RÉPUBLIQUE
FRANÇAISE
Liberté
Égalité
Fraternité

Silithium – A Compact, Efficient and Non-separable Hybrid Signature

Julien Devevey, Morgane Guerreau and Maxime Roméas
ANSSI, PQShield and ANSSI

Wednesday, July 22 2026

Overview

Silithium is a hybrid signature with

- hybrid security – forging requires breaking ML-DSA and discrete logarithm,
- strong unforgeability as high as ML-DSA,
- lower signing time and smaller signature size than composite,
- backward compatibility – using Silithium and ECDSA at the same time does not weaken security,
- but using Silithium and ML-DSA at the same time has the same limitations as composite.

Overview

What is the current status of hybrid signatures, and why yet another one?

Property	Silithium	Composite	Mothma	SUFHybSig
Hybrid unforgeability	PQ+T	PQ+T	PQ+T	PQ+T
Strong unforgeability	PQ	None	PQ	PQ+T
Backward compatibility	Yes	No	Maybe	Yes
“Interoperability”	Maybe	No	Maybe	No
Signing time	Lower	Baseline	Higher	Higher
Parallelizable	No	Yes	No	No
Signature size	Smaller	Baseline	Longer	Smaller

(To be fair, signing time and signature size discrepancies are a couple percents, at most)

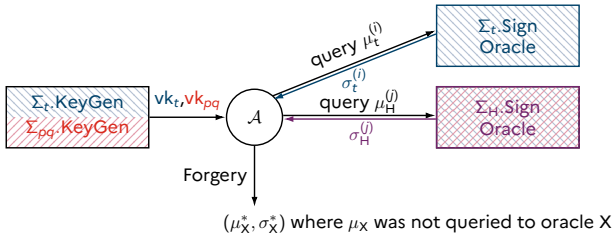
Overview

What is the current status of hybrid signatures, and why yet another one?

Property	Silithium	Composite	Mothma	SUFHybSig
Hybrid unforgeability	PQ+T	PQ+T	PQ+T	PQ+T
Strong unforgeability	PQ	None	PQ	PQ+T
Backward compatibility	Yes	No	Maybe	Yes
"Interoperability	Maybe	No	Maybe	No
Signing time	Lower	Baseline	Higher	Higher
Parallelizable	No	Yes	No	No
Signature size	Smaller	Baseline	Longer	Smaller

(To be fair, signing time and signature size discrepancies are a couple percents, at most)

Backward Compatibility



- Stronger than non-separability: forgery can call the other oracle on μ_X
- Could swap the Σ_t oracle with (or add) a Σ_{pq} oracle
- Implies that concurrent usage of Σ_t and Σ_H does not weaken security

1. Construction

Original Proposal (PQCrypto'26)

KeyGen(1^λ):

1. $(\mathsf{sk}_1, \mathsf{vk}_1) \leftarrow$ ECDSA.KeyGen
2. $(\mathsf{sk}_2, \mathsf{vk}_2) \leftarrow$ ML-DSA.KeyGen(1^λ)

Sign(sk_H, M):

1. $r \leftarrow U([0, p-1])$
2. $w_1 \leftarrow [r]G$
3. $\sigma \leftarrow$ ML-DSA.Sign($\mathsf{sk}_2, w_1 \| M$)
4. $(c, z_2, h_2) \leftarrow \sigma$
5. $z_1 \leftarrow r + c\mathsf{sk}_1 \bmod p$
6. Return (c, z_1, z_2, h_2)

Verify($\mathsf{vk}_H, M, (c, z_1, z_2, h)$):

Original Proposal (PQCrypto'26)

KeyGen(1^λ):

1. $(\mathsf{sk}_1, \mathsf{vk}_1) \leftarrow$ ECDSA.KeyGen
2. $(\mathsf{sk}_2, \mathsf{vk}_2) \leftarrow$ ML-DSA.KeyGen(1^λ)
3. $tr' = \mathcal{H}(\mathsf{vk}_1, \mathsf{vk}_2)$ and update sk_2
4. Return $\mathsf{vk}_H = (\mathsf{vk}_1, \mathsf{vk}_2)$, $\mathsf{sk}_H = (\mathsf{sk}_1, \mathsf{sk}_2)$

Verify($\mathsf{vk}_H, M, (c, z_1, z_2, h)$):

1. $w_1 \leftarrow [z_1]G + [-c]\mathsf{vk}_1$
2. $\mu \leftarrow \mathcal{H}(\mathcal{H}(\mathsf{vk}_H), w_1, M)$
3. $\sigma \leftarrow (c, z_2, h_2)$
3. Return ML-DSA.Verify($\mathsf{vk}_2, \sigma, \mu$)

Sign(sk_H, M):

1. $r \leftarrow U([0, p-1])$
2. $w_1 \leftarrow [r]G$
3. $\sigma \leftarrow$ ML-DSA.Sign($\mathsf{sk}_2, w_1 \| M$)
4. $(c, z_2, h_2) \leftarrow \sigma$
5. $z_1 \leftarrow r + c\mathsf{sk}_1 \bmod p$
6. Return (c, z_1, z_2, h_2)

Original Proposal (PQCrypto'26)

KeyGen(1^λ):

1. $(\mathsf{sk}_1, \mathsf{vk}_1) \leftarrow$ ECDSA.KeyGen
2. $(\mathsf{sk}_2, \mathsf{vk}_2) \leftarrow$ ML-DSA.KeyGen(1^λ)
3. $tr' = \mathcal{H}(\mathsf{vk}_1, \mathsf{vk}_2)$ and update sk_2
4. Return $\mathsf{vk}_H = (\mathsf{vk}_1, \mathsf{vk}_2)$, $\mathsf{sk}_H = (\mathsf{sk}_1, \mathsf{sk}_2)$

Verify($\mathsf{vk}_H, M, (c, z_1, z_2, h)$):

1. $w_1 \leftarrow [z_1]G + [-c]\mathsf{vk}_1$
2. $\mu \leftarrow \mathcal{H}(\mathcal{H}(\mathsf{vk}_H), w_1, M)$
3. $\sigma \leftarrow (c, z_2, h_2)$
3. Return ML-DSA.Verify($\mathsf{vk}_2, \sigma, \mu$)

Sign(sk_H, M):

1. $r \leftarrow U([0, p-1])$
2. $w_1 \leftarrow [r]G$
3. $\sigma \leftarrow$ ML-DSA.Sign($\mathsf{sk}_2, w_1 \| M$)
4. $(c, z_2, h_2) \leftarrow \sigma$
5. $z_1 \leftarrow r + c\mathsf{sk}_1 \bmod p$
6. Return (c, z_1, z_2, h_2)

Steps 1 and 2 may be replaced with $(r, w_1) \leftarrow$ ECDSA.KeyGen.

Scheme in the Draft

Sign(sk_H, M):

1. $r \leftarrow U([0, p-1])$
2. $w_1 \leftarrow [r]G$
3. $(c, z_2, h) \leftarrow \text{ML-DSA.Sign}(\mathrm{sk}_2, M, ctx = (w_1, \mathrm{vk}_1))$
4. $z_1 \leftarrow r + c\mathrm{sk}_1 \bmod p$
5. Return (c, z_1, z_2, h_2)

Verify($\mathrm{vk}_H, M, (c, z_1, z_2, h)$):

1. $w_1 \leftarrow [z_1]G + [-c]\mathrm{vk}_1$
2. Return $\text{ML-DSA.Verify}(\mathrm{vk}_2, (c, z_2, h_2), M, ctx = (w_1, \mathrm{vk}_1))$

Pros and Cons

Current proposal vs original one

+ No assumption on the available implementation of ML-DSA
+ Retains the smaller size + faster computation time
- Loses the ability to be used simultaneously with ML-DSA

Silithium vs fully black-box hybridization

- ECC in Silithium is not black-box
+ All ECC operations must already be implemented for ECDSA

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

2. Security

Unforgeability and strong unforgeability

Unforgeability attack path:

- Pick a nonce w_1
- Break ML-DSA to forge a signature
- Break the discrete logarithm problem with input $w_1 \mathrm{vk}_1^{\,c}$ to compute z_1

Requires breaking ML-DSA AND DL

Strong unfrogeability attack path:

- Find a Silithium signature for any message
- Forge a new ML-DSA for this message, while keeping the same c
- No need to refresh the nonce w_1/response z_1

Requires breaking ML-DSA only

Formal security proofs exist for the original proposal and could be extended to the submitted variant

Backward compatibility

Can we safely use the $(\mathsf{sk}_1, \mathsf{vk}_1)$ *part of* $(\mathsf{sk}_H, \mathsf{vk}_H)$ *with ECDSA?*

Yes!

- ECDSA signatures does not help forge Silithium signatures, except if they help recovering the signing key sk_1
- Silithium signatures are based on zero-knowledge procotols: they are useless to recover sk_H and the framework is not related to ECDSA signatures

Interoperability

Can we safely use the $(\mathsf{sk}_2, \mathsf{vk}_2)$ *part of* $(\mathsf{sk}_H, \mathsf{vk}_H)$ *with ML-DSA?*

Not really...

- Silithium signatures are comprised of a ML-DSA signature: the scheme is separable, even if weakly non-separable
- New, unpredictable context is used each time: applicative layer could forbid user-chosen contexts to retain security

3. Potential Variants

Appending w_1 to the message

- w_1 is appended to M in the original suggestion, but moved to the context in the I-D
- This makes the context unpredictable
- Instead, if w_1 is appended to M, the message becomes unpredictable, while the context is now fixed
- Neither fixes the seprability issue, but which ones makes the strongest “weakly non-separable” Silithium?

Adding a Context String

- Currently, Silithium offers no context string
- However, it only uses up 66 out of the 255 bytes allowed for ML-DSA's context string
- 123 bytes remains and could be used. If longer strings are desirable, the previous trick frees 66 additional bytes

Moving back to the original proposal

- The trade-off would be reversed
- Attacks no easier, even when Silithium, ECDSA and ML-DSA are used together
- Implementation becomes messier: to sign, one needs either ML-DSA with external-μ or to modify sk_2 and to verify, one needs ML-DSA with external-μ