Masking-Friendly Signatures and the Design of Raccoon

Rafael del Pino
PQShield

Thomas Espitau
PQShield

Shuichi Katsumata
PQShield
AIST

Mary Maller
PQShield
Ethereum Foundation

Fabrice Mouhartem
CryptPad

Thomas Prest
PQShield

Mélissa Rossi
ANSSI

Markku-Juhani Saarinen
PQShield
Tampere University

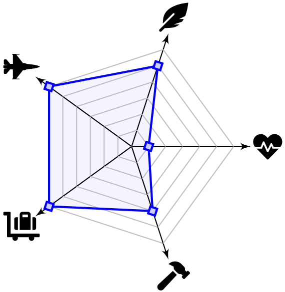
Motivation

Signature schemes strike a balance between:

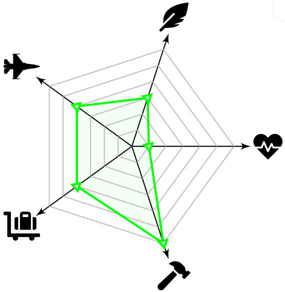
- Sizes (verification key and signatures)
- Speed (signing, verification)
- Portability
- Conservative assumptions
- **Resistance against side-channel attacks**

And so on...

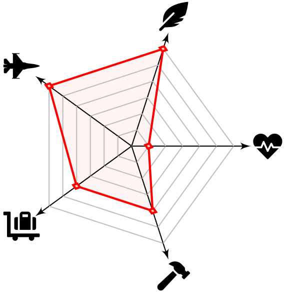
Dilithium



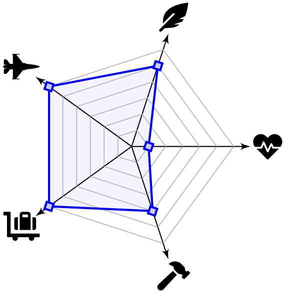
SPHINCS+



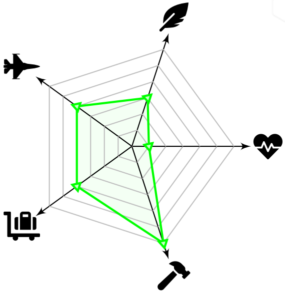
Falcon



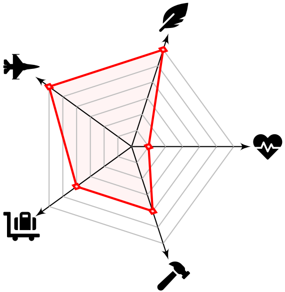
Dilithium



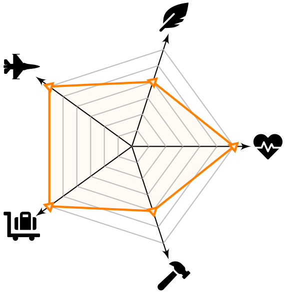
SPHINCS+



Falcon



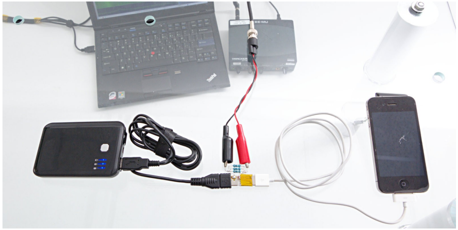
Raccoon



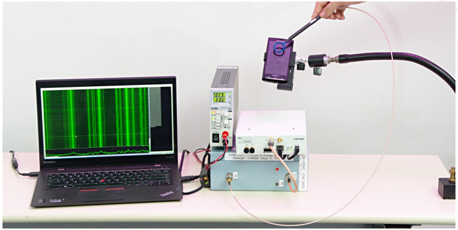
Side-Channel Attacks

Side-channel attacks in cryptography

Power consumption [KJJ99]



Electromagnetic emissions [Eck85]



Timing measurement [Koc96]

Acoustic emissions [AA04]

Example with Falcon

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 $\langle \mathbf{sig}, \mathbf{sk} \rangle$, or **sk** itself.

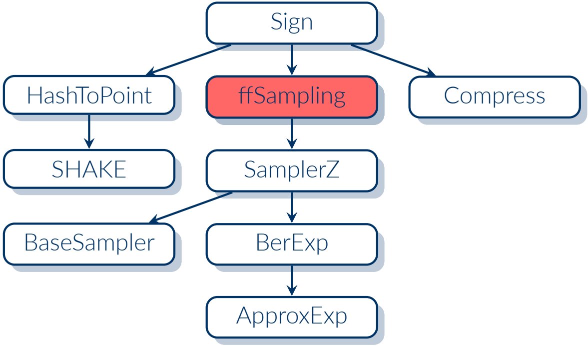
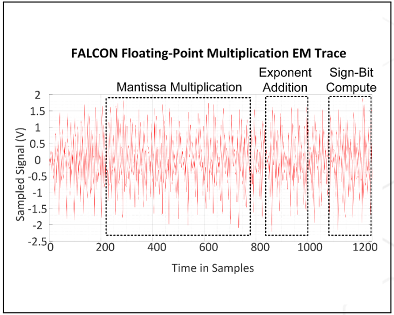


Figure 1: Flowchart of the signature



Learning **sk** directly

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

Example with Falcon

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 $\langle \mathbf{sig}, \mathbf{sk} \rangle$, or **sk** itself.

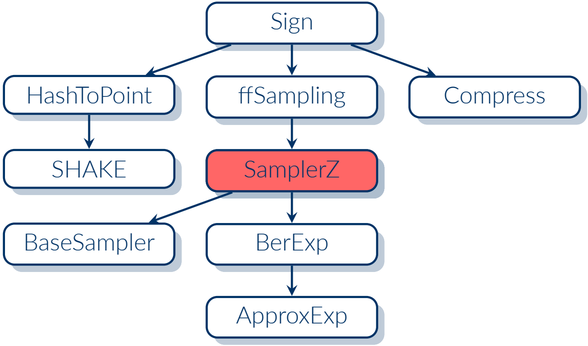
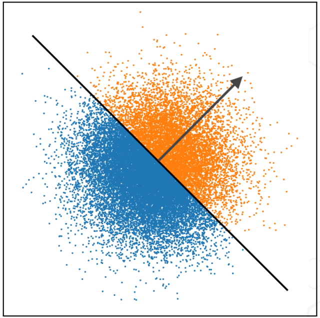


Figure 1: Flowchart of the signature



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

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

Masking and the t-probing model

t-probing model

- Adversary can probe t circuit values at runtime
- Unrealistic but a good starting point

Masking

- Each sensitive value x is split in d shares:

$$[\![x]\!] = (x_0, x_1, \ldots, x_{d-1}) \quad (1)$$

such that

$$x_0 + x_1 + \cdots + x_{d-1} = x \quad (2)$$

- In t-probing model, ideally 0 leakage if $d > t$
- In "real life", security is exponential in d
- What about computations?

Interlude: river-crossing puzzles

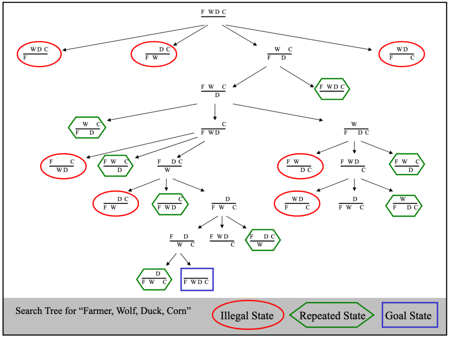
Remember this puzzle?

> " A farmer with a wolf, a goat, and a cabbage must cross a river by boat. The boat can carry only the farmer and a single item. If left unattended together, the wolf would eat the goat, or the goat would eat the cabbage. How can they cross the river without anything being eaten? "

Interlude: river-crossing puzzles

Remember this puzzle?

> “ A farmer with a wolf, a goat, and a cabbage must cross a river by boat. The boat can carry only the farmer and a single item. If left unattended together, the wolf would eat the goat, or the goat would eat the cabbage. How can they cross the river without anything being eaten? ”



It gets quickly complicated…

From river-crossing to masking

Now replace:

1. The set { farmer, wolf, goat, cabbage } by the shares $(x_0, \ldots, x_{d-1})$
2. The operation "*everyone crosses the river*" by an arbitrary function $f([\![x]\!]) \to [\![y]\!]$
3. The constraints "*never leave A alone with B*" by "*a probing adversary shall not learn anything*"

... and you obtain an inexhaustible source of headaches for cryptographers.

From river-crossing to masking

Now replace:

1. The set { farmer, wolf, goat, cabbage } by the shares $(x_0, \ldots, x_{d-1})$
2. The operation "*everyone crosses the river*" by an arbitrary function $f([\![x]\!]) \to [\![y]\!]$
3. The constraints "*never leave A alone with B*" by "*a probing adversary shall not learn anything*"

... and you obtain an inexhaustible source of headaches for cryptographers.

How difficult are operations to mask?

- **Addition ($[\![c]\!] = [\![a + b]\!]$)?**
  - Simple and fast: $\Theta(d)$ operations
- **Refresh ($[\![a]\!] \to [\![a]\!]'$)?**
  - Protect against attacks **and** allows composition frameworks (SNI, PINI, IOS, etc.)
  - Simple and fast: $\Theta(d \log d)$ operations
- **Multiplication ($[\![c]\!] = [\![a \cdot b]\!]$)?**
  - Complex and slower: $\Theta(d^2)$ operations
- **More complex operations?**
  - Use so-called *mask conversions*, very slow: $\Theta(d^2)$ operations **per bit to convert**

Masking Dilithium

Masking Dilithium step-by-step

Dilithium-Sign

1. Sample $\mathbf{r} \leftarrow \text{Uniform}(S)$
2. $\mathbf{w} := \mathbf{A}\,\mathbf{r}$
3. $\mathbf{w}_\top := \lfloor \mathbf{w} \rceil_k$
4. $c := H(\mathbf{w}_\top, \mathsf{msg})$
5. $\mathbf{z} := \mathbf{s}\,c + \mathbf{r}$
6. If $\mathbf{z}$ not in S', goto 1
7. $\mathbf{h} := \mathbf{w}_\top - \lfloor \mathbf{A}\,\mathbf{z} - \mathbf{t}\,c \rceil_k$
8. Output $\mathsf{sig} = (c, \mathbf{z}, \mathbf{h})$

Masking Dilithium step-by-step

Dilithium-Sign

1. Sample $\mathbf{r} \leftarrow \mathrm{Uniform}(S)$
2. $\mathbf{w} := \mathbf{A}\,\mathbf{r}$
3. $\mathbf{w}_\top := \lfloor \mathbf{w} \rceil_k$
4. $c := H(\mathbf{w}_\top, \mathbf{msg})$
5. $\mathbf{z} := \mathbf{s}\,c + \mathbf{r}$
6. If $\mathbf{z}$ not in S', goto 1
7. $\mathbf{h} := \mathbf{w}_\top - \lfloor \mathbf{A}\,\mathbf{z} - \mathbf{t}\,c \rceil_k$
8. Output $\mathbf{sig} = (c, \mathbf{z}, \mathbf{h})$

1. How do we sample a uniform d-sharing $[\![r]\!]$ of $r \leftarrow \mathrm{Uniform}(S)$ securely?

- $S = \mathbb{Z}_q^n$ is easy, $S \subsetneq \mathbb{Z}_q$ is **hard**
- Naive solutions do not work
- Best known method:
  1. Find a **boolean** circuit f that samples $(r_1, \ldots, r_{\log q}) \leftarrow \mathrm{Uniform}(S)$
  2. Evaluate f in masked boolean form:
  $$([\![r_1]\!]_b, \ldots, [\![r_{\log q}]\!]_b) \leftarrow [\![f]\!]_b \tag{3}$$
  3. Use mask conversion on each bit:
  $$[\![r_i]\!]_b \rightarrow [\![r_i]\!]_a \tag{4}$$
  4. Recombine the masked bits:
  $$[\![r]\!]_a := \sum_i 2^i\,[\![r_i]\!]_a \tag{5}$$

Complexity: $O\left(d^2\,(|f| + \log q)\right)$

Masking Dilithium step-by-step

Dilithium-Sign

1. Sample $\mathbf{r} \leftarrow \text{Uniform}(S)$
2. $\mathbf{w} := \mathbf{A}\,\mathbf{r}$
3. $\mathbf{w}_\top := \lfloor \mathbf{w} \rceil_k$
4. $c := H(\mathbf{w}_\top, \mathbf{msg})$
5. $\mathbf{z} := \mathbf{s}\,c + \mathbf{r}$
6. If $\mathbf{z}$ not in S', goto 1
7. $\mathbf{h} := \mathbf{w}_\top - \lfloor \mathbf{A}\,\mathbf{z} - \mathbf{t}\,c \rceil_k$
8. Output $\mathbf{sig} = (c, \mathbf{z}, \mathbf{h})$

2 Compute $\mathbf{A}\,\mathbf{r}$:

→ Linear operation thus **easy**

→ Complexity: $\tilde{O}(d)$

3 Bit dropping $\mathbf{w}_\top := \lfloor \mathbf{w} \rceil_k$:

→ The lower bits of **w** are sensitive:

$$\mathbf{w} - (\mathbf{A}\mathbf{z} - \mathbf{t}c) = \mathbf{e}c$$

→ Requires mask conversion (B2A + A2B)

→ Complexity: $O(d^2 \log q)$

4 Challenge computation is unmasked:

→ Previously: ad-hoc assumption [BBE+18]

→ Now: everyone cites [DFPS23]

Masking Dilithium step-by-step

Dilithium-Sign

1. Sample $\mathbf{r} \leftarrow \mathrm{Uniform}(S)$
2. $\mathbf{w} := \mathbf{A}\,\mathbf{r}$
3. $\mathbf{w}_\top := \lfloor \mathbf{w} \rceil_k$
4. $c := H(\mathbf{w}_\top, \mathbf{msg})$
5. $\mathbf{z} := \mathbf{s}\,c + \mathbf{r}$
6. If $\mathbf{z}$ not in S', goto 1
7. $\mathbf{h} := \mathbf{w}_\top - \lfloor \mathbf{A}\,\mathbf{z} - \mathbf{t}\,c \rceil_k$
8. Output $\mathbf{sig} = (c, \mathbf{z}, \mathbf{h})$

5 Compute $\mathbf{z} = \mathbf{s}\,c + \mathbf{r}$:

→ Linear thus fast

---

6 Rejection sampling:

→ Requires mask conversion (A2B), slow

---

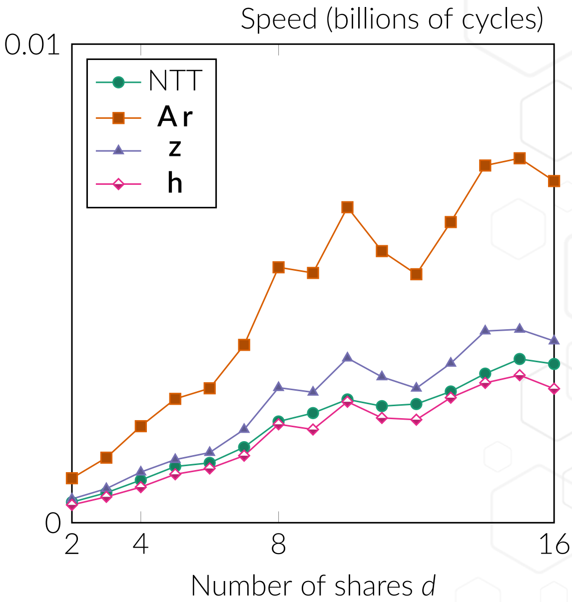
7 Compute $\mathbf{h}$:

→ Linear thus fast

Masked Dilithium [CGTZ23]

Dilithium-Sign

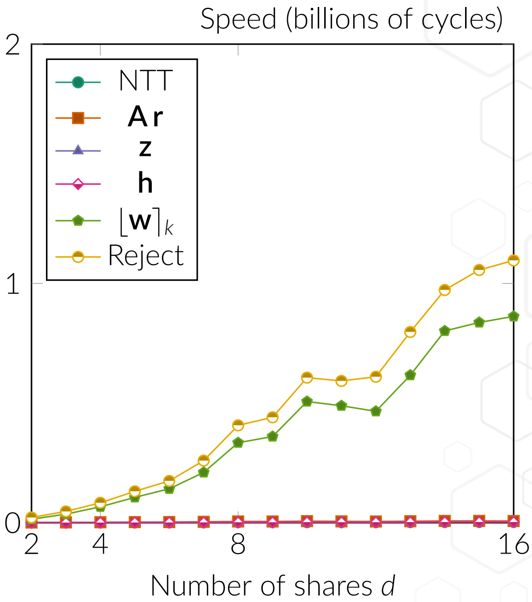
1. Sample $\mathbf{r} \leftarrow S$
2. $\mathbf{w} := \mathbf{A}\,\mathbf{r}$ $\quad\triangleright\ \tilde{O}(d)$
3. $\mathbf{w}_\top := \lfloor \mathbf{w} \rceil_k$
4. $c := H(\mathbf{w}_\top, \mathbf{msg})$ $\quad\triangleright$ No mask
5. $\mathbf{z} := \mathbf{s}\,c + \mathbf{r}$ $\quad\triangleright\ \tilde{O}(d)$
6. If $\mathbf{z}$ not in S', goto 1
7. $\mathbf{h} := \mathbf{w}_\top - \lfloor \mathbf{A}\,\mathbf{z} - \mathbf{t}\,c \rceil_k$ $\quad\triangleright\ \tilde{O}(d)$
8. Output $\mathbf{sig} = (c, \mathbf{z}, \mathbf{h})$



Masked Dilithium [CGTZ23]

Dilithium-Sign

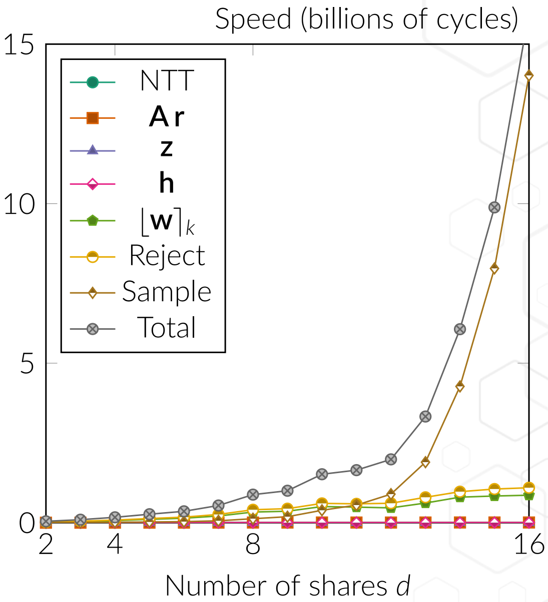
1. Sample $\mathbf{r} \leftarrow S$
2. $\mathbf{w} := \mathbf{A}\,\mathbf{r}$ $\triangleright\ \tilde{O}(d)$
3. $\mathbf{w}_\top := \lfloor \mathbf{w} \rceil_k$ $\triangleright\ O(d^2 \log q)$
4. $c := H(\mathbf{w}_\top, \mathbf{msg})$ $\triangleright$ No mask
5. $\mathbf{z} := \mathbf{s}\,c + \mathbf{r}$ $\triangleright\ \tilde{O}(d)$
6. If $\mathbf{z}$ not in S', goto 1 $\triangleright\ O(d^2 \log q)$
7. $\mathbf{h} := \mathbf{w}_\top - \lfloor \mathbf{A}\,\mathbf{z} - \mathbf{t}\,c \rceil_k$ $\triangleright\ \tilde{O}(d)$
8. Output $\mathbf{sig} = (c, \mathbf{z}, \mathbf{h})$



Masked Dilithium [CGTZ23]

Dilithium-Sign

1. Sample $\mathbf{r} \leftarrow S$ $\triangleright$ $O(d^2 \log q)$
2. $\mathbf{w} := \mathbf{A}\,\mathbf{r}$ $\triangleright$ $\tilde{O}(d)$
3. $\mathbf{w}_\top := \lfloor \mathbf{w} \rceil_k$ $\triangleright$ $O(d^2 \log q)$
4. $c := H(\mathbf{w}_\top, \mathbf{msg})$ $\triangleright$ No mask
5. $\mathbf{z} := \mathbf{s}\,c + \mathbf{r}$ $\triangleright$ $\tilde{O}(d)$
6. If $\mathbf{z}$ not in S', goto 1 $\triangleright$ $O(d^2 \log q)$
7. $\mathbf{h} := \mathbf{w}_\top - \lfloor \mathbf{A}\,\mathbf{z} - \mathbf{t}\,c \rceil_k$ $\triangleright$ $\tilde{O}(d)$
8. Output $\mathbf{sig} = (c, \mathbf{z}, \mathbf{h})$



Takeaway

Masking Dilithium efficiently remains difficult despite several years of works:

- *Masking the GLP lattice-based signature scheme at any order* [BBE+18]
- *Masking Dilithium - efficient implementation and side-channel evaluation* [MGTF19]
- *Protecting dilithium against leakage: Revisited sensitivity analysis and improved implementations* [ABC+23]
- *Improved Gadgets for the High-Order Masking of Dilithium* [CGTZ23]

None of these works manage to break the $\Theta(d^2 \log q)$ barrier.

Takeaway

Masking Dilithium efficiently remains difficult despite several years of works:

- *Masking the GLP lattice-based signature scheme at any order* [BBE⁺18]
- *Masking Dilithium - efficient implementation and side-channel evaluation* [MGTF19]
- *Protecting dilithium against leakage: Revisited sensitivity analysis and improved implementations* [ABC⁺23]
- *Improved Gadgets for the High-Order Masking of Dilithium* [CGTZ23]

None of these works manage to break the $\Theta(d^2 \log q)$ barrier.

What about Mitaka?

- Last year: *Mitaka: a simpler, parallelizable, maskable variant of Falcon* [EFG⁺22]
- Now: *A Key-Recovery Attack against Mitaka in the t-Probing Model* [Pre23]
  - Slides and video on `https://tprest.github.io/`

Mitaka cannot be masked efficiently with existing techniques.

Takeaway

Masking Dilithium efficiently remains difficult despite several years of works:

- → *Masking the GLP lattice-based signature scheme at any order* [BBE+18]
- → *Masking Dilithium - efficient implementation and side-channel evaluation* [MGTF19]
- → *Protecting dilithium against leakage: Revisited sensitivity analysis and improved implementations* [ABC+23]
- → *Improved Gadgets for the High-Order Masking of Dilithium* [CGTZ23]

None of these works manage to break the $\Theta(d^2 \log q)$ barrier.

What about Mitaka?

- → Last year: *Mitaka: a simpler, parallelizable, maskable variant of Falcon* [EFG+22]
- → Now: *A Key-Recovery Attack against Mitaka in the t-Probing Model* [Pre23]
  - › Slides and video on `https://tprest.github.io/`

Mitaka cannot be masked efficiently with existing techniques.

Back to the drawing board!

Raccoon

Main design principle: be like water

We build a masking-friendly scheme from scratch:

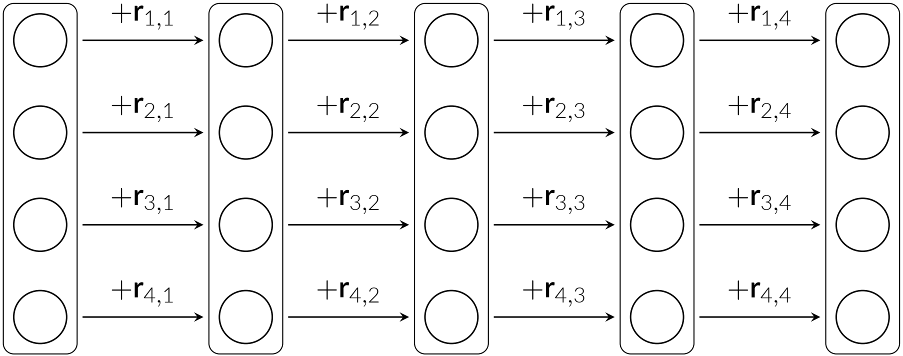
- We can completely deviate from existing schemes and frameworks
- Only hard constraints are security and masking-friendliness

Raccoon - masked key generation

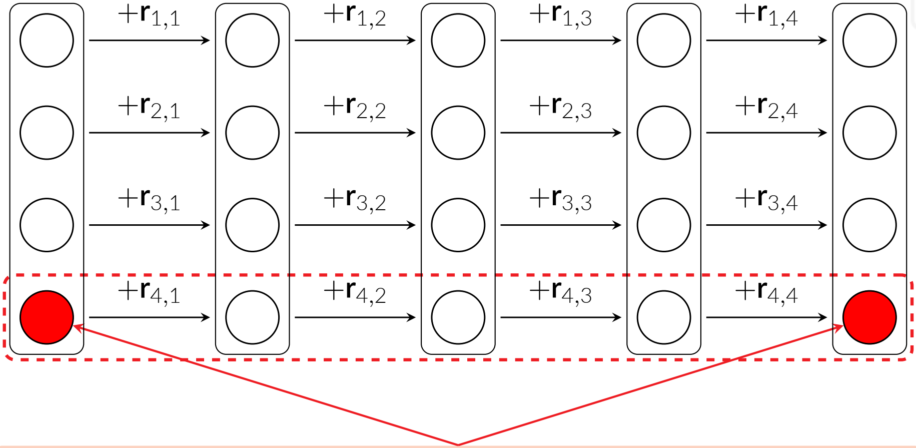
Keygen$(1^\lambda) \to (\mathsf{sk}, \mathsf{vk})$

1. Generate a large matrix $\mathbf{A} = [\,\mathbf{I}\,|\,\bar{\mathbf{A}}\,] \in \mathcal{R}_q^{k\times(k+\ell)}$ $\triangleright$ No mask
2. $[\![\mathbf{s}]\!] = (\mathbf{0}, \ldots, \mathbf{0})$ $\triangleright$ Fast
3. For $i \in [\mathsf{rep}]$: $\triangleright$ We call this "AddRepNoise"
    1. Sample short random shares in parallel: $[\![\mathbf{r}]\!] = (\mathbf{r}_0, \ldots, \mathbf{r}_{d-1})$ $\triangleright$ Fast
    2. $[\![\mathbf{s}]\!] := [\![\mathbf{s}]\!] + [\![\mathbf{r}]\!]$ $\triangleright$ Fast
    3. Refresh $[\![\mathbf{s}]\!]$ $\triangleright$ Fast
4. Compute $\mathbf{t} = \mathbf{A} \cdot [\![\mathbf{s}]\!]$ $\triangleright$ Fast
5. Unmask $[\![\mathbf{t}]\!]$ to obtain $\mathbf{t}$ $\triangleright$ Fast
6. Verification key is $\mathsf{vk} = (\mathbf{A}, \mathbf{t})$ $\triangleright$ No mask
7. Signing key is $\mathsf{sk} = [\![\mathbf{s}]\!]$

What happens inside AddRepNoise?

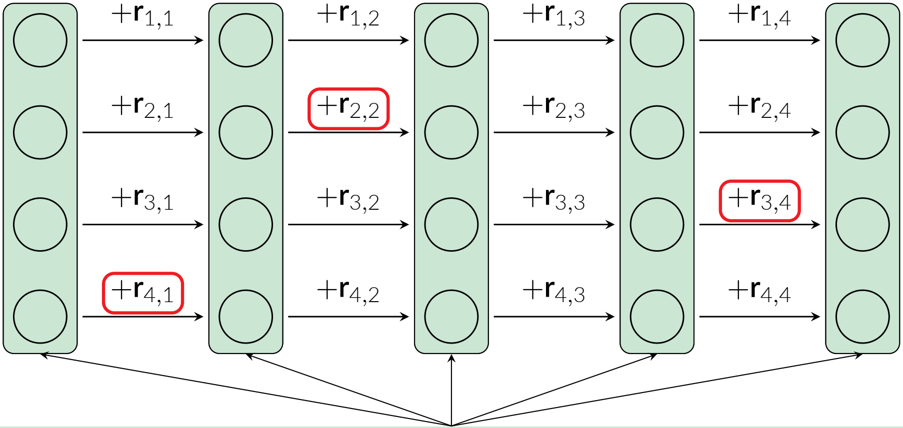


What happens inside AddRepNoise?



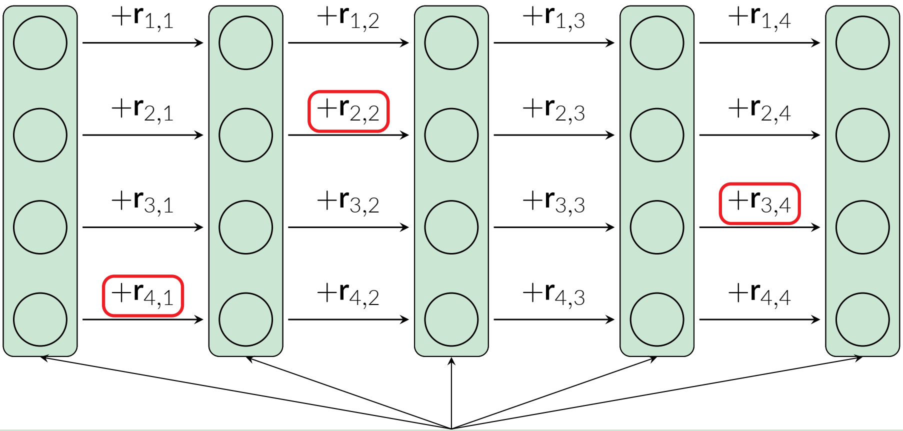
Problem: a probing adversary can learn the sum of T random in 2 probes.

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.

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.

"Thomas, this is not a t-probing secure gadget!"

Raccoon - masked key generation

Keygen$(1^\lambda) \to (\mathsf{sk}, \mathsf{vk})$

1. Generate $\mathbf{A} = [\,\mathbf{I} \,|\, \bar{\mathbf{A}}\,]$
2. Sample $[\![\mathbf{s}]\!]$ using AddRepNoise
3. Compute $\mathbf{t} = \mathbf{A} \cdot [\![\mathbf{s}]\!]$
4. Unmask $[\![\mathbf{t}]\!]$ to obtain $\mathbf{t}$
5. Verification key is $\mathbf{vk} = (\mathbf{A}, \mathbf{t})$
6. Signing key is $\mathbf{sk} = [\![\mathbf{s}]\!]$

LeakyKeygen$(1^\lambda) \to (\mathsf{sk}, \mathsf{vk}, \mathsf{aux})$

1. Generate $\mathbf{A} = [\,\mathbf{I} \,|\, \bar{\mathbf{A}}\,]$
2. $\mathbf{s}_0 \leftarrow \{\text{sum of } (\mathbf{rep}\, d - t) \text{ short noises}\}$
3. Sample t short noises $(\bar{\mathbf{s}}_1, \ldots, \bar{\mathbf{s}}_t)$
4. $\mathbf{s} := \mathbf{s}_0 + \sum_i \bar{\mathbf{s}}_i$
5. $\mathbf{t} := \mathbf{A}\,\mathbf{s}$
6. Return $\mathbf{vk} = (\mathbf{A}, \mathbf{t})$, $\mathbf{sk} = \mathbf{s}$, auxiliary information $\mathbf{aux} = (\bar{\mathbf{s}}_1, \ldots, \bar{\mathbf{s}}_t)$

Proof intuition:

- For any EUF-CMA t-probing adversary given access to **Keygen** (left alg.), we can construct an EUF-CMA adversary given access to **LeakyKeygen** (right alg.)
- **LeakyKeygen**() can be simulated given an LWE sample $(\mathbf{A}, \mathbf{t}_0 = \mathbf{A}\,\mathbf{s}_0)$

Dilithium - signature

Dilithium follows the Fiat-Shamir **with aborts** paradigm.

$\mathrm{Sign}(\mathsf{sk} = \mathbf{s}, \mathsf{vk} = (\mathbf{A}, \mathbf{t}), \mathsf{msg}) \to \mathsf{sig}$

1. Generate a short ephemeral secret $\mathbf{r}$ ▷ Slow
2. Compute the commitment $\mathbf{w} = \mathbf{A} \cdot \mathbf{r}$ ▷ Fast
3. Compute the challenge $c = \mathrm{H}(\mathbf{w}, \mathsf{msg}, \mathsf{vk})$ ▷ No mask
4. Compute the response $\mathbf{z} = \mathbf{s} \cdot c + \mathbf{r}$ ▷ Fast
5. Check that $\mathbf{z}$ is in a given interval. If not, restart. ▷ Slow
6. Signature is $\mathsf{sig} = (c, \mathbf{z})$

Masking bottlenecks:

- 😱 Short secret generation (1) requires B2A.
- 😱 Rejection sampling (5) requires A2B.

Total masking overhead: $\Theta(d^2 \log q)$

Raccoon - masked signature

$\mathsf{Sign}(\mathsf{sk} = [\![\mathbf{s}]\!], \mathsf{vk} = (\mathbf{A}, \mathbf{t}), \mathsf{msg}) \to \mathsf{sig}$

1. Generate a masked short ephemeral secret $[\![\mathbf{r}]\!]$ using "AddRepNoise" ▷ Fast
2. Compute the commitment $[\![\mathbf{w}]\!] = \mathbf{A} \cdot [\![\mathbf{r}]\!]$ ▷ Fast
3. Unmask $[\![\mathbf{w}]\!]$ to obtain $\mathbf{w}$ ▷ Fast
4. Compute the challenge $c = \mathsf{H}(\mathbf{w}, \mathsf{msg}, \mathsf{vk})$ ▷ No mask
5. Compute the response $[\![\mathbf{z}]\!] = [\![\mathbf{s}]\!] \cdot c + [\![\mathbf{r}]\!]$ ▷ Fast
6. Unmask $[\![\mathbf{z}]\!]$ to obtain $\mathbf{z}$ ▷ Fast
7. (No more rejection sampling!)
8. Signature is $\mathsf{sig} = (c, \mathbf{z})$

Total masking overhead: $O(d \log d)$

Raccoon - masked signature

$\mathsf{Sign}(\mathsf{sk} = [\![\mathbf{s}]\!], \mathsf{vk} = (\mathbf{A}, \mathbf{t}), \mathsf{msg}) \to \mathsf{sig}$

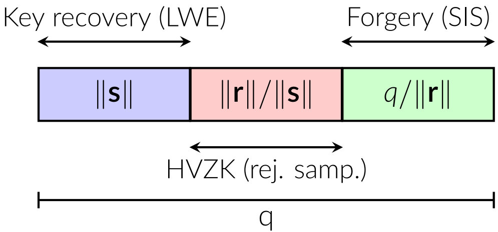
1. Generate a masked short ephemeral secret $[\![\mathbf{r}]\!]$ using "AddRepNoise" ▷ Fast
2. Compute the commitment $[\![\mathbf{w}]\!] = \mathbf{A} \cdot [\![\mathbf{r}]\!]$ ▷ Fast
3. Unmask $[\![\mathbf{w}]\!]$ to obtain $\mathbf{w}$ ▷ Fast
4. Compute the challenge $c = \mathsf{H}(\mathbf{w}, \mathsf{msg}, \mathsf{vk})$ ▷ No mask
5. Compute the response $[\![\mathbf{z}]\!] = [\![\mathbf{s}]\!] \cdot c + [\![\mathbf{r}]\!]$ ▷ Fast
6. Unmask $[\![\mathbf{z}]\!]$ to obtain $\mathbf{z}$ ▷ Fast
7. (No more rejection sampling!)
8. Signature is $\mathsf{sig} = (c, \mathbf{z})$

Total masking overhead: $O(d \log d)$

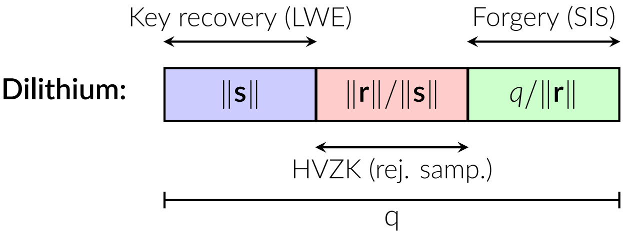
But why would it even be secure?

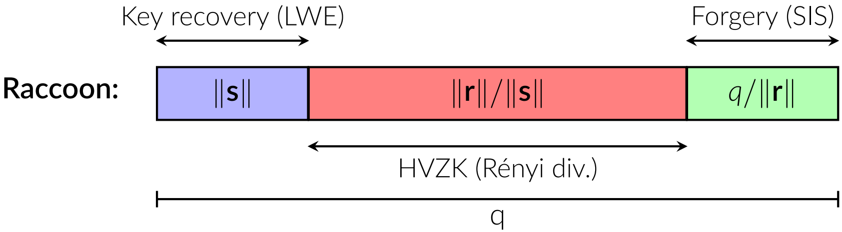
Impact on the modulus

Dilithium:



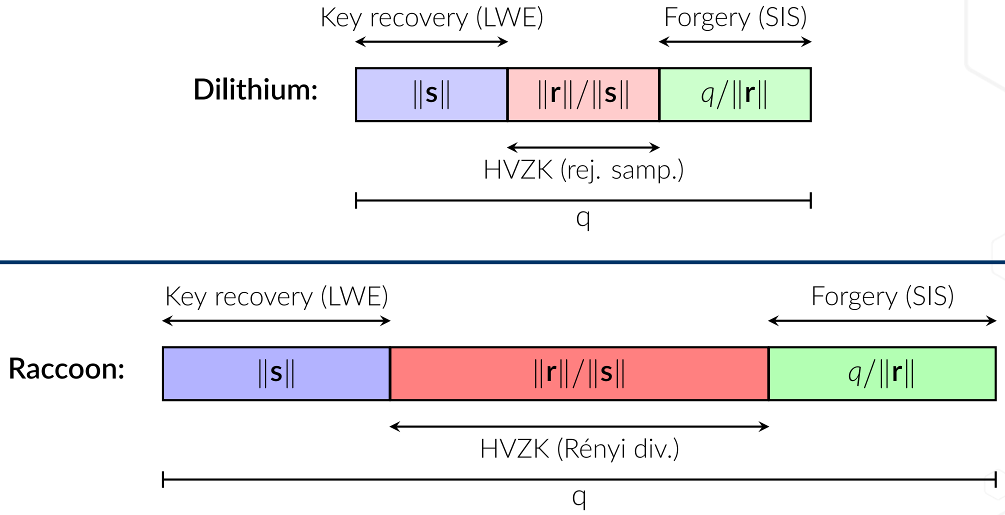
Impact on the modulus





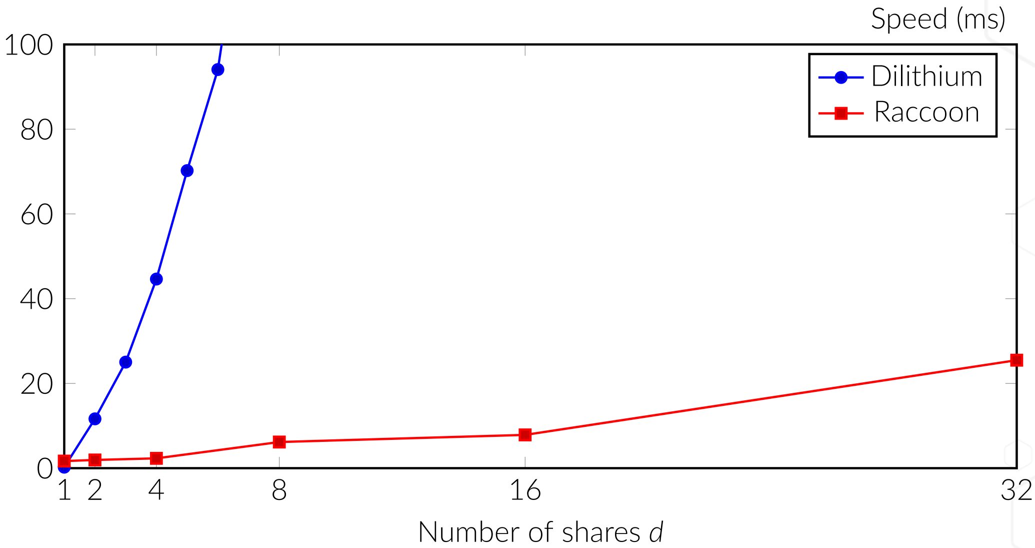
1 Removing rejection sampling increases $\|\mathbf{r}\|/\|\mathbf{s}\|$ from $\Theta(\dim \mathbf{s})$ to $\Theta\left(\|c\|\sqrt{\text{Queries}}\right)$

Impact on the modulus



1. Removing rejection sampling increases $\|\mathbf{r}\|/\|\mathbf{s}\|$ from $\Theta(\dim \mathbf{s})$ to $\Theta\left(\|c\|\sqrt{\text{Queries}}\right)$
2. The increased q in turn requires increasing $\|\mathbf{s}\|$, $q/\|\mathbf{r}\|$ and/or the dimensions.

Performances on a Desktop



With some tricks [SR23], RAM consumption is < 128 kB

Conclusion

Raccoon is a specific-purpose scheme aimed at high side-channel resistance:

- 😊 Same assumptions as Dilithium
- 😊 Simpler
- 😊 Verification key size is similar
- ☹️ Signature is 4x larger
- 🤩 **When masked, orders of magnitude faster than other schemes are**

Questions?

Dmitri Asonov and Rakesh Agrawal.
Keyboard acoustic emanations.
In *2004 IEEE Symposium on Security and Privacy*, pages 3–11. IEEE Computer Society Press, May 2004.

Melissa Azouaoui, Olivier Bronchain, Gaëtan Cassiers, Clément Hoffmann, Yulia Kuzovkova, Joost Renes, Tobias Schneider, Markus Schönauer, François-Xavier Standaert, and Christine van Vredendaal.
Protecting dilithium against leakage: Revisited sensitivity analysis and improved implementations.
IACR Transactions on Cryptographic Hardware and Embedded Systems, 2023(4):58–79, Aug. 2023.

Gilles Barthe, Sonia Belaïd, Thomas Espitau, Pierre-Alain Fouque, Benjamin Grégoire, Mélissa Rossi, and Mehdi Tibouchi.
Masking the GLP lattice-based signature scheme at any order.
In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 354–384. Springer, Heidelberg, April / May 2018.

Jean-Sébastien Coron, François Gérard, Matthias Trannoy, and Rina Zeitoun.
Improved gadgets for the high-order masking of dilithium.

IACR Transactions on Cryptographic Hardware and Embedded Systems, 2023(4):110–145, Aug. 2023.

Julien Devevey, Pouria Fallahpour, Alain Passelègue, and Damien Stehlé.
A detailed analysis of fiat-shamir with aborts.
In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023*, pages 327–357, Cham, 2023. Springer Nature Switzerland.

Wim Van Eck.
Electromagnetic radiation from video display units: An eavesdropping risk?
Computers & Security, 4:269–286, 1985.

Thomas Espitau, Pierre-Alain Fouque, François Gérard, Mélissa Rossi, Akira Takahashi, Mehdi Tibouchi, Alexandre Wallet, and Yang Yu.
Mitaka: A simpler, parallelizable, maskable variant of falcon.
In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part III*, volume 13277 of *LNCS*, pages 222–253. Springer, Heidelberg, May / June 2022.

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.

Paul C. Kocher, Joshua Jaffe, and Benjamin Jun.
Differential power analysis.
In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 388–397. Springer, Heidelberg, August 1999.

Paul C. Kocher.
Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems.
In Neal Koblitz, editor, *CRYPTO'96*, volume 1109 of *LNCS*, pages 104–113. Springer, Heidelberg, August 1996.

Vincent Migliore, Benoît Gérard, Mehdi Tibouchi, and Pierre-Alain Fouque.
Masking Dilithium - efficient implementation and side-channel evaluation.
In Robert H. Deng, Valérie Gauthier-Umaña, Martín Ochoa, and Moti Yung, editors, *ACNS 19*, volume 11464 of *LNCS*, pages 344–362. Springer, Heidelberg, June 2019.

Thomas Prest.
A key-recovery attack against mitaka in the t-probing model.

In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *PKC 2023, Part I*, volume 13940 of *LNCS*, pages 205–220. Springer, Heidelberg, May 2023.

Markku-Juhani O. Saarinen and Mélissa Rossi.
Mask compression: High-order masking on memory-constrained devices.
Cryptology ePrint Archive, Paper 2023/1117, 2023.
https://eprint.iacr.org/2023/1117.

Shiduo Zhang, Xiuhan Lin, Yang Yu, and Weijia Wang.
Improved power analysis attacks on falcon.
Cryptology ePrint Archive, Paper 2023/224, 2023.
https://eprint.iacr.org/2023/224.