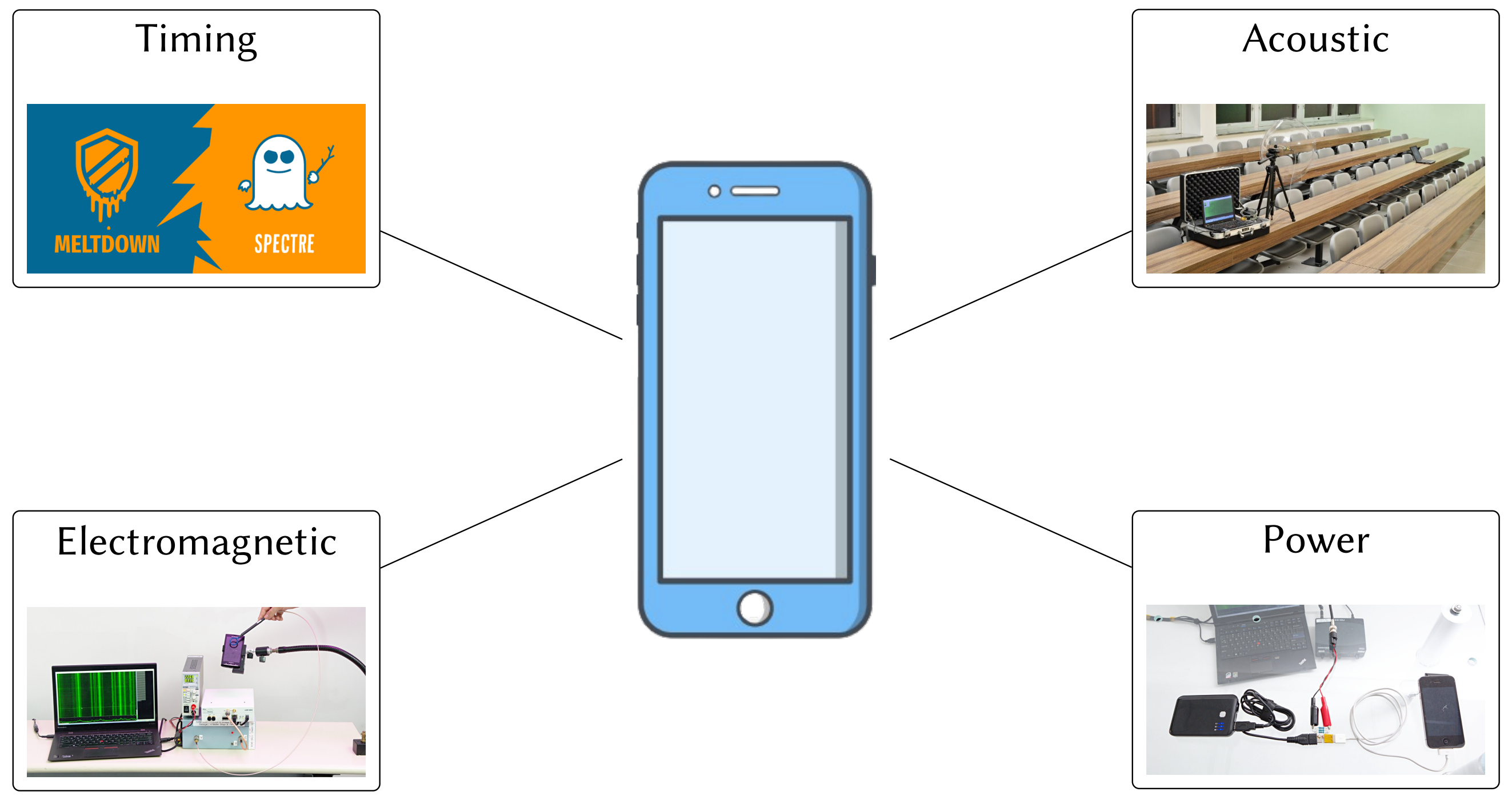
RACCOON: EASY TO MASK, EASY TO THRESHOLDIZE

SIDE-CHANNEL ATTACKS

When deployed on real-world devices, algorithms are vulnerable to **physical leakage**.

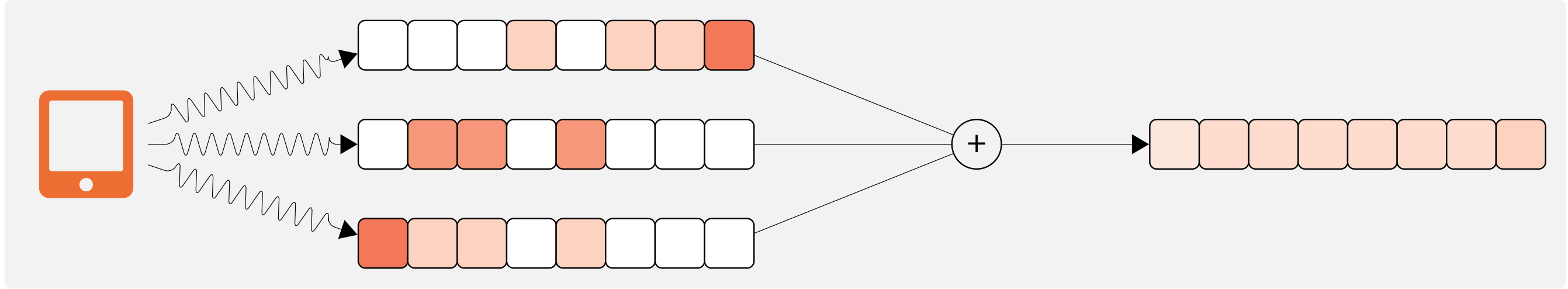


This requires **countermeasures**.

MASKING

Masking is the most common countermeasure against side-channel attacks. It splits every sensitive value x in d shares such that:

$$x_1 + \cdots + x_d = x \bmod q. \tag{1}$$



Masking is a *trade-off* between security and efficiency, parametrized by d:

- **Security.** An attacker needs to correctly guess the value of all shares $x_1, \ldots, x_d$ in order to recover x. This task becomes *exponentially* harder when d increases.
- **Efficiency.** The implementation becomes *polynomially* slower in d.

Type of operation	**Masking overhead**
Linear	$\tilde{O}(d)$
Multiplication	$O(d^2)$
Other	$O(d^2 \log q)$

SO YOU WANT TO MASK DILITHIUM?

Dilithium.Sign(msg, sk)

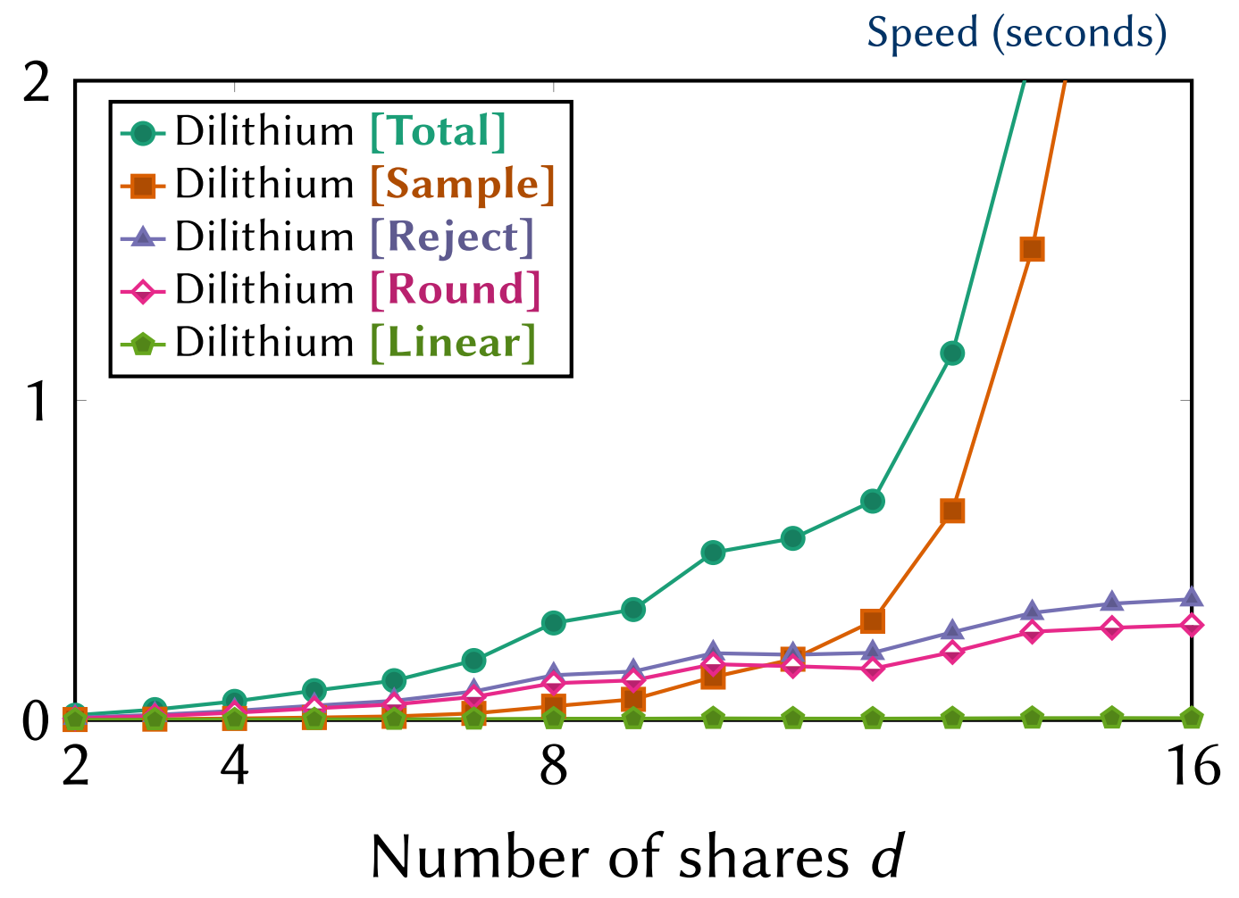
1. Sample a short secret vector $\mathbf{r}$ [Sample]
2. Compute a commitment $\mathbf{w} = \mathbf{A} \cdot \mathbf{r}$ [Linear]
3. Decompose w in its {high, low}-order bits: $\mathbf{w} = \mathbf{w}_0 + 2^k \cdot \mathbf{w}_1$. [Round]
4. Compute a challenge $c = H(\texttt{msg}, \mathbf{w}_1)$
5. Compute a response $\mathbf{z} = c \cdot \mathsf{sk} + \mathbf{r}$ [Linear]
6. If $\mathbf{z}$ is not in a given interval S, go to step 1. [Reject]
7. Output the signature $\texttt{sig} = (c, \mathbf{z})$

Dilithium contains several subroutines that are difficult to mask, marked with **[Round]**, **[Sample]** and **[Reject]**.

When masked, these incur a costly overhead $O(d^2 \log q)$.

Performance numbers from:

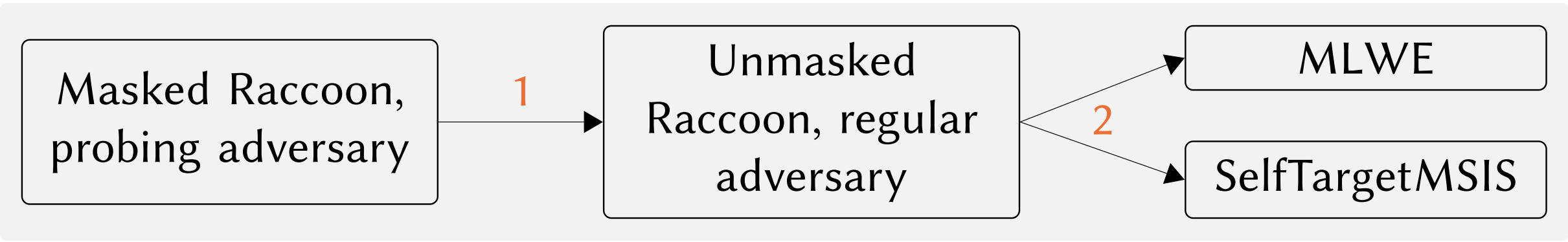
- Coron et al., *Improved Gadgets for the High-Order Masking of Dilithium*, TCHES 2023.



SECURITY

Our security proof is in **two steps (see picture)**:

1. "Masked Raccoon vs probing adversary" ≥ "Unmasked Raccoon vs regular adversary"
2. "Unmasked Raccoon vs regular adversary" ≥ MLWE + SelfTargetMSIS



Step 1. By leveraging the (SNI) composition properties of masked subroutines, we may assume that all probes are inside the `AddRepNoise` subroutine. We then perform a non-black box analysis of `AddRepNoise`. This gives a reduction to unmasked Raccoon against a regular (EUF-CMA) adversary with:

- ✓ The same dimensions
- ✓ The same modulus q
- ✓ A slightly smaller noise (by a factor $\sqrt{2}$)

Step 2. Unmasked Raccoon is EUF-CMA under **standard assumptions**:

- ✓ MLWE for the security of the secret signing key.
- ✓ SelfTargetMSIS for the unforgeability of signatures

Summary. We illustrate the impact of the security reduction on the modulus q.

$$\underbrace{\Omega(1)}_{\text{MLWE (key recovery)}} \overbrace{\|c\|\sqrt{Q \cdot \dim(\mathsf{sk}) \cdot \kappa}}^{\text{Rényi divergence}} \underbrace{\sqrt{2}}_{\text{Probing}} \overbrace{\Omega(1)}^{\text{MSIS (forgery)}}$$

Future improvements. We will replace the Rényi divergence by the recent Hint-MLWE assumption (Kim et al., CRYPTO 2023). It admits a **provable** reduction to MLWE for Gaussians, and a **plausible** one for sums of uniforms. Signatures become **20%** shorter.

$$\underbrace{\Omega(1)}_{\text{MLWE (key recovery)}} \overbrace{\|c\|\sqrt{Q}}^{\text{Hint-MLWE} \gtrsim \text{MLWE}} \underbrace{\sqrt{2}}_{\text{Probing}} \overbrace{\Omega(1)}^{\text{MSIS (forgery)}}$$

RACCOON

Be like water: a leading principle of Raccoon is to adapt our design to the constraints of masking. This led to the following choices:

Masking-friendly noise sampling. We developed a novel procedure called `AddRepNoise` that samples noise in a masked, secure and efficient way.

Raccoon.Sign(msg, sk)

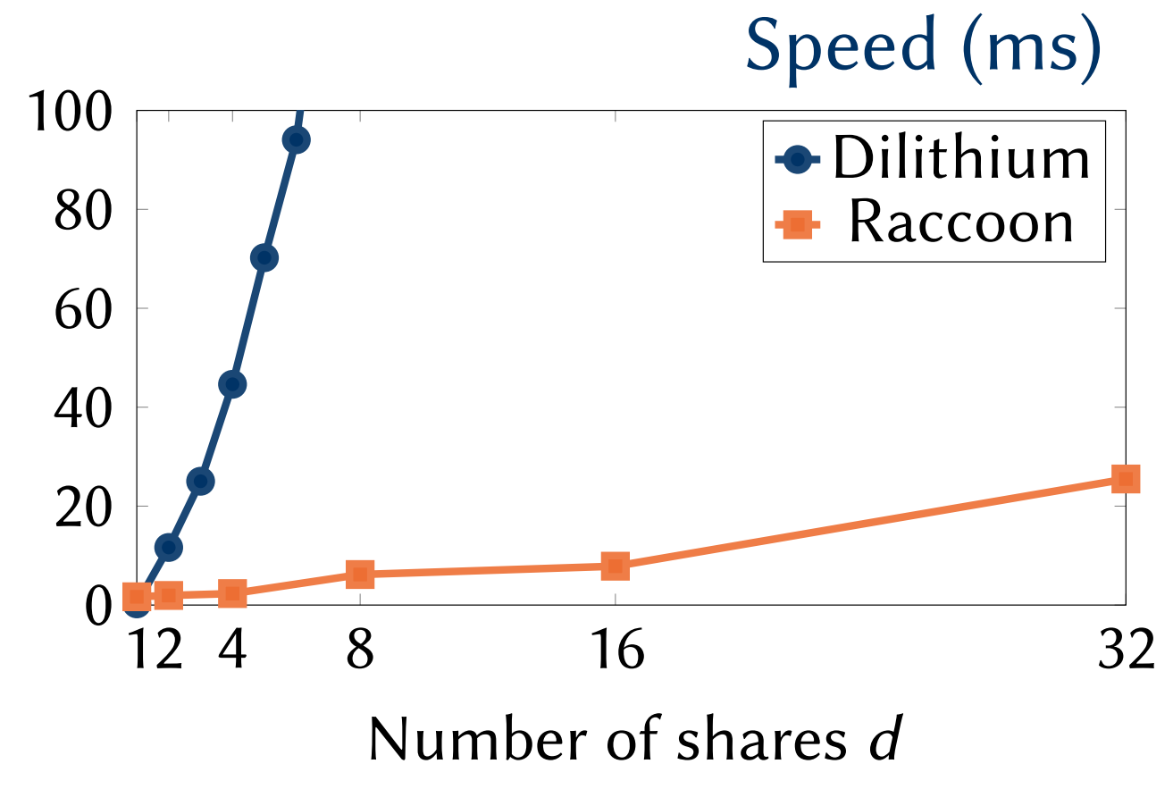
1. Sample a short secret vector $\mathbf{r}$ [Linear]
2. Compute a commitment $\mathbf{w} = \mathbf{A} \cdot \mathbf{r}$ [Linear]
3. Compute a challenge $c = H(\texttt{msg}, \mathbf{w}_1)$
4. Compute a response $\mathbf{z} = c \cdot \mathsf{sk} + \mathbf{r}$ [Linear]
   [No rejection sampling!]
5. Output the signature $\texttt{sig} = (c, \mathbf{z})$

No rejection sampling. Since this operation is costly to mask, we simply remove it and update the parameters.

BLAZING FAST PERFORMANCES

Speed: When masked, Raccoon is significantly faster than Dilithium:

- ⏩ 4 shares: Raccoon is **19** × faster
- ⏭ 16 shares: Raccoon is **536** × faster



Memory: We use new techniques that allow to keep the RAM usage **below 128 Kbytes**, even when masked at order 32. More details in:

- Saarinen and Rossi, *Mask Compression: High-Order Masking on Memory-Constrained Devices*, SAC 2023.

THRESHOLD

Threshold Raccoon

Round 1:

1. Generate uniform masks $\mathbf{m}_{i,j}$
2. Sample short $\mathbf{r}_i$
3. $\mathbf{w}_i = [\mathbf{A}\ \ \mathbf{I}] \cdot \mathbf{r}_i$
4. $\mathsf{com}_i = H_{\mathsf{com}}(\mathbf{w}_i, \texttt{msg}, \mathcal{S})$
5. Broadcast com_i & $\mathbf{m}_i = \sum_j \mathbf{m}_{i,j}$

Round 2: Broadcast $\mathbf{w}_i$ and signature of view of Round 1

Round 3:

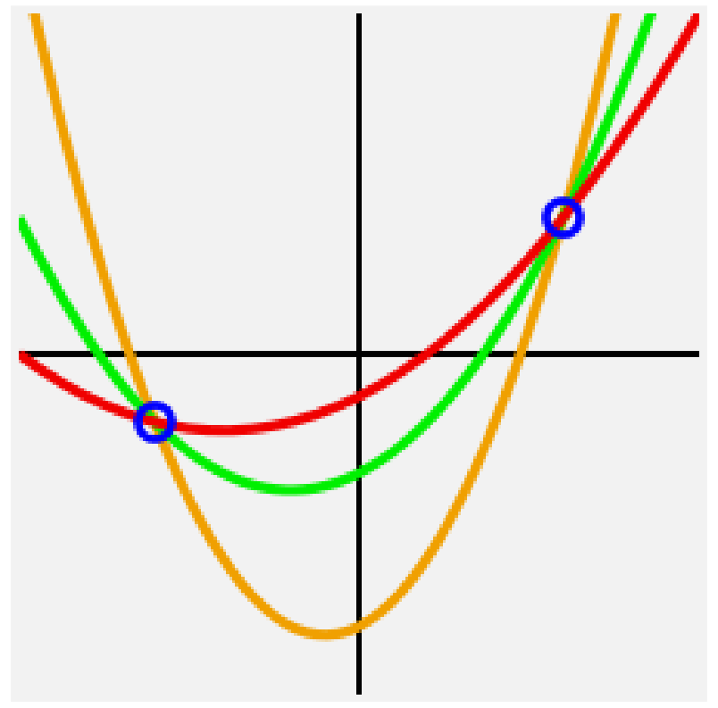
1. $\mathbf{w} = \sum_i \mathbf{w}_i$
2. $c = H(\mathsf{vk}, \texttt{msg}, \mathbf{w})$
3. $\mathbf{m}_i^* = \sum_i \mathbf{m}_{j,i}$
4. $\mathbf{z}_i = \mathbf{r}_i + c \cdot \lambda_i \cdot \mathsf{sk}_i + \mathbf{m}_i^*$
5. Broadcast $\mathbf{z}_i$

Combine: the final signature is

$$\left(c, \mathbf{z} = \sum_{i \in \mathcal{S}} (\mathbf{z}_i - \mathbf{m}_i)\right)$$

Raccoon is **easy to thresholdize**. Threshold Raccoon is similar to Raccoon, with two differences:

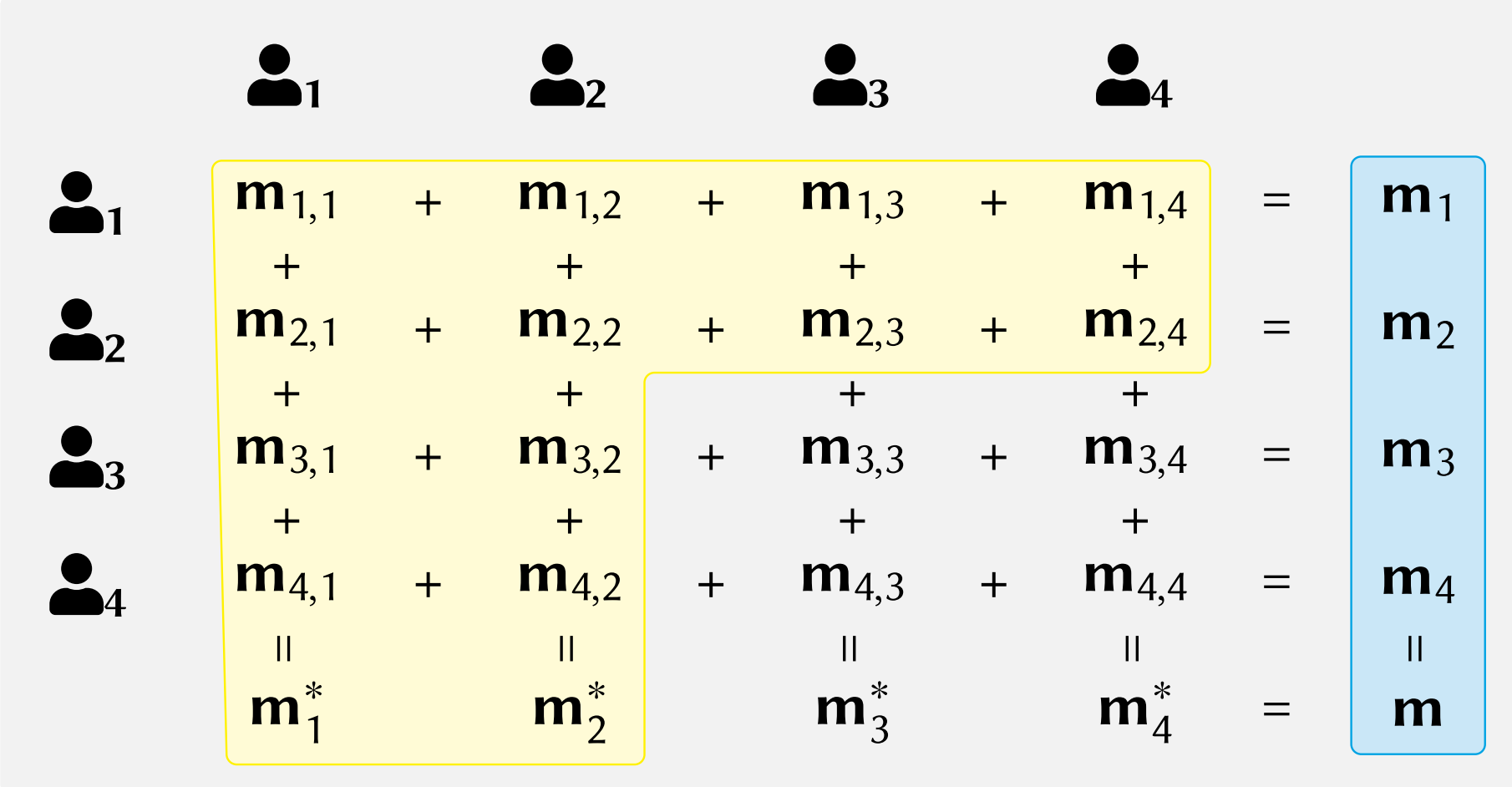
- Additive secret-sharing is replaced by **Shamir secret-sharing** (aka Lagrange interpolation)



- Parties hide their Round 3 responses using **one-time masks** (in yellow). Summing all Round 3 responses magically cancels out all masks.

How do we generate one-time masks? Each pair of users (i, j) share a symmetric key $K_{i,j}$. Passing $K_{i,j}$ into a PRF generate a different one-time mask $\mathbf{m}_{i,j}$ each session.

Example. Below, values in blue are made public, and values in yellow are learned by corrupted parties (1 and 2).



$$\begin{array}{c|ccccccccccc}
 & 1 & & 2 & & 3 & & 4 & & \\
1 & \mathbf{m}_{1,1} & + & \mathbf{m}_{1,2} & + & \mathbf{m}_{1,3} & + & \mathbf{m}_{1,4} & = & \mathbf{m}_1 \\
 & + & & + & & + & & + & & \\
2 & \mathbf{m}_{2,1} & + & \mathbf{m}_{2,2} & + & \mathbf{m}_{2,3} & + & \mathbf{m}_{2,4} & = & \mathbf{m}_2 \\
 & + & & + & & + & & + & & \\
3 & \mathbf{m}_{3,1} & + & \mathbf{m}_{3,2} & + & \mathbf{m}_{3,3} & + & \mathbf{m}_{3,4} & = & \mathbf{m}_3 \\
 & + & & + & & + & & + & & \\
4 & \mathbf{m}_{4,1} & + & \mathbf{m}_{4,2} & + & \mathbf{m}_{4,3} & + & \mathbf{m}_{4,4} & = & \mathbf{m}_4 \\
 & \parallel & & \parallel & & \parallel & & \parallel & & \parallel \\
 & \mathbf{m}_1^* & & \mathbf{m}_2^* & & \mathbf{m}_3^* & & \mathbf{m}_4^* & = & \mathbf{m}
\end{array}$$

Further reading:

- del Pino et al. *Threshold Raccoon: Practical Threshold Signatures from Standard Lattice Assumptions*, EUROCRYPT 2024.
- Espitau, Katsumata and Takemure. *Two-Round Threshold Signature from Algebraic One-More Learning with Errors*, ePrint 2024/496.

Brought to you by the Raccoon team: Rafaël del Pino, Thomas Espitau, Shuichi Katsumata, Mary Maller, Fabrice Mouhartem, Thomas Prest, Mélissa Rossi and Markku-Juhani Saarinen. `https://raccoonfamily.org`