# Masking-Friendly Lattice Schemes and Lattice-Friendly Masking Schemes

Thomas Prest (joint work with/by Rafael del Pino, Mélissa Rossi, Markku Saarinen & Shuichi Katsumata)

PQ SHIELD

April 3, 2025

# Why this Talk?

# Why this talk?

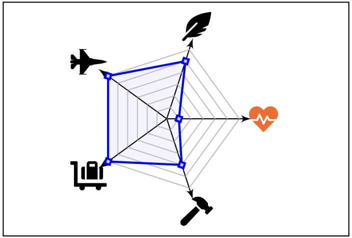
**Observation:** masking and post-quantum standards have poor compatibility.

1. Can we design lattice-based cryptosystems more suitable for masking?
2. Can we design masking schemes more suitable for lattice cryptosystems?

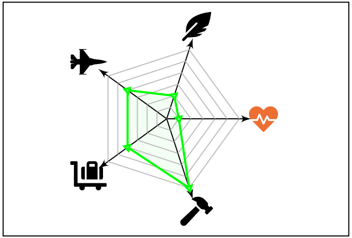
# Lattice Schemes & Masking

# Motivation

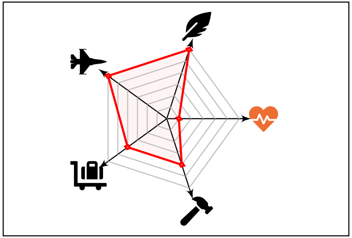
ML-DSA



SLH-DSA



FN-DSA



NIST PQC standards, selected in 2022, strike a balance between several criteria.

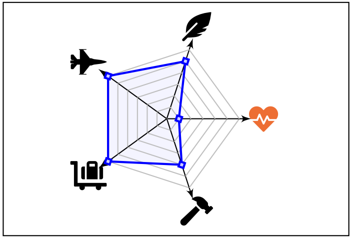
But what about :

- **Side-channel protection**?

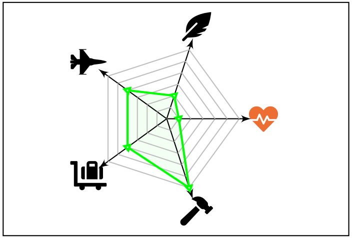
Size Speed Portability Assumptions SCA protection

# Motivation

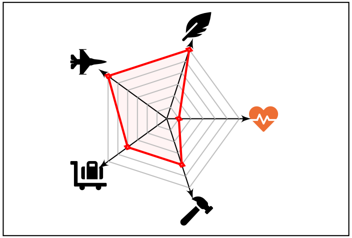
ML-DSA



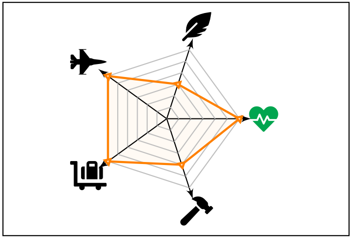
SLH-DSA



FN-DSA



Raccoon (2023)



Size   Speed   Portability   Assumptions   SCA protection

# Example: SCA on Falcon ($\rightarrow$ FN-DSA)

In Falcon, a signature $\mathsf{sig}$ is distributed as a Gaussian.
The signing key $\mathsf{sk}$ should remain private.
The power consumption leaks information about the dot product $\langle \mathsf{sig}, \mathsf{sk} \rangle$, or $\mathsf{sk}$ itself.

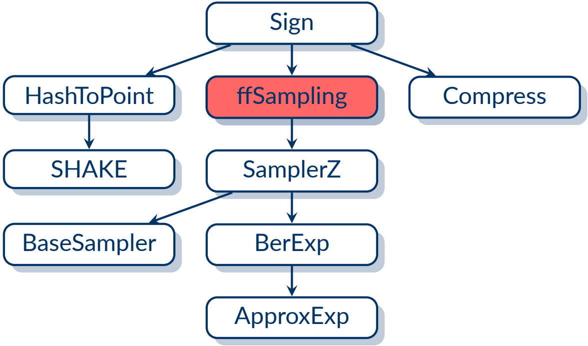
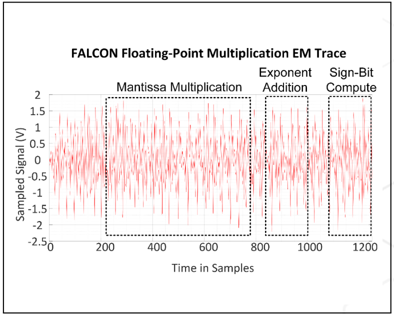



Figure 1: Flowchart of the signature




Learning $\mathsf{sk}$ directly

[1] *FALCON Down: Breaking FALCON Post-Quantum Signature Scheme through Side-Channel Attacks* [KA21]

# Example: SCA on Falcon ($\rightarrow$ FN-DSA)

PQ SHIELD

In Falcon, a signature sig is distributed as a Gaussian.
The signing key sk should remain private.
The power consumption leaks information about the dot product $\langle \mathsf{sig}, \mathsf{sk} \rangle$, or sk itself.

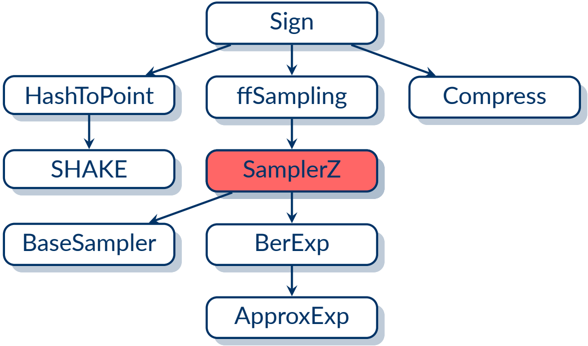
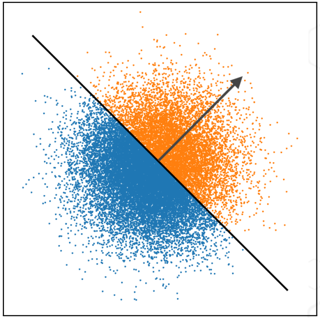



Figure 1: Flowchart of the signature



Filtering $\langle \mathsf{sig}, \mathsf{sk} \rangle > 0$

---

[2] *Improved Power Analysis Attacks on Falcon* [ZLYW23]

# Masking Dilithium: what to expect

PQSHIELD

**Dilithium-Sign**

1. Sample $\mathbf{y} \leftarrow \text{Uniform}(S)$
2. $\mathbf{w} := \mathbf{A} \cdot \mathbf{y}$
3. $\mathbf{w}_0, \mathbf{w}_1 := \text{Decompose}(\mathbf{w})$
4. $c := H(\mathbf{w}_1, \text{msg})$
5. $\mathbf{z} := \mathbf{y} + \mathbf{s}_1 \cdot c$
6. $\tilde{\mathbf{r}} := \mathbf{w}_0 - \mathbf{s}_0 \cdot c$
7. If $\|\mathbf{z}\|_\infty$ or $\|\tilde{\mathbf{r}}\|_\infty$ are too large, goto 1
8. $\mathbf{h} := \mathbf{w}_1 - \lfloor \mathbf{A} \cdot \mathbf{z} - \mathbf{t} \cdot c \rceil_k$
9. Output sig $= (c, \mathbf{z}, \mathbf{h})$

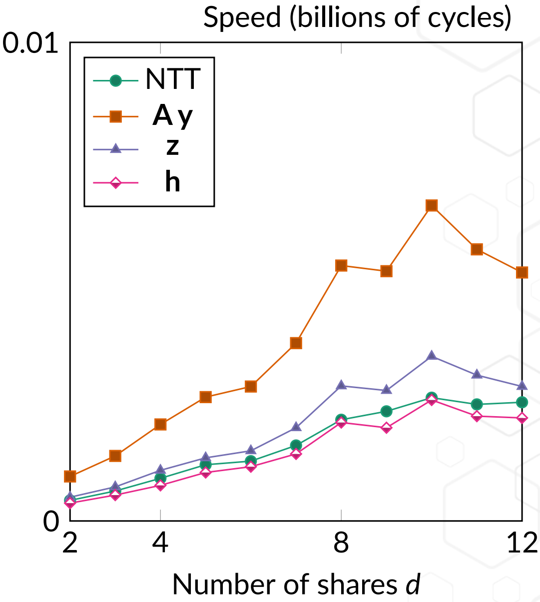
Observations:

- Some operations don't need to be masked (or conjectured to)
- Some operations are linear and are therefore easy to mask
- Three operations require mask conversions (overhead: $O(d^2 \log q)$):
  - 1 Sampling
  - 3 Decomposition
  - 6 Rejection sampling

# Masked Dilithium [CGTZ23] - only fast ops

**Dilithium-Sign**

1. Sample $\mathbf{y} \leftarrow S$
2. $\mathbf{w} := \mathbf{A} \cdot \mathbf{y}$ $\triangleright\ \tilde{O}(d)$
3. $\mathbf{w}_0, \mathbf{w}_1 := \mathrm{Decompose}(\mathbf{w})$
4. $c := H(\mathbf{w}_1, \mathrm{msg})$ $\triangleright$ No mask
5. $\mathbf{z} := \mathbf{y} + \mathbf{s}_1\, c$ $\triangleright\ \tilde{O}(d)$
6. $\tilde{\mathbf{r}} := \mathbf{w}_0 - \mathbf{s}_0 \cdot c$ $\triangleright\ \tilde{O}(d)$
7. If $\|\mathbf{z}\|_\infty$ or $\|\tilde{\mathbf{r}}\|_\infty$ are too large, goto 1
8. $\mathbf{h} := \mathbf{w}_1 - \lfloor \mathbf{A} \cdot \mathbf{z} - \mathbf{t} \cdot c \rceil_k$ $\triangleright$ No mask
9. Output sig $= (c, \mathbf{z}, \mathbf{h})$

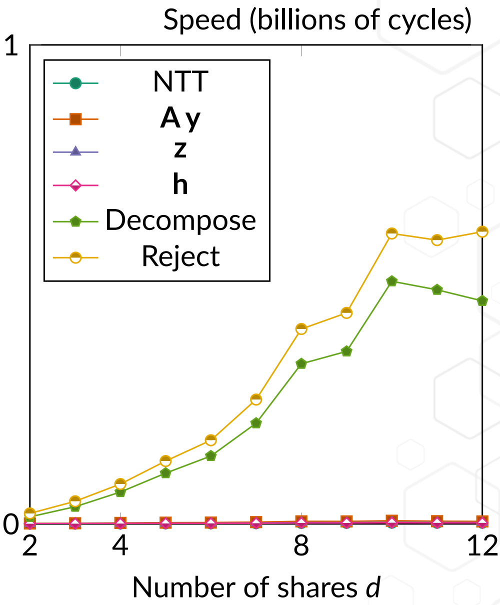


# Masked Dilithium [CGTZ23] - round & reject

PQSHIELD

**Dilithium-Sign**

1. Sample $\mathbf{y} \leftarrow S$
2. $\mathbf{w} := \mathbf{A} \cdot \mathbf{y}$ $\quad \triangleright \tilde{O}(d)$
3. $\mathbf{w}_0, \mathbf{w}_1 := \mathrm{Decompose}(\mathbf{w})$ $\quad \triangleright O(d^2 \log q)$
4. $c := H(\mathbf{w}_1, \mathrm{msg})$ $\quad \triangleright$ No mask
5. $\mathbf{z} := \mathbf{y} + \mathbf{s}_1\, c$ $\quad \triangleright \tilde{O}(d)$
6. $\tilde{\mathbf{r}} := \mathbf{w}_0 - \mathbf{s}_0 \cdot c$ $\quad \triangleright \tilde{O}(d)$
7. If $\|\mathbf{z}\|_\infty$ or $\|\tilde{\mathbf{r}}\|_\infty$ are too large, goto 1 $\quad \triangleright O(d^2 \log q)$
8. $\mathbf{h} := \mathbf{w}_1 - \lfloor \mathbf{A} \cdot \mathbf{z} - \mathbf{t} \cdot c \rceil_k$ $\quad \triangleright$ No mask
9. Output sig $= (c, \mathbf{z}, \mathbf{h})$

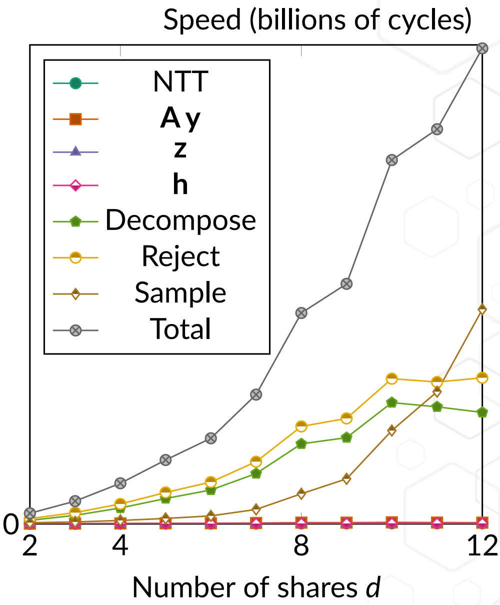


# Masked Dilithium [CGTZ23] - sampling

PQ SHIELD

**Dilithium-Sign**

1. Sample $\mathbf{y} \leftarrow S$ $\quad \triangleright\ O(d^2 \log q)$
2. $\mathbf{w} := \mathbf{A} \cdot \mathbf{y}$ $\quad \triangleright\ \tilde{O}(d)$
3. $\mathbf{w}_0, \mathbf{w}_1 := \text{Decompose}(\mathbf{w})$ $\quad \triangleright\ O(d^2 \log q)$
4. $c := H(\mathbf{w}_1, \text{msg})$ $\quad \triangleright$ No mask
5. $\mathbf{z} := \mathbf{y} + \mathbf{s}_1\, c$ $\quad \triangleright\ \tilde{O}(d)$
6. $\tilde{\mathbf{r}} := \mathbf{w}_0 - \mathbf{s}_0 \cdot c$ $\quad \triangleright\ \tilde{O}(d)$
7. If $\|\mathbf{z}\|_\infty$ or $\|\tilde{\mathbf{r}}\|_\infty$ are too large, goto 1 $\quad \triangleright\ O(d^2 \log q)$
8. $\mathbf{h} := \mathbf{w}_1 - \lfloor \mathbf{A} \cdot \mathbf{z} - \mathbf{t} \cdot c \rceil_k$ $\quad \triangleright$ No mask
9. Output sig $= (c, \mathbf{z}, \mathbf{h})$



# Raccoon

# Raccoon = Schnorr over lattices

### Raccoon.Keygen() $\to$ sk, vk

1. $\mathrm{vk} = \begin{bmatrix}\mathbf{A} & 1\end{bmatrix} \cdot \mathrm{sk}$, for sk short.

### Raccoon.Sign(sk, msg) $\to$ sig

1. Sample a short $\mathbf{r}$
2. $\mathbf{w} = \begin{bmatrix}\mathbf{A} & 1\end{bmatrix} \cdot \mathbf{r}$
3. $c = H(\mathbf{w}, \mathrm{msg})$
4. $\mathbf{z} = \mathbf{r} + c \cdot \mathrm{sk}$
5. Output $\mathrm{sig} = (c, \mathbf{z})$

### Raccoon.Verify(vk, msg, sig)

1. $\mathbf{w}' = \begin{bmatrix}\mathbf{A} & 1\end{bmatrix} \cdot \mathbf{z} - c \cdot \mathrm{vk}$
2. Assert $H(\mathbf{w}', \mathrm{msg}) = c$
3. Assert $\mathbf{z}$ is short

### Schnorr.Keygen() $\to$ sk, vk

1. $\mathrm{vk} = g^{\mathrm{sk}}$, for sk uniform.

### Schnorr.Sign(sk, msg) $\to$ sig

1. Sample $r$
2. $w = g^r$
3. $c = H(w, \mathrm{msg})$
4. $z = r + c \cdot \mathrm{sk}$
5. Output $\mathrm{sig} = (c, z)$

### Schnorr.Verify(vk, msg, sig)

1. $w' = g^z \cdot \mathrm{vk}^{-c}$
2. Assert $H(\mathbf{w}', \mathrm{msg}) = c$

# Security of Raccoon

**Raccoon.Keygen**$() \to \mathsf{sk}, \mathsf{vk}$

1. $\mathsf{vk} = [\mathbf{A} \quad 1] \cdot \mathsf{sk}$, for sk short.

**Raccoon.Sign**$(\mathsf{sk}, \mathsf{msg}) \to \mathsf{sig}$

1. Sample a short $\mathbf{r}$
2. $\mathbf{w} = [\mathbf{A} \quad 1] \cdot \mathbf{r}$
3. $c = H(\mathbf{w}, \mathsf{msg})$
4. $\mathbf{z} = \mathbf{r} + c \cdot \mathsf{sk}$
5. Output $\mathsf{sig} = (c, \mathbf{z})$

**Raccoon.Verify**$(\mathsf{vk}, \mathsf{msg}, \mathsf{sig})$

1. $\mathbf{w}' = [\mathbf{A} \quad 1] \cdot \mathbf{z} - c \cdot \mathsf{vk}$
2. Assert $H(\mathbf{w}', \mathsf{msg}) = c$
3. Assert $\mathbf{z}$ is short

**Security:** Raccoon is EUF-CMA assuming:

1. **Hint-MLWE [KLSS23]** (next slide)
   - Implied by lack of rejection sampling
   - Ensures uniformity of the public key
2. **Self-target MSIS [KLS18]**
   - Unforgeability

**Rounding:** we can round vk and $\mathbf{w}$:

- ✓ Reduces the size of vk and sig
- ✓ Zero impact on Hint-MLWE
- ✓ Minor impact on unforgeability
- ✓ Not a sensitive information
  - Will not need to be masked

# Hint-MLWE?

**(Hint-)MLWE [KLSS23]**

It is difficult to distinguish both distributions:

$$\left\{(\mathbf{A},\mathbf{b}) \mid \mathbf{A} \leftarrow \mathcal{R}_q^{k\times\ell}, \mathrm{sk} \leftarrow \chi_{\mathrm{sk}}, \mathbf{b} := \begin{bmatrix}\mathbf{A} & \mathbf{I}\end{bmatrix} \cdot \mathrm{sk}\right\}$$

$$\left\{(\mathbf{A},\mathbf{b}) \mid \mathbf{A} \leftarrow \mathcal{R}_q^{k\times\ell}, \mathrm{sk} \leftarrow \chi_{\mathrm{sk}}, \mathbf{b} \leftarrow \mathcal{R}_q^{k}\right\}$$

In Hint-MLWE, the adversary is additionally given Q "hints" of the shape:

$$(c_i, \mathbf{z}_i \leftarrow c_i \cdot \mathrm{sk} + \mathbf{r}_i), \quad \text{where } c_i \leftarrow \mathcal{C}, \mathbf{r}_i \leftarrow \chi_{\mathbf{r}}$$

# Hint-MLWE?

## (Hint-)MLWE [KLSS23]

It is difficult to distinguish both distributions:

$$\left\{(\mathbf{A}, \mathbf{b}) | \mathbf{A} \leftarrow \mathcal{R}_q^{k\times\ell}, \mathsf{sk} \leftarrow \chi_{\mathsf{sk}}, \mathbf{b} := \begin{bmatrix}\mathbf{A} & \mathbf{I}\end{bmatrix} \cdot \mathsf{sk}\right\}$$

$$\left\{(\mathbf{A}, \mathbf{b}) | \mathbf{A} \leftarrow \mathcal{R}_q^{k\times\ell}, \mathsf{sk} \leftarrow \chi_{\mathsf{sk}}, \mathbf{b} \leftarrow \mathcal{R}_q^{k}\right\}$$

In Hint-MLWE, the adversary is additionally given Q "hints" of the shape:

$$(c_i, \mathbf{z}_i \leftarrow c_i \cdot \mathsf{sk} + \mathbf{r}_i), \quad \text{where } c_i \leftarrow \mathcal{C}, \mathbf{r}_i \leftarrow \chi_{\mathbf{r}}$$

## Attack on Hint-MLWE

Assume $\forall i \in [Q], \|c_i\|^2 = \omega$. If we note $c^*(x) = c(x^{-1})$, we can recover sk by constructing this accumulator:

$$\begin{aligned} \mathsf{acc} &= \sum_i c_i^* \cdot \mathbf{z}_i \\ &= \sum_i c_i^* c_i \cdot \mathsf{sk} + \sum_i c_i^* \cdot \mathbf{r}_i \\ &\approx Q \cdot \omega \cdot \mathsf{sk} + O(\sqrt{Q \cdot \omega} \cdot \|\mathbf{r}\|) \end{aligned}$$

If $\|\mathbf{r}\| = o(\sqrt{Q \cdot \omega})$, rounding acc to the closest multiple of $Q \cdot \omega$ gives sk.

# Hint-MLWE?

## (Hint-)MLWE [KLSS23]

It is difficult to distinguish both distributions:

$$\left\{(\mathbf{A},\mathbf{b}) \middle| \mathbf{A} \leftarrow \mathcal{R}_q^{k\times\ell}, \mathsf{sk} \leftarrow \chi_{\mathsf{sk}}, \mathbf{b} := \begin{bmatrix}\mathbf{A} & \mathbf{I}\end{bmatrix} \cdot \mathsf{sk}\right\}$$

$$\left\{(\mathbf{A},\mathbf{b}) \middle| \mathbf{A} \leftarrow \mathcal{R}_q^{k\times\ell}, \mathsf{sk} \leftarrow \chi_{\mathsf{sk}}, \mathbf{b} \leftarrow \mathcal{R}_q^{k}\right\}$$

In Hint-MLWE, the adversary is additionally given Q “hints” of the shape:

$$(c_i, \mathbf{z}_i \leftarrow c_i \cdot \mathsf{sk} + \mathbf{r}_i), \quad \text{where } c_i \leftarrow \mathcal{C}, \mathbf{r}_i \leftarrow \chi_{\mathbf{r}}$$

## Attack on Hint-MLWE

Assume $\forall i \in [Q], \|c_i\|^2 = \omega$. If we note $c^*(x) = c(x^{-1})$, we can recover sk by constructing this accumulator:

$$\begin{aligned} \mathsf{acc} &= \sum_i c_i^* \cdot \mathbf{z}_i \\ &= \sum_i c_i^* c_i \cdot \mathsf{sk} + \sum_i c_i^* \cdot \mathbf{r}_i \\ &\approx Q \cdot \omega \cdot \mathsf{sk} + O(\sqrt{Q\cdot\omega}\cdot\|\mathbf{r}\|) \end{aligned}$$

If $\|\mathbf{r}\| = o(\sqrt{Q\cdot\omega})$, rounding acc to the closest multiple of $Q\cdot\omega$ gives sk.

## Security reduction, simplified [KLSS23, DKM⁺24]

If $\mathbf{s}$ and $\mathbf{r}_i$ are sampled from gaussians of standard deviation $\sigma_{\mathsf{sk}}$ and $\sigma_{\mathbf{r}}$, then:

$$\text{Hint-MLWE}_{\mathcal{R}_q,k,\ell,\sigma_{\mathsf{sk}},\sigma_{\mathbf{r}},Q} \geq \text{MLWE}_{\mathcal{R}_q,k,\ell,\sigma_0}, \quad \text{where} \quad \frac{1}{\sigma_0^2} \approx 2\left(\frac{1}{\sigma_{\mathsf{sk}}^2} + \frac{Q\cdot\omega}{\sigma_{\mathbf{r}}^2}\right) \tag{1}$$

# From unmasked to masked Raccoon

**Raccoon.Sign**$(\mathsf{sk}, \mathsf{msg}) \to \mathsf{sig}$

1. Sample a short $\mathbf{r}$
2. $\mathbf{w} = \begin{bmatrix}\mathbf{A} & 1\end{bmatrix} \cdot \mathbf{r}$
3. $c = H(\mathbf{w}, \mathsf{msg})$
4. $\mathbf{z} = \mathbf{r} + c \cdot \mathsf{sk}$
5. Output $\mathsf{sig} = (c, \mathbf{z})$

Starting point is "Schnorr over lattices":

- ✓ No Rejection sampling
- ✓ Steps 2 and 4 are easy to mask
- ✓ Steps 3 does not need to be masked (no conjecture!)
- ? What about Sampling (step 1)?

# From unmasked to masked Raccoon

PQ SHIELD

**Raccoon.Sign**(sk, msg) $\to$ sig

1. Sample a short $\mathbf{r}$
2. $\mathbf{w} = \begin{bmatrix} \mathbf{A} & 1 \end{bmatrix} \cdot \mathbf{r}$
3. $c = H(\mathbf{w}, \mathrm{msg})$
4. $\mathbf{z} = \mathbf{r} + c \cdot \mathrm{sk}$
5. Output sig $= (c, \mathbf{z})$

MaskSign($[\![\mathrm{sk}]\!]$, vk, msg) $\to$ sig

1. $[\![\mathbf{r}]\!] = [\![\mathbf{0}]\!]$
2. For $i \in [\mathrm{rep}]$:
   1. $[\![\mathbf{r}_i]\!] = (\mathbf{r}_{i,1}, \ldots, \mathbf{r}_{i,d}) \leftarrow \chi_{\mathbf{r}}^d$
   2. $[\![\mathbf{r}]\!] = [\![\mathbf{r}]\!] + [\![\mathbf{r}_i]\!]$
   3. Refresh($[\![\mathbf{r}]\!]$)
3. $[\![\mathbf{w}]\!] = \begin{bmatrix} \mathbf{A} & \mathbf{I} \end{bmatrix} \cdot [\![\mathbf{r}]\!]$
4. Refresh($[\![\mathbf{w}]\!]$)
5. $\mathbf{w} = \mathrm{Decode}([\![\mathbf{w}]\!])$
6. $c = \mathrm{H}(\mathbf{w}, \mathrm{msg}, \mathrm{vk})$
7. $[\![\mathbf{z}]\!] = [\![\mathrm{sk}]\!] \cdot c + [\![\mathbf{r}]\!]$
8. Refresh($[\![\mathbf{z}]\!], [\![\mathrm{sk}]\!]$)
9. $\mathbf{z} = \mathrm{Decode}([\![\mathbf{z}]\!])$
10. Output sig $= (c, \mathbf{z})$

# From unmasked to masked Raccoon

PQ SHIELD

**Raccoon.Sign**(sk, msg) $\to$ sig

1. Sample a short $\mathbf{r}$
2. $\mathbf{w} = \begin{bmatrix} \mathbf{A} & 1 \end{bmatrix} \cdot \mathbf{r}$
3. $c = H(\mathbf{w}, \mathrm{msg})$
4. $\mathbf{z} = \mathbf{r} + c \cdot \mathrm{sk}$
5. Output $\mathrm{sig} = (c, \mathbf{z})$

We note $[\![x]\!]$ a $d$-sharing of $x$.

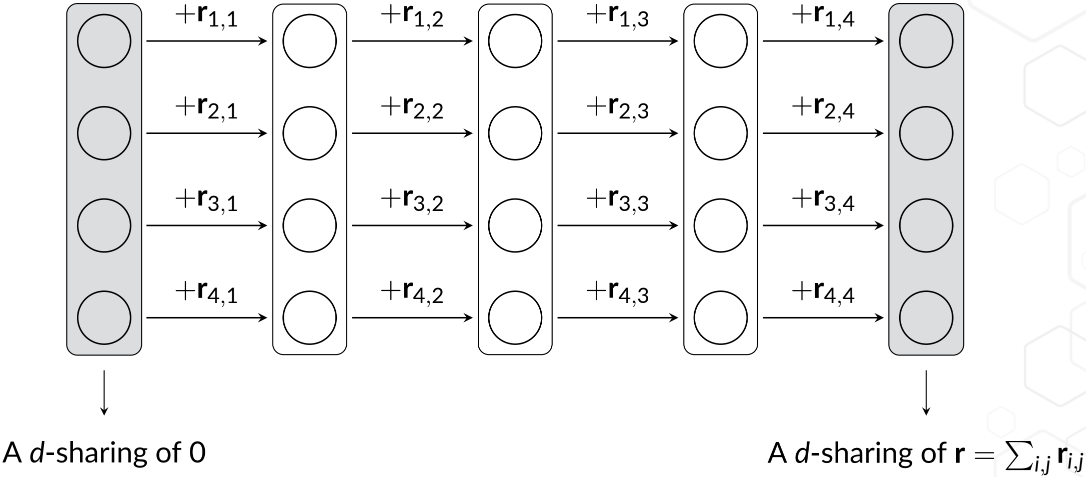
- AddRepNoise in lime green
  - A $t$-probing adversary learns at most $t$ of the $(d \cdot \mathrm{rep})$ values $\mathbf{r}_{i,j}$
  - Formal analysis in [EEN+24]
- Refresh is useful for:
  - Concrete security
  - Composing gadgets (SNI)
  - Moving probes around (SNI)

All operations take time $O(d \log d)$.

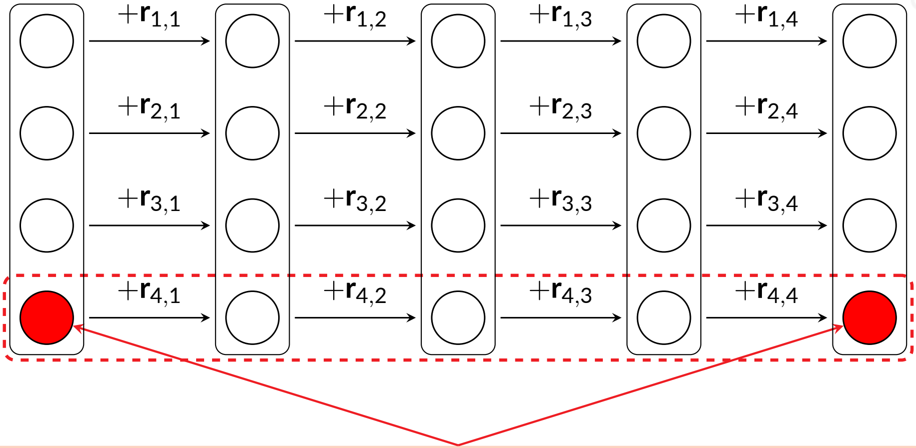
MaskSign($[\![\mathrm{sk}]\!]$, vk, msg) $\to$ sig

1. $[\![\mathbf{r}]\!] = [\![\mathbf{0}]\!]$
2. For $i \in [\mathrm{rep}]$:
   1. $[\![\mathbf{r}_i]\!] = (\mathbf{r}_{i,1}, \ldots, \mathbf{r}_{i,d}) \leftarrow \chi_{\mathbf{r}}^d$
   2. $[\![\mathbf{r}]\!] = [\![\mathbf{r}]\!] + [\![\mathbf{r}_i]\!]$
   3. $\mathrm{Refresh}([\![\mathbf{r}]\!])$
3. $[\![\mathbf{w}]\!] = \begin{bmatrix} \mathbf{A} & \mathbf{I} \end{bmatrix} \cdot [\![\mathbf{r}]\!]$
4. $\mathrm{Refresh}([\![\mathbf{w}]\!])$
5. $\mathbf{w} = \mathrm{Decode}([\![\mathbf{w}]\!])$
6. $c = \mathrm{H}(\mathbf{w}, \mathrm{msg}, \mathrm{vk})$
7. $[\![\mathbf{z}]\!] = [\![\mathrm{sk}]\!] \cdot c + [\![\mathbf{r}]\!]$
8. $\mathrm{Refresh}([\![\mathbf{z}]\!], [\![\mathrm{sk}]\!])$
9. $\mathbf{z} = \mathrm{Decode}([\![\mathbf{z}]\!])$
10. Output $\mathrm{sig} = (c, \mathbf{z})$

# What happens inside AddRepNoise?

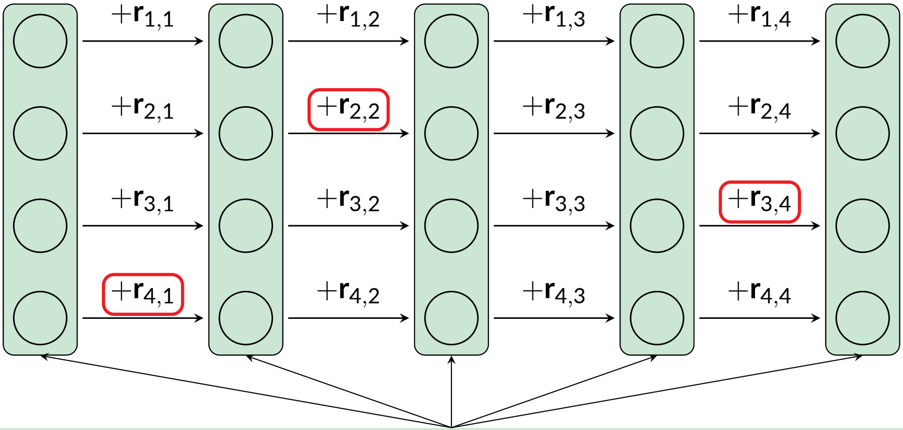


# What happens inside AddRepNoise?



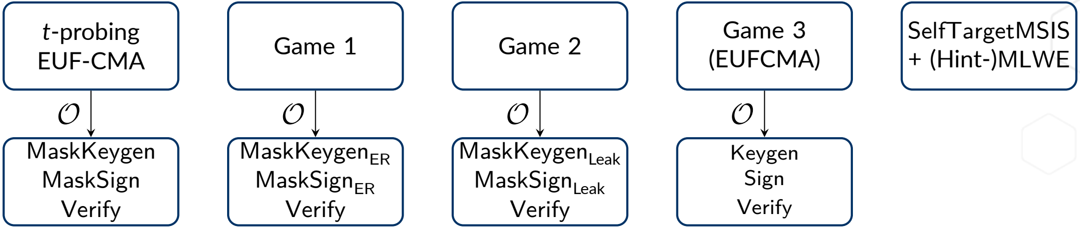

Without Refresh, a probing adversary could learn the sum of $T$ random in 2 probes.

# What happens inside AddRepNoise?

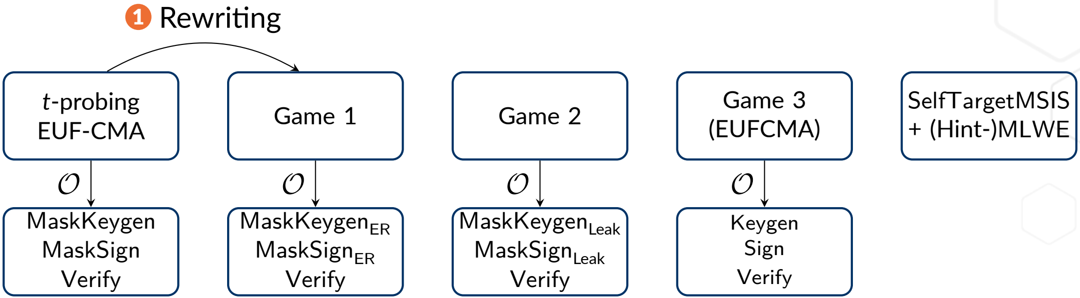



**Solution:** add refresh gadgets to separate the algorithm in independent layers
Now a probing adversary learns at most (the sum of) $t$ short noises.

# Security proof outline [dPKPR24] (simplified)

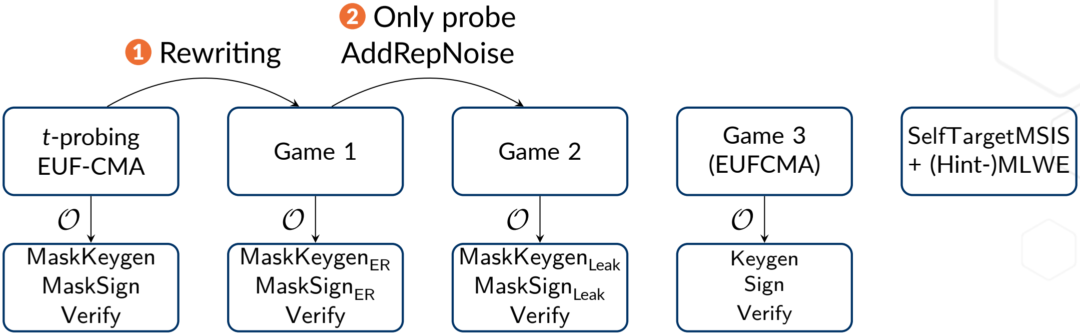


# Security proof outline [dPKPR24] (simplified)



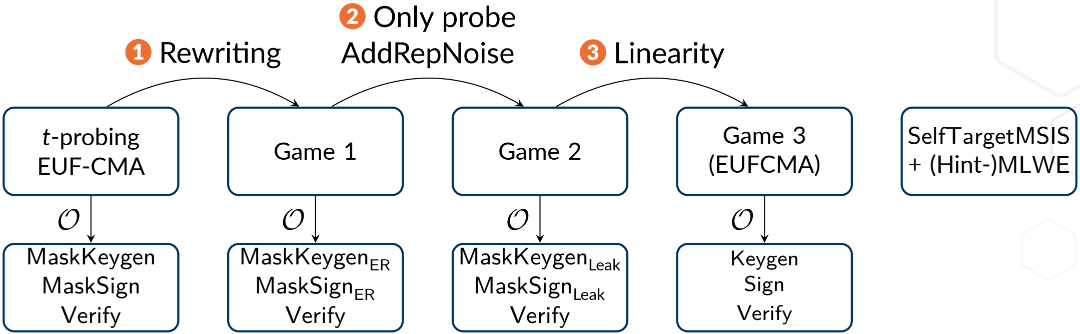

1 **Rewriting:** make randomness explicit as input

# Security proof outline [dPKPR24] (simplified)




1. **Rewriting:** make randomness explicit as input
2. **SNI(u) property:** move all probes to AddRepNoise randomness

# Security proof outline [dPKPR24] (simplified)

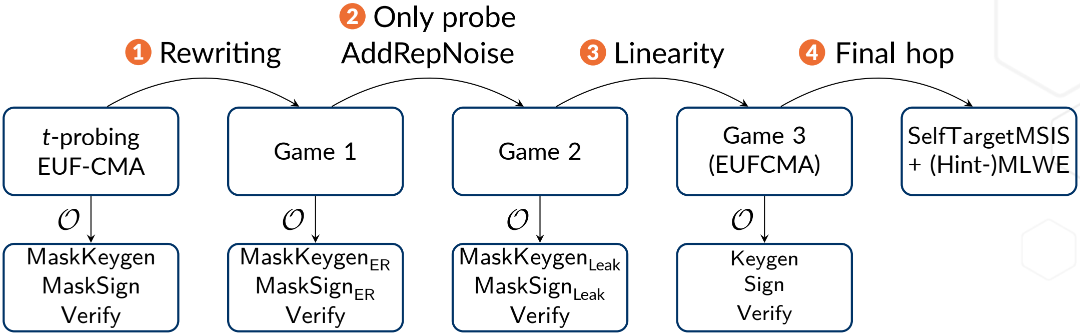



1. **Rewriting:** make randomness explicit as input
2. **SNI(u) property:** move all probes to AddRepNoise randomness
3. **Linearity:** we argue that we can can simulate Game 2 from Game 3

**Game 2:** $\mathbf{w} = [\mathbf{A} \quad \mathbf{I}] \cdot \mathbf{r}$ where $\mathbf{r} = \sum_{i \in [d \cdot rep]} \mathbf{r}_i$ and we leak $(\mathbf{r}_i)_{i \in S}$ for $|S| = t$

**Game 3:** $\mathbf{w} = [\mathbf{A} \quad \mathbf{I}] \cdot \mathbf{r}'$ where $\mathbf{r}' = \sum_{i \in [d \cdot rep - t]} \mathbf{r}_i$

# Security proof outline [dPKPR24] (simplified)




1. **Rewriting:** make randomness explicit as input
2. **SNI(u) property:** move all probes to AddRepNoise randomness
3. **Linearity:** we argue that we can can simulate Game 2 from Game 3

   **Game 2:** $\mathbf{w} = [\mathbf{A} \quad \mathbf{I}] \cdot \mathbf{r}$ where $\mathbf{r} = \sum_{i \in [d \cdot rep]} \mathbf{r}_i$ and we leak $(\mathbf{r}_i)_{i \in S}$ for $|S| = t$

   **Game 3:** $\mathbf{w} = [\mathbf{A} \quad \mathbf{I}] \cdot \mathbf{r}'$ where $\mathbf{r}' = \sum_{i \in [d \cdot rep - t]} \mathbf{r}_i$
4. **Final hop:** {EUFCMA of Raccoon} $\geq$ {SelfTargetMSIS + (Hint-)MLWE }

# Parameter selection and the modulus $q$.

Signature sizes are quadratic in $\log q$ (trust me), so we want to minimize $q$.

| Method | Modulus $q$ (logarithmic view) |
|---|---|
| ML-DSA | $\overbrace{\underbrace{\boxed{\sigma(\mathrm{sk})}}_{\text{MLWE}}\,\underbrace{\boxed{\\|c\\|_1 \cdot \dim(\mathrm{sk})}}_{\text{Rejection sampling}}}^{\sigma(\mathrm{sig})}\,\overbrace{\boxed{\Omega(1)}}^{\text{MSIS (forgery)}}$ |
| Raccoon, Smooth Rényi [Proven] | $\overbrace{\underbrace{\boxed{\sigma(\mathrm{sk})}}_{\text{MLWE (key rec.)}}\,\underbrace{\boxed{\\|c\\|\sqrt{Q_s} \cdot {\color{red}\dim(\mathrm{sk}) \cdot \lambda \cdot d^3}}}_{\text{Smooth Rényi divergence}}\,\underbrace{\boxed{\sqrt{2}}}_{\text{Probing}}}^{\sigma(\mathrm{sig})}\,\overbrace{\boxed{\Omega(1)}}^{\text{MSIS (forgery)}}$ |
| Raccoon, Hint-MLWE [Heuristic] | $\overbrace{\underbrace{\boxed{\sigma(\mathrm{sk})}}_{\text{MLWE}}\,\underbrace{\boxed{\\|c\\|\sqrt{Q_s}}}_{\text{Hint-MLWE reduction (heur.)}}\,\underbrace{\boxed{\sqrt{2}}}_{\text{Probing}}}^{\sigma(\mathrm{sig})}\,\overbrace{\boxed{\Omega(1)}}^{\text{MSIS}}$ |

# Parameter selection and the modulus $q$.

PQ SHIELD

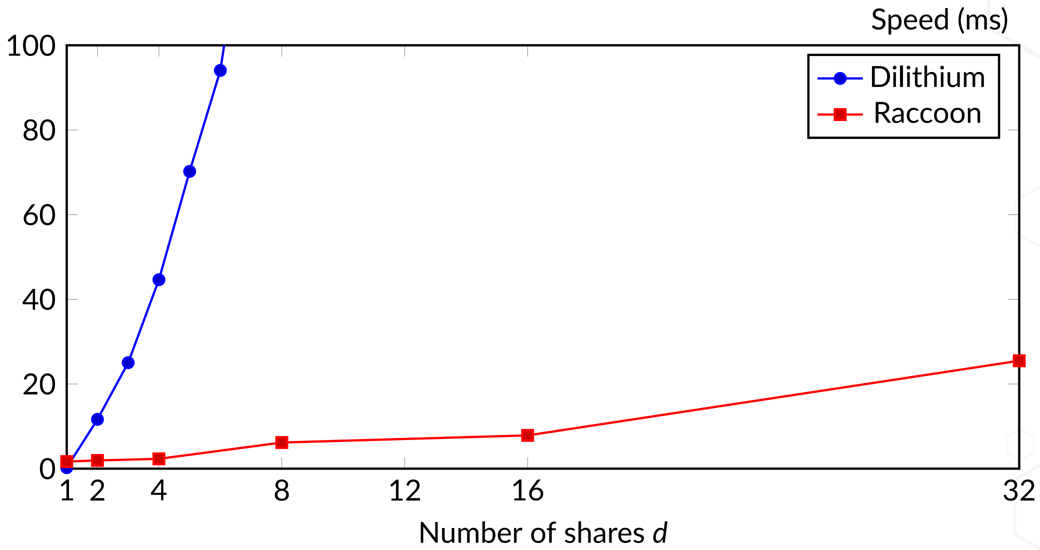
Signature sizes are quadratic in $\log q$ (trust me), so we want to minimize $q$.

| Method | Modulus $q$ (logarithmic view) |
|---|---|
| ML-DSA | $\overbrace{\underbrace{\sigma(\mathrm{sk})}_{\text{MLWE}}\ \underbrace{\lVert c \rVert_1 \cdot \dim(\mathrm{sk})}_{\text{Rejection sampling}}}^{\sigma(\mathrm{sig})}\ \overbrace{\Omega(1)}^{\text{MSIS (forgery)}}$ |
| Raccoon, Smooth Rényi [Proven] | $\overbrace{\underbrace{\sigma(\mathrm{sk})}_{\text{MLWE (key rec.)}}\ \underbrace{\lVert c \rVert \sqrt{Q_s} \cdot \dim(\mathrm{sk}) \cdot \lambda \cdot d^3}_{\text{Smooth Rényi divergence}}\ \underbrace{\sqrt{2}}_{\text{Probing}}}^{\sigma(\mathrm{sig})}\ \overbrace{\Omega(1)}^{\text{MSIS (forgery)}}$ |
| Raccoon, Hint-MLWE [Heuristic] | $\overbrace{\underbrace{\sigma(\mathrm{sk})}_{\text{MLWE}}\ \underbrace{\lVert c \rVert \sqrt{Q_s}}_{\text{Hint-MLWE reduction (heur.)}}\ \underbrace{\sqrt{2}}_{\text{Probing}}}^{\sigma(\mathrm{sig})}\ \overbrace{\Omega(1)}^{\text{MSIS}}$ |

→ **ML-DSA:** $q = 23$ bits, $|\mathrm{sig}| = 2420$ bytes

→ **Raccoon:** $q = 49$ bits, $|\mathrm{sig}| = 11524$ bytes

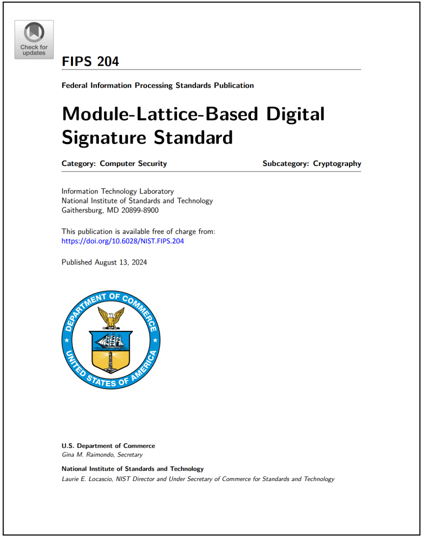
# Performances on a Desktop



# Mask Compression

# Reality check

PQSHIELD



Check for updates

FIPS 204

Federal Information Processing Standards Publication

**Module-Lattice-Based Digital Signature Standard**

**Category: Computer Security** **Subcategory: Cryptography**

Information Technology Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899-8900

This publication is available free of charge from:
https://doi.org/10.6028/NIST.FIPS.204

Published August 13, 2024

DEPARTMENT OF COMMERCE
UNITED STATES OF AMERICA

**U.S. Department of Commerce**
*Gina M. Raimondo, Secretary*

**National Institute of Standards and Technology**
*Laurie E. Locascio, NIST Director and Under Secretary of Commerce for Standards and Technology*


Right now, we need to implement ML-DSA

# Challenges in implementing ML-DSA

**Disclaimer:** I am not involved in any of the works/techniques in this section. I just think they're neat.

- Implementing Dilithium/ML-DSA in **RAM-constrained devices**
  - Reference implementation has a footprint $\geq$ 50 KiB
  - Using several tricks, [BRS22] compress it down to $\leq$ 7 KiB
- Implementing **high-order masked** Dilithium/ML-DSA
  - Feasible with $O(d^2 \log q)$ overhead [CGTZ23, CGL$^+$24]
- Implementing **high-order masked** Dilithium/ML-DSA in **RAM-constrained devices**?

# What is costly to store in ML-DSA?

**Consider ML-DSA-87:**

| | |
|---|---|
| One ring element | 768 B |
| Secret key $(\mathbf{s}_1, \mathbf{s}_2)$ | 3,840 B |
| Randomness $\mathbf{y}$ | 4,480 B |
| Commitment $\mathbf{w}$ | 5,888 B |
| Fully expanded matrix $\mathbf{A}$ | 41,216 B |
| … | … |

**Solutions:**

- → (Re-)generate everything from seeds ($\mathbf{A}$, $\mathbf{s}_1$, $\mathbf{s}_2$, $\mathbf{y}$, …)
- → Memory laziness (throw away values after usage)

# Is it any different with masking?

Masked sensitive values are **expensive** to store. At order 4:

- → $[\![\mathbf{y}]\!]$: between 17, 920 B and 20,608 B
- → $[\![\mathbf{s}_1]\!], [\![\mathbf{s}_2]\!]$: between 8, 192 B and 44,160 B

Makes it impractical to implement ML-DSA-87 on devices with $\leq$ 32 KiB of RAM.

Is there a chance we can use **seeds** to reduce storage?

# Mask compression

Markku-Juhani O. Saarinen, Mélissa Rossi: *Mask Compression: High-Order Masking on Memory-Constrained Devices*. SAC 2023. [SR24]

## Mask Compression

- **Key idea:** make all shares (except one) pseudorandom:
  - $[\![\mathbf{x}]\!]_d = (\mathbf{x}_0, \mathsf{seed}_1, \ldots, \mathsf{seed}_i)$
  - $\mathbf{x}_0 = \mathbf{x} - \sum_{i>0} PRF(\mathsf{seed}_i)$
- **Security (in isolation):** $\mathbf{x}$ remains secret even if $t < d$ values are probed

# Mask compression

Markku-Juhani O. Saarinen, Mélissa Rossi: *Mask Compression: High-Order Masking on Memory-Constrained Devices*. SAC 2023. [SR24]

## Mask Compression

**Key idea:** make all shares (except one) pseudorandom:

- $[\![\mathbf{x}]\!]_d = (\mathbf{x}_0, \mathsf{seed}_1, \ldots, \mathsf{seed}_i)$
- $\mathbf{x}_0 = \mathbf{x} - \sum_{i>0} PRF(\mathsf{seed}_i)$

**Security (in isolation):** $\mathbf{x}$ remains secret even if $t < d$ values are probed

**Efficiency:** Decrease the bitsize from $d \cdot |\mathbf{x}|$ down to $|\mathbf{x}| + (d-1) \cdot \lambda$.
If $\mathbf{x}$ has $k$ coefficients, we may either:

- Use one seed per coef $\Rightarrow$ bitsize becomes $|\mathbf{x}| + (d-1) \cdot k \cdot \lambda$
- Use different PRFs $\Rightarrow$ the $j$-th coef of the $i$-th share would be $PRF_j(\mathsf{seed}_i)$

# Mask compression

Markku-Juhani O. Saarinen, Mélissa Rossi: *Mask Compression: High-Order Masking on Memory-Constrained Devices*. SAC 2023. [SR24]

## Mask Compression

- **Key idea:** make all shares (except one) pseudorandom:
  - $[\![\mathbf{x}]\!]_d = (\mathbf{x}_0, \text{seed}_1, \ldots, \text{seed}_i)$
  - $\mathbf{x}_0 = \mathbf{x} - \sum_{i>0} PRF(\text{seed}_i)$
- **Security (in isolation):** $\mathbf{x}$ remains secret even if $t < d$ values are probed
- **Efficiency:** Decrease the bitsize from $d \cdot |\mathbf{x}|$ down to $|\mathbf{x}| + (d-1) \cdot \lambda$. If $\mathbf{x}$ has $k$ coefficients, we may either:
  - Use one seed per coef $\Rightarrow$ bitsize becomes $|\mathbf{x}| + (d-1) \cdot k \cdot \lambda$
  - Use different PRFs $\Rightarrow$ the $j$-th coef of the $i$-th share would be $PRF_j(\text{seed}_i)$
- **Computations?**
  - [SR24] show how to perform a SNI refresh (compatible w/ this structure)
  - Other ML-DSA operations (decompose, addition, rejection) may require all shares at once $\rightarrow$ see efficiency trade-offs

# Mask compression

Markku-Juhani O. Saarinen, Mélissa Rossi: *Mask Compression: High-Order Masking on Memory-Constrained Devices*. SAC 2023. [SR24]

## Mask Compression

**Key idea:** make all shares (except one) pseudorandom:

- $[\![\mathbf{x}]\!]_d = (\mathbf{x}_0, \text{seed}_1, \ldots, \text{seed}_i)$
- $\mathbf{x}_0 = \mathbf{x} - \sum_{i>0} PRF(\text{seed}_i)$

**Security (in isolation):** $\mathbf{x}$ remains secret even if $t < d$ values are probed

**Efficiency:** Decrease the bitsize from $d \cdot |\mathbf{x}|$ down to $|\mathbf{x}| + (d-1) \cdot \lambda$.
If $\mathbf{x}$ has $k$ coefficients, we may either:

- Use one seed per coef $\Rightarrow$ bitsize becomes $|\mathbf{x}| + (d-1) \cdot k \cdot \lambda$
- Use different PRFs $\Rightarrow$ the $j$-th coef of the $i$-th share would be $PRF_j(\text{seed}_i)$

**Computations?**

- [SR24] show how to perform a SNI refresh (compatible w/ this structure)
- Other ML-DSA operations (decompose, addition, rejection) may require all shares at once $\rightarrow$ see efficiency trade-offs

This allows to implement ML-DSA-87 with 4 shares and 16 KiB of RAM.

# Conclusion

# Conclusion

PQ SHIELD

## # Masking-friendly lattice schemes

- Requires flexibility in exploration of design space and security notions
- Can be extremely efficient
- ? Concrete SCA resilience?
- ? Better proofs/constructions in alternative leakage models?
- ? Masking-friendly KEMs?

## Lattice-friendly masking schemes

- We have barely scratched the surface
- ? What would be **really** nice is a masking scheme that:
  - Can be converted efficiently from/to arithmetic masking
  - Allows to perform efficiently decompose/sample/reject

# Questions?

https://raccoonfamily.org
https://ia.cr/2024/1291
https://ia.cr/2023/1117
https://tprest.github.io

Joppe W. Bos, Joost Renes, and Amber Sprenkels.
Dilithium for memory constrained devices.
In Lejla Batina and Joan Daemen, editors, *AFRICACRYPT 22*, volume 2022 of *LNCS*, pages 217–235. Springer, Cham, July 2022.

Jean-Sébastien Coron, François Gérard, Tancrède Lepoint, Matthias Trannoy, and Rina Zeitoun.
Improved high-order masked generation of masking vector and rejection sampling in dilithium.
*IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2024(4):335–354, Sep. 2024.

Jean-Sébastien Coron, François Gérard, Matthias Trannoy, and Rina Zeitoun.
Improved gadgets for the high-order masking of Dilithium.
*IACR TCHES*, 2023(4):110–145, 2023.

Rafaël Del Pino, Shuichi Katsumata, Mary Maller, Fabrice Mouhartem, Thomas Prest, and Markku-Juhani O. Saarinen.
Threshold raccoon: Practical threshold signatures from standard lattice assumptions.

In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part II*, volume 14652 of *LNCS*, pages 219–248. Springer, Cham, May 2024.

Rafaël del Pino, Shuichi Katsumata, Thomas Prest, and Mélissa Rossi.
Raccoon: A masking-friendly signature proven in the probing model.
In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part I*, volume 14920 of *LNCS*, pages 409–444. Springer, Cham, August 2024.

Muhammed F. Esgin, Thomas Espitau, Guilhem Niot, Thomas Prest, Amin Sakzad, and Ron Steinfeld.
Plover: Masking-friendly hash-and-sign lattice signatures.
In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part VII*, volume 14657 of *LNCS*, pages 316–345. Springer, Cham, May 2024.

Emre Karabulut and Aydin Aysu.
FALCON down: Breaking FALCON post-quantum signature scheme through side-channel attacks.
In *58th ACM/IEEE Design Automation Conference, DAC 2021, San Francisco, CA, USA, December 5-9, 2021*, pages 691–696. IEEE, 2021.

Eike Kiltz, Vadim Lyubashevsky, and Christian Schaffner.

A concrete treatment of Fiat-Shamir signatures in the quantum random-oracle model.
In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part III*, volume 10822 of *LNCS*, pages 552–586. Springer, Cham, April / May 2018.

Duhyeong Kim, Dongwon Lee, Jinyeong Seo, and Yongsoo Song.
Toward practical lattice-based proof of knowledge from hint-MLWE.
In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part V*, volume 14085 of *LNCS*, pages 549–580. Springer, Cham, August 2023.

Markku-Juhani O. Saarinen and Mélissa Rossi.
Mask compression: High-order masking on memory-constrained devices.
In Claude Carlet, Kalikinkar Mandal, and Vincent Rijmen, editors, *SAC 2023*, volume 14201 of *LNCS*, pages 65–81. Springer, Cham, August 2024.

Shiduo Zhang, Xiuhan Lin, Yang Yu, and Weijia Wang.
Improved power analysis attacks on falcon.
In Carmit Hazay and Martijn Stam, editors, *EUROCRYPT 2023, Part IV*, volume 14007 of *LNCS*, pages 565–595. Springer, Cham, April 2023.