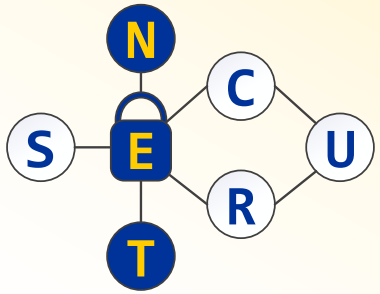


Enhancing Cross-Sectoral Collaboration in Cybersecurity in
Estonia, Czechia, Lithuania, Ukraine, and the Netherlands

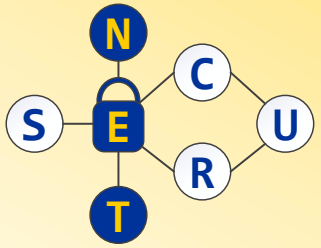
Funded by
the European Union

Protecting Dilithium with Mixed-Radix Representations

Secondment from RUN to CYBER
(May-June 2026)

Funded by the European Union Horizon Europe programme under Grant Agreement No. 101217315. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or European Research Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.

Post Quantum Cryptography



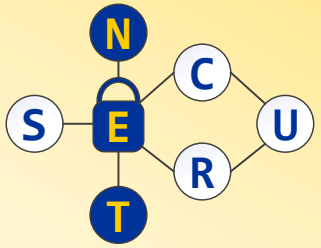
Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer*

Peter W. Shor†

PQC submissions (different families, based on different mathematical problems):

- code-based
- Isogeny-based
- Hash-based (SLH-DSA)
- Lattice-based (ML-KEM, ML-DSA)
- Multivariate Quadratic Equations

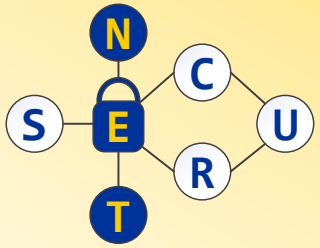
Practical Deployment Scenario



- Implementations deployed in real-world devices
  - Embedded platforms & Field Programmable Gate Arrays (FPGAs)
- Allows physical proximity to the target
- Mathematically secure ≠ physically secure
- Implementation-based Attacks

  - Target the specific implementation & not mathematical hardness
    - Side-channel attacks (*passive*) – adversary observes output and physical emanations
    - Fault injection attacks (*active*) – adversary perturbs during execution

Dilithium (ML-DSA) Overview



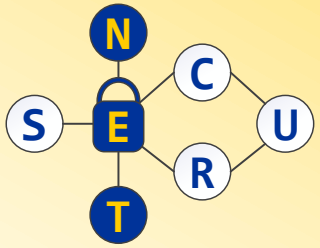
Signature generation. To sign a message $M \in \{0,1\}^*$, Alice does the following:

8 Compute $A = \text{ExpandA}(\rho)$.
9 Compute $\mu = H(tr \| M,\ 512)$.
10 Compute $\rho'' = H(K \| \text{rnd} \| \mu,\ 512)$ where either $\text{rnd} = 0^{256}$ or $\text{rnd} \in_R \{0,1\}^{256}$.
11 Set $\kappa = 0$ and Found = false.
12 **while** Found = false **do**
13 Compute $y = \text{ExpandMask}(\rho'', \kappa) \in \tilde{S}^{\ell}_{\gamma_1}$.
14 Compute $w = Ay$ and $w_1 = \text{HighBits}_0(w, 2\gamma_2)$.
15 Compute $\tilde{c} = H(\mu \| w_1, 2\lambda)$ and $c = \text{SampleInBall}(\tilde{c})$.
16 Compute $r_0 = \text{LowBits}_0(w - cs_2, 2\gamma_2)$.
17 **if** $\|r_0\|_\infty < \gamma_2 - \beta$ **then**
18 Compute $z = y + cs_1$.
19 **if** $\|z\|_\infty < \gamma_1 - \beta$ **then**
20 Found $\leftarrow$ true.
21 $\kappa \leftarrow \kappa + \ell$ (increment the counter κ).
22 **return** $(\sigma \leftarrow (\tilde{c}, z))$.

Key generation. Alice does the following:

1 Select $\xi \in_R \{0,1\}^{256}$.
2 Compute $(\rho, \rho', K) = H(\xi, 1024)$, where $\rho \in \{0,1\}^{256}$, $\rho' \in \{0,1\}^{512}$, $K \in \{0,1\}^{256}$.
3 Compute $A = \text{ExpandA}(\rho) \in R_q^{k \times \ell}$.
4 Compute $(s_1, s_2) = \text{ExpandS}(\rho') \in S_\eta^{\ell} \times S_\eta^{k}$.
5 Compute $t = As_1 + s_2$.
6 Compute $tr = H(\rho \| t,\ 2\lambda)$.
7 Alice's verification (public) key is (ρ, t); her signing (private) key is (ρ, K, tr, s_1, s_2).

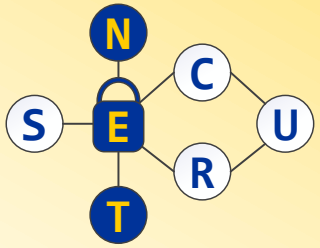
Dilithium (ML-DSA) Overview

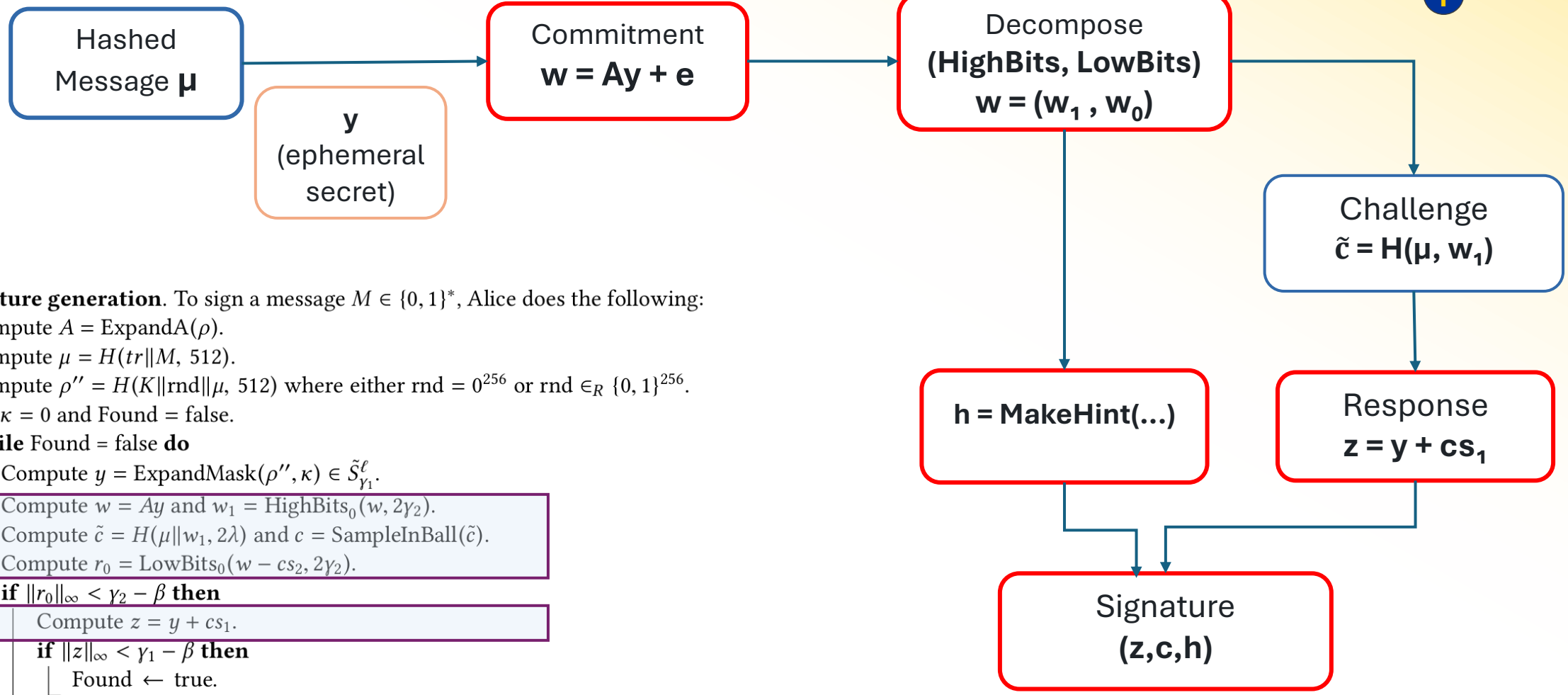


Signature generation. To sign a message $M \in \{0,1\}^*$, Alice does the following:

8 Compute $A = \text{ExpandA}(\rho)$.
9 Compute $\mu = H(tr \| M,\ 512)$.
10 Compute $\rho'' = H(K \| \text{rnd} \| \mu,\ 512)$ where either $\text{rnd} = 0^{256}$ or $\text{rnd} \in_R \{0,1\}^{256}$.
11 Set $\kappa = 0$ and Found = false.
12 **while** Found = false **do**
13 Compute $y = \text{ExpandMask}(\rho'', \kappa) \in \tilde{S}^{\ell}_{\gamma_1}$.
14 Compute $w = Ay$ and $w_1 = \text{HighBits}_0(w, 2\gamma_2)$.
15 Compute $\tilde{c} = H(\mu \| w_1, 2\lambda)$ and $c = \text{SampleInBall}(\tilde{c})$.
16 Compute $r_0 = \text{LowBits}_0(w - cs_2, 2\gamma_2)$.
17 **if** $\|r_0\|_\infty < \gamma_2 - \beta$ **then**
18 Compute $z = y + cs_1$.
19 **if** $\|z\|_\infty < \gamma_1 - \beta$ **then**
20 Found $\leftarrow$ true.
21 $\kappa \leftarrow \kappa + \ell$ (increment the counter κ).
22 **return** $(\sigma \leftarrow (\tilde{c}, z))$.

Dilithium: Secret Flow

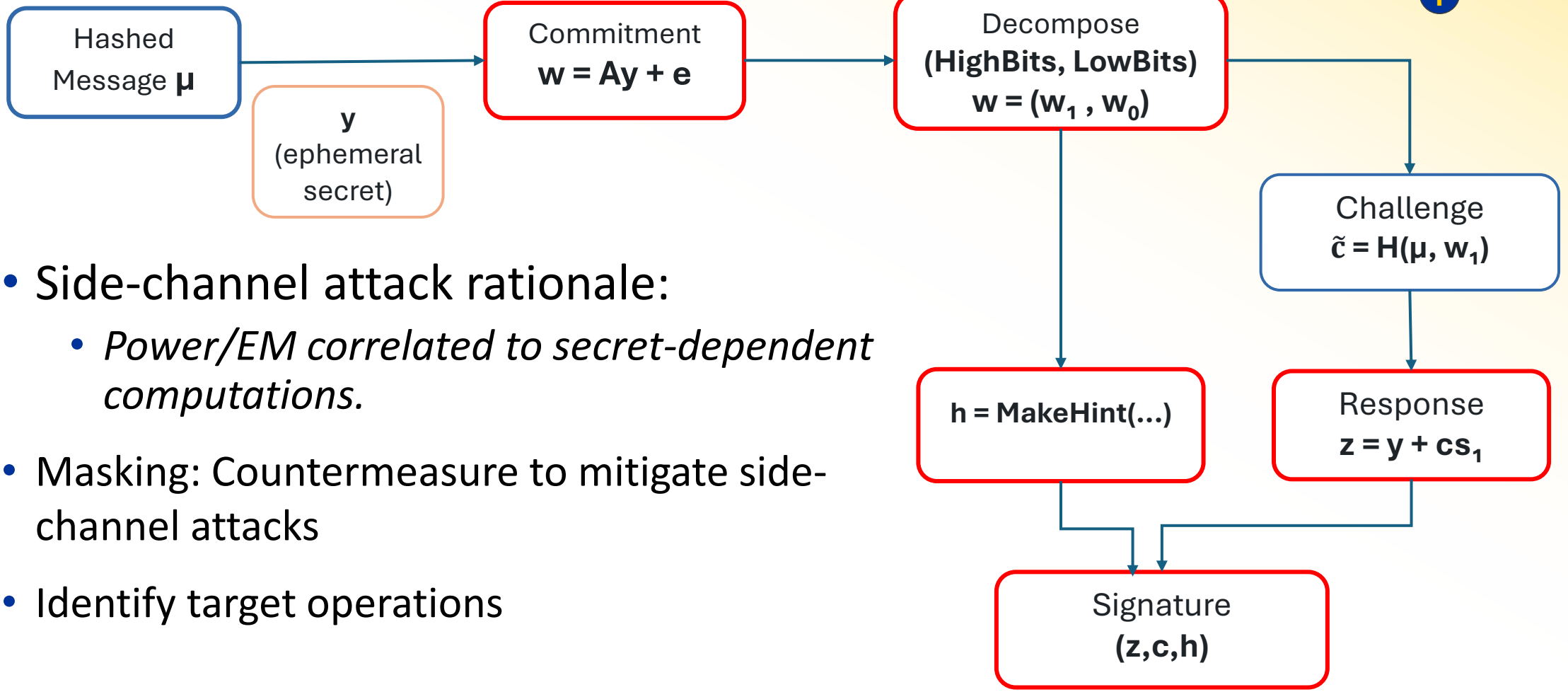




Signature generation. To sign a message $M \in \{0,1\}^*$, Alice does the following:

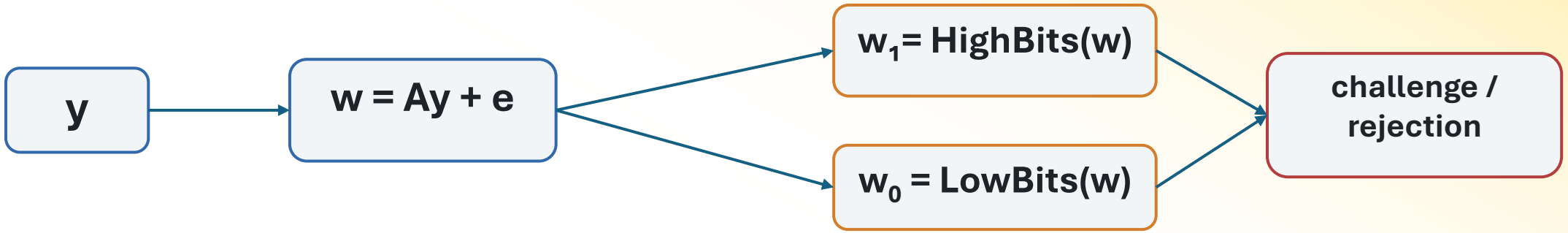
8 Compute $A = \text{ExpandA}(\rho)$.
9 Compute $\mu = H(tr \| M, 512)$.
10 Compute $\rho'' = H(K \| \text{rnd} \| \mu, 512)$ where either $\text{rnd} = 0^{256}$ or $\text{rnd} \in_R \{0,1\}^{256}$.
11 Set $\kappa = 0$ and Found = false.
12 **while** Found = false **do**
13 Compute $y = \text{ExpandMask}(\rho'', \kappa) \in \tilde{S}^{\ell}_{\gamma_1}$.
14 Compute $w = Ay$ and $w_1 = \text{HighBits}_0(w, 2\gamma_2)$.
15 Compute $\tilde{c} = H(\mu \| w_1, 2\lambda)$ and $c = \text{SampleInBall}(\tilde{c})$.
16 Compute $r_0 = \text{LowBits}_0(w - cs_2, 2\gamma_2)$.
17 **if** $\|r_0\|_\infty < \gamma_2 - \beta$ **then**
18 Compute $z = y + cs_1$.
19 **if** $\|z\|_\infty < \gamma_1 - \beta$ **then**
20 Found $\leftarrow$ true.
21 $\kappa \leftarrow \kappa + \ell$ (increment the counter κ).
22 **return** $(\sigma \leftarrow (\tilde{c}, z))$.

Dilithium: Secret Flow

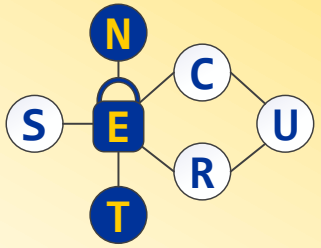


- Side-channel attack rationale:
  - *Power/EM correlated to secret-dependent computations.*
- Masking: Countermeasure to mitigate side-channel attacks
- Identify target operations

Dilithium: Secrets involved in Decomposition

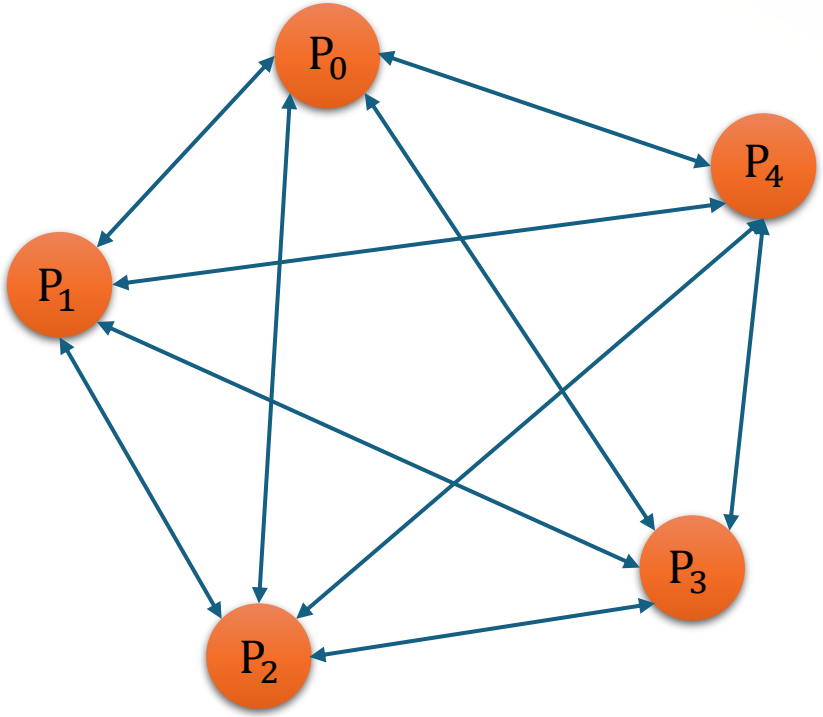


- Masking splits intermediates into multiple random shares.
  - Individual shares should reveal no information about the secret.
- Mitigates side-channel leakage from secret-dependent computations.
- Requires masked gadgets (operating on shares)
- Dilithium involves two masking domains:
  - Arithmetic Masking for (operations such as NTT, NTT_inv, Matrix multiplications)
  - Boolean Masking for (SHAKE)
  - Requires A2B and B2A gagets
- *Can there be another efficient representation that can decouple the side-channel leakage from secret-dependent operations?*



Can an MPC-inspired representation be used to reduce side-channel leakage?

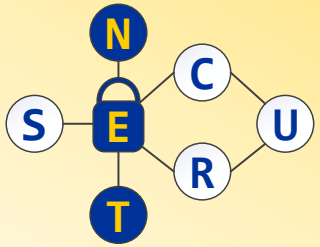
Secure Multiparty Computation



Additive secret sharing:

1. Generate shares of x
   $(x_0, x_1, x_2, x_3, x_4)$
2. Sharing variable x among parties
   $P_i \leftarrow x_i$
3. Recover
   $x = x_0 + x_1 + x_2 + x_3 + x_4$

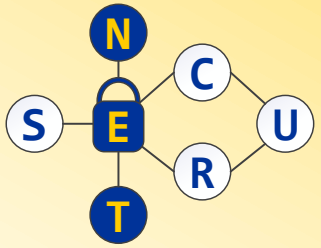
Mixed-Radix Representation in Trilithium



- Trilithium: Distributed Dilithium signing
- Fixed radix: $x = x_0 + x_1 B + x_2 B^2 + \ldots$
- Mixed radix: $x = x_0 + x_1 B_0 + x_2 B_0 B_1 + \ldots \ , 0 \leq x_i < B_i$
  - Different digits can have different ranges.
- Enables efficient representation and manipulation of secret-shared values in distributed signing.
- Key observation: the same value can be represented using a different set of digits and radices.
- Research question: *Can this alternative representation also change the side-channel leakage of Dilithium computations?*

Reference: Antonin Dufka, Semjon Kravtšenko, Peeter Laud, and Nikita Snetkov. "Trilithium: Efficient and universally composable distributed ML-DSA signing." *Cryptology ePrint Archive* (2025).

Mixed-Radix Representation



$$w = d_0 + r_0 d_1 + r_0 r_1 d_2 + r_0 r_1 r_2 d_3 + r_0 r_1 r_2 r_3 d_4$$

Digits

d_0 d_1 d_2 d_3 d_4

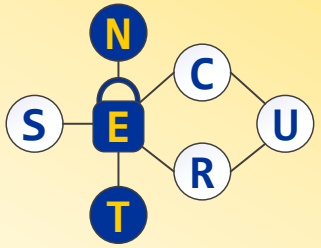
Mixed-radix representation

HighBits(w') = d_4

$$r_0\, r_1\, r_2\, r_3 = \alpha$$

- How does digit-wise computation impact side-channel leakage?

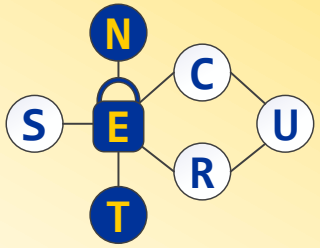
Gadgets using mixed-radix representation



- Several functions can reuse the mixed-radix representation

$$\text{Arithmetic} \to \boxed{\text{MixedRadix}} \to \begin{cases} \text{HighBits} \\ \text{LowBits} \\ \text{MakeHint} \\ \text{UseHint} \end{cases} \to \text{Arithmetic}$$

Next Steps



- Formal Security Evaluation
  - Evaluate the security of multi-radix gadgets in threshold probing model
- Software Implementation on ARM Cortex-M4
  - Implement HighBits, Decompose in the mixed-radix representation
  - Performance overhead computation + comparison with state-of-the-art
- Practical Security Evaluation using power side-channel
  - Compare TVLA results between unprotected and mixed-radix gadgets
  - Deep Learning-based Side Channel Analysis?