A Single-Trace Fault Injection Attack on Hedged Module Lattice Digital Signature Algorithm (ML-DSA)

Sönke Jendral, John Preuß Mattsson, Elena Dubrova

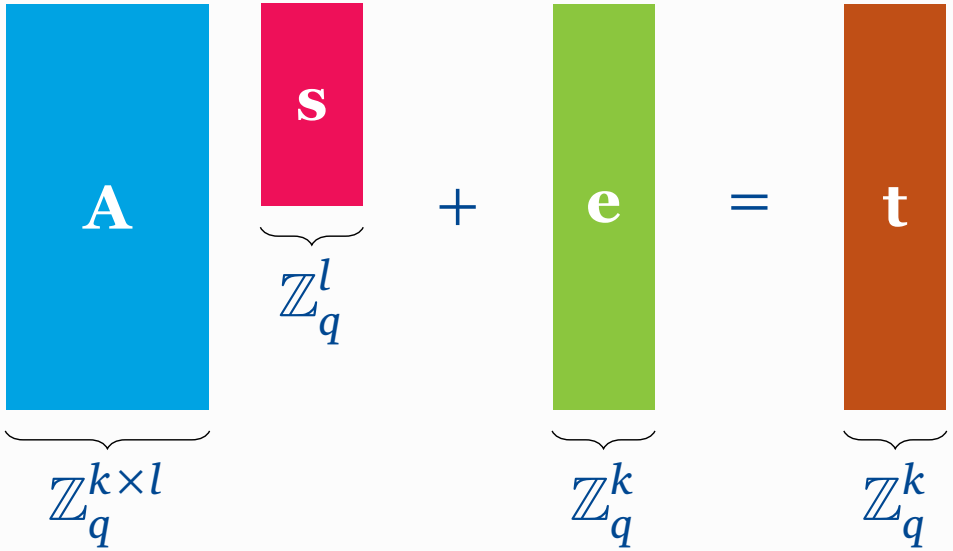
September 4, 2024 — KTH & Ericsson Research

Post-quantum cryptography

- Quantum computing: Shor's algorithm may break public-key cryptography
- Post-quantum cryptography: New algorithms, secure against quantum attacks
- NIST competition: Several algorithms currently being standardised
- Soon widely deployed:
  - 3GPP/IETF are working on integration into 5G and internet standards
  - Required by CNSA 2.0 by 2025
  - Signature algorithms: Firmware signing, PKI

- How can we build secure implementations?
- How are current implementations insecure?

Learning with Errors (LWE) [Reg05]

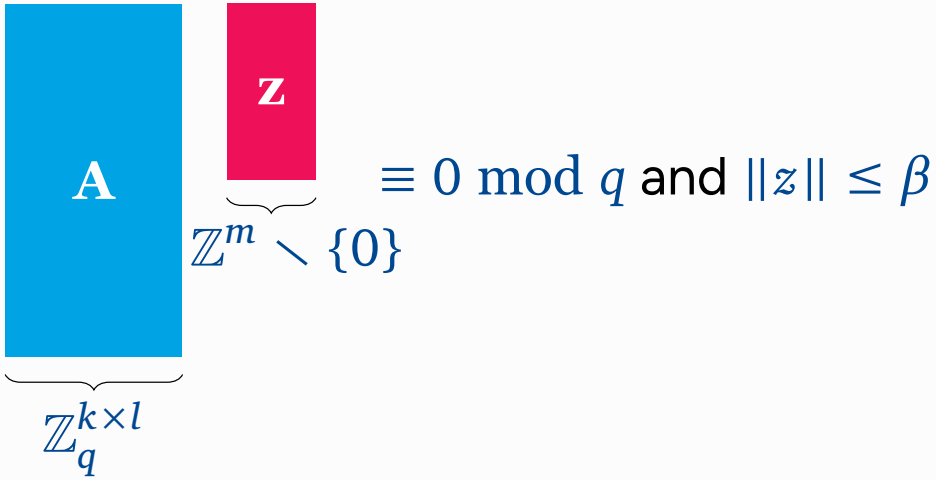


Given A and t:

- Search: Find s
- Decision: Is t random?

Hardness: Classically and quantumly NP-hard*

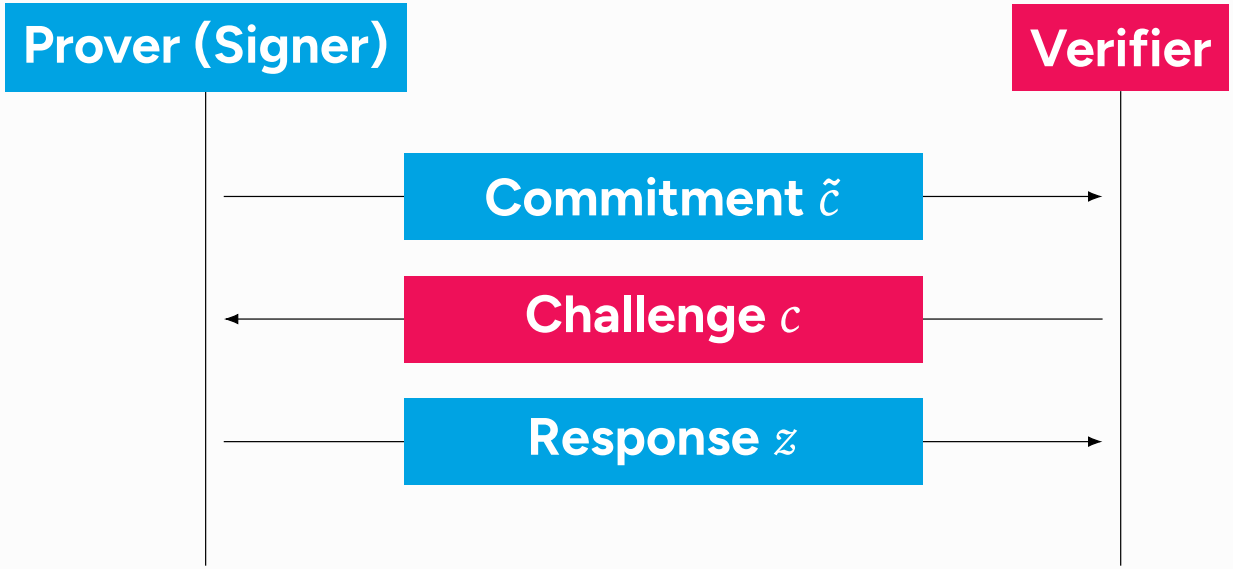
Short Integer Solution (SIS) [MR07]



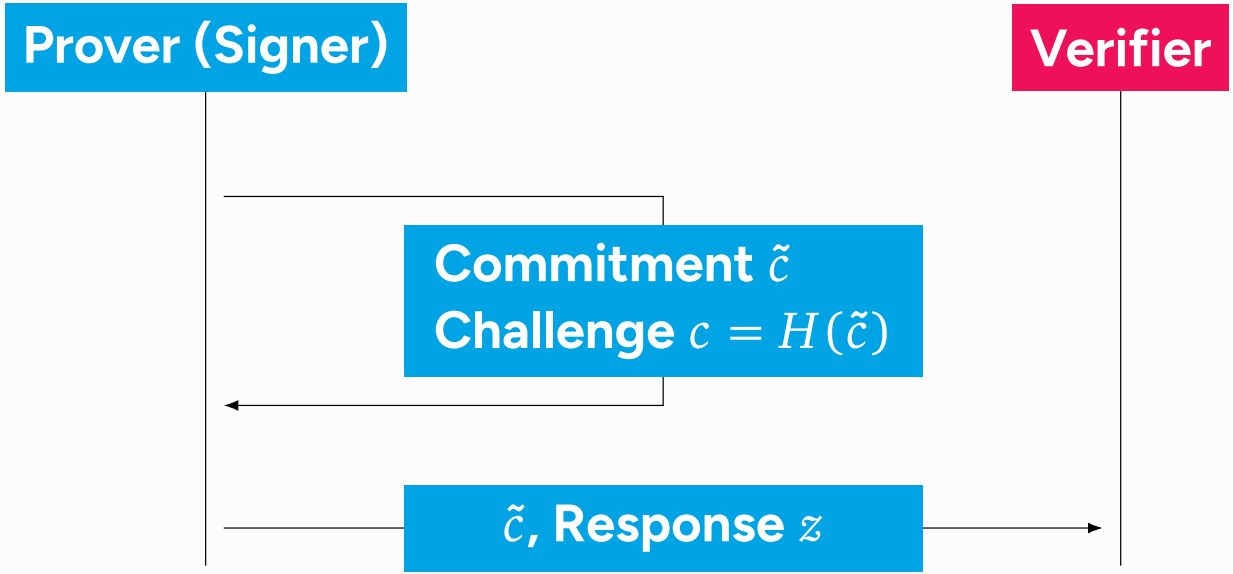
Given A and β, find z

Hardness: Classically and quantumly NP-hard*

Digital signatures & Fiat-Shamir transform



Digital signatures & Fiat-Shamir transform



Lattice-based digital signatures [Lyu09; Lyu12]

- Key generation: Secret $s_1 \in R_q^l, s_2 \in R_q^k$, Public $a \in R_q^{k \times l}$ and $t = as_1 + s_2$.

- Signing: Masks $y_1 \in R_q^l, y_2 \in R_q^k$ uniformly at random.
  Commitment $\tilde{c} \leftarrow ay_1 + y_2$, Challenge $c \leftarrow H(\tilde{c}, \mu)$
  Responses $z_1 = cs_1 + y_1$ and $z_2 = cs_2 + y_2$.
  Reject z_1, z_2 if outside safe range and try again.

- Verification: $c' = H(az_1 + z_2 - tc, \mu)$. Accept if $c' = c$ and $\|z_1\|, \|z_2\| \leq \gamma$.

Lattice-based digital signatures [Lyu09; Lyu12]

- Key generation: Secret $s_1 \in R_q^l, s_2 \in R_q^k$, Public $a \in R_q^{k \times l}$ and $t = as_1 + s_2$.

- Signing: Masks $y_1 \in R_q^l, y_2 \in R_q^k$ uniformly at random.
  Commitment $\tilde{c} \leftarrow ay_1 + y_2$, Challenge $c \leftarrow H(\tilde{c}, \mu)$
  Responses $z_1 = cs_1 + y_1$ and $z_2 = cs_2 + y_2$.
  Reject z_1, z_2 if outside safe range and try again.

- Verification: $c' = H(az_1 + z_2 - tc, \mu)$. Accept if $c' = c$ and $\|z_1\|, \|z_2\| \leq \gamma$.

Lattice-based digital signatures [Lyu09; Lyu12]

- Key generation: Secret $s_1 \in R_q^l, s_2 \in R_q^k$, Public $a \in R_q^{k \times l}$ and $t = as_1 + s_2$.

- Signing: Masks $y_1 \in R_q^l, y_2 \in R_q^k$ uniformly at random.
  Commitment $\tilde{c} \leftarrow ay_1 + y_2$, Challenge $c \leftarrow H(\tilde{c}, \mu)$
  Responses $z_1 = cs_1 + y_1$ and $z_2 = cs_2 + y_2$.
  Reject z_1, z_2 if outside safe range and try again.

Cannot get s_1 from t (LWE)

- Verification: $c' = H(az_1 + z_2 - tc, \mu)$. Accept if $c' = c$ and $\|z_1\|, \|z_2\| \leq \gamma$.

Lattice-based digital signatures [Lyu09; Lyu12]

- Key generation: Secret $s_1 \in R_q^l, s_2 \in R_q^k$, Public $a \in R_q^{k \times l}$ and $t = as_1 + s_2$.

- Signing: Masks $y_1 \in R_q^l, y_2 \in R_q^k$ uniformly at random.
  Commitment $\tilde{c} \leftarrow ay_1 + y_2$, Challenge $c \leftarrow H(\tilde{c}, \mu)$
  Responses $z_1 = cs_1 + y_1$ and $z_2 = cs_2 + y_2$.
  Reject z_1, z_2 if outside safe range and try again.

- Verification: $c' = H(az_1 + z_2 - tc, \mu)$. Accept if $c' = c$ and $\|z_1\|, \|z_2\| \leq \gamma$.

Lattice-based digital signatures [Lyu09; Lyu12]

- Key generation: Secret $s_1 \in R_q^l, s_2 \in R_q^k$, Public $a \in R_q^{k \times l}$ and $t = as_1 + s_2$.

- Signing: Masks $y_1 \in R_q^l, y_2 \in R_q^k$ uniformly at random.
  Commitment $\tilde{c} \leftarrow ay_1 + y_2$, Challenge $c \leftarrow H(\tilde{c}, \mu)$
  Responses $z_1 = cs_1 + y_1$ and $z_2 = cs_2 + y_2$.
  Reject z_1, z_2 if outside safe range and try again.

- Verification: $c' = H(az_1 + z_2 - tc, \mu)$. Accept if $c' = c$ and $\|z_1\|, \|z_2\| \leq \gamma$.

Lattice-based digital signatures [Lyu09; Lyu12]

- Key generation: Secret $s_1 \in R_q^l, s_2 \in R_q^k$, Public $a \in R_q^{k \times l}$ and $t = as_1 + s_2$.

- Signing: Masks $y_1 \in R_q^l, y_2 \in R_q^k$ uniformly at random.
  Commitment $\tilde{c} \leftarrow ay_1 + y_2$, Challenge $c \leftarrow H(\tilde{c}, \mu)$
  Responses $z_1 = cs_1 + y_1$ and $z_2 = cs_2 + y_2$.
  Reject z_1, z_2 if outside safe range and try again.

- Verification: $c' = H(az_1 + z_2 - tc, \mu)$. Accept if $c' = c$ and $\|z_1\|, \|z_2\| \leq \gamma$.

Lattice-based digital signatures [Lyu09; Lyu12]

- Key generation: Secret $s_1 \in R_q^l, s_2 \in R_q^k$, Public $a \in R_q^{k \times l}$ and $t = as_1 + s_2$.

- Signing: Masks $y_1 \in R_q^l, y_2 \in R_q^k$ uniformly at random.
  Commitment $\tilde{c} \leftarrow ay_1 + y_2$, Challenge $c \leftarrow H(\tilde{c}, \mu)$
  Responses $z_1 = cs_1 + y_1$ and $z_2 = cs_2 + y_2$. ← Cannot get s_1, s_2 from z_1, z_2 (LWE)
  Reject z_1, z_2 if outside safe range and try again.

- Verification: $c' = H(az_1 + z_2 - tc, \mu)$. Accept if $c' = c$ and $\|z_1\|, \|z_2\| \leq \gamma$.

Lattice-based digital signatures [Lyu09; Lyu12]

- Key generation: Secret $s_1 \in R_q^l, s_2 \in R_q^k$, Public $a \in R_q^{k \times l}$ and $t = as_1 + s_2$.

- Signing: Masks $y_1 \in R_q^l, y_2 \in R_q^k$ uniformly at random.
  Commitment $\tilde{c} \leftarrow ay_1 + y_2$, Challenge $c \leftarrow H(\tilde{c}, \mu)$
  Responses $z_1 = cs_1 + y_1$ and $z_2 = cs_2 + y_2$.
  Reject z_1, z_2 if outside safe range and try again.

- Verification: $c' = H(az_1 + z_2 - tc, \mu)$. Accept if $c' = c$ and $\|z_1\|, \|z_2\| \leq \gamma$.

Lattice-based digital signatures [Lyu09; Lyu12]

- Key generation: Secret $s_1 \in R_q^l, s_2 \in R_q^k$, Public $a \in R_q^{k \times l}$ and $t = as_1 + s_2$.

- Signing: Masks $y_1 \in R_q^l, y_2 \in R_q^k$ uniformly at random.
  Commitment $\tilde{c} \leftarrow ay_1 + y_2$, Challenge $c \leftarrow H(\tilde{c}, \mu)$
  Responses $z_1 = cs_1 + y_1$ and $z_2 = cs_2 + y_2$.
  Reject z_1, z_2 if outside safe range and try again.

- Verification: $c' = H(az_1 + z_2 - tc, \mu)$. Accept if $c' = c$ and $\|z_1\|, \|z_2\| \leq \gamma$.

$$az_1 + z_2 - tc = a(cs_1 + y_1) + cs_2 + y_2 - (as_1 + s_2)c = ay_1 + y_2 = \tilde{c}$$

Lattice-based digital signatures [Lyu09; Lyu12]

- Key generation: Secret $s_1 \in R_q^l, s_2 \in R_q^k$, Public $a \in R_q^{k \times l}$ and $t = as_1 + s_2$.

- Signing: Masks $y_1 \in R_q^l, y_2 \in R_q^k$ uniformly at random.
  Commitment $\tilde{c} \leftarrow ay_1 + y_2$, Challenge $c \leftarrow H(\tilde{c}, \mu)$
  Responses $z_1 = cs_1 + y_1$ and $z_2 = cs_2 + y_2$.
  Reject z_1, z_2 if outside safe range and try again.

- Verification: $c' = H(az_1 + z_2 - tc, \mu)$. Accept if $c' = c$ and $\|z_1\|, \|z_2\| \leq \gamma$.

Cannot compute z_1, z_2 without s_1, s_2 (SIS)

ML-DSA signing (simplified)

Input: Private key sk, message M
Output: Signature σ

$(\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0) \leftarrow \text{skDecode}(sk)$
$\mathbf{A} \leftarrow \text{ExpandA}(\rho)$
$\mu \leftarrow \text{H}(tr \parallel M, 512)$
$rnd \leftarrow \{0,1\}^{256}$
$\rho' \leftarrow \text{H}(K \parallel rnd \parallel \mu, 512)$
$\kappa \leftarrow 0$
$(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
while $(\mathbf{z}, \mathbf{h}) = \perp$ **do**
    $\mathbf{y} \leftarrow \text{ExpandMask}(\rho', \kappa)$
    $\tilde{c} \leftarrow \mathbf{Ay}$
    $c \leftarrow \text{H}(\tilde{c} \parallel \mu)$
    $\mathbf{z} \leftarrow \mathbf{y} + cs_1$
    if $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ **then** $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
    $\kappa \leftarrow \kappa + l$
$\sigma \leftarrow \text{sigEncode}(\tilde{c}, \mathbf{z} \bmod^{\pm} q)$
return σ

ML-DSA signing (simplified)

Input: Private key sk, message M
Output: Signature σ

1: $(\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0) \leftarrow \text{skDecode}(sk)$
2: $\mathbf{A} \leftarrow \text{ExpandA}(\rho)$
3: $\mu \leftarrow \text{H}(tr \parallel M, 512)$
4: $rnd \leftarrow \{0,1\}^{256}$
5: $\rho' \leftarrow \text{H}(K \parallel rnd \parallel \mu, 512)$
6: $\kappa \leftarrow 0$
7: $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
8: **while** $(\mathbf{z}, \mathbf{h}) = \perp$ **do**
9: $\quad \mathbf{y} \leftarrow \text{ExpandMask}(\rho', \kappa)$
10: $\quad \tilde{c} \leftarrow \mathbf{Ay}$
11: $\quad c \leftarrow \text{H}(\tilde{c} \parallel \mu)$
12: $\quad \mathbf{z} \leftarrow \mathbf{y} + cs_1$
13: $\quad$ **if** $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ **then** $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
14: $\quad \kappa \leftarrow \kappa + l$
15: $\sigma \leftarrow \text{sigEncode}(\tilde{c}, \mathbf{z} \bmod^{\pm} q)$
16: **return** σ

ML-DSA signing (simplified)

Input: Private key sk, message M
Output: Signature σ

1: $(\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0) \leftarrow \text{skDecode}(sk)$
2: $\mathbf{A} \leftarrow \text{ExpandA}(\rho)$
3: $\mu \leftarrow \text{H}(tr \parallel M, 512)$
4: $rnd \leftarrow \{0,1\}^{256}$
5: $\rho' \leftarrow \text{H}(K \parallel rnd \parallel \mu, 512)$
6: $\kappa \leftarrow 0$
7: $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
8: **while** $(\mathbf{z}, \mathbf{h}) = \perp$ **do**
9: $\quad \mathbf{y} \leftarrow \text{ExpandMask}(\rho', \kappa)$
10: $\quad \tilde{c} \leftarrow \mathbf{Ay}$
11: $\quad c \leftarrow \text{H}(\tilde{c} \parallel \mu)$
12: $\quad \mathbf{z} \leftarrow \mathbf{y} + cs_1$
13: $\quad$ **if** $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ **then** $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
14: $\quad \kappa \leftarrow \kappa + l$
15: $\sigma \leftarrow \text{sigEncode}(\tilde{c}, \mathbf{z} \bmod^{\pm} q)$
16: **return** σ

ML-DSA signing (simplified)

Input: Private key sk, message M
Output: Signature σ

1: $(\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0) \leftarrow \text{skDecode}(sk)$
2: $\mathbf{A} \leftarrow \text{ExpandA}(\rho)$
3: $\mu \leftarrow \text{H}(tr \parallel M, 512)$
4: $rnd \leftarrow \{0,1\}^{256}$
5: $\rho' \leftarrow \text{H}(K \parallel rnd \parallel \mu, 512)$
6: $\kappa \leftarrow 0$
7: $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
8: **while** $(\mathbf{z}, \mathbf{h}) = \perp$ **do**
9: $\quad \mathbf{y} \leftarrow \text{ExpandMask}(\rho', \kappa)$
10: $\quad \tilde{c} \leftarrow \mathbf{Ay}$
11: $\quad c \leftarrow \text{H}(\tilde{c} \parallel \mu)$
12: $\quad \mathbf{z} \leftarrow \mathbf{y} + cs_1$
13: $\quad$ **if** $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ **then** $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
14: $\quad \kappa \leftarrow \kappa + l$
15: $\sigma \leftarrow \text{sigEncode}(\tilde{c}, \mathbf{z} \bmod^{\pm} q)$
16: **return** σ

ML-DSA signing (simplified)

Input: Private key sk, message M
Output: Signature σ

1: $(\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0) \leftarrow \text{skDecode}(sk)$
2: $\mathbf{A} \leftarrow \text{ExpandA}(\rho)$
3: $\mu \leftarrow \text{H}(tr \parallel M, 512)$
4: $rnd \leftarrow \{0,1\}^{256}$
5: $\rho' \leftarrow \text{H}(K \parallel rnd \parallel \mu, 512)$
6: $\kappa \leftarrow 0$
7: $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
8: **while** $(\mathbf{z}, \mathbf{h}) = \perp$ **do**
9: $\quad \mathbf{y} \leftarrow \text{ExpandMask}(\rho', \kappa)$
10: $\quad \tilde{c} \leftarrow \mathbf{Ay}$
11: $\quad c \leftarrow \text{H}(\tilde{c} \parallel \mu)$
12: $\quad \mathbf{z} \leftarrow \mathbf{y} + cs_1$
13: $\quad$ **if** $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ **then** $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
14: $\quad \kappa \leftarrow \kappa + l$
15: $\sigma \leftarrow \text{sigEncode}(\tilde{c}, \mathbf{z} \bmod^{\pm} q)$
16: **return** σ

ML-DSA signing (simplified)

Input: Private key sk, message M
Output: Signature σ

1: $(\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0) \leftarrow \text{skDecode}(sk)$
2: $\mathbf{A} \leftarrow \text{ExpandA}(\rho)$
3: $\mu \leftarrow \text{H}(tr \parallel M, 512)$
4: $rnd \leftarrow \{0,1\}^{256}$
5: $\rho' \leftarrow \text{H}(K \parallel rnd \parallel \mu, 512)$
6: $\kappa \leftarrow 0$
7: $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
8: **while** $(\mathbf{z}, \mathbf{h}) = \perp$ **do**
9: $\quad \mathbf{y} \leftarrow \text{ExpandMask}(\rho', \kappa)$
10: $\quad \tilde{c} \leftarrow \mathbf{Ay}$
11: $\quad c \leftarrow \text{H}(\tilde{c} \parallel \mu)$
12: $\quad \mathbf{z} \leftarrow \mathbf{y} + cs_1$
13: $\quad$ **if** $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ **then** $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
14: $\quad \kappa \leftarrow \kappa + l$
15: $\sigma \leftarrow \text{sigEncode}(\tilde{c}, \mathbf{z} \bmod^{\pm} q)$
16: **return** σ

ML-DSA signing (simplified)

Input: Private key sk, message M
Output: Signature σ

$(\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0) \leftarrow \text{skDecode}(sk)$
$\mathbf{A} \leftarrow \text{ExpandA}(\rho)$
$\mu \leftarrow \text{H}(tr \parallel M, 512)$
$rnd \leftarrow \{0,1\}^{256}$
$\rho' \leftarrow \text{H}(K \parallel rnd \parallel \mu, 512)$
$\kappa \leftarrow 0$
$(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
while $(\mathbf{z}, \mathbf{h}) = \perp$ **do**
  $\mathbf{y} \leftarrow \text{ExpandMask}(\rho', \kappa)$
  $\tilde{c} \leftarrow \mathbf{Ay}$
  $c \leftarrow \text{H}(\tilde{c} \parallel \mu)$
  $\mathbf{z} \leftarrow \mathbf{y} + cs_1$
  if $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ **then** $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
  $\kappa \leftarrow \kappa + l$
$\sigma \leftarrow \text{sigEncode}(\tilde{c}, \mathbf{z} \bmod^{\pm} q)$
return σ

ML-DSA signing (simplified)

Input: Private key sk, message M
Output: Signature σ

1: $(\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0) \leftarrow \text{skDecode}(sk)$
2: $\mathbf{A} \leftarrow \text{ExpandA}(\rho)$
3: $\mu \leftarrow \text{H}(tr \parallel M, 512)$
4: $rnd \leftarrow \{0,1\}^{256}$
5: $\rho' \leftarrow \text{H}(K \parallel rnd \parallel \mu, 512)$
6: $\kappa \leftarrow 0$
7: $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
8: **while** $(\mathbf{z}, \mathbf{h}) = \perp$ **do**
9: $\quad \mathbf{y} \leftarrow \text{ExpandMask}(\rho', \kappa)$
10: $\quad \tilde{c} \leftarrow \mathbf{Ay}$
11: $\quad c \leftarrow \text{H}(\tilde{c} \parallel \mu)$
12: $\quad \mathbf{z} \leftarrow \mathbf{y} + cs_1$
13: $\quad$ **if** $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ **then** $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
14: $\quad \kappa \leftarrow \kappa + l$
15: $\sigma \leftarrow \text{sigEncode}(\tilde{c}, \mathbf{z} \bmod^{\pm} q)$
16: **return** σ

Hedged-mode ML-DSA

Input: Private key sk, message M
Output: Signature σ

1: $(\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0) \leftarrow \text{skDecode}(sk)$
2: $\mathbf{A} \leftarrow \text{ExpandA}(\rho)$
3: $\mu \leftarrow \text{H}(tr \parallel M, 512)$
4: $rnd \leftarrow \{0,1\}^{256}$
5: $\rho' \leftarrow \text{H}(K \parallel rnd \parallel \mu, 512)$
6: $\kappa \leftarrow 0$
7: $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$

- Private random seed ρ' used in $\mathbf{y} \leftarrow \text{ExpandMask}(\rho', \kappa)$
- "Hedged" mode

- Offers protection against RNG zeroisation
- Only in ML-DSA, not in CRYSTALS-Dilithium
  - not well studied …

SHAKE256

- e**X**tendable **O**utput **F**unction (**XOF**)
- Part of FIPS-202 (SHA-3)
- Based on Keccak family of permutations
- Uses sponge construction

- Used in ML-DSA to expand seeds, hash messages and sample matrices/vectors (directly or indirectly interacts with secret keys!)

Sponge constructions [Ber+11]

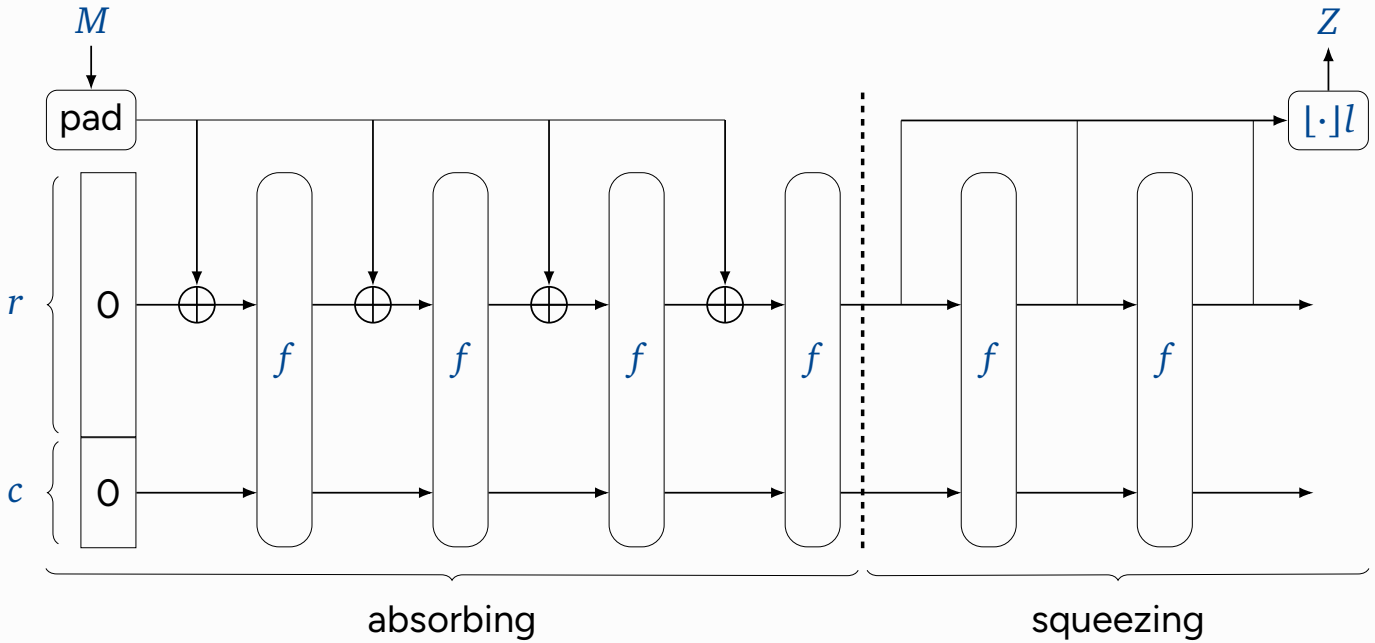


Figure adapted from [Ber+11]

Sponge constructions [Ber+11]

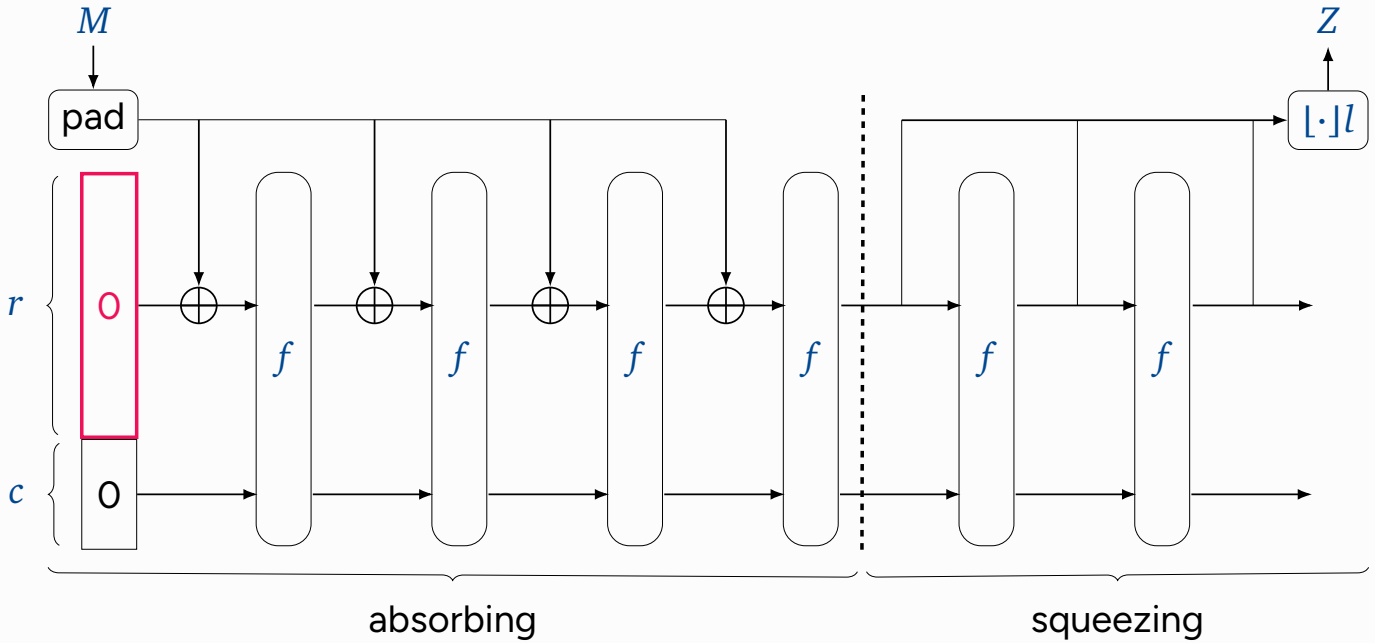


Figure adapted from [Ber+11]

Sponge constructions [Ber+11]

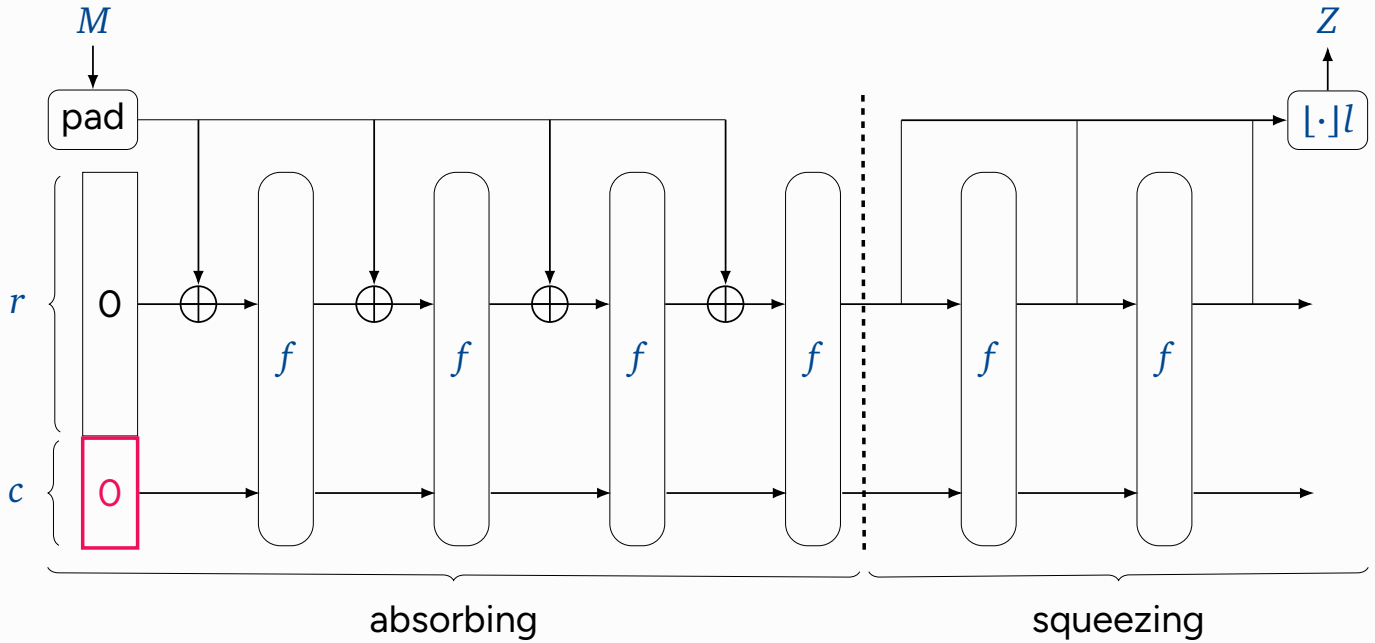


Figure adapted from [Ber+11]

Sponge constructions [Ber+11]

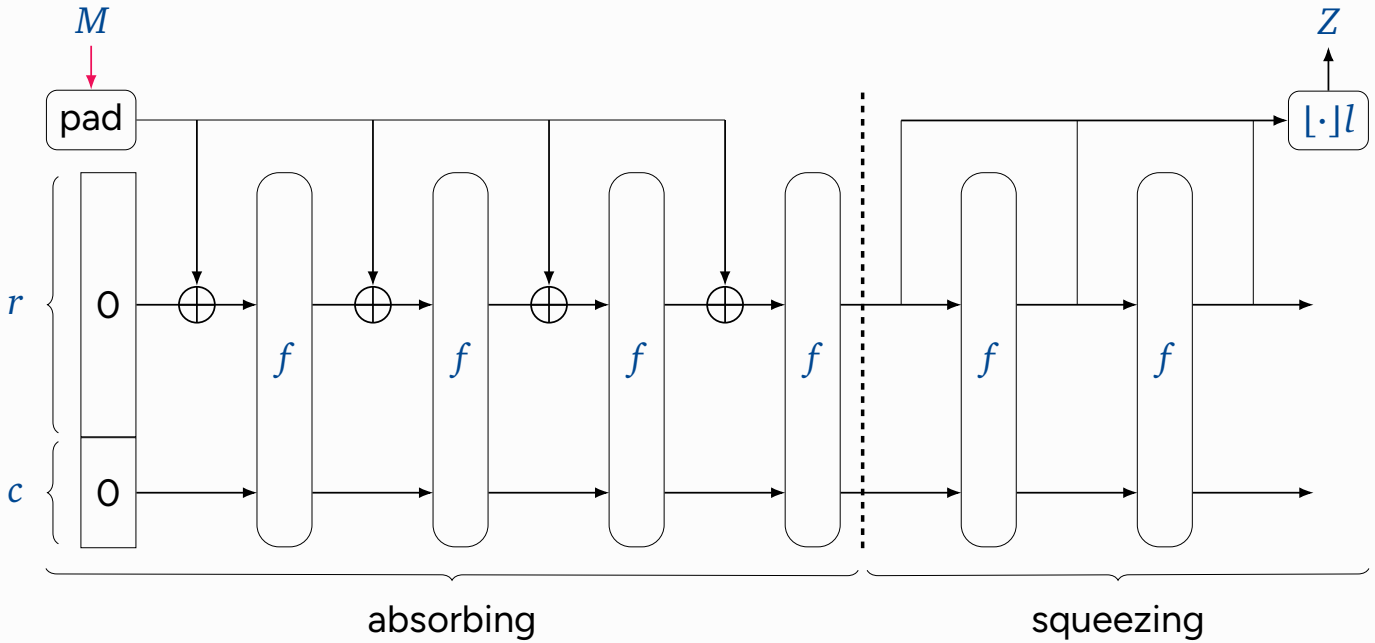


Figure adapted from [Ber+11]

Sponge constructions [Ber+11]

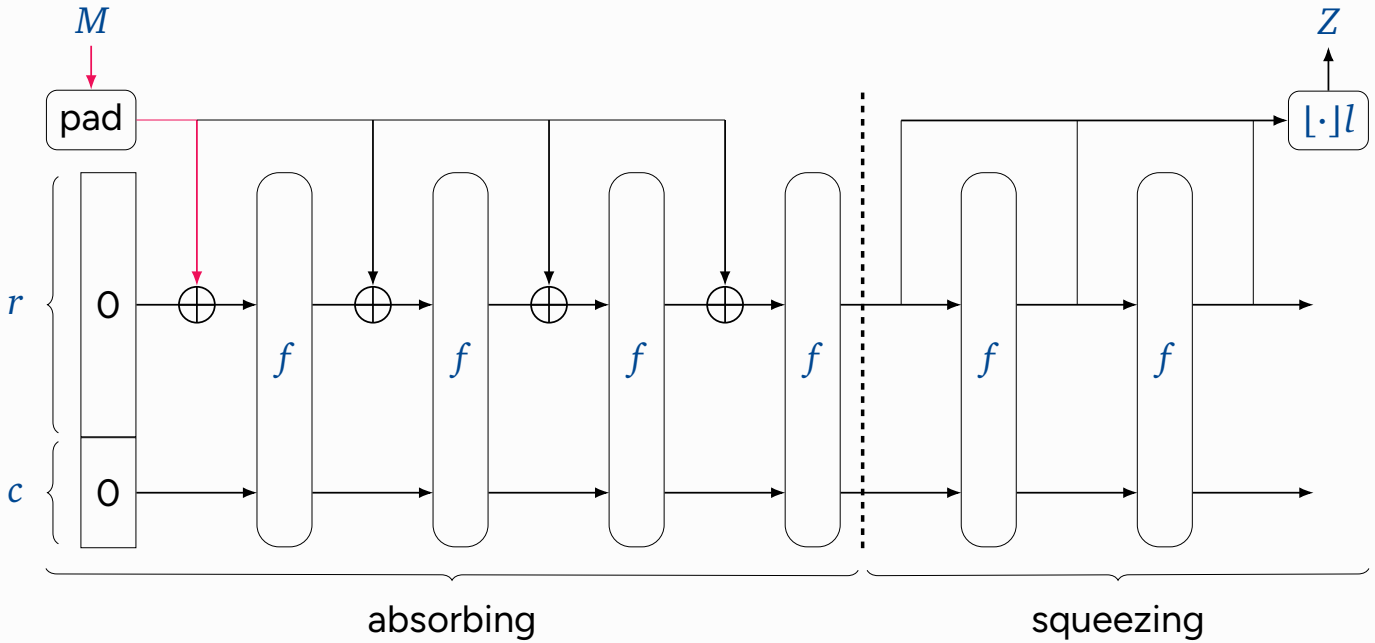


Figure adapted from [Ber+11]

Sponge constructions [Ber+11]

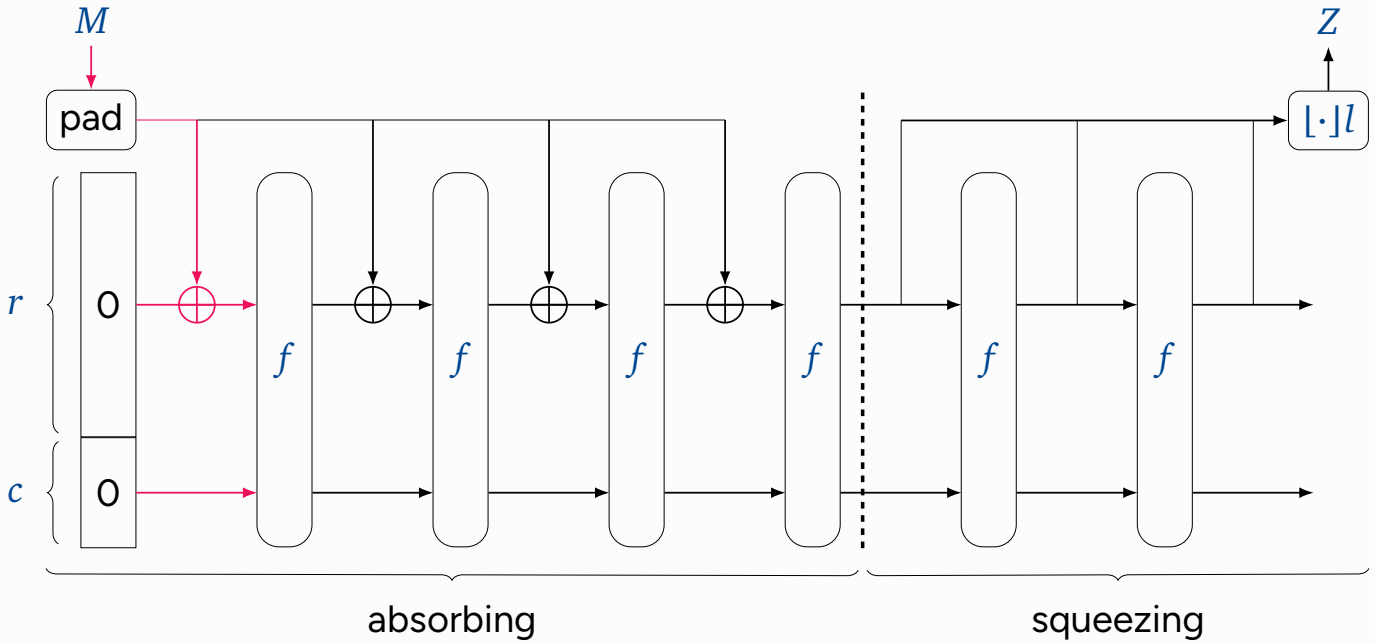


Figure adapted from [Ber+11]

Sponge constructions [Ber+11]

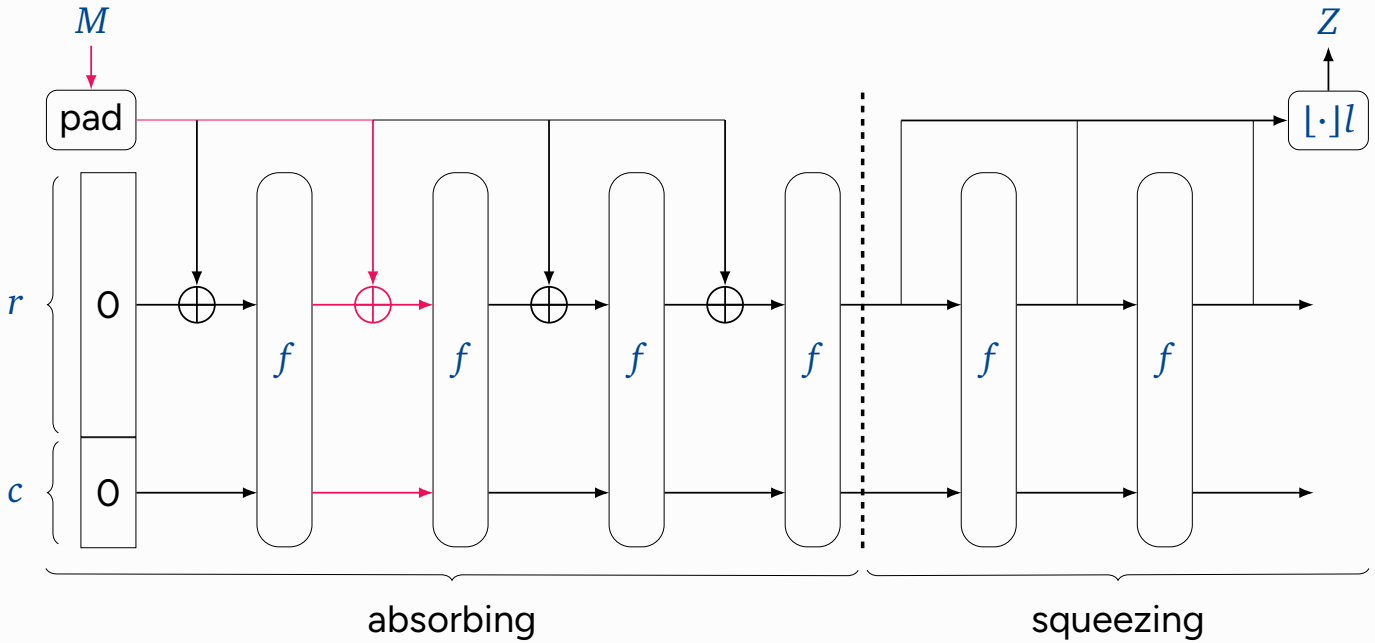


Figure adapted from [Ber+11]

Sponge constructions [Ber+11]

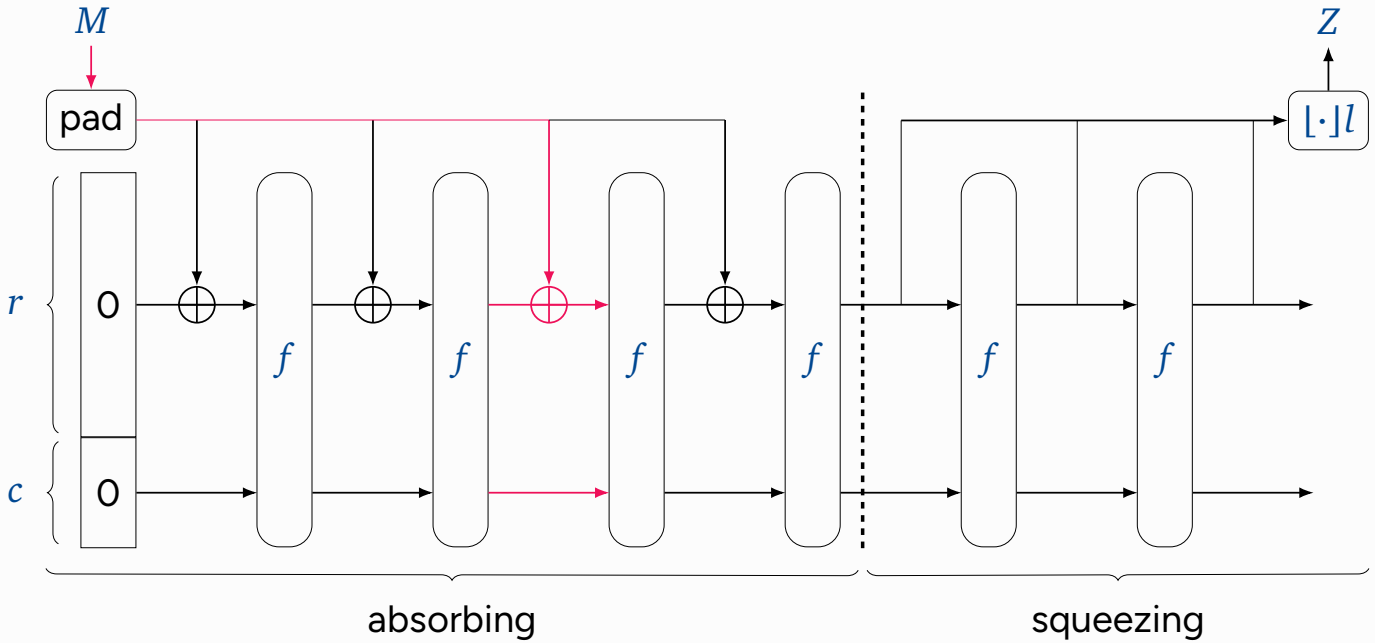


Figure adapted from [Ber+11]

Sponge constructions [Ber+11]

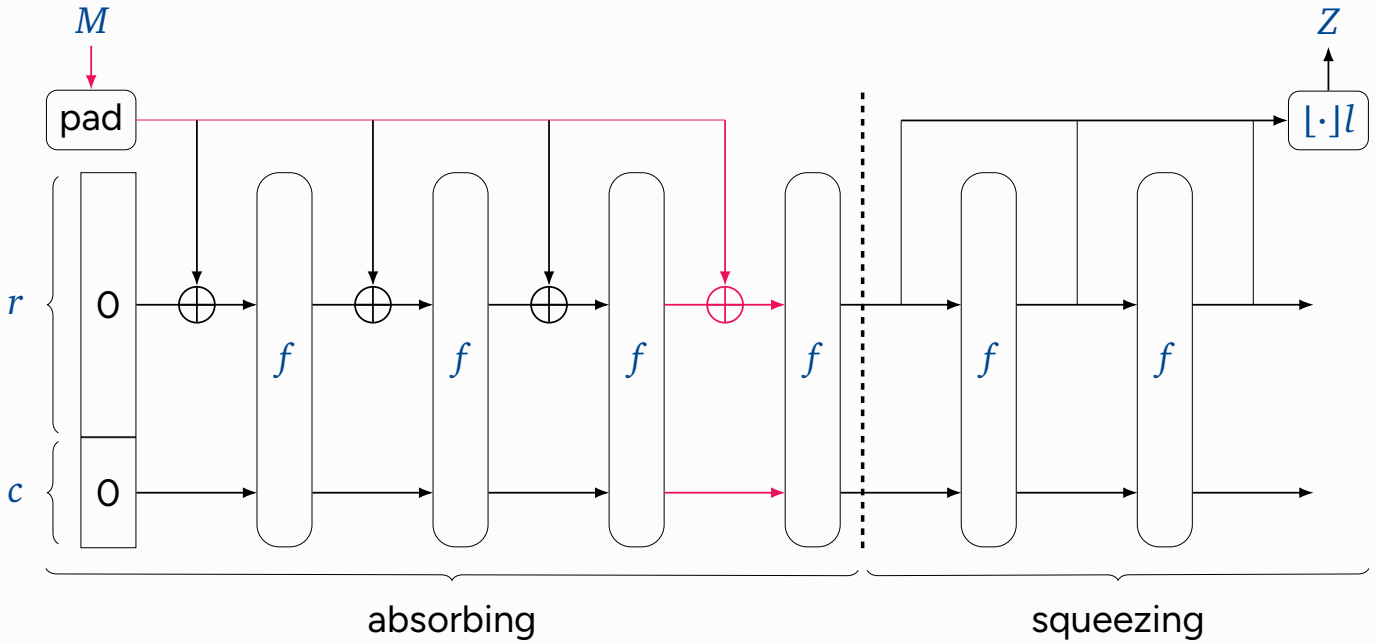


Figure adapted from [Ber+11]

Sponge constructions [Ber+11]

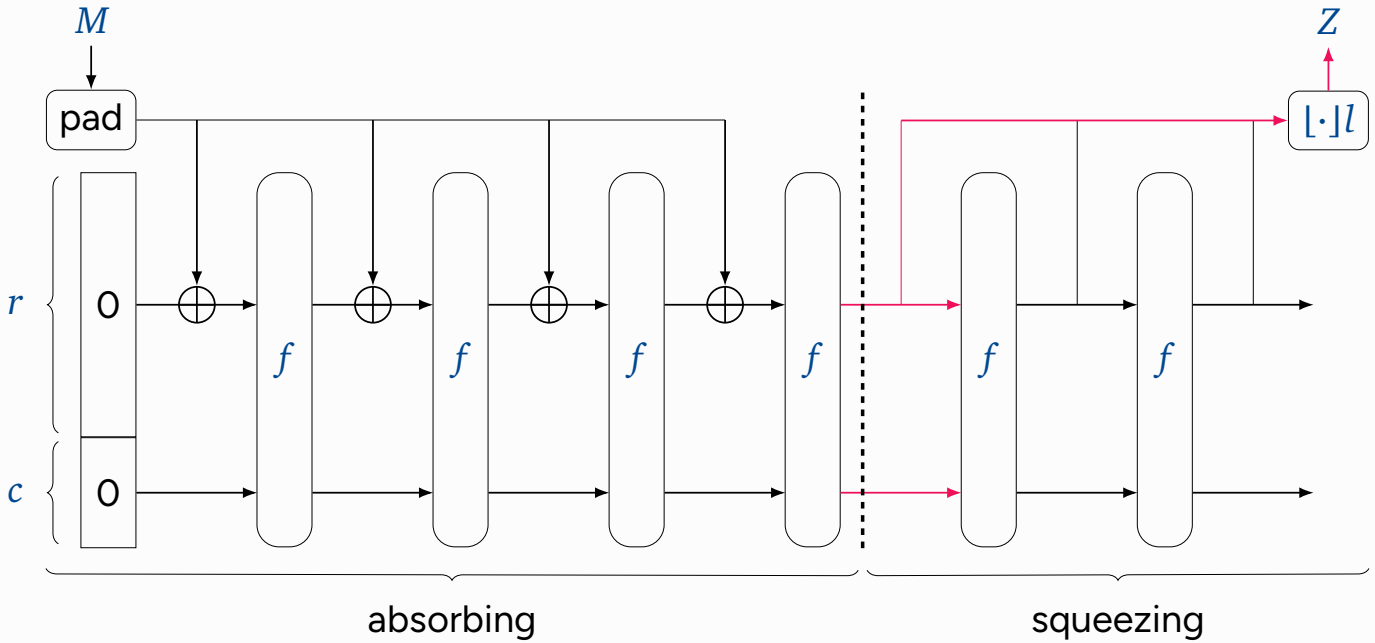


Figure adapted from [Ber+11]

Sponge constructions [Ber+11]

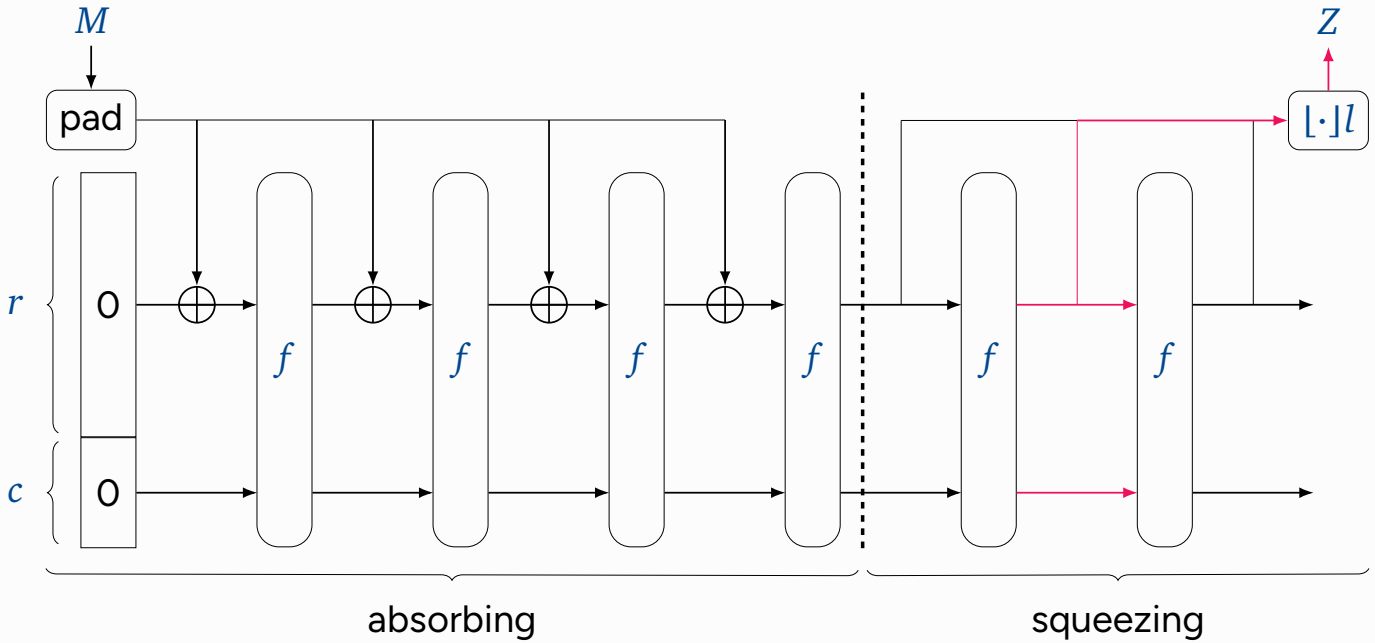


Figure adapted from [Ber+11]

Sponge constructions [Ber+11]

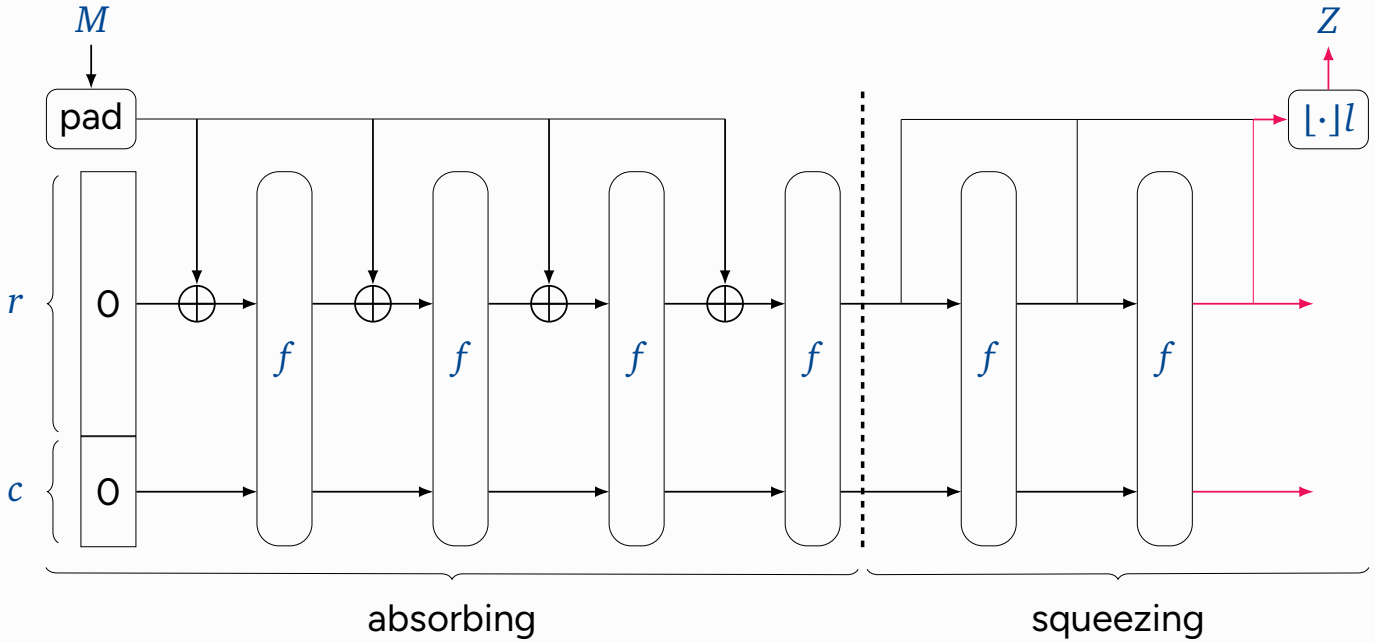


Figure adapted from [Ber+11]

Attack: Skipping absorption

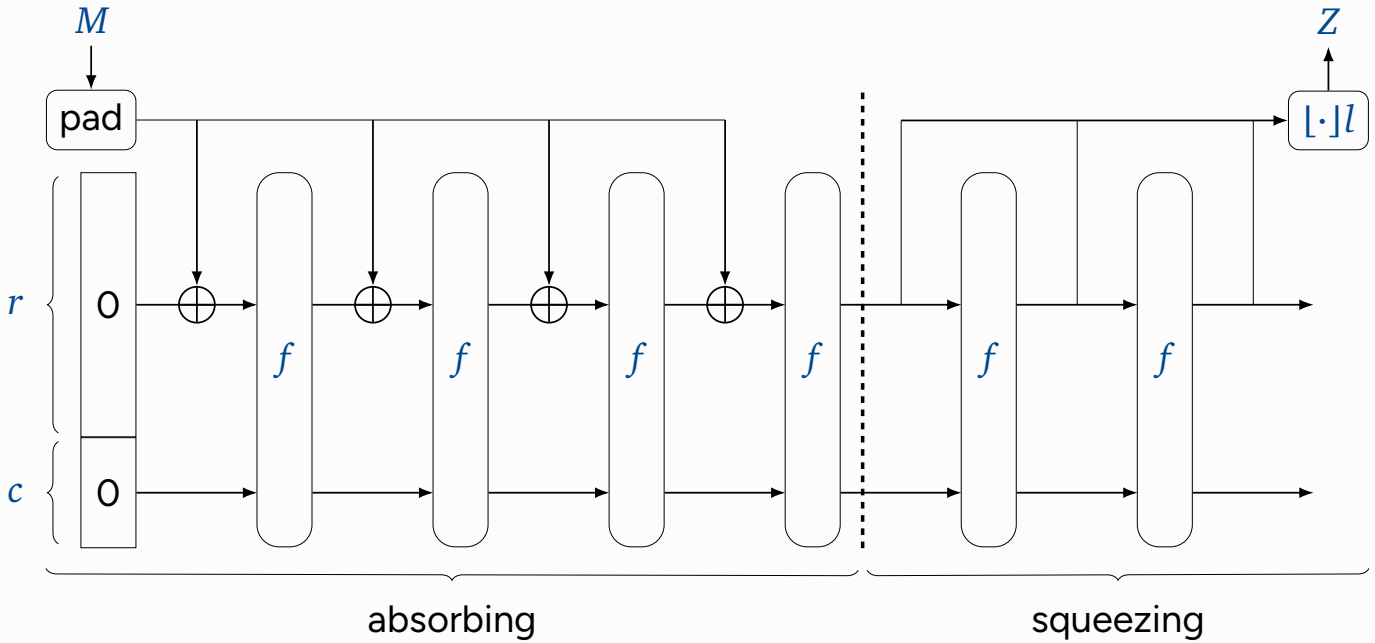


Figure adapted from [Ber+11]

Attack: Skipping absorption

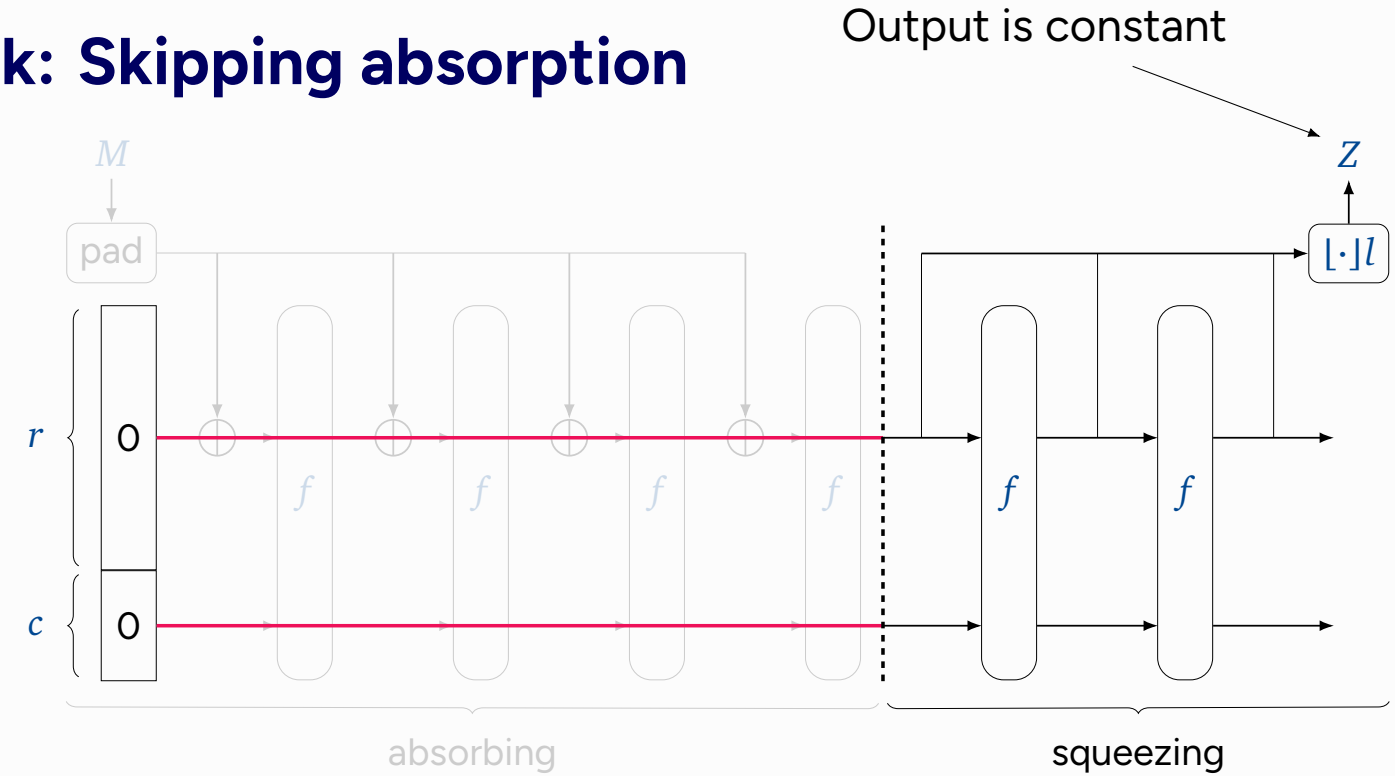


Figure adapted from [Ber+11]

Key recovery from known ρ'

Input: Private key sk, message M
Output: Signature σ

1: $(\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0) \leftarrow \text{skDecode}(sk)$
2: $\mathbf{A} \leftarrow \text{ExpandA}(\rho)$
3: $\mu \leftarrow \text{H}(tr \parallel M, 512)$
4: $rnd \leftarrow \{0,1\}^{256}$
5: $\rho' \leftarrow \text{H}(K \parallel rnd \parallel \mu, 512)$
6: $\kappa \leftarrow 0$
7: $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
8: **while** $(\mathbf{z}, \mathbf{h}) = \perp$ **do**
9: $\quad \mathbf{y} \leftarrow \text{ExpandMask}(\rho', \kappa)$
10: $\quad \tilde{c} \leftarrow \mathbf{Ay}$
11: $\quad c \leftarrow \text{H}(\tilde{c} \parallel \mu)$
12: $\quad \mathbf{z} \leftarrow \mathbf{y} + cs_1$
13: $\quad$ **if** $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ **then** $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
14: $\quad \kappa \leftarrow \kappa + l$
15: $\sigma \leftarrow \text{sigEncode}(\tilde{c}, \mathbf{z} \bmod^{\pm} q)$
16: **return** σ

Key recovery from known ρ'

Input: Private key sk, message M
Output: Signature σ

1: $(\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0) \leftarrow \text{skDecode}(sk)$
2: $\mathbf{A} \leftarrow \text{ExpandA}(\rho)$
3: $\mu \leftarrow \text{H}(tr \parallel M, 512)$
4: $rnd \leftarrow \{0,1\}^{256}$
5: $\rho' \leftarrow \text{H}(K \parallel rnd \parallel \mu, 512)$
6: $\kappa \leftarrow 0$
7: $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
8: **while** $(\mathbf{z}, \mathbf{h}) = \perp$ **do**
9: $\quad \mathbf{y} \leftarrow \text{ExpandMask}(\rho', \kappa)$
10: $\quad \tilde{c} \leftarrow \mathbf{Ay}$
11: $\quad c \leftarrow \text{H}(\tilde{c} \parallel \mu)$
12: $\quad \mathbf{z} \leftarrow \mathbf{y} + cs_1$
13: $\quad$ **if** $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ **then** $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
14: $\quad \kappa \leftarrow \kappa + l$
15: $\sigma \leftarrow \text{sigEncode}(\tilde{c}, \mathbf{z} \bmod^{\pm} q)$
16: **return** σ

Key recovery from known ρ'

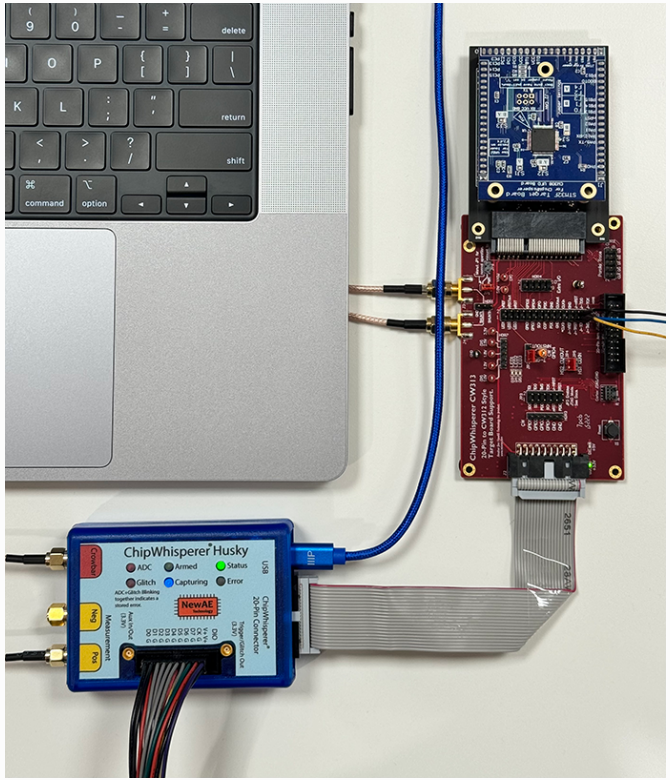
Input: Private key sk, message M
Output: Signature σ

1: $(\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0) \leftarrow \text{skDecode}(sk)$
2: $\mathbf{A} \leftarrow \text{ExpandA}(\rho)$
3: $\mu \leftarrow \text{H}(tr \parallel M, 512)$
4: $rnd \leftarrow \{0,1\}^{256}$
5: $\rho' \leftarrow \text{H}(K \parallel rnd \parallel \mu, 512)$
6: $\kappa \leftarrow 0$
7: $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
8: **while** $(\mathbf{z}, \mathbf{h}) = \perp$ **do**
9: $\quad \mathbf{y} \leftarrow \text{ExpandMask}(\rho', \kappa)$
10: $\quad \tilde{c} \leftarrow \mathbf{Ay}$
11: $\quad c \leftarrow \text{H}(\tilde{c} \parallel \mu)$
12: $\quad \mathbf{z} \leftarrow \mathbf{y} + cs_1$
13: $\quad$ **if** $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ **then** $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
14: $\quad \kappa \leftarrow \kappa + l$
15: $\sigma \leftarrow \text{sigEncode}(\tilde{c}, \mathbf{z} \bmod^{\pm} q)$
16: **return** σ

Fault injection method

ChipWhisperer-Husky:

- Inexpensive ($550), easy to use
- Clock glitching:
  - Inject/withhold rising edge in clock signal
- Voltage glitching:
  - Short power supply
- Requires precise timing



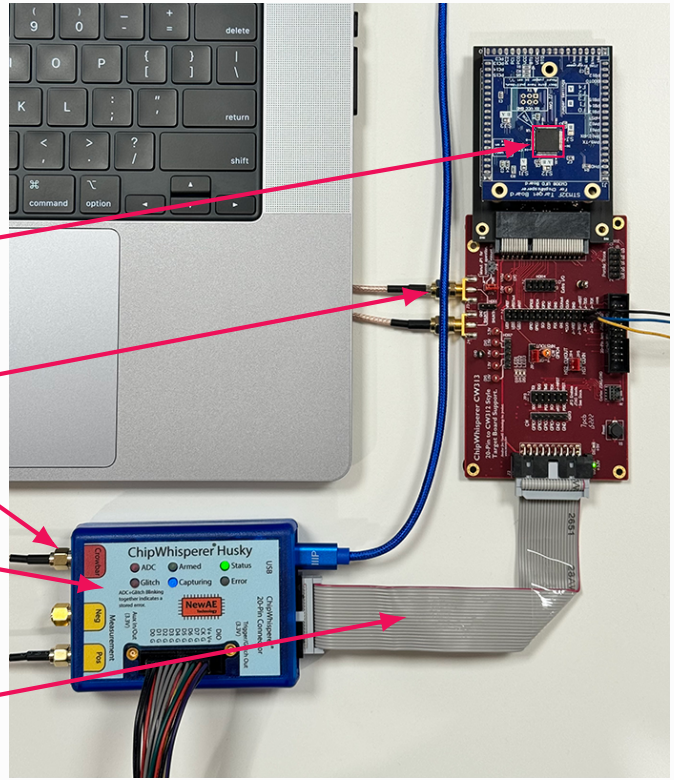
Fault injection method

ARM Cortex-M4 Target

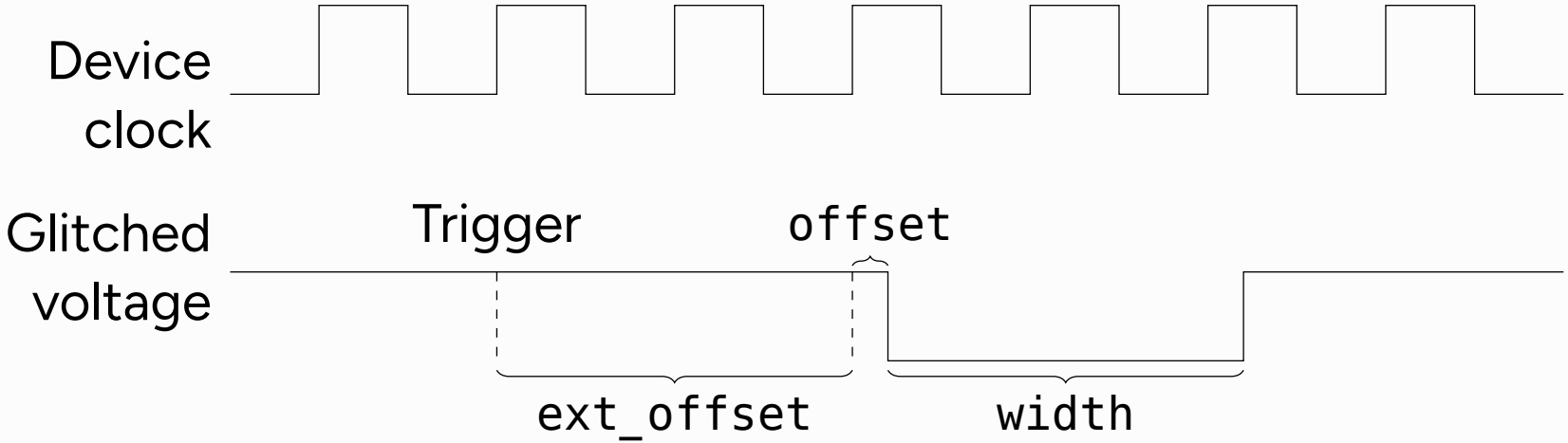
Glitched voltage

ChipWhisperer-Husky

Synchronised clock signal



Voltage glitching parameters



Attack: Skipping absorption

```
1  size_t keccak_inc_absorb(uint64_t *state, size_t n,
2                           uint8_t *m, size_t mlen) {
3    while (mlen + n >= 136) {
4      KeccakF1600_StateXORBytes(state, m, n);
5      mlen -= 136 - n;
6      m += 136 - n;
7      n = 0;
8      KeccakF1600_StatePermute(state);
9    }
10
11   KeccakF1600_StateXORBytes(state, m, n, mlen);
12   return n + mlen;
13 }
```

```
1   ...
2   mov r1, r8
3   mov r3, r4
4   mov r0, r7
5   bl KeccakF1600_StateXORBytes
6   ldr r2, [sp, #208]
7   ldr r3, [sp, #212]
8   ...
```

Results and countermeasures

Succeeds in a single trace with probability 52.8%

Countermeasures:

- Generic fault detection / CFI
- Eliminate branches
- Sample ρ' inside the rejection sampling loop
- Include rnd or K when sampling $\mathbf{y}$

Input: Private key sk, message M
Output: Signature σ

1: $(\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0) \leftarrow \text{skDecode}(sk)$
2: $\mathbf{A} \leftarrow \text{ExpandA}(\rho)$
3: $\mu \leftarrow \text{H}(tr \parallel M, 512)$
4: $rnd \leftarrow \{0,1\}^{256}$
5: $\rho' \leftarrow \text{H}(K \parallel rnd \parallel \mu, 512)$
6: $\kappa \leftarrow 0$
7: $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
8: **while** $(\mathbf{z}, \mathbf{h}) = \perp$ **do**
9: $\quad \mathbf{y} \leftarrow \text{ExpandMask}(\rho', \kappa)$
10: $\quad \tilde{c} \leftarrow \mathbf{A}\mathbf{y}$
11: $\quad c \leftarrow \text{H}(\tilde{c} \parallel \mu)$
12: $\quad \mathbf{z} \leftarrow \mathbf{y} + cs_1$
13: $\quad$ **if** $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ **then** $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
14: $\quad \kappa \leftarrow \kappa + l$
15: $\sigma \leftarrow \text{sigEncode}(\tilde{c}, \mathbf{z} \bmod^{\pm} q)$
16: **return** σ

Results and countermeasures

Succeeds in a single trace with probability 52.8%

Countermeasures:

- Generic fault detection / CFI
- Eliminate branches
- Sample ρ' inside the rejection sampling loop
- Include rnd or K when sampling $\mathbf{y}$

Input: Private key sk, message M
Output: Signature σ

1: $(\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0) \leftarrow \text{skDecode}(sk)$
2: $\mathbf{A} \leftarrow \text{ExpandA}(\rho)$
3: $\mu \leftarrow \text{H}(tr \parallel M, 512)$
4: $rnd \leftarrow \{0,1\}^{256}$
5: $\rho' \leftarrow \text{H}(K \parallel rnd \parallel \mu, 512)$
6: $\kappa \leftarrow 0$
7: $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
8: **while** $(\mathbf{z}, \mathbf{h}) = \perp$ **do**
9: $\quad \mathbf{y} \leftarrow \text{ExpandMask}(\rho', \kappa)$
10: $\quad \tilde{c} \leftarrow \mathbf{A}\mathbf{y}$
11: $\quad c \leftarrow \text{H}(\tilde{c} \parallel \mu)$
12: $\quad \mathbf{z} \leftarrow \mathbf{y} + cs_1$
13: $\quad$ **if** $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ **then** $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
14: $\quad \kappa \leftarrow \kappa + l$
15: $\sigma \leftarrow \text{sigEncode}(\tilde{c}, \mathbf{z} \bmod^{\pm} q)$
16: **return** σ

Results and countermeasures

Succeeds in a single trace with probability 52.8%

Countermeasures:

- Generic fault detection / CFI
- Eliminate branches
- Sample ρ' inside the rejection sampling loop
- Include rnd or K when sampling $\mathbf{y}$

Input: Private key sk, message M
Output: Signature σ

1: $(\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0) \leftarrow \text{skDecode}(sk)$
2: $\mathbf{A} \leftarrow \text{ExpandA}(\rho)$
3: $\mu \leftarrow \text{H}(tr \parallel M, 512)$
4: $rnd \leftarrow \{0,1\}^{256}$
5: $\rho' \leftarrow \text{H}(K \parallel rnd \parallel \mu, 512)$
6: $\kappa \leftarrow 0$
7: $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
8: **while** $(\mathbf{z}, \mathbf{h}) = \perp$ **do**
9: $\quad \mathbf{y} \leftarrow \text{ExpandMask}(\rho', \kappa)$
10: $\quad \tilde{c} \leftarrow \mathbf{A}\mathbf{y}$
11: $\quad c \leftarrow \text{H}(\tilde{c} \parallel \mu)$
12: $\quad \mathbf{z} \leftarrow \mathbf{y} + cs_1$
13: $\quad$ **if** $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ **then** $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
14: $\quad \kappa \leftarrow \kappa + l$
15: $\sigma \leftarrow \text{sigEncode}(\tilde{c}, \mathbf{z} \bmod^{\pm} q)$
16: **return** σ

Results and countermeasures

Succeeds in a single trace with probability 52.8%

Countermeasures:

- Generic fault detection / CFI
- Eliminate branches
- Sample ρ' inside the rejection sampling loop
- Include rnd or K when sampling $\mathbf{y}$

Input: Private key sk, message M
Output: Signature σ

1: $(\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0) \leftarrow \text{skDecode}(sk)$
2: $\mathbf{A} \leftarrow \text{ExpandA}(\rho)$
3: $\mu \leftarrow \text{H}(tr \parallel M, 512)$
4: $\kappa \leftarrow 0$
5: $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
6: **while** $(\mathbf{z}, \mathbf{h}) = \perp$ **do**
7: $\quad rnd \leftarrow \{0,1\}^{256}$
8: $\quad \rho' \leftarrow \text{H}(K \parallel rnd \parallel \mu, 512)$
9: $\quad \mathbf{y} \leftarrow \text{ExpandMask}(\rho', \kappa)$
10: $\quad \tilde{c} \leftarrow \mathbf{Ay}$
11: $\quad c \leftarrow \text{H}(\tilde{c} \parallel \mu)$
12: $\quad \mathbf{z} \leftarrow \mathbf{y} + cs_1$
13: $\quad$ **if** $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ **then** $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
14: $\quad \kappa \leftarrow \kappa + l$
15: $\sigma \leftarrow \text{sigEncode}(\tilde{c}, \mathbf{z} \bmod^{\pm} q)$
16: **return** σ

Results and countermeasures

Succeeds in a single trace with probability 52.8%

Countermeasures:

- Generic fault detection / CFI
- Eliminate branches
- Sample ρ' inside the rejection sampling loop
- Include rnd or K when sampling $\mathbf{y}$

Input: Private key sk, message M
Output: Signature σ

1: $(\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0) \leftarrow \text{skDecode}(sk)$
2: $\mathbf{A} \leftarrow \text{ExpandA}(\rho)$
3: $\mu \leftarrow \text{H}(tr \parallel M, 512)$
4: $rnd \leftarrow \{0,1\}^{256}$
5: $\rho' \leftarrow \text{H}(K \parallel rnd \parallel \mu, 512)$
6: $\kappa \leftarrow 0$
7: $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
8: **while** $(\mathbf{z}, \mathbf{h}) = \perp$ **do**
9: $\quad \mathbf{y} \leftarrow \text{ExpandMask}(\rho', \kappa)$
10: $\quad \tilde{c} \leftarrow \mathbf{A}\mathbf{y}$
11: $\quad c \leftarrow \text{H}(\tilde{c} \parallel \mu)$
12: $\quad \mathbf{z} \leftarrow \mathbf{y} + cs_1$
13: $\quad$ **if** $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ **then** $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
14: $\quad \kappa \leftarrow \kappa + l$
15: $\sigma \leftarrow \text{sigEncode}(\tilde{c}, \mathbf{z} \bmod^{\pm} q)$
16: **return** σ

Results and countermeasures

Succeeds in a single trace with probability 52.8%

Countermeasures:

- Generic fault detection / CFI
- Eliminate branches
- Sample ρ' inside the rejection sampling loop
- Include rnd or K when sampling $\mathbf{y}$

Input: Private key sk, message M
Output: Signature σ

1: $(\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0) \leftarrow \text{skDecode}(sk)$
2: $\mathbf{A} \leftarrow \text{ExpandA}(\rho)$
3: $\mu \leftarrow \text{H}(tr \parallel M, 512)$
4: $rnd \leftarrow \{0,1\}^{256}$
5: $\rho' \leftarrow \text{H}(K \parallel rnd \parallel \mu, 512)$
6: $\kappa \leftarrow 0$
7: $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
8: **while** $(\mathbf{z}, \mathbf{h}) = \perp$ **do**
9: $\quad \mathbf{y} \leftarrow \text{ExpandMask}(\rho', \kappa, rnd, K)$
10: $\quad \tilde{c} \leftarrow \mathbf{A}\mathbf{y}$
11: $\quad c \leftarrow \text{H}(\tilde{c} \parallel \mu)$
12: $\quad \mathbf{z} \leftarrow \mathbf{y} + cs_1$
13: $\quad$ **if** $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ **then** $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
14: $\quad \kappa \leftarrow \kappa + l$
15: $\sigma \leftarrow \text{sigEncode}(\tilde{c}, \mathbf{z} \bmod^{\pm} q)$
16: **return** σ

Conclusions & Future work

- Point-of-failure ρ' allows trivial single trace key recovery
- Fault injection attack makes strong assumptions:
  - Physical access, ability to precisely skip instructions
  - ChipWhisperer is idealised target

- No hedged ML-DSA implementations yet
- ML-DSA final standard: Init-Absorb-Squeeze now explicit
  - Implementations unlikely to provide multiple variants
  - Variable-sized hash function when only fixed-sized is required
- SHAKE256 used in other PQC algorithms


KTH
VETENSKAP
OCH KONST