Physical Security of Code-based Cryptosystems based on the Syndrome Decoding Problem

IAA/IMATH Seminar

Brice Colombier, Pierre-Louis Cayrel, Vlad-Florin Drăgoi, Vincent Grosso

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

UNIVERSITATEA AUREL VLAICU din ARAD

November 16th 2023

Context

2016: NIST called for proposals for **post-quantum cryptography** algorithms

- Digital signature
- **Key encapsulation mechanisms**

Four rounds:

2017 Round 1

2019 Round 2

2020 Round 3: CRYSTALS-Kyber (lattices)

2022 Round 4: 3 candidates left

- BIKE
- HQC
- **ClassicMcEliece** [1]

Research challenges

- *"More hardware implementations"*
- *"Side-channel attacks / resistant implem."*

Dustin Moody (NIST), PKC 2022

[1] M. R. Albrecht et al. **Classic McEliece: conservative code-based cryptography: cryptosystem specification**. Tech. rep. National Institute of Standards and Technology, 2022

Classic McEliece

Classic McEliece

Classic McEliece is a **Key Encapsulation Mechanism**, based on the **Niederreiter cryptosystem** [2].

- `KeyGen()` -> ($\mathbf{H}_{\text{pub}}$, k_{priv})
- `Encap`($\mathbf{H}_{\text{pub}}$) -> ($\mathbf{s}$, k_{session})
- `Decap`($\mathbf{s}$, k_{priv}) -> (k_{session})

The Encapsulation procedure establishes a **shared secret**.

- `Encap`($\mathbf{H}_{\text{pub}}$) -> ($\mathbf{s}$, k_{session})
  - Generate a random vector $\mathbf{e} \in \mathbb{F}_2^n$ of Hamming weight t
  - Compute $\mathbf{s} = \mathbf{H}_{\text{pub}}\mathbf{e}$
  - Compute the hash: $k_{\text{session}} = \mathrm{H}(1, \mathbf{e}, \mathbf{s})$

[2] H. Niederreiter. "Knapsack-Type Cryptosystems and Algebraic Coding Theory". In: **Problems of Control and Information Theory** (1986).

Security

The security of the Niederreiter cryptosystem relies on the **syndrome decoding problem**.

Syndrome decoding problem

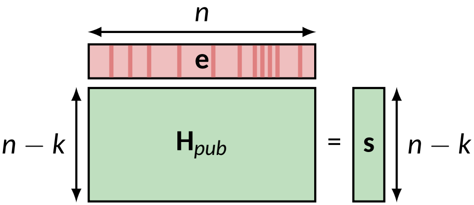
Input: a binary matrix $\mathbf{H} \in \mathbb{F}_2^{(n-k)\times n}$
a binary vector $\mathbf{s} \in \mathbb{F}_2^{n-k}$
a scalar $t \in \mathbb{N}^+$

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-hard** problem [3].

[3] E. R. Berlekamp et al. "On the inherent intractability of certain coding problems (Corresp.)". In: **IEEE Transactions on Information Theory** (1978).

Classic McEliece parameters



n	k	t	Equivalent bit-level security
3488	2720	64	128
4608	3360	96	196
6688	5024	128	256
6960	5413	119	256
8192	6528	128	256

The public key $\mathbf{H}_{pub}$ is **very large**!

Hardware implementations

Implementations on embedded systems are now feasible : [4] [5] [6]
Reference hardware target : ARM® Cortex®-M4

Several **strategies** to store the (very large) keys :

- Streaming,
- Use a structured code,
- Use a very large microcontroller.

New threats

That makes them vulnerable to **physical attacks** (fault injection & side-channel analysis)

---

[4] S. Heyse. “Low-Reiter: Niederreiter Encryption Scheme for Embedded Microcontrollers”. In: **International Workshop on Post-Quantum Cryptography**. 2010.

[5] J. Roth et al. “Classic McEliece Implementation with Low Memory Footprint”. In: **CARDIS**. 2020.

[6] M. Chen et al. “Classic McEliece on the ARM Cortex-M4”. In: **IACR Transactions on Cryptographic Hardware and Embedded Systems** (2021).

“Modified” syndrome decoding problem

Syndrome decoding problem

Binary syndrome decoding problem (Binary SDP)

Input: a binary 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 : $\mathbf{Hx} = \mathbf{s}$

Syndrome decoding problem

Binary syndrome decoding problem (Binary SDP)

Input: a binary 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 : $\mathbf{Hx} = \mathbf{s}$

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

Input: a binary 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 : $\mathbf{Hx} = \mathbf{s}$

Syndrome decoding problem

Binary syndrome decoding problem (Binary SDP)

Input: a binary 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 : $\mathbf{Hx} = \mathbf{s}$

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

Input: a binary matrix $\mathbf{H} \in \{0,1\}^{(n-k)\times n}$
a ~~binary~~ vector $\mathbf{s} \in \mathbb{N}^{n-k}$ ← How do we get this integer syndrome?
a scalar $t \in \mathbb{N}^+$

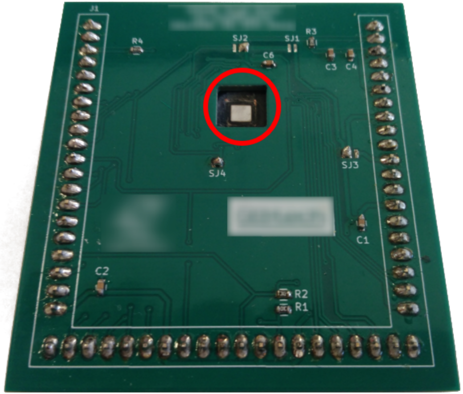
Output: a binary vector $\mathbf{x} \in \{0,1\}^n$ with a Hamming weight $\mathrm{HW}(\mathbf{x}) \leq t$ such that : $\mathbf{Hx} = \mathbf{s}$

Physical attack #1: Fault injection

Laser fault injection attack

Physical attack : an attacker has a **physical access** to the device.

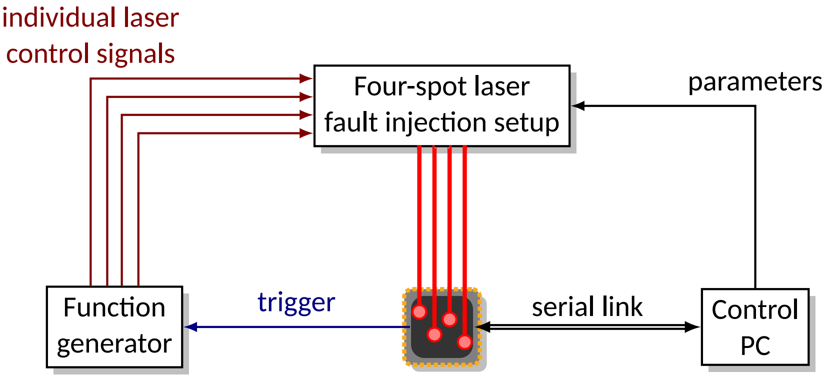
- ChipWhisperer platform [7],
- Custom board with an opening,
- Decapsulated chip
  - access to the backside of the die



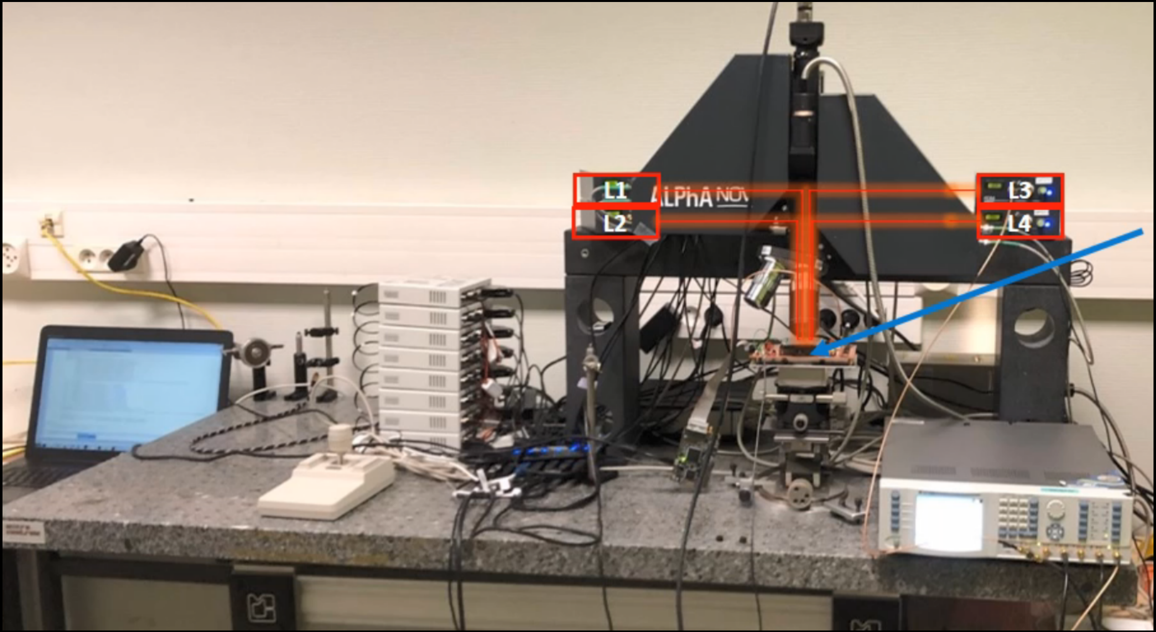
[7] C. O'Flynn et al. "ChipWhisperer: An Open-Source Platform for Hardware Embedded Security Research". In: **COSADE**. 2014

Laser fault injection setup

4-spot laser fault injection setup [8]



[8] B. Colombier et al. "Multi-spot Laser Fault Injection Setup: New Possibilities for Fault Injection Attacks". In: **CARDIS**. 2021.



L1
L2
L3
L4

Syndrome computation : $\mathbf{Hx} = \mathbf{s}$

In [9] we target the **syndrome computation**: $\mathbf{s} = \mathbf{H}_{pub}\mathbf{e}$

Matrix-vector multiplication performed over $\mathbb{F}_2$

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

```
function MAT_VEC_MULT_SCHOOLBOOK(matrix, vector)
  for row ← 0 to n − k − 1 do
    syndrome[row] = 0                                        ▷ Initialisation
  for row ← 0 to n − k − 1 do
    for col ← 0 to n − 1 do
      syndrome[row] ^= matrix[row][col] & vector[col]        ▷ Multiplication and addition
  return syndrome
```

[9] P.-L. Cayrel et al. “Message-Recovery Laser Fault Injection Attack on the Classic McEliece Cryptosystem”. In: **EUROCRYPT**. 2021.

Laser fault injection attack on the schoolbook matrix-vector multiplication

Targeting the XOR operation, considering the Thumb instruction set.

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

Laser fault injection in Flash memory : **mono-bit, bit-set fault model** [10].

[10] B. Colombier et al. “Laser-induced Single-bit Faults in Flash Memory: Instructions Corruption on a 32-bit Microcontroller”. In: **IEEE HOST**. 2019.

Laser fault injection attack on the schoolbook matrix-vector multiplication

Targeting the XOR operation, considering the Thumb instruction set.

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

Laser fault injection in Flash memory : **mono-bit, bit-set fault model** [10].

ADCS: R1 $=$ R0 $+$ R1	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	1

[10] B. Colombier et al. “Laser-induced Single-bit Faults in Flash Memory: Instructions Corruption on a 32-bit Microcontroller”. In: **IEEE HOST**. 2019.

Laser fault injection attack on the schoolbook matrix-vector multiplication

Targeting the XOR operation, considering the Thumb instruction set.

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

Laser fault injection in Flash memory : **mono-bit, bit-set fault model** [10].

ADCS: R1 = R0 + R1	0	1	0	0	0	0	0	1	0	1	0	0	0	0	0	1

Outcome: switching from $\mathbb{F}_2$ to $\mathbb{N}$

The exclusive-OR (addition over $\mathbb{F}_2$) is turned into an **addition with carry** (addition over $\mathbb{N}$)

[10] B. Colombier et al. “Laser-induced Single-bit Faults in Flash Memory: Instructions Corruption on a 32-bit Microcontroller”. In: **IEEE HOST**. 2019.

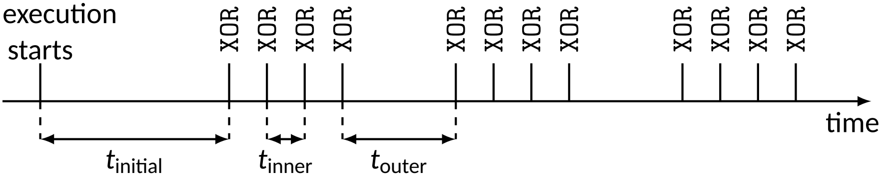
Multiple faults

Three independent delays must be tuned to fault the full matrix-vector multiplication:

t_{initial} : **initial** delay before the multiplication starts

t_{inner} : delay in the **inner** `for` loop

t_{outer} : delay in the **outer** `for` loop



Outcome

After $n.(n-k)$ faults, we get a **faulty syndrome s** $\in \mathbb{N}^{n-k}$

A single bit-set ?

The ADCS instruction was just one bit-set away from the EORS instruction. Did we just get lucky?

[11] https://ww1.microchip.com/downloads/en/devicedoc/31029a.pdf
[12] https://github.com/riscv/riscv-isa-manual/releases/download/Ratified-IMAFDQC/riscv-spec-20191213.pdf
[13] ARMv7-M Architecture Reference Manual https://developer.arm.com/documentation/ddi0403

A single bit-set ?

The `ADCS` instruction was just one bit-set away from the `EORS` instruction. Did we just get lucky?

Answer: No

It happens for other instructions sets too:

PIC `XORWF` → `ADDWF` with one bit-set [11]

RISC-V `C.XOR` → `C.ADDW` with one bit-set [12]

ARMv7 `EORS.W` → `QADD` with six (1-4-1) bit-sets [13]

Other instruction corruptions could be equivalent to addition over $\mathbb{N}$ (shifts, rotations, etc)

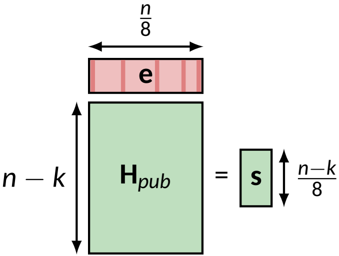
[11] https://ww1.microchip.com/downloads/en/devicedoc/31029a.pdf
[12] https://github.com/riscv/riscv-isa-manual/releases/download/Ratified-IMAFDQC/riscv-spec-20191213.pdf
[13] ARMv7-M Architecture Reference Manual https://developer.arm.com/documentation/ddi0403

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 Packed matrix-vector multiplication

```
function Mat_vec_mult_packed(mat, vector)
  for row ← 0 to ((n − k)/8 − 1) do
    syn[row] = 0                              ▷ Initialisation
  for row ← 0 to (n − k − 1) do
    b = 0
    for col ← 0 to (n/8 − 1) do
      b ^= mat[row][col] & vector[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

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

Algorithm Packed matrix-vector multiplication

function Mat_vec_mult_packed(`mat, vector`)
  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] & vector[col]`
    $b \mathbin{\hat{}}= b >> 4$
    $b \mathbin{\hat{}}= b >> 2$ ▷ Exclusive-OR folding
    $b \mathbin{\hat{}}= b >> 1$
    $b \mathrel{\&}= 1$ ▷ LSB extraction
    `syn[row/8] |=` b `<<` $(row\%8)$ ▷ Packing
  return `syn`

Attack not directly applicable here

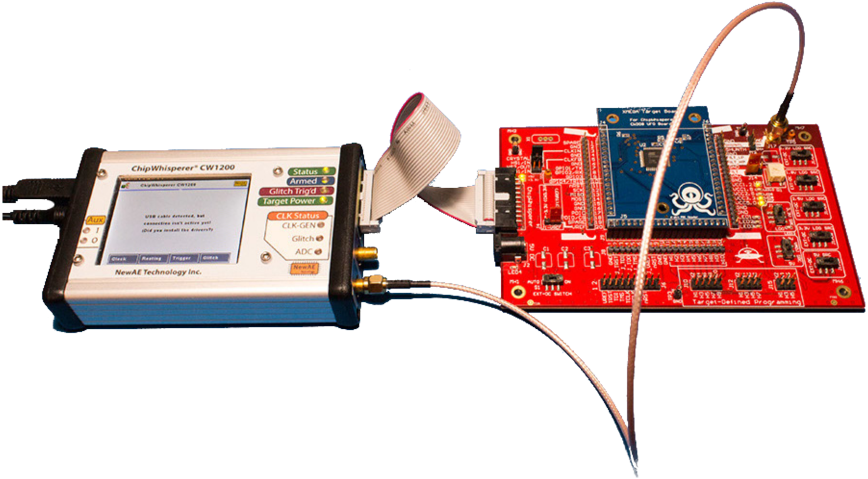
We suggested the following strategy (**admittedly not feasible**):

- Prematurely exit the **inner** `for` loop to keep only one byte
- Reverse the exclusive-OR folding permutation over $\mathbb{F}_2^8$
- Mask with `0xFF` instead of 1
- For bit packing:
  - Turn shift into `CMP`
  - Prematurely exit the **outer** `for` loop to keep only one byte

Physical attack #2: Side-channel analysis

Side-channel analysis setup

ChipWhisperer platform (again) [14]



[14] C. O'Flynn et al. "ChipWhisperer: An Open-Source Platform for Hardware Embedded Security Research". In: **COSADE**. 2014.

Side-channel analysis to obtain the integer syndrome

Algorithm Packed matrix-vector multiplication

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

Side-channel analysis to obtain the integer syndrome

Algorithm Packed matrix-vector multiplication

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

Side-channel analysis to obtain the integer syndrome

Algorithm Packed matrix-vector multiplication

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

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

Side-channel analysis to obtain the integer syndrome

Algorithm Packed matrix-vector multiplication

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

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

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

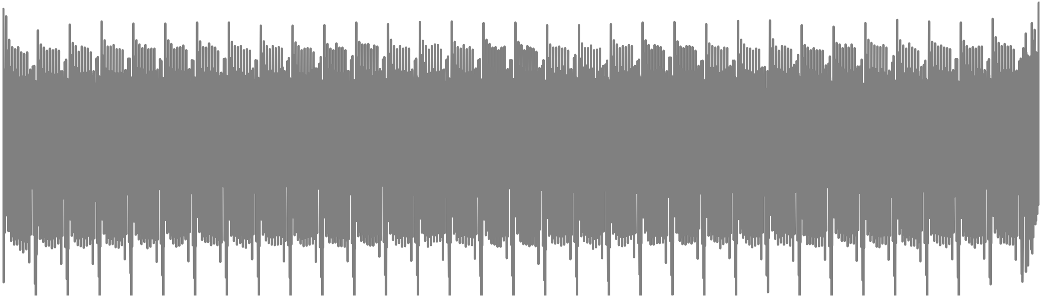
	b	=	00001000	HW=1
HD = 2				
	b	=	00000100	HW=1

Happens if:
$\mathrm{HW}(\texttt{mat[r][c]} \,\&\, \texttt{e\_vec[c]}) > 1$
Unlikely, since $\mathrm{HW}(\mathbf{e}) = t$ is low.

Side-channel analysis for Hamming weight recovery

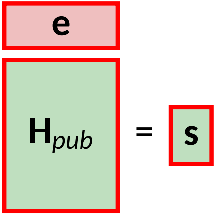
$$\mathbf{s} = \mathbf{H}_{pub}\mathbf{e}$$

$$\begin{array}{c} \boxed{\mathbf{e}} \\ \boxed{\mathbf{H}_{pub}} = \boxed{\mathbf{s}} \end{array}$$



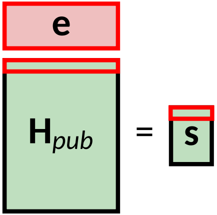
Side-channel analysis for Hamming weight recovery

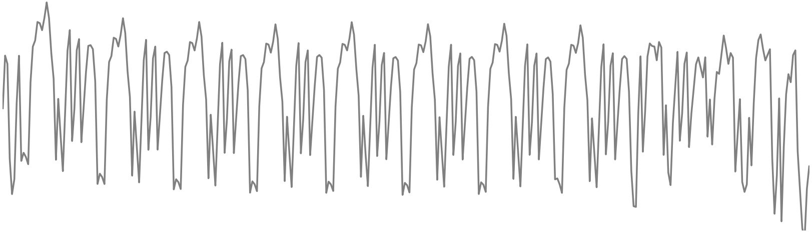
$$\mathbf{s} = \mathbf{H}_{pub}\mathbf{e}$$



Side-channel analysis for Hamming weight recovery

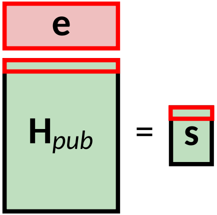
$$\mathbf{s}_j = \mathbf{H}_{pub_{[j,]}}\mathbf{e}$$

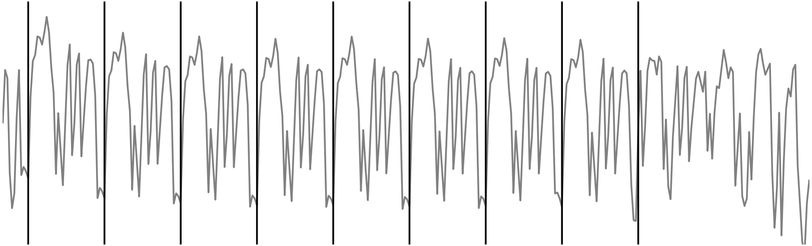




Side-channel analysis for Hamming weight recovery

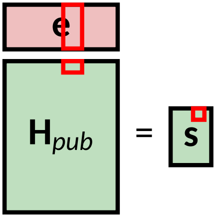
$$\mathbf{s}_j = \mathbf{H}_{pub_{[j,]}}\mathbf{e}$$



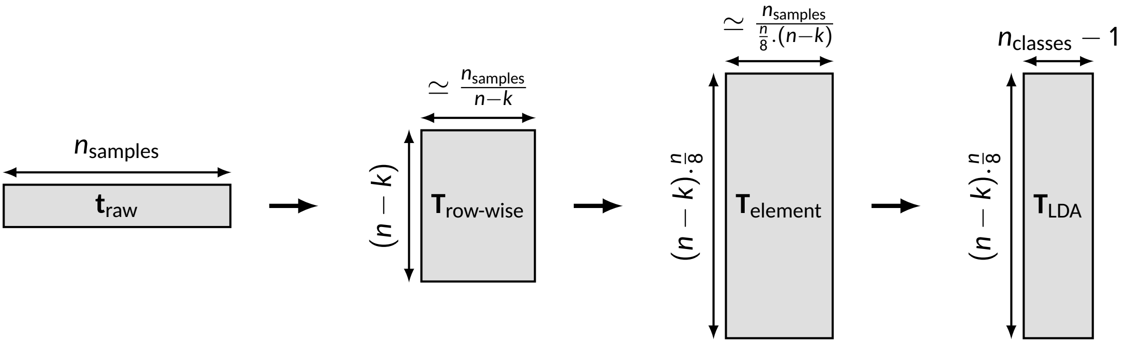


Side-channel analysis for Hamming weight recovery

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



Trace(s) reshaping process



Training phase

- Linear Discriminant Analysis (LDA) for dimensionality reduction,
- From a **single** trace, we get $(n-k) \times \frac{n}{8}$ training samples $n = 8192$ → more than 1.7×10^6
- Fed to a **single** Random Forest classifier (`sklearn.ensemble.RandomForestClassifier`)

Random Forest classifier

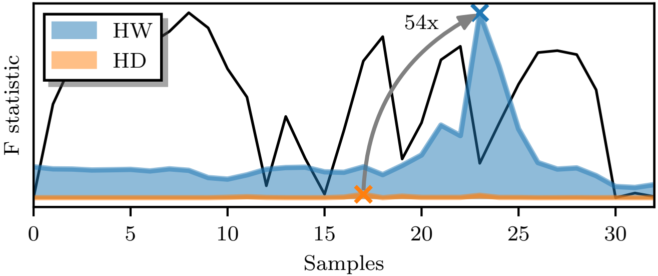
Random Forest classifier training:

- Hamming weight:
  - $> 99.5\,\%$ test accuracy,
- Hamming distance:
  - $\approx 80\,\%$ test accuracy.

Random Forest classifier

Random Forest classifier training:

- Hamming weight:
  - $> 99.5\,\%$ test accuracy,
- Hamming distance:
  - $\approx 80\,\%$ test accuracy.



Outcome

- We can recover the **Hamming weight** very accurately,
- but **not the Hamming distance**...
- We can compute a *slightly innacurate* integer syndrome.

Exploiting the integer syndrome

Exploiting the integer syndrome

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

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

Input: a matrix $\mathbf{H}_{pub} \in \mathcal{M}_{n-k,n}(\mathbb{N})$ 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 HW($\mathbf{x}$) $\leq t$ such that : $\mathbf{H}_{pub}\mathbf{e} = \mathbf{s}$

ILP problem

Let $\mathbf{b} \in \mathbb{N}^n$, $\mathbf{c} \in \mathbb{N}^m$ and $\mathbf{A} \in \mathcal{M}_{m,n}(\mathbb{N})$
We have the following optimization problem:

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

Exploiting the integer syndrome

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

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

Input: a matrix $\mathbf{H}_{pub} \in \mathcal{M}_{n-k,n}(\mathbb{N})$ 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 HW($\mathbf{x}$) $\leq t$ such that : $\mathbf{H}_{pub}\mathbf{e} = \mathbf{s}$

ILP problem

Let $\mathbf{b} \in \mathbb{N}^n$, $\mathbf{c} \in \mathbb{N}^m$ and $\mathbf{A} \in \mathcal{M}_{m,n}(\mathbb{N})$
We have the following optimization problem:

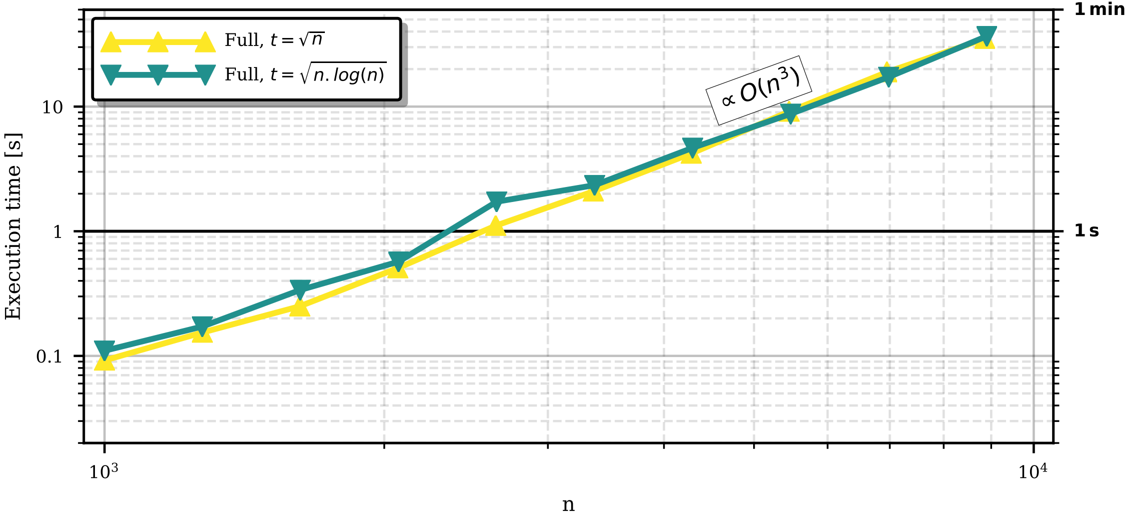
$$\min\{\mathbf{b}^T\mathbf{x} \mid \mathbf{A}\mathbf{x} = \mathbf{c}, \mathbf{x} \in \mathbb{N}^n, \mathbf{x} \geq 0\}$$

Can be solved by **integer linear programming**.

Solver used: `Scipy.optimize.linprog`.

Cannot deal with errors in the recovered syndrome.

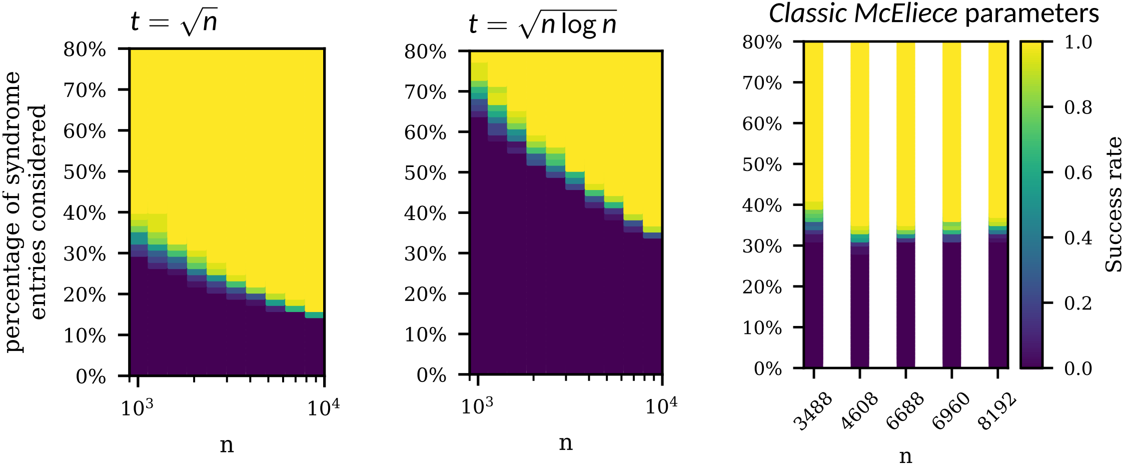
Experimental results



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

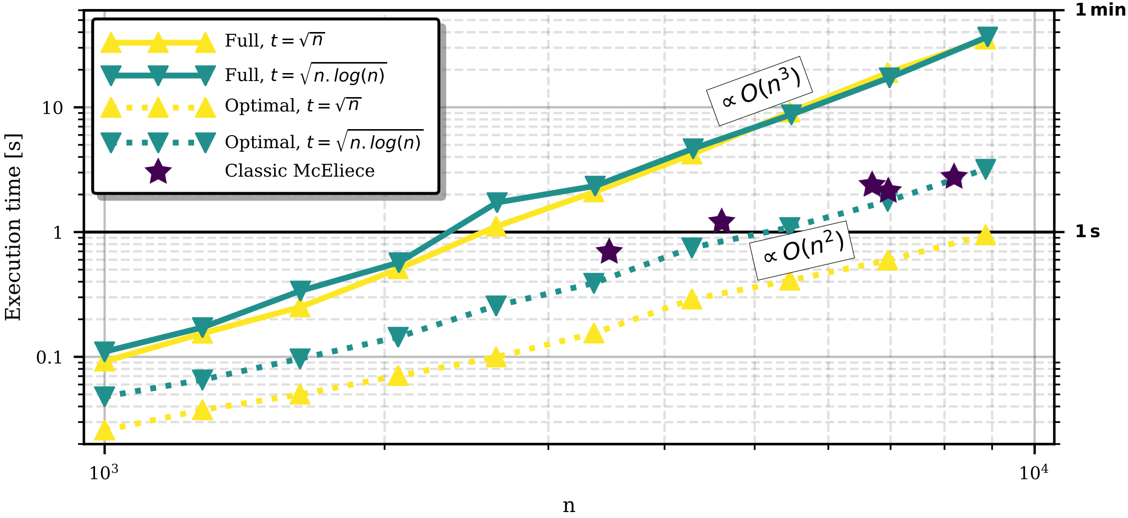
Required fraction of faulty syndrome entries

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



Empirically, when considering the **optimal fraction**, time complexity drops from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2)$.

Exploiting the integer syndrome

Option 2 (*Quantitative Group Testing* [15]): which columns of $\mathbf{H}_{pub}$ "contributed" to the syndrome.

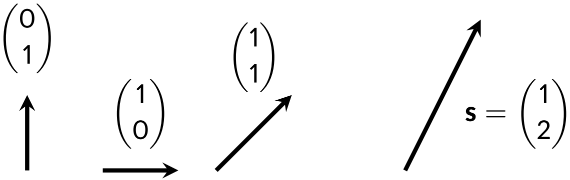
[15] U. Feige et al. "Quantitative Group Testing and the rank of random matrices". In: **CoRR** (2020). arXiv: `2006.09074`.

Exploiting the integer syndrome

Option 2 (*Quantitative Group Testing* [15]): which columns of $\mathbf{H}_{pub}$ "contributed" to the syndrome.

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

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



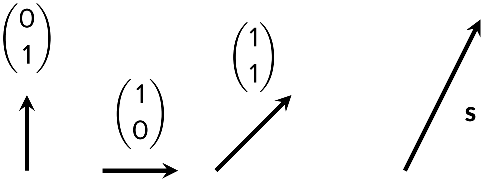
[15] U. Feige et al. "Quantitative Group Testing and the rank of random matrices". In: **CoRR** (2020). arXiv: 2006.09074.

Exploiting the integer syndrome

Option 2 (*Quantitative Group Testing* [15]): which columns of $\mathbf{H}_{pub}$ "contributed" to the syndrome.

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

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



$\mathbf{s} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}$

Score function

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

$$\psi(i) = \mathbf{H}_{pub[,i]} \cdot \mathbf{s} + \bar{\mathbf{H}}_{pub[,i]} \cdot \bar{\mathbf{s}} \qquad \text{with } \bar{\mathbf{H}} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \qquad \text{and } \bar{\mathbf{s}} = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

- $\psi(0) = 1 \times 0 + 2 \times 1 + 1 \times 1 + 0 \times 0 = 3$
- $\psi(1) = 1$
- $\psi(2) = 3$

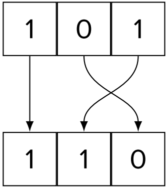
[15] U. Feige et al. "Quantitative Group Testing and the rank of random matrices". In: **CoRR** (2020). arXiv: 2006.09074.

Score function : advantages

The score of the columns of $\mathbf{H}_{pub}$ provides us with a **ranking**.
This defines a **permutation** over **e** too, the **most likely** to bring t ones in the first positions.

Scores : [3, 1, 3]

Permutation : [0, 2, 1]



Bringing t ones in the first $(n-k)$ positions is sufficient.

Information-set decoding methods can then be used to recover the error vector.

Computational complexity

- Computing the dot product of two vectors is **very fast**,
- Overall cost for all columns of $\mathbf{H}_{pub}$: $\mathcal{O}((n-k)\times n) = \mathcal{O}(n^2)$
- $n = 8192$: $\approx 0.2\,\mathrm{s}$

Conclusion

Conclusion

The results of the NIST PQC standardisation process are (almost) known.
With implementations comes the **threat of physical attacks**.
This threat must be considered and properly evaluated.

Considered approach: use known cryptanalysis tools **“augmented”** with additional information.

- Additional information realistically obtained by physical attacks:
  - Fault injection attacks,
  - Side-channel attacks.
- Integer syndrome decoding problem,
  - Challenge: recover the **integer syndrome** as **accurately** as possible.
- Information-set decoding methods starting with a plausible permutation.

Future works

Future works:

- Improve the **recovery** of the integer syndrome,
- Improve the efficiency of the **message-recovery step**,
- Try to apply similar ideas to attack the **long-term secret key**,
- Apply the idea to **other problems** (and NIST PQC candidates).

Future works

Future works:

- Improve the **recovery** of the integer syndrome,
- Improve the efficiency of the **message-recovery step**,
- Try to apply similar ideas to attack the **long-term secret key**,
- Apply the idea to **other problems** (and NIST PQC candidates).

— Questions ? —