

Fault Injection Attacks on NIST Post-Quantum Standards: ML-DSA and ML-KEM

Andersson Calle Viera

JAIF, Montpellier, 01 october 2026

Current State

Quantum threat: Quantum Computers could be a reality from 10 to 30 years, or never...

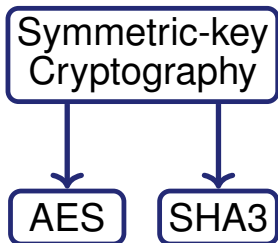
Current State

Quantum threat: Quantum Computers could be a reality from 10 to 30 years, or never...

Symmetric-key
Cryptography

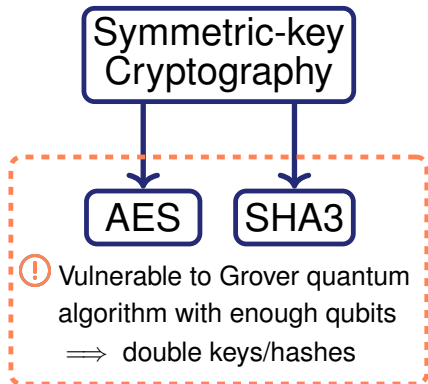
Current State

Quantum threat: Quantum Computers could be a reality from 10 to 30 years, or never...



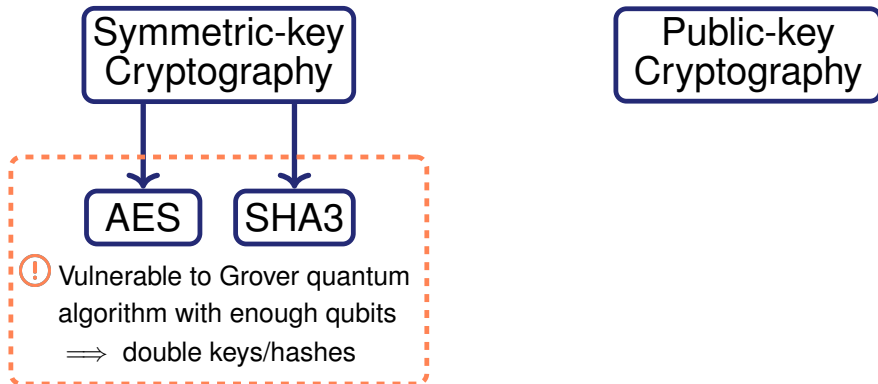
Current State

Quantum threat: Quantum Computers could be a reality from 10 to 30 years, or never...



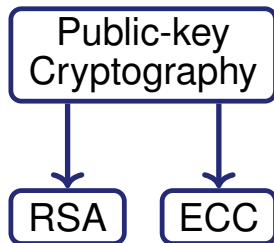
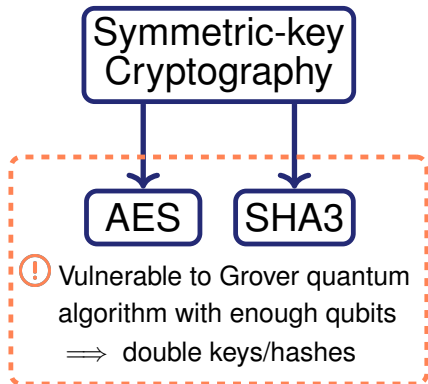
Current State

Quantum threat: Quantum Computers could be a reality from 10 to 30 years, or never...



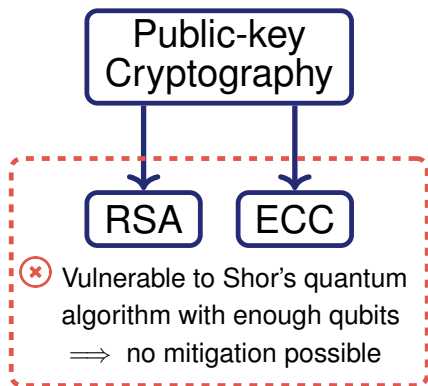
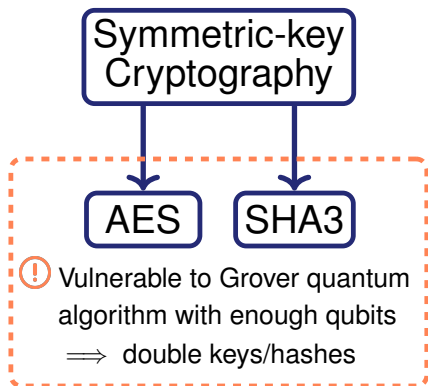
Current State

Quantum threat: Quantum Computers could be a reality from 10 to 30 years, or never...



Current State

Quantum threat: Quantum Computers could be a reality from 10 to 30 years, or never...

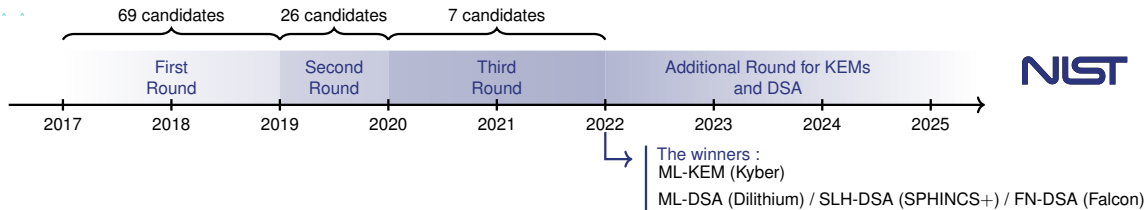


Current State

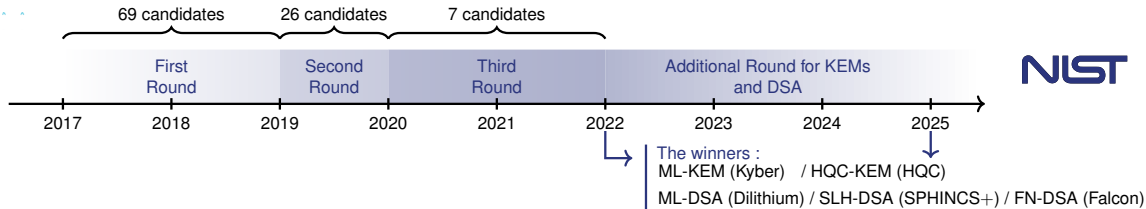


OPEN

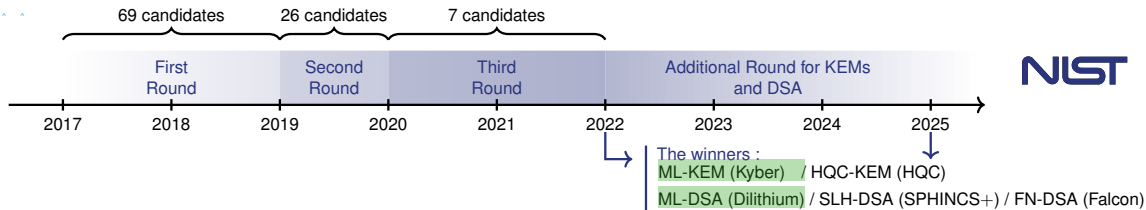
So ... what do we do?



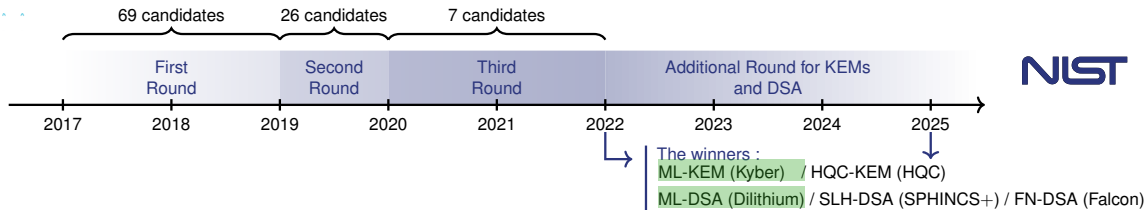
So ... what do we do?



So ... what do we do?

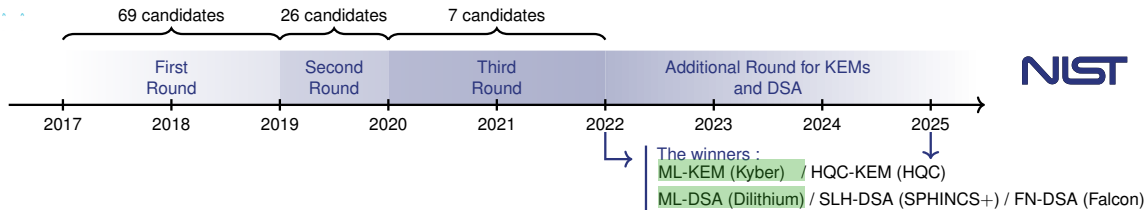


So ... what do we do?



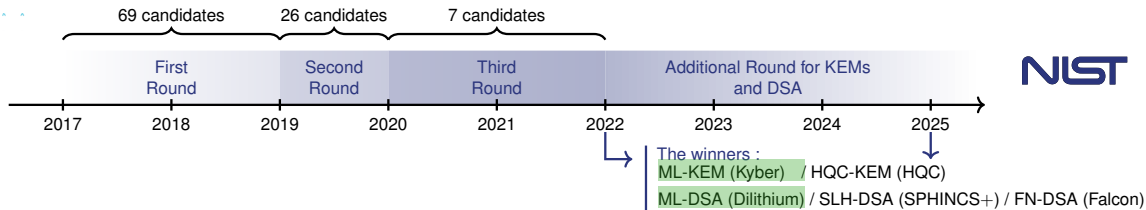
- But why choose to only present **ML-DSA** and **ML-KEM**?

So ... what do we do?



- But why choose to only present **ML-DSA** and **ML-KEM**?
 - For relevance to embedded systems:

So ... what do we do?



- But why choose to only present **ML-DSA** and **ML-KEM**?

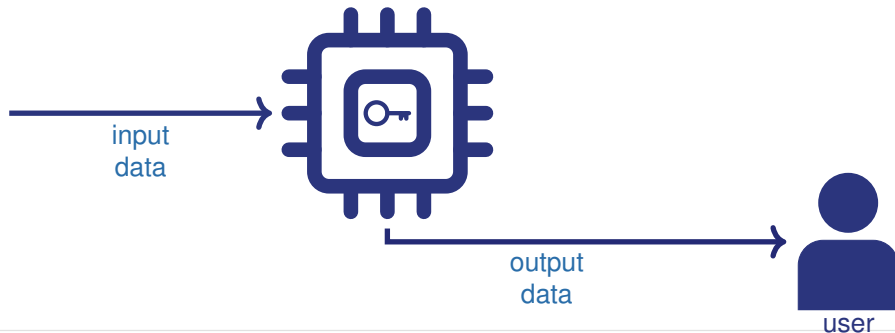
➤ For relevance to embedded systems:

Algorithms	ML-DSA-87	SLH-DSA	FN-DSA
pk (<i>bytes</i>)	2 592	64	1 793
sig (<i>bytes</i>)	4 627	49 856	1 280
secure implem	medium	hard	hard
stack usage	113 200	2 206 200	82 532



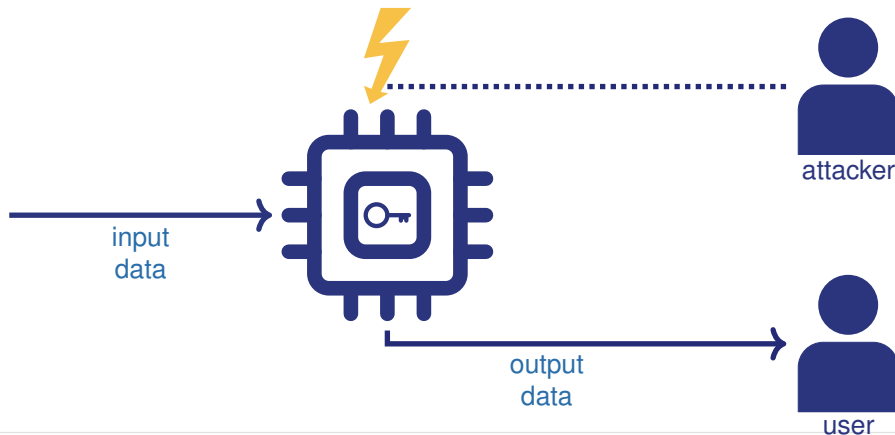
Fault Attacks (FA)

- Instead of observing...



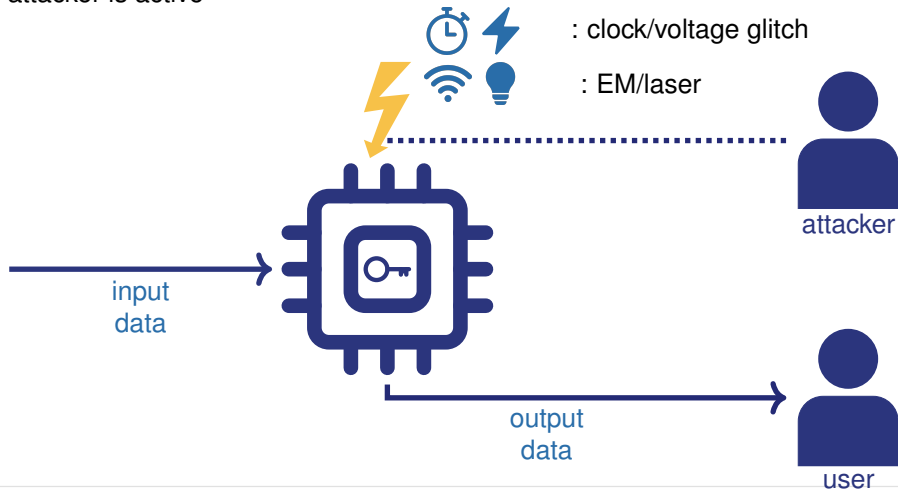
Fault Attacks (FA)

- Instead of observing...
...the attacker is active



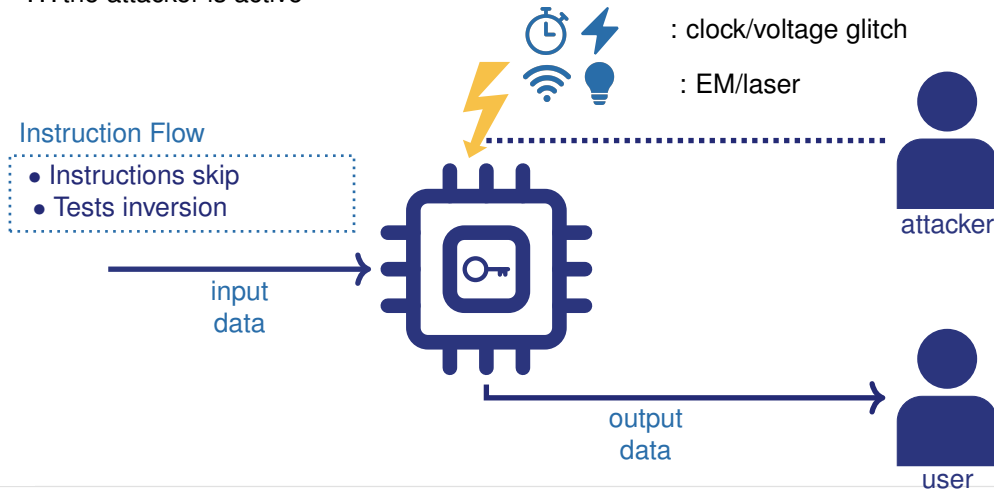
Fault Attacks (FA)

- Instead of observing...
...the attacker is active



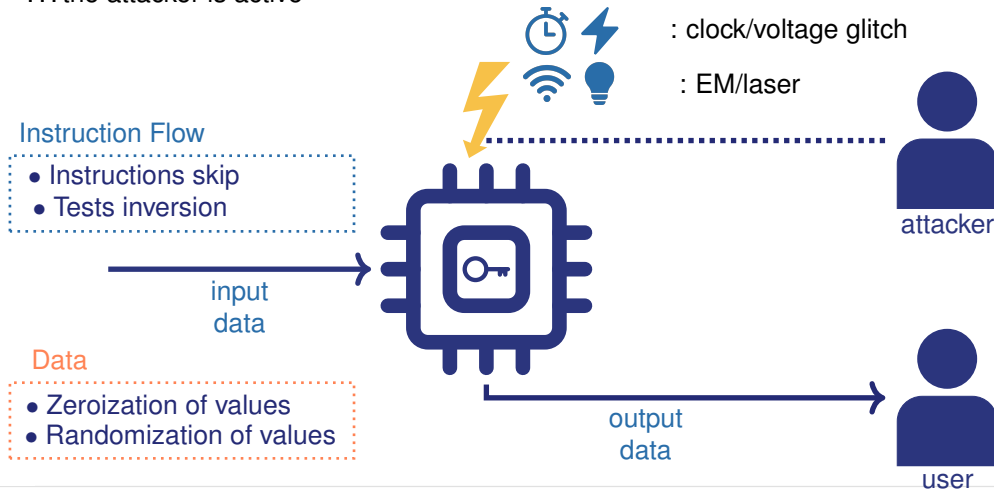
Fault Attacks (FA)

- Instead of observing...
- ...the attacker is active



Fault Attacks (FA)

- Instead of observing...
- ...the attacker is active



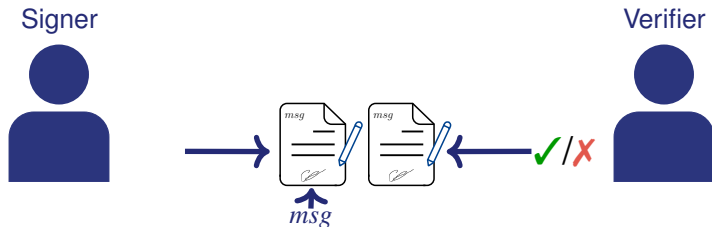


ML-DSA aka Dilithium

Digital Signatures

- **Reproduce the characteristics of a handwritten signature:**

- Link a document to its author
- Guarantee the integrity of the document
- Make the signature forgery-proof



Digital Signatures

- **Reproduce the characteristics of a handwritten signature:**

- Link a document to its author
- Guarantee the integrity of the document
- Make the signature forgery-proof



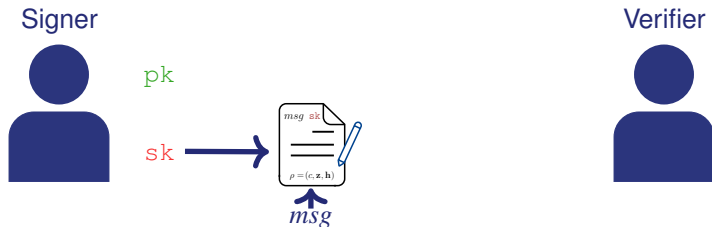
- **Goals:**

- Only the signer with their **private key** sk can generate a signature for a *msg*
- Anyone with the **public key** pk can verify the authenticity of the signature of a *msg*

Digital Signatures

- **Reproduce the characteristics of a handwritten signature:**

- Link a document to its author
- Guarantee the integrity of the document
- Make the signature forgery-proof



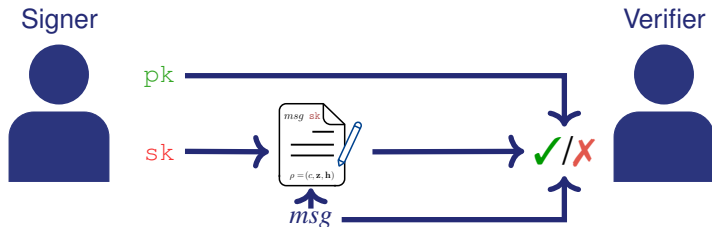
- **Goals:**

- Only the signer with their **private key** sk can generate a signature for a msg
- Anyone with the **public key** pk can verify the authenticity of the signature of a msg

Digital Signatures

- **Reproduce the characteristics of a handwritten signature:**

- Link a document to its author
- Guarantee the integrity of the document
- Make the signature forgery-proof



- **Goals:**

- Only the signer with their **private key** sk can generate a signature for a msg
- Anyone with the **public key** pk can verify the authenticity of the signature of a msg

ML-DSA

- Digital Signature Algorithm based on **M-LWE** and **M-SIS** problems
(no classical or quantum algorithms known to solve these problems efficiently)

ML-DSA

- Digital Signature Algorithm based on **M-LWE** and **M-SIS** problems
(no classical or quantum algorithms known to solve these problems efficiently)
- Ring $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$ with $n = 256 = 2^8$ et $q = 8\,380\,417 = 2^{23} - 2^{13} + 1$

Version	ML-DSA-44 (SHA-256 🏹)	ML-DSA-65 (AES-192 🏹)	ML-DSA-87 (AES-256 🏹)
η	2	4	2
(k, ℓ)	(4, 4)	(6, 5)	(8, 7)
γ_1	2^{17}	2^{19}	2^{19}
γ_2	$\frac{q-1}{88}$	$\frac{q-1}{32}$	$\frac{q-1}{32}$
$\beta = \eta \times \tau$	$2 \times 39 = 78$	$4 \times 49 = 196$	$2 \times 60 = 120$

ML-DSA

- Digital Signature Algorithm based on **M-LWE** and **M-SIS** problems
(no classical or quantum algorithms known to solve these problems efficiently)
- Ring $\mathcal{R}_q = \mathbb{Z}_q[X]/(X^n + 1)$ with $n = 256 = 2^8$ et $q = 8\,380\,417 = 2^{23} - 2^{13} + 1$

Version	ML-DSA-44 (SHA-256 🏆)	ML-DSA-65 (AES-192 🏆)	ML-DSA-87 (AES-256 🏆)
η	2	4	2
(k, ℓ)	(4, 4)	(6, 5)	(8, 7)
γ_1	2^{17}	2^{19}	2^{19}
γ_2	$\frac{q-1}{88}$	$\frac{q-1}{32}$	$\frac{q-1}{32}$
$\beta = \eta \times \tau$	$2 \times 39 = 78$	$4 \times 49 = 196$	$2 \times 60 = 120$

- PQC DSA recommended by NIST for most use cases:
 - Easy scalable security: modify k and ℓ
 - Size of pk + size of sign reduced: to optimize bandwidth
 - Constant time and "hedged" version: security against SCA and FA

OPEN

ML-DSA Remarks

- Reference implementations may differ from the official specification . . . but expected
 - Alternative implementation from the FIPS and the reference paper (for performances)
 - Implementation by polynomial instead of vector (for performances)
 - Change in polynomial multiplication used (for performances)

ML-DSA Remarks

- Reference implementations may differ from the official specification ... but expected
 - Alternative implementation from the FIPS and the reference paper (for performances)
 - Implementation by polynomial instead of vector (for performances)
 - Change in polynomial multiplication used (for performances)
- t_0 is not private and can be recovered with reasonable amount of signatures [1]
 - Between 200 000 and 500 000 signatures depending on the version
- ML-DSA Sizes:

	Private Key	Public Key	Signature Size
ML-DSA-44	2 560	1 312	2 420
ML-DSA-65	4 032	1 952	3 309
ML-DSA-87	4 896	2 592	4 627

[1] P. Azevedo-Oliveira, A. Calle Viera, B. Cogliati, L. Goubin,
Uncompressing Dilithium's public key, CRYPTO25

OPEN

KeyGen:

1 $\mathbf{A} \xleftarrow{\text{die}} \mathcal{R}_q^{k \times \ell}$

2 $(\mathbf{s}_1, \mathbf{s}_2) \xleftarrow{\text{die}} \mathcal{S}_\eta^\ell \times \mathcal{S}_\eta^k$

3 $\mathbf{t} = \mathbf{A} \mathbf{s}_1 + \mathbf{s}_2 \in \mathcal{R}_q^k$

4 $\mathbf{t}_1, \mathbf{t}_0 = \text{Decompose}(\mathbf{t})$

5 return $\text{pk} = (\mathbf{A}, \mathbf{t}_1), \text{sk} = (\mathbf{A}, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0)$

KeyGen:

1 $\mathbf{A} \xleftarrow{\text{dice}} \mathcal{R}_q^{k \times \ell} \rightarrow \begin{pmatrix} (\mathbf{A})_{0,0} & \cdots & (\mathbf{A})_{0,\ell-1} \\ (\mathbf{A})_{1,0} & \cdots & (\mathbf{A})_{1,\ell-1} \\ \vdots & \ddots & \vdots \\ (\mathbf{A})_{k-1,0} & \cdots & (\mathbf{A})_{k-1,\ell-1} \end{pmatrix}$

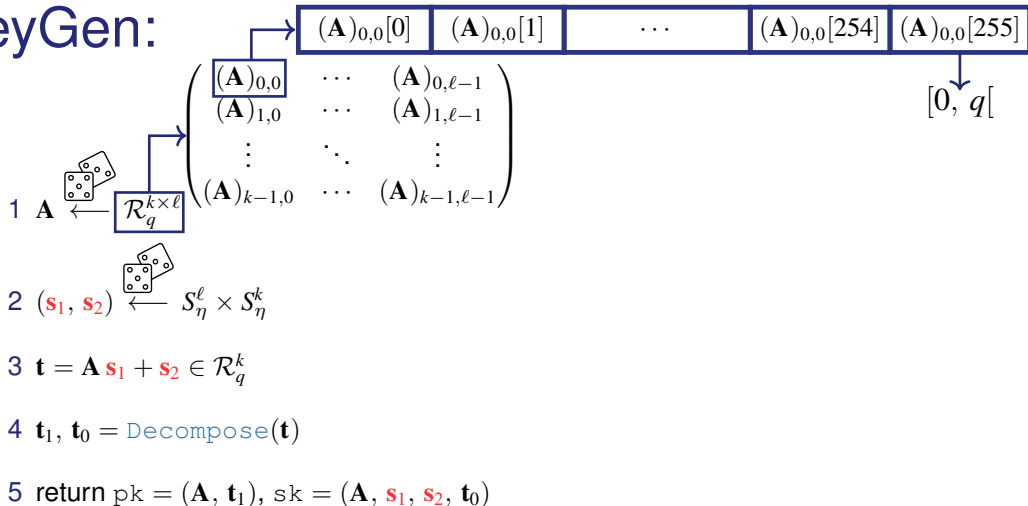
2 $(\mathbf{s}_1, \mathbf{s}_2) \xleftarrow{\text{dice}} \mathcal{S}_\eta^\ell \times \mathcal{S}_\eta^k$

3 $\mathbf{t} = \mathbf{A} \mathbf{s}_1 + \mathbf{s}_2 \in \mathcal{R}_q^k$

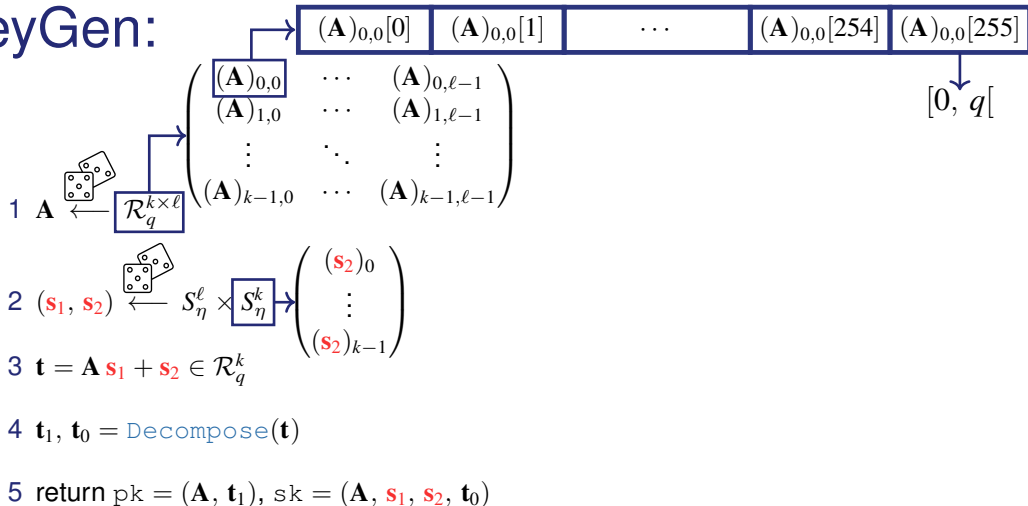
4 $\mathbf{t}_1, \mathbf{t}_0 = \text{Decompose}(\mathbf{t})$

5 return $\text{pk} = (\mathbf{A}, \mathbf{t}_1), \text{sk} = (\mathbf{A}, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0)$

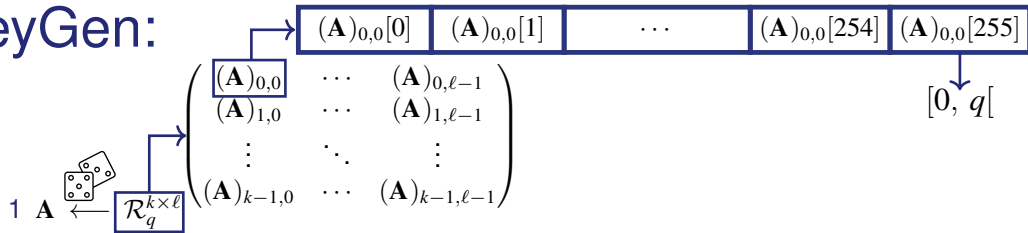
KeyGen:



KeyGen:



KeyGen:



2 $(\mathbf{s}_1, \mathbf{s}_2) \xleftarrow{\text{die}} \mathcal{S}_\eta^\ell \times \mathcal{S}_\eta^k \rightarrow \begin{pmatrix} (\mathbf{s}_2)_0 \\ \vdots \\ (\mathbf{s}_2)_{k-1} \end{pmatrix}$

3 $\mathbf{t} = \mathbf{A} \mathbf{s}_1 + \mathbf{s}_2 \in \mathcal{R}_q^k$

4 $\mathbf{t}_1, \mathbf{t}_0 = \text{Decompose}(\mathbf{t}) \longrightarrow \mathbf{t} = \mathbf{t}_1 2^{13} + \mathbf{t}_0$

5 return $\text{pk} = (\mathbf{A}, \mathbf{t}_1), \text{sk} = (\mathbf{A}, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0)$

- The matrix $\mathbf{A}$ is stored as a public seed to reduce storage requirements
The coefficients of s_1/s_2 are small ($\eta = 2$ or 4)

Sign(msg , $sk = (\mathbf{A}, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0)$):

```
1  $(\mathbf{z}, \mathbf{h}) = \perp$ 
2 while  $(\mathbf{z}, \mathbf{h}) = \perp$  do
3     $\mathbf{y} \leftarrow \tilde{S}_{\gamma_1}^\ell$ 
4    $\mathbf{w} = \mathbf{A} \mathbf{y}$ 
5    $\mathbf{w}_1, \mathbf{w}_0 = \text{Decompose}(\mathbf{w})$ 
6    $c \in B_\tau = H(pk \parallel msg \parallel \mathbf{w}_1)$ 
7    $\mathbf{z} = \mathbf{y} + c \mathbf{s}_1$ 
8   if  $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$  or  $\|\mathbf{w}_0 - c \mathbf{s}_2\|_\infty \geq \gamma_2 - \beta$ , then  $(\mathbf{z}, \mathbf{h}) = \perp$ 
9   else,
10      $\mathbf{h} = \text{MakeHint}(\mathbf{w}_1, \mathbf{w}_0 - c \mathbf{s}_2 + c \mathbf{t}_0)$ 
11     if  $\|c \mathbf{t}_0\|_\infty \geq \gamma_2$  or  $\|\mathbf{h}\|_1 > \omega$ , then  $(\mathbf{z}, \mathbf{h}) = \perp$ 
12   return  $\sigma = (c, \mathbf{z}, \mathbf{h})$ 
```

Sign(msg , $sk = (\mathbf{A}, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0)$):

- 3 $\mathbf{y} \xleftarrow{\tilde{S}_{\gamma_1}^\ell}$ 
- 4 $\mathbf{w} = \mathbf{A} \mathbf{y}$
- 5 $\mathbf{w}_1, \mathbf{w}_0 = \text{Decompose}(\mathbf{w}) \longrightarrow \mathbf{w} = \mathbf{w}_1 2\gamma_2 + \mathbf{w}_0$
- 6 $c \in B_\tau = H(pk \parallel msg \parallel \mathbf{w}_1)$

Sign(msg , $sk = (\mathbf{A}, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0)$):

- 3 $\mathbf{y} \xleftarrow{\tilde{S}_{\gamma_1}^\ell}$
- 4 $\mathbf{w} = \mathbf{A} \mathbf{y}$
- 5 $\mathbf{w}_1, \mathbf{w}_0 = \text{Decompose}(\mathbf{w}) \longrightarrow \mathbf{w} = \mathbf{w}_1 2\gamma_2 + \mathbf{w}_0$
- 6 $c \in B_\tau = H(pk \parallel msg \parallel \mathbf{w}_1)$
- 7 $\mathbf{z} = \mathbf{y} + c \mathbf{s}_1$
- 9 if $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$ or $\|\mathbf{w}_0 - c \mathbf{s}_2\|_\infty \geq \gamma_2 - \beta$, then $(\mathbf{z}, \mathbf{h}) = \perp$

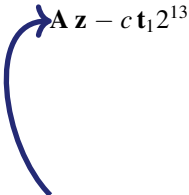
Sign(msg , $sk = (\mathbf{A}, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0)$):

```
2 while  $(\mathbf{z}, \mathbf{h}) = \perp$  do
  3     $\mathbf{y} \leftarrow \tilde{S}_{\gamma_1}^\ell$ 
  4    $\mathbf{w} = \mathbf{A} \mathbf{y}$ 
  5    $\mathbf{w}_1, \mathbf{w}_0 = \text{Decompose}(\mathbf{w}) \longrightarrow \mathbf{w} = \mathbf{w}_1 2\gamma_2 + \mathbf{w}_0$ 
  6    $c \in B_\tau = H(pk \parallel msg \parallel \mathbf{w}_1)$ 
  7    $\mathbf{z} = \mathbf{y} + c \mathbf{s}_1$ 
  8   if  $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$  or  $\|\mathbf{w}_0 - c \mathbf{s}_2\|_\infty \geq \gamma_2 - \beta$ , then  $(\mathbf{z}, \mathbf{h}) = \perp$ 
  9   else,
 10      $\mathbf{h} = \text{MakeHint}(\mathbf{w}_1, \mathbf{w}_0 - c \mathbf{s}_2 + c \mathbf{t}_0)$ 
 11     if  $\|c \mathbf{t}_0\|_\infty \geq \gamma_2$  or  $\|\mathbf{h}\|_1 > \omega$ , then  $(\mathbf{z}, \mathbf{h}) = \perp$ 
 12   return  $\sigma = (c, \mathbf{z}, \mathbf{h})$ 
```

Verify(pk = (A, t₁), msg, σ = (c, z, h)):

$$1 \text{ } w'_1 = \text{UseHint}(\mathbf{h}, \mathbf{A} \mathbf{z} - c \mathbf{t}_1 2^{13})$$

Verify(pk = (A, t₁), msg, σ = (c, z, h)):


$$1 \ w'_1 = \text{UseHint}(\mathbf{h}, \boxed{\mathbf{A} \mathbf{z} - c \mathbf{t}_1 2^{13}})$$
$$\mathbf{A} \mathbf{z} - c \mathbf{t}_1 2^{13}$$

Verify(pk = (A, t₁), msg, σ = (c, z, h)):

$$\mathbf{A} \mathbf{z} - c \mathbf{t}_1 2^{13} = \mathbf{A} \overbrace{(\mathbf{y} + c \mathbf{s}_1)}^{\mathbf{z}} - c \overbrace{(\mathbf{A} \mathbf{s}_1 + \mathbf{s}_2 - \mathbf{t}_0)}^{\mathbf{t}_1 2^{13}}$$

$$1 \text{ } w'_1 = \text{UseHint}(\mathbf{h}, \boxed{\mathbf{A} \mathbf{z} - c \mathbf{t}_1 2^{13}})$$

Verify(pk = (A, t₁), msg, σ = (c, z, h)):

$$\begin{aligned} \mathbf{A} \mathbf{z} - c \mathbf{t}_1 2^{13} &= \mathbf{A} \overbrace{(\mathbf{y} + c \mathbf{s}_1)}^{\mathbf{z}} - c \overbrace{(\mathbf{A} \mathbf{s}_1 + \mathbf{s}_2 - \mathbf{t}_0)}^{\mathbf{t}_1 2^{13}} \\ &= \underbrace{\mathbf{A} \mathbf{y}}_{\mathbf{w}} - c \mathbf{s}_2 + c \mathbf{t}_0 \\ &= \mathbf{w} - c \mathbf{s}_2 + c \mathbf{t}_0 \end{aligned}$$

$$1 \text{ w}'_1 = \text{UseHint}(\mathbf{h}, \boxed{\mathbf{A} \mathbf{z} - c \mathbf{t}_1 2^{13}})$$

Verify(pk = (A, t₁), msg, σ = (c, z, h)):

$$\begin{aligned} \mathbf{A} \mathbf{z} - c \mathbf{t}_1 2^{13} &= \mathbf{A} \overbrace{(\mathbf{y} + c \mathbf{s}_1)}^{\mathbf{z}} - c \overbrace{(\mathbf{A} \mathbf{s}_1 + \mathbf{s}_2 - \mathbf{t}_0)}^{\mathbf{t}_1 2^{13}} \\ &= \mathbf{A} \mathbf{y} - c \mathbf{s}_2 + c \mathbf{t}_0 \\ &= \underbrace{\mathbf{A} \mathbf{y}}_{\mathbf{w}} - c \mathbf{s}_2 + c \mathbf{t}_0 \end{aligned}$$

$$\text{Lemme 1.1 [2]} \implies \text{UseHint}(\mathbf{h}, \mathbf{w} - c \mathbf{s}_2 + c \mathbf{t}_0) = \text{HighBits}(\mathbf{w} - \overbrace{c \mathbf{s}_2}^{\leq \beta})$$

$$1 \ w'_1 = \text{UseHint}(\mathbf{h}, \boxed{\mathbf{A} \mathbf{z} - c \mathbf{t}_1 2^{13}})$$

[2] S. Bai, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, D. Stehlé,
CRYSTALS - Dilithium: Digital Signatures from Module Lattices

OPEN

Verify(pk = (A, t₁), msg, σ = (c, z, h)):

$$\begin{aligned} \mathbf{A} \mathbf{z} - c \mathbf{t}_1 2^{13} &= \mathbf{A} \overbrace{(\mathbf{y} + c \mathbf{s}_1)}^{\mathbf{z}} - c \overbrace{(\mathbf{A} \mathbf{s}_1 + \mathbf{s}_2 - \mathbf{t}_0)}^{\mathbf{t}_1 2^{13}} \\ &= \mathbf{A} \mathbf{y} - c \mathbf{s}_2 + c \mathbf{t}_0 \\ &= \underbrace{\mathbf{w}} - c \mathbf{s}_2 + c \mathbf{t}_0 \end{aligned}$$

$$\text{Lemme 1.1 [2]} \implies \text{UseHint}(\mathbf{h}, \mathbf{w} - c \mathbf{s}_2 + c \mathbf{t}_0) = \text{HighBits}(\mathbf{w} - \overbrace{c \mathbf{s}_2}^{\leq \beta})$$

$$\begin{aligned} \text{Lemme 2 [2]} \implies \text{HighBits}(\mathbf{w} - c \mathbf{s}_2) &= \text{HighBits}(\mathbf{w}) \\ &= \underbrace{\mathbf{w}_1} \end{aligned}$$

$$1 \mathbf{w}'_1 = \text{UseHint}(\mathbf{h}, \boxed{\mathbf{A} \mathbf{z} - c \mathbf{t}_1 2^{13}})$$

[2] S. Bai, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, D. Stehlé,
CRYSTALS - Dilithium: Digital Signatures from Module Lattices

OPEN

Verify(pk = (A, t₁), msg, σ = (c, z, h)):

$$\begin{aligned} \mathbf{A} \mathbf{z} - c \mathbf{t}_1 2^{13} &= \mathbf{A} \overbrace{(\mathbf{y} + c \mathbf{s}_1)}^{\mathbf{z}} - c \overbrace{(\mathbf{A} \mathbf{s}_1 + \mathbf{s}_2 - \mathbf{t}_0)}^{\mathbf{t}_1 2^{13}} \\ &= \mathbf{A} \mathbf{y} - c \mathbf{s}_2 + c \mathbf{t}_0 \\ &= \underbrace{\mathbf{w}} - c \mathbf{s}_2 + c \mathbf{t}_0 \end{aligned}$$

$$\begin{aligned} \text{Lemme 1.1 [2]} \implies \text{UseHint}(\mathbf{h}, \mathbf{w} - c \mathbf{s}_2 + c \mathbf{t}_0) &= \text{HighBits}(\mathbf{w} - \overbrace{c \mathbf{s}_2}^{\leq \beta}) \\ \text{Lemme 2 [2]} \implies \text{HighBits}(\mathbf{w} - c \mathbf{s}_2) &= \underbrace{\text{HighBits}(\mathbf{w})}_{\mathbf{w}_1} \end{aligned}$$

$$1 \mathbf{w}'_1 = \text{UseHint}(\mathbf{h}, \boxed{\mathbf{A} \mathbf{z} - c \mathbf{t}_1 2^{13}})$$

$$2 \text{ if } \|\mathbf{z}\|_\infty < \gamma_1 - \beta \text{ and } c = \text{H}(\text{pk} \parallel \text{msg} \parallel \mathbf{w}'_1) \text{ and } \|\mathbf{h}\|_1 \leq \omega$$

$$3 \quad \text{return } \textit{True}$$

$$4 \text{ else}$$

$$5 \quad \text{return } \textit{False}$$

[2] S. Bai, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, D. Stehlé,
CRYSTALS - Dilithium: Digital Signatures from Module Lattices

OPEN

ML-DSA FA summary (until april 2025)

Attack	Value targeted	References
DFA	c	Bruinderink and Pessl, 2018; Wang et al., 2024
Nonce reuse	s_1 and s_2	Ravi, Roy, et al., 2019
Skip Addition	$As_1 + s_2$	Bindel et al., 2016
	$y + cs_1$	Bindel et al., 2016; Krahmer et al., 2024; Ravi, Jhanwar, et al., 2019
Skipping equality check	$c == c'$	
Loop abort Fault	y	Espitau et al., n.d.; Ulitzsch et al., 2023
NTT conversion	c, y	Ravi et al., 2023
Fault on public value	A	Krahmer et al., 2024
Subtraction	$Az - ct_12^d$	Bauer and De Santis, 2023; Calle Viera et al., 2024
Sampling	c	Calle Viera et al., 2024
Exponentiation	t_12^d	Calle Viera et al., 2024
Signature correction	s_1	Islam et al., 2022
KECCAK	$\rho, \rho'', \tilde{c}$	Jendral, 2024; Wang et al., 2024
Sampling	s_1, s_2, y	Wang et al., 2024
Check norm	z, w_0	Azevedo-Oliveira et al., 2025

OPEN

ML-DSA FA summary (until april 2025)

Attack	Value targeted	References
DFA	c	Bruinderink and Pessl, 2018; Wang et al., 2024
Nonce reuse	s_1 and s_2	Ravi, Roy, et al., 2019
Skip Addition	$As_1 + s_2$	Bindel et al., 2016
	$y + cs_1$	Bindel et al., 2016; Krahmer et al., 2024; Ravi, Jhanwar, et al., 2019
Skipping equality check	$c == c'$	
Loop abort Fault	y	Espitau et al., n.d.; Ulitzsch et al., 2023
NTT conversion	c, y	Ravi et al., 2023
Fault on public value	A	Krahmer et al., 2024
Subtraction	$Az - ct_12^d$	Bauer and De Santis, 2023; Calle Viera et al., 2024
Sampling	c	Calle Viera et al., 2024
Exponentiation	t_12^d	Calle Viera et al., 2024
Signature correction	s_1	Islam et al., 2022
KECCAK	$\rho, \rho'', \tilde{c}$	Jendral, 2024; Wang et al., 2024
Sampling	s_1, s_2, y	Wang et al., 2024
Check norm	z, w_0	Azevedo-Oliveira et al., 2025

OPEN

ML-DSA FA summary (until april 2025)

Attack	Value targeted	References
DFA	c	Bruinderink and Pessl, 2018; Wang et al., 2024
Nonce reuse	s_1 and s_2	Ravi, Roy, et al., 2019
Skip Addition	$As_1 + s_2$	Bindel et al., 2016
	$y + cs_1$	Bindel et al., 2016; Krahmer et al., 2024; Ravi, Jhanwar, et al., 2019
Skipping equality check	$c == c'$	
Loop abort Fault	y	Espitau et al., n.d.; Ulitzsch et al., 2023
NTT conversion	c, y	Ravi et al., 2023
Fault on public value	A	Krahmer et al., 2024
Subtraction	$Az - ct_12^d$	Bauer and De Santis, 2023; Calle Viera et al., 2024
Sampling	c	Calle Viera et al., 2024
Exponentiation	t_12^d	Calle Viera et al., 2024
Signature correction	s_1	Islam et al., 2022
KECCAK	$\rho, \rho'', \tilde{c}$	Jendral, 2024; Wang et al., 2024
Sampling	s_1, s_2, y	Wang et al., 2024
Check norm	z, w_0	Azevedo-Oliveira et al., 2025

OPEN

Sign(msg , $sk = (\mathbf{A}, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0)$):

```
1  $(\mathbf{z}, \mathbf{h}) = \perp$ 
2 while  $(\mathbf{z}, \mathbf{h}) = \perp$  do
3    $\mathbf{y} \xleftarrow{\text{die}} \tilde{S}_{\gamma_1}^l$ 
4    $\mathbf{w} = \mathbf{A} \mathbf{y}$ 
5    $\mathbf{w}_1, \mathbf{w}_0 = \text{Decompose}(\mathbf{w})$ 
6    $c \in B_\tau = H(\text{pk} \parallel msg \parallel \mathbf{w}_1)$ 
7    $\mathbf{z} = \mathbf{y} + c \mathbf{s}_1$ 
8    $\mathbf{r}_0 = \mathbf{w}_0 - c \mathbf{s}_2$ 
9   if  $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$  or  $\|\mathbf{r}_0\|_\infty \geq \gamma_2 - \beta$  then  $(\mathbf{z}, \mathbf{h}) = \perp$ 
10  else,
11     $\mathbf{h} = \text{MakeHint}(\mathbf{w}_1, \mathbf{r}_0 + c \mathbf{t}_0)$ 
12    if  $\|c \mathbf{t}_0\|_\infty \geq \gamma_2$  or  $\|\mathbf{h}\|_1 > \omega$ , then  $(\mathbf{z}, \mathbf{h}) = \perp$ 
13 return  $\sigma = (c, \mathbf{z}, \mathbf{h})$ 
```

• Necessary for security as stated in [1]

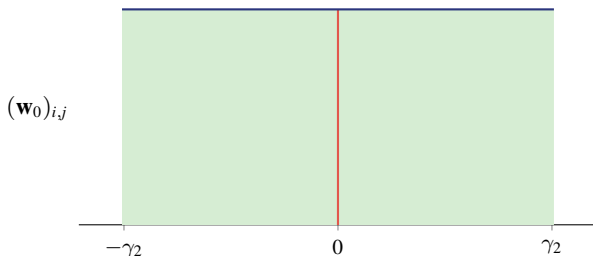
[2] S. Bai, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, D. Stehlé,

CRYSTALS - Dilithium: Digital Signatures from Module Lattices

Sign(msg , $sk = (\mathbf{A}, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0)$):

```
1  $(\mathbf{z}, \mathbf{h}) = \perp$ 
2 while  $(\mathbf{z}, \mathbf{h}) = \perp$  do
3    $\mathbf{y} \xleftarrow{\text{die}} \tilde{S}_{\gamma_1}^l$ 
4    $\mathbf{w} = \mathbf{A} \mathbf{y}$ 
5    $\mathbf{w}_1, \mathbf{w}_0 = \text{Decompose}(\mathbf{w})$ 
6    $c \in B_\tau = H(pk \parallel msg \parallel \mathbf{w}_1)$ 
7    $\mathbf{z} = \mathbf{y} + c \mathbf{s}_1$ 
8    $\mathbf{r}_0 = \mathbf{w}_0 - c \mathbf{s}_2$ 
9   if  $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$  or  $\|\mathbf{r}_0\|_\infty \geq \gamma_2 - \beta$  then  $(\mathbf{z}, \mathbf{h}) = \perp$ 
10  else,
11     $\mathbf{h} = \text{MakeHint}(\mathbf{w}_1, \mathbf{r}_0 + c \mathbf{t}_0)$ 
12    if  $\|c \mathbf{t}_0\|_\infty \geq \gamma_2$  or  $\|\mathbf{h}\|_1 > \omega$ , then  $(\mathbf{z}, \mathbf{h}) = \perp$ 
13 return  $\sigma = (c, \mathbf{z}, \mathbf{h})$ 
```

• Necessary for security as stated in [1]

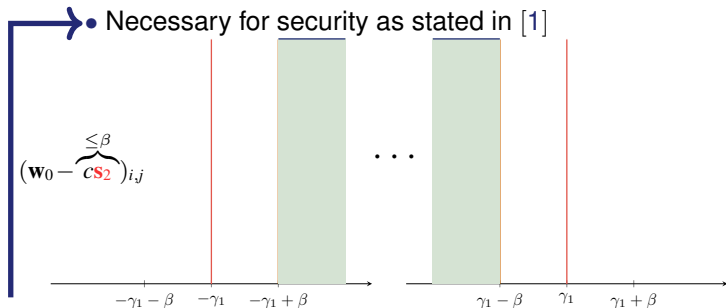


[2] S. Bai, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, D. Stehlé,

CRYSTALS - Dilithium: Digital Signatures from Module Lattices

Sign(msg , $sk = (\mathbf{A}, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0)$):

```
1  $(\mathbf{z}, \mathbf{h}) = \perp$ 
2 while  $(\mathbf{z}, \mathbf{h}) = \perp$  do
3    $\mathbf{y} \xleftarrow{\text{die}} \tilde{S}_{\gamma_1}^l$ 
4    $\mathbf{w} = \mathbf{A} \mathbf{y}$ 
5    $\mathbf{w}_1, \mathbf{w}_0 = \text{Decompose}(\mathbf{w})$ 
6    $c \in B_\tau = H(pk \parallel msg \parallel \mathbf{w}_1)$ 
7    $\mathbf{z} = \mathbf{y} + c \mathbf{s}_1$ 
8    $\mathbf{r}_0 = \mathbf{w}_0 - c \mathbf{s}_2$ 
9   if  $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$  or  $\|\mathbf{r}_0\|_\infty \geq \gamma_2 - \beta$  then  $(\mathbf{z}, \mathbf{h}) = \perp$ 
10  else,
11     $\mathbf{h} = \text{MakeHint}(\mathbf{w}_1, \mathbf{r}_0 + c \mathbf{t}_0)$ 
12    if  $\|c \mathbf{t}_0\|_\infty \geq \gamma_2$  or  $\|\mathbf{h}\|_1 > \omega$ , then  $(\mathbf{z}, \mathbf{h}) = \perp$ 
13 return  $\sigma = (c, \mathbf{z}, \mathbf{h})$ 
```

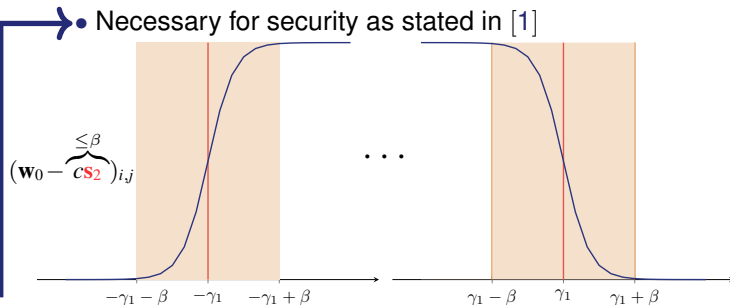


[2] S. Bai, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, D. Stehlé,

CRYSTALS - Dilithium: Digital Signatures from Module Lattices

Sign(msg, sk = (A, s₁, s₂, t₀)):

```
1 (z, h) = ⊥
2 while (z, h) = ⊥ do
3   y ←  S̃γ1l
4   w = A y
5   w1, w0 = Decompose(w)
6   c ∈ Bτ = H(pk || msg || w1)
7   z = y + c s1
8   r0 = w0 - c s2
9   if ||z||∞ ≥ γ1 - β or ||r0||∞ ≥ γ2 - β, then (z, h) = ⊥
10  else,
11    h = MakeHint(w1, r0 + c t0)
12    if ||c t0||∞ ≥ γ2 or ||h||1 > ω, then (z, h) = ⊥
13  return σ = (c, z, h)
```



• How hard is it to exploit such signatures?

[2] S. Bai, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, P. Schwabe, G. Seiler, D. Stehlé,

CRYSTALS - Dilithium: Digital Signatures from Module Lattices

Sign(msg , $sk = (\mathbf{A}, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0)$):

```
1  ( $\mathbf{z}, \mathbf{h}$ ) =  $\perp$ 
2  while ( $\mathbf{z}, \mathbf{h}$ ) =  $\perp$  do
3       $\mathbf{y} \in \tilde{S}_{\gamma_1}^l$ 
4       $\mathbf{w} = \mathbf{A} \mathbf{y}$ 
5       $\mathbf{w}_1, \mathbf{w}_0 = \text{Decompose}(\mathbf{w})$ 
6       $c \in B_\tau = H(pk \parallel msg \parallel \mathbf{w}_1)$ 
7       $\mathbf{z} = \mathbf{y} + c \mathbf{s}_1$ 
8       $\mathbf{r}_0 = \mathbf{w}_0 - c \mathbf{s}_2$ 
9      if  $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$  or  $\|\mathbf{r}_0\|_\infty \geq \gamma_2 - \beta$ , then ( $\mathbf{z}, \mathbf{h}$ ) =  $\perp$ 
10     else,
11          $\mathbf{h} = \text{MakeHint}(\mathbf{w}_1, \mathbf{r}_0 + c \mathbf{t}_0)$ 
12         if  $\|c \mathbf{t}_0\|_\infty \geq \gamma_2$  or  $\|\mathbf{h}\|_1 > \omega$ , then ( $\mathbf{z}, \mathbf{h}$ ) =  $\perp$ 
13 return  $\sigma = (c, \mathbf{z}, \mathbf{h})$ 
```

Sign(msg , $sk = (\mathbf{A}, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0)$):

```
1 ( $\mathbf{z}, \mathbf{h}$ ) =  $\perp$ 
2 while ( $\mathbf{z}, \mathbf{h}$ ) =  $\perp$  do
3    $\mathbf{y} \in \tilde{S}_{\gamma_1}^l$ 
4    $\mathbf{w} = \mathbf{A} \mathbf{y}$ 
5    $\mathbf{w}_1, \mathbf{w}_0 = \text{Decompose}(\mathbf{w})$ 
6    $c \in B_\tau = H(\text{pk} || msg || \mathbf{w}_1)$ 
7    $\mathbf{z} = \mathbf{y} + c \mathbf{s}_1$ 
8    $\mathbf{r}_0 = \mathbf{w}_0 - c \mathbf{s}_2$ 
9   if  $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$  or  $\|\mathbf{r}_0\|_\infty \geq \gamma_2 - \beta$ , then ( $\mathbf{z}, \mathbf{h}$ ) =  $\perp$ 
10  else,
11     $\mathbf{h} = \text{MakeHint}(\mathbf{w}_1, \mathbf{r}_0 + c \mathbf{t}_0)$ 
12    if  $\|c \mathbf{t}_0\|_\infty \geq \gamma_2$  or  $\|\mathbf{h}\|_1 > \omega$ , then ( $\mathbf{z}, \mathbf{h}$ ) =  $\perp$ 
13 return  $\sigma = (c, \mathbf{z}, \mathbf{h})$ 
```



```
168 polyveck_sub(&r0, &w0, &cs2);
169 polyveck_reduce(&r0);
170 if(polyveck_chknorm(&r0, GAMMA2 - BETA))
171   goto rej;
```

Sign(msg , $sk = (\mathbf{A}, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0)$):

```
1 ( $\mathbf{z}, \mathbf{h}$ ) =  $\perp$ 
2 while ( $\mathbf{z}, \mathbf{h}$ ) =  $\perp$  do
3    $\mathbf{y} \in \tilde{S}_{\gamma_1}^l$ 
4    $\mathbf{w} = \mathbf{A} \mathbf{y}$ 
5    $\mathbf{w}_1, \mathbf{w}_0 = \text{Decompose}(\mathbf{w})$ 
6    $c \in B_\tau = H(pk || msg || \mathbf{w}_1)$ 
7    $\mathbf{z} = \mathbf{y} + c \mathbf{s}_1$ 
8    $\mathbf{r}_0 = \mathbf{w}_0 - c \mathbf{s}_2$ 
9   if  $\|\mathbf{z}\|_\infty \geq \gamma_1 - \beta$  or  $\|\mathbf{r}_0\|_\infty \geq \gamma_2 - \beta$ , then ( $\mathbf{z}, \mathbf{h}$ ) =  $\perp$ 
10  else,
11     $\mathbf{h} = \text{MakeHint}(\mathbf{w}_1, \mathbf{r}_0 + c \mathbf{t}_0)$ 
12    if  $\|c \mathbf{t}_0\|_\infty \geq \gamma_2$  or  $\|\mathbf{h}\|_1 > \omega$ , then ( $\mathbf{z}, \mathbf{h}$ ) =  $\perp$ 
13  return  $\sigma = (c, \mathbf{z}, \mathbf{h})$ 
```

```
168 polyveck_sub(&r0, &w0, &cs2);
169 polyveck_reduce(&r0);
170 if(polyveck_chknorm(&r0, GAMMA2 - BETA))
171     goto rej;
```

Cortex M4 : -Os

```
ldf6 mov    r1, fp    ;fp = GAMMA2 - BETA
ldf8 adds   r0, #104
ldfa bl     23b4      ;call to polyveck_chknorm
ldfe cmp    r0, #0    ;polyveck_chknorm out
1e00 bne.w   1cc0      ;if not 0 go to rej
```

Sign(msg, sk = (A, s₁, s₂, t₀)):

```
1 (z, h) = ⊥
2 while (z, h) = ⊥ do
3   y ∈ S̃γ1l
4   w = A y
5   w1, w0 = Decompose(w)
6   c ∈ Bτ = H(pk || msg || w1)
7   z = y + c s1
8   r0 = w0 - c s2
9   if ||z||∞ ≥ γ1 - β or ||r0||∞ ≥ γ2 - β, then (z, h) = ⊥
10  else,
11    h = MakeHint(w1, r0 + c t0)
12    if ||c t0||∞ ≥ γ2 or ||h||1 > ω, then (z, h) = ⊥
13  return σ = (c, z, h)
```

```
168 polyveck_sub(&r0, &w0, &cs2);
169 polyveck_reduce(&r0);
170 if(polyveck_chknorm(&r0, GAMMA2 - BETA))
171     goto rej;
```

Cortex M4 : -Os

```
ldf6 mov    r1, fp    ;fp = GAMMA2 - BETA
ldf8 adds   r0, #104
ldfa bl     23b4      ;call to polyveck_chknorm
ldfe cmp    r0, #0    ;polyveck_chknorm out
1e00 bne.w   1cc0      ;if not 0 go to rej
```

Only one instruction to fault

- Validated on ChipWhisperer: clock glitch

Resolution

- For every polynomial $i \in [0, k - 1]$ we solve the linear programming problem:

$$\begin{aligned} & \text{maximize} && 0 \\ & \text{under constraints} && \mathbf{C}_i^+(\mathbf{s}_2)_i \geq b_i^+ \\ & && \mathbf{C}_i^-(\mathbf{s}_2)_i \leq b_i^- \\ & && (\mathbf{s}_2)_i \in [-\eta, \eta]^{256} \end{aligned}$$

- If we have enough inequalities, then there is a real solution
- For every solution $i \in [0, k - 1]$, we round the 256 coefficients to the nearest integer
- And it should correspond to the true solution
- As we know $\mathbf{t}_0$ (by hypothesis) and $\mathbf{s}_2$ (that we just get) then:

$$\mathbf{s}_1 = ({}^t\mathbf{A} \mathbf{A})^{-1} {}^t\mathbf{A} (\mathbf{t}_1 2^{13} + (\mathbf{t}_0 - \mathbf{s}_2)) \quad (1)$$

- And with $\mathbf{s}_1$ we can forge signatures [3]

[3] L. Groot Bruinderink, P. Pessl,
Differential Fault Attacks on Deterministic Lattice Signatures

ML-DSA Sign FA: summary

- As the faulted signatures (without the $\|\mathbf{r}_0\|_\infty$ condition) are always verified

Remark

No way to detect signatures produced without the verification on $\|\mathbf{r}_0\|_\infty$

- Except to do the attack and build the inequalities . . .
- Practical evaluation on a ChipWhisperer with a clock glitch
- Between 1 500 000 and 4 000 000 signatures to get enough inequalities to recover s_2 [8]

[8] P. Azevedo Oliveira, A. Calle Viera, B. Cogliati, L. Goubin,
Finding a polytope: A practical fault attack against dilithium

How to make accept a false signature?

Goal: Make accept false signatures by **Verify** with faults injected

Why: Access confidential data/Install arbitray software

Verification **checks** are the **most sensitive** and usually **hardened**: 3 checks $\approx$ 3 faults

How to make accept a false signature?

Goal: Make accept false signatures by **Verify** with faults injected

Why: Access confidential data/Install arbitray software

Verification **checks** are the **most sensitive** and usually **hardened**: 3 checks $\approx$ 3 faults

Other sensitive **locations** requiring possibly **less faults** to inject?

How to make accept a false signature?

Goal: Make accept false signatures by **Verify** with faults injected

Why: Access confidential data/Install arbitray software

Verification **checks** are the **most sensitive** and usually **hardened**: 3 checks $\approx$ 3 faults

Other sensitive locations requiring possibly **less faults** to inject?

$$1 ||z||_{\infty} < \gamma_1 - \beta$$

How to make accept a false signature?

Goal: Make accept false signatures by **Verify** with faults injected

Why: Access confidential data/Install arbitray software

Verification **checks** are the **most sensitive** and usually **hardened**: 3 checks $\approx$ 3 faults

Other sensitive locations requiring possibly **less faults** to inject?

1 Choose random z such that $\|z\|_{\infty} < \gamma_1 - \beta$ 1 $\|z\|_{\infty} < \gamma_1 - \beta$

How to make accept a false signature?

Goal: Make accept false signatures by **Verify** with faults injected

Why: Access confidential data/Install arbitray software

Verification **checks** are the **most sensitive** and usually **hardened**: 3 checks $\approx$ 3 faults

Other sensitive locations requiring possibly **less faults** to inject?

1 Choose random z such that $\|z\|_{\infty} < \gamma_1 - \beta$

1 $\|z\|_{\infty} < \gamma_1 - \beta$

2 $c = H(pk || M || w'_1)$

How to make accept a false signature?

Goal: Make accept false signatures by **Verify** with faults injected

Why: Access confidential data/Install arbitray software

Verification **checks** are the **most sensitive** and usually **hardened**: 3 checks $\approx$ 3 faults

Other sensitive locations requiring possibly **less faults** to inject?

1 Choose random z such that $\|z\|_{\infty} < \gamma_1 - \beta$

1 $\|z\|_{\infty} < \gamma_1 - \beta$

2 $\text{HighBits}(Az - ct_1 2^d) = \text{HighBits}(Az)$

How to make accept a false signature?

Goal: Make accept false signatures by **Verify** with faults injected

Why: Access confidential data/Install arbitray software

Verification **checks** are the **most sensitive** and usually **hardened**: 3 checks $\approx$ 3 faults

Other sensitive locations requiring possibly **less faults** to inject?

1 Choose random z such that $\|z\|_{\infty} < \gamma_1 - \beta$

1 $\|z\|_{\infty} < \gamma_1 - \beta$

2 Assure that $ct_1 2^d$ doesn't affect the high bits of Az

2 $\text{HighBits}(Az - ct_1 2^d) = \text{HighBits}(Az)$

How to make accept a false signature?

Goal: Make accept false signatures by **Verify** with faults injected

Why: Access confidential data/Install arbitray software

Verification **checks** are the **most sensitive** and usually **hardened**: 3 checks $\approx$ 3 faults

Other sensitive locations requiring possibly **less faults** to inject?

- | | |
|---|--|
| 1 Choose random z such that $\ z\ _{\infty} < \gamma_1 - \beta$ | 1 $\ z\ _{\infty} < \gamma_1 - \beta$ |
| 2 Assure that $ct_1 2^d$ doesn't affect the high bits of Az | 2 $\text{HighBits}(Az - ct_1 2^d) = \text{HighBits}(Az)$ |
| | 3 $\# \text{ 1's in } h \leq \omega$ |

How to make accept a false signature?

Goal: Make accept false signatures by **Verify** with faults injected

Why: Access confidential data/Install arbitray software

Verification **checks** are the **most sensitive** and usually **hardened**: 3 checks $\approx$ 3 faults

Other sensitive locations requiring possibly **less faults** to inject?

- | | |
|---|--|
| 1 Choose random z such that $\ z\ _{\infty} < \gamma_1 - \beta$ | 1 $\ z\ _{\infty} < \gamma_1 - \beta$ |
| 2 Assure that $ct_1 2^d$ doesn't affect the high bits of Az | 2 $\text{HighBits}(Az - ct_1 2^d) = \text{HighBits}(Az)$ |
| 3 Compute h with $\#$ 1's in $h \leq \omega$ accordingly | 3 $\#$ 1's in $h \leq \omega$ |

Where to target?

$$1 \mathbf{w}'_1 = \text{UseHint}(\mathbf{h}, \mathbf{A} \mathbf{z} - c \mathbf{t}_1 2^d)$$

Where to target?

NTT transform [6]



$$1 \mathbf{w}'_1 = \text{UseHint}(\mathbf{h}, \mathbf{A} \mathbf{z} - \boxed{c} \mathbf{t}_1 2^d)$$

Sampling of c [4, 5]



[4] A. Berzati, A. Calle Viera, K. Heydemann, FIA sensitivity of public parameters in the Dilithium verification **CARDIS 23**

[5] F. Garipay, Bypassing PQC Signature Verification with Fault Injection: Dilithium, XMSS, SPHINCS+ **BlackHat 25**

[6] P. Ravi, B. Yang, S. Bhasin et al, Fiddling the twiddle constants - fault injection analysis of the NTT **CHES 23**



Where to target?

$$w'_1 = \text{UseHint}(\mathbf{h}, \mathbf{A} \mathbf{z} - \boxed{c} \mathbf{t}_1 2^{\boxed{d}})$$

NTT transform [6] ←

Sampling of c [4, 5] ←

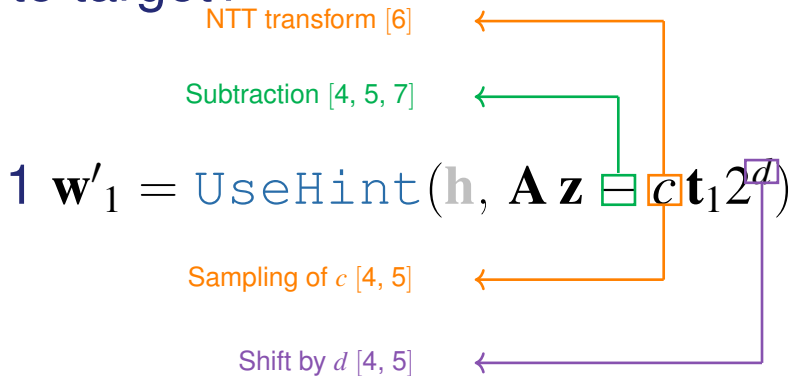
Shift by d [4, 5] ←

[4] A. Berzati, A. Calle Viera, K. Heydemann, FIA sensitivity of public parameters in the Dilithium verification **CARDIS 23**

[5] F. Garipay, Bypassing PQC Signature Verification with Fault Injection: Dilithium, XMSS, SPHINCS+ **BlackHat 25**

[6] P. Ravi, B. Yang, S. Bhasin et al, Fiddling the twiddle constants - fault injection analysis of the NTT **CHES 23**

Where to target?



- [4] A. Berzati, A. Calle Viera, K. Heydemann, FIA sensitivity of public parameters in the Dilithium verification **CARDIS 23**
- [5] F. Garipay, Bypassing PQC Signature Verification with Fault Injection: Dilithium, XMSS, SPHINCS+ **BlackHat 25**
- [6] P. Ravi, B. Yang, S. Bhasin et al, Fiddling the twiddle constants - fault injection analysis of the NTT **CHES 23**
- [7] S. Bauer, F. De Santis, Forging Dilithium and Falcon signatures by single fault injection **FDTC 23**

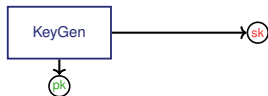


ML-KEM aka Kyber

But what is a KEM?

- Goal: Securely share a symmetric key over an insecure channel

PKE:

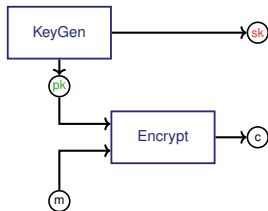


KEM:

But what is a KEM?

- Goal: Securely share a symmetric key over an insecure channel

PKE:

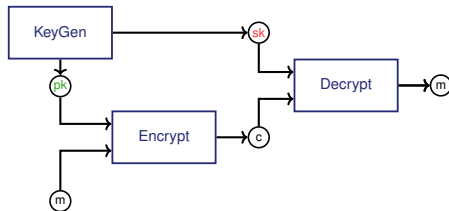


KEM:

But what is a KEM?

- Goal: Securely share a symmetric key over an insecure channel

PKE:

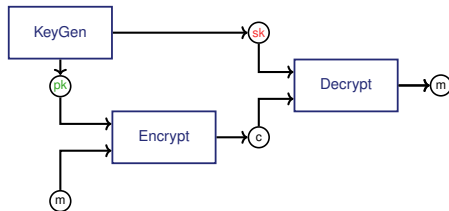


KEM:

But what is a KEM?

- Goal: Securely share a symmetric key over an insecure channel

PKE:



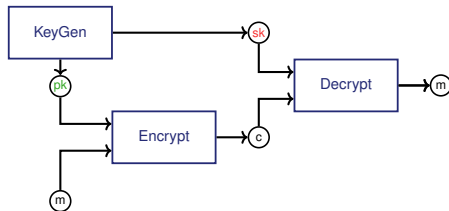
KEM:

- Allows to encrypt any message
- Typically secure against Chosen Plaintext Attacks
- Can be transformed into more secured KEM

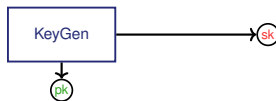
But what is a KEM?

- Goal: Securely share a symmetric key over an insecure channel

PKE:



KEM:

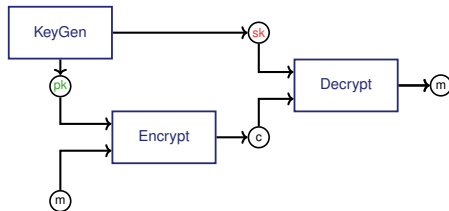


- Allows to encrypt any message
- Typically secure against Chosen Plaintext Attacks
- Can be transformed into more secured KEM

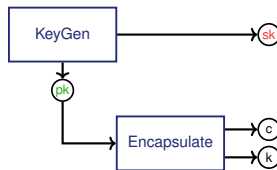
But what is a KEM?

- Goal: Securely share a symmetric key over an insecure channel

PKE:



KEM:

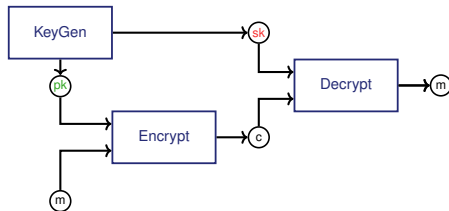


- Allows to encrypt any message
- Typically secure against Chosen Plaintext Attacks
- Can be transformed into more secured KEM

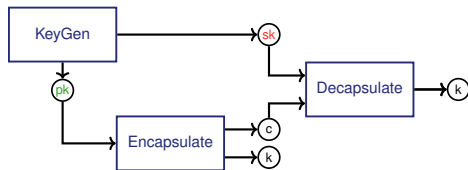
But what is a KEM?

- Goal: Securely share a symmetric key over an insecure channel

PKE:



KEM:

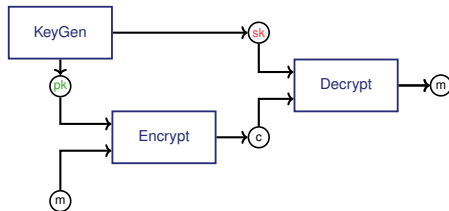


- Allows to encrypt any message
- Typically secure against Chosen Plaintext Attacks
- Can be transformed into more secured KEM

But what is a KEM?

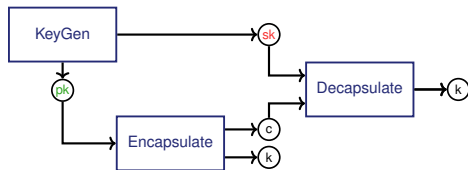
- Goal: Securely share a symmetric key over an insecure channel

PKE:



- Allows to encrypt any message
- Typically secure against Chosen Plaintext Attacks
- Can be transformed into more secured KEM

KEM:



- $\approx$ Diffie-Hellman (*but with only one pk/sk*)
- Typically secure against Chosen Ciphertext Attacks
- Can be transformed into a CCA secure PKE

ML-KEM

- CPA-secure PKE scheme, hardness based on M-LWE
- Three security levels: ML-KEM-512 (AES-128 🗑️), ML-KEM-768 (AES-192 🗑️), ML-KEM-1024 (AES-256 🗑️)
- Not approved to use as a stand-alone by NIST

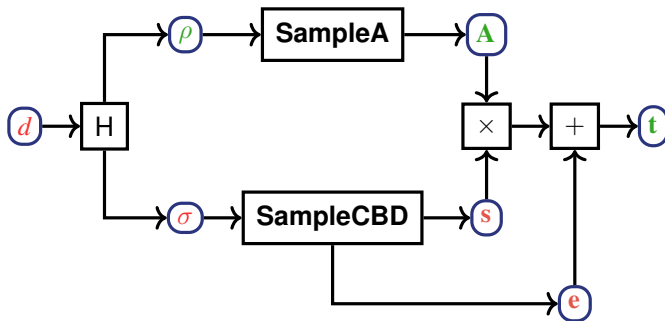
ML-KEM

- CPA-secure PKE scheme, hardness based on M-LWE
- Three security levels: ML-KEM-512 (AES-128 🗝️), ML-KEM-768 (AES-192 🗝️), ML-KEM-1024 (AES-256 🗝️)
- Not approved to use as a stand-alone by NIST
- Using Fujisaki-Okamoto transform we get a CCA-secure KEM
- Approved by NIST under the name ML-KEM512, ML-KEM768, and ML-KEM1024 [1]

ML-KEM

- CPA-secure PKE scheme, hardness based on M-LWE
- Three security levels: ML-KEM-512 (AES-128 🗝️), ML-KEM-768 (AES-192 🗝️), ML-KEM-1024 (AES-256 🗝️)
- Not approved to use as a stand-alone by NIST
- Using Fujisaki-Okamoto transform we get a CCA-secure KEM
- Approved by NIST under the name ML-KEM512, ML-KEM768, and ML-KEM1024 [1]
- Quotient Ring $\mathbb{Z}_q[x]/(x^n + 1)$ where $n = 2^8 = 256$ and $q = 13 \times 2^8 + 1 = 3329$
 - Offers more “bandwidth”, as we can encrypt $n = 256$ bits at a time
 - Easy “scalability” in security (only increase k)
 - Noise sampled from a centered Binomial distribution ($\{2, 3\}$, limits the error growth)
 - Polynomial multiplication with non-fully splitting NTT (for performance)
 - Ciphertext Compression (for performance)

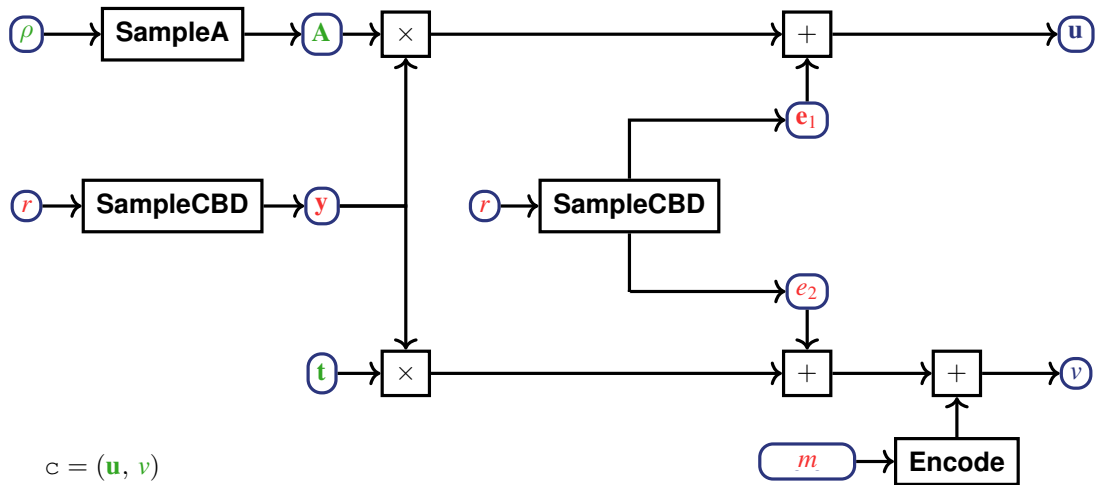
KeyGen(d):

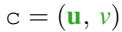


$$\text{pk} = (\mathbf{t}, \rho)$$

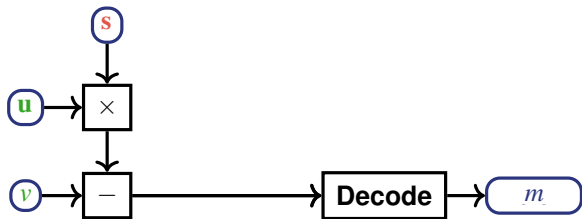
$$\text{sk} = (\mathbf{s})$$

Encrypt(m , $pk=(t, \rho)$, r):

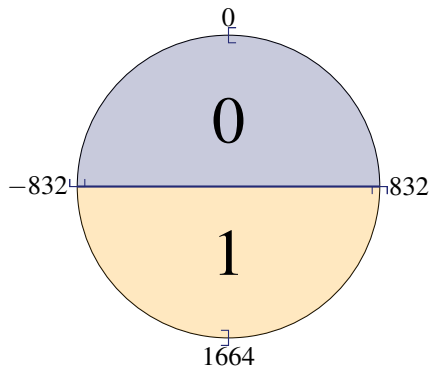
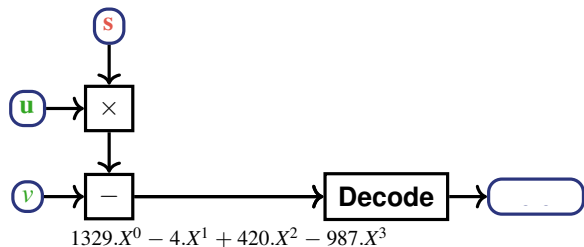




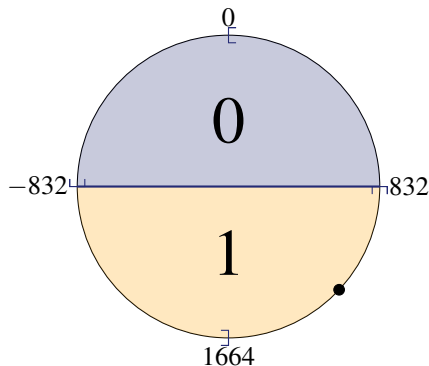
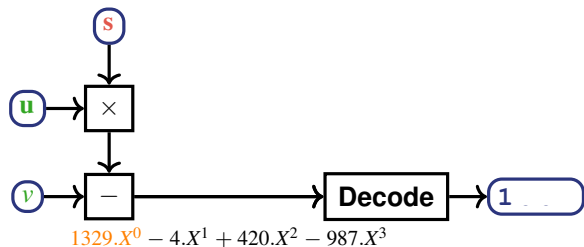
Decrypt($c = (u, v)$, $sk = (s)$):



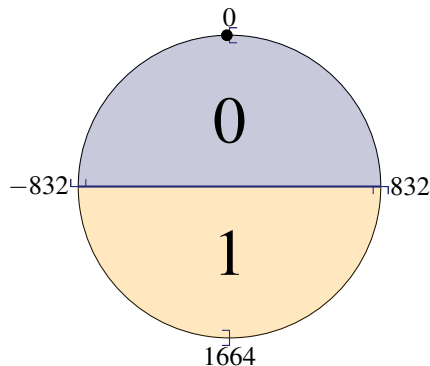
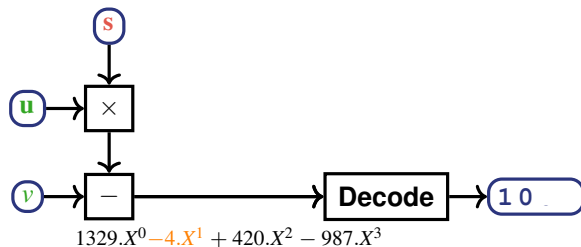
Decrypt($c = (u, v)$, $sk = (s)$):



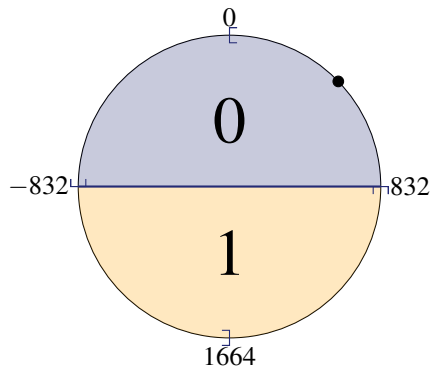
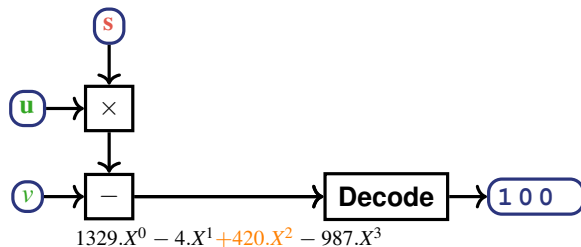
Decrypt($c = (u, v)$, $sk = (s)$):



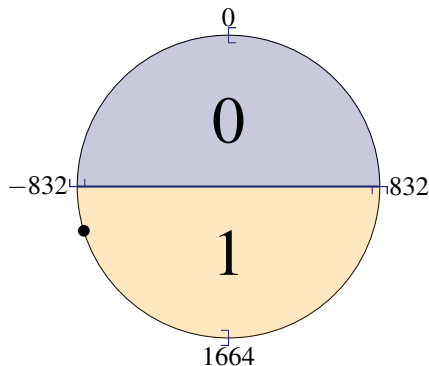
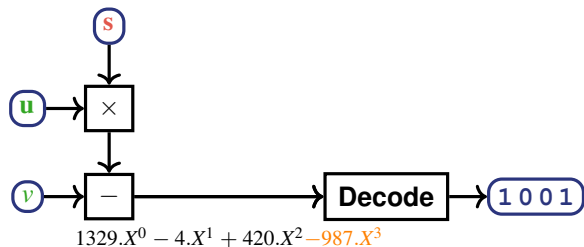
Decrypt($c = (u, v)$, $sk = (s)$):



Decrypt($c = (u, v)$, $sk = (s)$):

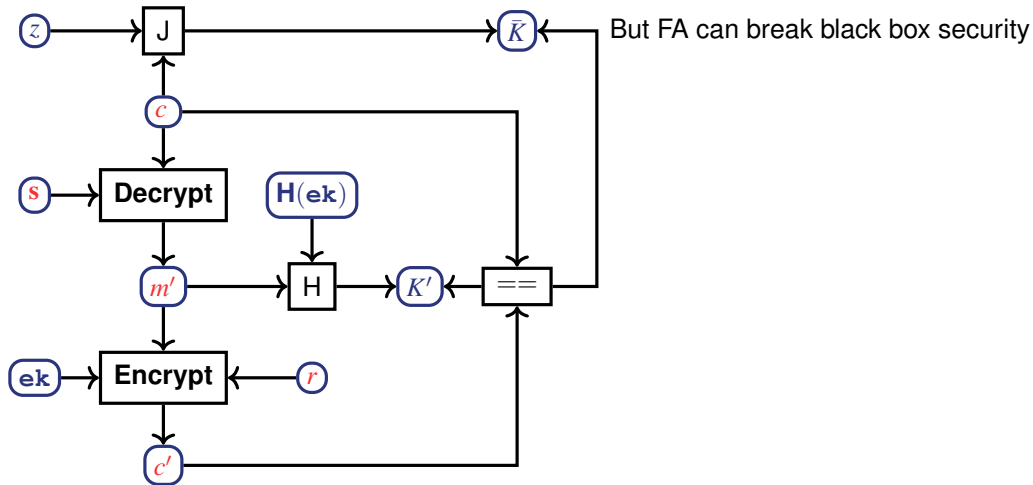


Decrypt($c = (u, v)$, $sk = (s)$):



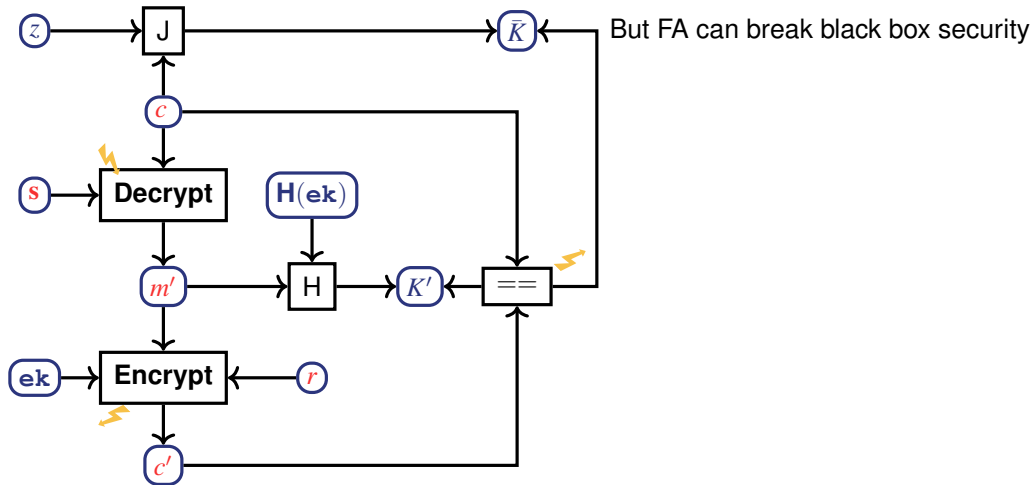
Decaps($\text{dk} = (\text{s}, \text{ek}, \text{H}(\text{ek}), \text{z}), c = (\text{u}, \text{v})$):

- Verify the correct decryption by re-encrypting the message and comparing ciphertexts



Decaps($\text{dk} = (\text{s}, \text{ek}, \text{H}(\text{ek}), \text{z}), c = (\text{u}, \text{v})$):

- Verify the correct decryption by re-encrypting the message and comparing ciphertexts



ML-KEM FA summary (until april 2025)

Attack	Value targeted	References
Skipping addition	$As + e, {}^t t y + e_2 + m$	Bindel et al., 2016; Krahmer et al., 2024; Ravi, Jhanwar, et al., 2019
NTT conversion	s, c, y	Ravi et al., 2023
Nonce reuse	s and e , y and e_1	Ravi, Roy, et al., 2019
Skipping ciphertext comparison	$c == c'$	Xagawa et al., 2021
Decryption failure attack	m' decoding	Pessl and Prokop, 2021
Fault correction attack	m'	Delvaux, 2022; Hermelink et al., 2021

Conclusion

- State of the art:
 - A lot of attack paths on both ML-KEM, ML-DSA, and in common
 - Not a lot of papers about countermeasures
 - Hard to secure against all of them, or rather easy . . . mask everything :)
- What's left to do:
 - New attack vectors?
 - Improvements on previous attacks?
 - New attacks against hardened/secure implementations?
 - New countermeasures that jointly protect against FA and SCA?

Thank you!

Bibliography I

-  Azevedo-Oliveira, P., Calle Viera, A., Cogliati, B., & Goubin, L. (2025). Finding a polytope: A practical fault attack against dilithium. In T. Jager & J. Pan (Eds.), *Public-key cryptography – pkc 2025* (pp. 259–283). Springer Nature Switzerland.
-  Bauer, S., & De Santis, F. (2023). Forging dilithium and falcon signatures by single fault injection. *2023 Workshop on Fault Detection and Tolerance in Cryptography (FDTC)*, 81–88. <https://doi.org/10.1109/FDTC60478.2023.00017>
-  Bindel, N., Buchmann, J., & Krämer, J. (2016). Lattice-based signature schemes and their sensitivity to fault attacks. *2016 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2016, Santa Barbara, CA, USA, August 16, 2016*, 63–77. <https://doi.org/10.1109/FDTC.2016.11>
-  Bruinderink, L. G., & Pessl, P. (2018). Differential fault attacks on deterministic lattice signatures. *2018(3)*, 21–43. <https://doi.org/10.13154/tches.v2018.i3.21-43>
-  Calle Viera, A., Berzati, A., & Heydemann, K. (2024). Fault attacks sensitivity of public parameters in the dilithium verification. In S. Bhasin & T. Roche (Eds.), *Smart card research and advanced applications* (pp. 62–83). Springer Nature Switzerland.

OPEN

Bibliography II

-  Delvaux, J. (2022). Roulette: A diverse family of feasible fault attacks on masked Kyber. 2022(4), 637–660. <https://doi.org/10.46586/tches.v2022.i4.637-660>
-  Espitau, T., Fouque, P.-A., Gérard, B., & Tibouchi, M. (n.d.). Loop-abort faults on lattice-based Fiat-Shamir and hash-and-sign signatures. https://doi.org/10.1007/978-3-319-69453-5_8
-  Hermelink, J., Pessl, P., & Pöppelmann, T. (2021). Fault-enabled chosen-ciphertext attacks on kyber. 311–334. https://doi.org/10.1007/978-3-030-92518-5%5C_15
-  Islam, S., Mus, K., Singh, R., Schaumont, P., & Sunar, B. (2022). Signature correction attack on dilithium signature scheme. <https://doi.org/10.1109/EuroSP53844.2022.00046>
-  Jendral, S. (2024). A single trace fault injection attack on hedged CRYSTALS-dilithium. <https://eprint.iacr.org/2024/238>
-  Krahmer, E., Pessl, P., Land, G., & Güneysu, T. (2024). Correction fault attacks on randomized CRYSTALS-dilithium. <https://eprint.iacr.org/2024/138>

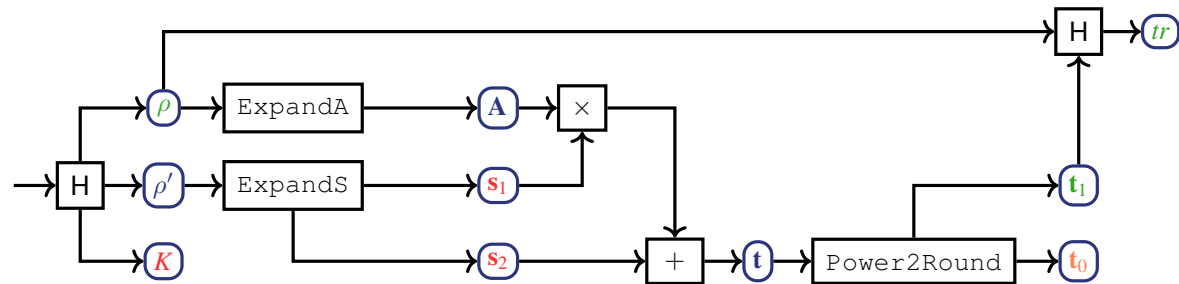
Bibliography III

-  Pessl, P., & Prokop, L. (2021). Fault attacks on CCA-secure lattice KEMs. *2021(2)*, 37–60. <https://doi.org/10.46586/tches.v2021.i2.37-60>
-  Ravi, P., Jhanwar, M. P., Howe, J., Chattopadhyay, A., & Bhasin, S. (2019). Exploiting determinism in lattice-based signatures: Practical fault attacks on pqm4 implementations of NIST candidates. <https://doi.org/10.1145/3321705.3329821>
-  Ravi, P., Roy, D. B., Bhasin, S., Chattopadhyay, A., & Mukhopadhyay, D. (2019). Number “not used” once - practical fault attack on pqm4 implementations of NIST candidates. https://doi.org/10.1007/978-3-030-16350-1_13
-  Ravi, P., Yang, B., Bhasin, S., Zhang, F., & Chattopadhyay, A. (2023). Fiddling the twiddle constants - fault injection analysis of the number theoretic transform. *2023(2)*, 447–481. <https://doi.org/10.46586/tches.v2023.i2.447-481>
-  Ulitzsch, V. Q., Marzougui, S., Bagia, A., Tibouchi, M., & Seifert, J.-P. (2023). Loop aborts strike back: Defeating fault countermeasures in lattice signatures with ILP. *2023(4)*, 367–392. <https://doi.org/10.46586/tches.v2023.i4.367-392>

Bibliography IV

-  Wang, Y., Yu, J., Qu, S., Zhang, X., Li, X., Zhang, C., & Gu, D. (2024). Beware of keccak: Practical fault attacks on SHA-3 to compromise kyber and dilithium on ARM cortex-m devices. <https://eprint.iacr.org/2024/1522>
-  Xagawa, K., Ito, A., Ueno, R., Takahashi, J., & Homma, N. (2021). Fault-injection attacks against NIST's post-quantum cryptography round 3 KEM candidates. https://doi.org/10.1007/978-3-030-92075-3_2

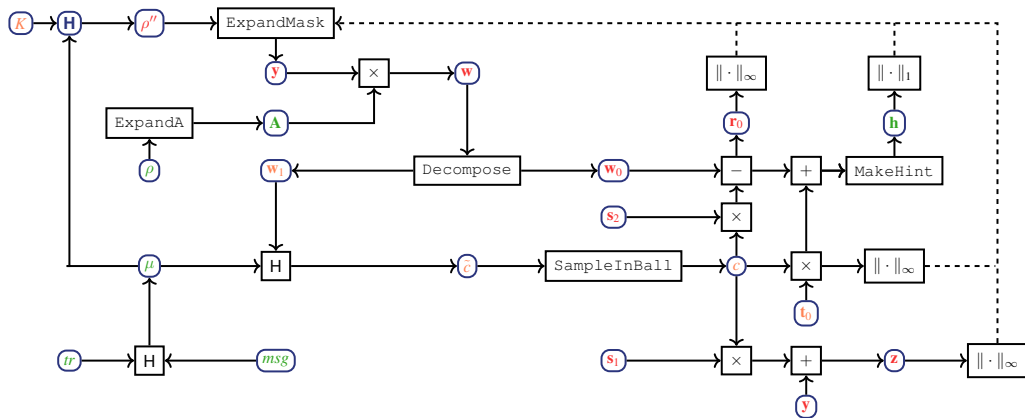
KeyGen



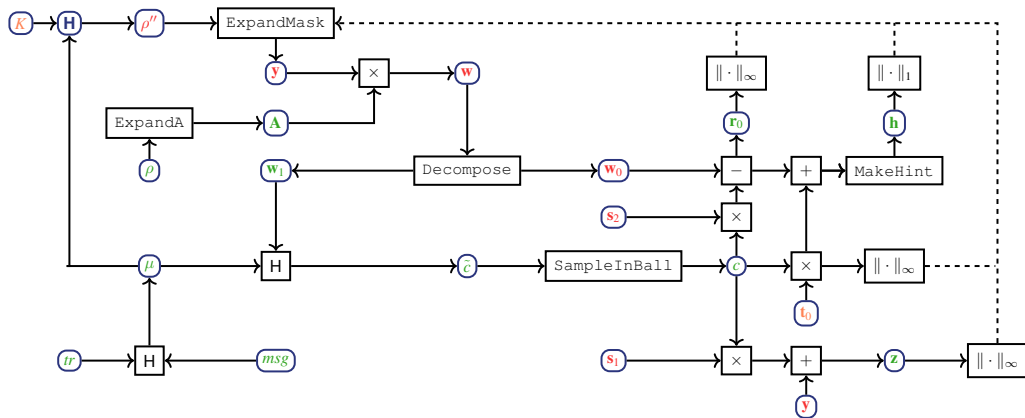
$$\text{pk} = (\rho, \mathbf{t}_1)$$

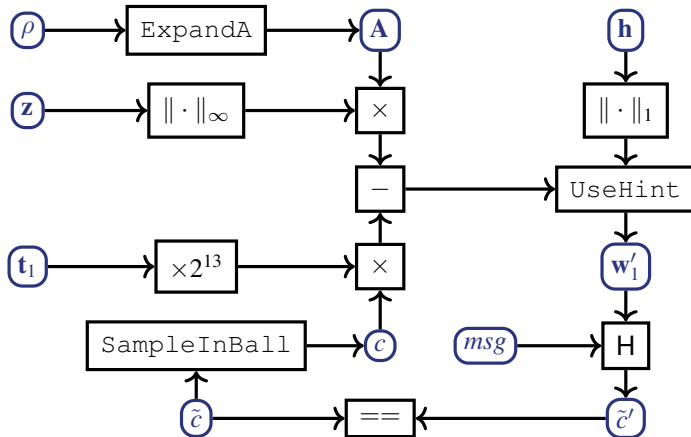
$$\text{sk} = (\rho, K, \mathbf{H}(\text{pk}), \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0)$$

Sign(msg , $sk = (\rho, K, tr, s_1, s_2, t_0)$):



Sign(msg , $sk = (\rho, K, tr, s_1, s_2, t_0)$):





ML-KEM algorithms (simplified)

KeyGen():

1. $\mathbf{A} \in \mathcal{R}_q^{k \times K} = \text{ExpandA}(\rho)$
2. $\mathbf{s}, \mathbf{e} \in \mathcal{B}(\eta)^k \times \mathcal{B}(\eta)^k$
3. $\mathbf{t} = \mathbf{A}\mathbf{s} + \mathbf{e}$
4. return $\text{pk} = (\mathbf{t}, \rho)$, $\text{sk} = (\mathbf{s})$

Encrypt ($m, \text{pk} = (\mathbf{t}, \rho), r$):

1. $\mathbf{A} \in \mathcal{R}_q^{k \times k} = \text{ExpandA}(\rho)$
2. $\mathbf{r} \in \mathcal{B}(\eta)^k$
3. $\mathbf{e}_1 \in \mathcal{B}(\eta)^k$
4. $\mathbf{e}_2 \in \mathcal{B}(\eta)$
5. $\mathbf{u} = \mathbf{A}^T \mathbf{r} + \mathbf{e}_1$
6. $v = \mathbf{t}^T \mathbf{r} + \mathbf{e}_2 + m$
7. return $c = (\text{Compress}_q(\mathbf{u}, d_u), \text{Compress}_q(v, d_v))$

Decrypt ($\text{sk} = (\mathbf{s}), c$):

1. return $m = v - \mathbf{s}^T \mathbf{u}$

KeyGen():

1. $z \in \{0, 1\}^{32}$
2. $(\text{pk}', \text{sk}') = \text{KeyGen}()$
3. return $\text{pk} = (\text{pk}')$, $\text{sk} = (\text{sk}', \text{pk}, \text{H}(\text{pk}), z)$

Encapsulate (pk):

1. $m \in \{0, 1\}^{32}$
2. $(K, r) = \text{G}(m \parallel \text{H}(\text{pk}))$
3. $c = \text{Encrypt}(m, \text{pk}, r)$
4. return (K, c)

Decapsulate (c, sk):

1. $m' = \text{Decrypt}(\text{sk}', c)$
2. $(K', r') = \text{G}(m' \parallel \text{H}(\text{pk}))$
3. $c' = \text{Encrypt}(m', \text{pk}, r')$
4. if $c \neq c'$:
5. return $\text{J}(z \parallel c)$
6. return K'