

---

---

# VINAIGRETTE

## AN OPTIMISED FRAMEWORK TO THRESHOLDISE UOV AND MAYO

Diego F. Aranha, Ward Beullens, Giacomo Borin, Fabio Campos,  
**Sofía Celi(\*)**, Basil Hess, Lisa Kohl, Matthias J. Kannwischer, Guilhem Niot

<https://eprint.iacr.org/2024/1960.pdf>

<https://pqvinaigrette.org/>

<https://eprint.iacr.org/2026/TBD>

Presented at MPTS 2026: NIST Workshop for Multi-Party Threshold Schemes

2026-01-27

Speaker: Sofia Celi (Brave & University of Bristol)



# Setting the scene: threshold schemes

Renewed interest in providing *quantum-security* to *threshold signatures*:

- “Quantum resistance” listed as an important criterion on the recent NIST draft call for threshold signatures<sup>1</sup>
- Interest on **thresholdizing potentially-to-be-standardised schemes**: *UOV* and *MAYO* are currently part of the “on-ramp” NIST post-quantum process, and exhibit great performance

---

[1] See: <https://nvlpubs.nist.gov/nistpubs/ir/2025/NIST.IR.8214C.2pd.pdf>

# Our framework

- Starting point: the work of Cozzo et. al [CS19]
  - Explored the viability of applying generic MPC techniques to many of the NIST PQC submissions at the time (Round-2 of the NIST post-quantum first standardization process)
- Focus on:
  - Thresholdizing multivariate-based signature schemes:
    - Focus on **UOV** and **MAYO** → **(U)OV-based schemes**
      - Part of the Round-2 of the “onramp” post-quantum NIST process
    - They seem to arrive to practical communicational and computational measures
    - They seem to be tailored and feasible for real-world applications
    - Focus on potentially to-be-standardised schemes
    - Why a framework? Our techniques work on any OV-based scheme

---

[CS19] Daniele Cozzo and Nigel P. Smart. “Sharing the LUOV: Threshold Post-quantum Signatures”. doi: 10.1007/978-3-030-35199-1\_7

---

---

**Multivariate Quadratic (MQ) cryptography** is based on the assumed hardness of finding a solution to a **system of multivariate quadratic equations** (over a **finite field**). This problem is called the *MQ problem*.

Known to be *NP-hard* (for the decisional version) → seems to be hard on average for an extensive range of parameters:

The current record *mod 31* is solving a system of 22 equations in 22 variables.

$$\begin{aligned}x + 5x^2 + 3xy &= 4 \pmod{7} \\x^2 + 5xy + 5y^2 &= 1 \pmod{7}\end{aligned}$$

# Multivariate Quadratic Polynomials

A system of  $m$  (or  $k$ ) equations ( $f_k$ ) in  $n$  variables in  $\mathbf{F}_q$

$$f_k(x_1, x_2, \dots, x_n) = \sum_{1 \leq i < j \leq n} a_{i,j}^{(k)} x_i x_j + \sum_{1 \leq i \leq n} b_i^{(k)} x_i + c^{(k)}$$

where:

- First term consists of the quadratic terms with  $a_{i,j}$  as the coefficient
- Second term consists of the linear terms, with  $b_i$  as the coefficient
- $c$  is the constant term
- All the coefficients are in  $\mathbf{F}_q$

$$\begin{cases} y_1 = p_1(x_1, \dots, x_n) \\ y_2 = p_2(x_1, \dots, x_n) \\ \vdots \\ y_m = p_m(x_1, \dots, x_n) \end{cases}$$

# Multivariate quadratic (MQ) problem

$$\mathcal{P} : F_n \rightarrow F_m : x \rightarrow \mathcal{P}(x) = (p_1(x), \dots, p_m(x))$$

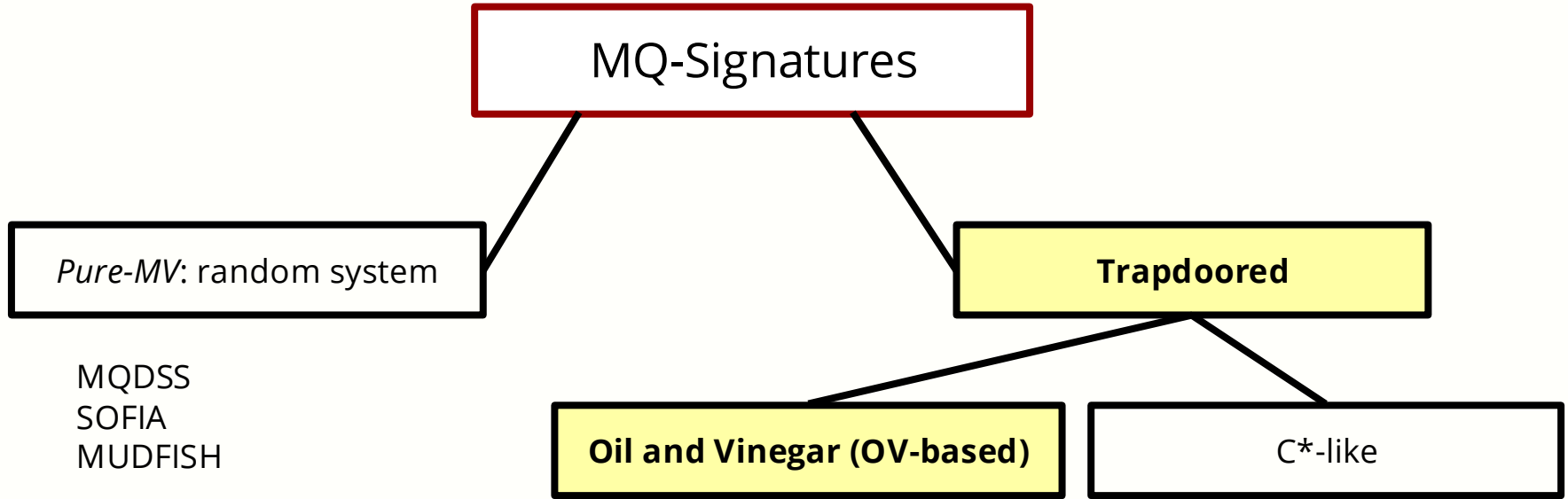
The MQ problem asks:

*given a multivariate quadratic map  $P : F_q^n \rightarrow F_q^m$  over a finite field  $F_q$  and a target  $\mathbf{t} \in F_q^m$  to find a solution  $\mathbf{s}$  such that  $\mathbf{P}(\mathbf{s}) = \mathbf{t}$ .*

or

*given  $(y_1, \dots, y_m)$  in  $F_q^m$ , find  $(x_1, \dots, x_n)$  in  $F_q^n$  with  $f_k(x_1, \dots, x_n) = y_k$  for  $1 \leq k \leq m$  if they exist.*

# Signatures schemes



# OV-based: Key Generation

*Key generation* (trapdoor information is just the basis for  $O$ ):

$P^{(1)} \rightarrow$  square upper triangular matrices, random

$P^{(2)} \rightarrow$  rectangular matrices, random

Solve for  $P^{(3)}$ :

$$P_i^{(3)} := \text{Upper} \left( -O^T P_i^{(1)} O - O^T P_i^{(2)} \right), \quad \forall i \in [m].$$

$$P_i = \begin{bmatrix} P_i^{(1)} & P_i^{(2)} \\ \mathbf{0} & P_i^{(3)} \end{bmatrix}$$



$$\begin{bmatrix} O^T & I_m \end{bmatrix} P_i \begin{bmatrix} O \\ I_m \end{bmatrix} = O^T P_i^{(1)} O + O^T P_i^{(2)} + P_i^{(3)} \in \mathbb{F}_q^{n \times n}$$

is skew symmetric

# Matrix-based: OV-based: Key Generation

*Key generation* (trapdoor info is just the basis for  $O$ ):

## Algorithm 1 $OV$ -based.KeyGen()

**Output:** A key pair  $(pk, sk)$ .

- 1: //Derive  $O$  and  $(P^{(1)}, P^{(2)})$  randomly. Note that this differs from the specification, as we are not deriving them from a fixed seed.
- 2:  $O \xleftarrow{\$} \mathbb{F}_q^{(n-o) \times o}$
- 3:  $\{P_i^{(1)}, P_i^{(2)}\}_{i \in [m]} \xleftarrow{\$} (\mathbb{F}_q^{(n-o) \times (n-o)}, \mathbb{F}_q^{(n-o) \times o})$
- 4: //Compute  $P_i^{(3)} \in \mathbb{F}_q^{o \times o}$ .
- 5: **for**  $i$  from 0 to  $(m - 1)$  **do**
- 6:    $P_i^{(3)} \leftarrow \text{Upper}(-O^T(P_i^{(1)}O - P_i^{(2)}))$
- 7:    $L_i \leftarrow ((P_i^{(1)} + P_i^{(1)T})O + P_i^{(2)})$
- 8: **return**  $(pk = (\{P_i^{(1)}, P_i^{(2)}, P_i^{(3)}\}_{i \in [m]}), sk = (O, \{L_i\}_{i \in [m]}))$ .

# Our framework: trapdoor-based

Trapdoor-based (both UOV and MAYO):

- Maps look random but have a “hidden structure” that allows a signer to compute pre-images efficiently and securely
- They are based on the *Hash-and-Sign with Retries* approach [CFGM]
  - Trapdoor is probabilistic

**PK:**  $P$ , description of the trapdoor function

**SK:** trapdoor information

**Signature:** input  $s$  from  $P(s) = H(m \parallel salt)$

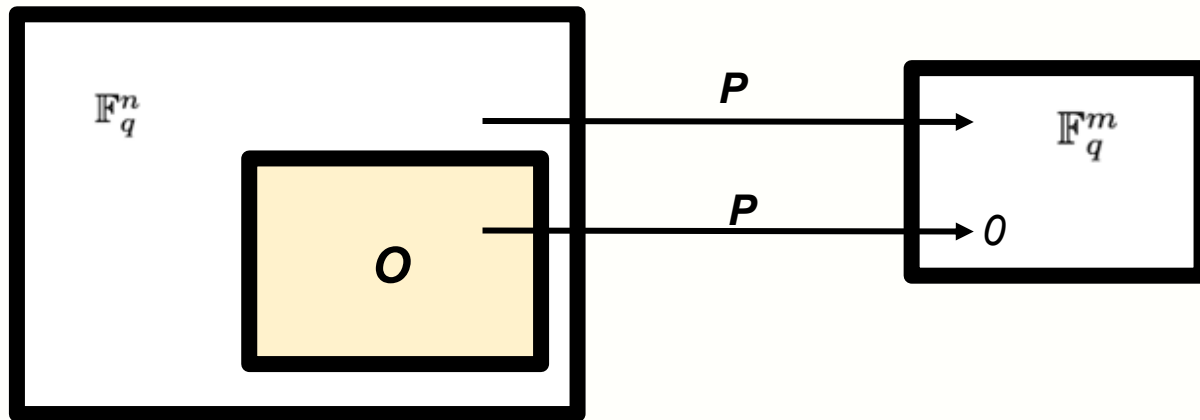
```
proc sign(sk : skey, m : msg) : signature = {
  var i1, i2, z, salt, sol, s;

  (i1, i2) <- sk;
  z < $ i1;
  (* repeat until *)
  salt < $ dsalt;
  sol < @ H.get(m, salt);
  s <- i2 z sol;
  while (s = None) {
    salt < $ dsalt;
    sol < @ H.get(m, salt);
    s <- i2 z sol;
  }
  return (salt, oget s);
}
```

# Trapdoor-based: (*Unbalanced*) Oil and Vinegar

## (Unbalanced) Oil and Vinegar: OV-based (UOV and MAYO)

**Trapdoor:** linear subspace  $O \subset \mathbb{F}_q^n$  (oil space) of dimension  $m$  on which  $P$  vanishes (for every vector:  $P(o) = 0 \forall o \in O$ )



# OV-based: Signing

Problem:  $P(s = (v+Ox) \parallel x) = H(msg \parallel salt)$

- Pick random  $v \in F_q^n$  (the vinegar)
- Solve for  $o \in O$  (the oil), such that  $P(v, o) = t$ :

$$\mathcal{P}(o + v) = \mathcal{P}(v) + \mathcal{P}(o) + \Delta_v(o) = t$$

$$\mathcal{P}(o + v) = \cancel{\mathcal{P}(v)} + \overset{\text{a constant}}{\cancel{\mathcal{P}(o)}} + \Delta_v(o) = t$$

linear system of  $m$  equations in  $m$  variables.

If no solution, **retry** for another  $v$ .

- Signature:  $s = o + v$

# Matrix description: OV-based: signing

**Algorithm 4** OV-based.Sign(sk, pk, msg)

**Input:** Secret key (sk), public key (pk).

**Input:** Message msg.

**Output:** Signature Sig = (S, salt).

```
1: //Parse  $(\mathbf{O}, \{\mathbf{L}_i\}_{i \in [m]})$  and  $(\{\mathbf{P}_i^{(1)}\}_{i \in [m]}, \{\mathbf{P}_i^{(2)}\}_{i \in [m]})$  from sk and pk.
2:  $(\mathbf{O}, \{\mathbf{L}\}_{i \in [m]}) \leftarrow \text{sk}$ 
3:  $\{\mathbf{P}_i^{(1)}, \mathbf{P}_i^{(2)}\}_{i \in [m]} \leftarrow \text{pk}$ 
4: //Hash salted message.
5:  $\text{salt} \xleftarrow{\$} \{0, 1\}^{\lambda+64}$ 
6:  $\mathbf{t} \leftarrow \mathcal{H}(\text{msg} \parallel \text{salt}) \triangleright \mathbf{t} \in \mathbb{F}_q^m$ 
7:
8:  $\mathbf{V} \xleftarrow{\$} \mathbb{F}_q^{k \times (n-o)}$ 
9: for  $i$  from 1 to  $m$  do
10:    $\mathbf{M}_i \leftarrow \mathbf{V} \cdot \mathbf{L}_i \triangleright \mathbf{M}_i \in \mathbb{F}_q^{k \times o}$ 
11:    $\mathbf{Y}_i \leftarrow \mathbf{V} \cdot \mathbf{P}_i^{(1)} \cdot \mathbf{V}^T \triangleright \mathbf{Y}_i \in \mathbb{F}_q^{k \times k}$ 
12: //Build the linear system  $\mathbf{Ax} = \mathbf{y}$ .
13:  $\mathbf{A} \leftarrow \text{OV-based.Compute\_A}(\{\mathbf{M}_i\}_{i \in [m]})$ 
14:  $\mathbf{y} \leftarrow \mathbf{t} - \text{OV-based.Compute\_y}(\{\mathbf{Y}_i\}_{i \in [m]})$ 
```

1. Build the linear system

# Matrix description: OV-based: signing

```
15:
16: //Try to sample a random solution  $\mathbf{x}$  to  $\mathbf{Ax} = \mathbf{y}$ .
17:  $\mathbf{x} \leftarrow \text{SampleSolution}(\mathbf{A}, \mathbf{y}) \triangleright \mathbf{x} \in \mathbb{F}_q^{k \times o} \cup \{\perp\}$ 
18: if  $\mathbf{x} = \perp$  then
19:   go to 7
20: //Output the signature.
21:  $\mathbf{X} \leftarrow \text{Matrixify}(\mathbf{x}) \triangleright \mathbf{X} \in \mathbb{F}_q^{k \times o}$ , s.t.  $\mathbf{x}$  is concatenation of rows of  $\mathbf{X}$ 
22:  $\mathbf{S} \leftarrow (\mathbf{V} + (\mathbf{OX}^T)^T, \mathbf{X}) \triangleright \mathbf{S} \in \mathbb{F}_q^{k \times n}$ 
23: return Sig = (S, salt).
```

2. Find a solution (if any)  
to the system

The algorithm involves retries (with small probability):

- Single-party setting: the signer *does not have to keep secret how many attempts* were required to find a solution

# Matrix description: OV-based: signing

```
15:
16: //Try to sample a random solution  $\mathbf{x}$  to  $\mathbf{Ax} = \mathbf{y}$ .
17:  $\mathbf{x} \leftarrow \text{SampleSolution}(\mathbf{A}, \mathbf{y}) \triangleright \mathbf{x} \in \mathbb{F}_q^{k \times o} \cup \{\perp\}$ 
18: if  $\mathbf{x} = \perp$  then
19:   go to 7
20: //Output the signature.
21:  $\mathbf{X} \leftarrow \text{Matrixify}(\mathbf{x}) \triangleright \mathbf{X} \in \mathbb{F}_q^{k \times o}$ , s.t.  $\mathbf{x}$  is concatenation of rows of  $\mathbf{X}$ 
22:  $\mathbf{S} \leftarrow (\mathbf{V} + (\mathbf{OX}^\top)^\top, \mathbf{X}) \triangleright \mathbf{S} \in \mathbb{F}_q^{k \times n}$ 
23: return Sig = (S, salt).
```

The most difficult procedure to thresholdize ( $\mathbf{Ax} = \mathbf{y}$ ):

- solving a **system of linear equations** in which **both the input matrix and the target vector remain secret** → solving in an oblivious manner

# Solving Systems of Linear Equations Obliviously

The most difficult<sup>(\*)</sup> procedure to thresholdize:

- Solving a **system of linear equations** in which **both the input matrix and the target vector remain secret** → solving in an oblivious manner

$$A \cdot x = y$$

where  $A$  has a dependence on  $O$ , and  $y$  on the *message*, *salt* and  $v$ .

In the context of OV-based schemes:

○ Input:

- $A$ :  $\sum_{i=1}^m ((\mathbf{P}_i^{(1)} + \mathbf{P}_i^{(1)\top}) \cdot \mathbf{O} - \mathbf{P}_i^{(2)\top}) \cdot \mathbf{v} \cdot x_i = 0$ .
- Relationship with trapdoor information  $O$

# Solving Systems of Linear Equations Obliviously

## One side-masking [CS19]:

- Only masking  $A$  on one side:
  - Compute  $T = A \cdot S$  (“**right-masking**”) → Matrix  $T$  retains the same image space as  $A$ 
    - If  $A$  is not full rank, the image space is non-trivial
    - In UOV and MAYO, with low probability,  $A$  is not full rank:
      - Restart probability
    - Leaks the structure of  $O$
  - Compute  $T = R \cdot A$  (“**left-masking**”)
    - Leaks the kernel of  $A$

**High leakage that arguably appears insecure**

# Interlude: framework model

- Our work is set in the *Universal-Composability (UC)* framework
- It is agnostic to any security choice, but:
  - We set our work in the **actively UC-secure** model with **dishonest majority** (e.g. up to  $t - 1$  corrupted parties in a  $t$ -out-of- $n$  setting) over a synchronous network
  - We make usage of the arithmetic black box model (ABB):
    - basic I/O, arithmetic on field elements and matrices, and randomness sampling operations

# Solving Systems of Linear Equations Obliviously

## Low leakage version

### Protocol 1: $\Pi_{\text{Solve}}$

The protocol is set in the  $\mathcal{F}_{\text{ABB}}$ -hybrid. All the commands except for (solve) are forwarded directly to  $\mathcal{F}_{\text{ABB}}$ .


On input (solve,  $[\mathbf{A}]$ ,  $[\mathbf{b}]$ ), where  $\mathbf{A}$  has dimensions  $s \times t$  and  $[\mathbf{b}]$  has dimension  $s$ , the parties proceed as follows:

1. Parties call  $[\mathbf{R}] \leftarrow \text{rand}(\mathbb{F}_q^{s \times s})$  and  $[\mathbf{S}] \leftarrow \text{rand}(\mathbb{F}_q^{t \times t})$ .
2. Parties call  $[\mathbf{A} \cdot \mathbf{S}] \leftarrow [\mathbf{A}] \cdot [\mathbf{S}]$ .
3. Parties call  $[\mathbf{T}] \leftarrow [\mathbf{R}] \cdot [\mathbf{A} \cdot \mathbf{S}]$ .
4. Parties open  $\mathbf{T} \leftarrow [\mathbf{T}]$ . If  $r = \text{rank}(\mathbf{T}) < s$  then the parties output (rank-defect,  $r$ ).
5. Otherwise, let  $\mathbf{T}^{-1} \in \mathbb{F}_q^{t \times s}$  be a right inverse of  $\mathbf{T}$ , that is,  $\mathbf{T}\mathbf{T}^{-1} = \mathbf{I}_{s \times s}$ . The parties call  $[\mathbf{A}^{-1}] \leftarrow [\mathbf{S}] \cdot \mathbf{T}^{-1} \cdot [\mathbf{R}]$ . It can be checked that  $\mathbf{A}^{-1} \in \mathbb{F}_q^{t \times s}$  satisfies  $\mathbf{A} \cdot \mathbf{A}^{-1} = \mathbf{I}_{s \times s}$ .
6. Let  $\beta_1, \dots, \beta_{t-s} \in \mathbb{F}_q^{t-s}$  be a basis for  $\ker(\mathbf{T})$ . The parties call  $([z_1], \dots, [z_{t-s}]) \leftarrow \text{rand}(\mathbb{F}_q^{t-s})$ .
7. Parties compute locally  $[\mathbf{z}] \leftarrow \sum_{i=1}^{t-s} [z_i] \cdot \beta_i$ .
8. Parties call  $[\mathbf{x}] \leftarrow [\mathbf{A}^{-1}] \cdot [\mathbf{b}] + [\mathbf{S}] \cdot [\mathbf{z}]$ .
9. Output  $[\mathbf{x}]$ .

# Solving Systems of Linear Equations Obliviously

## Low leakage version

Only reveals if  
the matrix is  
rank-deficient  
and its rank



### Protocol 1: $\Pi_{\text{Solve}}$

The protocol is set in the  $\mathcal{F}_{\text{ABB}}$ -hybrid. All the commands except for (solve) are forwarded directly to  $\mathcal{F}_{\text{ABB}}$ .

On input (solve,  $[\mathbf{A}]$ ,  $[\mathbf{b}]$ ), where  $\mathbf{A}$  has dimensions  $s \times t$  and  $[\mathbf{b}]$  has dimension  $s$ , the parties proceed as follows:

1. Parties call  $[\mathbf{R}] \leftarrow \text{rand}(\mathbb{F}_q^{s \times s})$  and  $[\mathbf{S}] \leftarrow \text{rand}(\mathbb{F}_q^{t \times t})$ .
2. Parties call  $[\mathbf{A} \cdot \mathbf{S}] \leftarrow [\mathbf{A}] \cdot [\mathbf{S}]$ .
3. Parties call  $[\mathbf{T}] \leftarrow [\mathbf{R}] \cdot [\mathbf{A} \cdot \mathbf{S}]$ .
4. Parties open  $\mathbf{T} \leftarrow [\mathbf{T}]$ . If  $r = \text{rank}(\mathbf{T}) < s$  then the parties output (rank-defect,  $r$ ).
5. Otherwise, let  $\mathbf{T}^{-1} \in \mathbb{F}_q^{t \times s}$  be a right inverse of  $\mathbf{T}$ , that is,  $\mathbf{T}\mathbf{T}^{-1} = \mathbf{I}_{s \times s}$ . The parties call  $[\mathbf{A}^{-1}] \leftarrow [\mathbf{S}] \cdot \mathbf{T}^{-1} \cdot [\mathbf{R}]$ . It can be checked that  $\mathbf{A}^{-1} \in \mathbb{F}_q^{t \times s}$  satisfies  $\mathbf{A} \cdot \mathbf{A}^{-1} = \mathbf{I}_{s \times s}$ .
6. Let  $\beta_1, \dots, \beta_{t-s} \in \mathbb{F}_q^{t-s}$  be a basis for  $\ker(\mathbf{T})$ . The parties call  $([z_1], \dots, [z_{t-s}]) \leftarrow \text{rand}(\mathbb{F}_q^{t-s})$ .
7. Parties compute locally  $[\mathbf{z}] \leftarrow \sum_{i=1}^{t-s} [z_i] \cdot \beta_i$ .
8. Parties call  $[\mathbf{x}] \leftarrow [\mathbf{A}^{-1}] \cdot [\mathbf{b}] + [\mathbf{S}] \cdot [\mathbf{z}]$ .
9. Output  $[\mathbf{x}]$ .

# Solving Systems of Linear Equations Obliviously

## Low leakage version

Valid solution:

$$\begin{aligned} \mathbf{A} \cdot \mathbf{x} &= \mathbf{A} \cdot (\mathbf{A}^{-1}\mathbf{b} + \mathbf{S}\mathbf{z}) \\ &= \mathbf{A} \cdot (\mathbf{S}\mathbf{T}^{-1}\mathbf{R}\mathbf{b}) + \mathbf{A}\mathbf{S}\mathbf{z} \\ &= \mathbf{R}^{-1} \cdot (\mathbf{R}\mathbf{A}\mathbf{S} \cdot \mathbf{T}^{-1} \cdot \mathbf{R}\mathbf{b} + \mathbf{R}\mathbf{A}\mathbf{S} \cdot \mathbf{z}) \\ &= \mathbf{R}^{-1} \cdot (\mathbf{T} \cdot \mathbf{T}^{-1} \cdot \mathbf{R}\mathbf{b} + \mathbf{T} \cdot \mathbf{z}) \\ &= \mathbf{R}^{-1} \cdot (\mathbf{I}_{s \times s} \cdot \mathbf{R}\mathbf{b} + \mathbf{0}) \\ &= \mathbf{b}. \end{aligned}$$

### Protocol 1: $\Pi_{\text{Solve}}$

The protocol is set in the  $\mathcal{F}_{\text{ABB}}$ -hybrid. All the commands except for (solve) are forwarded directly to  $\mathcal{F}_{\text{ABB}}$ .

On input (solve,  $[\mathbf{A}]$ ,  $[\mathbf{b}]$ ), where  $\mathbf{A}$  has dimensions  $s \times t$  and  $[\mathbf{b}]$  has dimension  $s$ , the parties proceed as follows:

1. Parties call  $[\mathbf{R}] \leftarrow \text{rand}(\mathbb{F}_q^{s \times s})$  and  $[\mathbf{S}] \leftarrow \text{rand}(\mathbb{F}_q^{t \times t})$ .
2. Parties call  $[\mathbf{A} \cdot \mathbf{S}] \leftarrow [\mathbf{A}] \cdot [\mathbf{S}]$ .
3. Parties call  $[\mathbf{T}] \leftarrow [\mathbf{R}] \cdot [\mathbf{A} \cdot \mathbf{S}]$ .
4. Parties open  $\mathbf{T} \leftarrow [\mathbf{T}]$ . If  $r = \text{rank}(\mathbf{T}) < s$  then the parties output (rank-defect,  $r$ ).
5. Otherwise, let  $\mathbf{T}^{-1} \in \mathbb{F}_q^{t \times s}$  be a right inverse of  $\mathbf{T}$ , that is,  $\mathbf{T}\mathbf{T}^{-1} = \mathbf{I}_{s \times s}$ . The parties call  $[\mathbf{A}^{-1}] \leftarrow [\mathbf{S}] \cdot \mathbf{T}^{-1} \cdot [\mathbf{R}]$ . It can be checked that  $\mathbf{A}^{-1} \in \mathbb{F}_q^{t \times s}$  satisfies  $\mathbf{A} \cdot \mathbf{A}^{-1} = \mathbf{I}_{s \times s}$ .
6. Let  $\beta_1, \dots, \beta_{t-s} \in \mathbb{F}_q^{t-s}$  be a basis for  $\ker(\mathbf{T})$ . The parties call  $([z_1], \dots, [z_{t-s}]) \leftarrow \text{rand}(\mathbb{F}_q^{t-s})$ .
7. Parties compute locally  $[\mathbf{z}] \leftarrow \sum_{i=1}^{t-s} [z_i] \cdot \beta_i$ .
8. Parties call  $[\mathbf{x}] \leftarrow [\mathbf{A}^{-1}] \cdot [\mathbf{b}] + [\mathbf{S}] \cdot [\mathbf{z}]$ .
9. Output  $[\mathbf{x}]$ .

# Solving Systems of Linear Equations Obliviously

## Low leakage version

Expensive: six rounds!

Can we do it better?

### Protocol 1: $\Pi_{\text{Solve}}$

The protocol is set in the  $\mathcal{F}_{\text{ABB}}$ -hybrid. All the commands except for (solve) are forwarded directly to  $\mathcal{F}_{\text{ABB}}$ .

On input (solve,  $[\mathbf{A}]$ ,  $[\mathbf{b}]$ ), where  $\mathbf{A}$  has dimensions  $s \times t$  and  $[\mathbf{b}]$  has dimension  $s$ , the parties proceed as follows:

1. Parties call  $[\mathbf{R}] \leftarrow \text{rand}(\mathbb{F}_q^{s \times s})$  and  $[\mathbf{S}] \leftarrow \text{rand}(\mathbb{F}_q^{t \times t})$ .
2. Parties call  $[\mathbf{A} \cdot \mathbf{S}] \leftarrow [\mathbf{A}] \cdot [\mathbf{S}]$ .
3. Parties call  $[\mathbf{T}] \leftarrow [\mathbf{R}] \cdot [\mathbf{A} \cdot \mathbf{S}]$ .
4. Parties open  $\mathbf{T} \leftarrow [\mathbf{T}]$ . If  $r = \text{rank}(\mathbf{T}) < s$  then the parties output (rank-defect,  $r$ ).
5. Otherwise, let  $\mathbf{T}^{-1} \in \mathbb{F}_q^{t \times s}$  be a right inverse of  $\mathbf{T}$ , that is,  $\mathbf{T}\mathbf{T}^{-1} = \mathbf{I}_{s \times s}$ . The parties call  $[\mathbf{A}^{-1}] \leftarrow [\mathbf{S}] \cdot \mathbf{T}^{-1} \cdot [\mathbf{R}]$ . It can be checked that  $\mathbf{A}^{-1} \in \mathbb{F}_q^{t \times s}$  satisfies  $\mathbf{A} \cdot \mathbf{A}^{-1} = \mathbf{I}_{s \times s}$ .
6. Let  $\beta_1, \dots, \beta_{t-s} \in \mathbb{F}_q^{t-s}$  be a basis for  $\ker(\mathbf{T})$ . The parties call  $([z_1], \dots, [z_{t-s}]) \leftarrow \text{rand}(\mathbb{F}_q^{t-s})$ .
7. Parties compute locally  $[\mathbf{z}] \leftarrow \sum_{i=1}^{t-s} [z_i] \cdot \beta_i$ .
8. Parties call  $[\mathbf{x}] \leftarrow [\mathbf{A}^{-1}] \cdot [\mathbf{b}] + [\mathbf{S}] \cdot [\mathbf{z}]$ .
9. Output  $[\mathbf{x}]$ .

# Solving Systems of Linear Equations Obliviously

## Low leakage version

Can we do it better?

Yes: 4 rounds!  
(This procedure can  
be used  
independently by  
other systems)

### Procedure 1: $\Pi_{\text{Solve}}$

The protocol is set in the  $\mathcal{F}_{\text{ABB}}$ -hybrid. All the commands are forwarded directly to  $\mathcal{F}_{\text{ABB}}$ .

On input (solve,  $[\mathbf{A}]$ ,  $[\mathbf{b}]$ ), where  $\mathbf{A}$  has dimensions  $s \times t$  and  $[\mathbf{b}]$  has dimension  $s$ , the parties proceed as follows:

1. Parties call  $[\mathbf{R}] \leftarrow \text{rand}(\mathbb{F}^{s \times s})$  and  $[\mathbf{S}] \leftarrow \text{rand}(\mathbb{F}^{t \times t})$ .
2. Parties call  $[\mathbf{R} \cdot \mathbf{A}] \leftarrow [\mathbf{R}] \cdot [\mathbf{A}]$ .
3. Parties call  $[\mathbf{T}] \leftarrow [\mathbf{R} \cdot \mathbf{A}] \cdot [\mathbf{S}]$ .  $\triangleright \mathbf{R} \cdot \mathbf{A} \cdot \mathbf{S} = \mathbf{T}$ .
4. Parties call  $[\mathbf{h}] \leftarrow [\mathbf{R}] \cdot [\mathbf{b}]$ .  $\triangleright \mathbf{R} \cdot \mathbf{b} = \mathbf{h}$
5. Parties open  $\mathbf{T} \leftarrow [\mathbf{T}]$ . If  $\text{rank}(\mathbf{T}) < s$  parties output (rank-deficient,  $\text{rank}(\mathbf{T})$ ).
6. Otherwise, compute locally  $\mathbf{T}^{-1} \in \mathbb{F}_q^{t \times s}$  and a basis  $\{\beta_i\}_{i=1}^{t-s}$  of  $\ker(\mathbf{T})$  (right kernel).
7. Call locally  $([z_i])_{i=1}^{t-s} \leftarrow \text{rand}(\mathbb{F}^{t-s})$ .
8. Compute locally  $[\mathbf{z}] \leftarrow \sum_i [z_i] \cdot \beta_i$ .  $\triangleright \mathbf{z}$  is uniformly random in  $\ker(\mathbf{T})$
9. Compute locally  $[\mathbf{T}^{-1} \cdot \mathbf{h} + \mathbf{z}] \leftarrow \mathbf{T}^{-1} \cdot [\mathbf{h}] + [\mathbf{z}]$  and  $[\mathbf{T}^{-1} \cdot \mathbf{R}] \leftarrow \mathbf{T}^{-1} \cdot [\mathbf{R}]$ .
10. Parties call  $[\mathbf{x}] \leftarrow [\mathbf{S}] \cdot [\mathbf{T}^{-1} \cdot \mathbf{h} + \mathbf{z}]$ .
11.  $\triangleright$  only for solve-inv  $\triangleleft$  Parties call  $[\mathbf{A}^{-1}] \leftarrow [\mathbf{S}] \cdot [\mathbf{T}^{-1} \cdot \mathbf{R}]$ .
12. Output  $[\mathbf{x}]$  and  $\triangleright$  only for solve-inv  $\triangleleft$   $[\mathbf{A}^{-1}]$ .

# Threshold signing in short

## Procedure 3: $\pi_{\text{Sign}}$

**Input:** Public key and secret key stored in  $\mathcal{F}_{\text{ABB+Solve}}$ :  $\text{pk} = (\mathbf{P}_i^{(1)}, \mathbf{P}_i^{(2)}, \mathbf{P}_i^{(3)})_{i \in [m]}$ ,  $\text{sk} = ([\mathbf{O}], [\mathbf{L}_1], \dots, [\mathbf{L}_m])$ . A message  $\text{msg}$  to be signed.

**Output:** A signature  $\mathbf{S} \in \mathbb{F}_q^{k \times n}$  on  $\text{msg}$ .

1. Parties call  $\text{salt} \leftarrow \text{coin}(\{0, 1\}^{\lambda+64})$  and let  $\mathbf{t} \leftarrow \mathcal{H}(\text{msg} \parallel \text{salt}) \in \mathbb{F}_q^m$ . Let  $\text{Leaks} = [ ]$ .
2. Parties call  $[\mathbf{V}] \leftarrow \text{rand}(\mathbb{F}_q^{k \times (n-o)})$ .
3. For  $i \in [m]$ : Parties call  $[\mathbf{M}_i] \leftarrow [\mathbf{V}] \cdot [\mathbf{L}_i]$ .
4. For  $i \in [m]$ : Parties call  $[\mathbf{Y}_i] \leftarrow [\mathbf{V}] \cdot \mathbf{P}_i^{(1)} \cdot [\mathbf{V}^\top]$ .
5. Compute locally  $[\mathbf{A}] \leftarrow \text{OV-based.Compute\_A}(\{[\mathbf{M}_i]\}_{i \in [m]})$  and  $[\mathbf{y}] \leftarrow [\mathbf{t}] - \text{OV-based.Compute\_y}(\{[\mathbf{Y}_i]\}_{i \in [m]})$ . This is possible since both  $\text{OV-based.Compute\_y}$  and  $\text{OV-based.Compute\_A}$  are *linear* functions of their arguments.
6. Parties call the command `solve` of  $\mathcal{F}_{\text{ABB+Solve}}$  on inputs  $[\mathbf{A}]$  and  $[\mathbf{y}]$ . If the output is (rank-deficient,  $r$ ), parties append  $r$  to  $\text{Leaks}$  and **go to 2**. Else, let  $[\mathbf{x}]$  be the output.
7. Parties compute locally  $[\mathbf{X}] \leftarrow \text{Matrixify}([\mathbf{x}])$  and call  $[\mathbf{X} \cdot \mathbf{O}^\top] \leftarrow [\mathbf{X}] \cdot [\mathbf{O}^\top]$ .
8. Parties compute locally  $[\mathbf{S}'] \leftarrow [\mathbf{V} + (\mathbf{O}\mathbf{X}^\top)^\top]$ , and they open  $\mathbf{S}' \leftarrow [\mathbf{S}']$ .
9. Parties return as output  $(\text{Sig} = (\mathbf{S}, \text{salt}), \text{Leaks})$ , where  $\mathbf{S} = (\mathbf{S}' \mid \mathbf{X}) \in \mathbb{F}_q^{k \times n}$ .

Can use a trusted dealer or a Distributed Key Generation (DKG) protocol

# Threshold signing in short

## Procedure 3: $\pi_{\text{Sign}}$

**Input:** Public key and secret key stored in  $\mathcal{F}_{\text{ABB+Solve}}$ :  $\text{pk} = (\mathbf{P}_i^{(1)}, \mathbf{P}_i^{(2)}, \mathbf{P}_i^{(3)})_{i \in [m]}$ ,  $\text{sk} = ([\mathbf{O}], [\mathbf{L}_1], \dots, [\mathbf{L}_m])$ . A message  $\text{msg}$  to be signed.

**Output:** A signature  $\mathbf{S} \in \mathbb{F}_q^{k \times n}$  on  $\text{msg}$ .

1. Parties call  $\text{salt} \leftarrow \text{coin}(\{0, 1\}^{\lambda+64})$  and let  $\mathbf{t} \leftarrow \mathcal{H}(\text{msg} \parallel \text{salt}) \in \mathbb{F}_q^m$ . Let  $\text{Leaks} = [ ]$ .
2. Parties call  $[\mathbf{V}] \leftarrow \text{rand}(\mathbb{F}_q^{k \times (n-o)})$
3. For  $i \in [m]$ : Parties call  $[\mathbf{M}_i] \leftarrow [\mathbf{V}] \cdot [\mathbf{L}_i]$ .
4. For  $i \in [m]$ : Parties call  $[\mathbf{Y}_i] \leftarrow [\mathbf{V}] \cdot \mathbf{P}_i^{(1)} \cdot [\mathbf{V}^\top]$ .
5. Compute locally  $[\mathbf{A}] \leftarrow \text{OV-based.Compute\_A}(\{[\mathbf{M}_i]\}_{i \in [m]})$  and  $[\mathbf{y}] \leftarrow [\mathbf{t}] - \text{OV-based.Compute\_y}(\{[\mathbf{Y}_i]\}_{i \in [m]})$ . This is possible since both  $\text{OV-based.Compute\_y}$  and  $\text{OV-based.Compute\_A}$  are *linear* functions of their arguments.
6. Parties call the command `solve` of  $\mathcal{F}_{\text{ABB+Solve}}$  on inputs  $[\mathbf{A}]$  and  $[\mathbf{y}]$ . If the output is (rank-deficient,  $r$ ), parties append  $r$  to  $\text{Leaks}$  and **go to 2**. Else, let  $[\mathbf{x}]$  be the output.
7. Parties compute locally  $[\mathbf{X}] \leftarrow \text{Matrixify}([\mathbf{x}])$  and call  $[\mathbf{X} \cdot \mathbf{O}^\top] \leftarrow [\mathbf{X}] \cdot [\mathbf{O}^\top]$ .
8. Parties compute locally  $[\mathbf{S}'] \leftarrow [\mathbf{V} + (\mathbf{O}\mathbf{X}^\top)^\top]$ , and they open  $\mathbf{S}' \leftarrow [\mathbf{S}']$ .
9. Parties return as output  $(\text{Sig} = (\mathbf{S}, \text{salt}), \text{Leaks})$ , where  $\mathbf{S} = (\mathbf{S}' \mid \mathbf{X}) \in \mathbb{F}_q^{k \times n}$ .

Secure, but expensive!

- Many depths of multiplications
- Many message dependent values
- Many rounds
- Can we do better?
- Can we achieve one-message dependent round?

# Efficient threshold signing

```

OV-based.Sign(sk, pk, msg) / OV-based.SignDet(sk, pk, msg, salt)
1: // Parse secret key and public key
2: Parse sk = (O, {Li}i∈[m])
3: Parse pk = ({Pi(1)}i∈[m], {Pi(2), · }i∈[m])
4: // 1. Hash salted message
5: salt  $\xleftarrow{\$}$  {0,1}λ+εk
6: // Deterministic variant: salt is chosen by the caller
7: t ← H(msg||salt)
8: // 2. Preprocessing quadratic forms
9: for ctr = 1, ..., ∞ do // in [BCC+23] bounded to 256 iterations.
10: V  $\xleftarrow{\$}$  Fqk×(n-o)
11: for i ← 1 to m do // Mi ∈ Fqk×n
12: Mi ← V · Li
13: Mi ← V · [Pi(1) + Pi(1)T Pi(2)]
14: Yi ← V · Pi(1) · VT // Yi ∈ Fqk×k
15: A ← OV-based.Compute_A(O, {Mi}i∈[m])
16: if A is not full-rank then continue
17: // 3. Message-dependent adjustment
18: y ← t - OV-based.Compute_y({Yi}i∈[m])
19: y ← [0k(n-o) y]
20: // 4. Sampling a solution to the linear system: x to A'x = y
21: x ← SampleSolution(A, y) // x ∈ Fqk×n
22: // 5. Assemble the signature
23: X ← Matrixify(x) // X ∈ Fqk×n, s.t. x is concatenation of rows
24: S ← (V + (OXT)T, X)
25: S ← (V + X[:, : n - o], X[:, n - o :]) // S ∈ Fqk×n
26: return Sig = (S, salt)
    
```

- Dashed boxes show the original operations;
- solid boxes show the modified operations for **multiplicative depth minimisation**;
- double-bordered boxes show the **explicit-salt variant**

# Efficient threshold signing

OV-based schemes are flexible:

- We can remove the usage of salt and not degrade security → **explicit-salt variant**
- We can use parity-check matrix:
  - In standard OV signing, the signature requires an online multiplication  $S = O \cdot X$  because  $X$  is message-derived. In our variant, the solution  $X$  is already in the oil space, so  $S$  is built by additions/concatenations of  $X$  with precomputed randomness  $V$  → fewer online ops + lower MPC round complexity → **depth-reduced variant**
  - Instead of implicitly enforcing oil-space membership using secret matrices  $L_i$ , we work over the full space  $Fq^n$  and explicitly impose oil-space constraints using a parity-check matrix:

$$H_{1,O} = (I_{n-o} \quad -O), \quad z \in O \iff H_{1,O}z = 0,$$

# Efficient threshold signing

OV-based schemes are flexible:

- We can pre-compute in advance many values

Protocol		Offline Rounds	Online Rounds	System Size	
[CEN25]		–	$6 + 3/p_{\text{inv}}$	$m \times ko$	
This Work	<b>Toggle A</b>	<b>Toggle B</b>			
	Randomized	Standard	$1 + 4/p_{\text{inv}}$	2	$m \times ko$
	Randomized	Reduced	$1 + 3/p_{\text{inv}}$	2	$(k(n - o) + m) \times kn$
	Explicit	Standard	$1 + 4/p_{\text{inv}}$	1	$m \times ko$
	Explicit	Reduced	$1 + 3/p_{\text{inv}}$	1	$(k(n - o) + m) \times kn$

Table 1:  $p_{\text{inv}}$  denotes the probability that the sampled matrix is full rank,  $k$  is the whipping factor in MAYO. “Explicit” uses an explicit salt, while, “Randomized” does not. “Standard” uses the pre-processed standard algorithm; while “Reduced” uses the reduced multiplicative-depth one.

# Efficient threshold signing

Using:

- Efficient solving procedure
- Pre-processing as many values as possible
- Explicit-salt variant OR Explicit-depth-reduce variant
- We end in an scheme that is:
  - One message-dependent online round, while all offline rounds are independent
  - Works with the verification algorithm of single-party OV-based schemes

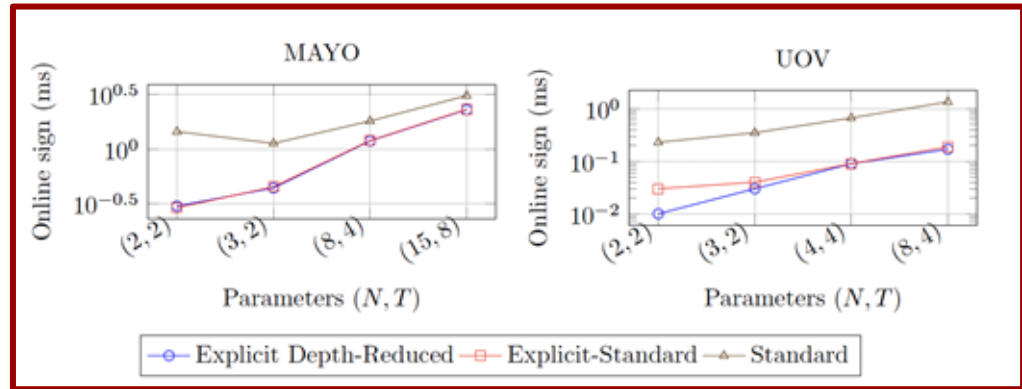


Figure 1: Online signing times, for security level 1.

# On active security

- We augment all shared values with information-theoretic message authentication codes (MACs) in the standard SPDZ manner.

	(N,T)	Standard (3 online rounds)		Explicit-Standard		Explicit Depth-Reduced	
		Non-Active	Active	Non-Active	Active	Non-Active	Active
MAYO 1	(2,2)	1.45 (252.38)	5.73 (1033.36)	0.29 (173.52)	4.99 (2007.09)	0.30 (555.42)	4.82 (5182.61)
	(3,2)	1.13 (329.96)	9.98 (5190.42)	0.45 (276.18)	7.67 (6591.91)	0.44 (790.84)	7.34 (13798.31)
	(8,4)	1.81 (458.39)	23.20 (11658.59)	1.20 (656.26)	19.69 (18649.15)	1.19 (1869.27)	19.58 (38854.61)
	(15,8)	3.10 (1340.48)	48.12 (27422.00)	2.32 (1332.61)	37.25 (43788.12)	2.30 (3415.58)	37.02 (86646.07)
	<i>Comm./party</i>	4.0 (272.6)	-	0.4 (213.4)	-	0.4 (671.9)	-

Table 2: Values in ms or kb for security level 1. Online signing reported and offline signing in (). All experiments were conducted on macOS 15.6.1 (Darwin 24.6.0) running on Apple Silicon (arm64), with an Apple M3 CPU and 24GB of RAM. All benchmarks were executed using Go 1.24.2, and correspond to wall-clock time measured using Go's standard timing facilities. The results are the mean over *100 local signing executions* that completed successfully, where each party performs the protocol rounds sequentially. The runtime was configured with GOMAXPROCS=1: execution of a single logical core.

# Future-upcoming work

- Implementation: our implementation is in Golang:
  - We have not explicitly implemented for constant-time
  - We have started to add hand-written assembly for operations:
    - CLMUL/PMULL in ARMv8
  - We have started network emulation
  - Working on efficient procedure for correlated randomness
- Security:
  - The crux of OV-based schemes is leakage, parameters and efficiency-optimizations → we continue to develop cryptanalysis
  - We have a proof for active security in the dishonest-majority setting:
    - We are expanding for adaptive security → the scheme is based on “hash-and-sign” and UC-secure, which makes it less prone to adaptive attacks (as in Schnorr-based schemes)
- Optimizations:
  - The majority of computational work is spent on matrix-matrix multiplications:
    - We are exploring further matrix-matrix multiplication optimizations

# Acknowledgments

- Thank you François Dupressoir, Daniel Escudero, Hamed Haddadi, Intak Hwang, Seonhong Min for input, work and discussing all of this!



# THANK YOU!

<https://pqvinaigrette.org/>

