Physical Security of the Classic McEliece KEM

Brice Colombier

b.colombier@univ-st-etienne.fr

https://bcolombier.fr

Séminaire SemSecuElec

20/03/2026

Laboratoire Hubert Curien

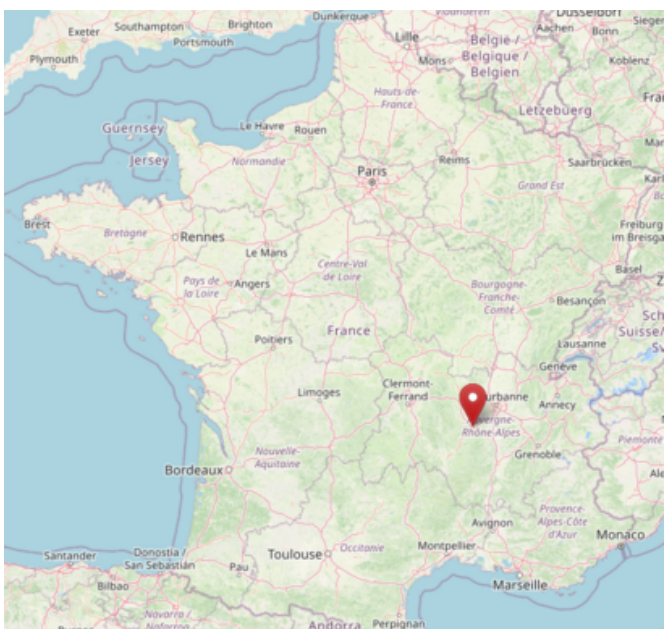
UMR ▪ CNRS ▪ 5516 ▪ Saint-Étienne

Who am I?

Associate professor at Université Jean Monnet in Saint-Étienne.
Member of the SESAM team in Laboratoire Hubert Curien.

Joint work with:

- Pierre-Louis Cayrel (SESAM team, LabHC)
- Vlad-Florin Drăgoi (Univ. Arad, Romania)
- Vincent Grosso (SESAM team, LabHC)
- Alexandre Menu (EMSE Gardanne)
- Lilian Bossuet (SESAM team, LabHC)
- Nicolas Vallet (SESAM team, LabHC)



Introduction and context

The need for post-quantum cryptography

Today's public-key cryptography relies on hard problems from Number Theory.

- integer factorization
- discrete logarithm

⚠️ Shor's quantum algorithm [1] can solve them in **polynomial** time.
(*given a quantum computer of sufficient capacity*...)

Let's not wait until it exists and start thinking about **alternatives**.
→ prevent **store-now-decrypt-later** attacks.

---

[1] Peter W. Shor "*Polynomial time algorithms for discrete logarithms and factoring on a quantum computer*", ANTS (1994)

The NIST PQC standardization process

Objective:
“secure electronic information against the future threat of quantum computers”

Timeline:

- `2016` Call for Proposals
- `2017` Round 1 (69)
- `2019` Round 2 (26)
- `2020` Round 3 (7 (+8 alternate))
- `2022` Four are standardized

CRYSTALS-Kyber CRYSTALS-Dilithium FALCON SPHINCS+

- `2022` Round 4 algorithms announced (4)
- `2025` HQC is standardized too

Classic McEliece

Classic McEliece

Classic McEliece [2] is a **Key Encapsulation Mechanism**

Used to **share a secret** and **derive a session key** from it.

- KeyGen() $\rightarrow (\boldsymbol{H}_{\text{pub}}, \boldsymbol{k}_{\text{priv}})$
- Encap($\boldsymbol{H}_{\text{pub}}$) $\rightarrow (\boldsymbol{s}, \boldsymbol{k}_{\text{session}})$
- Decap($\boldsymbol{s}, \boldsymbol{k}_{\text{priv}}$) $\rightarrow (\boldsymbol{k}_{\text{session}})$

Security parameters

Field $\mathbb{F}_{2^m}$	$\mathbb{F}_{2^{12}}$	$\mathbb{F}_{2^{13}}$	$\mathbb{F}_{2^{13}}$	$\mathbb{F}_{2^{13}}$
n	3488	4608	6688	8192
t	64	96	128	128
$\sim$ security level	>128	>128	>192	>256

[2] Martin R. Albrecht, Daniel J. Bernstein, Tung Chou, et al. "*Classic McEliece: conservative code-based cryptography*", (2022)

Classic McEliece: KeyGen

The KeyGen procedure generates the public and private keys:

- $\boldsymbol{H}_{\text{pub}}$
  - a **large** “public” binary parity-check matrix of a **private** Goppa code

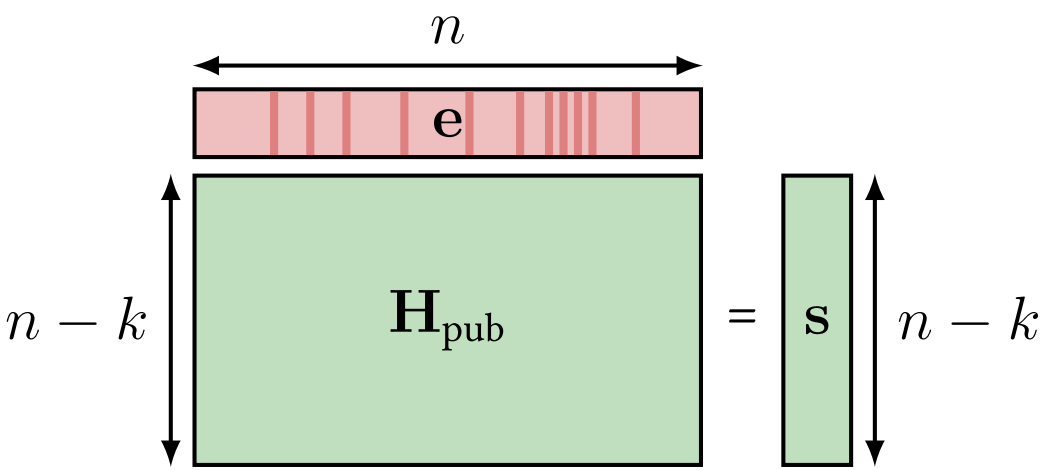
- $\boldsymbol{k}_{\text{priv}} = (\boldsymbol{g}, \mathcal{L})$ the **private** Goppa code
  - $\boldsymbol{g}$ a monic polynomial
    - with coefficients in $\mathbb{F}_{2^m}$
    - of degree t
  - $\mathcal{L}$ a random subset $\{\alpha_0, \alpha_1, ..., \alpha_{n-1}\}$ of $\mathbb{F}_{2^m}$

Classic McEliece: Encapsulation

Encap encapsulates a **secret vector** e from which k_{session} **is derived**

Encap(H_{pub}) $\rightarrow$ (s, k_{session})

1. Sample $e \xleftarrow{\$} \mathbb{F}_2^n$ with HW(e) $= t$
2. $s = H_{\text{pub}} e^{\mathsf{T}}$
3. $k_{\text{session}} = \text{hash}(1, e, s)$



Classic McEliece: Decapsulation

Decap decapsulates the **secret vector** e from which k_{session} **is derived**

Decap(s, k_{priv}) $\rightarrow$ (k_{session})

1. Compute a “private” parity-check matrix $H_{\text{priv}_{g^2}} =$

$$\begin{pmatrix} g^{-2}(\alpha_0) & g^{-2}(\alpha_1) & \dots & g^{-2}(\alpha_{n-1}) \\ \alpha_0 g^{-2}(\alpha_0) & \alpha_1 g^{-2}(\alpha_1) & \dots & \alpha_{n-1} g^{-2}(\alpha_{n-1}) \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_0^{2t-1} g^{-2}(\alpha_0) & \alpha_1^{2t-1} g^{-2}(\alpha_1) & \dots & \alpha_{n-1}^{2t-1} g^{-2}(\alpha_{n-1}) \end{pmatrix}$$

2. Recover the error vector e after multiplying s and $H_{\text{priv}_{g^2}}$
3. $k_{\text{session}} = \text{hash}(1, e, s)$

Classic McEliece: the hard problem

Syndrome decoding problem

Inputs $\boldsymbol{H} \in \mathbb{F}_2^{(n-k)\times n}$ $\boldsymbol{s} \in \mathbb{F}_2^{n-k}$ $t \in \mathbb{N}^+$

Output $\boldsymbol{x} \in \mathbb{F}_2^n$ such that $\boldsymbol{H}\boldsymbol{x}^\mathsf{T} = \boldsymbol{s}$ with HW($\boldsymbol{x}$) $< t$

Known to be $\mathcal{NP}$-complete [3]

Essentially the Niederreiter cryptosystem [4]

---

[3] Elwyn R. Berlekamp, Robert J. McEliece, Henk C. A. van Tilborg "*On the inherent intractability of certain coding problems (Corresp.)*", IEEE Transactions on Information Theory (1978)

[4] Harald Niederreiter "*Knapsack-Type Cryptosystems and Algebraic Coding Theory*", Problems of Control and Information Theory (1986)

Agenda

1. A **laser fault injection** attack on **Encap** [5] $\rightarrow$ **recovers** k_{session}
2. A **profiled side-channel** attack on Encap [6] $\rightarrow$ recovers k_{session}
3. An **unprofiled** side-channel attack on Encap [7] $\rightarrow$ recovers k_{session}
4. A profiled side-channel attack on **Decap** [8] $\rightarrow$ **recovers** k_{priv}

---

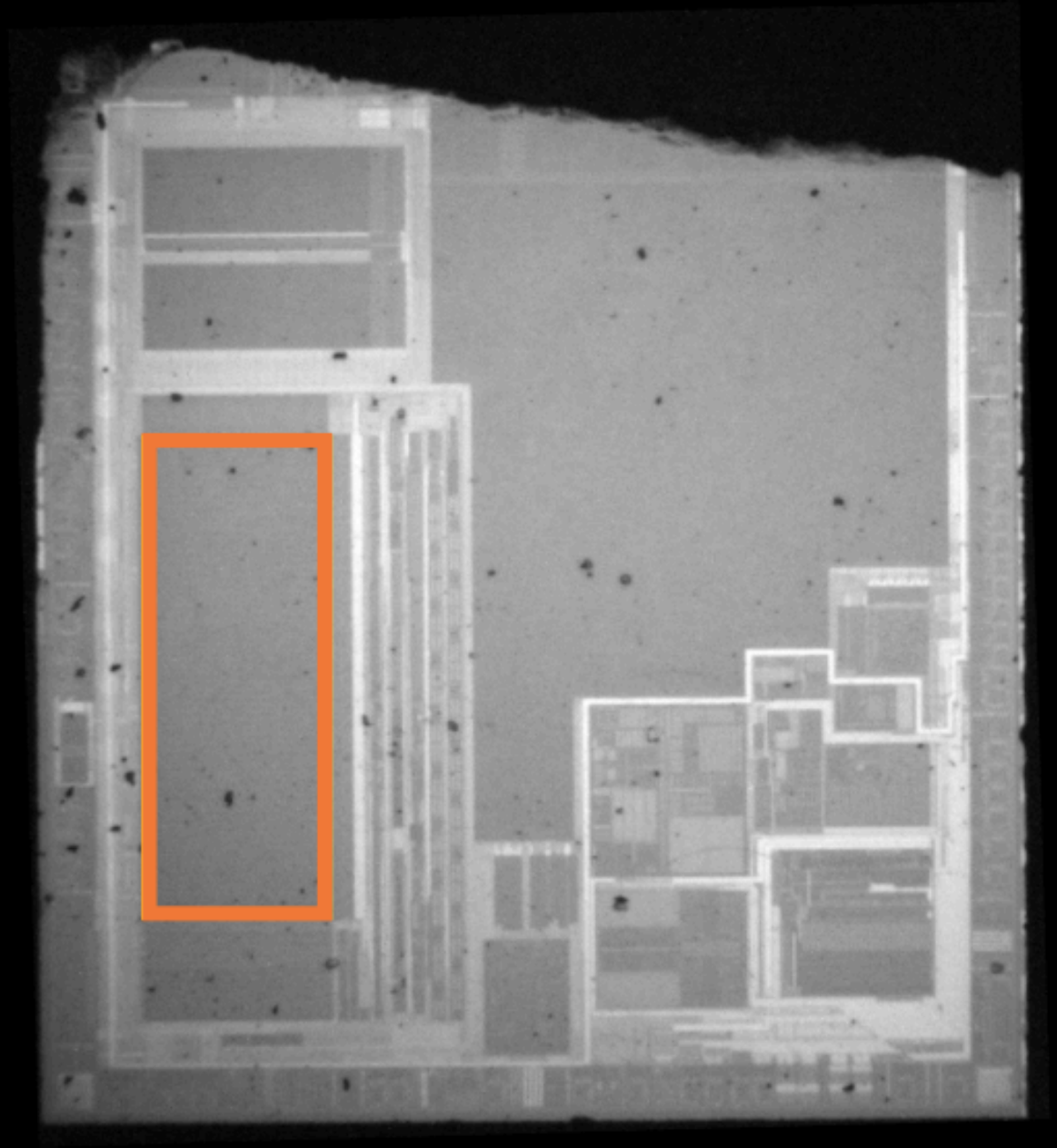
[5] Pierre-Louis Cayrel, Brice Colombier, Vlad-Florin Drăgoi, et al. "*Message-Recovery Laser Fault Injection Attack on the Classic McEliece Cryptosystem*", EUROCRYPT (2021)

[6] Brice Colombier, Vlad-Florin Drăgoi, Pierre-Louis Cayrel, Vincent Grosso "*Profiled Side-Channel Attack on Cryptosystems Based on the Binary Syndrome Decoding Problem*", IEEE TIFS (2022)

[7] Brice Colombier, Vincent Grosso, Pierre-Louis Cayrel, Vlad-Florin Drăgoi "*Message-recovery Horizontal Correlation Attack on Classic McEliece*", CASCADE (2025)

[8] Vlad-Florin Drăgoi, Brice Colombier, Nicolas Vallet, et al. "*Full Key-Recovery Cubic-Time Template Attack on Classic McEliece Decapsulation*", IACR TCHES (2025)

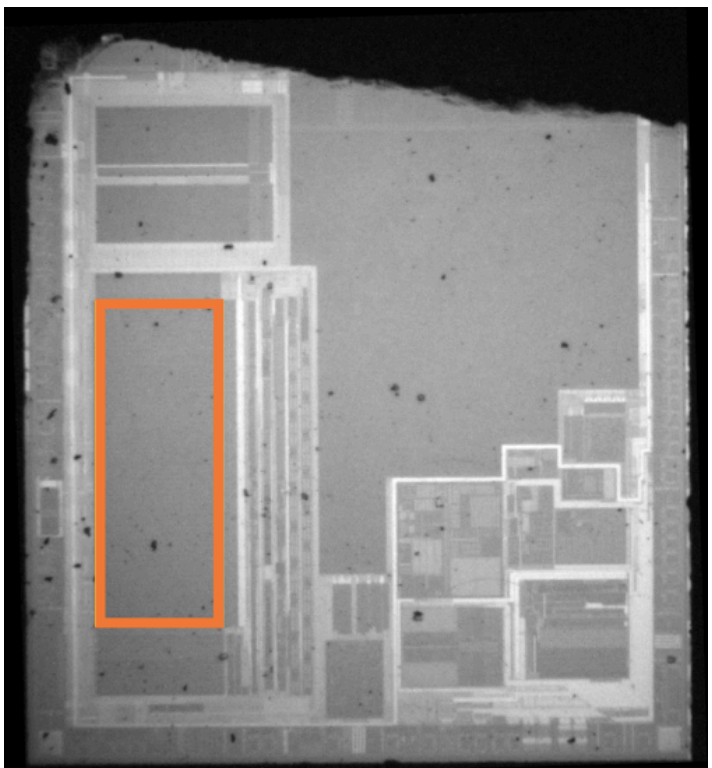
Laser fault injection attack on the Encapsulation step



Laser fault injection in flash memory

Well-understood **fault model** [9]

- Physical level
  - **Photoelectric effect** at near-infrared λ
- Binary level
  - **Bit-set** ($0 \rightarrow 1$) on **fetched** data
  - Down to **single-bit**
- Instruction level
  - Registers
  - Immediate
  - Opcode



[9] Brice Colombier, Alexandre Menu, Jean-Max Dutertre, et al. "*Laser-induced Single-bit Faults in Flash Memory: Instructions Corruption on a 32-bit Microcontroller*", HOST (2019)

Laser fault injection on the matrix-vector mult.

Encap($\boldsymbol{H}_{\text{pub}}$) $\to$ ($\boldsymbol{s}$, $\boldsymbol{k}_{\text{session}}$)

1. Generate a random vector $\boldsymbol{e} \in \mathbb{F}_2^n$ of Hamming weight t
2. Compute $\boldsymbol{s} = \boldsymbol{H}_{\text{pub}} \boldsymbol{e}^{\mathsf{T}}$
3. $\boldsymbol{k}_{\text{session}} = \text{hash}(1, \boldsymbol{e}, \boldsymbol{s})$

Laser fault injection on the matrix-vector mult.

Encap($\boldsymbol{H}_{\text{pub}}$) $\rightarrow$ ($\boldsymbol{s}$, $\boldsymbol{k}_{\text{session}}$)

1. Generate a random vector $\boldsymbol{e} \in \mathbb{F}_2^n$ of Hamming weight t
2. Compute $\boldsymbol{s} = \boldsymbol{H}_{\text{pub}} \boldsymbol{e}^{\mathsf{T}}$
3. $\boldsymbol{k}_{\text{session}} = \text{hash}(1, \boldsymbol{e}, \boldsymbol{s})$

---

for $\text{row} \in [0, n-k-1]$ **do**
    $\boldsymbol{s}_{\text{row}} = 0$
    for $\text{col} \in [0, n-1]$ **do**
        $\boldsymbol{s}_{\text{row}} \oplus= \boldsymbol{H}_{\text{row,col}} \wedge \boldsymbol{e}_{\text{col}}$
return $\boldsymbol{s}$

Laser fault injection on the matrix-vector mult.

Encap($\boldsymbol{H}_{\text{pub}}$) $\rightarrow$ ($\boldsymbol{s}$, $\boldsymbol{k}_{\text{session}}$)

1. Generate a random vector $\boldsymbol{e} \in \mathbb{F}_2^n$ of Hamming weight t
2. Compute $\boldsymbol{s} = \boldsymbol{H}_{\text{pub}} \boldsymbol{e}^{\mathsf{T}}$
3. $\boldsymbol{k}_{\text{session}} = \text{hash}(1, \boldsymbol{e}, \boldsymbol{s})$

---

for $\text{row} \in [0, n-k-1]$ **do**
    $\boldsymbol{s}_{\text{row}} = 0$
    for $\text{col} \in [0, n-1]$ **do**
        $\boldsymbol{s}_{\text{row}} \oplus= \boldsymbol{H}_{\text{row,col}} \wedge \boldsymbol{e}_{\text{col}}$
return $\boldsymbol{s}$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

EORS: Rd = Rm ⊕ Rn											
0	1	0	0	0	0	0	0	0	1	Rm	Rdn

Laser fault injection on the matrix-vector mult.

Encap($\boldsymbol{H}_{\text{pub}}$) $\rightarrow$ ($\boldsymbol{s}$, $\boldsymbol{k}_{\text{session}}$)

1. Generate a random vector $\boldsymbol{e} \in \mathbb{F}_2^n$ of Hamming weight t
2. Compute $\boldsymbol{s} = \boldsymbol{H}_{\text{pub}} \boldsymbol{e}^{\mathsf{T}}$
3. $\boldsymbol{k}_{\text{session}} = \text{hash}(1, \boldsymbol{e}, \boldsymbol{s})$

---

for $\text{row} \in [0, n-k-1]$ **do**
    $\boldsymbol{s}_{\text{row}} = 0$
    for $\text{col} \in [0, n-1]$ **do**
        $\boldsymbol{s}_{\text{row}} \oplus= \boldsymbol{H}_{\text{row,col}} \wedge \boldsymbol{e}_{\text{col}}$
return $\boldsymbol{s}$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

EORS: Rd = Rm ⊕ Rn											
0	1	0	0	0	0	0	0	0	1	Rm	Rdn

ADCS: Rd = Rm + Rn											
0	1	0	0	0	0	0	1	0	1	Rm	Rdn

Laser fault injection on the matrix-vector mult.

Encap($\boldsymbol{H}_{\text{pub}}$) $\rightarrow$ ($\boldsymbol{s}$, $\boldsymbol{k}_{\text{session}}$)

1. Generate a random vector $\boldsymbol{e} \in \mathbb{F}_2^n$ of Hamming weight t
2. Compute $\boldsymbol{s} = \boldsymbol{H}_{\text{pub}} \boldsymbol{e}^{\mathsf{T}}$
3. $\boldsymbol{k}_{\text{session}} = \text{hash}(1, \boldsymbol{e}, \boldsymbol{s})$

---

for $\text{row} \in [0, n-k-1]$ **do**
  $\boldsymbol{s}_{\text{row}} = 0$
  for $\text{col} \in [0, n-1]$ **do**
    $\boldsymbol{s}_{\text{row}} \oplus= \boldsymbol{H}_{\text{row,col}} \wedge \boldsymbol{e}_{\text{col}}$
return $\boldsymbol{s}$

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

EORS: Rd = Rm ⊕ Rn											
0	1	0	0	0	0	0	0	0	1	Rm	Rdn

ADCS: Rd = Rm + Rn											
0	1	0	0	0	0	0	1	0	1	Rm	Rdn

We get $\boldsymbol{s} \in \mathbb{N}^{n-k}$ instead of $\mathbb{F}_2^{n-k}$

Integer syndrome decoding problem

Integer SDP

Inputs $\boldsymbol{H} \in \mathbb{F}_2^{(n-k)\times n}$ $\boldsymbol{s} \in \mathbb{N}^{n-k}$ $t \in \mathbb{N}^+$

Output $\boldsymbol{x} \in \mathbb{F}_2^n$ such that $\boldsymbol{H}\boldsymbol{x}^\mathsf{T} = \boldsymbol{s}$
with $\mathrm{HW}(\boldsymbol{x}) < t$

Syndrome decoding problem

Inputs $\boldsymbol{H} \in \mathbb{F}_2^{(n-k)\times n}$ $\boldsymbol{s} \in \mathbb{F}_2^{n-k}$ $t \in \mathbb{N}^+$

Output $\boldsymbol{x} \in \mathbb{F}_2^n$ such that $\boldsymbol{H}\boldsymbol{x}^\mathsf{T} = \boldsymbol{s}$
with $\mathrm{HW}(\boldsymbol{x}) < t$

Integer syndrome decoding problem

Integer SDP

Inputs $\boldsymbol{H} \in \mathbb{F}_2^{(n-k)\times n}$ $\boldsymbol{s} \in \mathbb{N}^{n-k}$ $t \in \mathbb{N}^+$

Output $\boldsymbol{x} \in \mathbb{F}_2^n$ such that $\boldsymbol{H}\boldsymbol{x}^\mathsf{T} = \boldsymbol{s}$
with $\mathrm{HW}(\boldsymbol{x}) < t$

Syndrome decoding problem

Inputs $\boldsymbol{H} \in \mathbb{F}_2^{(n-k)\times n}$ $\boldsymbol{s} \in \mathbb{F}_2^{n-k}$ $t \in \mathbb{N}^+$

Output $\boldsymbol{x} \in \mathbb{F}_2^n$ such that $\boldsymbol{H}\boldsymbol{x}^\mathsf{T} = \boldsymbol{s}$
with $\mathrm{HW}(\boldsymbol{x}) < t$

Integer syndrome decoding problem

Integer SDP

Inputs $\boldsymbol{H} \in \mathbb{F}_2^{(n-k)\times n}$ $\boldsymbol{s} \in \mathbb{N}^{n-k}$ $t \in \mathbb{N}^+$

Output $\boldsymbol{x} \in \mathbb{F}_2^n$ such that $\boldsymbol{H}\boldsymbol{x}^\mathsf{T} = \boldsymbol{s}$
with $\mathrm{HW}(\boldsymbol{x}) < t$

Syndrome decoding problem

Inputs $\boldsymbol{H} \in \mathbb{F}_2^{(n-k)\times n}$ $\boldsymbol{s} \in \mathbb{F}_2^{n-k}$ $t \in \mathbb{N}^+$

Output $\boldsymbol{x} \in \mathbb{F}_2^n$ such that $\boldsymbol{H}\boldsymbol{x}^\mathsf{T} = \boldsymbol{s}$
with $\mathrm{HW}(\boldsymbol{x}) < t$

ILP program

Inputs $\boldsymbol{A} \in \mathbb{N}^{(n-k)\times n}$ $\boldsymbol{c} \in \mathbb{N}^{n-k}$ $\boldsymbol{b} \in \mathbb{N}^n$

Object. $\min\{\boldsymbol{b}^\mathsf{T}\boldsymbol{x} \mid \boldsymbol{A}\boldsymbol{x}^\mathsf{T} = \boldsymbol{c}, \boldsymbol{x} \in \mathbb{N}^n\}$

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

Integer syndrome decoding problem

Integer SDP

Inputs $\boldsymbol{H} \in \mathbb{F}_2^{(n-k)\times n}$ $\boldsymbol{s} \in \mathbb{N}^{n-k}$ $t \in \mathbb{N}^+$
Output $\boldsymbol{x} \in \mathbb{F}_2^n$ such that $\boldsymbol{H}\boldsymbol{x}^\mathsf{T} = \boldsymbol{s}$
with $\mathrm{HW}(\boldsymbol{x}) < t$

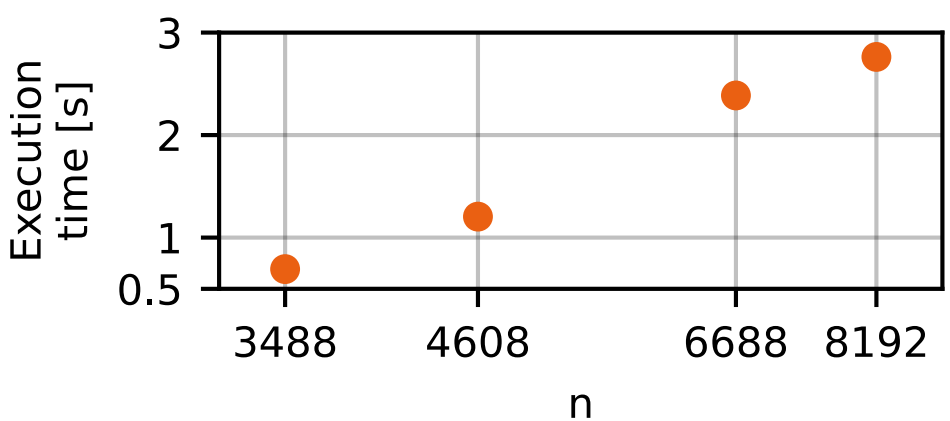
Syndrome decoding problem

Inputs $\boldsymbol{H} \in \mathbb{F}_2^{(n-k)\times n}$ $\boldsymbol{s} \in \mathbb{F}_2^{n-k}$ $t \in \mathbb{N}^+$
Output $\boldsymbol{x} \in \mathbb{F}_2^n$ such that $\boldsymbol{H}\boldsymbol{x}^\mathsf{T} = \boldsymbol{s}$
with $\mathrm{HW}(\boldsymbol{x}) < t$

ILP program

Inputs $\boldsymbol{A} \in \mathbb{N}^{(n-k)\times n}$ $\boldsymbol{c} \in \mathbb{N}^{n-k}$ $\boldsymbol{b} \in \mathbb{N}^n$
Object. $\min\{\boldsymbol{b}^\mathsf{T}\boldsymbol{x} \mid \boldsymbol{A}\boldsymbol{x}^\mathsf{T} = \boldsymbol{c}, \boldsymbol{x} \in \mathbb{N}^n\}$
with $\boldsymbol{b} = (1, 1, ..., 1)$, $\boldsymbol{x} \in \{0, 1\}^n$

Solved with `Scipy.optimize.linprog`



Summary

- 👍 easy to **express**
- 👍 solvable with a **generic** ILP solver
- 👍 reasonably **efficient**

Summary

- 👍 easy to **express**
- 👍 solvable with a **generic** ILP solver
- 👍 reasonably **efficient**

- 👎 restrictive **attacker model**
- 👎 does not tolerate **any error** in the integer syndrome $s \in \mathbb{N}^{n-k}$
- 👎 matrix-vector product over $\mathbb{F}_2$ is **not** implemented like this...

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

Bits are **packed** into bytes and arithmetic over $\mathbb{F}_2$ is **bitsliced**.

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

Bits are **packed** into bytes and arithmetic over $\mathbb{F}_2$ is **bitsliced**.

---

for $\mathrm{row} \in [0, n - mt - 1]$ **do**

  $b = 0$

  for $\mathrm{col} \in \left[0, \frac{n}{8} - 1\right]$ **do**

    $b \oplus = \boldsymbol{H}_{\mathrm{row},\mathrm{col}} \wedge \boldsymbol{e}_{\mathrm{col}}$ bitsliced multiply and accumulate

  $b \oplus = b \gg 4$

  $b \oplus = b \gg 2$ XOR folding

  $b \oplus = b \gg 1$

  $b \wedge = 1$ LSB extraction

  $\boldsymbol{s}_{\mathrm{row}/8} \vee = b \ll (\mathrm{row} \bmod 8)$ bit-level packing

return $\boldsymbol{s}$

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

Bits are **packed** into bytes and arithmetic over $\mathbb{F}_2$ is **bitsliced**.

---

for $\mathrm{row} \in [0, n - mt - 1]$ **do**

  $b = 0$

  for $\mathrm{col} \in \left[0, \frac{n}{8} - 1\right]$ **do**

    $b \oplus = \boldsymbol{H}_{\mathrm{row},\mathrm{col}} \wedge \boldsymbol{e}_{\mathrm{col}}$ bitsliced multiply and accumulate

  $b \oplus = b \gg 4$

  $b \oplus = b \gg 2$ XOR folding

  $b \oplus = b \gg 1$

  $b \wedge = 1$ LSB extraction

  $\boldsymbol{s}_{\mathrm{row}/8} \vee = b \ll (\mathrm{row} \bmod 8)$ bit-level packing

return $\boldsymbol{s}$

Profiled side-channel attack on the packed matrix-vector multiplication

Side-channel analysis of multiply and accumulate

for $\mathrm{col} \in \left[0, \frac{n}{8} - 1\right]$ **do**
  $b \oplus = \boldsymbol{H}_{\mathrm{row},\mathrm{col}} \wedge \boldsymbol{e}_{\mathrm{col}}$

Side-channel analysis of multiply and accumulate

for $\mathrm{col} \in \left[0, \frac{n}{8} - 1\right]$ **do**
  $b \oplus = \boldsymbol{H}_{\mathrm{row},\mathrm{col}} \wedge \boldsymbol{e}_{\mathrm{col}}$

$b = \texttt{00000000}$
$b = \texttt{0000}\texttt{1}\texttt{000}$
$b = \texttt{0000}\texttt{1}\texttt{000}$
$b = \texttt{0000}\texttt{1}\texttt{0}\texttt{1}\texttt{0}$

Side-channel analysis of multiply and accumulate

for $\mathrm{col} \in \left[0, \frac{n}{8} - 1\right]$ **do**
  $b \oplus = \boldsymbol{H}_{\mathrm{row},\mathrm{col}} \wedge \boldsymbol{e}_{\mathrm{col}}$

$b =$ `00000000` HW = 0
$b =$ `0000`1`000` HW = 1
$b =$ `0000`1`000` HW = 1
$b =$ `0000`1`0`1`0` HW = 2

Side-channel analysis of multiply and accumulate

for $\mathrm{col} \in \left[0, \frac{n}{8} - 1\right]$ **do**
  $b \oplus = \boldsymbol{H}_{\mathrm{row},\mathrm{col}} \wedge \boldsymbol{e}_{\mathrm{col}}$

HD = 1
HD = 0
HD = 1

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

Side-channel analysis of multiply and accumulate

for $\mathrm{col} \in \left[0, \frac{n}{8} - 1\right]$ **do**
  $b \oplus = \boldsymbol{H}_{\mathrm{row},\mathrm{col}} \wedge \boldsymbol{e}_{\mathrm{col}}$

HD = 1
HD = 0
HD = 1

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

Integer syndrome from Hamming distance or Hamming weight leakage

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

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

Hamming distance VS Hamming weight leakage

Hamming distance

$$s_j = \sum_{i=1}^{\frac{n}{8}-1} \mathtt{HD}\left(\boldsymbol{b}_{j,i}, \boldsymbol{b}_{j,i-1}\right)$$

Sum the number of ones in $\boldsymbol{H}_{\mathrm{row,col}} \wedge \boldsymbol{e}_{\mathrm{col}}$

Hamming weight

$$s_j = \sum_{i=1}^{\frac{n}{8}-1} \mid \mathtt{HW}\left(\boldsymbol{b}_{j,i}\right) - \mathtt{HW}\left(\boldsymbol{b}_{j,i-1}\right) \mid$$
$$\text{if } \mathtt{HD}\left(\boldsymbol{b}_{j,i}, \boldsymbol{b}_{j,i-1}\right) \leq 1$$

Hamming distance VS Hamming weight leakage

Hamming distance

$$s_j = \sum_{i=1}^{\frac{n}{8}-1} \mathtt{HD}\left(\boldsymbol{b}_{j,i}, \boldsymbol{b}_{j,i-1}\right)$$

Sum the number of ones in $\boldsymbol{H}_{\text{row,col}} \wedge \boldsymbol{e}_{\text{col}}$

Hamming weight

$$s_j = \sum_{i=1}^{\frac{n}{8}-1} \mid \mathtt{HW}\left(\boldsymbol{b}_{j,i}\right) - \mathtt{HW}\left(\boldsymbol{b}_{j,i-1}\right) \mid$$
$$\text{if } \mathtt{HD}\left(\boldsymbol{b}_{j,i}, \boldsymbol{b}_{j,i-1}\right) \leq 1$$

Edge case:

HD=2

$b =$ `00001000` HW=1

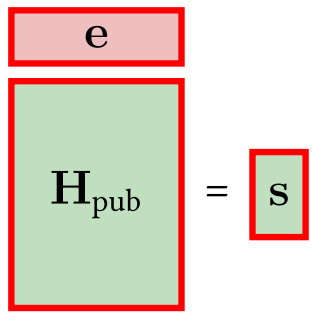
$b =$ `00000100` HW=1

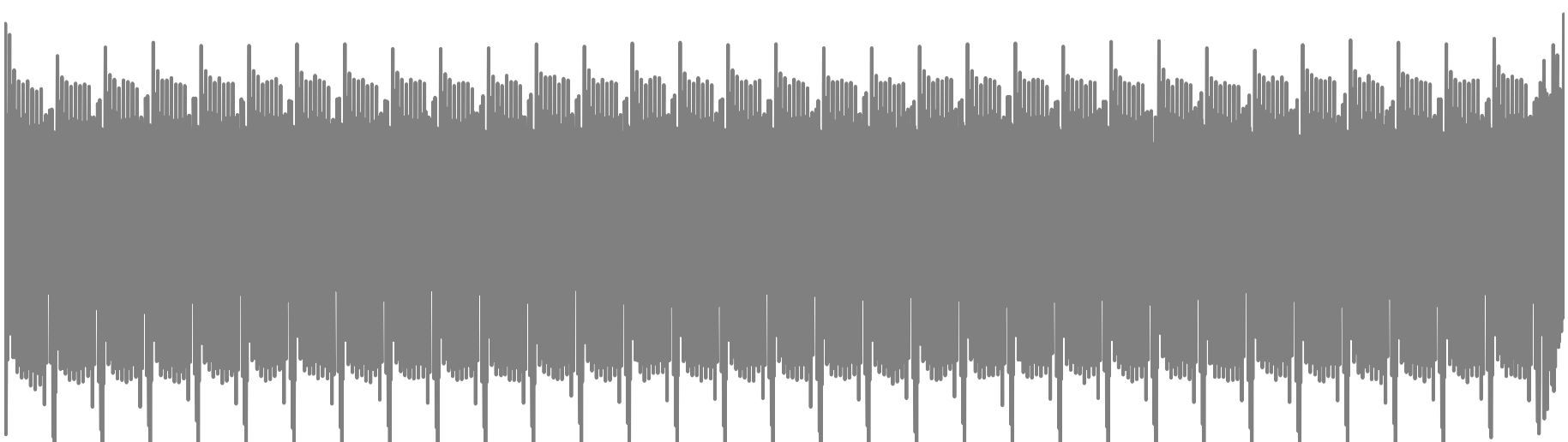
Happens when $\mathtt{HW}\left(\boldsymbol{H}_{\text{row,col}} \wedge \boldsymbol{e}_{\text{col}}\right) > 1$

Unlikely since $\mathtt{HW}(\boldsymbol{e}) = t$

Side-channel analysis

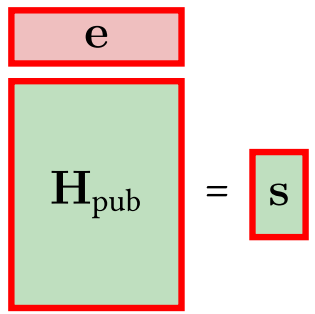
$$s = H_{\text{pub}} e^{\mathsf{T}}$$

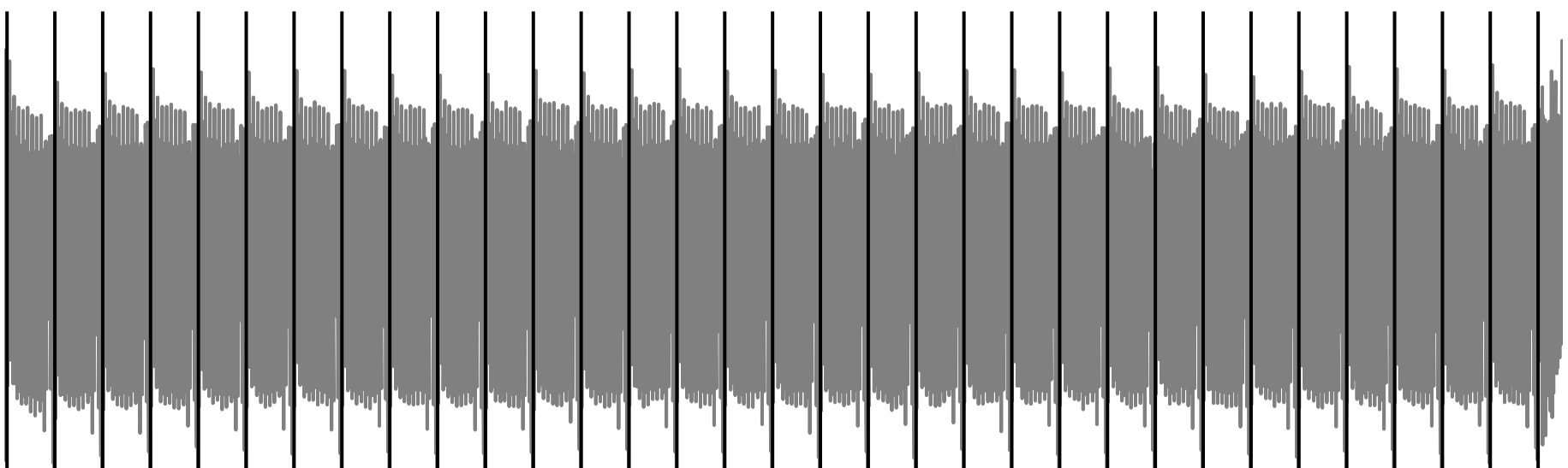




Side-channel analysis

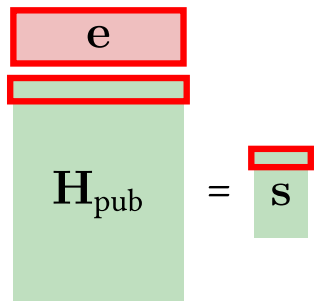
$$s = H_{\text{pub}} e^{\mathsf{T}}$$

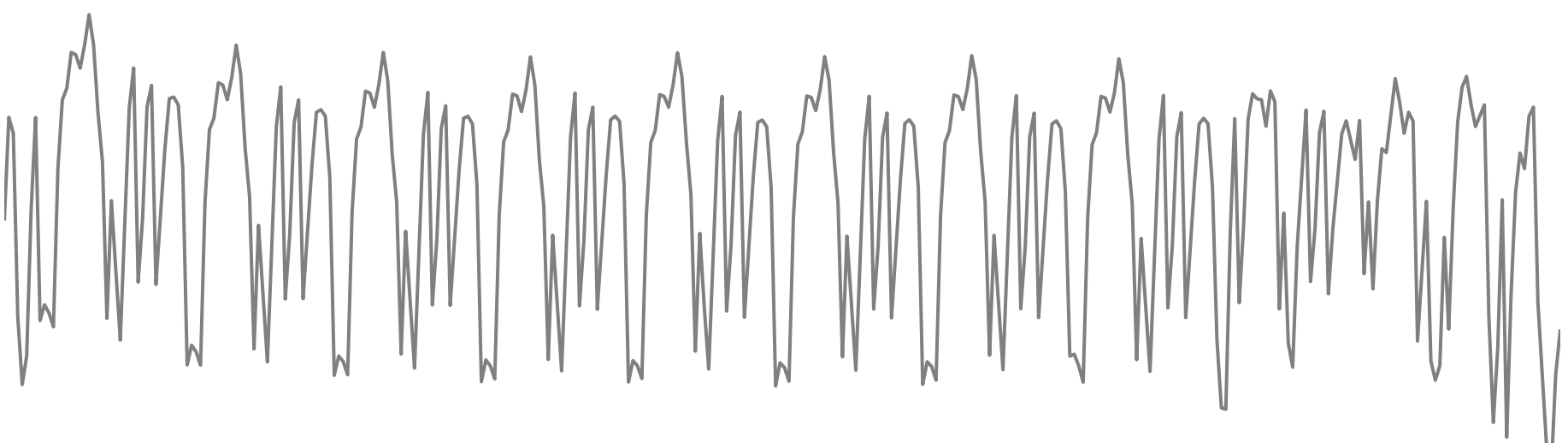




Side-channel analysis

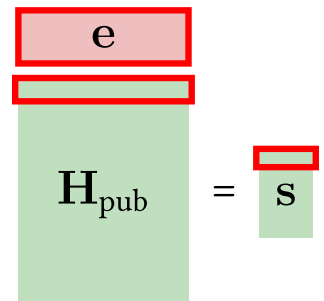
$$s_j = H_{\text{pub}_{[j,]}} \cdot e$$

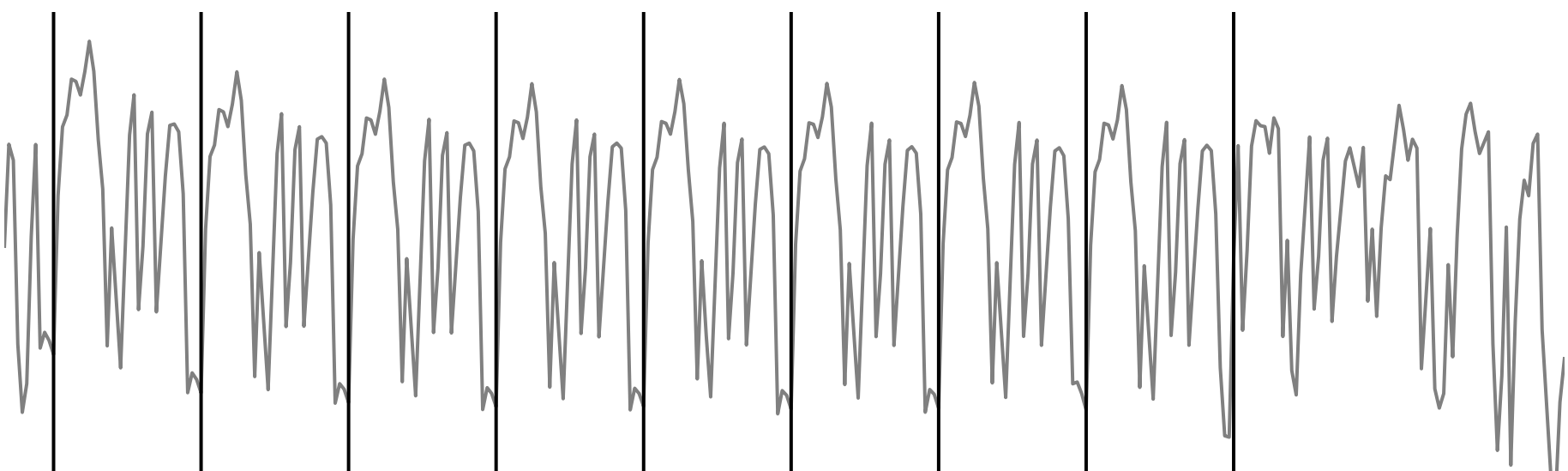




Side-channel analysis

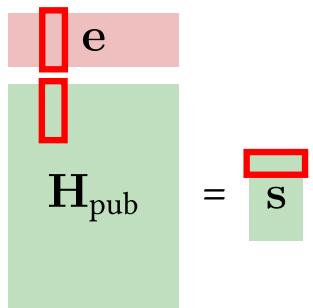
$$s_j = H_{\text{pub}_{[j,]}} \cdot e$$



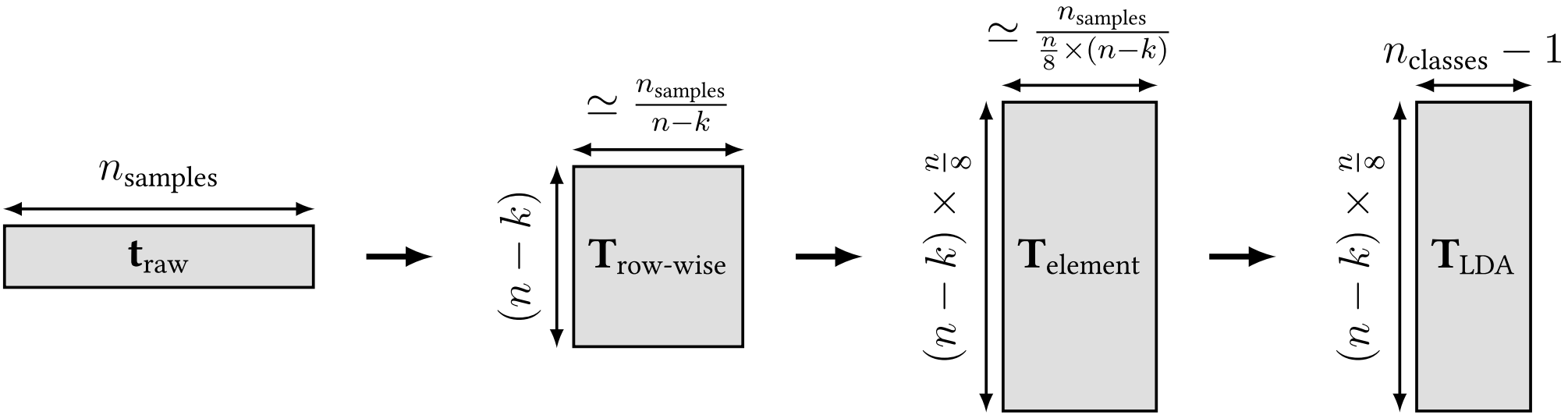


Side-channel analysis

$$b = \boldsymbol{H}_{\mathrm{pub}_{[j,i]}} \boldsymbol{e}_i$$



Trace reshaping process



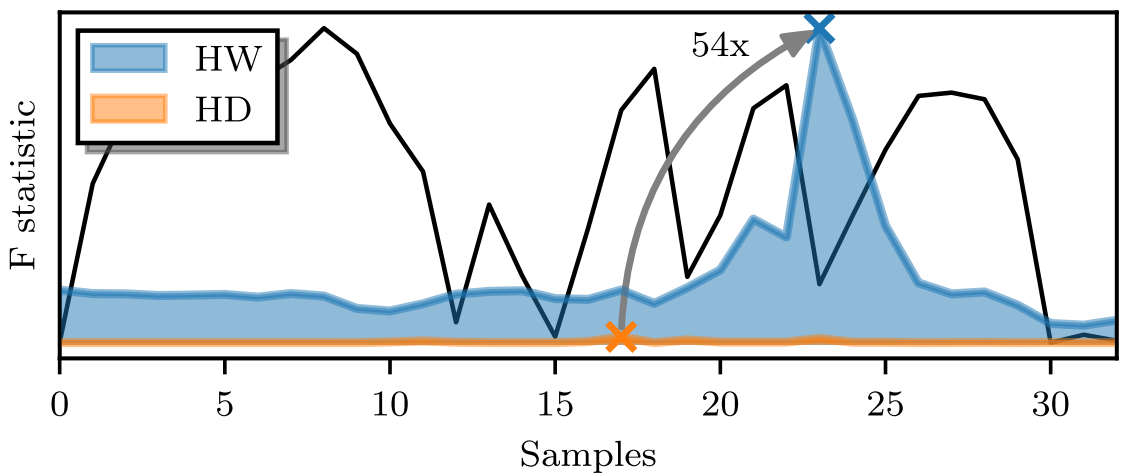
Training phase

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

Random Forest classifier

Random Forest classifier training:

- Hamming weight
  - $> 99.5\%$ accuracy
- Hamming distance
  - 80% accuracy



Outcome

- **Hamming weight** can be recovered very accurately,
- but not the **Hamming distance**...
- One can compute a *slightly innacurate* integer syndrome

Quantitative group testing

The $\mathbb{N}$-SDP can be reframed as the Quantitative Group Testing problem [10]

Abstract

Given a random Bernoulli matrix $A \in \{0,1\}^{m \times n}$, an integer $0 < k < n$ and the vector $y := Ax$, where $x \in \{0,1\}^n$ is of Hamming weight k, the objective in the *Quantitative Group Testing* (QGT) problem is to recover x.

We want to find **which columns** of H_{pub} **contributed** to s.

[10] Uriel Feige, Amir Lellouche "*Quantitative Group Testing and the rank of random matrices*", CoRR (2020)

The score function

The **dot product** is used to compute a score for the columns of $\boldsymbol{H}_{\text{pub}}$

Score function

$$\psi_i(\boldsymbol{s}) = \boldsymbol{H}_{\text{pub}_{[,i]}} \cdot \boldsymbol{s} + \overline{\boldsymbol{H}_{\text{pub}_{[,i]}}} \cdot \overline{\boldsymbol{s}} \quad \text{where } \overline{\boldsymbol{H}_{[i,j]}} = 1 - \boldsymbol{H}_{[i,j]} \quad \text{and} \quad \overline{\boldsymbol{s}_i} = t - \boldsymbol{s}_i$$

The score function

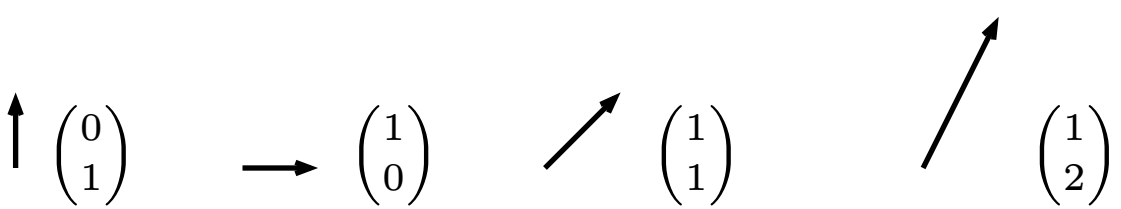
The **dot product** is used to compute a score for the columns of $\boldsymbol{H}_{\text{pub}}$

Score function

$$\psi_i(\boldsymbol{s}) = \boldsymbol{H}_{\text{pub}_{[,i]}} \cdot \boldsymbol{s} + \overline{\boldsymbol{H}_{\text{pub}_{[,i]}}} \cdot \overline{\boldsymbol{s}} \quad \text{where } \overline{\boldsymbol{H}_{[i,j]}} = 1 - \boldsymbol{H}_{[i,j]} \quad \text{and} \quad \overline{\boldsymbol{s}_i} = t - \boldsymbol{s}_i$$

Exemple: $t = 2 = \mathtt{HW}(\boldsymbol{e})$

$$\boldsymbol{H}_{\text{pub}}\boldsymbol{e}^\intercal = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}\boldsymbol{e}^\intercal = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \in \mathbb{N}^2$$



The score function

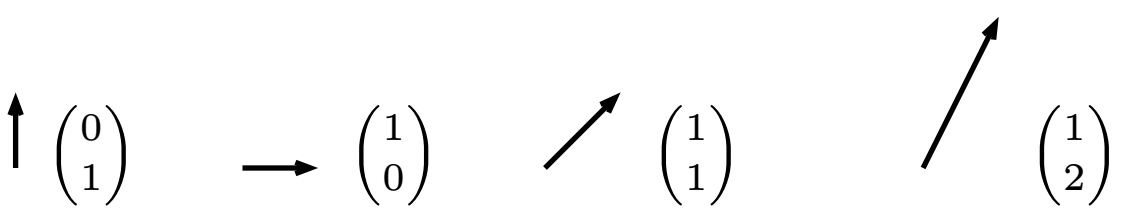
The **dot product** is used to compute a score for the columns of $\boldsymbol{H}_{\text{pub}}$

Score function

$$\psi_i(\boldsymbol{s}) = \boldsymbol{H}_{\text{pub}_{[,i]}} \cdot \boldsymbol{s} + \overline{\boldsymbol{H}_{\text{pub}_{[,i]}}} \cdot \overline{\boldsymbol{s}} \quad \text{where } \overline{\boldsymbol{H}_{[i,j]}} = 1 - \boldsymbol{H}_{[i,j]} \quad \text{and} \quad \overline{\boldsymbol{s}_i} = t - \boldsymbol{s}_i$$

Exemple: $t = 2 = \mathtt{HW}(\boldsymbol{e})$

$$\boldsymbol{H}_{\text{pub}}\boldsymbol{e}^{\mathsf{T}} = \begin{pmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix}\boldsymbol{e}^{\mathsf{T}} = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \in \mathbb{N}^2$$



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

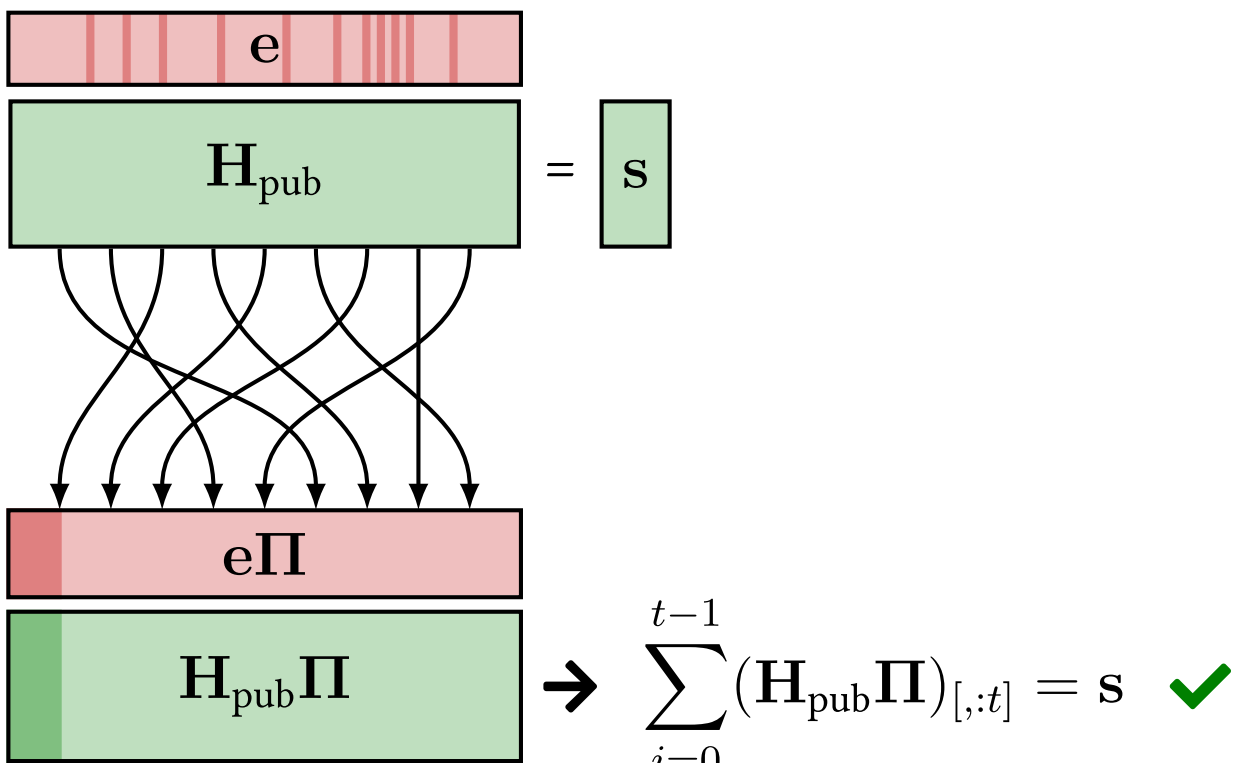
From the score to the support

for $i \in [0, n-1]$ **do**
    Compute $\psi_i(\boldsymbol{s})$
$\Pi \leftarrow \text{sort } \psi(\boldsymbol{s})$
return Π

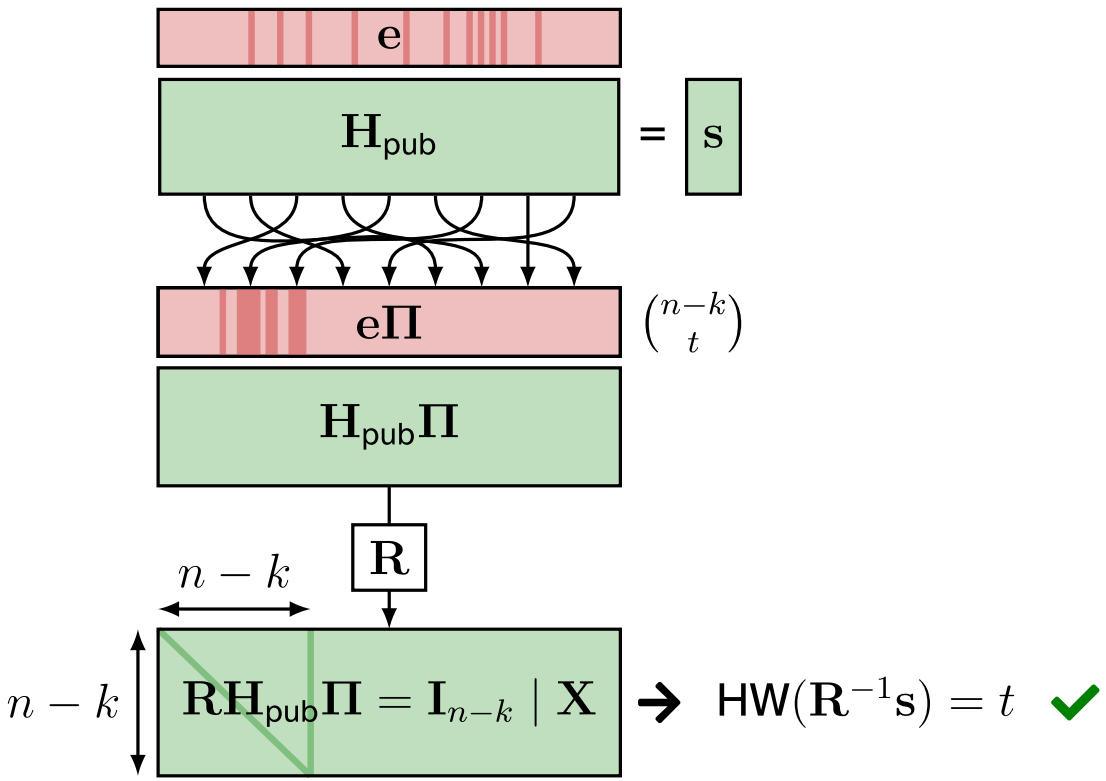
From the score to the support

1 **for** $i \in [0, n-1]$ **do**
2 ⌞ Compute $\psi_i(\boldsymbol{s})$
3 $\Pi \leftarrow \text{sort } \psi(\boldsymbol{s})$
4 **return** Π

Best case scenario: t-threshold decoder



Information-set decoding [11] strategies



[11] Eugene Prange "*The use of information sets in decoding cyclic codes*", IRE Transactions on Information Theory (1962)

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

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

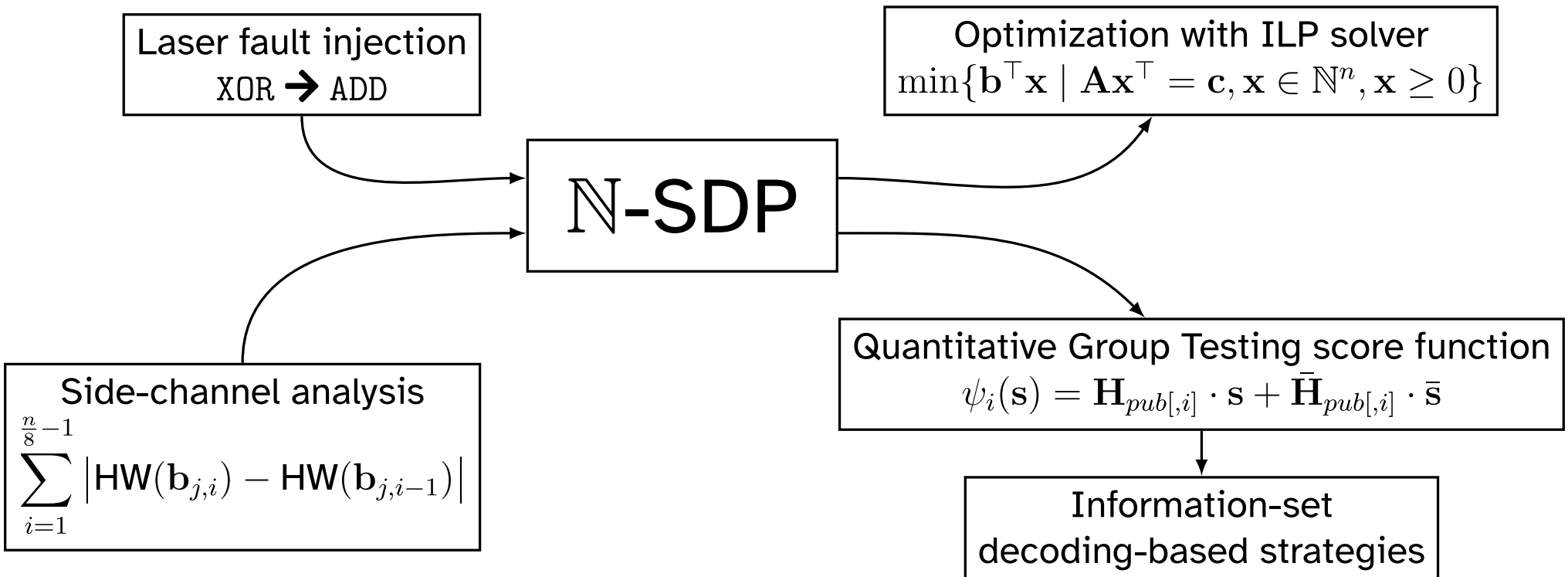
- 👍 is computationally efficient
- 👍 tolerates some errors in the integer syndrome
- 👍 gets more efficient with larger cryptographic parameters
- 👎 cannot handle high noise levels $\rightarrow$ **[12]**
- 👎 cannot handle large (32/64-bit) registers $\rightarrow$ **[13]**

---

[12] Vlad-Florin Drăgoi, Brice Colombier, Pierre-Louis Cayrel, Vincent Grosso "*Integer Syndrome Decoding in the Presence of Noise*", Springer Cryptography and Communications (2024)

[13] Vincent Grosso, Pierre-Louis Cayrel, Brice Colombier, Vlad-Florin Drăgoi "*Punctured Syndrome Decoding Problem - Efficient Side-Channel Attacks Against Classic McEliece*", COSADE (2023)

Conclusion on the $\mathbb{N}$-SDP



Unprofiled side-channel attack on the packed matrix-vector multiplication

Unprofiled side-channel attack

All attacks introduced previously were **profiled** attacks, running in two steps:

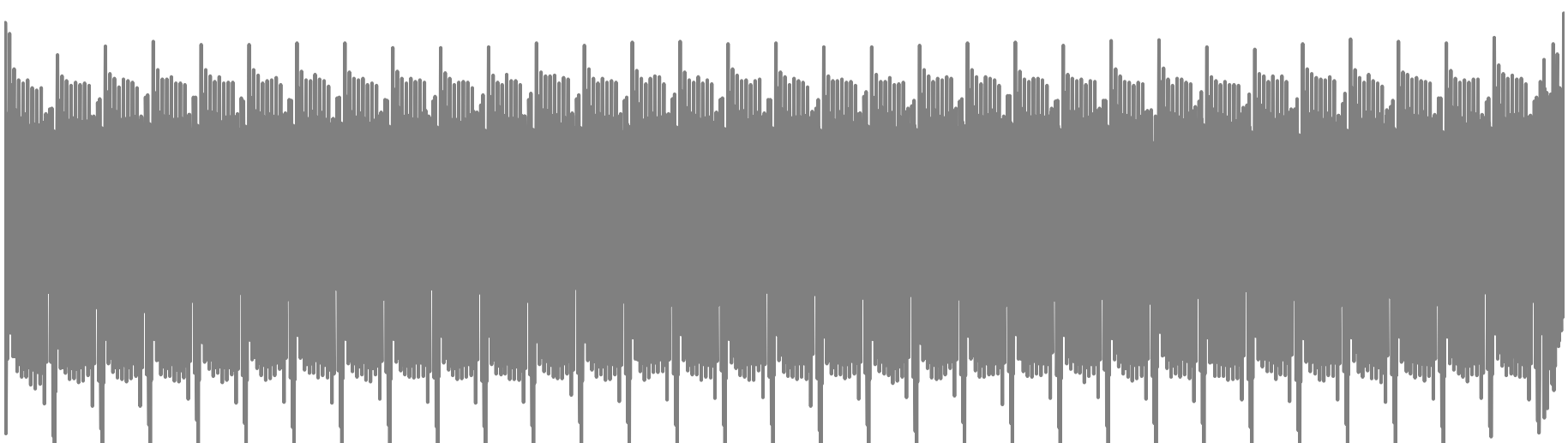
1. **build** templates on an **open** device,
2. **use** the templates on a **target** device.

Unprofiled side-channel attack

All attacks introduced previously were **profiled** attacks, running in two steps:

1. **build** templates on an **open** device,
2. **use** the templates on a **target** device.

Back to an attack trace:



Unprofiled side-channel attack

All attacks introduced previously were **profiled** attacks, running in two steps:

1. **build** templates on an **open** device,
2. **use** the templates on a **target** device.

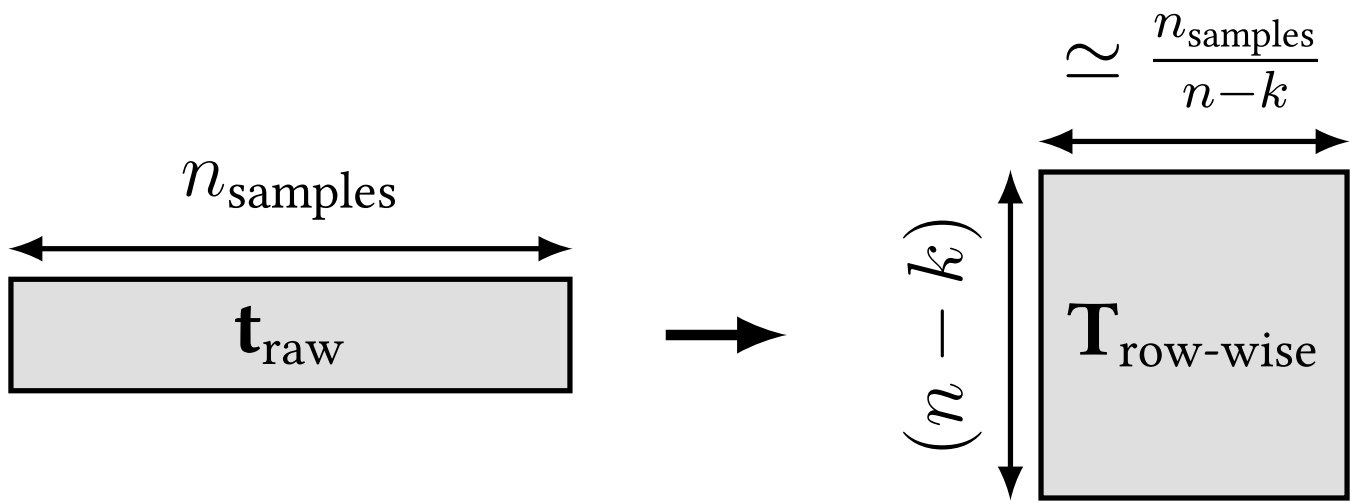
Back to an attack trace:

Unprofiled side-channel attack

All attacks introduced previously were **profiled** attacks, running in two steps:

1. **build** templates on an **open** device,
2. **use** the templates on a **target** device.

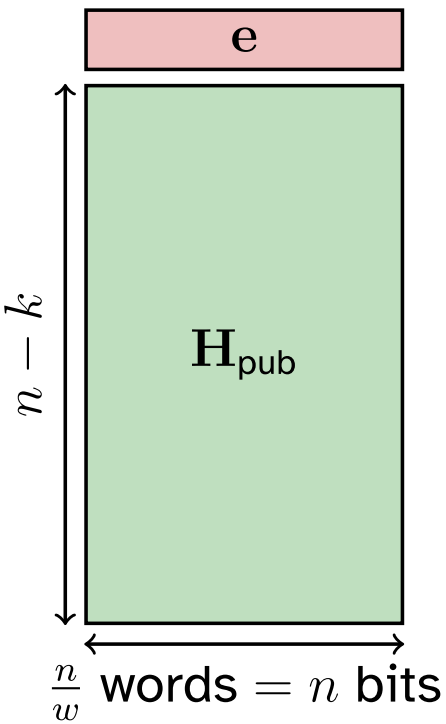
Back to an attack trace:

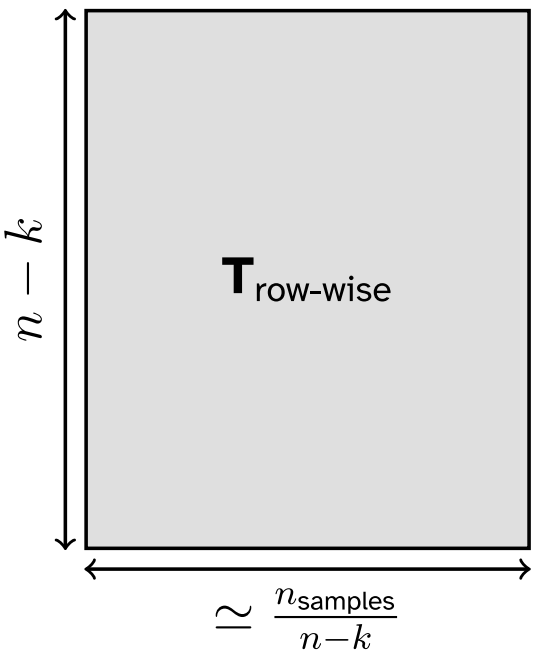


From error positions to Hamming dist. leakage

$\mathbf{e}[\mathtt{i}]$	$\mathbf{H}_{\text{pub}}[\mathtt{i},\mathtt{j}]$	$\mathbf{e} \wedge \mathbf{H}_{\text{pub}}$
0	0	0
0	1	0
1	0	0
1	1	1

$\mathtt{b}[\mathtt{i}]$	$\mathbf{e} \wedge \mathbf{H}_{\text{pub}}$	$\oplus$	$\mathcal{L}_{HD}$
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	1

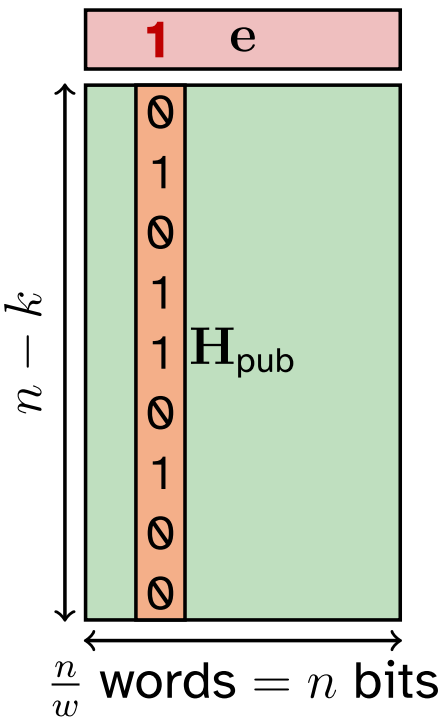


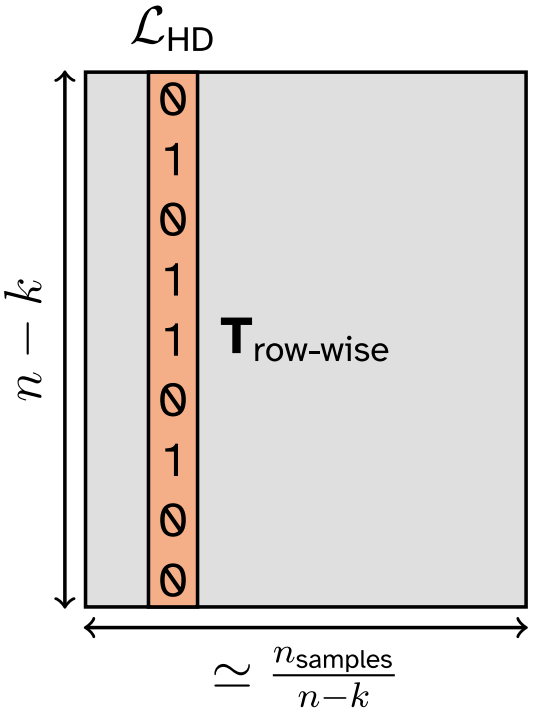


From error positions to Hamming dist. leakage

$\mathbf{e}[\mathtt{i}]$	$\mathbf{H}_{\text{pub}}[\mathtt{i},\mathtt{j}]$	$\mathbf{e} \wedge \mathbf{H}_{\text{pub}}$
0	0	0
0	1	0
1	0	0
1	1	1

$\mathtt{b}[\mathtt{i}]$	$\mathbf{e} \wedge \mathbf{H}_{\text{pub}}$	$\oplus$	$\mathcal{L}_{HD}$
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	1

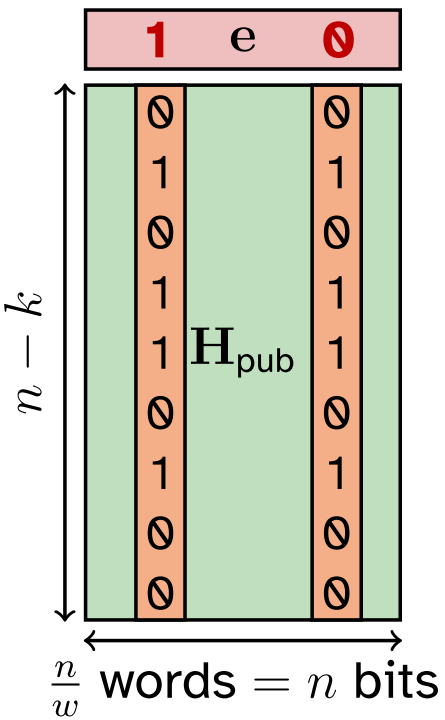


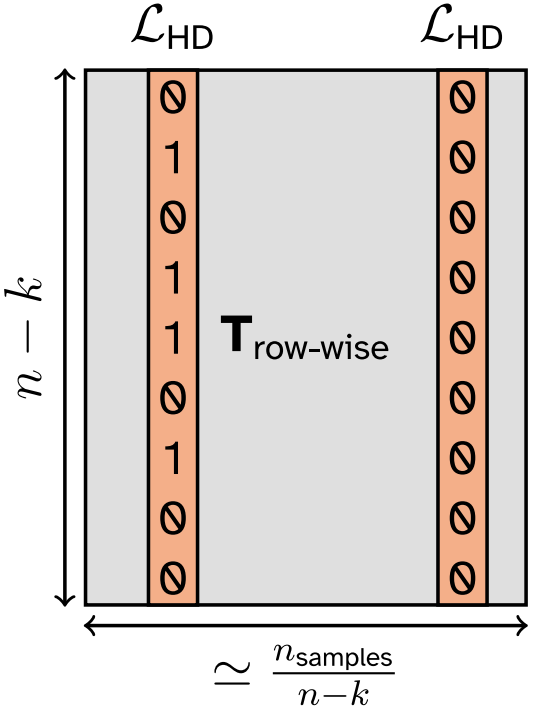


From error positions to Hamming dist. leakage

$\mathbf{e}[\mathtt{i}]$	$\mathbf{H}_{\text{pub}}[\mathtt{i},\mathtt{j}]$	$\mathbf{e} \wedge \mathbf{H}_{\text{pub}}$
0	0	0
0	1	0
1	0	0
1	1	1

$\mathtt{b}[\mathtt{i}]$	$\mathbf{e} \wedge \mathbf{H}_{\text{pub}}$	$\oplus$	$\mathcal{L}_{HD}$
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	1

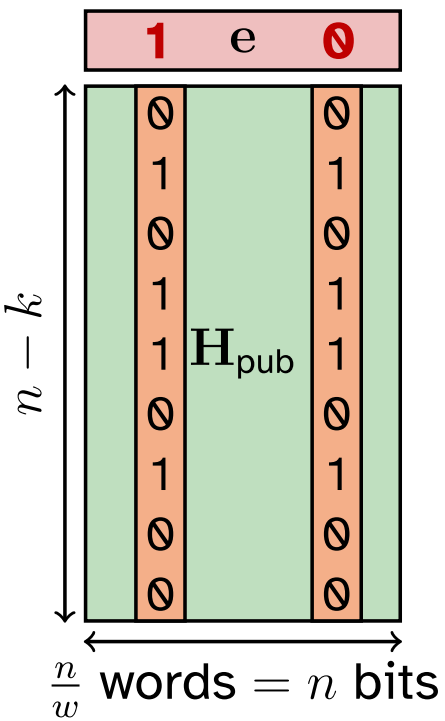


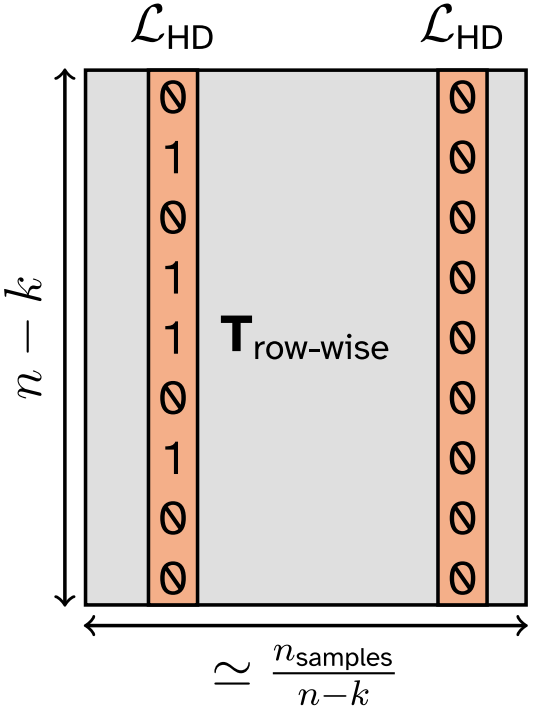


From error positions to Hamming dist. leakage

$\mathbf{e}[\mathtt{i}]$	$\mathbf{H}_{\text{pub}}[\mathtt{i},\mathtt{j}]$	$\mathbf{e} \wedge \mathbf{H}_{\text{pub}}$
0	0	0
0	1	0
1	0	0
1	1	1

$\mathtt{b}[\mathtt{i}]$	$\mathbf{e} \wedge \mathbf{H}_{\text{pub}}$	$\oplus$	$\mathcal{L}_{HD}$
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	1



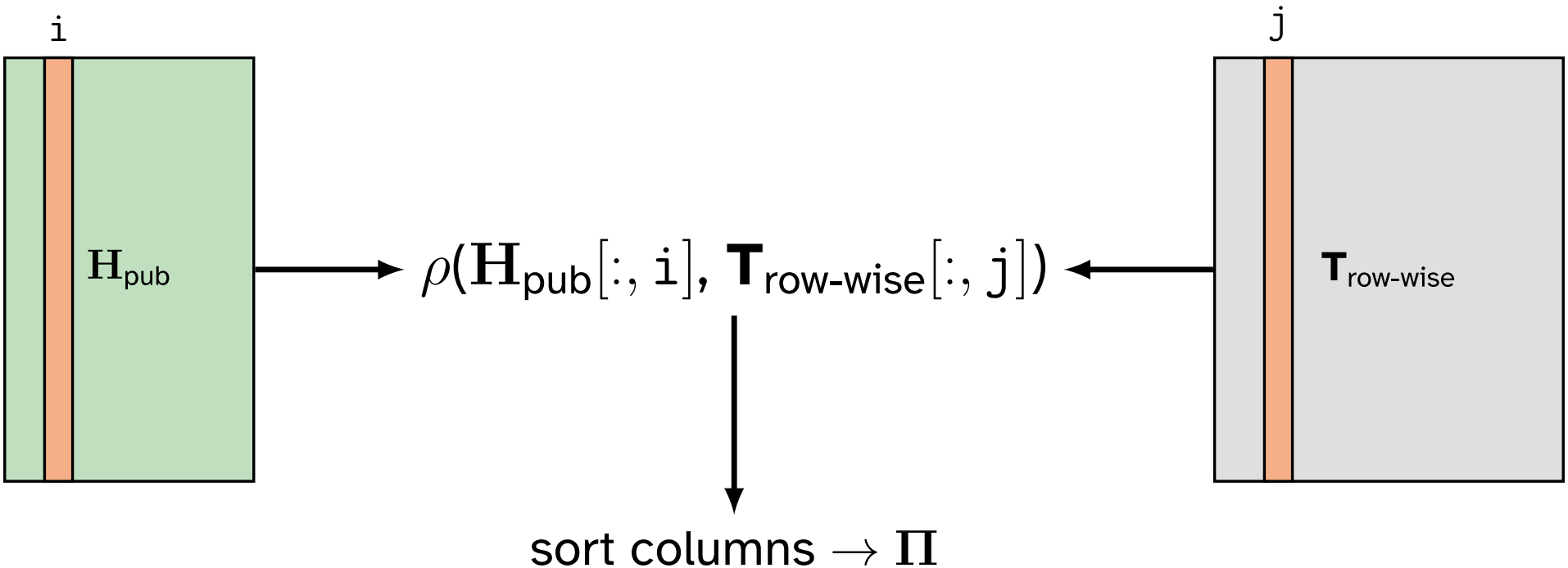


t columns of HD leakage in $\boldsymbol{T}_{\text{row-wise}}$ match columns of $\boldsymbol{H}_{\text{pub}}$ that face a 1 in $\boldsymbol{e}$.

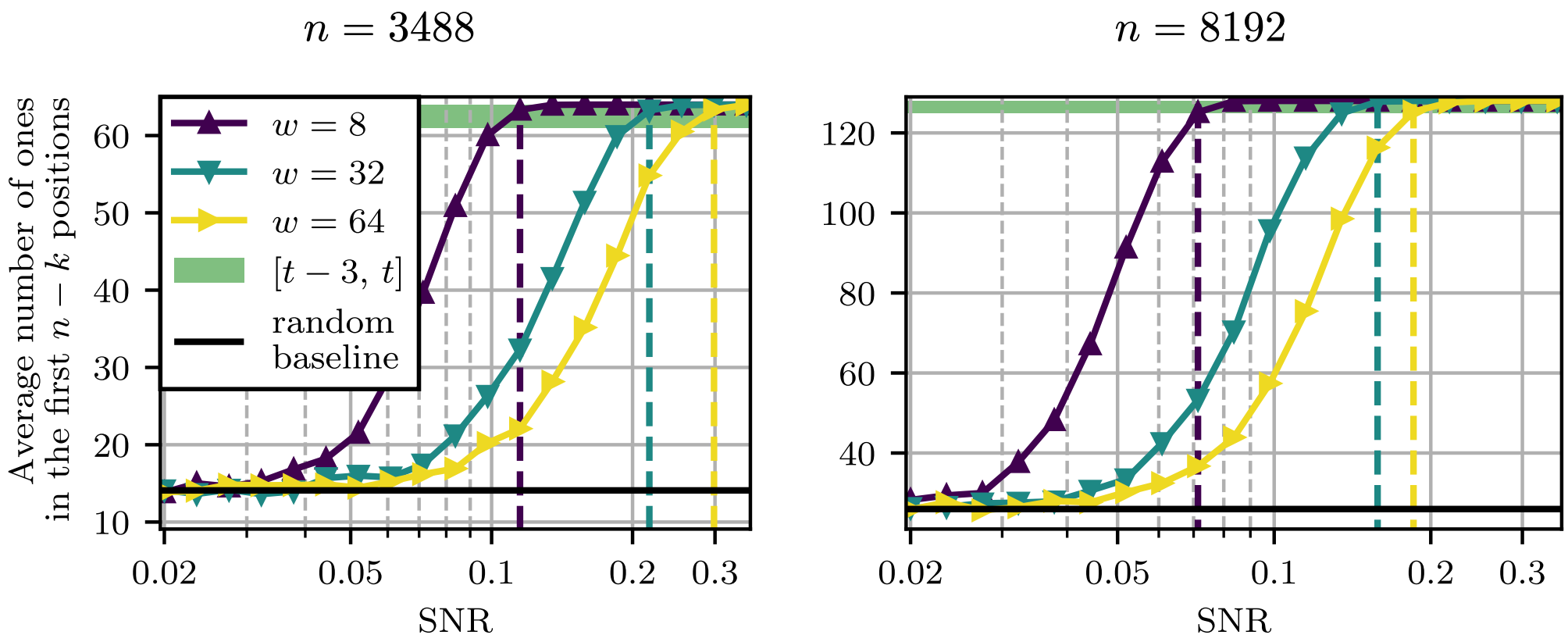
Horizontal correlation attack

Horizontal **correlation** attack between:

- the columns of $\boldsymbol{H}_{\text{pub}}$
- the “columns” of observed Hamming dist. leakage



Experimental results (simulated traces)



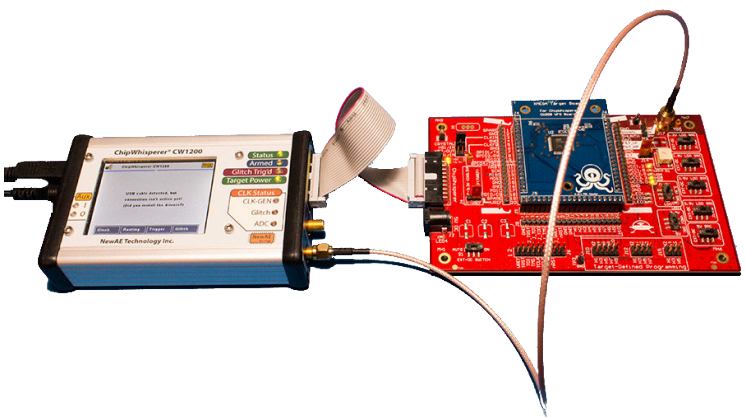
Best success rate with **smaller words** and **larger cryptographic parameters**.

Real-life traces

Reference implementation running on the ChipWhisperer [14] platform.

Target device:

- ARM Cortex-M4 core with **32-bit** registers
- $256\,\mathrm{kB}$ of flash memory (only...)
  - cryptographic parameters are scaled [15]
  - $(n, k, t) = (1600, 1280, 30)$
- compilation optimization levels:
  - `-O0`, `-O1`, `-O2`, `-O3` and `-Os`
  - from 419 to 7080 clock cycles

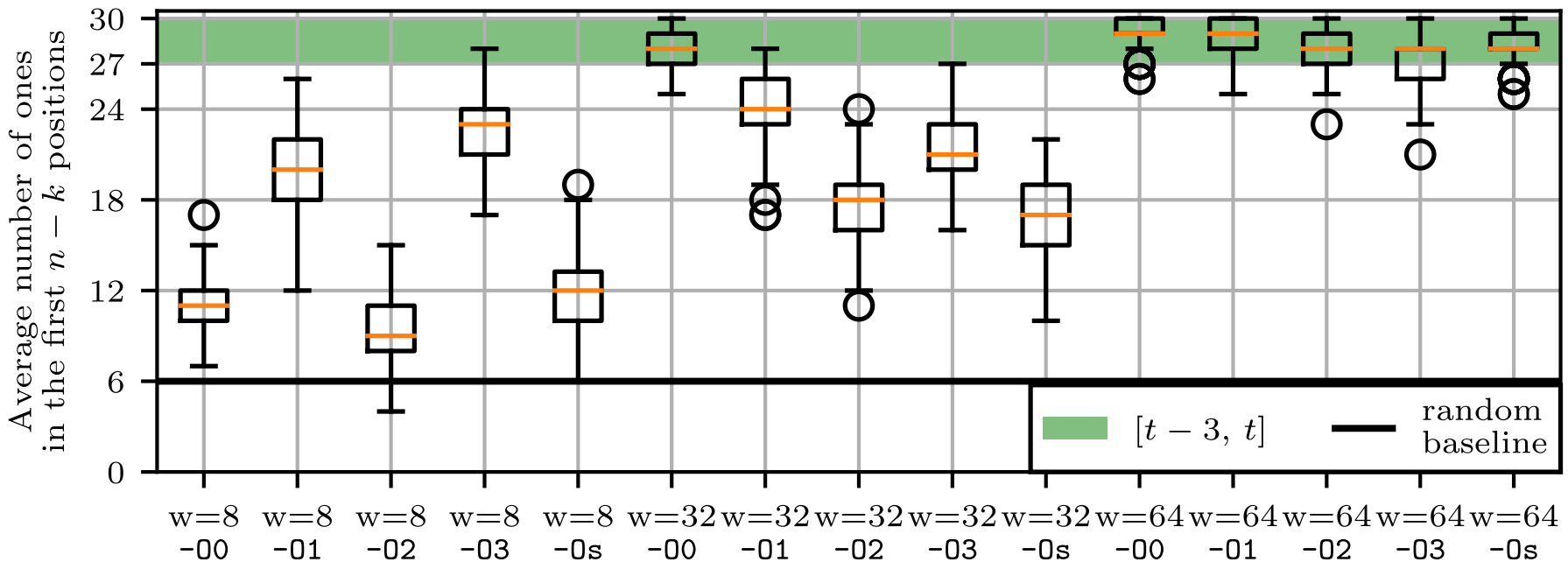


---

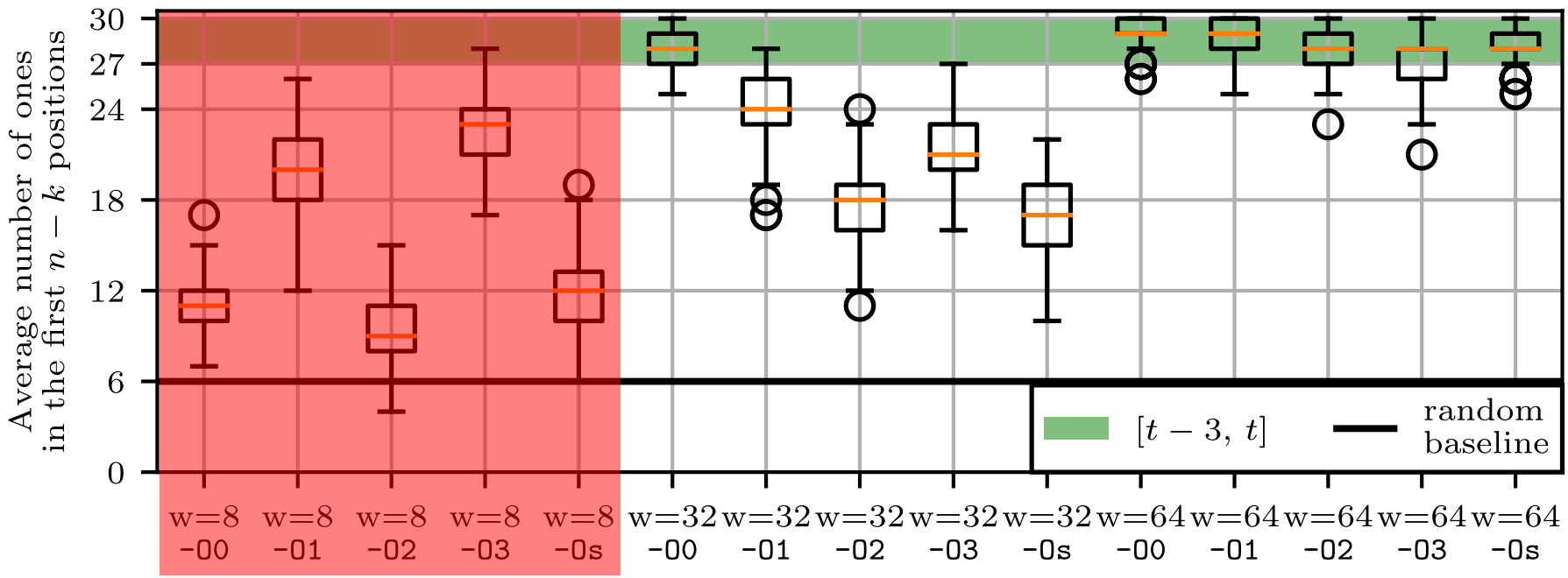
[14] Colin O'Flynn, Zhizhang (David) Chen "*ChipWhisperer: An Open-Source Platform for Hardware Embedded Security Research*", COSADE (2014)

[15] https://decodingchallenge.org/goppa

Experimental results (real-life traces)



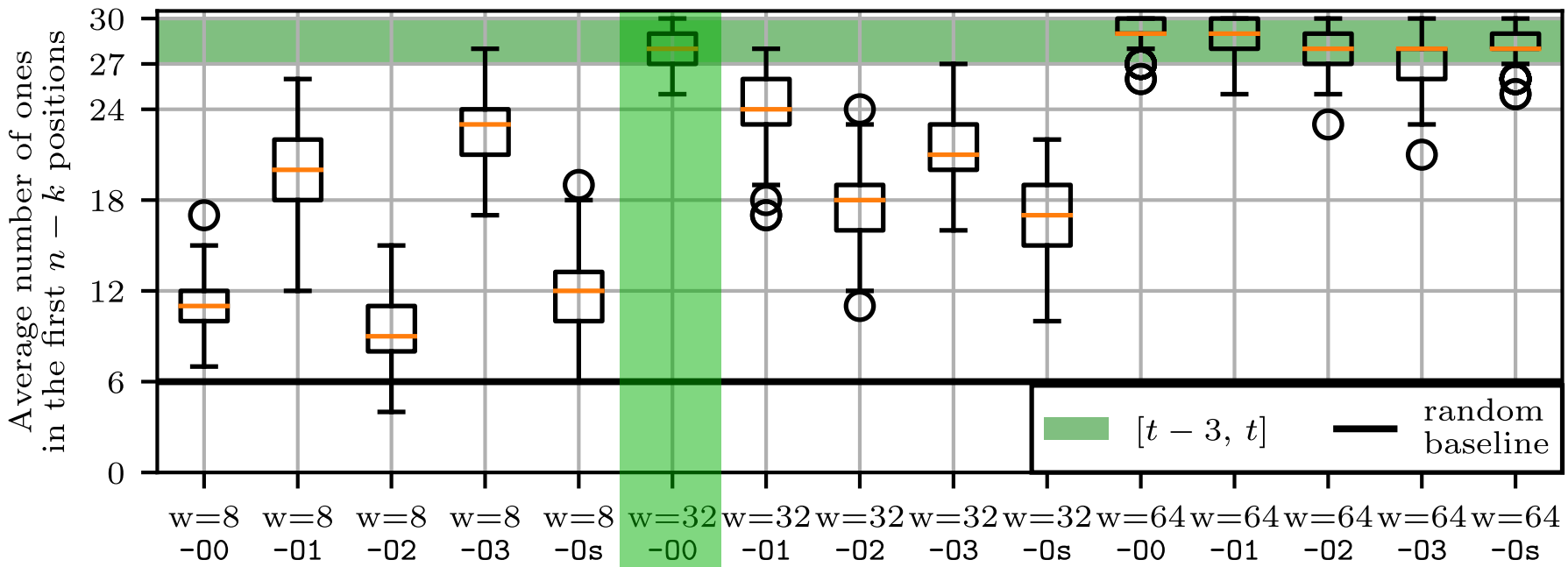
Experimental results (real-life traces)



Attack does not work for $w = 8$

- Lots of sub-word-size memory accesses,
- Strong Hamming weight leakage, not much Hamming distance.

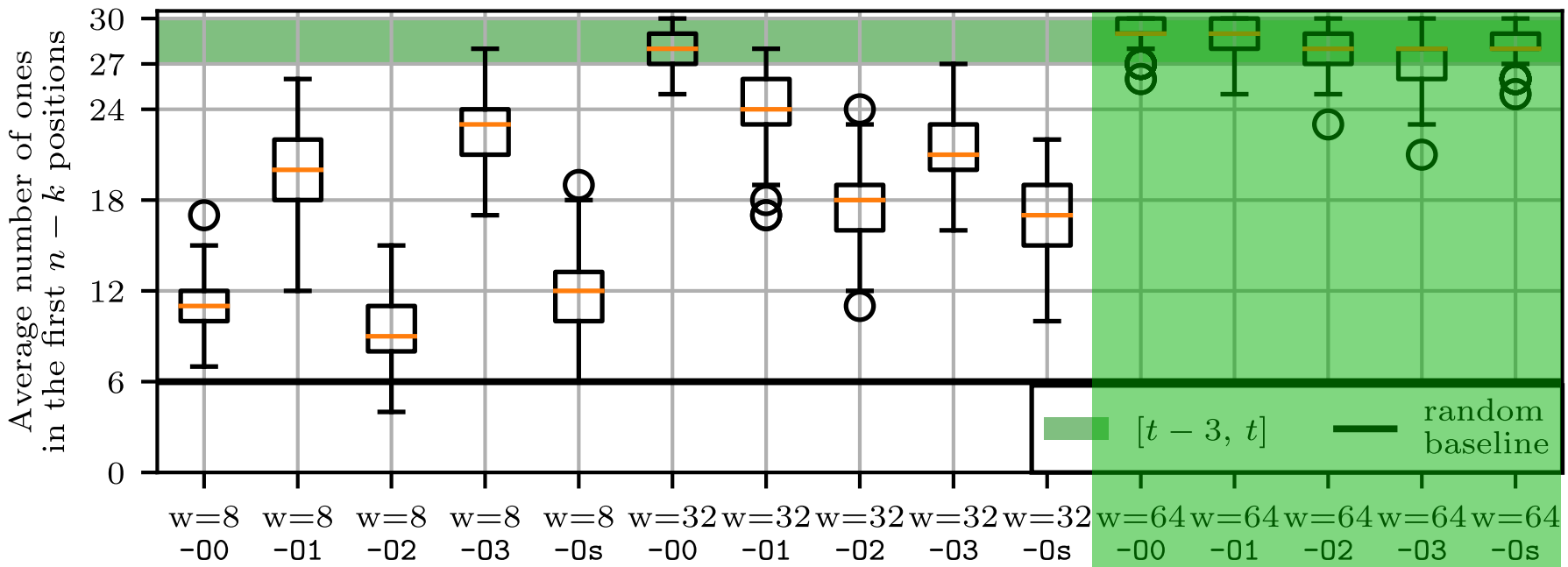
Experimental results (real-life traces)



Attack works for $w = 32$ and `-00`

- Strong Hamming distance leakage,
- `load` $\rightarrow$ `eors` $\rightarrow$ `store` sequence

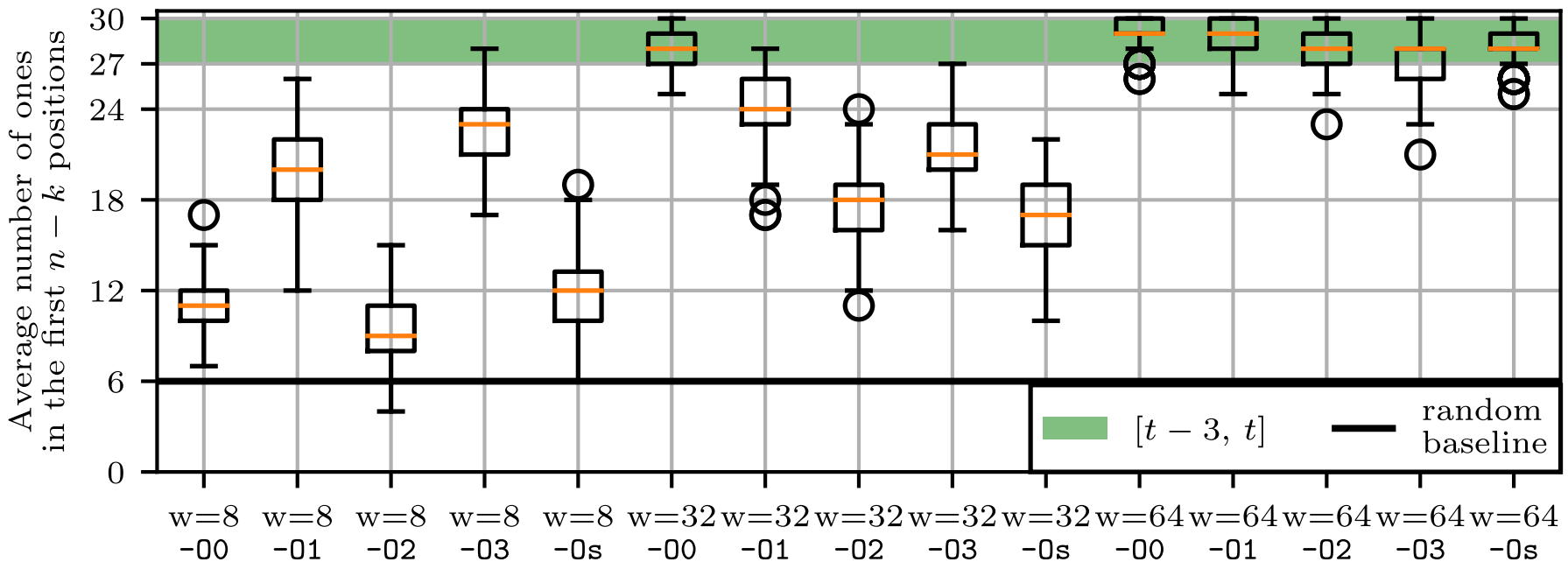
Experimental results (real-life traces)



Attack works for $w = 64$

- ???

Experimental results (real-life traces)



Experiments contradict simulations: **larger** words are **easier** to attack...

Conclusion on message-recovery attacks

✓✓ **profiled** attacks apply on a wide range of implementations

✗ **unprofiled** attacks would require more experiments:

- microcontrollers for which $w = 8$ and $w = 64$ are the **native** word width
- hardware implementations
- assembly-level countermeasures to prevent Hamming distance leakage

Conclusion on message-recovery attacks

✓✓ **profiled** attacks apply on a wide range of implementations

✗ **unprofiled** attacks would require more experiments:

- microcontrollers for which $w = 8$ and $w = 64$ are the **native** word width
- hardware implementations
- assembly-level countermeasures to prevent Hamming distance leakage

Next step: target **long-term** secrets.

Key-recovery attack

Classic McEliece private key

- $\boldsymbol{g}$ a monic polynomial of degree t in $\mathbb{F}_{2^m}[x]$
- $\mathcal{L}$ a random subset $\{\alpha_0, \alpha_1, ..., \alpha_{n-1}\}$ of $\mathbb{F}_{2^m}$

From there, the **public** key $\boldsymbol{H}_{\text{pub}}$ is computed as follows:

1. Compute the $t \times n$ parity-check matrix $\boldsymbol{H} =$

$$\begin{pmatrix} \boldsymbol{g}^{-1}(\alpha_0) & \boldsymbol{g}^{-1}(\alpha_1) & \dots & \boldsymbol{g}^{-1}(\alpha_{n-1}) \\ \alpha_0\boldsymbol{g}^{-1}(\alpha_0) & \alpha_1\boldsymbol{g}^{-1}(\alpha_1) & \dots & \alpha_{n-1}\boldsymbol{g}^{-1}(\alpha_{n-1}) \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_0^{t-1}\boldsymbol{g}^{-1}(\alpha_0) & \alpha_1^{t-1}\boldsymbol{g}^{-1}(\alpha_1) & \dots & \alpha_{n-1}^{t-1}\boldsymbol{g}^{-1}(\alpha_{n-1}) \end{pmatrix}$$

2. Map $\boldsymbol{H}$ from $\mathbb{F}_{2^m}$ to $\mathbb{F}_2^m$
3. Get $\boldsymbol{H}$ into standard form $\left(\boldsymbol{I}_{mt} \mid \boldsymbol{H}_{\text{pub}}\right)$ $\rightarrow$ secret change-of-basis matrix

Previous works

- Attack on KeyGen during the Gaussian elimination [16]
  - restrictive attacker model + software simulation only
- Attacks on Decap
  - [17] requires n traces + invalid syndromes (corresponding to $\mathtt{HW}(e) = 1$)
  - [18] Hamming weight of g coeff. + exhaustive search $\rightarrow$ very complex
- Study on how some “hints” can help to recover the key [19]

---

[16] Marcus Brinkmann, Chitchanok Chuengsatiansup, Alexander May, et al. "*Leaky McEliece: Secret Key Recovery From Highly Erroneous Side-Channel Information*", IACR TCHES (2025)

[17] Qian Guo, Andreas Johansson, Thomas Johansson "*A Key-Recovery Side-Channel Attack on Classic McEliece Implementations*", IACR TCHES (2022)

[18] Boly Seck, Pierre-Louis Cayrel, Vlad-Florin Drăgoi, et al. "*A Side-Channel Attack Against Classic McEliece When Loading the Goppa Polynomial*", AFRICACRYPT (2023)

[19] Elena Kirshanova, Alexander May "*Breaking Goppa-based McEliece with hints*", Information and Computation (2023)

Private key usage

The private key $\boldsymbol{k}_{\text{priv}}$

- $\boldsymbol{g}$ a monic polynomial of degree t in $\mathbb{F}_{2^m}[x]$
- $\mathcal{L}$ a random subset $\{\alpha_0, \alpha_1, ..., \alpha_{n-1}\}$ of $\mathbb{F}_{2^m}$

is used in **Decap** to compute the **private** parity-check matrix $\boldsymbol{H}_{\text{priv}_{g^2}}$

$$\begin{pmatrix} \boldsymbol{g}^{-2}(\alpha_0) & \boldsymbol{g}^{-2}(\alpha_1) & \dots & \boldsymbol{g}^{-2}(\alpha_{n-1}) \\ \alpha_0\boldsymbol{g}^{-2}(\alpha_0) & \alpha_1\boldsymbol{g}^{-2}(\alpha_1) & \dots & \alpha_{n-1}\boldsymbol{g}^{-2}(\alpha_{n-1}) \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_0^{2t-1}\boldsymbol{g}^{-2}(\alpha_0) & \alpha_1^{2t-1}\boldsymbol{g}^{-2}(\alpha_1) & \dots & \alpha_{n-1}^{2t-1}\boldsymbol{g}^{-2}(\alpha_{n-1}) \end{pmatrix}$$

Private key usage

The private key $\boldsymbol{k}_{\text{priv}}$

- $\boldsymbol{g}$ a monic polynomial of degree t in $\mathbb{F}_{2^m}[x]$
- $\mathcal{L}$ a random subset $\{\alpha_0, \alpha_1, ..., \alpha_{n-1}\}$ of $\mathbb{F}_{2^m}$

is used in **Decap** to compute the **private** parity-check matrix $\boldsymbol{H}_{\text{priv}_{g^2}}$

$$\begin{pmatrix} \boldsymbol{g}^{-2}(\alpha_0) & \boldsymbol{g}^{-2}(\alpha_1) & \dots & \boldsymbol{g}^{-2}(\alpha_{n-1}) \\ \alpha_0\boldsymbol{g}^{-2}(\alpha_0) & \alpha_1\boldsymbol{g}^{-2}(\alpha_1) & \dots & \alpha_{n-1}\boldsymbol{g}^{-2}(\alpha_{n-1}) \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_0^{2t-1}\boldsymbol{g}^{-2}(\alpha_0) & \alpha_1^{2t-1}\boldsymbol{g}^{-2}(\alpha_1) & \dots & \alpha_{n-1}^{2t-1}\boldsymbol{g}^{-2}(\alpha_{n-1}) \end{pmatrix}$$

Private key usage

The private key $\boldsymbol{k}_{\text{priv}}$

- $\boldsymbol{g}$ a monic polynomial of degree t in $\mathbb{F}_{2^m}[x]$
- $\mathcal{L}$ a random subset $\{\alpha_0, \alpha_1, ..., \alpha_{n-1}\}$ of $\mathbb{F}_{2^m}$

is used in **Decap** to compute the **private** parity-check matrix $\boldsymbol{H}_{\text{priv}_{g^2}}$

$$\begin{pmatrix} \boldsymbol{g}^{-2}(\alpha_0) & \boldsymbol{g}^{-2}(\alpha_1) & \dots & \boldsymbol{g}^{-2}(\alpha_{n-1}) \\ \alpha_0\boldsymbol{g}^{-2}(\alpha_0) & \alpha_1\boldsymbol{g}^{-2}(\alpha_1) & \dots & \alpha_{n-1}\boldsymbol{g}^{-2}(\alpha_{n-1}) \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_0^{2t-1}\boldsymbol{g}^{-2}(\alpha_0) & \alpha_1^{2t-1}\boldsymbol{g}^{-2}(\alpha_1) & \dots & \alpha_{n-1}^{2t-1}\boldsymbol{g}^{-2}(\alpha_{n-1}) \end{pmatrix}$$

Private parity-check matrix usage

For decapsulation, the syndrome $\boldsymbol{s}$ is:

1. Padded with zeros: $\boldsymbol{v} = (\boldsymbol{s}, 0, ..., 0)$
2. Multiplied by $\boldsymbol{H}_{\text{priv}_{g^2}}$ which is **computed on-the-fly** to get $\boldsymbol{s}_{\text{priv}} = \boldsymbol{H}_{\text{priv}_{g^2}} \boldsymbol{v}$

Private parity-check matrix usage

For decapsulation, the syndrome s is:

1. Padded with zeros: $v = (s, 0, ..., 0)$
2. Multiplied by $H_{\text{priv}_{g^2}}$ which is **computed on-the-fly** to get $s_{\text{priv}} = H_{\text{priv}_{g^2}} v$

$$\begin{pmatrix} g^{-2}(\alpha_0) & g^{-2}(\alpha_1) & \dots & g^{-2}(\alpha_{n-1}) \\ \alpha_0 g^{-2}(\alpha_0) & \alpha_1 g^{-2}(\alpha_1) & \dots & \alpha_{n-1} g^{-2}(\alpha_{n-1}) \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_0^{2t-1} g^{-2}(\alpha_0) & \alpha_1^{2t-1} g^{-2}(\alpha_1) & \dots & \alpha_{n-1}^{2t-1} g^{-2}(\alpha_{n-1}) \end{pmatrix}$$

for row $\in [0, 2t-1]$ **do**
  $s_{\text{priv}_{\text{row}}} = 0$
for col $\in [0, n-1]$ **do**
  $b \leftarrow g^{-2}(\alpha_{\text{col}})$
  for row $\in [0, 2t-1]$ **do**
    $s_{\text{priv}_{\text{row}}} \leftarrow s_{\text{priv}_{\text{row}}} + v_{\text{col}} \times b$
    $b \leftarrow b \times \alpha_{\text{col}}$
return s

Private parity-check matrix usage

For decapsulation, the syndrome s is:

1. Padded with zeros: $v = (s, 0, ..., 0)$
2. Multiplied by $H_{\text{priv}_{g^2}}$ which is **computed on-the-fly** to get $s_{\text{priv}} = H_{\text{priv}_{g^2}} v$

for row $\in [0, 2t-1]$ **do**
  $s_{\text{priv}_{\text{row}}} = 0$
for col $\in [0, n-1]$ **do**
  $b \leftarrow g^{-2}(\alpha_{\text{col}})$
  for row $\in [0, 2t-1]$ **do**
    $s_{\text{priv}_{\text{row}}} \leftarrow s_{\text{priv}_{\text{row}}} + v_{\text{col}} \times b$
    $b \leftarrow b \times \alpha_{\text{col}}$
return s

$$\begin{pmatrix} g^{-2}(\alpha_0) & g^{-2}(\alpha_1) & \dots & g^{-2}(\alpha_{n-1}) \\ \alpha_0 g^{-2}(\alpha_0) & \alpha_1 g^{-2}(\alpha_1) & \dots & \alpha_{n-1} g^{-2}(\alpha_{n-1}) \\ \vdots & \vdots & \ddots & \vdots \\ \alpha_0^{2t-1} g^{-2}(\alpha_0) & \alpha_1^{2t-1} g^{-2}(\alpha_1) & \dots & \alpha_{n-1}^{2t-1} g^{-2}(\alpha_{n-1}) \end{pmatrix}$$

Consecutive values of b are:

$$[g^{-2}(\alpha), \alpha g^{-2}(\alpha), \alpha^2 g^{-2}(\alpha), ..., \alpha^{2t-1} g^{-2}(\alpha)]$$

A sequence of interest

We consider the following sequence (a column of $\boldsymbol{H}_{\mathrm{priv}_{g^2}}$)

$$[\boldsymbol{g}^{-2}(\alpha), \alpha\boldsymbol{g}^{-2}(\alpha), \alpha^2\boldsymbol{g}^{-2}(\alpha), ..., \alpha^{2t-1}\boldsymbol{g}^{-2}(\alpha)]$$
$$[\beta \qquad , \alpha\beta \qquad , \alpha^2\beta \qquad , ..., \alpha^{2t-1}\beta]$$

can be rewritten as: $\left[\alpha^j\beta\right]_{j\in[0,2t-1]}$ with $\alpha \in \mathbb{F}_{2^m}^*$ and $\beta \in \mathbb{F}_{2^m}^*$

A sequence of interest

We consider the following sequence (a column of $\boldsymbol{H}_{\mathrm{priv}_{g^2}}$)

$$[\boldsymbol{g}^{-2}(\alpha), \alpha\boldsymbol{g}^{-2}(\alpha), \alpha^2\boldsymbol{g}^{-2}(\alpha), ..., \alpha^{2t-1}\boldsymbol{g}^{-2}(\alpha)]$$
$$[\beta \qquad , \alpha\beta \qquad , \alpha^2\beta \qquad , ..., \alpha^{2t-1}\beta]$$

can be rewritten as: $\left[\alpha^j\beta\right]_{j\in[0,2t-1]}$ with $\alpha \in \mathbb{F}_{2^m}^*$ and $\beta \in \mathbb{F}_{2^m}^*$

Side-channel analysis can recover the Hamming weights: $\left[\mathtt{HW}\left(\alpha^j\beta\right)\right]_{j\in[0,2t-1]}$

A sequence of interest

We consider the following sequence (a column of $\boldsymbol{H}_{\mathrm{priv}_{g^2}}$)

$$[\boldsymbol{g}^{-2}(\alpha), \alpha\boldsymbol{g}^{-2}(\alpha), \alpha^2\boldsymbol{g}^{-2}(\alpha), ..., \alpha^{2t-1}\boldsymbol{g}^{-2}(\alpha)]$$
$$[\beta \quad , \alpha\beta \quad , \alpha^2\beta \quad , ..., \alpha^{2t-1}\beta]$$

can be rewritten as: $\left[\alpha^j\beta\right]_{j\in[0,2t-1]}$ with $\alpha \in \mathbb{F}_{2^m}^*$ and $\beta \in \mathbb{F}_{2^m}^*$

Side-channel analysis can recover the Hamming weights: $\left[\mathtt{HW}\left(\alpha^j\beta\right)\right]_{j\in[0,2t-1]}$

Vandermonde distinguisher

The sequence of Hamming weights $\left[\mathtt{HW}\left(\alpha^j\beta\right)\right]_{j\in[0,2t-1]}$ is a very strong ($> 99\%$) distinguisher for the (α, β) pair. (if the sequence is long enough)

Profiled attack scenario: offline phase

1. Precompute all possible $\left[\mathtt{HW}\left(\alpha^j \beta\right)\right]_{j \in [0, 2t-1]}$ for $\alpha \in \mathbb{F}_{2^m}^*$ and $\beta \in \mathbb{F}_{2^m}^*$
   - less than $10\,\mathrm{min}$ for $m = 13$
   - does not depend on the hardware target
   - stored in a hash table for fast look-up

Profiled attack scenario: offline phase

1. Precompute all possible $\left[\mathtt{HW}\left(\alpha^j \beta\right)\right]_{j \in [0, 2t-1]}$ for $\alpha \in \mathbb{F}_{2^m}^*$ and $\beta \in \mathbb{F}_{2^m}^*$
   - less than $10\,\mathrm{min}$ for $m = 13$
   - does not depend on the hardware target
   - stored in a hash table for fast look-up

2. Build two sets of $m + 1$ Hamming weight templates

- $\mathtt{HW}\left(\boldsymbol{g}^{-2}(\alpha_i)\right)$
  - evaluation $\boldsymbol{g}(\alpha_i)$
  - squaring $\boldsymbol{g}^2(\alpha_i)$
  - inversion $\boldsymbol{g}^{-2}(\alpha_i)$
- $\mathtt{HW}\left(\alpha_i^j \boldsymbol{g}^{-2}(\alpha_i)\right)$
  - multiplication by consecutive powers of α_i

Profiled attack scenario: online phase (1)

Recovering g

1. Use the templates to recover the sequence $\left[\mathtt{HW}\left(\alpha_i^j \boldsymbol{g}^{-2}(\alpha_i)\right)\right]_{j\in[0,2t-1]}$
2. Use the precomputed hash table to get the unique pairs $(\alpha_i, \boldsymbol{g}^{-2}(\alpha_i))$
3. Compute $\boldsymbol{g}(\alpha_i)$ from $\boldsymbol{g}^{-2}(\alpha_i)$
4. Interpolate $(\alpha_i, \boldsymbol{g}(\alpha_i))$ pairs to recover $\boldsymbol{g}$

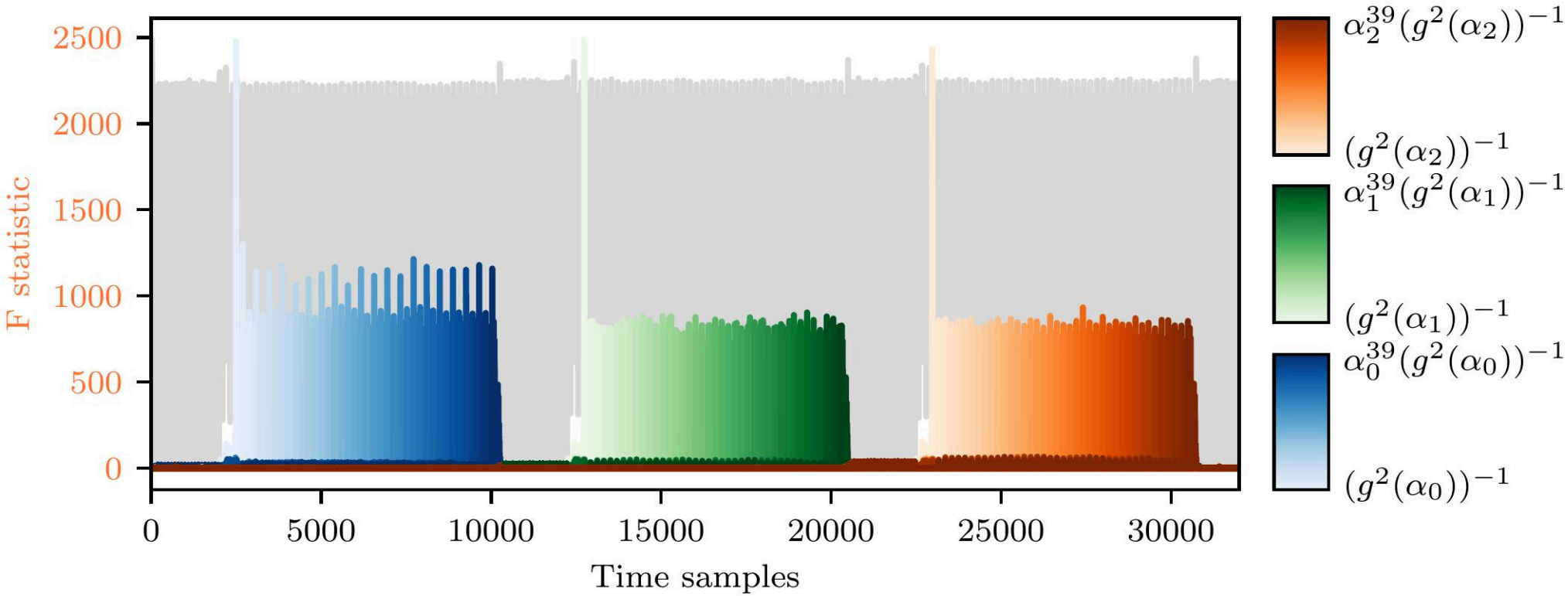
Profiled attack scenario: online phase (2)

Recovering $\mathcal{L}$

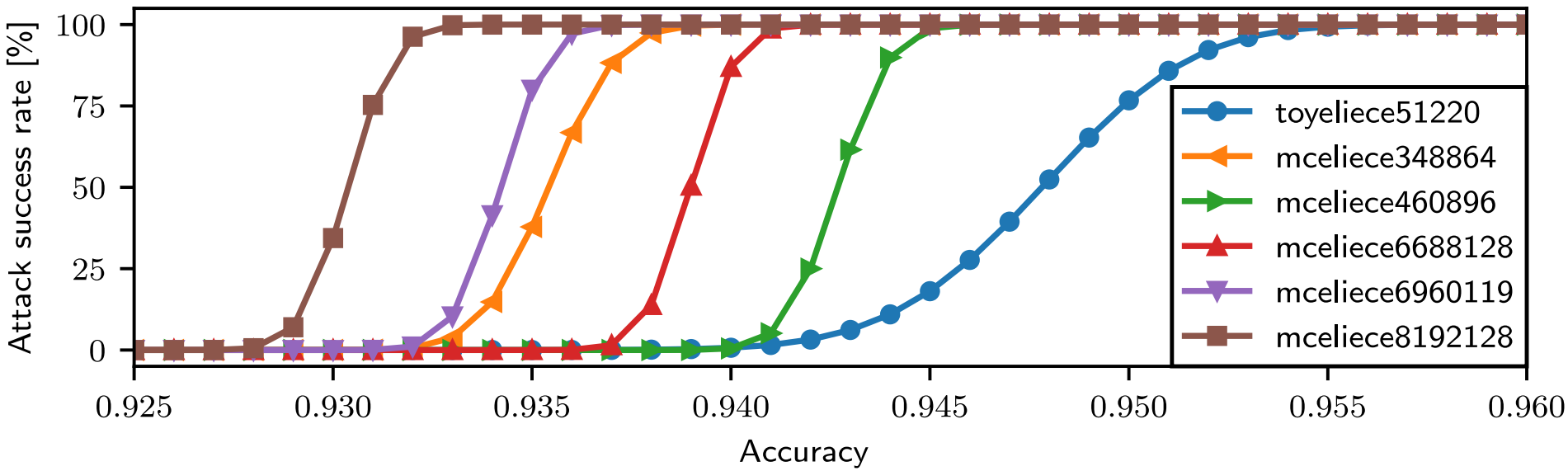
1. Build a pseudo public parity-check matrix $\boldsymbol{H}_{\boldsymbol{g}} = \left(\alpha^j \boldsymbol{g}^{-1}(\alpha)\right)_{j \in [0, t-1]}$
2. Map from $\mathbb{F}_{2^m}$ to $\mathbb{F}_2^m$
3. Recover the secret change-of-basis matrix from $\boldsymbol{H}_{\boldsymbol{g}}$ to $\boldsymbol{H}_{\text{pub}}$
   $\rightarrow$ matrix muliplication / inversion $\mathcal{O}(n^3)$
4. Map back from $\mathbb{F}_2^m$ to $\mathbb{F}_{2^m}$
5. $\mathcal{L} = \frac{H[1:]}{H[0:]} = \frac{\alpha_i \boldsymbol{g}(\alpha_i)}{\boldsymbol{g}(\alpha_i)}$

Experimental results: leakage assessment

$(m, n, t) = (9, 512, 20)$



Experimental results: classifier accuracy



Larger cryptographic parameters can be attacked with **less accurate** classifiers.

Can be lowered to 0.81 if Hamming weight errors are bounded by ± 1 **[20]**

---

[20] Nicolas Vallet, Pierre-Louis Cayrel, Brice Colombier, et al. "*Optimizing Key Recovery in Classic McEliece: Advanced Error Correction for Noisy Side-Channel Measurements*", IACR CiC (2025)

Experimental results: execution times

n	512	3488	4608	6688	8192
t	20	64	96	128	128
total time [s]	0.5	3.5	16	26	400

The **largest** cryptographic parameters are attacked in approximately 10 minutes.

Conclusion & perspectives

Conclusion on Classic McEliece

Classic McEliece was **not standardized** by NIST / FIPS.

🔍 Still under consideration at ISO

Ongoing works:

- Better understanding of the algebraic structure [21] for the distinguishability
  - Hamming weight $\rightarrow$ other $\mathbb{F}_2$-linear forms
  - dependency on the generator polynomials

    $\mathbb{F}_{2^{12}}$ $z^{12}+z^3+1$ $\qquad$ $\mathbb{F}_{2^{13}}$ $z^{13}+z^4+z^3+z+1$
- Other parts of the algorithms

---

[21] Michaël Bulois, Pierre-Louis Cayrel, Vlad-Florin Dragoi, Vincent Grosso "*Algebraic Key-Recovery Side-Channel Attack on Classic McEliece*", SAC (2025)

Perspectives

- Design countermeasures
  - implementation-level (shuffling)
  - algorithm-level (cryptosystem parameters)
- Apply the idea of changing the field (like $\mathbb{F}_2 \to \mathbb{N}$) to other cryptosystems
- Study other code-based cryptosystems (like HQC, signatures)

Perspectives

- Design countermeasures
  - implementation-level (shuffling)
  - algorithm-level (cryptosystem parameters)
- Apply the idea of changing the field (like $\mathbb{F}_2 \to \mathbb{N}$) to other cryptosystems
- Study other code-based cryptosystems (like HQC, signatures)

— Questions? —