}_{j,i}, \mathbf{b}_{j,i-1}) \leq 1$$

	b = 00001000	HW=1
HD = 2		
	b = 00000100	HW=1

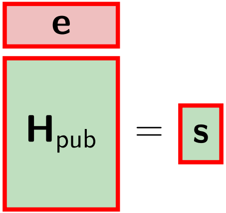
Happens if:

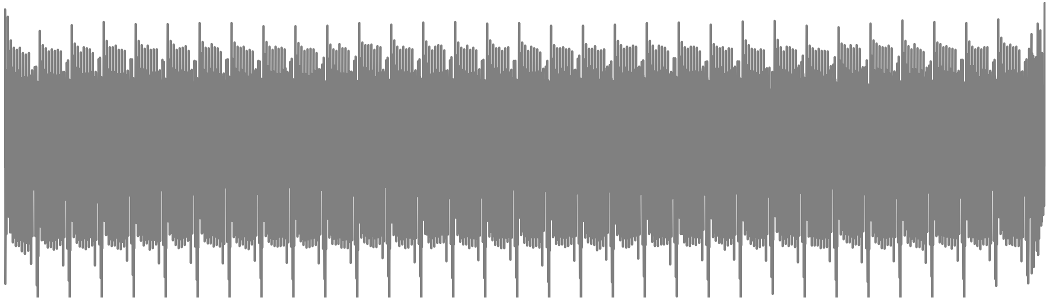
HW(mat[r][c] & vec[c]) > 1

Unlikely since $\mathrm{HW}(\mathbf{e}) = t$ is low.

Side-channel analysis for Hamming weight recovery

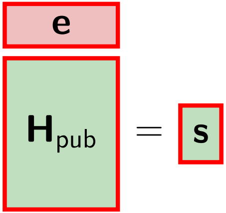
$$\mathbf{s} = \mathbf{H}_{\text{pub}}\mathbf{e}$$





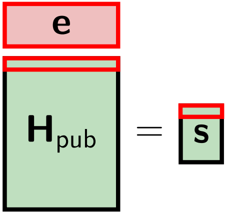
Side-channel analysis for Hamming weight recovery

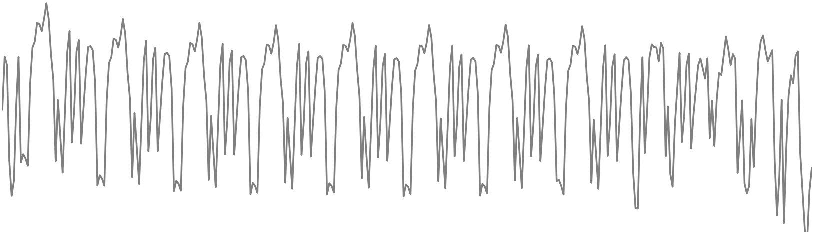
$$\mathbf{s} = \mathbf{H}_{\text{pub}}\mathbf{e}$$



Side-channel analysis for Hamming weight recovery

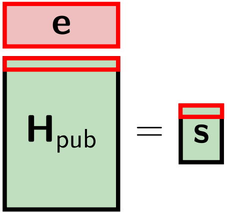
$$\mathbf{s}_j = \mathbf{H}_{pub_{[j,]}} \mathbf{e}$$

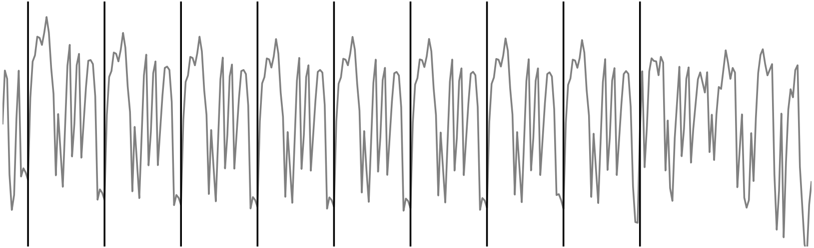




Side-channel analysis for Hamming weight recovery

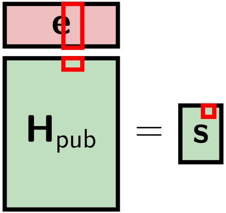
$$\mathbf{s}_j = \mathbf{H}_{pub_{[j,]}}\mathbf{e}$$





Side-channel analysis for Hamming weight recovery

`b ^=` $\mathbf{H}_{pub_{[j,i]}}\mathbf{e}_i$



Physical attacks on code-based cryptosystems

$\rightarrow$ Key-recovery attack on the decapsulation step

Classic McEliece decapsulation

The Decapsulation procedure derives the same **session key** from the **ciphertext**.

- `Decap`($\mathbf{c}$, k_{priv}) `->` ($k_{session}$)

  Compute $\mathbf{v}$ by padding $\mathbf{c}$ with $n - mt$ zeros

  Compute the $2t \times n$ parity-check matrix $\boldsymbol{H}_{priv_{g^2}}$

$$\boldsymbol{H}_{priv_{g^2}} = \begin{pmatrix} g^{-2}(\alpha_0) & \dots & g^{-2}(\alpha_{n-1}) \\ \vdots & \ddots & \vdots \\ \alpha_0^{2t-1} g^{-2}(\alpha_0) & \dots & \alpha_{n-1}^{2t-1} g^{-2}(\alpha_{n-1}) \end{pmatrix}$$

  Compute the syndrome $\mathbf{s} = \boldsymbol{H}_{priv_{g^2}} \mathbf{v}^T$

  ...

Side-channel leakage during the parity check matrix computation

$$\boldsymbol{H}_{priv_{g^2}} = \begin{pmatrix} g^{-2}(\alpha_0) & \dots & g^{-2}(\alpha_{n-1}) \\ \vdots & \ddots & \vdots \\ \alpha_0^{2t-1} g^{-2}(\alpha_0) & \dots & \alpha_{n-1}^{2t-1} g^{-2}(\alpha_{n-1}) \end{pmatrix}$$

$$\Big\downarrow$$

$$\begin{pmatrix} \mathrm{HW}(g^{-2}(\alpha_0)) & \dots & \mathrm{HW}(g^{-2}(\alpha_{n-1})) \\ \vdots & \ddots & \vdots \\ \mathrm{HW}(\alpha_0^{2t-1} g^{-2}(\alpha_0)) & \dots & \mathrm{HW}(\alpha_{n-1}^{2t-1} g^{-2}(\alpha_{n-1})) \end{pmatrix}$$

Side-channel leakage during the parity check matrix computation

$$\boldsymbol{H}_{priv_{g^2}} = \begin{pmatrix} g^{-2}(\alpha_0) & \dots & g^{-2}(\alpha_{n-1}) \\ \vdots & \ddots & \vdots \\ \alpha_0^{2t-1} g^{-2}(\alpha_0) & \dots & \alpha_{n-1}^{2t-1} g^{-2}(\alpha_{n-1}) \end{pmatrix}$$

$$\Big\downarrow$$

$$\begin{pmatrix} \mathrm{HW}(g^{-2}(\alpha_0)) & \dots & \mathrm{HW}(g^{-2}(\alpha_{n-1})) \\ \vdots & \ddots & \vdots \\ \mathrm{HW}(\alpha_0^{2t-1} g^{-2}(\alpha_0)) & \dots & \mathrm{HW}(\alpha_{n-1}^{2t-1} g^{-2}(\alpha_{n-1})) \end{pmatrix}$$

Hamming weight distinguisher

Let $g(\alpha)^{-2} = \beta$

The sequence $\left(\mathrm{HW}(\alpha^j \beta)\right)_{j=0}^{2t-1}$ is a very good distinguisher for the (α, β) pair.

Exploiting the sequence of Hamming weights

Require: A side-channel trace of the execution of the *Classic McEliece* Decapsulation
Ensure: The private key $\mathsf{sk} = (g, \mathcal{L})$

1: Estimate the Hamming weight of $\left(\alpha^i g(\alpha)^{-2}\right)_{i=0}^{2t-1} \forall \alpha \in \mathcal{L}$
2: Recover $mt + \delta$ pairs $(\alpha, g(\alpha))$
3: Recover polynomial g from t pairs $(\alpha, g(\alpha))$ ▷ via interpolation
4: Construct the Vandermonde matrix $\boldsymbol{V}$ using g and the $mt + \delta$ pairs
5: Compute the change-of-basis $\boldsymbol{S}$ using $\boldsymbol{V}$ and $\mathbf{H}_{\mathrm{pub}}$
6: Recover $\boldsymbol{H}_{\mathrm{priv}_g} = \boldsymbol{S}^{-1}\mathbf{H}_{\mathrm{pub}}$
7: Recover the full permuted support $\mathcal{L} = \frac{\boldsymbol{H}_{\mathrm{priv}_g}[1:]}{\boldsymbol{H}_{\mathrm{priv}_g}[0:]} \left(= \frac{\alpha_j g^{-1}(\alpha_j)}{g^{-1}(\alpha_j)}\right)$

Perspectives & open questions

Perspectives

- Other code-based cryptosystems
- Design **countermeasures** to prevent the aformentionned attacks from happeningx
  - Program-level
  - Algorithm-level
- Better study the mathematical structures involved in the side-channel leakage

Several open questions to ICJ mathematicians

- How can we efficiently find the $\alpha_i \in \mathbb{F}_{2^m}$ and $\beta_i \in \mathbb{F}^*_{2^m}$ from the Hamming weight leakage?
- How can we correct errors during the measurements? How many?
- We notice that generating polynomial for $\mathbb{F}^*_{2^m}$ matters, why?
- How to choose the $f(X)$ of degree m to make the attack harder?
- Is the Hamming weight of the values the best source of information?

Work in progress with Michaël Bulois (ICJ) to be presented at SAC 2025, to be continued ...
(thanks to FIL, two master students will help us to answer them)