RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

Silithium – A Compact, Efficient and Non-separable Hybrid Signature (draft-devevey-cfrg-silithium-00)

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

Wednesday, July 22 2026

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	Cond.	Yes
“Interoperability”	Cond.	No	Cond.	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	Cond.	Yes
“Interoperability”	Cond.	No	Cond.	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

We define it as the situation where

- a user relies on a hybrid signature scheme,
- but also uses the ECC component of the hybrid key to sign with an ECC algorithm (ECDSA/EdDSA).

The goal is to ensure that in this setting, both the hybrid and ECC schemes remain as secure as if they were used alone.

For instance, composite is not backward compatible as one can extract an ECC signature from the hybrid scheme.

1. Construction

Silithium

Keygen: $(\mathsf{vk}_1, \mathsf{sk}_1) \leftarrow$ ECDSA.KeyGen, $(\mathsf{vk}_2, \mathsf{sk}_2) \leftarrow$ ML-DSA.KeyGen.
Return $\mathsf{vk}_H = (\mathsf{vk}_1, \mathsf{vk}_2)$ and $\mathsf{sk}_H = (\mathsf{sk}_1, \mathsf{sk}_2)$.

$\mathsf{Sign}(\mathsf{sk}_H, M)$:

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

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

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

Concrete Instances

We propose the following instances in the draft:

ML-DSA44+P256 ML-DSA65+P384 ML-DSA87+P521

Concrete Instances

We propose the following instances in the draft:

ML-DSA44+P256 ML-DSA65+P384 ML-DSA87+P521

Question from the mailing list

What about IETF Curve25519 and Curve448?

There is no issue using these curves! In fact, we can even generate the nonce w_1 similarly to EdDSA, giving rise to Edilithium.

Would you prefer a Curve25519 variant with

- SHA-512 deterministic nonce,
- SHA3 deterministic nonce,
- random nonce?

Non-black Box ECC

Remark from the CFRG mailing list

The ECC crypto used in the scheme is non-black box.

Yes, but...

- all operations used in Silithium are used in ECDSA/EdDSA: ECC operations, big integer arithmetic,
- we can expect those operations to be available when implementing Silithium.

In practice, implementing the scheme in Rust with RustCrypto takes < 150 lines of code.

RÉPUBLIQUE FRANÇAISE
Liberté
Égalité
Fraternité

2. Security

Backward compatibility

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

Yes!

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

We should, but be careful:

- Silithium signatures are comprised of a ML-DSA signature: the scheme may be separable, but has unpredictable context
- As long as the application does not allow user-submitted contexts during verification, we are in the clear

3. Fine-tuning

Different “Interoperability” Security Trade-offs

Is any of the following two options more attractive than the current proposal?

1. Append w_1 to M instead of putting it in the context
   - The message becomes unpredictable, while the context is now fixed: is it easier to filter out extracted ML-DSA signatures?

2. Move back to the original PQCrypto proposal
   - \+ Full “H-EU-CMA” security
   - \- Requires specific implementations of ML-DSA

Adding a Context String

Is a context string needed? How long does it need to be?

- Silithium uses up 132 out of the 255 bytes allowed for ML-DSA's context string
- 123 bytes remains and could be used as Silithium's context
- If longer is needed: append it to the message

Moving Forward

We are looking for feedback on the following questions:

1. Should we move w_1 in the message instead of the context or even go back to the PQCrypto proposal?
2. Does Silithium need its own (long) context strings?
3. How many and which parameter sets should we put in the draft?
4. Is there an interest in Edilithium/Curve25519?

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)