Single Trace Side-Channel Attacks on the MPC-in-the-Head Framework

Authors: Julie Godard[1,2], Nicolas Aragon[1], Philippe Gaborit[1], Antoine Loiseau[2] and Julien Maillard[2]

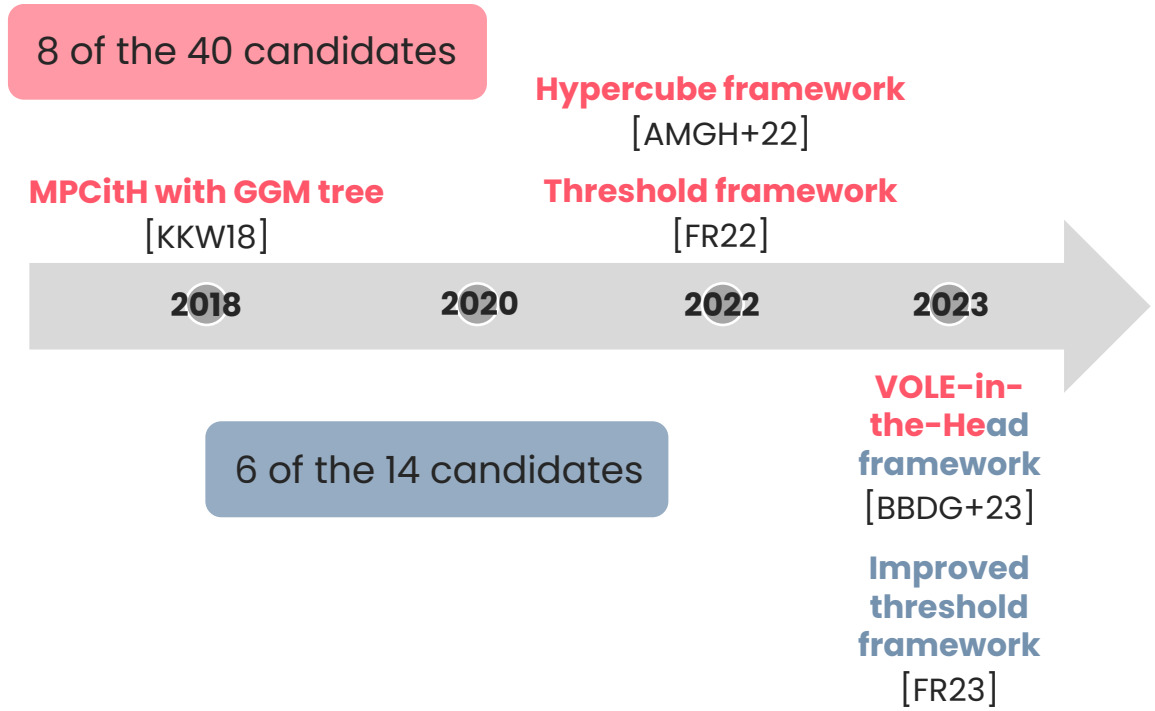
1: XLIM, University of Limoges, Limoges
2: Univ. Grenoble Alpes, CEA, Leti, F-38000 Grenoble, France

The second call of the NIST

- **Sep 2022**: Call for Additional Digital Signature Schemes
- **June 2023**: Deadline for submissions
- **July 2023**: Round 1 Additional Digital Signatures annonces (40 submissions accepted)
- **October 2024**: Round 2 Additional Digital Signatures annonces (14 candidates left)

For the **MPC-based candidates**:

- Aim → have **better performances than SPHINCS**.
- MPCitH framework → select parameters to achieve a **trade-off between efficiency and signature size**.

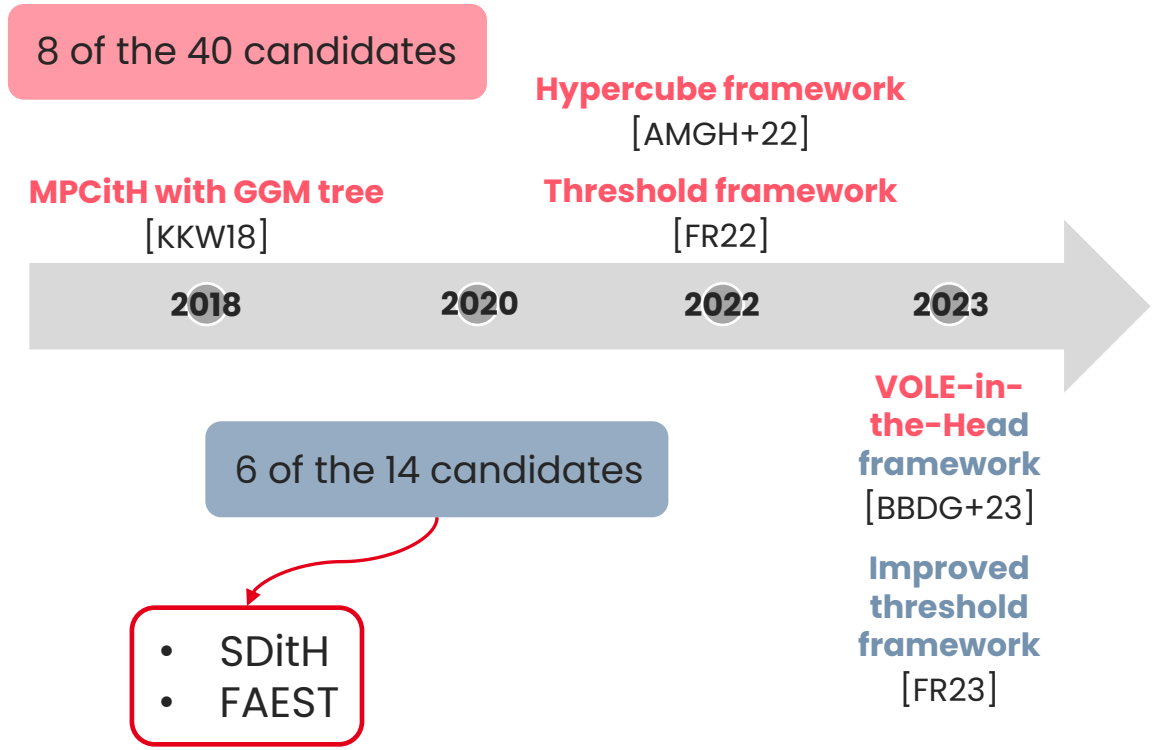


The second call of the NIST

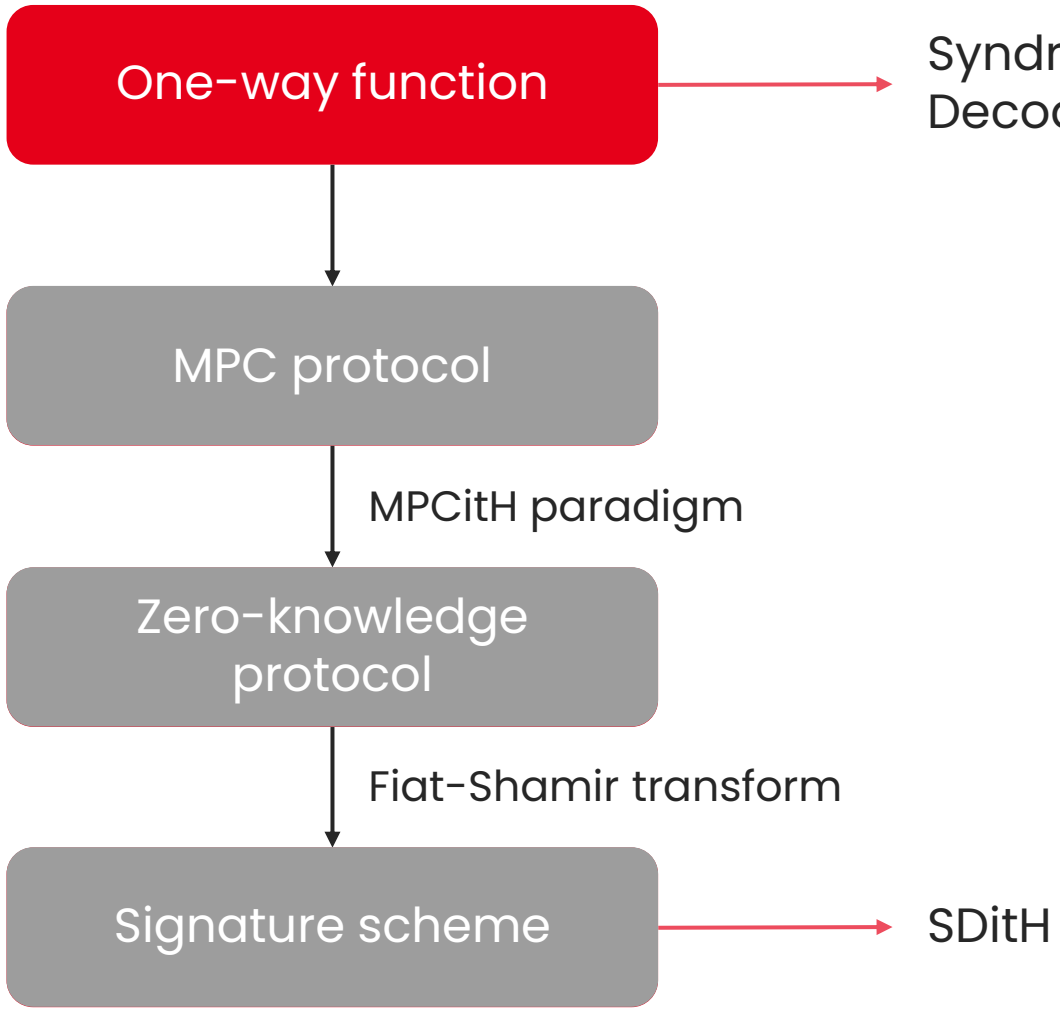
- **Sep 2022**: Call for Additional Digital Signature Schemes
- **June 2023**: Deadline for submissions
- **July 2023**: Round 1 Additional Digital Signatures annonces (40 submissions accepted)
- **October 2024**: Round 2 Additional Digital Signatures annonces (14 candidates left)

For the **MPC-based candidates**:

- Aim → have **better performances than SPHINCS**.
- MPCitH framework → select parameters to achieve a **trade-off between efficiency and signature size**.



Construction of a signature scheme



Construction of a signature scheme

One-way function → Syndrome Decoding

MPC protocol

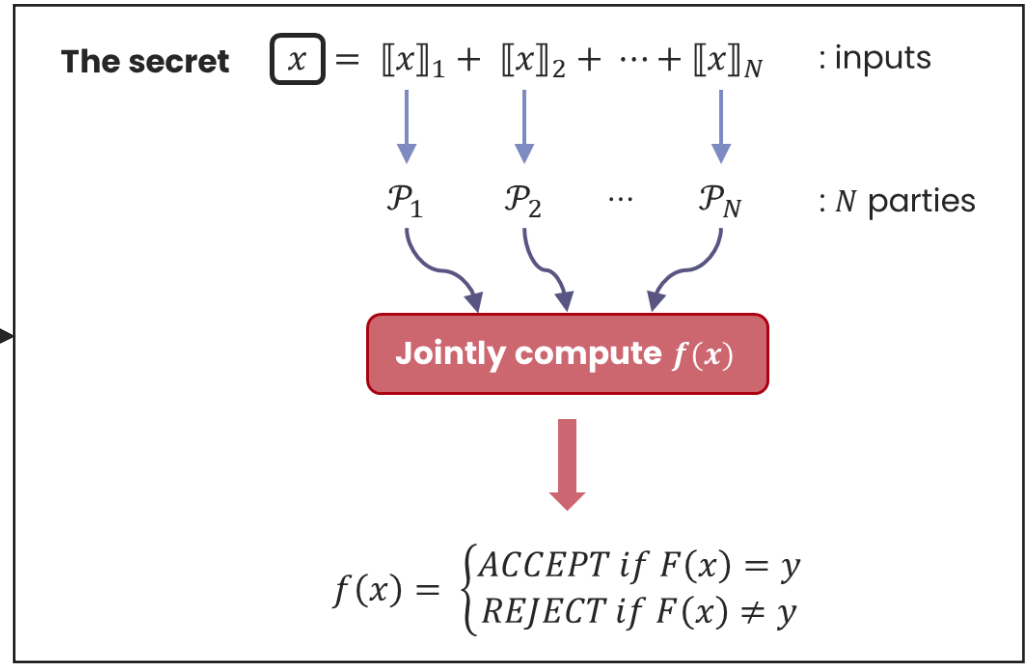
MPCitH paradigm

Zero-knowledge protocol

Fiat-Shamir transform

Signature scheme → SDitH

MPC protocol



Construction of a signature scheme

One-way function → Syndrome Decoding

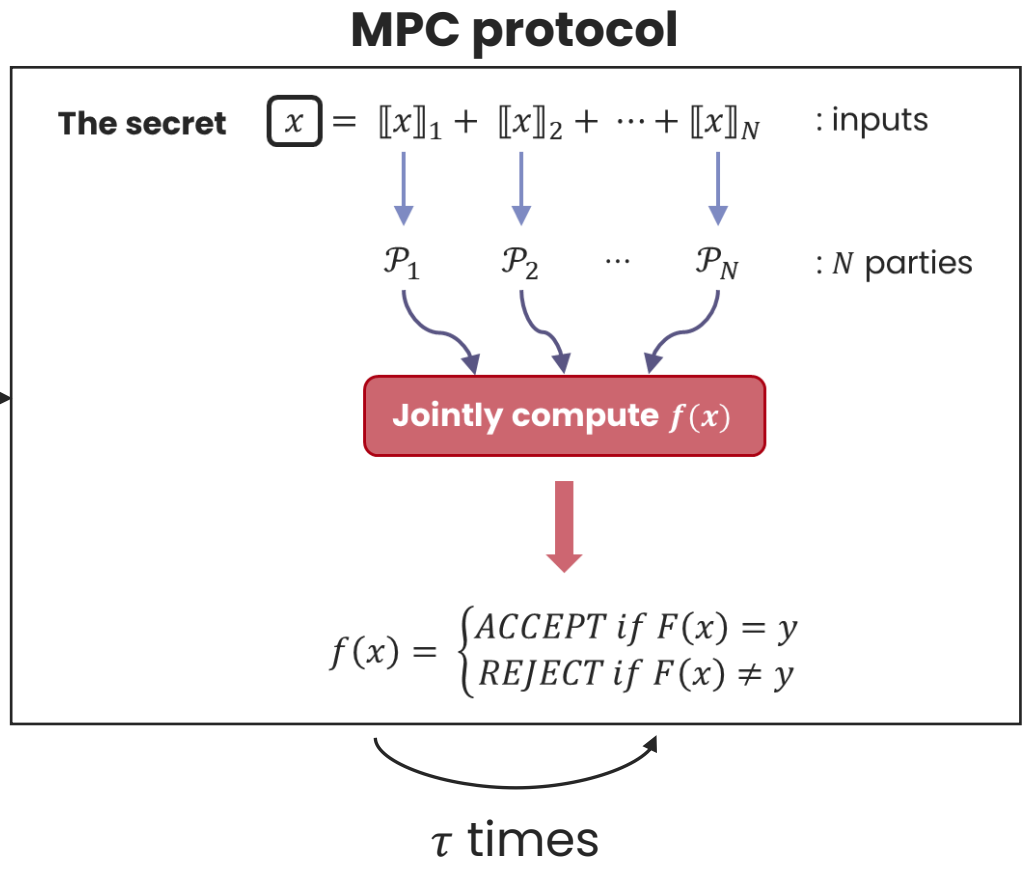
MPC protocol

MPCitH paradigm

Zero-knowledge protocol

Fiat-Shamir transform

Signature scheme → SDitH



Construction of a signature scheme

One-way function → Syndrome Decoding

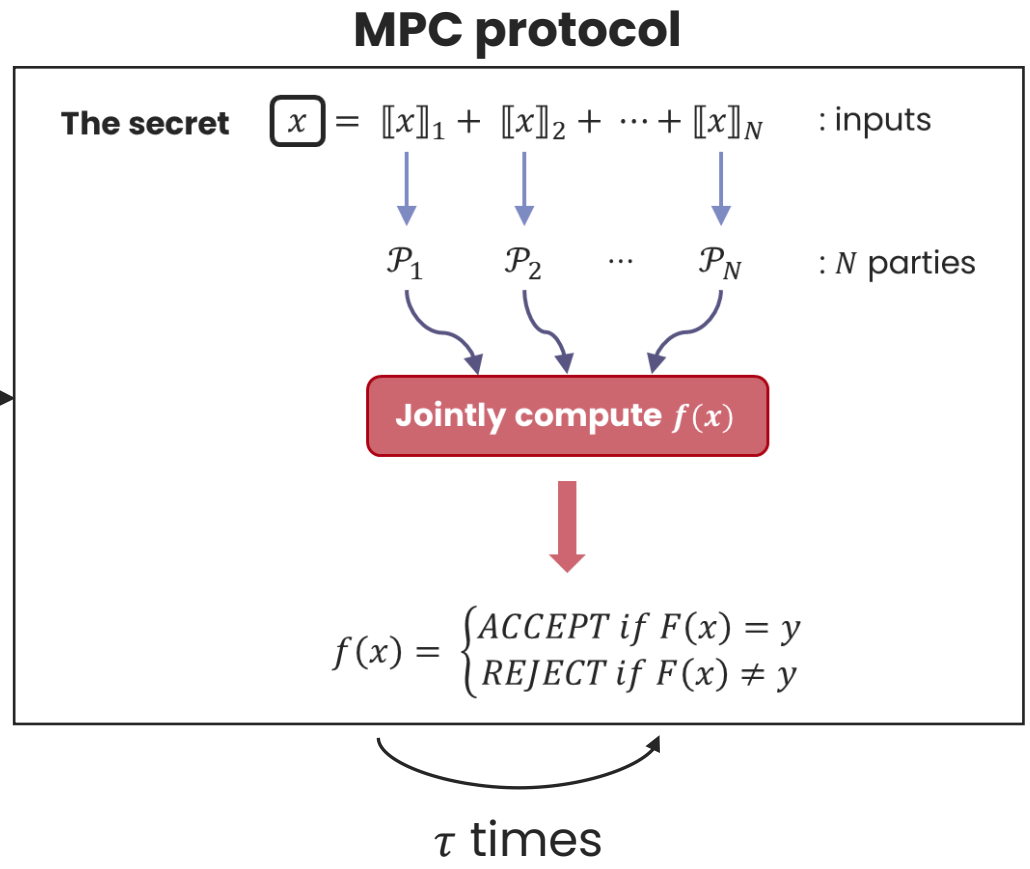
MPC protocol

MPCitH paradigm

Zero-knowledge protocol

Fiat-Shamir transform

Signature scheme → SDitH

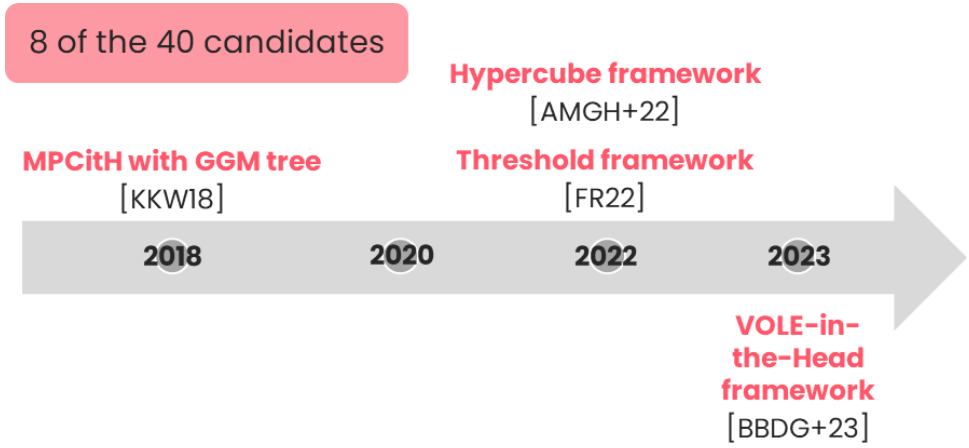


Threshold framework

A sharing $[\![s]\!] = ([\![s]\!]_1, \dots, [\![s]\!]_N)$ of a value $s \in \mathbb{F}_q$ is generated by sampling l random elements $(s_1, \dots, s_l) \leftarrow \mathbb{F}_q^l$, letting $s_0 \coloneqq s$, and defining the shares as follows:

$$\begin{cases} [\![s]\!]_1 = \sum_{i=0}^{l} s_i \cdot f_1^i \\ \quad \vdots \\ [\![s]\!]_N = \sum_{i=0}^{l} s_i \cdot f_N^i \end{cases}.$$

where $f_1, \dots, f_N$ denotes N fixed non-zero elements of the field $\mathbb{F}_q$.



Transformation of the SD problem

SD Problem → A solution to the problem is a vector $x \in \mathbb{F}^m$ such that $y = Hx$ and $wt(x) \leq w$.

In SDitH, we assume that:
- H is in standard form, meaning $H = (H'|I_{m-k})$,
- x in the form $x = (x_A|x_B)$.

→

- $y := Hx = H'x_A + x_B$
- $x_B = y - H'x_A$

→ Give x_A as input to the MPC protocol

Transformation of the SD problem

SD Problem → A solution to the problem is a vector $x \in \mathbb{F}^m$ such that $y = Hx$ and $wt(x) \leq w$.

non-linear

In SDitH, we assume that:
- H is in standard form, meaning $H = (H'|I_{m-k})$,
- x in the form $x = (x_A|x_B)$.

→
- $y := Hx = H'x_A + x_B$
- $x_B = y - H'x_A$

→ Give x_A as input to the MPC protocol

$$wt(x) \leq w \quad \Leftrightarrow \quad S \cdot Q = P \cdot F$$

$$S(f_i) = x_i, \quad i \in [1:m]$$

Transformation of the SD problem

SD Problem → A solution to the problem is a vector $x \in \mathbb{F}^m$ such that $y = Hx$ and $wt(x) \leq w$.

non-linear

In SDitH, we assume that:
- H is in standard form, meaning $H = (H'|I_{m-k})$,
- x in the form $x = (x_A|x_B)$.

→

- $y \coloneqq HVx$
- $s \coloneqq (s_A|s_B) = Vx$
- $s_B = y - H's_A$

→ Give s_A instead of x_A **as input to the MPC protocol**

$$wt(x) \leq w \quad \Leftrightarrow \quad S \cdot Q = P \cdot F$$

$$S(f_i) = x_i, \quad i \in [1:m]$$

Signature algorithm

Signature:

1. Generate random sharing $[\![x_A]\!]$, $[\![P]\!]$, $[\![Q]\!]$, $[\![a]\!]$, $[\![b]\!]$, $[\![c]\!]$
2. Commit the parties' shares:

$$[\![x_A]\!]_i, [\![P]\!]_i, [\![Q]\!]_i, [\![a]\!]_i, [\![b]\!]_i, [\![c]\!]_i \xrightarrow{\text{Commit}} \mathsf{com}_i$$

3. Derive the first challenge (randomness of MPC protocol):

$$\mathsf{com}_1, \ldots, \mathsf{com}_N \xrightarrow{\text{Hash}} h_1 \to r, \varepsilon$$

4. Simulate the MPC protocol: τ times

$$[\![x_A]\!], [\![P]\!], [\![Q]\!], [\![a]\!], [\![b]\!], [\![c]\!], r, \varepsilon \xrightarrow{\text{MPC}} [\![\alpha]\!], [\![\beta]\!], [\![v]\!]$$

5. Derive the second challenge (index of non-opened party):

$$h_1, [\![\alpha]\!], [\![\beta]\!], [\![v]\!] \xrightarrow{\text{Hash}} h_2 \to I$$

6. Build the signature from

$$h_1,\ h_2,\ \{[\![x_A]\!]_i, [\![P]\!]_i, [\![Q]\!]_i, [\![a]\!]_i, [\![b]\!]_i, [\![c]\!]_i\}_{i \in I},\ \{\mathsf{com}_i, [\![\alpha]\!]_i, [\![\beta]\!]_i, [\![v]\!]_i\}_{i \notin I}$$

Protocol 4: SDitH signature scheme [AFG+23]

$$\text{MPC_inputs} := [\![s_A]\!], [\![P]\!], [\![Q]\!], [\![a]\!], [\![b]\!], [\![c]\!], r, \epsilon$$

$$[\![\mathrm{s_A}]\!]_i := \mathrm{s_A} + \sum_{j=1}^{l} f_i^{\,j} \cdot \mathrm{s_A_coef}_j$$

τ times

Signature algorithm

Signature:

1. Generate random sharing $[\![x_A]\!]$, $[\![P]\!]$, $[\![Q]\!]$, $[\![a]\!]$, $[\![b]\!]$, $[\![c]\!]$
2. Commit the parties' shares:

$$[\![x_A]\!]_i, [\![P]\!]_i, [\![Q]\!]_i, [\![a]\!]_i, [\![b]\!]_i, [\![c]\!]_i \xrightarrow{\text{Commit}} \mathsf{com}_i$$

3. Derive the first challenge (randomness of MPC protocol):

$$\mathsf{com}_1, \ldots, \mathsf{com}_N \xrightarrow{\text{Hash}} h_1 \to r, \varepsilon$$

4. Simulate the MPC protocol: τ times

$$[\![x_A]\!], [\![P]\!], [\![Q]\!], [\![a]\!], [\![b]\!], [\![c]\!], r, \varepsilon \xrightarrow{\text{MPC}} [\![\alpha]\!], [\![\beta]\!], [\![v]\!]$$

5. Derive the second challenge (index of non-opened party):

$$h_1, [\![\alpha]\!], [\![\beta]\!], [\![v]\!] \xrightarrow{\text{Hash}} h_2 \to I$$

6. Build the signature from

$$h_1,\ h_2,\ \{[\![x_A]\!]_i, [\![P]\!]_i, [\![Q]\!]_i, [\![a]\!]_i, [\![b]\!]_i, [\![c]\!]_i\}_{i\in I},\ \{\mathsf{com}_i, [\![\alpha]\!]_i, [\![\beta]\!]_i, [\![v]\!]_i\}_{i\notin I}$$

Protocol 4: SDitH signature scheme [AFG+23]

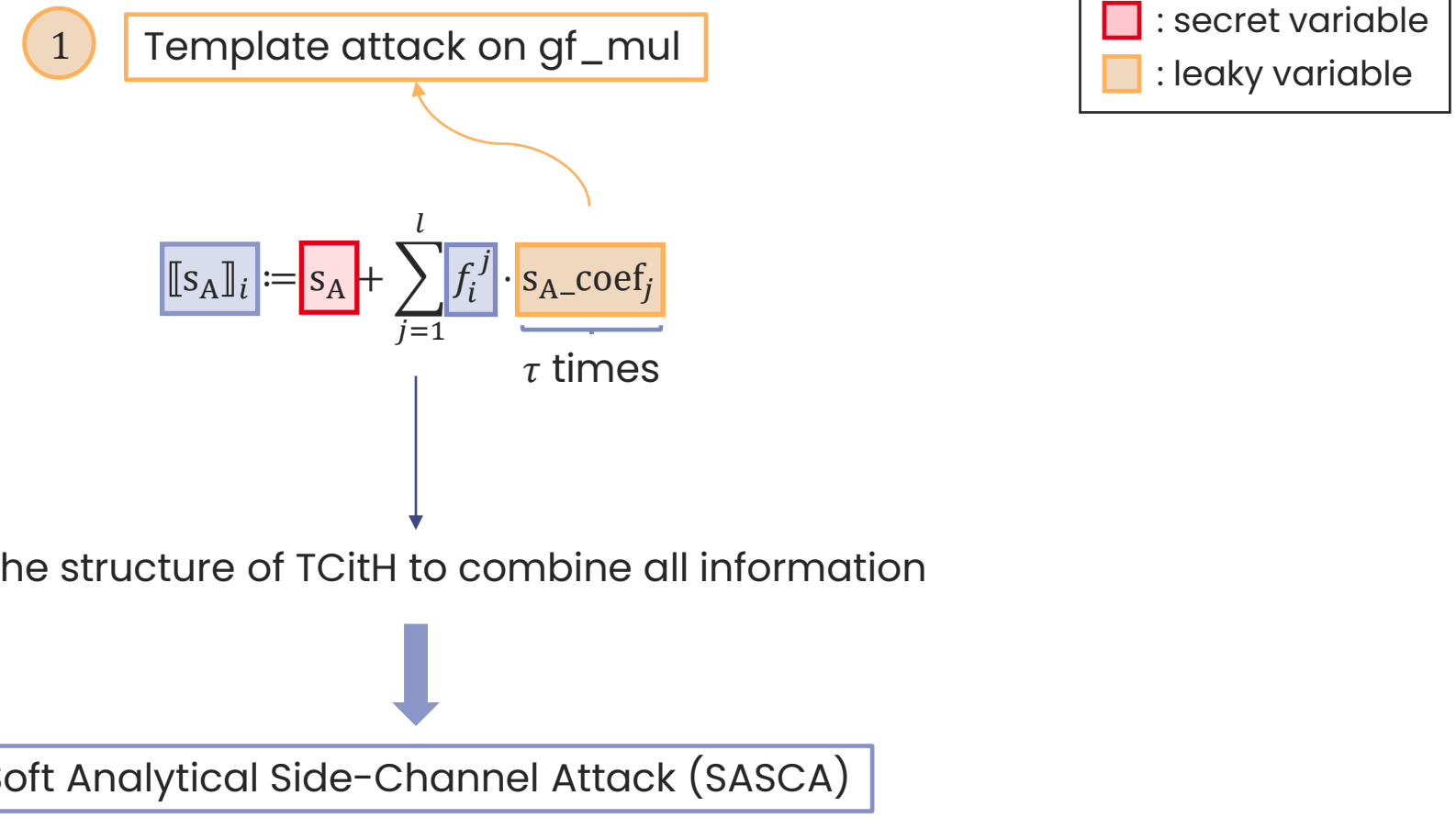
$$\text{MPC_inputs} := [\![s_A]\!], [\![P]\!], [\![Q]\!], [\![a]\!], [\![b]\!], [\![c]\!], r, \epsilon$$

$$[\![\mathrm{s_A}]\!]_i := \mathrm{s_A} + \sum_{j=1}^{l} f_i^j \cdot \mathrm{s_A_coef}_j$$

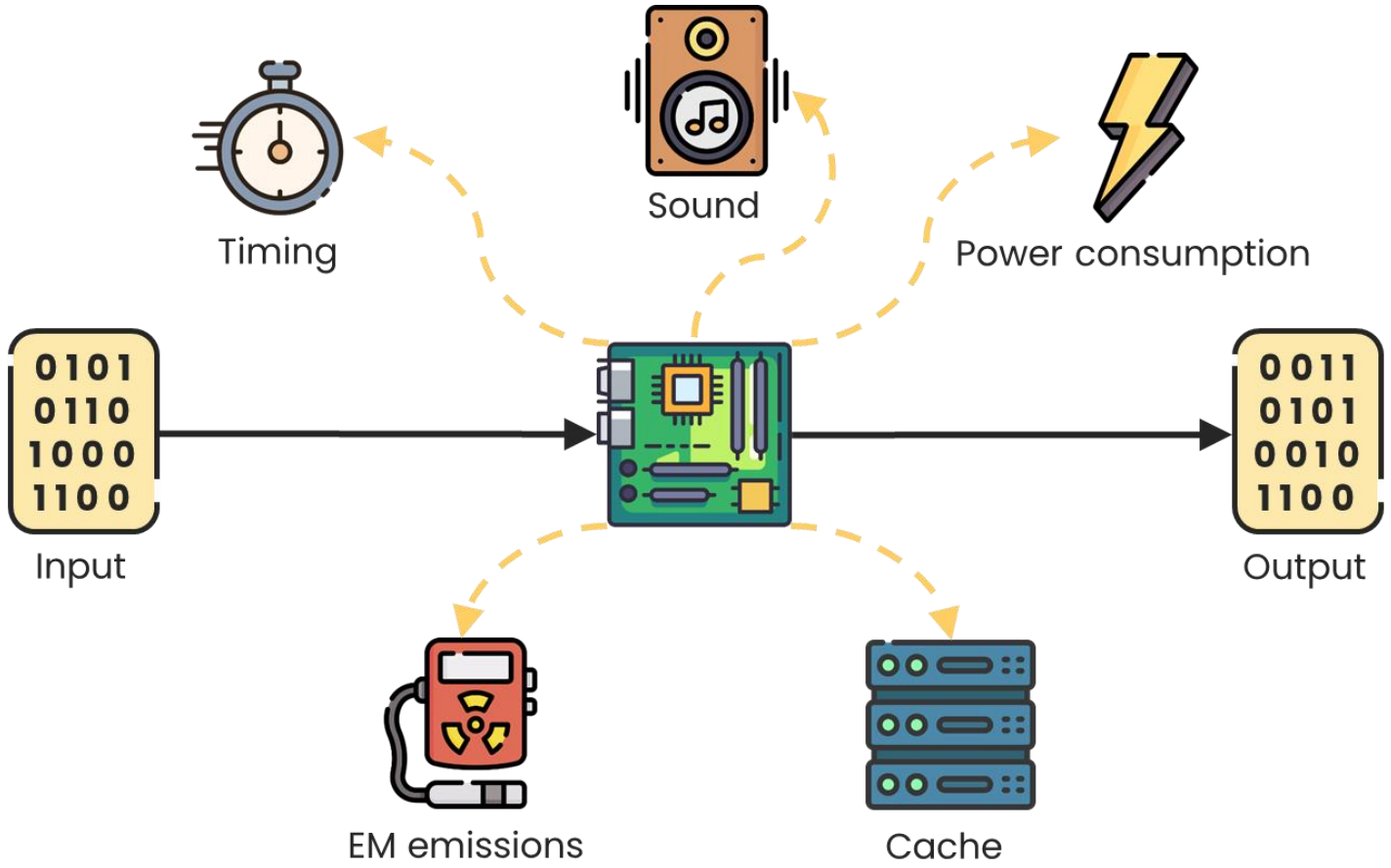
τ times

: public variable
: secret variable
: leaky variable

Construction of the attack



Side-channel attack

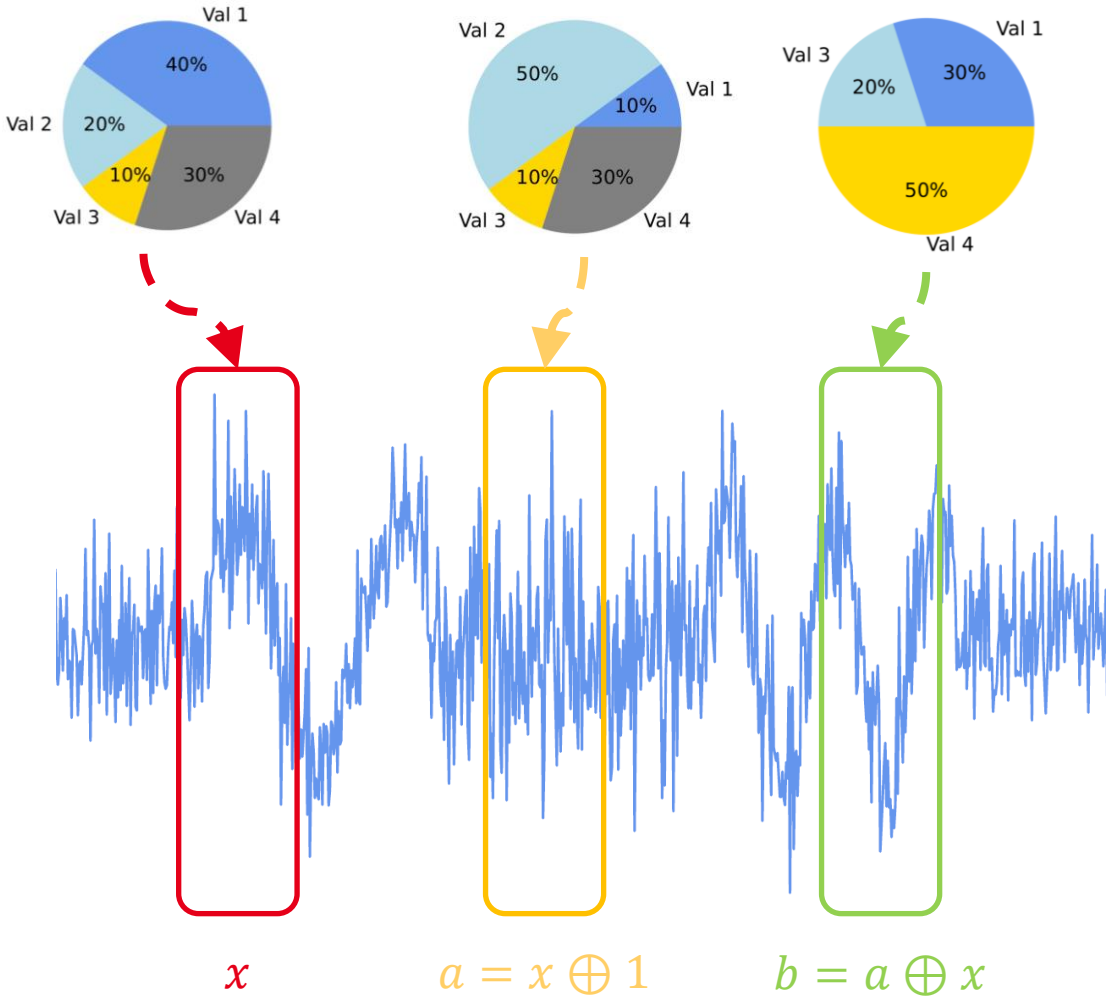


Side-channel attacks

- enabled by **leakage of information** from a physical cryptosystem
- be achieved by **measuring/analyzing** various **physical parameters**

Template attack on gf_mul on the STM32F407.

Soft Analytical Side-Channel Attack

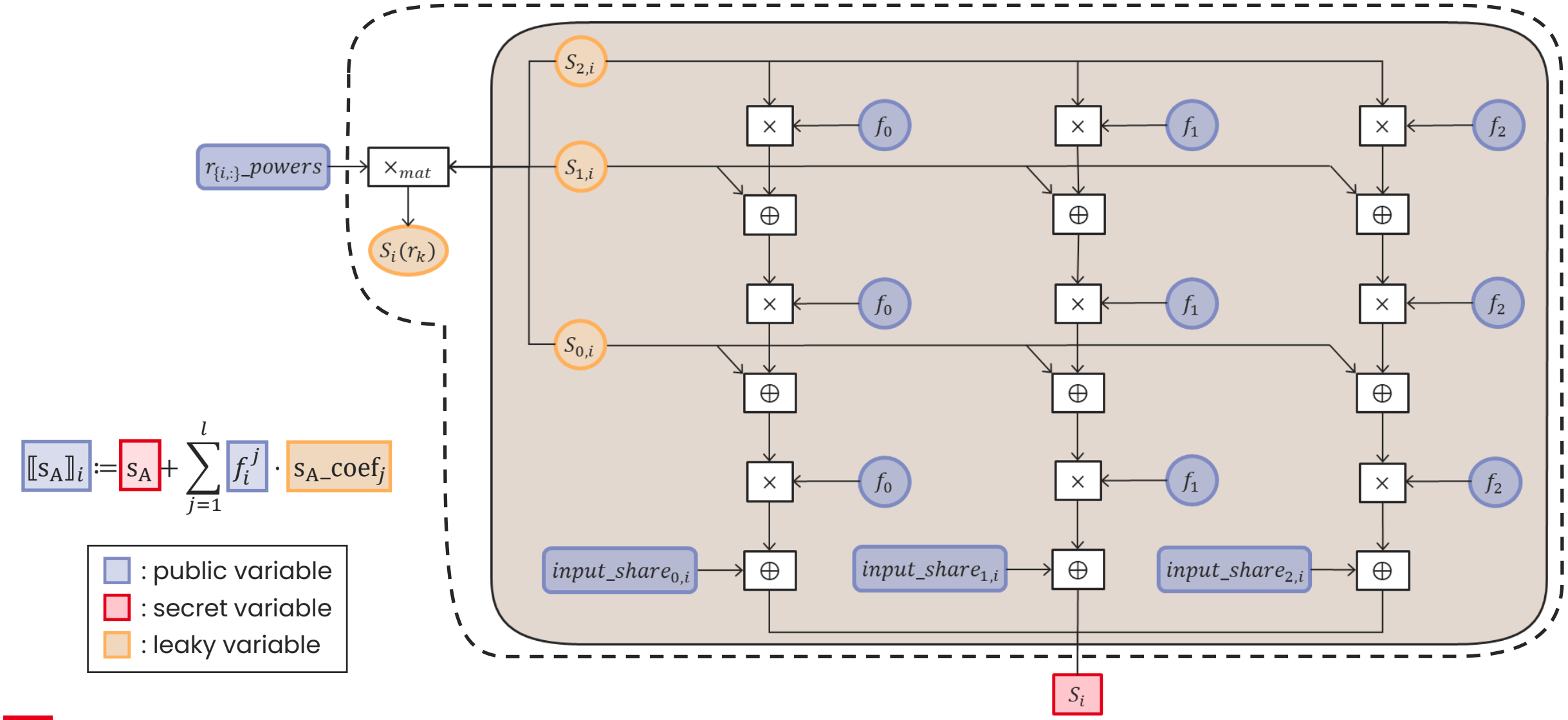


SASCA

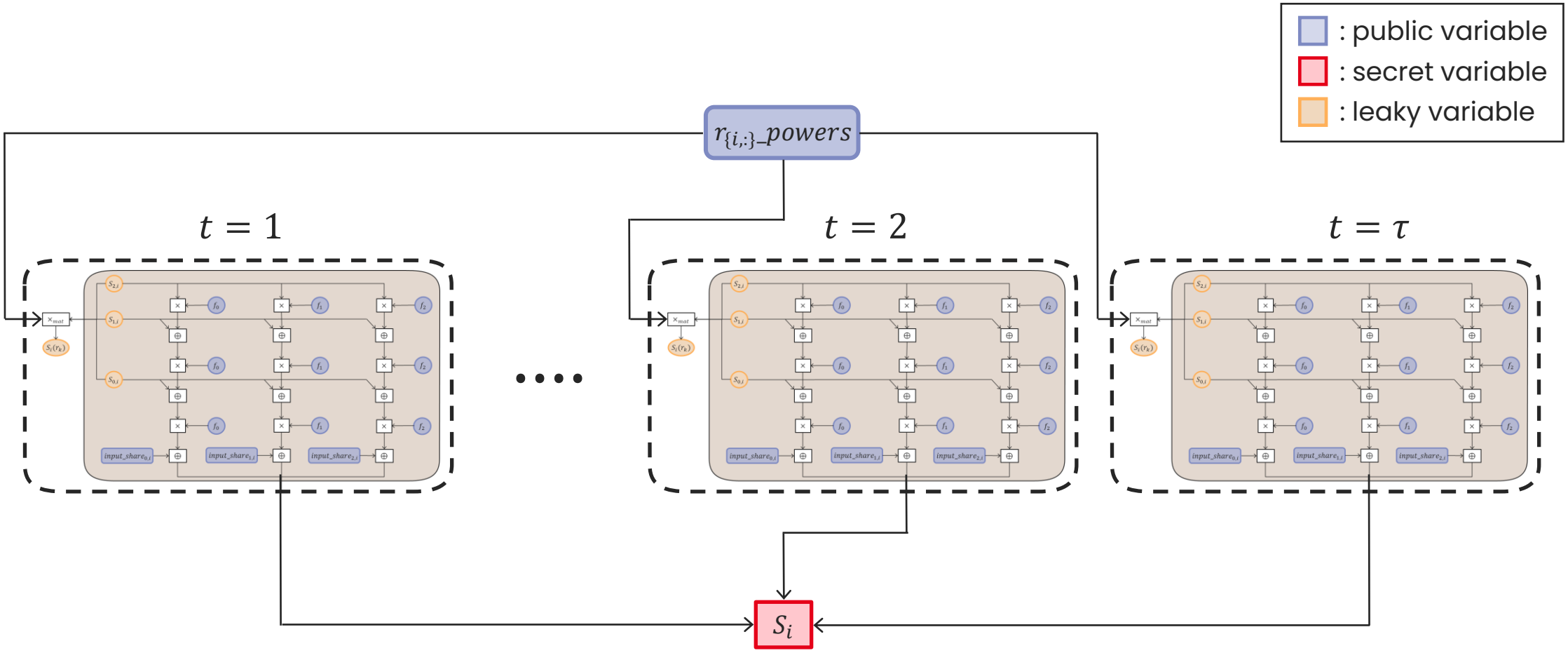
1. Focuses on identifying operations dependent on secret data and **analyzing their intermediate values**
2. They are then linked to the classifier outputs to **obtain their probability distributions.**

→ Enhance our prediction of the secret by modelling a maximum a posteriori problem which takes into account the gathered distributions → we can take advantage of the **Belief Propagation** (BP) algorithm on a **factor graph**.

SDitH Polynomial Evaluation Graph

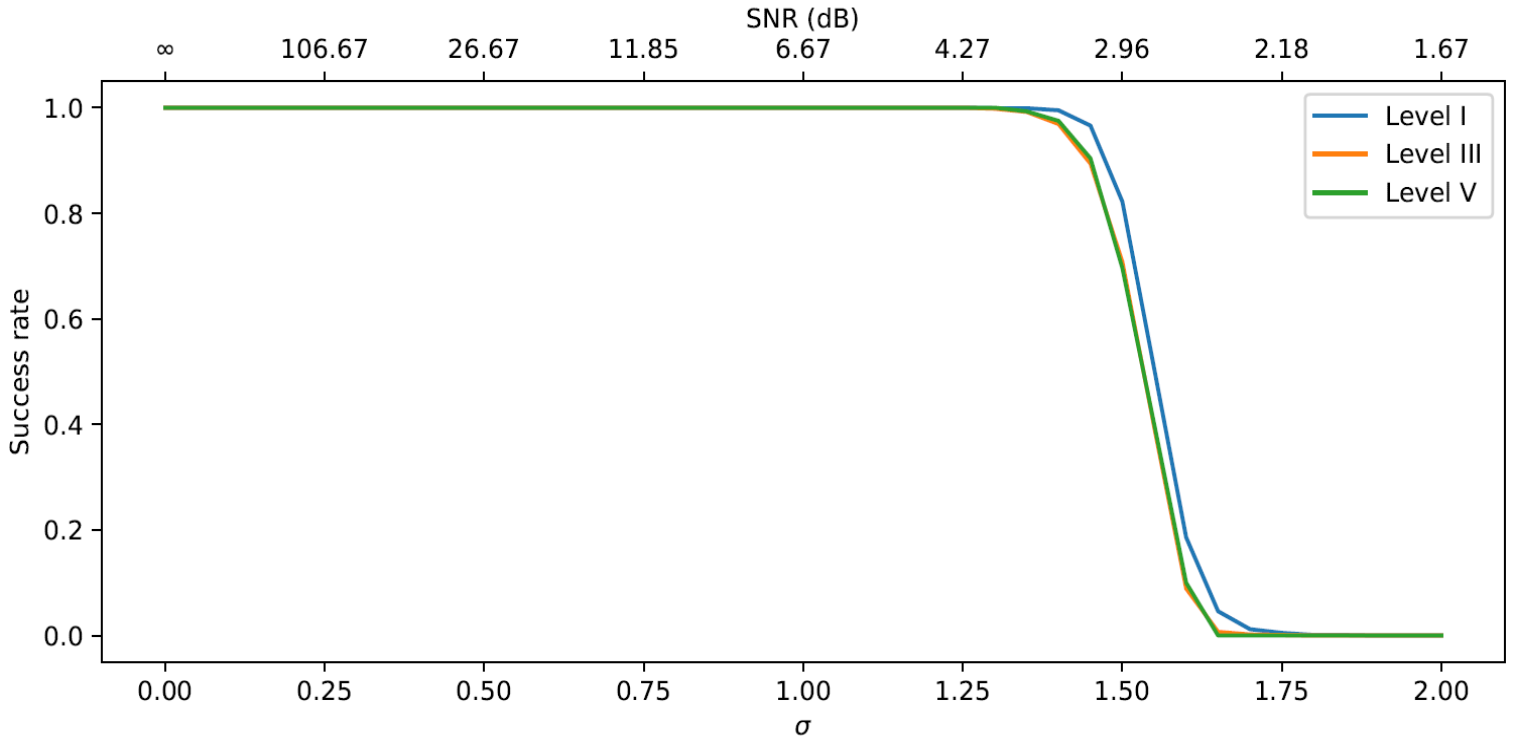


SDitH Polynomial Evaluation Graph



Simulation of the attack

- We use a **noisy HW model** → give us a generic analysis of our attack.
- We start by initializing the marginal probabilities of the variables involves in gf_mul computation from a HW leakage model with a Gaussian noise σ.



Success rate: all coefficients except those multiplied by the first row of the r_powers matrix → add a time complexity of 70 for one coefficient

Our attack used the reference implementation of SDitH on a STM32F407 (EM measurement).

Practical attack

1. We build **three templates on the gf_mul function** that targets the HW of:
   - its inputs - operands 0 and 1
   - its output

	Hamming weight template accuracy
Operand 0	0.4023
Operand 1	0.5142
Output	0.9965

Practical attack

① We build **three templates on the gf_mul function** that targets the HW of:

- its inputs - operands 0 and 1
- its output

	Hamming weight template accuracy
Operand 0	0.4023
Operand 1	0.5142
Output	0.9965

② We derived three attacks

- leakage on the operand 0 and output;
- leakage on the operand 0;
- leakage on the output.

We look at how the coefficients of s_A **converge depending on the number of epochs** of the BP algorithm.

Practical attack

① We build **three templates on the gf_mul function** that targets the HW of:

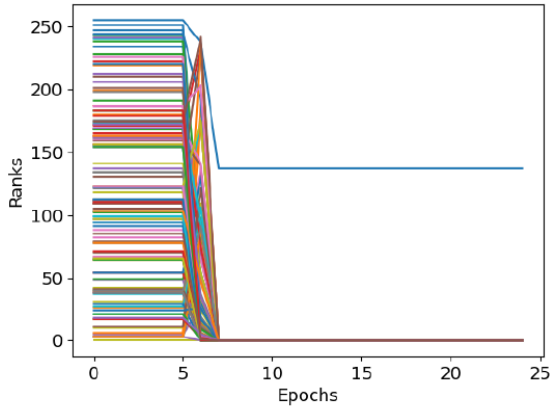
- its inputs - operands 0 and 1
- its output

	Hamming weight template accuracy
Operand 0	0.4023
Operand 1	0.5142
Output	0.9965

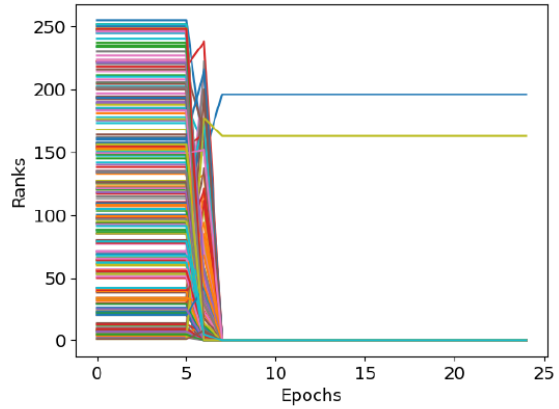
② We derived three attacks

- **leakage on the operand 0 and output;**
- leakage on the operand 0;
- leakage on the output.

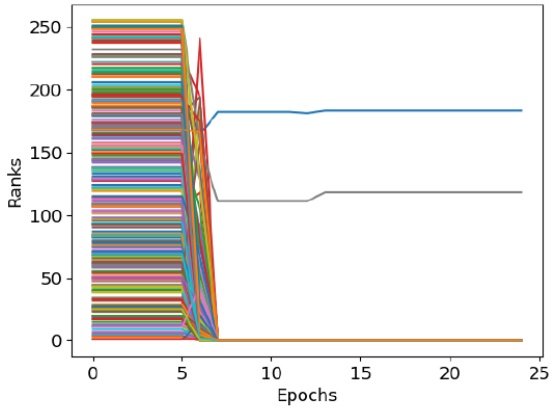
We look at how the coefficients of s_A **converge depending on the number of epochs** of the BP algorithm.



(a) Level I

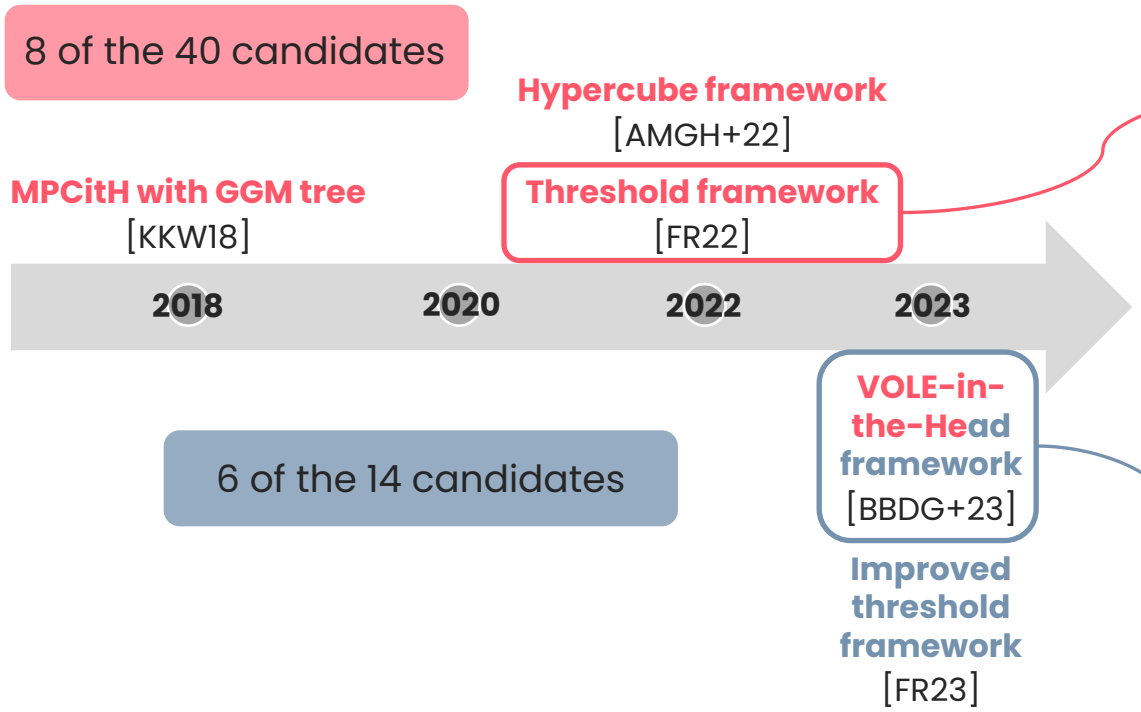


(b) Level III



(c) Level V

Second-round frameworks and new attack



[GAG+24] Single Trace Side-Channel Attack on the MPC-in-the-Head Framework

- Not applicable as is to second-round frameworks

Actual work. Non-supervised SASCA on the VOLEitH framework

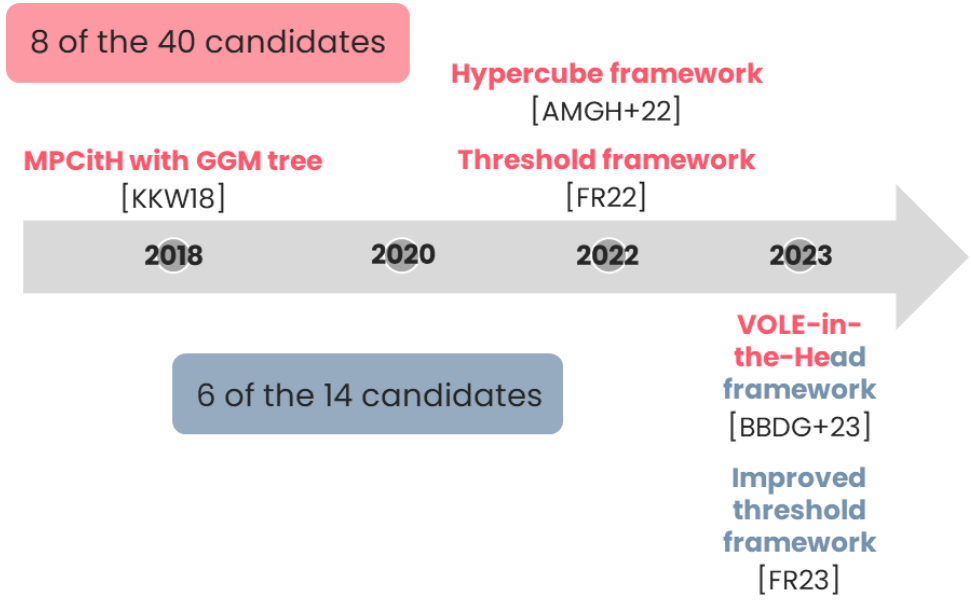
- Use cases of FAEST and SDitH

VOLE correlations

A **VOLE** (Vector Oblivious Linear Evaluation) **correlation** of length l is a multiparty gadget, where :

- the prover, learns random bits $\boldsymbol{u} \in \mathbb{F}_p^l$, and random VOLE tags $\boldsymbol{v} \in \mathbb{F}_{p^\kappa}^l$;
- the verifier learns a random global key $\Delta \in \mathbb{F}_{p^\kappa}$ and VOLE keys $\boldsymbol{q} = \boldsymbol{u} \cdot \Delta + \boldsymbol{v} \in \mathbb{F}_{p^\kappa}^l$.

The VOLE correlations can be used to build a **proof of knowledge** of the random bits u_i.



VOLEitH specificities

Witness embedding

Given a VOLE correlation $q = u\Delta + v$ for a random $u \in \mathbb{F}_2$, **it is possible to embed arbitrary value** $w \in \mathbb{F}_2$. We proceed as follows:

- the prover computes $t = w \oplus u$ and sends t to the verifier. Since u is uniform random and unknown to the verifier, t does not leak any information about w ;
- the verifier then computes

$$q_w = q + t\Delta = w\Delta + v.$$

Large field embedding

The prover has τ small, independent VOLE correlations:

- the prover has $(\boldsymbol{u}_i, \boldsymbol{V}_i)$ for $i = 0, \dots, \tau - 1$
- the verifier has $(\Delta_i, \boldsymbol{Q}_i)$ such that

$$\boldsymbol{Q}_i = \boldsymbol{V}_i + (\delta_0 \cdot \boldsymbol{u}_i \cdots \delta_{\kappa-1} \cdot \boldsymbol{u}_i)$$

where $\delta_0, \dots, \delta_{\kappa-1}$ is the bit decomposition of Δ.

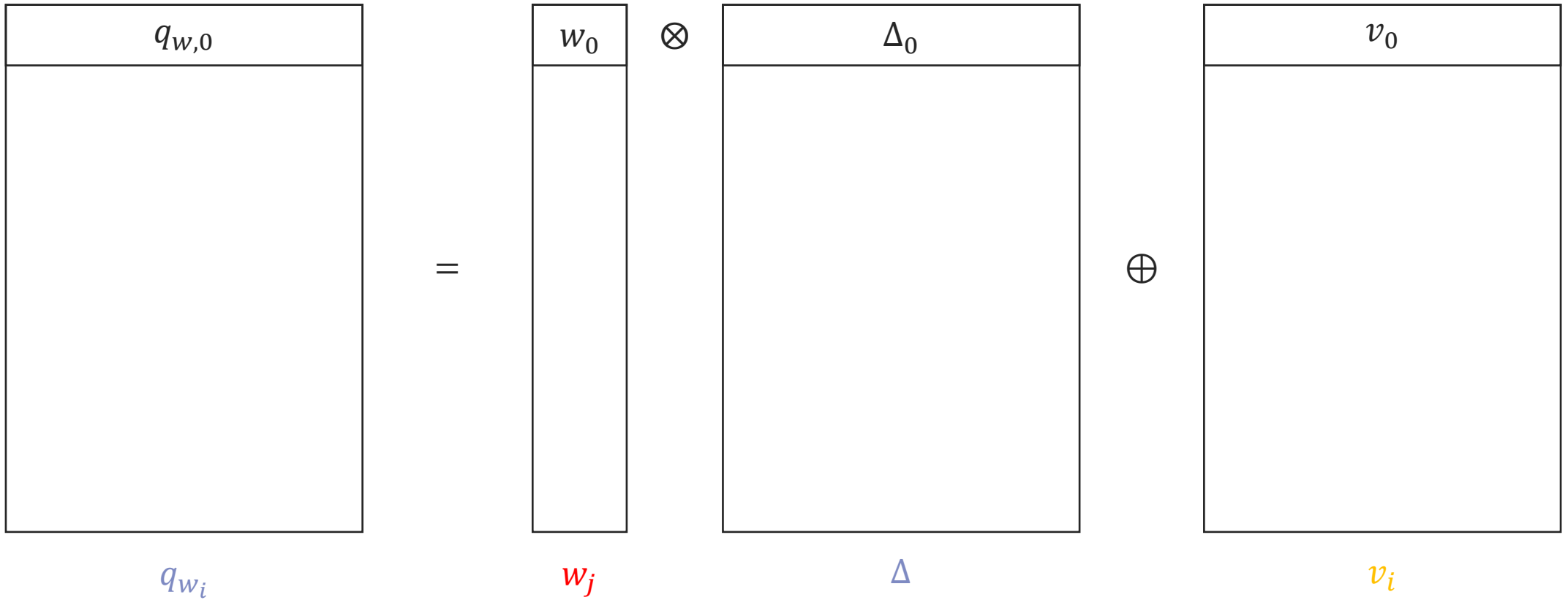
The parties can simply **concatenate the τ VOLE instances**, by forming the $l \times \lambda$ matrices

$$\boldsymbol{V} = (\boldsymbol{V}_0 \cdots \boldsymbol{V}_{\tau-1}), \qquad \boldsymbol{Q} = (\boldsymbol{Q}_0 \cdots \boldsymbol{Q}_{\tau-1}).$$

The prover and verifier now possess **their respective parts for the VOLE correlation** $q_w = w\Delta + v$.

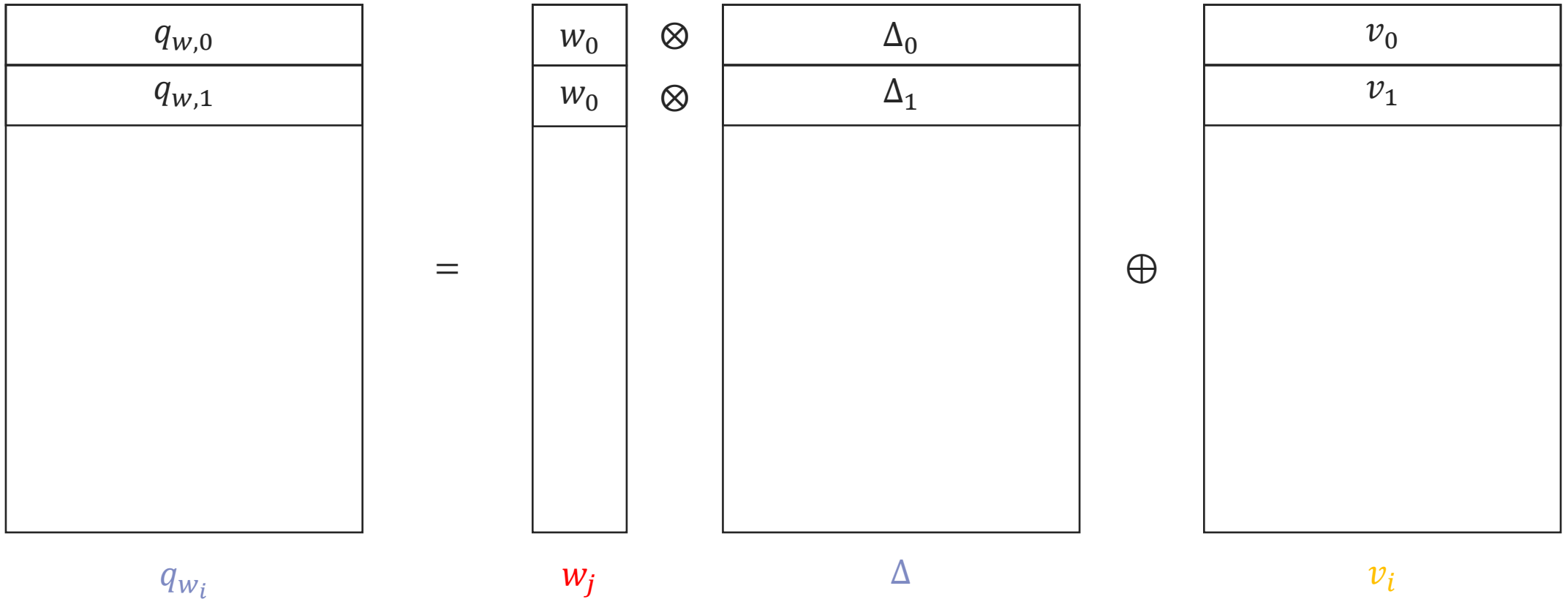
VOLEitH Graph

The **prover and verifier possess their respective parts for the VOLE correlation** $q_w = w\Delta + v$.



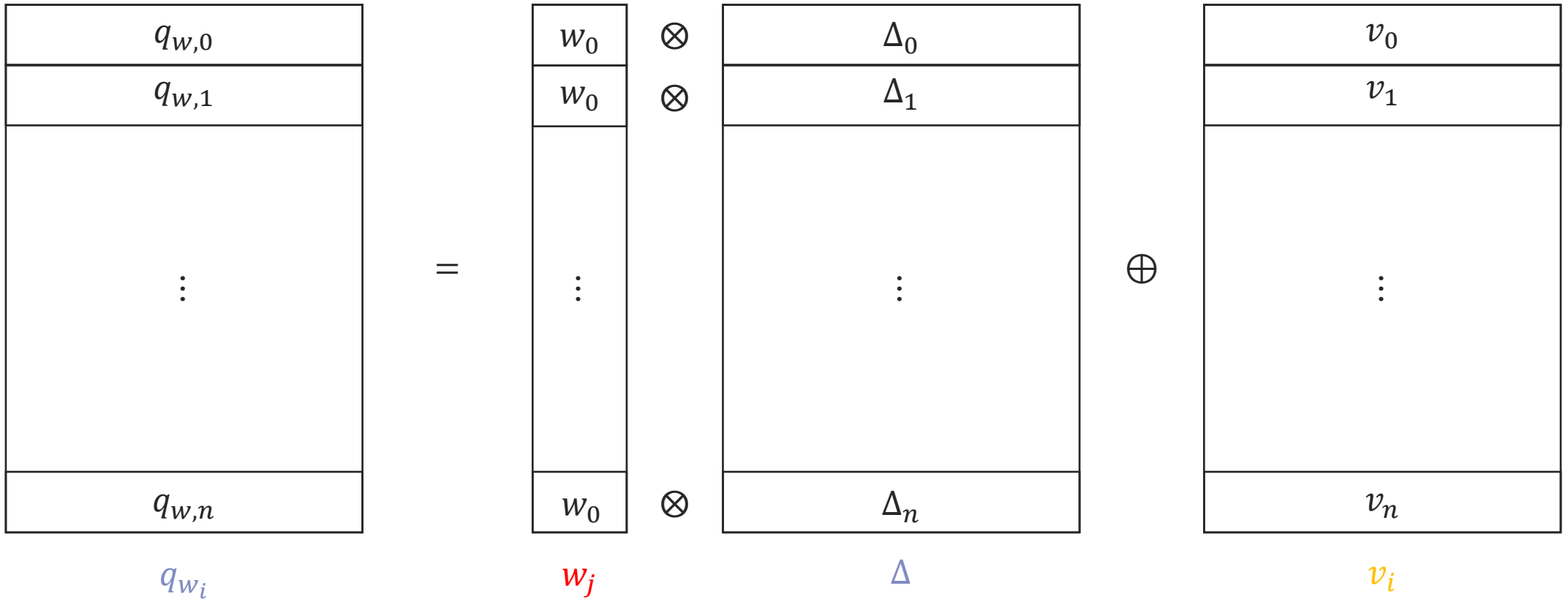
VOLEitH Graph

The **prover and verifier possess their respective parts for the VOLE correlation** $q_w = w\Delta + v$.



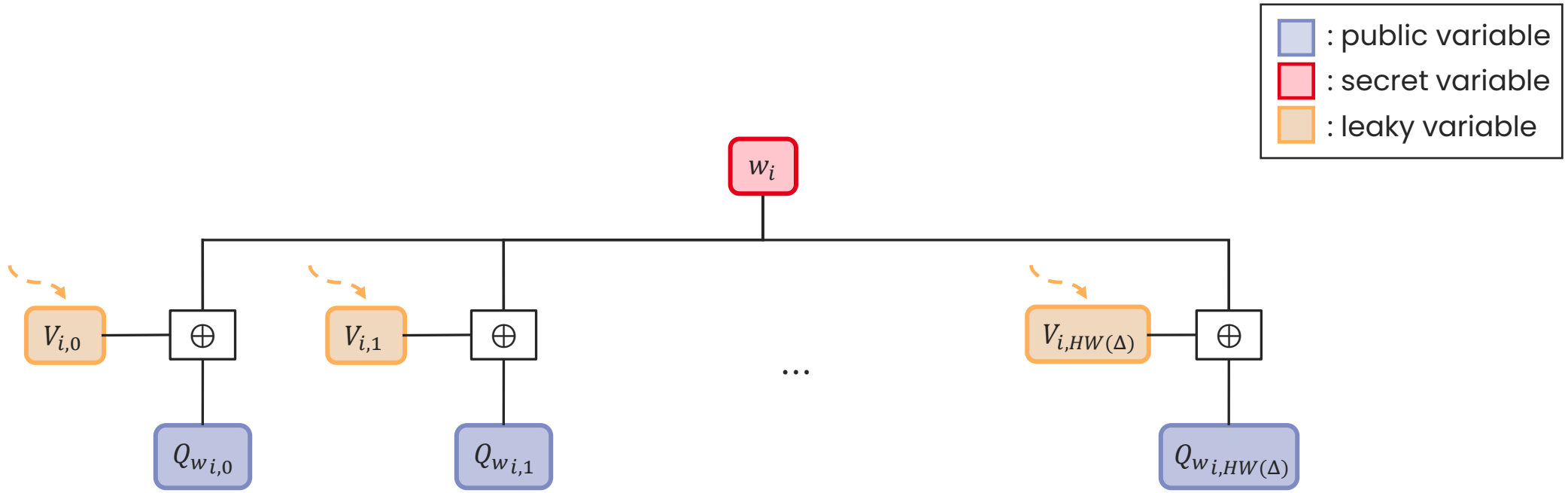
VOLEitH Graph

The **prover and verifier possess their respective parts for the VOLE correlation** $q_w = w\Delta + v$.



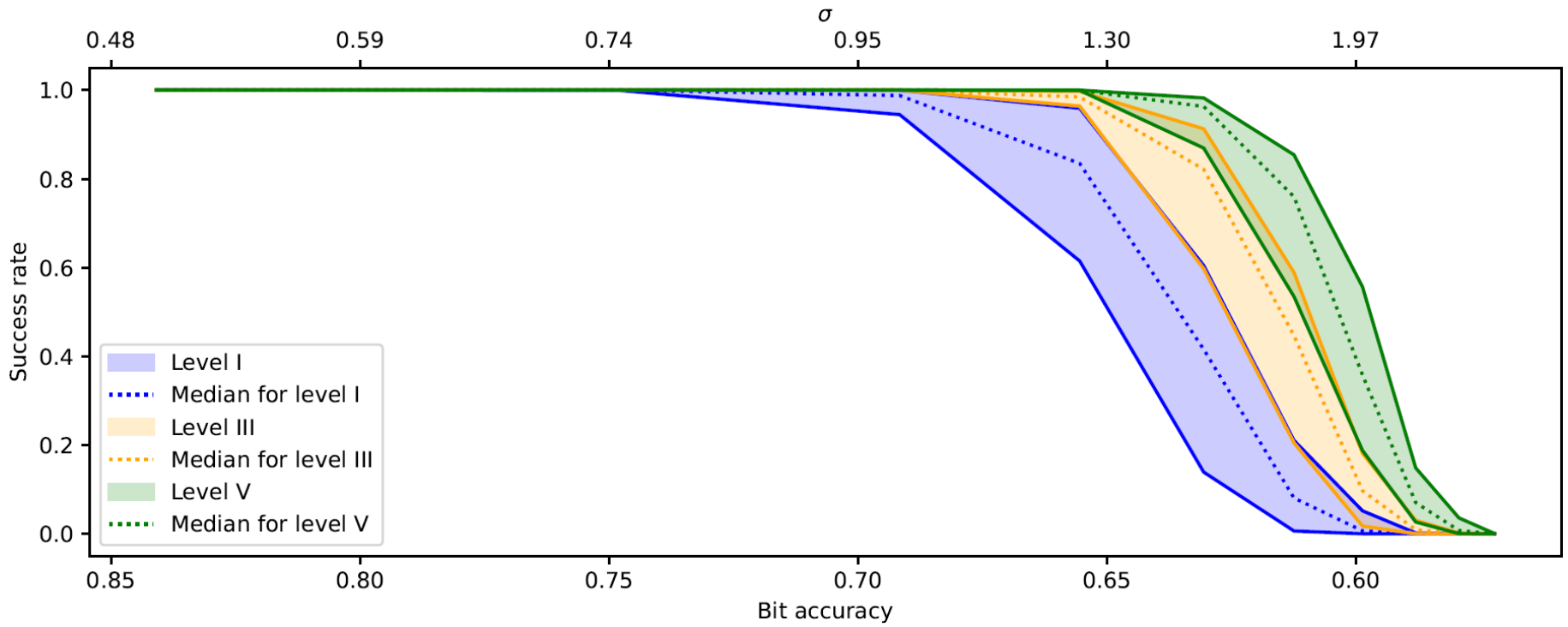
VOLEitH Graph – Bit level

The **prover and verifier possess their respective parts for the VOLE correlation** $q_w = w\Delta + v$.



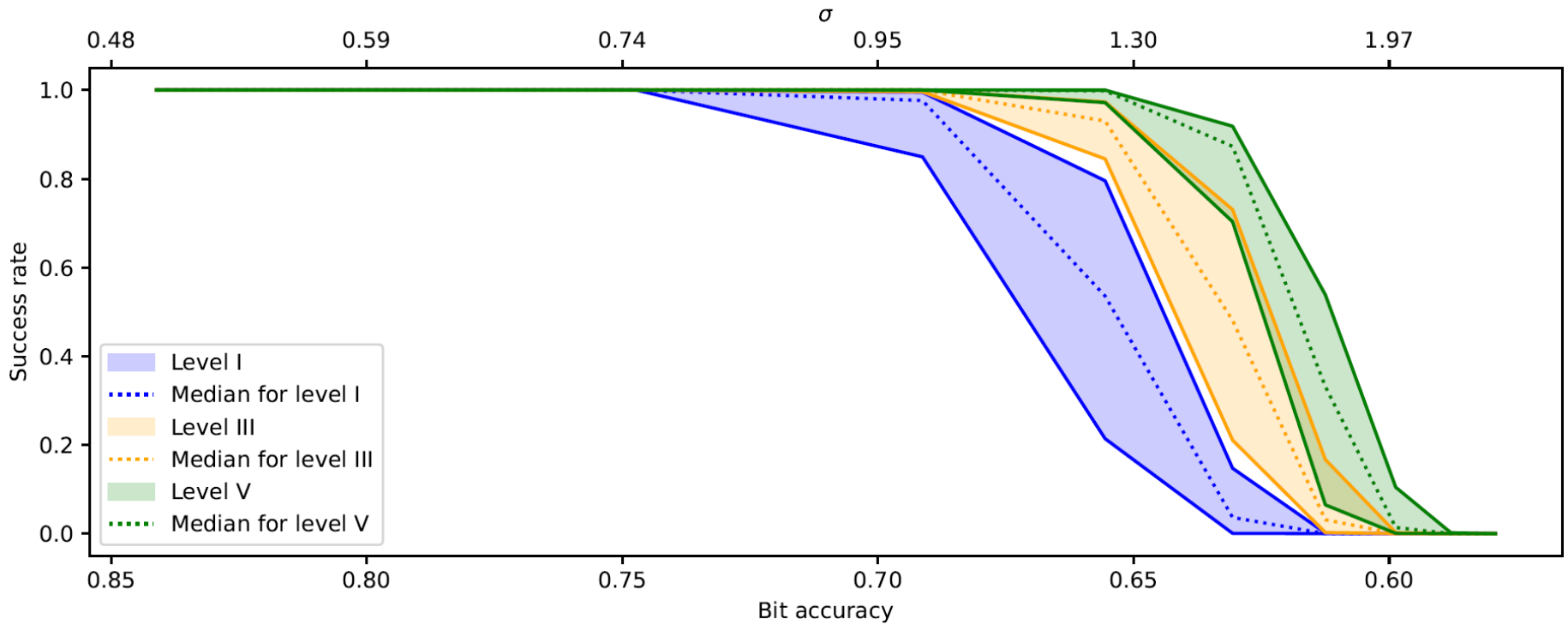
Simulation for FAEST

The **FAEST** algorithm is a digital signature scheme based on the VOLEitH framework and whose security is based on **AES**. More specifically, in FAEST is build a ZKPoK for the AES an Rijndael circuits using a variant of the QuickSilver proof system.



Simulation for SDitH

The algorithm **SDitH** is a digital signature scheme based on the VOLEitH framework and whose security is based on the **Syndrome Decoding** (SD) problem for random linear codes on a finite field. More specifically, in SDitH is build a ZKPoK of a low-weight vector x solution of a SD instance $y = Hx$.



Applicability of the attack

8 of the 40 candidates

Hypercube framework
[AMGH+22]

MPCitH with GGM tree
[KKW18]

Threshold framework
[FR22]

6 of the 14 candidates

VOLE-in-the-Head framework
[BBDG+23]

Improved threshold framework
[FR23]

- SDitH
- FAEST

- RYDE
- MQOM
- MiratH

→ VOLEitH variant

- PERK

→ VOLEitH framework ?

Non-supervised attack

- Target the **multiplication function in the field extension** used for the evaluations of the polynomial contraints
  - ➢ Power trace on a STM32F303RE

1. Non-supervised attack

- PCA + K-Means

2. K-Means predictions

2. Gaussian Mixture Model

3. Get the clusters probabilities

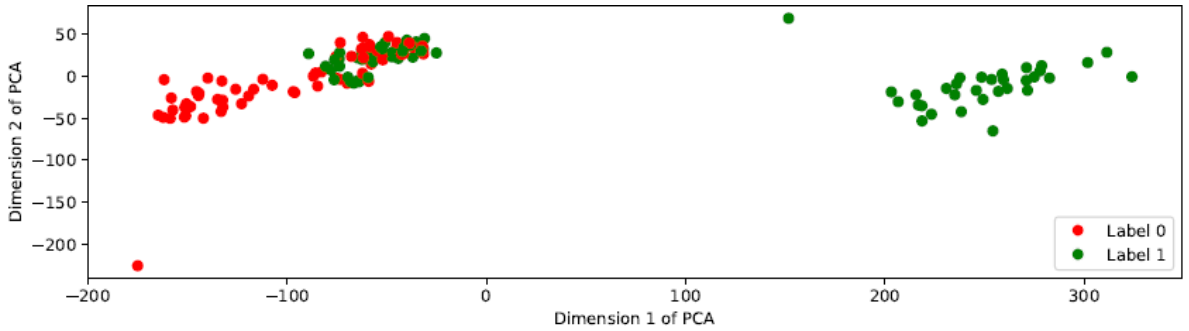


Figure 3: Principal Component Analysis

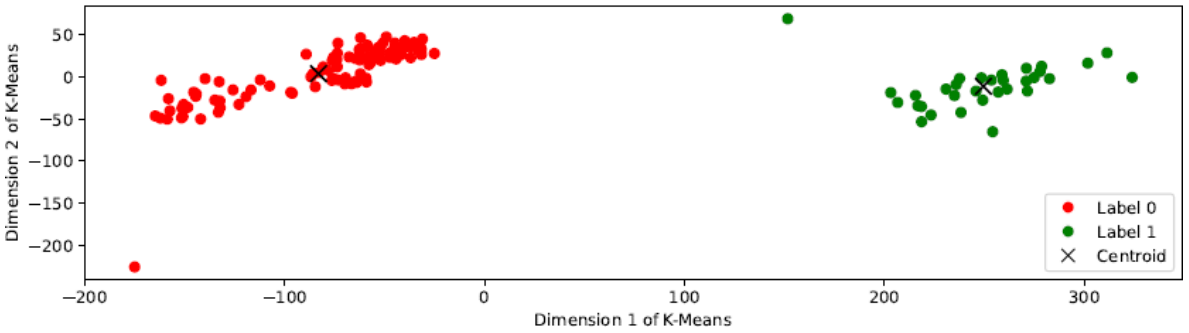


Figure 4: Clustering

Non-supervised attack

- Target the **multiplication function in the field extension** used for the evaluations of the polynomial contraints
  - ➢ Power trace on a STM32F303RE

1. Non-supervised attack

- PCA + K-Means

Accuracy ≃ **0,712**

2. K-Means predictions

2. Gaussian Mixture Model

3. Get the clusters probabilities

Accuracy = **1**

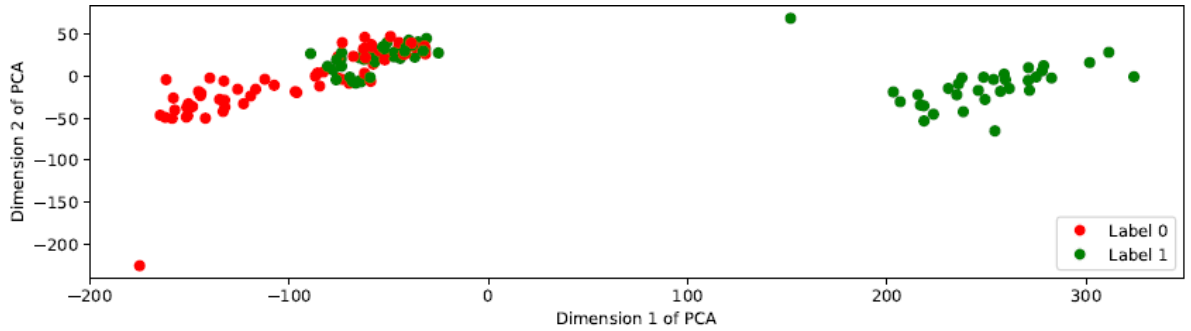


Figure 3: Principal Component Analysis

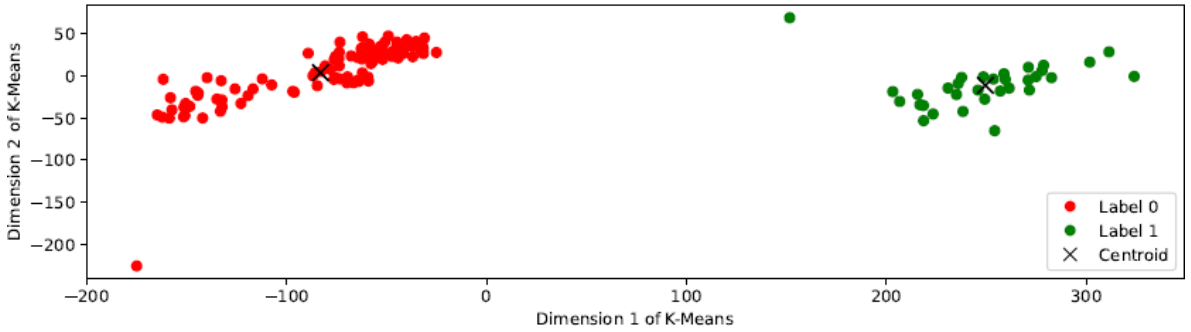
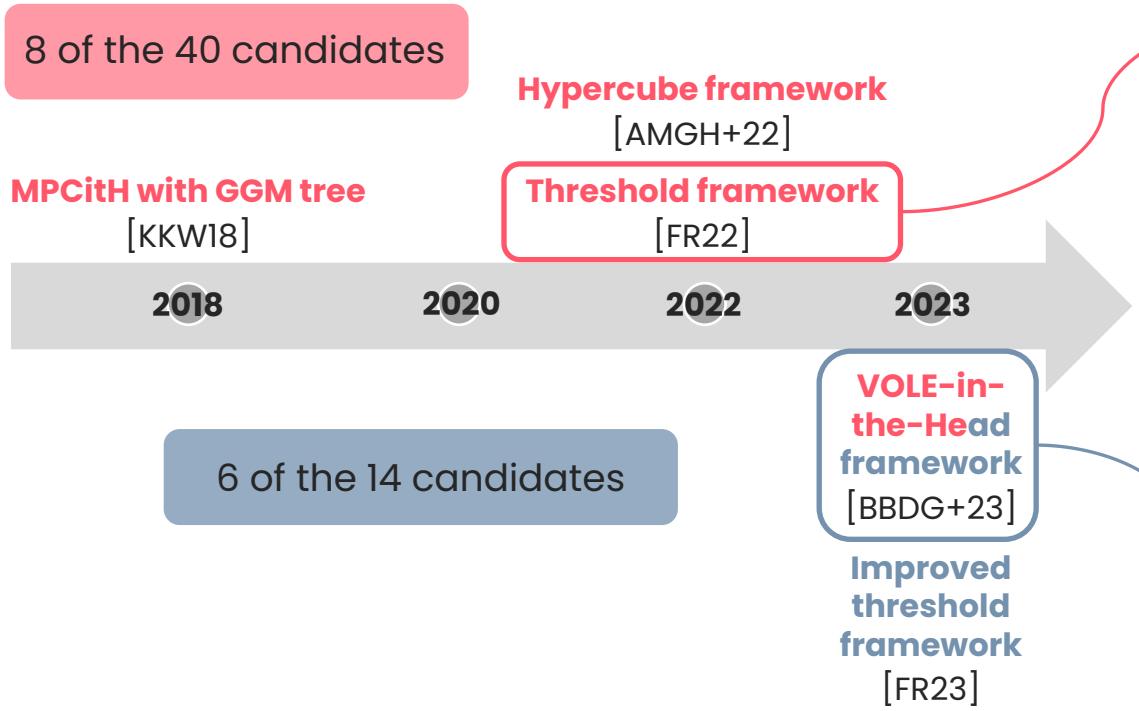


Figure 4: Clustering

Conclusion and future work



[GAG+24] Single Trace Side-Channel Attack on the MPC-in-the-Head Framework

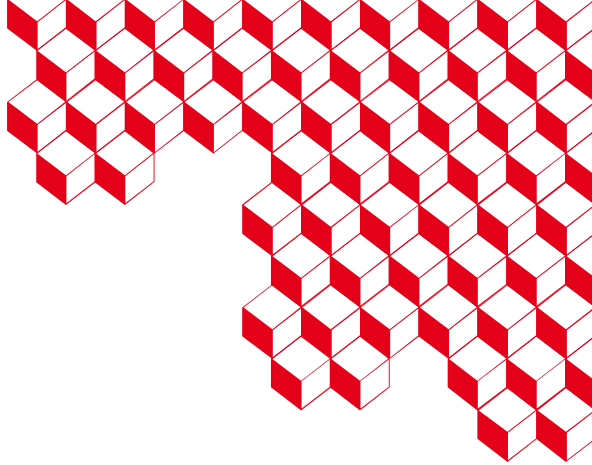
- Not applicable as is to second-round frameworks

Actual work. Non-supervised SASCA on the VOLEitH framework

- Use cases of FAEST and SDitH
- Future work:
  - RYDE
  - MQOM
  - MiratH
  - PERK

  } VOLEitH variant

  } VOLEitH framework ?

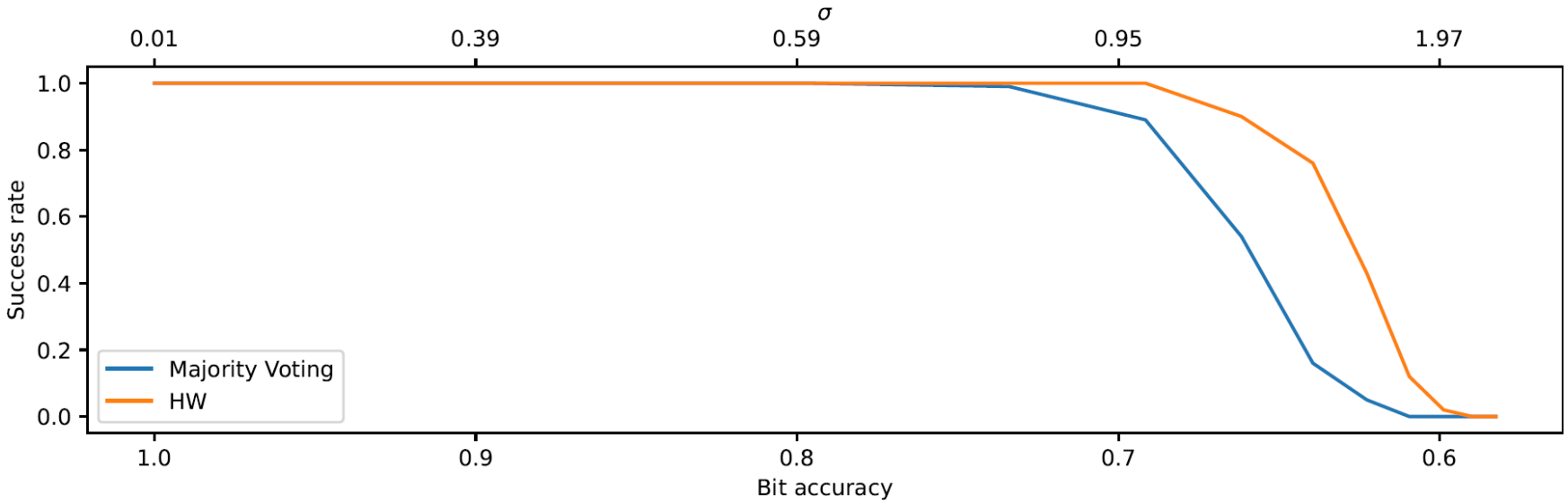


Thanks for your attention !

CEA-Leti, Grenoble, France
julie.godard@cea.fr

BP and Majority Voting

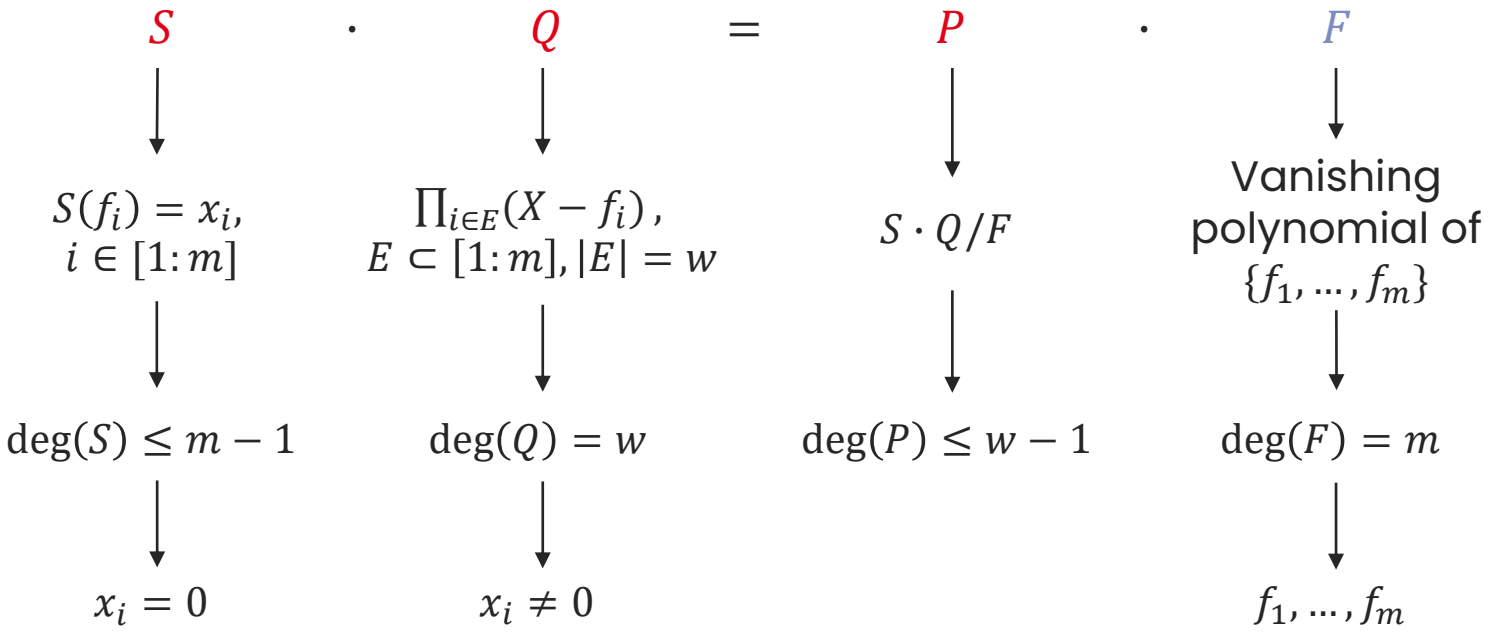
In a more generic case, with jitter or a noisier model, the clusters will overlap more and we will no longer have these labeling errors with such a strong belief → the BP algorithm no longer performs majority voting



Transformation of the SD problem

Let $f_1, \dots, f_q$ denote the elements of $\mathbb{F}_q$.
We want to build four polynomials S, Q, P and F such that:

- : witness-dependant
- : public



→ Q has degree w, so it can be zero in at most w points → S in non-zero in at most w points → $wt(x) \leq w$