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

Sönke Jendral, John Preuß Mattsson, Elena Dubrova

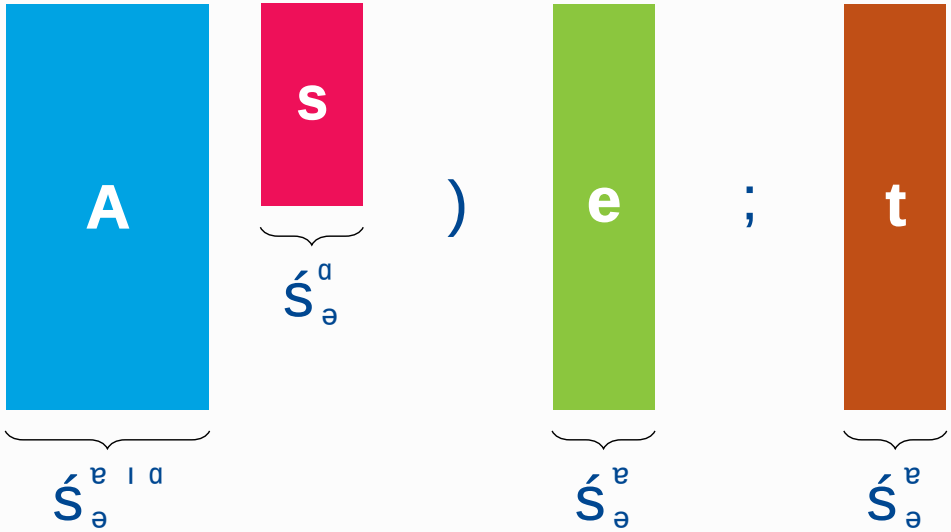
September 4, 2024 — KTH & Ericsson Research

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]

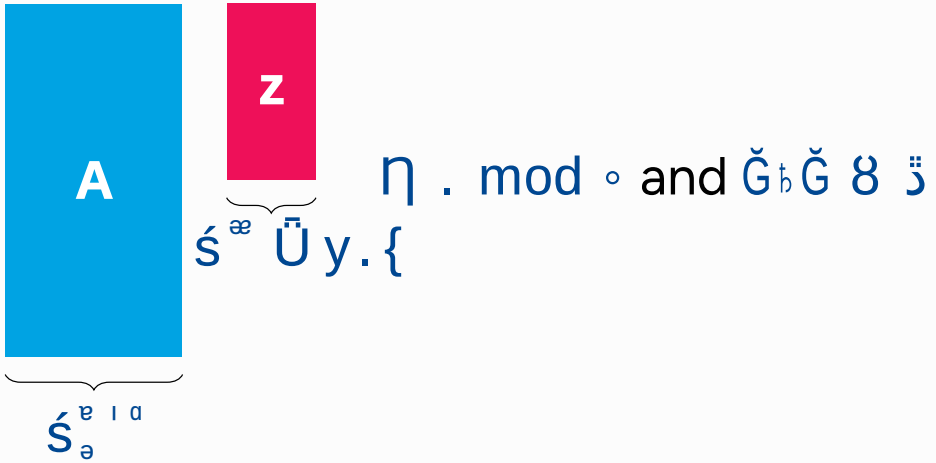


Given ╣ and ☼:

- Ģ Search: Find •
- Ģ Decision: Is ☼ random?

Hardness: Classically and quantumly NP-hard*

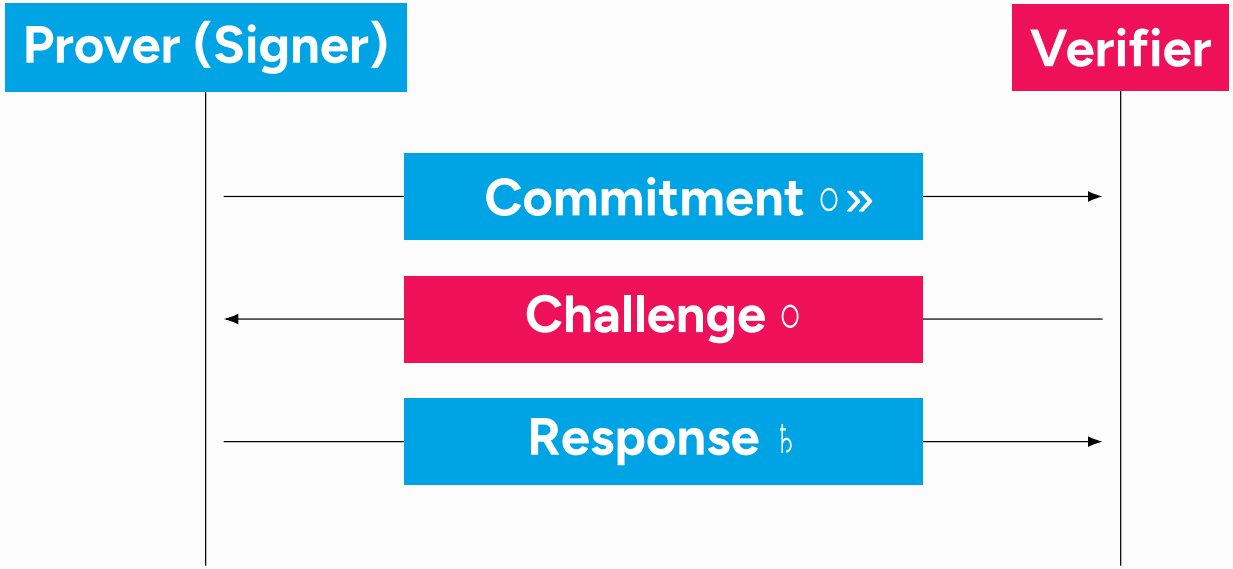
Short Integer Solution (SIS) [MR07]



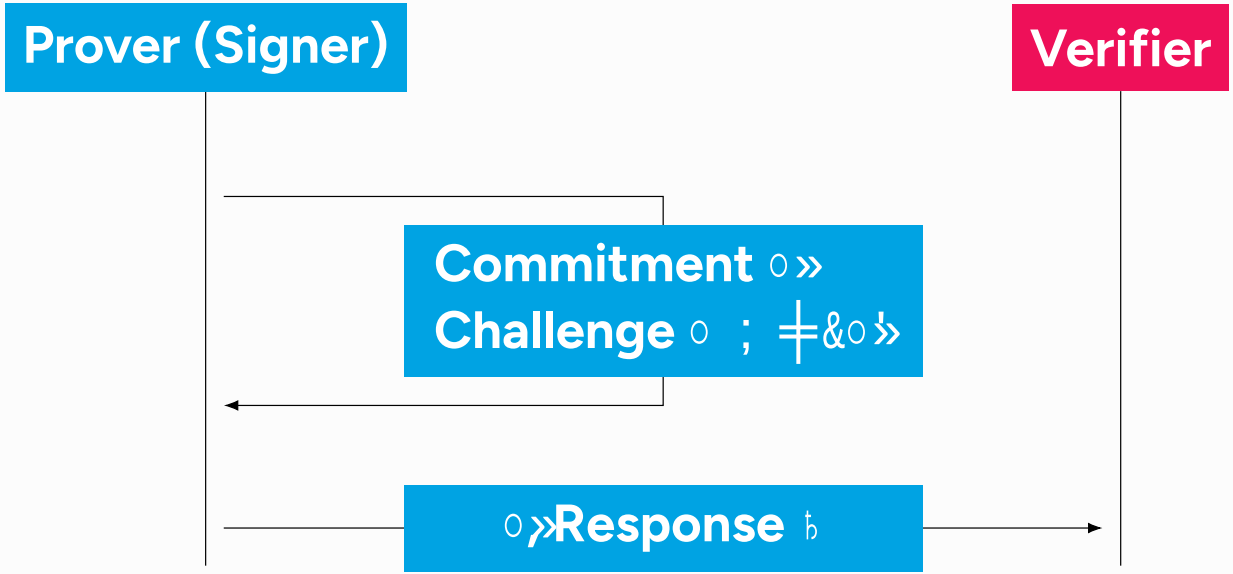
Given ╣ and ɜ̈, find ƀ

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 $\mathbf{s}_1 \in R_q^{\ell} \times R_q^{k}$, Public $\mathbf{A} \in R_q^{k \times \ell}$ and $\mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$.

- Signing: Masks $\mathbf{y} \in R_q^{\ell} \times R_q^{k}$ uniformly at random.
  Commitment $\mathbf{w}_1 = \mathrm{HighBits}(\mathbf{A}\mathbf{y})$, Challenge $c = H(M \| \mathbf{w}_1)$
  Responses $\mathbf{z} = \mathbf{y} + c\mathbf{s}_1$ and $\mathbf{r}_0 = \mathrm{LowBits}(\mathbf{A}\mathbf{y} - c\mathbf{s}_2)$.
  Reject $\mathbf{z}$ & $\mathbf{r}_0$ if outside safe range and try again.

- Verification: $\mathbf{w}_1' = \mathrm{UseHint}(\mathbf{h}, \mathbf{A}\mathbf{z} - c\mathbf{t})$. Accept if $\mathbf{w}_1' = \mathbf{w}_1$ and $\|\mathbf{z}\|_\infty < \gamma_1 - \beta$.

Lattice-based digital signatures [Lyu09; Lyu12]

- Key generation: Secret $\mathbf{s}_Q \in R_q^{\ell} \times R_q^{k}$, Public $\mathbf{A} \in R_q^{k \times \ell}$ and $\mathbf{t} := \mathbf{A}\mathbf{s}_Q + \mathbf{s}_q$.

- Signing: Masks $\mathbf{y}_Q \in R_q^{\ell} \times R_q^{k}$ uniformly at random.
  Commitment $\mathbf{w} := \mathbf{A}\mathbf{y}_Q + \mathbf{y}_q$, Challenge $\mathbf{c} \in \{-1, 0, 1\}^{n}$
  Responses $\mathbf{z}_Q := \mathbf{y}_Q + \mathbf{s}_Q$ and $\mathbf{z}_q := \mathbf{y}_q + \mathbf{s}_q$.
  Reject $\mathbf{z}_Q \times \mathbf{z}_q$ if outside safe range and try again.

- Verification: $\mathbf{w}' := \mathbf{A}\mathbf{z}_Q + \mathbf{z}_q - \mathbf{c}\mathbf{t}$. Accept if $\mathbf{w}' = \mathbf{w}$ and $\mathrm{HighBits}(\mathbf{w}') = \mathrm{HighBits}(\mathbf{w}) \le \gamma_2 - \beta$.

Lattice-based digital signatures [Lyu09; Lyu12]

- Key generation: Secret $\mathbf{s}_Q \in R_q^{\ell} \times R_q^{k}$, Public $\mathbf{A} \in R_q^{k \times \ell}$ and $\mathbf{t} = \mathbf{A}\mathbf{s}_Q + \mathbf{s}_q$.

- Signing: Masks $\mathbf{y}_Q \in R_q^{\ell} \times R_q^{k}$ uniformly at random.
  Commitment $\mathbf{w} = \mathbf{A}\mathbf{y}_Q + \mathbf{y}_q$, Challenge $\mathbf{c} = H(\mathbf{w}, \mu)$
  Responses $\mathbf{z}_Q = \mathbf{y}_Q + c\mathbf{s}_Q$ and $\mathbf{z}_q = \mathbf{y}_q + c\mathbf{s}_q$.
  Reject $\mathbf{z}_Q \times \mathbf{z}_q$ if outside safe range and try again.

Cannot get $\mathbf{s}_Q$ from $\mathbf{t}$ (LWE)

- Verification: $\mathbf{w}' = \mathbf{A}\mathbf{z}_Q - c\mathbf{t}_q$. Accept if $\mathbf{w}' = \mathbf{w}$ and $\|\mathbf{z}_Q\| \times \|\mathbf{z}_q\| < \beta$.

Lattice-based digital signatures [Lyu09; Lyu12]

- Key generation: Secret $s_1 \in R_q^{\ell} \times R_q^{k}$, Public $t = As_1 + s_2$ and $(t_1, t_0) = \mathrm{Power2Round}_q(t, d)$.
- Signing: Masks $y = (y_1, y_2) \in R_q^{\ell} \times R_q^{k}$ uniformly at random.
  Commitment $w = \mathrm{HighBits}_q(Ay, 2\gamma_2)$, Challenge $c = H(M \| w_1)$
  Responses $z = y + cs_1$ and $r_0 = \mathrm{LowBits}_q(w - cs_2, 2\gamma_2)$.
  Reject $z \notin \gamma_1 - \beta$ if outside safe range and try again.
- Verification: $w_1' = \mathrm{UseHint}_q(h, Az - ct, 2\gamma_2)$. Accept if $c = H(M \| w_1')$ and $\|z\|_\infty < \gamma_1 - \beta$.

Lattice-based digital signatures [Lyu09; Lyu12]

- Key generation: Secret $s_1 \in S_\eta^\ell * s_2 \in S_\eta^k$, Public $A \in R_q^{k \times \ell}$ and $t := As_1 + s_2$.

- Signing: Masks $y_1 \in S_{\gamma_1 - 1}^\ell * y_2 \in S_{\gamma_2}^k$ uniformly at random.
  Commitment $w_1 := \mathrm{HighBits}(Ay_1, 2\gamma_2)$, Challenge $c \in B_{60}$
  Responses $z_1 := y_1 + cs_1$ and $z_2 := y_2 - cs_2$.
  Reject $z_1 * z_2$ if outside safe range and try again.

- Verification: $w_1' := \mathrm{UseHint}(h, Az_1 - ct, 2\gamma_2)$. Accept if $\|z_1\|_\infty < \gamma_1 - \beta$ and $c = \tilde{c} = H(\mu \| w_1')$.

Lattice-based digital signatures [Lyu09; Lyu12]

- Key generation: Secret $s_1 \in R_q^{\ell} * s_2 \in R_q^{k}$, Public $t = A s_1 + s_2$ and $(t_1, t_0) = \mathrm{Power2Round}_q(t)$.

- Signing: Masks $y_1 \in R_q^{\ell} * y_2 \in R_q^{k}$ uniformly at random.
  Commitment $c = H(w_1)$, Challenge $c \in B_{60}$
  Responses $z_1 = y_1 + c s_1$ and $z_2 = y_2 + c s_2$.
  Reject $z_1 * z_2$ if outside safe range and try again.

- Verification: $w_1' = \mathrm{UseHint}_q(h, A z_1 - c t_1 \cdot 2^d)$. Accept if $\|z_1\|_\infty < \gamma_1 - \beta$ and $c = \tilde{H}(\mu \| w_1')$.

Lattice-based digital signatures [Lyu09; Lyu12]

- Key generation: Secret $\mathbf{s}_1 \in R_q^{\ell} \times R_q^{k}$, Public $\mathbf{A} \in R_q^{k \times \ell}$ and $\mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$.

- Signing: Masks $\mathbf{y} \in R_q^{\ell} \times R_q^{k}$ uniformly at random.
  Commitment $\mathbf{w} = \mathbf{A}\mathbf{y}$, Challenge $\mathbf{c} = H(\mathbf{w}, M)$
  Responses $\mathbf{z}_1 = \mathbf{y}_1 + \mathbf{c}\mathbf{s}_1$ and $\mathbf{z}_2 = \mathbf{y}_2 + \mathbf{c}\mathbf{s}_2$. ← Cannot get $\mathbf{s}_1 * \mathbf{s}_2$ from $\mathbf{z}_1 * \mathbf{z}_2$ (LWE)
  Reject $\mathbf{z}_1 * \mathbf{z}_2$ if outside safe range and try again.

- Verification: $\mathbf{w}' = \mathbf{A}\mathbf{z}_1 - \mathbf{c}\mathbf{t}_1 2^d$. Accept if $\mathbf{w}' = \mathbf{w}$ and $\|\mathbf{z}_1\|_\infty < \gamma_1 - \beta$ and $\|\mathbf{r}_0\|_\infty < \gamma_2 - \beta$.

Lattice-based digital signatures [Lyu09; Lyu12]

- Key generation: Secret $s_1 \in R_q^{\ell}$, $s_2 \in R_q^{k}$, Public $A \in R_q^{k \times \ell}$ and $t := As_1 + s_2$.

- Signing: Masks $y_1 \in R_q^{\ell}$ uniformly at random.
  Commitment $w_1 := \mathrm{HighBits}(Ay_1)$, Challenge $c \in B_{60}$
  Responses $z_1 := y_1 + cs_1$ and $z_2 := \mathrm{LowBits}(Ay_1 - cs_2)$.
  Reject z_1 if outside safe range and try again.

- Verification: $w_1' := \mathrm{UseHint}(h, Az_1 - ct)$. Accept if $\|z_1\|_\infty < \gamma_1 - \beta$ and $c = \tilde{H}(\mu \| w_1')$.

Lattice-based digital signatures [Lyu09; Lyu12]

- Key generation: Secret $\mathbf{s}_1 \in S_\eta^\ell, \mathbf{s}_2 \in S_\eta^k$, Public $\mathbf{A} \in R_q^{k \times \ell}$ and $\mathbf{t} := \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$.

- Signing: Masks $\mathbf{y}_1 \in S_\gamma^\ell, \mathbf{y}_2 \in S_\gamma^k$ uniformly at random.
  Commitment $\mathbf{w} := \mathbf{A}\mathbf{y}_1 + \mathbf{y}_2$, Challenge $c := H(\mathbf{w}, \mu)$
  Responses $\mathbf{z}_1 := c\mathbf{s}_1 + \mathbf{y}_1$ and $\mathbf{z}_2 := c\mathbf{s}_2 + \mathbf{y}_2$.
  Reject $\mathbf{z}_1, \mathbf{z}_2$ if outside safe range and try again.

- Verification: $c' := H(\mathbf{A}\mathbf{z}_1 + \mathbf{z}_2 - c\mathbf{t}, \mu)$. Accept if $c' = c$ and $\|\mathbf{z}_1\|, \|\mathbf{z}_2\| \leq \beta$.

$$\mathbf{A}\mathbf{z}_1 + \mathbf{z}_2 - c\mathbf{t} = \mathbf{A}(c\mathbf{s}_1 + \mathbf{y}_1) + (c\mathbf{s}_2 + \mathbf{y}_2) - c(\mathbf{A}\mathbf{s}_1 + \mathbf{s}_2) = \mathbf{A}\mathbf{y}_1 + \mathbf{y}_2 = \mathbf{w}$$

Lattice-based digital signatures [Lyu09; Lyu12]

- Key generation: Secret $\mathbf{s}_Q \in R_q^{\ell} * \mathbf{s}_q \in R_q^{\ell}$, Public $\mathbf{A} \in R_q^{k \times \ell}$ and $\mathbf{t} := \mathbf{A}\mathbf{s}_Q + \mathbf{s}_q$.

- Signing: Masks $\mathbf{y}_Q \in R_q^{\ell} * \mathbf{y}_q \in R_q^{\ell}$ uniformly at random.
  Commitment $\mathbf{w} := \mathbf{A}\mathbf{y}_Q + \mathbf{y}_q$, Challenge $\mathbf{c} := H(\mathbf{w}, M)$
  Responses $\mathbf{z}_Q := \mathbf{y}_Q + \mathbf{c}\mathbf{s}_Q$ and $\mathbf{z}_q := \mathbf{y}_q + \mathbf{c}\mathbf{s}_q$.
  Reject $\mathbf{z}_Q * \mathbf{z}_q$ if outside safe range and try again.

- Verification: $\mathbf{w}' := \mathbf{A}\mathbf{z}_Q + \mathbf{z}_q - \mathbf{c}\mathbf{t}$. Accept if $\mathbf{c} = H(\mathbf{w}', M)$ and $\|\mathbf{z}_Q * \mathbf{z}_q\| \le \beta$.

Cannot compute $\mathbf{z}_Q * \mathbf{z}_q$ without $\mathbf{s}_Q * \mathbf{s}_q$ (SIS)

ML-DSA signing (simplified)

Input: Private key ᶟᵉ, message ψ
Output: Signature ɉ

```
ȩģ₰ п ₰ ᵍᵋ₰ s_ß ₰ s_A ₰ t_ĩ ȩ  ᵘ  Ė÷1×Ĩ ąÓ×ȩᶟᵉ ȩ
A  ᵘ  5İďÀÿÓ  ȩģ ȩ
ʉ  ᵘ  Hȩᵍᵋ ế ψ ₰  Qqȩ
ᵋ b ȣ  ᵘ  ɟ л₰ Q ᶣÀảẪ
ģ ᴬ  ᵘ  Hȩп  ế ᵋ b ȣ  ế ʉ₰  Qqȩ
ɨ  ᵘ  л
ȩz₰ hȩ  ᵘ  ʽ
while ȩz₰ hȩ ᵖ ʽ  do
    y  ᵘ  5İďÀÿÓ Z ÀĖ÷ȩģ ᴬ₰ ɨ ȩ
    ᴼ»ᵘ  Ay
    ᴼ  ᵘ  Hȩᴼ» ế ʉ ȩ
    z  ᵘ  y ʒ ᴼᶟ_ß
    if ˡ z ˡ_ῶ  Ḧ ŧ_ß Ḍ ꞩ then ȩz₰ hȩ  ᵘ  ʽ
    ɨ  ᵘ  ɨ  ʒ  ᵅ
ɉ  ᵘ  Ėė á5ÿ Ĩ ąÓ×ȩᴼ» z mod ᵅ̇ ᵊ ȩ
return ɉ
```

ML-DSA signing (simplified)

Input: Private key ³ᵉ, message Ψ
Output: Signature ȷ

```
ɐɡ,ɘ п ɘ ɡᵋ ɘ s_β ɘ s_A ɘ t_ɟ ɛ ᵁ Ė÷1×ĺąÓ×ɐ³ᵉɛ
A ᵁ 5İďÀÿÓ ɐɡ,ɛ
ʉ ᵁ Hɐɡᵋ ế Ψ ɘ Qqɛ
ᵋ b ȣ ᵁ ɟ лɘ Qᵘ Ảả Ẩ
ɡ,ᴬ ᵁ Hɐп ế ᵋ b ȣ ế ʉɘ Qqɛ
ɨ ᵁ л
ɐzɘ hɛ ᵁ '
while ɐzɘ hɛ ᵖ ' do
    y ᵁ 5İďÀÿÓZÀĖ÷ɐɡ,ᴬɘ ɨ ɛ
    ᴼ»ᵁ Ay
    ᴼ ᵁ Hɐᴼ≫ế ʉɛ
    z ᵁ y ʒ ᴼᶾ_β
    if ˡzˡ_ῷ Ḧ ŧ_β Ḍ ᵴ then ɐzɘ hɛ ᵁ '
    ɨ ᵁ ɨ ʒ ᵅ
ȷ ᵁ Ėéá5ÿĺąÓ×ɐᴼ≫z mod^α ɘ ɛ
return ȷ
```

ML-DSA signing (simplified)

Input: Private key ᵌᵊ, message ᴪ
Output: Signature ɉ

```
ęɡ̦ə̣ п ə̣ ᵍᵋ ə̣ s_β ə̣ s_A ə̣ t_į ę ᵕ Ė÷1×Íą̧Ó×ęᵌᵊę
A ᵕ 5İďÀÿÓ ęɡ̦ę
ʉ ᵕ Hęᵍᵋ ế ᴪ ə̣ Qqę
ᵋ b ʊ ᵕ ɟ л ə̣ Q ᵠÀảẪ
ɡ̦ ᵕ Hęп ế ᵋ b ʊ ế ʉ ə̣ Qqę
ɨ ᵕ л
ęzə̣ hę ᵕ '
while ęzə̣ hę ᴰ ' do
    y ᵕ 5İďÀÿÓZÀĖ÷ęɡ̦ ə̣ ɨ ę
    ᴼ»ᵕ Ay
    ᴼ ᵕ HęᴼẌé ʉę
    z ᵕ y ʒ ᴼᶾ_β
    if ˡzˡ_ῷ Ḧ ŧ_β Ḍ ꞩ then ęzə̣ hę ᵕ '
    ɨ ᵕ ɨ ʒ ᵅ
ɉ ᵕ Ėéá5ÿÍą̧Ó×ęᴼ»z mod ᵅ ᵊę
return ɉ
```

ML-DSA signing (simplified)

Input: Private key $\mathbf{s}_{\beta}$, message Ψ
Output: Signature $\jmath$

$\varrho\mathbf{s}_{\beta}$
$\mathbf{A} \leftarrow \mathrm{ExpandA}(\varrho)$
$\mathbf{s}_{\beta} \leftarrow \mathrm{ExpandS}(\varrho')$
$\mathbf{t} \leftarrow \mathbf{A}\mathbf{s}_{\beta}$
$(\mathbf{t}_1, \mathbf{t}_0) \leftarrow \mathrm{Power2Round}(\mathbf{t})$
$\mathrm{pk} \leftarrow \mathrm{pkEncode}(\varrho, \mathbf{t}_1)$
$\mathrm{tr} \leftarrow \mathrm{H}(\mathrm{pk}, 64)$
$\mathrm{M}' \leftarrow \mathrm{tr} \| \Psi$
$\mathbf{z} \leftarrow \mathbf{y}$
$\mathbf{w}_1 \leftarrow \mathrm{HighBits}(\mathbf{w})$
$\mathrm{c}\tilde{} \leftarrow \mathrm{H}(\mathrm{M}' \| \mathrm{w}_1\mathrm{Encode}(\mathbf{w}_1), \lambda/4)$
$\mathrm{c} \leftarrow \mathrm{SampleInBall}(\mathrm{c}\tilde{})$
$\mathbf{z} \leftarrow \mathbf{y} + \mathrm{c}\mathbf{s}_1$
$(\mathbf{r}_0, \mathbf{h}) \leftarrow \mathrm{Decompose}(\mathbf{w} - \mathrm{c}\mathbf{s}_2)$
$\mathbf{r}_0 \leftarrow \mathbf{r}_0 + \mathrm{c}\mathbf{t}_0$
$\jmath \leftarrow \mathrm{sigEncode}(\mathrm{c}\tilde{}, \mathbf{z} \bmod^{\pm} q, \mathbf{h})$
return $\jmath$

ML-DSA signing (simplified)

Input: Private key ³ɐ, message ᴪ
Output: Signature ȷ

```
ɐg₈ꟹ п ꟹ gᵋ ꟹ s_ß ꟹ s_A ꟹ t_ɾ ɛ ᵁ Ė÷1×ĺ ąÓ×ɐ³ɐ ɛ
A ᵁ 5İďÀÿÓ ɐg₈ɛ
ʉ ᵁ Hɐgᵋ ế ᴪ ꟹ Qqɛ
ɛbȣ ᵁ ɟ ꟹ Q^{ɥÀảẤ}
g₈^A ᵁ Hɐп ế ɛbȣ ế ʉ ꟹ Qqɛ
ɫ ᵁ ȵ
ɐz ꟹ hɛ ᵁ ʿ
while ɐz ꟹ hɛ ᵖ ʿ do
    y ᵁ 5İďÀÿÓ Z ÀĖ÷ɐg₈^A ꟹ ɫ ɛ
    O»ᵁ Ay
    O ᵁ HɐO»ế ʉɛ
    z ᵁ y ɜ O ɜ_ß
    if ˡzˡ_ῷ Ḧ ŧ_ß Ḍ ꞩ then ɐz ꟹ hɛ ᵁ ʿ
    ɫ ᵁ ɫ ɜ ɑ
ȷ ᵁ Ėéá5ÿĺ ąÓ×ɐO ꟹ»z mod^ἀ ə ɛ
return ȷ
```

ML-DSA signing (simplified)

Input: Private key $\mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0$, message M
Output: Signature σ

$\hat{\mathbf{A}} \leftarrow \mathrm{ExpandA}(\rho)$
$\mu \leftarrow \mathrm{H}(\mathrm{BytesToBits}(\mathrm{tr}) \| M, 64)$
$\kappa \leftarrow 0$, $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
while $(\mathbf{z}, \mathbf{h}) = \perp$ **do**
  $\mathbf{y} \leftarrow \mathrm{ExpandMask}(\rho', \kappa)$
  $\mathbf{w} \leftarrow \mathrm{NTT}^{-1}(\hat{\mathbf{A}} \circ \mathrm{NTT}(\mathbf{y}))$
  $\mathbf{w}_1 \leftarrow \mathrm{HighBits}(\mathbf{w})$
  $\tilde{c} \leftarrow \mathrm{H}(\mu \| \mathrm{BitsToBytes}(\mathbf{w}_1), \lambda/4)$
  $c \leftarrow \mathrm{SampleInBall}(\tilde{c})$
  $\mathbf{z} \leftarrow \mathbf{y} + c\mathbf{s}_1$
  if $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ or $\|\mathrm{LowBits}(\mathbf{w} - c\mathbf{s}_2)\|_\infty \geq \gamma_2 - \beta$ **then** $(\mathbf{z}, \mathbf{h}) \leftarrow \perp$
  $\kappa \leftarrow \kappa + \ell$
$\sigma \leftarrow \mathrm{SigEncode}(\tilde{c}, \mathbf{z} \bmod^{\pm} q, \mathbf{h})$
return σ

ML-DSA signing (simplified)

Input: Private key ³ᵉ, message Ψ
Output: Signature ǰ

```
ꞓꞡ꞉ п꞉ᵍᵋ꞉ s꞉ s꞉ t ꞓ ᶸ Ė÷1×Í ąÓ×ꞓ³ᵉꞓ
A ᶸ 5İďÀÿÓ ꞓꞡꞓ
ʉ ᶸ Hꞓᵍᵋ ế Ψ꞉ Qqꞓ
ᵋ b ʊ ᶸ ɟЛ꞉Q
ꞡ ᶸ Hꞓп ế ᵋ b ʊ ế ʉ꞉ Qqꞓ
ɨ ᶸ Л
ꞓz꞉ hꞓ ᶸ ʻ
while ꞓz꞉ hꞓ ᵖ ʻ do
    y ᶸ 5İďÀÿÓZÀĖ÷ꞓꞡ꞉ ɨ ꞓ
    ᴼ»ᶸ Ay
    ᴼ ᶸ Hꞓᴼ» ế ʉꞓ
    z ᶸ y ȝ ᴼȝ
    if ˡzˡ Ḧ ŧ Ḍ ꞩ then ꞓz꞉ hꞓ ᶸ ʻ
    ɨ ᶸ ɨ ȝ ᵅ
ǰ ᶸ Ėéá5ÿÍąÓ×ꞓᴼ»z mod ᵅ ᵊꞓ
return ǰ
```

ML-DSA signing (simplified)

Input: Private key ³ᵉ, message ᴪ
Output: Signature ȷ

```
ꞔꞡ₉ ⲡ₉ ᵍᵋ₉ s_β₉ s_A₉ t_ɾ ꞓ ᵘ Ė÷1×ĺ ąÓ×ꞔ³ᵉꞓ
A ᵘ 5İďÀÿÓ ꞔꞡꞓ
ʉ ᵘ Hꞔᵍᵋ ế ᴪ₉ Qqꞓ
ᵋbᵿ ᵘ ɟл₉Qᶣ̣Ảå̂
ꞡᴬ ᵘ Hꞔп ế ᵋbᵿ ế ʉ₉ Qqꞓ
ɨ ᵘ л
ꞔz₉ hꞓ ᵘ ʿ
while ꞔz₉ hꞓ ᵖ ʿ do
    y ᵘ 5İďÀÿÓZÀĖ÷ꞔꞡᴬ₉ ɨ ꞓ
    ᴼ»ᵘ Ay
    ᴼ ᵘ Hꞔᴼ»ế ʉꞓ
    z ᵘ y ȝ ᴼᶾ_β
    if ˡzˡ_ῶ Ḧ ŧ_β Ḍ ꞩ then ꞔz₉ hꞓ ᵘ ʿ
    ɨ ᵘ ɨ ȝ ᵅ
ȷ ᵘ Ėȩá5ÿĺ ąÓ×ꞔᴼ₉»z mod^ᵅ̇ ᵊꞓ
return ȷ
```

Hedged-mode ML-DSA

Input: Private key $\mathbf{s}_{\beta}, \mathbf{s}_{A}, \mathbf{t}_{i}$, message Ψ
Output: Signature $\mathbf{y}$

1: $\mathbf{A} \leftarrow \mathrm{ExpandA}(\rho)$
2: $\mu \leftarrow \mathrm{H}(\mathrm{BytesToBits}(\mathrm{tr}) \| M', 64)$
3: $\rho'' \leftarrow \mathrm{H}(\mathrm{BytesToBits}(K) \| \mathrm{BytesToBits}(\mathrm{rnd}) \| \mu, 64)$
4: $\hat{\mathbf{s}}_{\beta} \leftarrow \mathrm{NTT}(\mathbf{s}_{\beta})$, $\hat{\mathbf{s}}_{A} \leftarrow \mathrm{NTT}(\mathbf{s}_{A})$, $\hat{\mathbf{t}}_{i} \leftarrow \mathrm{NTT}(\mathbf{t}_{i})$
5: $\kappa \leftarrow 0$, $(\mathbf{z}, \mathbf{h}) \leftarrow \mathrm{NULL}$
6: **while** $(\mathbf{z}, \mathbf{h}) = \mathrm{NULL}$ **do**
7: $\mathbf{y} \leftarrow \mathrm{ExpandMask}(\rho'', \kappa)$

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

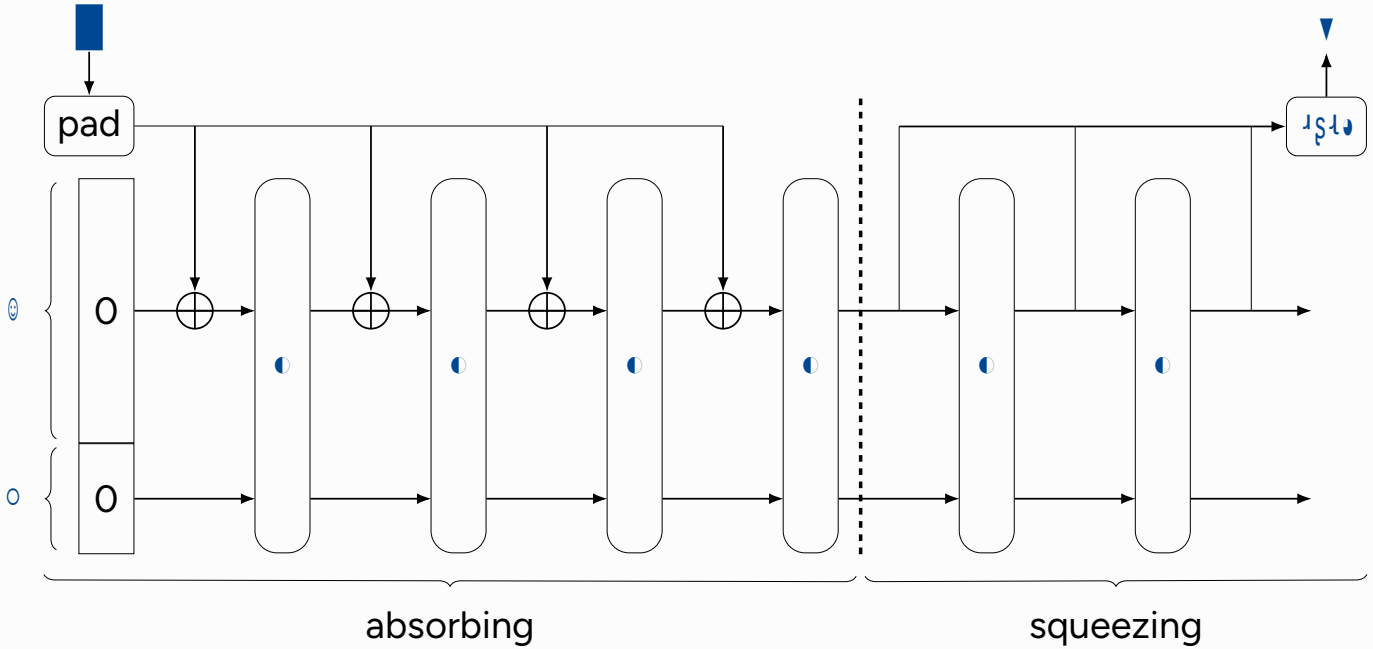


Figure adapted from [Ber+11]

Sponge constructions [Ber+11]

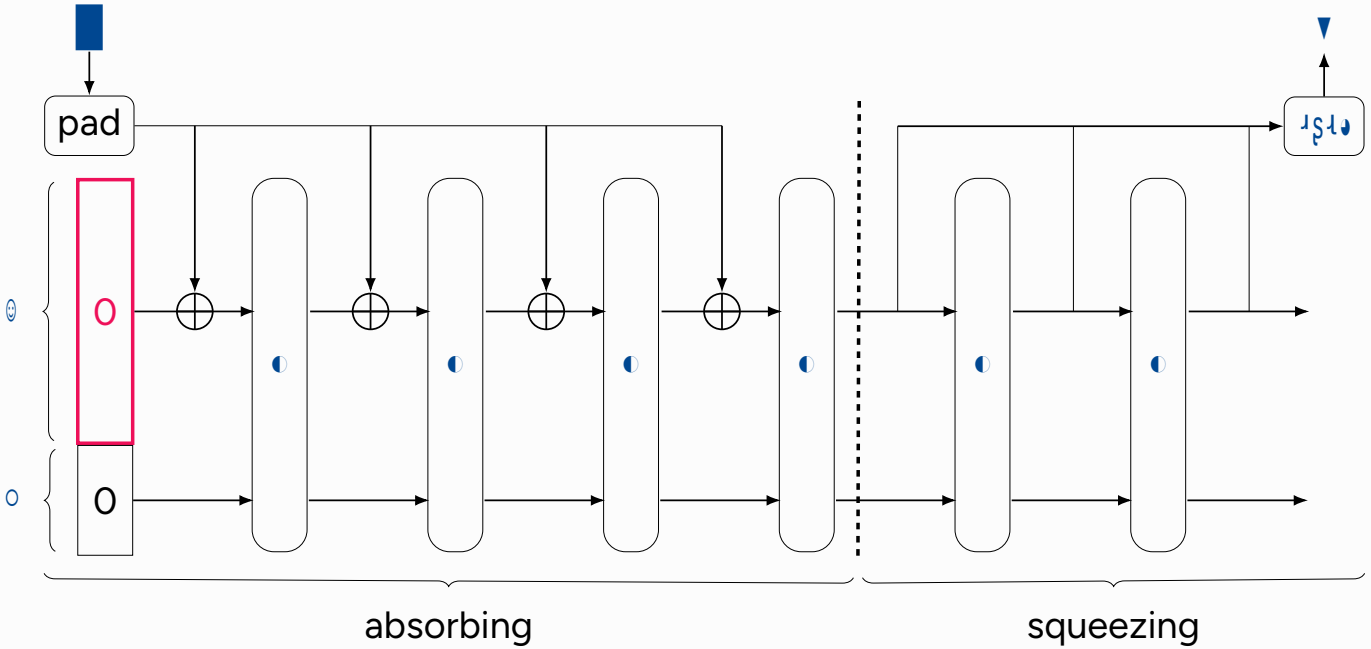


Figure adapted from [Ber+11]

Sponge constructions [Ber+11]

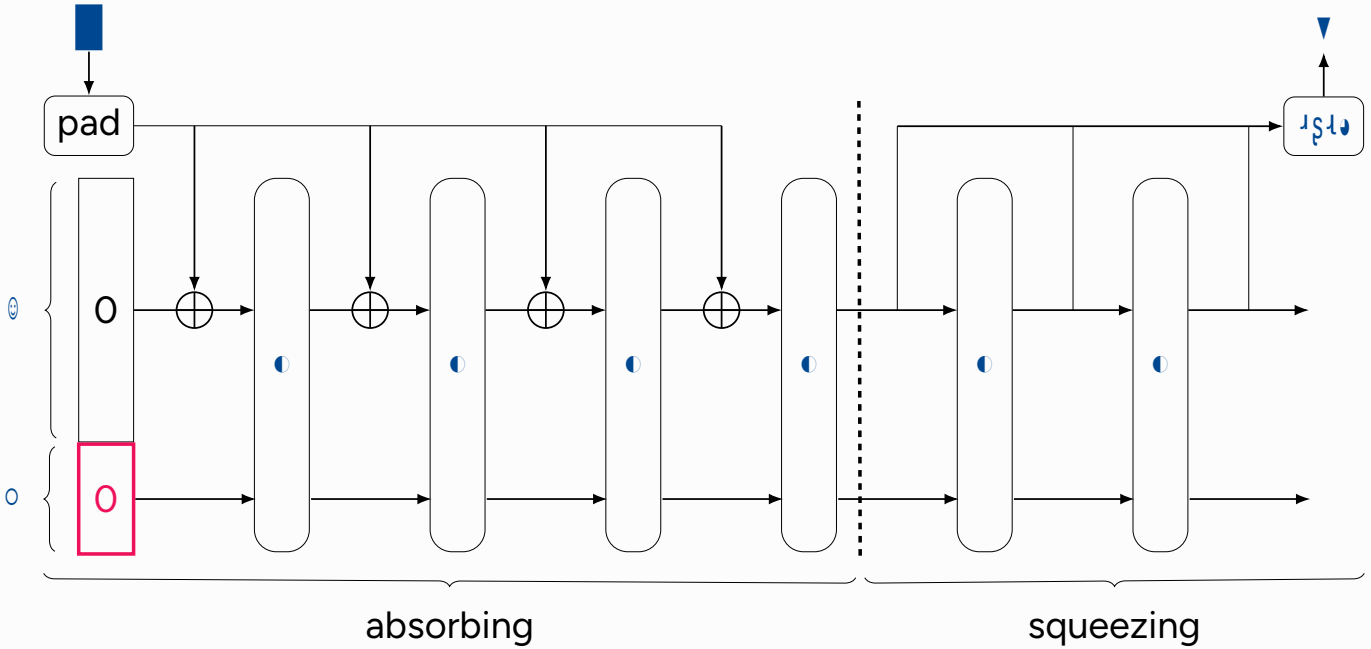


Figure adapted from [Ber+11]

Sponge constructions [Ber+11]

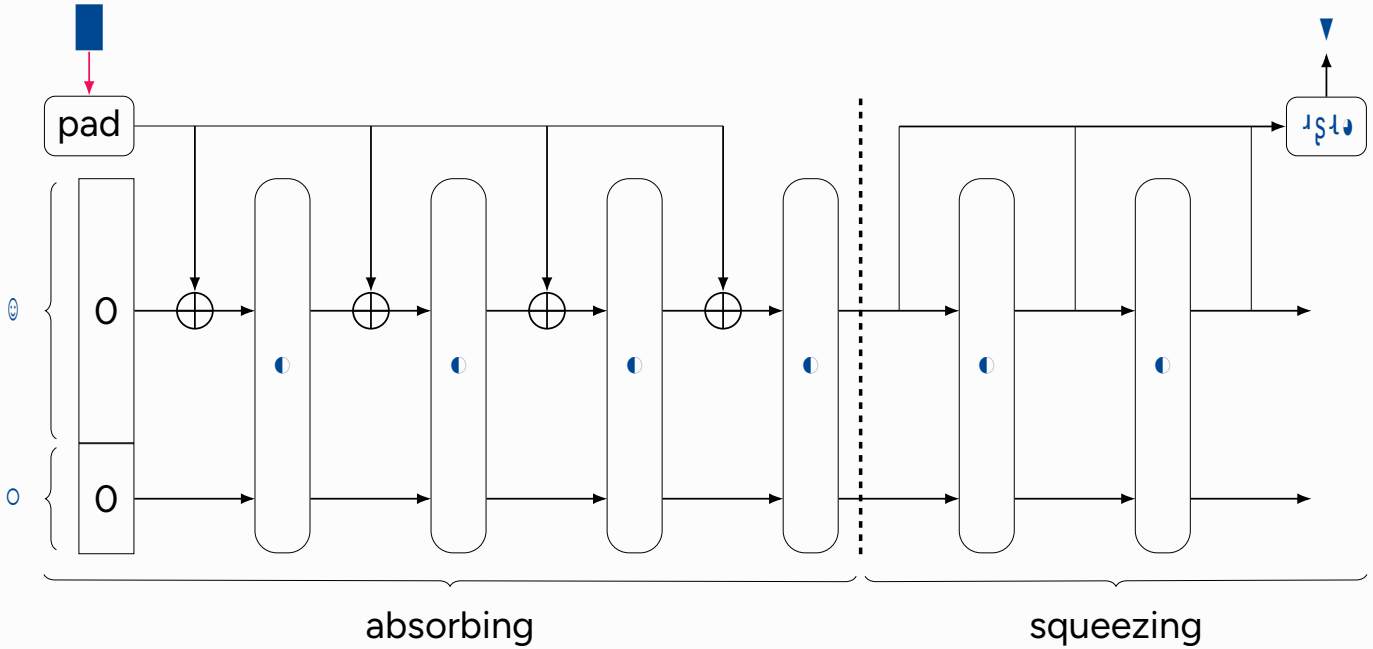


Figure adapted from [Ber+11]

Sponge constructions [Ber+11]

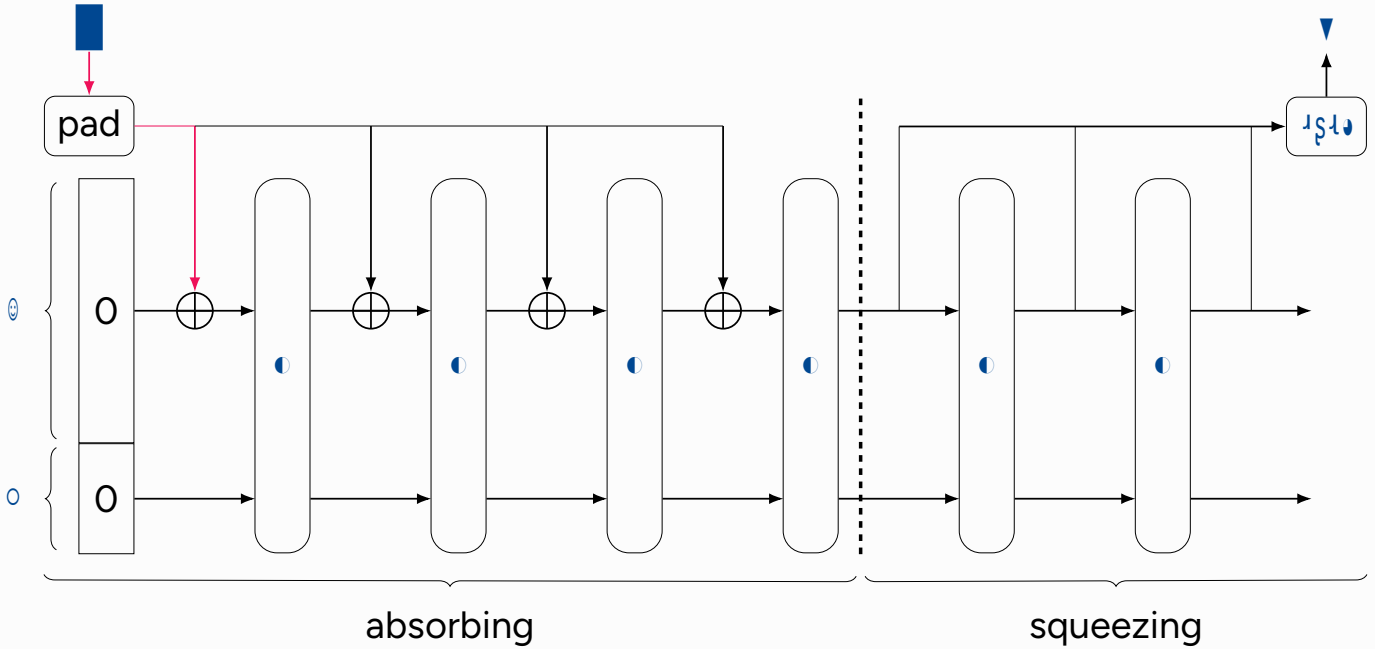


Figure adapted from [Ber+11]

Sponge constructions [Ber+11]

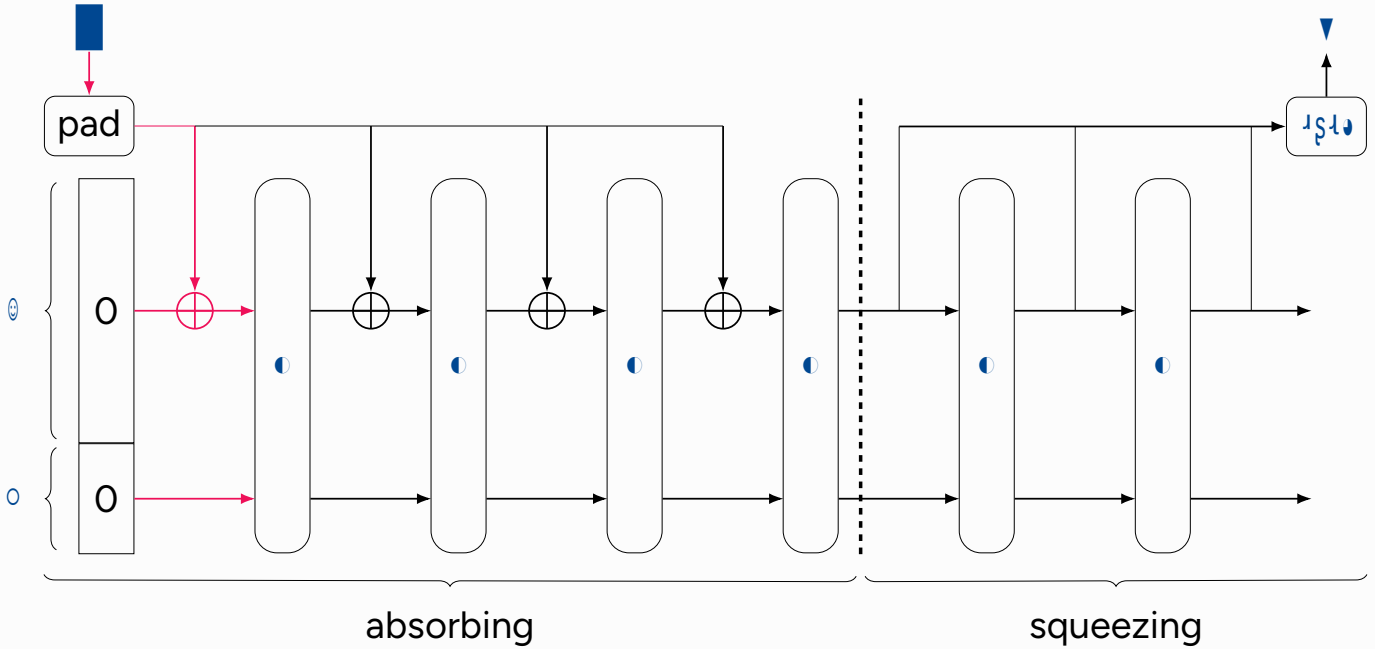


Figure adapted from [Ber+11]

Sponge constructions [Ber+11]

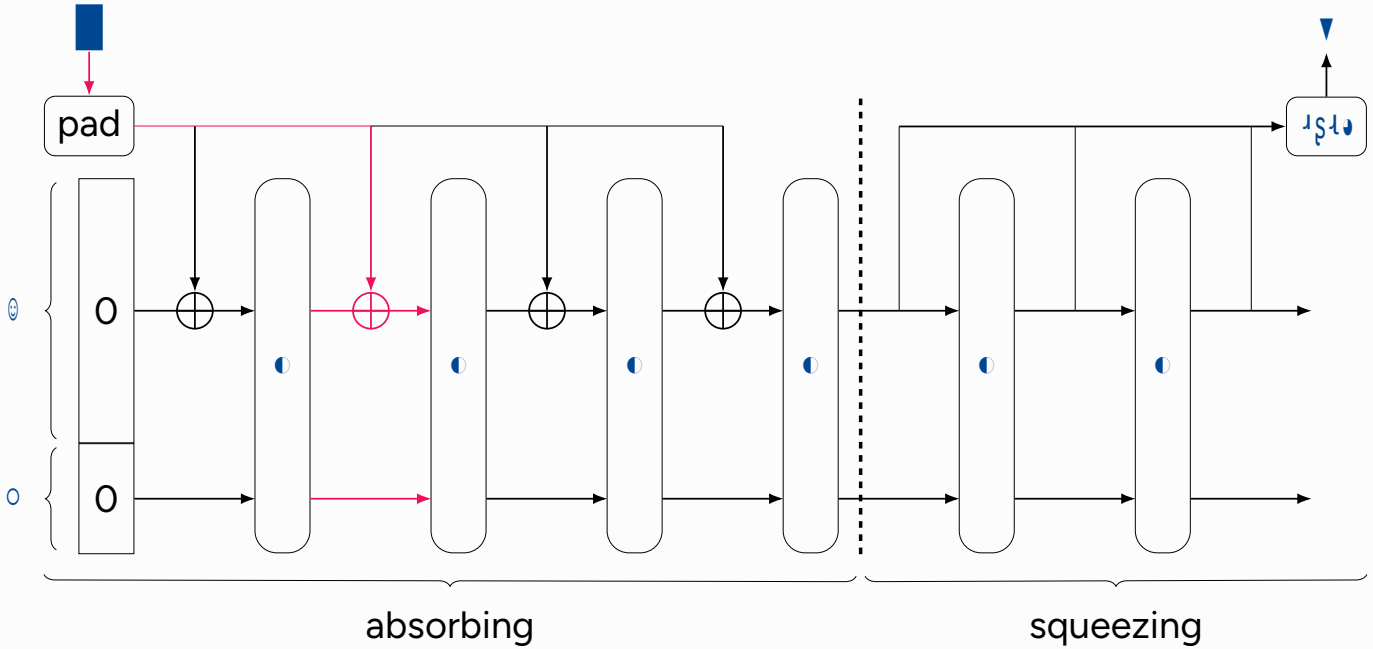


Figure adapted from [Ber+11]

Sponge constructions [Ber+11]

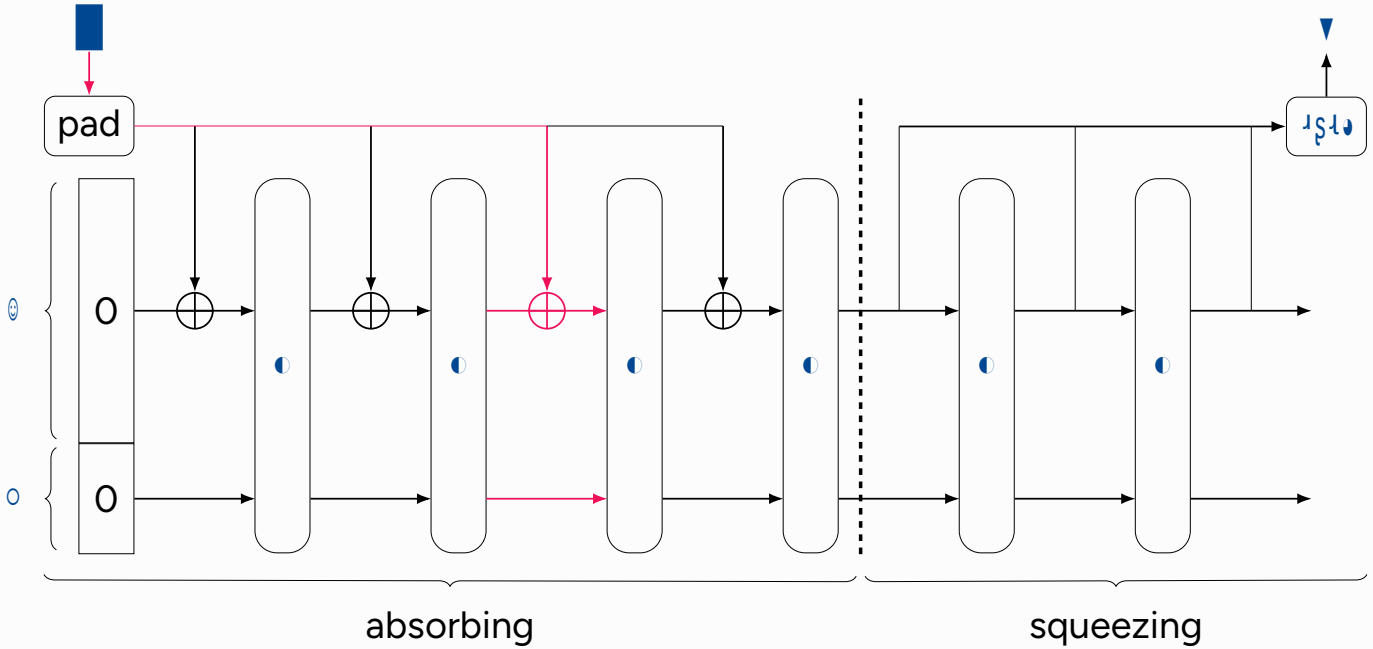


Figure adapted from [Ber+11]

Sponge constructions [Ber+11]

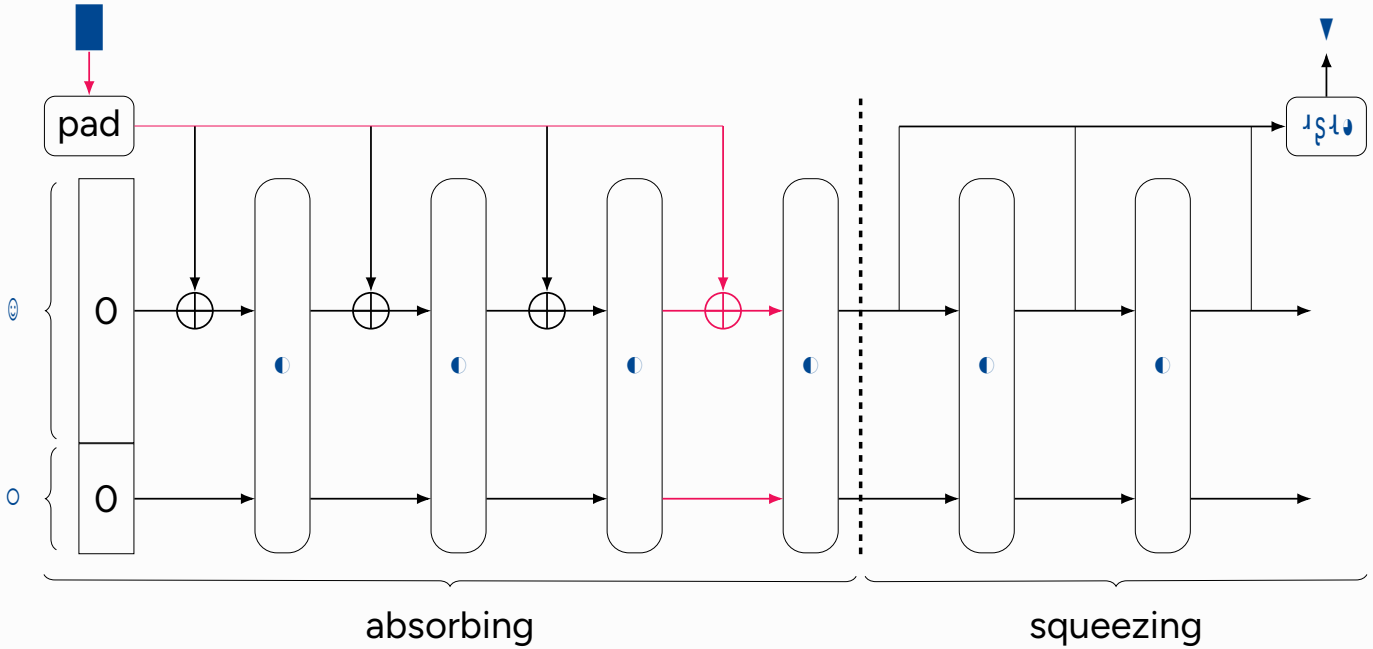


Figure adapted from [Ber+11]

Sponge constructions [Ber+11]

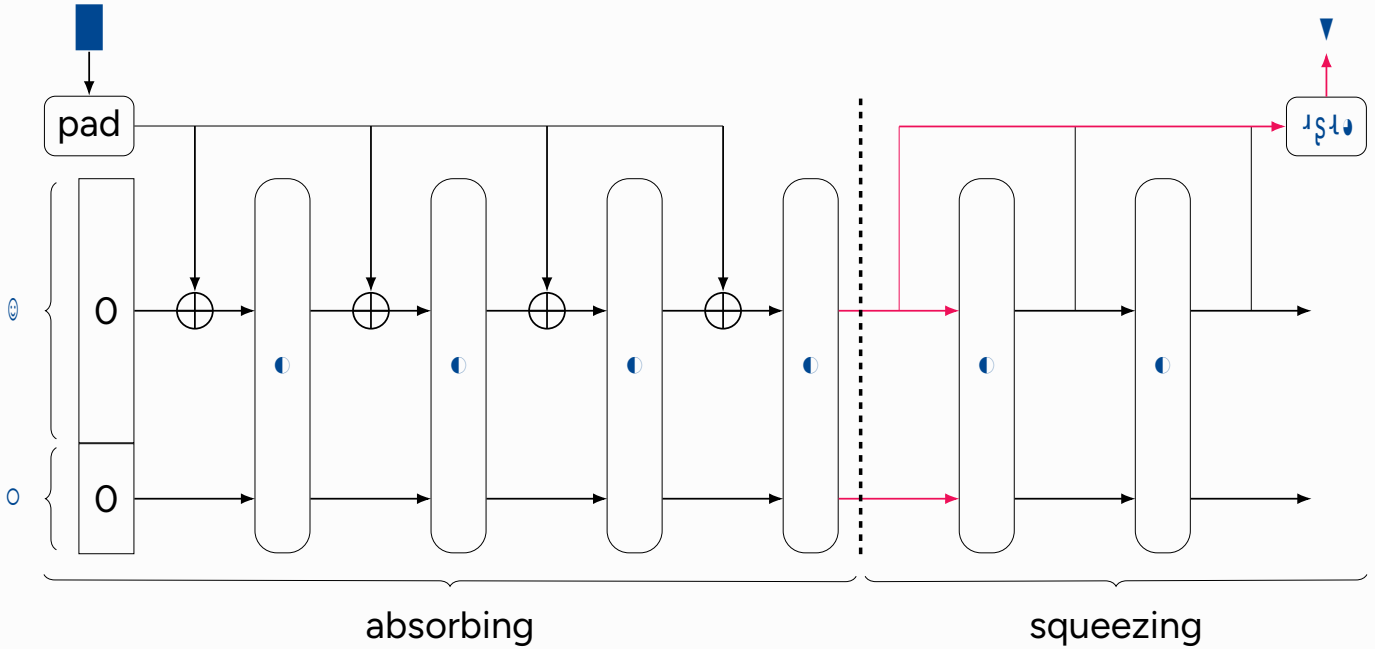


Figure adapted from [Ber+11]

Sponge constructions [Ber+11]

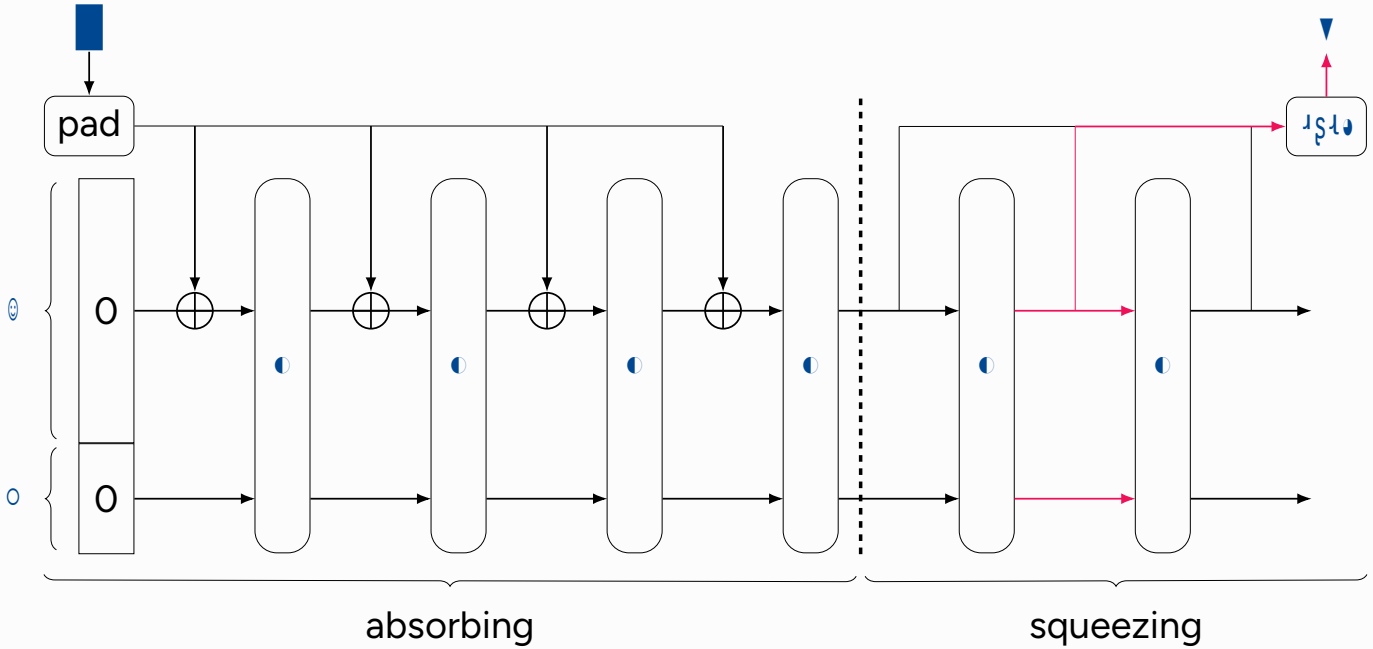


Figure adapted from [Ber+11]

Sponge constructions [Ber+11]

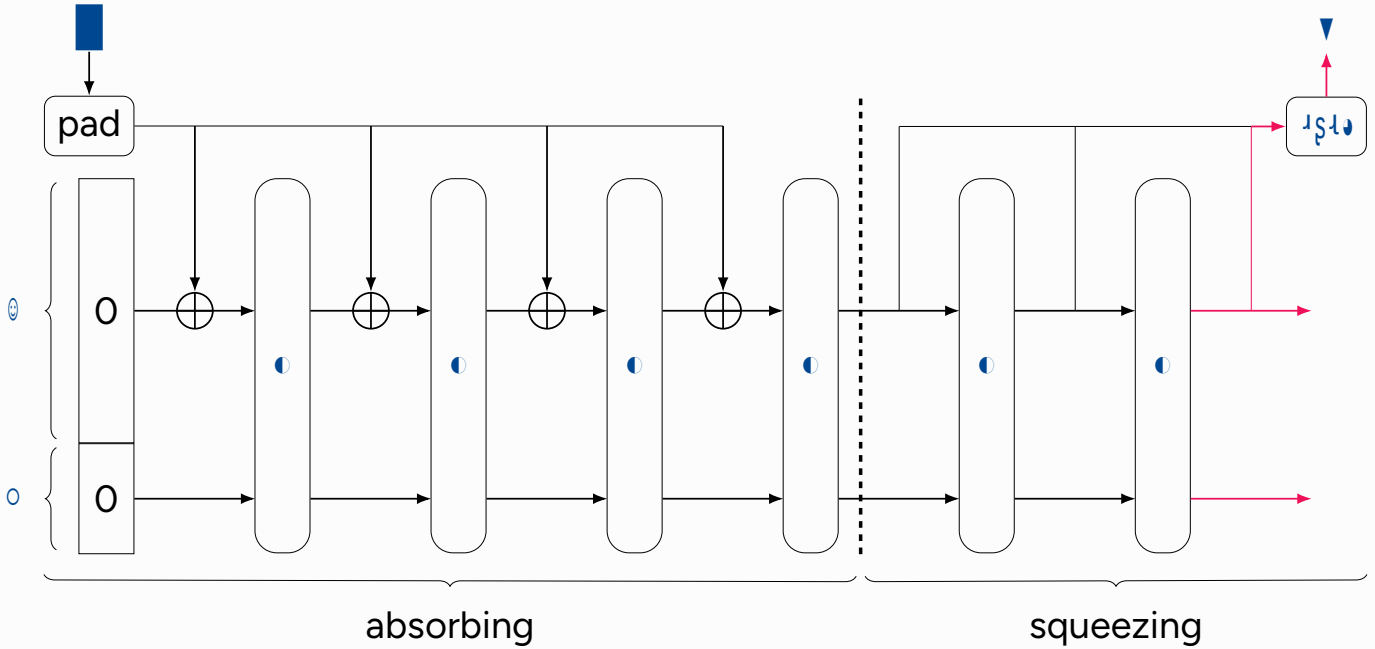


Figure adapted from [Ber+11]

Attack: Skipping absorption

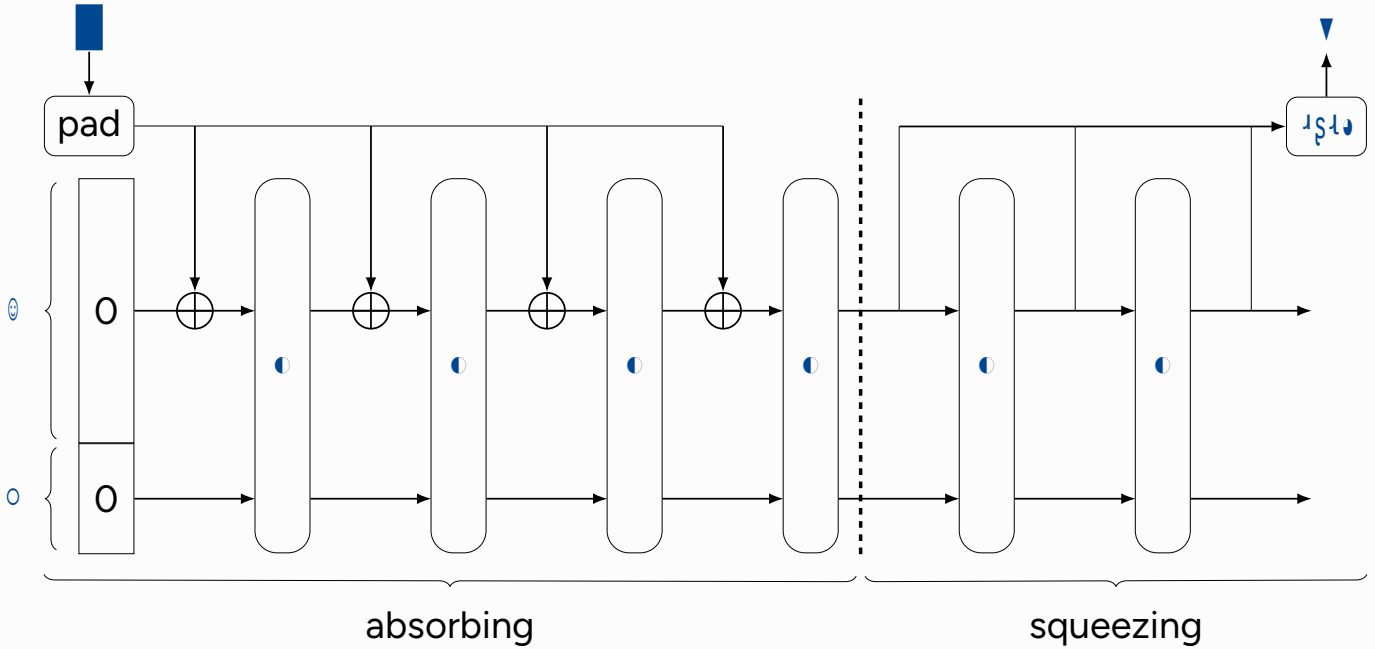


Figure adapted from [Ber+11]

Attack: Skipping absorption

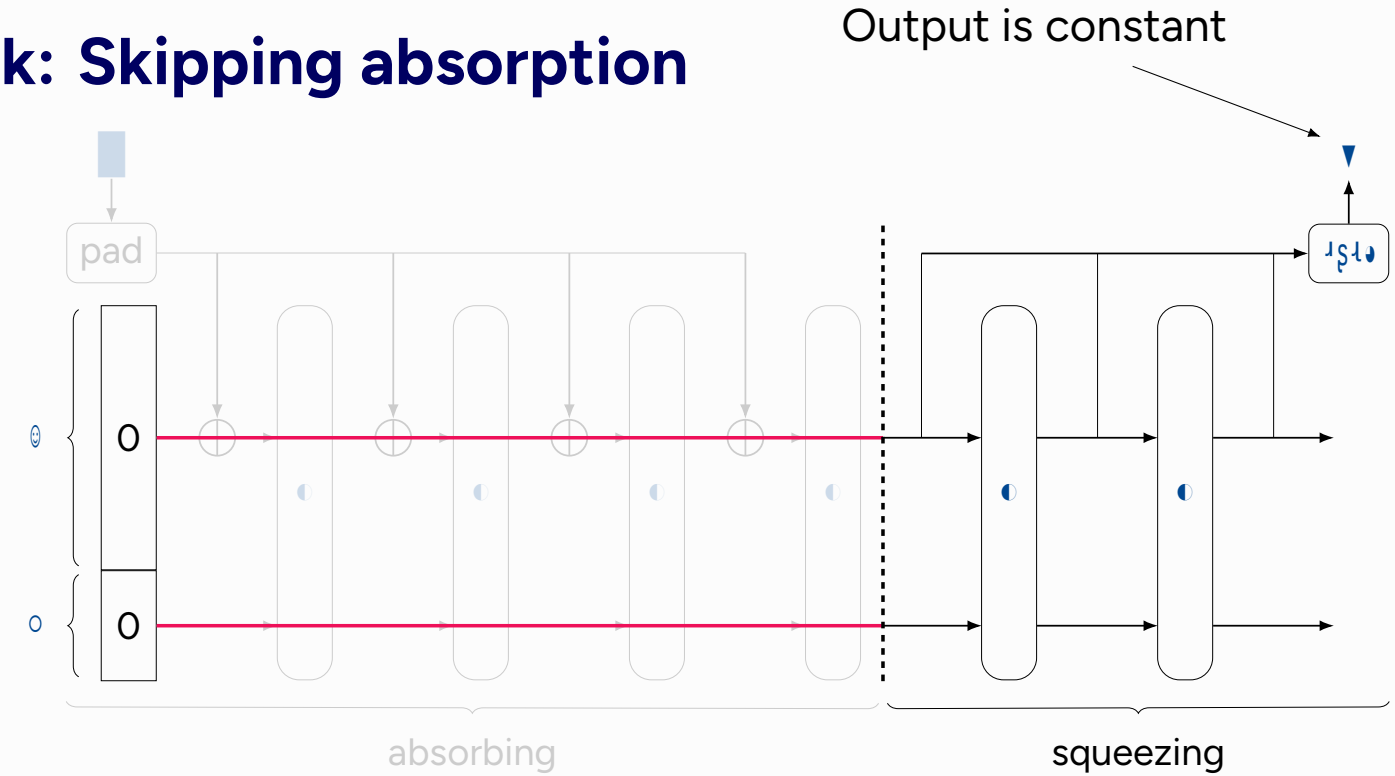


Figure adapted from [Ber+11]

Key recovery from known ẓ̄

Input: Private key ³ᵉ, message ᴪ
Output: Signature ɉ

```
ɐg̦ɚ ᴨɚ ᵍᵋɚ **s**ᵦɚ **s**ᴬɚ **t**ᵢ ɛ ᵁ Ė÷1×ĺąÓ×ɐ³ᵉɛ
**A** ᵁ 5İďÀÿÓ ɐg̦ɛ
ʉ ᵁ Hɐᵍᵋ ế ᴪɚ Qqɛ
ᵋbᴕ ᵁ ɟɲɚQᶣȦảẪ
g̦ᴬ ᵁ Hɐᴨ ế ᵋbᴕ ế ʉɚ Qqɛ
ɨ ᵁ ɲ
ɐzɚ hɛ ᵁ ʿ
while ɐzɚ hɛ ᴰ ʿ do
    y ᵁ 5İďÀÿÓZÀĖ÷ɐg̦ᴬɚ ɨɛ
    ᴼ»ᵁ Ay
    ᴼ ᵁ Hɐᴼ»ế ʉɛ
    z ᵁ y ȝ ᴼᶾᵦ
    if ˡzˡ ῶ Ḧ ŧᵦ Ḍ ꞩ then ɐzɚ hɛ ᵁ ʿ
    ɨ ᵁ ɨ ȝ ᵅ
ɉ ᵁ Ėếá5ÿĺąÓ×ɐᴼ»z mod ᶷ ᵊɛ
return ɉ
```

Key recovery from known ẟ̄

Input: Private key ᵌᵄ, message ψ
Output: Signature ȷ

ꬲg̡ꬸ п ꬸ ᵍᵋ ꬸ **s**ᵦ ꬸ **s**ᴀ ꬸ **t**ᵢ ꞓ ᵁ Ė÷1×Íą̓Ó×ꬲᵌᵄꞓ
A ᵁ 5İďÀÿÓ ꬲg̡ꞓ
ʉ ᵁ Hꬲᵍᵋ ế ψ ꬸ Qqꞓ
ᵋbᵿ ᵁ ɟɲꬸQᶣᴬảẨ
g̡ᴬ ᵁ Hꬲп ế ᵋbᵿ ế ʉꬸ Qqꞓ
ɨ ᵁ ɲ
ꬲ**z**ꬸ **h**ꞓ ᵁ ʿ
while ꬲ**z**ꬸ **h**ꞓ ᵖ ʿ **do**
y ᵁ 5İďÀÿÓZÀĖ÷ꬲg̡ᴬꬸ ɨ ꞓ
ᴼ»ᵁ **Ay**
ᴼ ᵁ Hꬲᴼ» ế ʉꞓ
z ᵁ **y** ȝ ᴼᶾᵦ
if ˡ**z**ˡ ῳ̣ Ḧ ŧᵦ Ḍ ꞩ **then** ꬲ**z**ꬸ **h**ꞓ ᵁ ʿ
ɨ ᵁ ɨ ȝ ᵅ
ȷ ᵁ Ėꬸá5ÿÍą̓Ó×ꬲᴼ»**z** mod ᵅ̀ ᵊꞓ
return ȷ

Key recovery from known ظ ̄

Input: Private key ᶾᵋ, message ᴪ
Output: Signature ᶅ

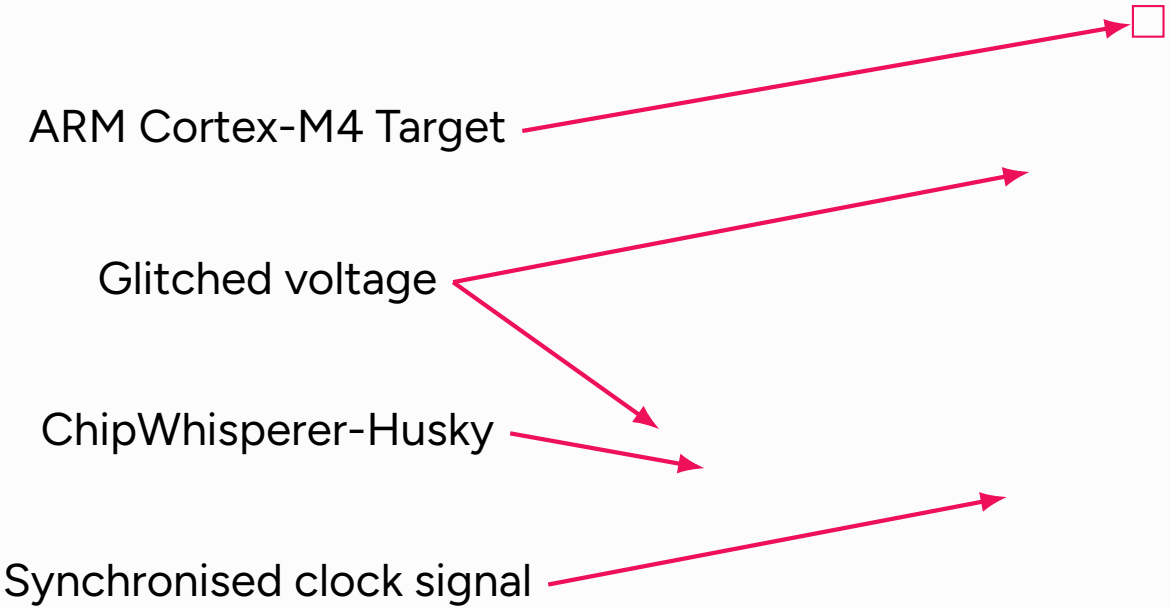
```
ȩg̦ꝧ п ꝧ ᵍᵋ ꝧ s_ꞵ ꝧ s_Ạ ꝧ t_ᵢ ꜫ ᵁ Ė÷1×Í ąÓ×ȩᶾᵋ ꜫ
A ᵁ 5İďÀÿÓ ȩg̦ꜫ
ᵾ ᵁ Hȩᵍᵋ ế ᴪ ꝧ̫ Qqꜫ
ᵋ ᵇ ᴕ ᵁ ꟸ Ꞃꝧ Qᶣ Ạảẫ
g̦ᴬ̉ ᵁ Hȩп ế ᵋ ᵇ ᴕ ế ᵾꝧ̫ Qqꜫ
ɨ ᵁ Ꞃ
ȩzꝧ hꜫ ᵁ ʿ
while ȩzꝧ hꜫ ᵖ ʿ do
    y ᵁ 5İďÀÿÓZÀĖ÷ȩg̦ᴬ̉ꝧ ɨ ꜫ
    ᴼ»ᵁ Ay
    ᴼ ᵁ Hȩᴼ»ế ᵾꜫ
    z ᵁ y ȝ ᴼᶾ_ꞵ
    if ˡzˡ_ῷ Ḧ ŧ_ꞵ Ḍ ꞩ then ȩzꝧ hꜫ ᵁ ʿ
    ɨ ᵁ ɨ ȝ ᵅ
ᶅ ᵁ Ėɵá5ÿÍąÓ×ȩᴼꝧ»z mod ᵅ̉ ᵊꜫ
return ᶅ
```

Fault injection method

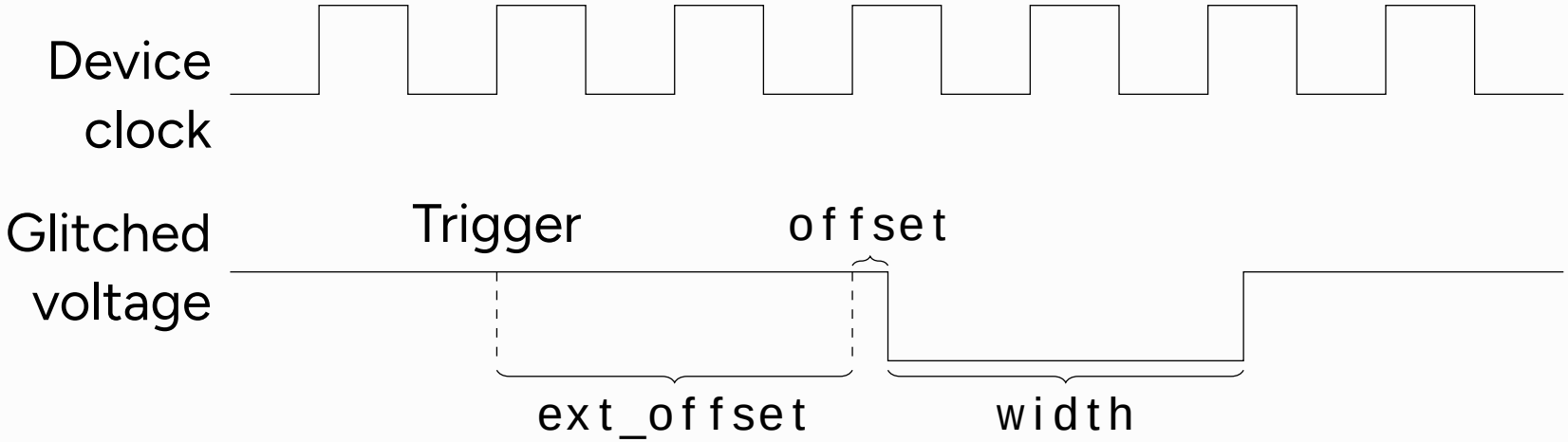
ChipWhisperer-Husky:

- 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



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 ﻟ inside the rejection sampling loop
- Include ◌ or ■ when sampling **y**

Input: Private key ³ᵉ, message Ψ
Output: Signature ⌡

```
ęg ꟸ ⲡ ꟸ ᵍᵋ ꟸ s_ß ꟸ s_A ꟸ t_ſ ε ᑌ Ė÷1×Íą Ó×ę³ᵉε
A ᑌ 5İďÀÿÓ ęgε
ᵾ ᑌ Hęᵍᵋ é Ψ ꟸ Qqε
ᵋbɤ ᑌ ɟЛꟸQ
gᴬ ᑌ Hęⲡ é ᵋbɤ é ᵾ ꟸ Qqε
ɨ ᑌ Л
ęz ꟸ hε ᑌ '
while ęz ꟸ hε ᵖ ' do
    y ᑌ 5İďÀÿÓZÀĖ÷ęgᴬ ꟸ ɨε
    ᴼ»ᑌ Ay
    ᴼ ᑌ Hęᴼ é ᵾε
    z ᑌ y ʒ ᴼ³_ß
    if ˡzˡ Ḧ ŧ_ß Ḍ ꞩ then ęz ꟸ hε ᑌ '
    ɨ ᑌ ɨ ʒ ᵅ
⌡ ᑌ Ėéá5ÿÍąÓ×ęᴼz mod ᵅ ᵊε
return ⌡
```

Results and countermeasures

Succeeds in a single trace with probability 52.8%

Countermeasures:

- Generic fault detection / CFI
- Eliminate branches
- Sample ꞓ inside the rejection sampling loop
- Include ◌ or ■ when sampling **y**

Input: Private key ³ᵉ, message Ψ
Output: Signature ȷ

```
ęg,ą п ą ᵍᵋ ą s_ß ą s_A ą t_ĭ ɛ ᵁ Ė÷1×Í ąÓ×ę³ᵉ ɛ
A ᵁ 5İďÀÿÓ ęg,ɛ
ʉ ᵁ Hęᵍᵋ ế Ψ ą Qqɛ
ɛ b ȣ ᵁ ɟ Ꞃ ą Qᵠᴬᵃᴬ
g,ᴬ ᵁ Hęп ế ɛ b ȣ ế ʉą Qqɛ
ɟ ᵁ Ꞃ
ęzą hɛ ᵁ ʿ
while ęzą hɛ ᵖ ʿ do
    y ᵁ 5İďÀÿÓZÀĖ÷ęg,ᴬ ą ɟ ɛ
    O»ᵁ Ay
    O ᵁ HęO ế ʉɛ
    z ᵁ y ʒ O₃ß
    if ˡ z ˡ ῷ Ḧ ŧß Ḍ ꞩ then ęzą hɛ ᵁ ʿ
    ɟ ᵁ ɟ ʒ ᵅ
ȷ ᵁ Ėéá5ÿÍ ąÓ×ęO z mod ᵅ ᵊ ɛ
return ȷ
```

Results and countermeasures

Succeeds in a single trace with probability 52.8%

Countermeasures:

- Generic fault detection / CFI
- Eliminate branches
- Sample ߬ inside the rejection sampling loop
- Include ◌ ◗ ◌ or ■ when sampling **y**

Input: Private key ³ᵉ, message Ψ
Output: Signature ȷ

```
ęg,ə п ə ᵍᵉ ə s_β ə s_A ə t_ſ ε ᵁ Ė÷1×Íą Ó×ę³ᵉε
A ᵁ 5İďÀÿÓ ęg,ε
ᵾ ᵁ Hęᵍᵉ é Ψ ə Qqε
ᵋbȣ ᵁ ɟ Ʌ ə Qᴴ
g,ᴬ ᵁ Hęп é ᵋbȣ é ᵾ ə Qqε
ɟ ᵁ Ʌ
ęz ə hε ᵁ '
while ęz ə hε ᴰ ' do
    y ᵁ 5İďÀÿÓZÀĖ÷ęg,ᴬ ə ɟ ε
    O»ᵁ Ay
    O ᵁ HęO×é ᵾε
    z ᵁ y ȝ Oȝ_β
    if ˡzˡ_ῶ Ḧ ŧ_β Ḍ ꞩ then ęz ə hε ᵁ '
    ɟ ᵁ ɟ ȝ ᵅ
ȷ ᵁ Ėé á5ÿÍąÓ×ęO ᵻz modᵅ ᵊε
return ȷ
```

Results and countermeasures

Succeeds in a single trace with probability 52.8%

Countermeasures:

- Generic fault detection / CFI
- Eliminate branches
- Sample inside the rejection sampling loop
- Include or when sampling **y**

Input: Private key , message
Output: Signature

```
1:
2:
3:
4:
5:
6:  while do
7:      
8:      
9:      
10:     
11:     
12:     
13:     if then 
14:     
15:     
16: return 
```

Results and countermeasures

Succeeds in a single trace with probability 52.8%

Countermeasures:

- Generic fault detection / CFI
- Eliminate branches
- Sample ᷉ inside the rejection sampling loop
- Include ◌ ◗ ◌ or ■ when sampling **y**

Input: Private key ᶟᵉ, message ψ
Output: Signature ȷ

```
ȩg̦ ꞑ ᵍᵋ s_β s_A t_ḯ ȩ ᵁ Ė÷1×Í ąÓ×ȩᶟᵉ ȩ
A ᵁ 5İďÀÿÓ ȩg̦ȩ
ʉ ᵁ Hȩᵍᵋ é ψ Qqȩ
ᵋbŏ ᵁ ɟ ԓ Qᶣᴬᵃᴬ
g̦ᴬ ᵁ Hȩꞑ é ᵋbŏ é ʉ Qqȩ
ɨ ᵁ ԓ
ȩz hȩ ᵁ ʽ
while ȩz hȩ ᵖ ʽ do
    y ᵁ 5İďÀÿÓZÀĖ÷ȩg̦ᴬ ɨ ȩ
    ᴼ»ᵁ Ay
    ᴼ ᵁ Hȩᴼ×é ʉȩ
    z ᵁ y ȝ ᴼᶾ_β
    if ˡ z ˡ_ῶ Ḧ ŧ_β Ḍ ꞩ then ȩz hȩ ᵁ ʽ
    ɨ ᵁ ɨ ȝ ᵅ
ȷ ᵁ Ėéá5ÿÍąÓ×ȩᴼ»z modᵅ ᵊȩ
return ȷ
```

Results and countermeasures

Succeeds in a single trace with probability 52.8%

Countermeasures:

- Generic fault detection / CFI
- Eliminate branches
- Sample ḃ inside the rejection sampling loop
- Include ◌ or ■ when sampling **y**

Input: Private key ɜɐ, message ψ
Output: Signature ȷ

```
ɐ ɡ ɒ п ɒ ɡ ɛ ɒ s ɒ s ɒ t ɛ ʋ Ė÷1×ĺąÓ×ɐ ɜɐ ɛ
A ʋ 5İďÀÿÓ ɐ ɡ ɛ
ʉ ʋ Hɐ ɡɛ é ψ ɒ Qq ɛ
ɛ b ʊ ʋ ɟ л ɒ Q ɥ À ả Ȃ
ɡ ʋ Hɐ п é ɛ b ʊ é ʉ ɒ Qq ɛ
ɨ ʋ л
ɐ z ɒ h ɛ ʋ '
while ɐ z ɒ h ɛ ɒ ' do
    y ʋ 5İďÀÿÓZÀĖ÷ɐ ɡ ɒ ɨ ɒ ɛ b ʊ ɒ п ɛ
    O»ʋ Ay
    O ʋ Hɐ O ×é ʉ ɛ
    z ʋ y ɜ Oɜ
    if ǀ z ǀ ῷ Ḧ ŧ Ḍ ꞩ then ɐ z ɒ h ɛ ʋ '
    ɨ ʋ ɨ ɜ ɑ
ȷ ʋ Ėéá5ÿĺąÓ×ɐ O ɒ z mod ɑ ɘ ɛ
return ȷ
```

Conclusions & Future work

- Point-of-failure $\hat{z}^{\prime\prime}$ 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