

Laser fault injection attack on the *Classic McEliece* cryptosystem

Brice Colombier

brice.colombier@grenoble-inp.fr

November 10, 2021

Algorithmes de chiffrement post-quantiques et sécurité matérielle



Acknowledgement

Joint work with :

Pierre-Louis Cayrel : Laboratoire Hubert Curien, Saint-Étienne,

Vlad-Florin Drăgoi : University of Arad, Romania,

Émilie Chanavat : Laboratoire Hubert Curien, Saint-Étienne,

Paul Grandamme : Laboratoire Hubert Curien, Saint-Étienne,

Alexandre Menu : École des Mines de Saint-Étienne, Gardanne,

Julien Vernay : Laboratoire Hubert Curien, Saint-Étienne,

Lilian Bossuet : Laboratoire Hubert Curien, Saint-Étienne,

Work carried out in the framework of the FUIAAP22 Project PILAS



Most public key cryptosystems rely on the hardness of **number theoretic** problems:

- prime factorization,
- discrete logarithm.

Peter Shor showed that quantum algorithms can solve these problems in **polynomial time** [1].

In 2016, NIST initiated a process for cryptography standards that are **quantum resistant** [2].

One of the four finalists of Round 3 in the *Key Encapsulation Mechanism* category (announced July 22, 2020) is *Classic McEliece* [3], based on **error-correcting codes**.

[1] P. W. Shor. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer". In: *SIAM Journal on Computing* (1997).

[2] <https://csrc.nist.gov/Projects/post-quantum-cryptography/>

[3] M. R. Albrecht, D. J. Bernstein, T. Chou, C. Cid, J. Gilcher, T. Lange, V. Maram, I. von Maurich, R. Misoczki, R. Niederhagen, K. G. Paterson, E. Persichetti, C. Peters, P. Schwabe, N. Sendrier, J. Szefer, C. J. Tjhai, M. Tomlinson, and W. Wang. *Classic McEliece*. Tech. rep. National Institute of Standards and Technology, 2020.

Code-based cryptography

Niederreiter cryptosystem

Classic McEliece is based on the Niederreiter cryptosystem [4]:

➤ $\text{KeyGen}(n, k, t) = (\text{pk}, \text{sk})$

\mathbf{H} : parity-check matrix of \mathcal{C} [5]

\mathbf{S} : random invertible matrix of size $n - k$

\mathbf{P} : random permutation matrix of size n

Compute $\mathbf{H}_{\text{pub}} = \mathbf{SHP}$

$\text{pk} = (\mathbf{H}_{\text{pub}}, t)$ /* public key */

$\text{sk} = (\mathbf{S}, \mathbf{H}, \mathbf{P})$ /* secret key */

➤ $\text{Encrypt}(\mathbf{m}, \text{pk}) = \mathbf{s}$

Encode \mathbf{m} into a constant-weight vector \mathbf{e} of Hamming weight t

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

[4] H. Niederreiter. "Knapsack-type cryptosystems and algebraic coding theory". In: *Problems of Control and Information Theory* (1986).

[5] \mathcal{C} is an $[n, k]$ linear code that admits an efficient decoding algorithm that can correct up to t errors.

Niederreiter cryptosystem

Classic McEliece is based on the Niederreiter cryptosystem [4]:

➤ $\text{KeyGen}(n, k, t) = (\text{pk}, \text{sk})$

\mathbf{H} : parity-check matrix of \mathcal{C} [5]

\mathbf{S} : random invertible matrix of size $n - k$

\mathbf{P} : random permutation matrix of size n

Compute $\mathbf{H}_{\text{pub}} = \mathbf{SHP}$

$\text{pk} = (\mathbf{H}_{\text{pub}}, t)$ /* public key */

$\text{sk} = (\mathbf{S}, \mathbf{H}, \mathbf{P})$ /* secret key */

➤ $\text{Encrypt}(\mathbf{m}, \text{pk}) = \mathbf{s}$

Encode \mathbf{m} into a constant-weight vector \mathbf{e} of Hamming weight t

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

[4] H. Niederreiter. "Knapsack-type cryptosystems and algebraic coding theory". In: *Problems of Control and Information Theory* (1986).

[5] \mathcal{C} is an $[n, k]$ linear code that admits an efficient decoding algorithm that can correct up to t errors.

The security of the Niederreiter cryptosystem is based on the syndrome decoding problem.

Syndrome decoding problem

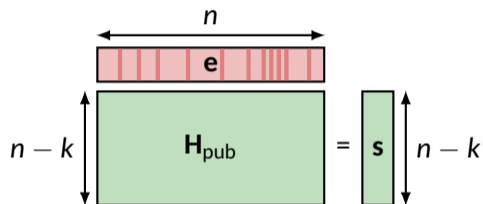
Input: a binary matrix $\mathbf{H} \in \mathcal{M}_{n-k,n}(\mathbb{F}_2)$
a binary vector $\mathbf{s} \in \mathbb{F}_2^{n-k}$
a scalar $t \in \mathbb{N}^+$

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

Known to be an **NP-hard** problem [6].

[6] E. R. Berlekamp, R. J. McEliece, and H. C. A. van Tilborg. "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 (H_{pub}, t) is very large!

Hardware implementations

Implementations on embedded systems are possible : [7] [8] [9]

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 these implementations vulnerable to **physical attacks**

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

[8] J. Roth, E. G. Karatsiolis, and J. Krämer. “Classic McEliece Implementation with Low Memory Footprint”. In: *CARDIS*. 2020.

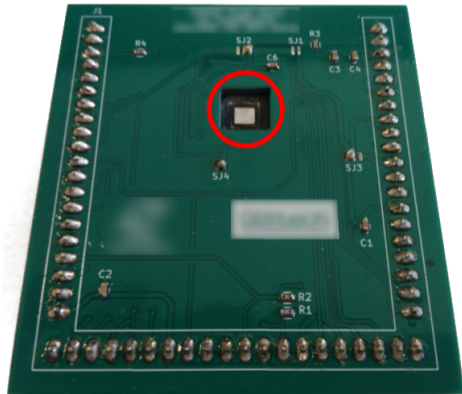
[9] M.-S. Chen and T. Chou. “Classic McEliece on the ARM Cortex-M4”. In: *IACR TCHES (2021)*.

Attacker model

Laser fault injection attacks

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

- ChipWhisperer platform [10],
- Custom board with an opening,
- Decapsulated chip
 - access to the backside of the die



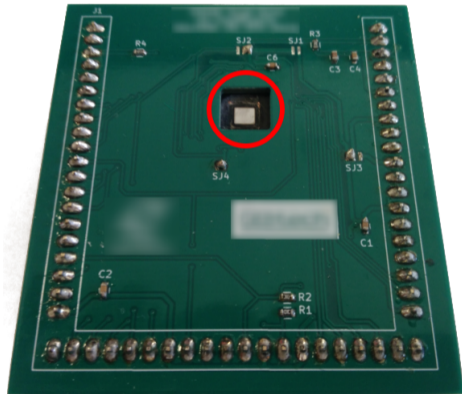
[10] C. O'Flynn and Z. Chen. "ChipWhisperer: An Open-Source Platform for Hardware Embedded Security Research". In: COSADE. 2014

Laser fault injection attacks

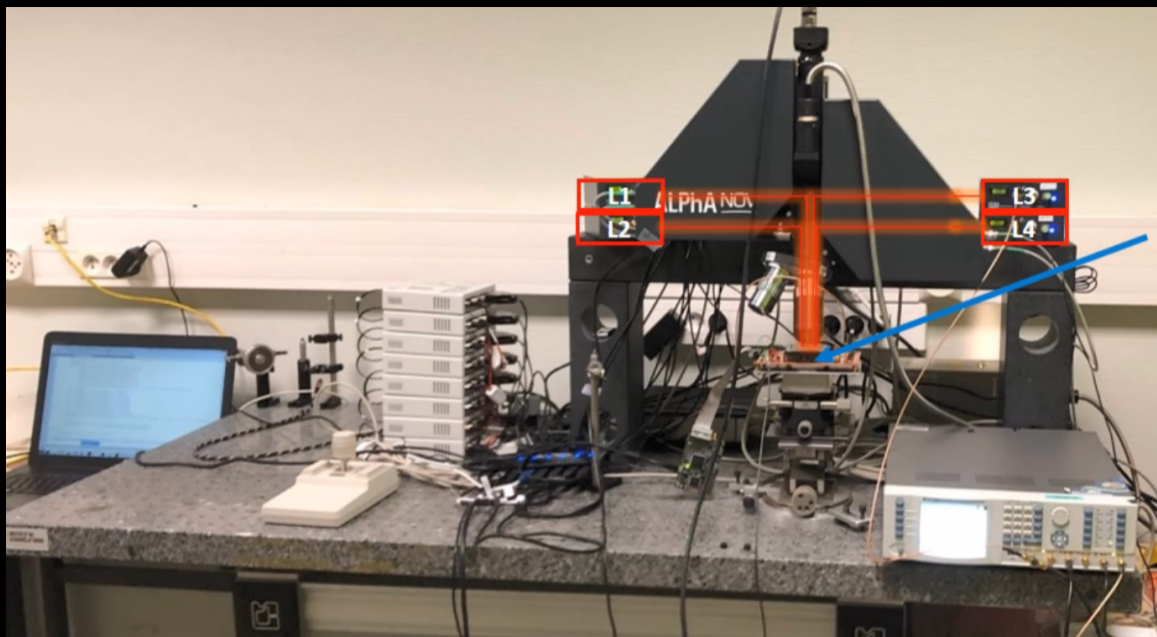
Physical attack : an attacker has a **physical access** to the device.

- ChipWhisperer platform [10],
- Custom board with an opening,
- Decapsulated chip
 - access to the backside of the die

Many thanks to Jean-Max Dutertre!



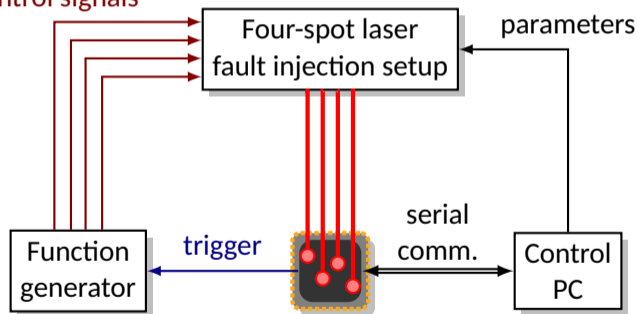
[10] C. O'Flynn and Z. Chen. "ChipWhisperer: An Open-Source Platform for Hardware Embedded Security Research". In: COSADE. 2014



Laser fault injection setup

Setup presented tomorrow at CARDIS [11]

individual laser
control signals

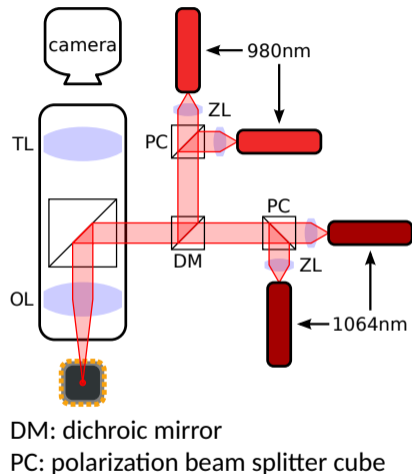
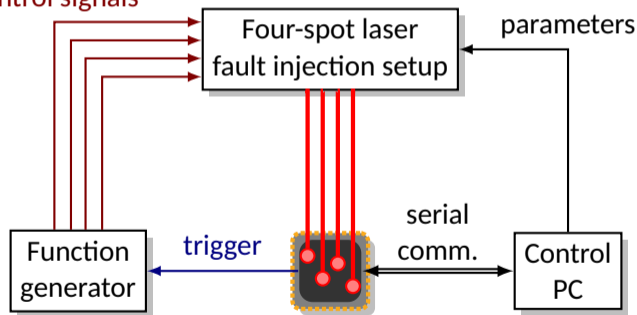


[11] B. Colombier, J. Vernay, P. Grandamme, É. Chavanat, L. Bossuet, L. de Laulanié, and B. Chassagne. “Multi-spot Laser Fault Injection Setup: New Possibilities for Fault Injection Attacks”. In: *CARDIS*. 2021

Laser fault injection setup

Setup presented tomorrow at CARDIS [11]

individual laser
control signals



[11] B. Colombier, J. Vernay, P. Grandamme, É. Chavanat, L. Bossuet, L. de Laulanié, and B. Chassagne. "Multi-spot Laser Fault Injection Setup: New Possibilities for Fault Injection Attacks". In: *CARDIS*. 2021

Recent line of work on laser fault injection in Flash memory [12] [13] [14] [15].

Fault model

Fully reproducible single-bit bit-set on data (and instructions) fetched from Flash memory

[12] D. S. V. Kumar, A. Beckers, J. Balasch, B. Gierlichs, and I. Verbauwhede. “An In-Depth and Black-Box Characterization of the Effects of Laser Pulses on ATmega328P”. In: *CARDIS*. 2018.

[13] B. Colombier, A. Menu, J.-M. Dutertre, P.-A. Moëllic, J.-B. Rigaud, and J.-L. Danger. “Laser-induced Single-bit Faults in Flash Memory: Instructions Corruption on a 32-bit Microcontroller”. In: *HOST*. 2019.

[14] A. Menu, J.-M. Dutertre, J.-B. Rigaud, B. Colombier, P.-A. Moëllic, and J.-L. Danger. “Single-bit Laser Fault Model in NOR Flash Memories: Analysis and Exploitation”. In: *FDTC*. 2020.

[15] K. Garb and J. Obermaier. “Temporary Laser Fault Injection into Flash Memory: Calibration, Enhanced Attacks, and Countermeasures”. In: *IOLTS*. 2020.

VIDEO TIME

also available at:

<https://www.youtube.com/watch?v=QY2N2B1fR3Q>

Proposed attack

Syndrome decoding problem

Binary syndrome decoding problem (Binary SDP)

Input: a binary matrix $\mathbf{H} \in \mathcal{M}_{n-k,n}(\mathbb{F}_2)$

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 $\text{HW}(\mathbf{x}) \leq t$ such that : $\mathbf{H}\mathbf{x} = \mathbf{s}$

Syndrome decoding problem

Binary syndrome decoding problem (Binary SDP)

Input: a binary matrix $\mathbf{H} \in \mathcal{M}_{n-k,n}(\mathbb{F}_2)$
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 $\text{HW}(\mathbf{x}) \leq t$ such that : $\mathbf{H}\mathbf{x} = \mathbf{s}$

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

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

Output: a vector \mathbf{x} 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 : $\mathbf{H}\mathbf{x} = \mathbf{s}$

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

Input: a matrix $\mathbf{H} \in \mathcal{M}_{n-k,n}(\mathbb{N})$ 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{x} 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 : $\mathbf{H}\mathbf{x} = \mathbf{s}$

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 aim at solving the following optimization problem:

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

\mathbb{N} -SDP and integer linear programming

- The \mathbb{N} syndrome decoding and the integer linear programming problems are **equivalent**,
- Integer linear programming solvers are **very efficient**,
- The \mathbb{N} syndrome decoding problem can be solved **very efficiently**.

\mathbb{N} -SDP and integer linear programming

- The \mathbb{N} syndrome decoding and the integer linear programming problems are **equivalent**,
- Integer linear programming solvers are **very efficient**,
- The \mathbb{N} syndrome decoding problem can be solved **very efficiently**.

\mathbb{N} -SDP framework

To be in the \mathbb{N} -SDP framework, we must obtain a **faulty syndrome**.

Instead of $\mathbf{s} \in \mathbb{F}_2^{n-k}$, we need $\mathbf{s} \in \mathbb{N}^{n-k}$.

Target: syndrome computation

We target the syndrome computation: $\mathbf{s} = \mathbf{H}_{\text{pub}} \mathbf{e}$

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

Algorithm 1 Schoolbook matrix-vector multiplication

```
1: function MAT_VEC_MULT_SCHOOLBOOK(matrix, error_vector)
2:   for r ← 0 to n - k - 1 do
3:     syndrome[r] = 0 ▷ Initialisation
4:   for r ← 0 to n - k - 1 do
5:     for c ← 0 to n - 1 do
6:       syndrome[r] ^= matrix[r][c] & error_vector[c] ▷ Multiplication and addition
7:   return syndrome
```

Exclusive-OR operation

We consider 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

Exclusive-OR operation

We consider 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



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

Outcome

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

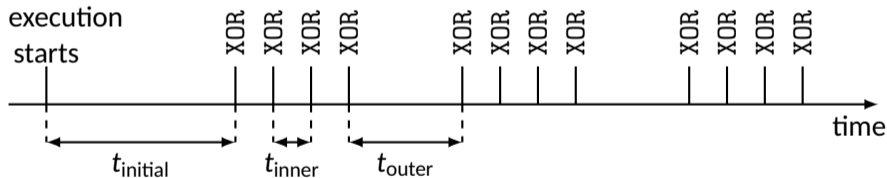
Multiple faults

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

Having the faulty syndrome $\mathbf{s} \in \mathbb{N}^{n-k}$ and the public key \mathbf{H}_{pub} we solve:

$$\min\{\mathbf{b}^T \mathbf{e} \mid \mathbf{H}_{\text{pub}} \mathbf{e} = \mathbf{s}, \mathbf{e} \in \mathbb{N}^n, \mathbf{e} \geq 0\}. \quad (1)$$

to recover the error-vector \mathbf{e} of Hamming weight t .

We used `scipy.optimize.linprog` from the Scipy Python package [16].

[16] <https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.linprog.html>

Discussion about attack feasibility

Observation 1

Observation 1:

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

[17] <https://ww1.microchip.com/downloads/en/devicedoc/31029a.pdf>

[18] <https://github.com/riscv/riscv-isa-manual/releases/download/Ratified-IMAFDQC/riscv-spec-20191213.pdf>

[19] ARMv7-M Architecture Reference Manual <https://developer.arm.com/documentation/ddi0403>

Observation 1

Observation 1:

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 [17]

RISC-V C.XOR → C.ADDW with one bit-set [18]

ARMv7 EORS.W → QADD with six (1-4-1) bit-sets [19]

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

[17] <https://ww1.microchip.com/downloads/en/devicedoc/31029a.pdf>

[18] <https://github.com/riscv/riscv-isa-manual/releases/download/Ratified-IMAFDQC/riscv-spec-20191213.pdf>

[19] ARMv7-M Architecture Reference Manual <https://developer.arm.com/documentation/ddi0403>

Observation 2

Observation 2:

The schoolbook matrix-vector multiplication algorithm is highly inefficient!

Algorithm 2 Schoolbook matrix-vector multiplication

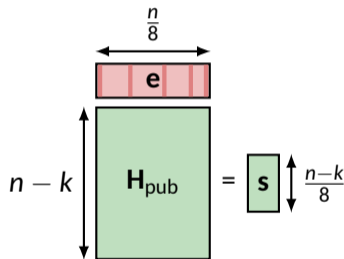
```
1: function MAT_VEC_MULT_SCHOOLBOOK(matrix, error_vector)
2:   for  $r \leftarrow 0$  to  $n - k - 1$  do
3:     syndrome[r] = 0 ▷ Initialisation
4:   for  $r \leftarrow 0$  to  $n - k - 1$  do
5:     for  $c \leftarrow 0$  to  $n - 1$  do
6:       syndrome[r] ^= matrix[r][c] & error_vector[c] ▷ Multiplication and addition
7:   return syndrome
```

Bits are stored **independently**: a lot of memory is wasted.

Packed matrix-vector multiplication

Algorithm 3 Packed matrix-vector multiplication

```
1: function Mat_vec_mult_packed(mat, error_vector)
2:   for r ← 0 to ((n - k)/8 - 1) do
3:     syn[r] = 0                                ▷ Initialisation
4:   for r ← 0 to (n - k - 1) do
5:     b = 0
6:     for c ← 0 to (n/8 - 1) do
7:       b ^= mat[r][c] & error_vector[c]
8:       b ^= b >> 4
9:       b ^= b >> 2                                ▷ Exclusive-OR folding
10:      b ^= b >> 1
11:      b &= 1                                       ▷ LSB extraction
12:      syn[r/8] |= b << (r%8)                       ▷ Bit packing
13:   return syn
```



Packed matrix-vector multiplication

Algorithm 4 Packed matrix-vector multiplication

```
1: function Mat_vec_mult_packed(mat, error_vector)
2:   for r ← 0 to ((n - k)/8 - 1) do
3:     syn[r] = 0                                ▷ Initialisation
4:   for r ← 0 to (n - k - 1) do
5:     b = 0
6:     for c ← 0 to (n/8 - 1) do
7:       b ^= mat[r][c] & error_vector[c]
8:       b ^= b >> 4
9:       b ^= b >> 2                                ▷ Exclusive-OR folding
10:      b ^= b >> 1
11:      b &= 1                                       ▷ LSB extraction
12:      syn[r/8] |= b << (r%8)                       ▷ Bit packing
13: return syn
```

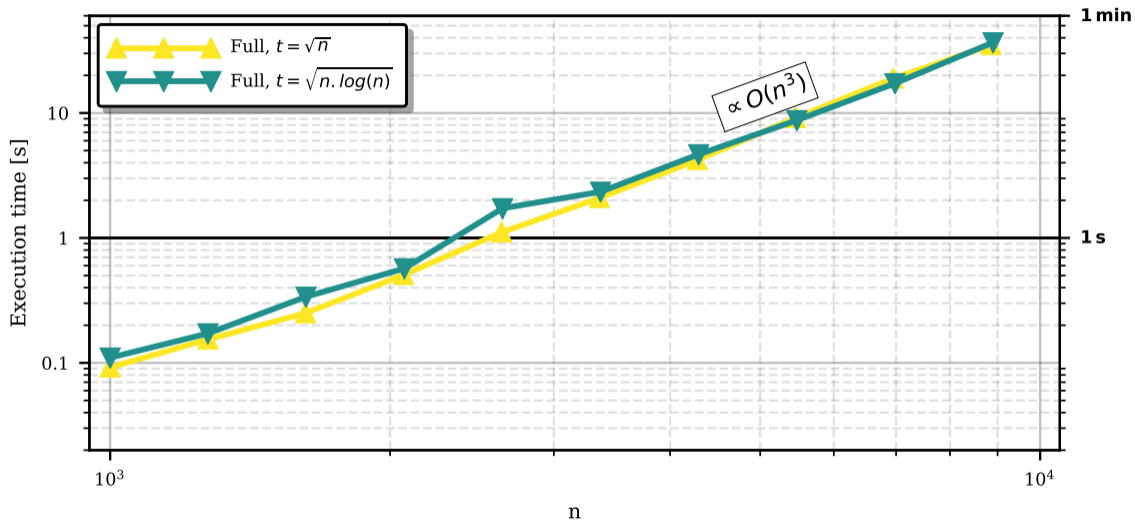
Attack **not directly applicable** here.

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

Experimental results

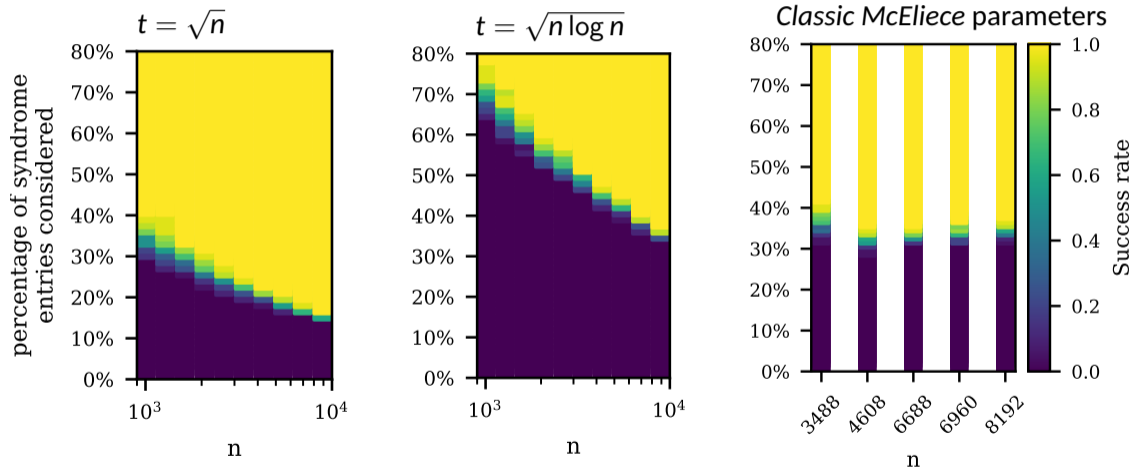
Experimental results



For Classic McEliece : $3488 < n < 8192$

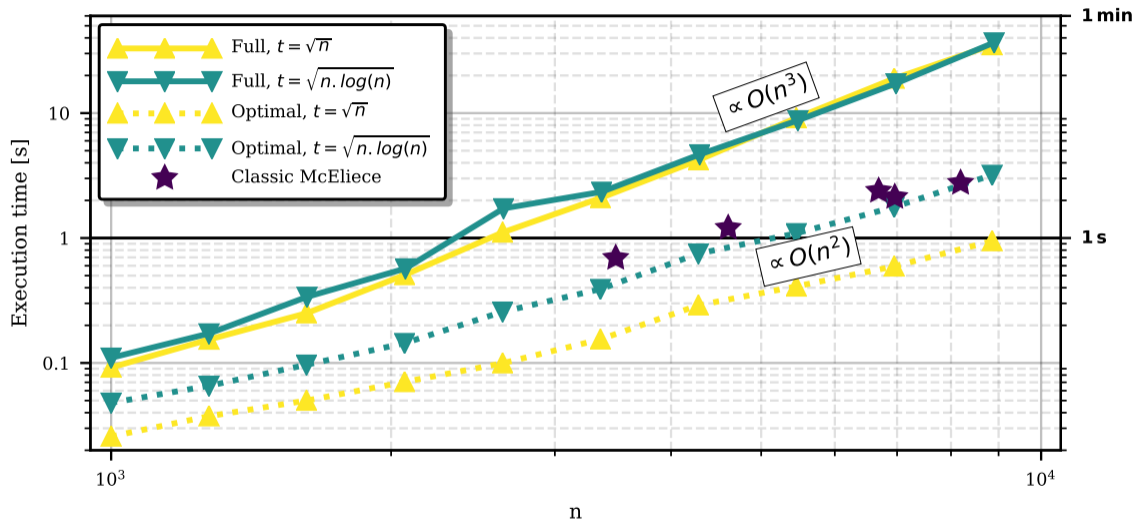
Required fraction of faulty syndrome entries

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



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

Conclusion and future work

New laser fault injection attack on the *Classic McEliece* cryptosystem [20]

Attack steps:

1. Laser fault injection: **instruction corruption** turning EORS into ADCS
2. Get a **faulty syndrome** in \mathbb{N} instead of \mathbb{F}_2
3. **Equivalence** between the \mathbb{N} -SDP and integer linear programming problem
4. Use of **efficient ILP solvers** ($\mathcal{O}(n^2)$ empirical time complexity) with only a **fraction** of faulty syndrome entries

[20] P.-L. Cayrel, B. Colombier, V.-F. Dragoi, A. Menu, and L. Bossuet. “Message-Recovery Laser Fault Injection Attack on the Classic McEliece Cryptosystem”. In: *EUROCRYPT*. 2021.

Future work

On this attack:

- Improve the **practicality** of the attack (less faults)
- Attack **state-of-the-art** hardware **implementations** and extend to **FPGA**,
- Study the **complexity drop** from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2)$ when considering the optimal fraction of faulty syndrom entries,
- Develop **countermeasures**.

Extending the idea:

- Target other operations to recover the **key** instead,
- Apply to **other cryptosystems**.

On this attack:

- Improve the **practicality** of the attack (less faults)
- Attack **state-of-the-art** hardware **implementations** and extend to **FPGA**,
- Study the **complexity drop** from $\mathcal{O}(n^3)$ to $\mathcal{O}(n^2)$ when considering the optimal fraction of faulty syndrom entries,
- Develop **countermeasures**.

Extending the idea:

- Target other operations to recover the **key** instead,
- Apply to **other cryptosystems**.

— Questions? —