Kohaku Hardware Signer: First Post-Quantum Secure Element Implementation

Simon Masson, Renaud Dubois
ZKNox

Account Abstraction Hub

November 19th, 2025 – Buenos Aires, DevConnect

ZKNOX team

Nicolas Bacca
20^+ years experience (10^+y web3)
Security and hardware specialist
Prev. Ledger cofounder/CTO

Renaud Dubois
20^+ years experience (3^+y web3)
Cryptographer
Prev. Ledger, Thales

Simon Masson
8^+ years experience (4^+y web3)
Cryptographer
Prev. Heliax, Thales

ZKNOX team

Nicolas Bacca
20^+ years experience (10^+y web3)
Security and hardware specialist
Prev. Ledger cofounder/CTO

Renaud Dubois
20^+ years experience (3^+y web3)
Cryptographer
Prev. Ledger, Thales

Simon Masson
8^+ years experience (4^+y web3)
Cryptographer
Prev. Heliax, Thales

Expertise and innovation to every challenge on the whole security chain:

- user end
  (secure enclaves, hardware wallets),
- back end
  (TEE, HSMs),
- on-chain
  (smart contracts).

ZKNOX team

Nicolas Bacca
20^+ years experience (10^+y web3)
Security and hardware specialist
Prev. Ledger cofounder/CTO

Renaud Dubois
20^+ years experience (3^+y web3)
Cryptographer
Prev. Ledger, Thales

Simon Masson
8^+ years experience (4^+y web3)
Cryptographer
Prev. Heliax, Thales

Expertise and innovation to every challenge on the whole security chain:

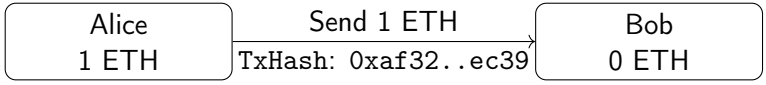
- user end
  (secure enclaves,
  hardware wallets),
- back end
  (TEE, HSMs),
- on-chain
  (smart contracts).

`https://zknox.eth.limo/`

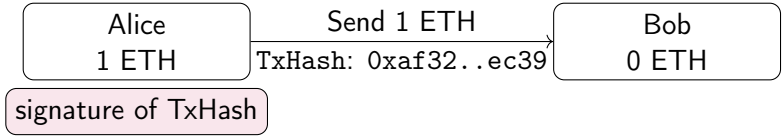
`https://github.com/zknoxhq/`

Account Abstraction

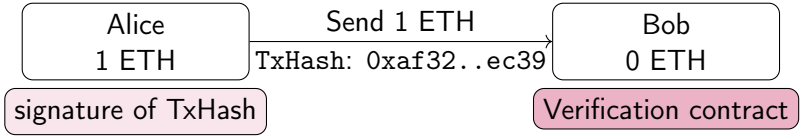
Account Abstraction



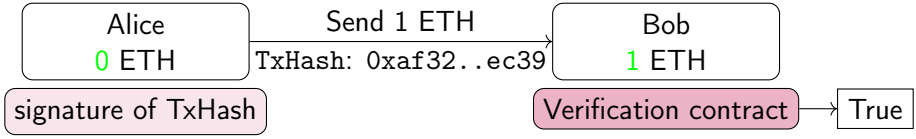
Account Abstraction



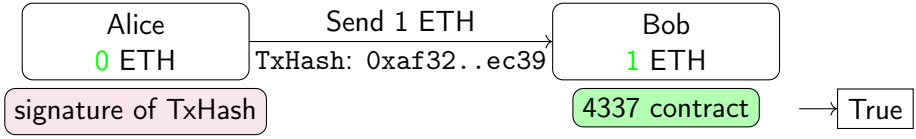
Account Abstraction



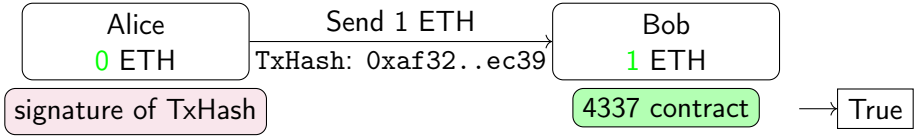
Account Abstraction



Account Abstraction

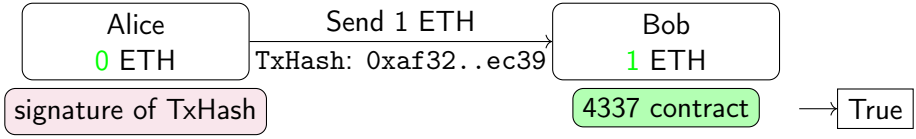


Account Abstraction



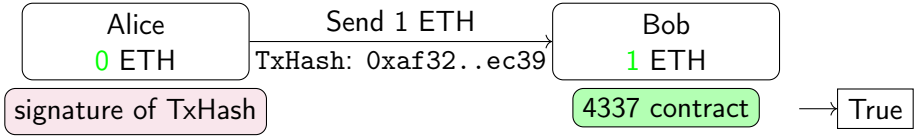
Execution layer: ECDSA signature (not resistant to quantum attacks)

Account Abstraction



Execution layer: ECDSA signature (not resistant to quantum attacks)
Account abstraction (4337): enables custom signature.

Account Abstraction



Execution layer: ECDSA signature (not resistant to quantum attacks)
Account abstraction (4337): enables custom signature.
This work:

- smart account with hybrid ECDSA+MLDSA verification,
- same user experience, but hardware and post-quantum security.

Post-quantum signature candidates

Standard	**FN-DSA**	**ML-DSA**	**SLH-DSA**
Name	Falcon	Dilithium	Sphincs
FIPS	203	204	205
Family	Lattice	Lattice	Hash
Signature			
Public key			
Signer			
Verifier			

Post-quantum signature candidates

Standard	**FN-DSA**	**ML-DSA**	**SLH-DSA**
Name	Falcon	Dilithium	Sphincs
FIPS	203	204	205
Family	Lattice	Lattice	Hash
Signature	666B	2420B	7856B
Public key	897B	1312B	32B
Signer			
Verifier			

Post-quantum signature candidates

Standard	**FN-DSA**	**ML-DSA**	**SLH-DSA**
Name	Falcon	Dilithium	Sphincs
FIPS	203	204	205
Family	Lattice	Lattice	Hash
Signature	666B	2420B	7856B
Public key	897B	1312B	32B
Signer	floating point, high RAM	Simple	Very slow
Verifier	separation hash-core, efficient	Simple	Very slow

Post-quantum signature candidates

Standard	**FN-DSA**	**ML-DSA**	**SLH-DSA**
Name	Falcon	Dilithium	Sphincs
FIPS	203	204	205
Family	Lattice	Lattice	Hash
Signature	666B	2420B	7856B
Public key	897B	1312B	32B
Signer	floating point, high RAM	Simple	Very slow
Verifier	separation hash-core, efficient	Simple	Very slow

We implement ML-DSA (Dilithium) following EIP 8051

EIP 8051

Two precompiles:

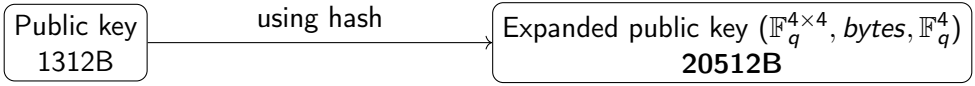
- ML-DSA-NIST: using the hash function SHAKE256 (compliant to NIST)
- ML-DSA-ETH: using the hash function KECCAK256 (efficient in Ethereum contracts)

EIP 8051

Two precompiles:

- ML-DSA-NIST: using the hash function SHAKE256 (compliant to NIST)
- ML-DSA-ETH: using the hash function KECCAK256 (efficient in Ethereum contracts)

Public key is expanded during verification:

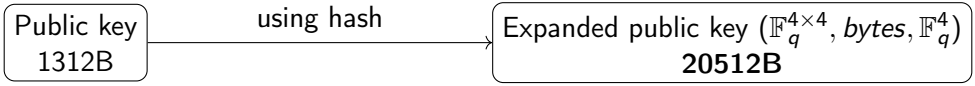


EIP 8051

Two precompiles:

- ML-DSA-NIST: using the hash function SHAKE256 (compliant to NIST)
- ML-DSA-ETH: using the hash function KECCAK256 (efficient in Ethereum contracts)

Public key is expanded during verification:



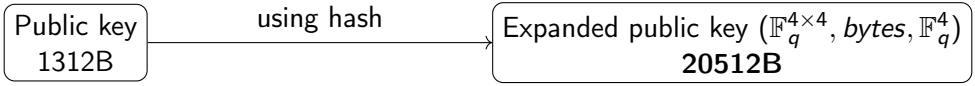
Public key provided in expanded form to save hashes.

EIP 8051

Two precompiles:

- ML-DSA-NIST: using the hash function SHAKE256 (compliant to NIST)
- ML-DSA-ETH: using the hash function KECCAK256 (efficient in Ethereum contracts)

Public key is expanded during verification:



Public key provided in expanded form to save hashes.
Verifier cost: **5** calls to hash (SHAKE256 or KECCAK256) and **9** NTTs.

Hardware implementation

- Start from NIST FIPS 204 reference code

pq-crystals / **dilithium**

Hardware implementation

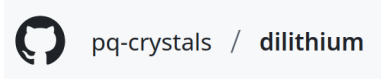
- Start from NIST FIPS 204 reference code

pq-crystals / **dilithium**

- Adaptations for low ram constraints:

Hardware implementation

- Start from NIST FIPS 204 reference code

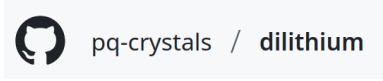


- Adaptations for low ram constraints:

- Ledger RAM $\leq$ 50kB $\Rightarrow$ implem. as in ia.cr/2022/323 (thanks to `dop-amin`)

Hardware implementation

- Start from NIST FIPS 204 reference code

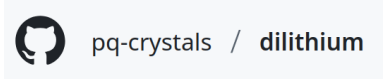


- Adaptations for low ram constraints:

- Ledger RAM $\leq$ 50kB $\Rightarrow$ implem. as in ia.cr/2022/323 (thanks to `dop-amin`)
- Entire computation in the Secure Element (the secret never leaves the enclave)

Hardware implementation

- Start from NIST FIPS 204 reference code

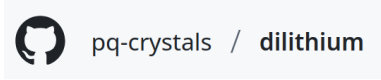


- Adaptations for low ram constraints:

- Ledger RAM $\leq$ 50kB $\Rightarrow$ implem. as in ia.cr/2022/323 (thanks to `dop-amin`)
- Entire computation in the Secure Element (the secret never leaves the enclave)
- Provide two signers for the two precompiles:

Hardware implementation

- Start from NIST FIPS 204 reference code

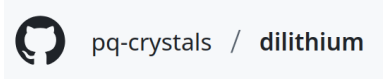


- Adaptations for low ram constraints:

- Ledger RAM $\leq$ 50kB $\Rightarrow$ implem. as in ia.cr/2022/323 (thanks to `dop-amin`)
- Entire computation in the Secure Element (the secret never leaves the enclave)
- Provide two signers for the two precompiles:
  - MLDSA (SHAKE256 using NIST code) signing in 700ms,

Hardware implementation

- Start from NIST FIPS 204 reference code

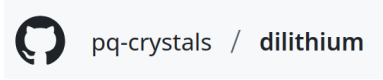


- Adaptations for low ram constraints:

- Ledger RAM $\leq$ 50kB $\Rightarrow$ implem. as in ia.cr/2022/323 (thanks to `dop-amin`)
- Entire computation in the Secure Element (the secret never leaves the enclave)
- Provide two signers for the two precompiles:
  - MLDSA (SHAKE256 using NIST code) signing in 700ms,
  - MLDSA-ETH (Keccak256 BOLOS calls) signing slower (BOLOS communication).

Hardware implementation

- Start from NIST FIPS 204 reference code

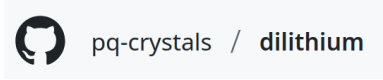


- Adaptations for low ram constraints:

- Ledger RAM $\leq$ 50kB $\Rightarrow$ implem. as in ia.cr/2022/323 (thanks to `dop-amin`)
- Entire computation in the Secure Element (the secret never leaves the enclave)
- Provide two signers for the two precompiles:
  - MLDSA (SHAKE256 using NIST code) signing in 700ms,
  - MLDSA-ETH (Keccak256 BOLOS calls) signing slower (BOLOS communication).
- Implementation tricks:

Hardware implementation

- Start from NIST FIPS 204 reference code

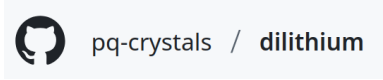


- Adaptations for low ram constraints:

- Ledger RAM $\leq$ 50kB $\Rightarrow$ implem. as in ia.cr/2022/323 (thanks to `dop-amin`)
- Entire computation in the Secure Element (the secret never leaves the enclave)
- Provide two signers for the two precompiles:
  - MLDSA (SHAKE256 using NIST code) signing in 700ms,
  - MLDSA-ETH (Keccak256 BOLOS calls) signing slower (BOLOS communication).
- Implementation tricks:
  - Recomputation of keygen to save RAM access,

Hardware implementation

- Start from NIST FIPS 204 reference code

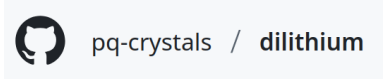


- Adaptations for low ram constraints:

- Ledger RAM $\leq$ 50kB $\Rightarrow$ implem. as in ia.cr/2022/323 (thanks to `dop-amin`)
- Entire computation in the Secure Element (the secret never leaves the enclave)
- Provide two signers for the two precompiles:
  - MLDSA (SHAKE256 using NIST code) signing in 700ms,
  - MLDSA-ETH (Keccak256 BOLOS calls) signing slower (BOLOS communication).
- Implementation tricks:
  - Recomputation of keygen to save RAM access,
  - Tweaked BIP32 to derive unique secret per hardware

Hardware implementation

- Start from NIST FIPS 204 reference code

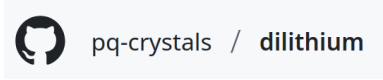


- Adaptations for low ram constraints:

- Ledger RAM $\leq$ 50kB $\Rightarrow$ implem. as in ia.cr/2022/323 (thanks to `dop-amin`)
- Entire computation in the Secure Element (the secret never leaves the enclave)
- Provide two signers for the two precompiles:
  - MLDSA (SHAKE256 using NIST code) signing in 700ms,
  - MLDSA-ETH (Keccak256 BOLOS calls) signing slower (BOLOS communication).
- Implementation tricks:
  - Recomputation of keygen to save RAM access,
  - Tweaked BIP32 to derive unique secret per hardware
  - Public key (resp. signature) fetching requires 6 APDUs (resp. 10 APDUs) to get 255B chunks

Hardware implementation

- Start from NIST FIPS 204 reference code



- Adaptations for low ram constraints:

- Ledger RAM $\leq$ 50kB $\Rightarrow$ implem. as in ia.cr/2022/323 (thanks to `dop-amin`)
- Entire computation in the Secure Element (the secret never leaves the enclave)
- Provide two signers for the two precompiles:
  - MLDSA (SHAKE256 using NIST code) signing in 700ms,
  - MLDSA-ETH (Keccak256 BOLOS calls) signing slower (BOLOS communication).
- Implementation tricks:
  - Recomputation of keygen to save RAM access,
  - Tweaked BIP32 to derive unique secret per hardware
  - Public key (resp. signature) fetching requires 6 APDUs (resp. 10 APDUs) to get 255B chunks
- Soon on other HW...

Demonstration

Let's see how it works in practice

soon in

Kohaku

Hybrid 4337

- MLDSA Expanded public key stored on-chain (20512B) per user

Hybrid 4337

- MLDSA Expanded public key stored on-chain (20512B) per user
- Hybrid contract:

Hybrid 4337

- MLDSA Expanded public key stored on-chain (20512B) per user
- Hybrid contract:
  1. Fetches the MLDSA public key

Hybrid 4337

- MLDSA Expanded public key stored on-chain (20512B) per user
- Hybrid contract:
  1. Fetches the MLDSA public key
  2. Verifies the Post-Quantum signature

Hybrid 4337

- MLDSA Expanded public key stored on-chain (20512B) per user
- Hybrid contract:
  1. Fetches the MLDSA public key
  2. Verifies the Post-Quantum signature
  3. Fetches the ECDSA public key (or Ethereum address)

Hybrid 4337

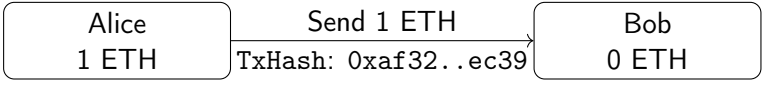
- MLDSA Expanded public key stored on-chain (20512B) per user
- Hybrid contract:
  1. Fetches the MLDSA public key
  2. Verifies the Post-Quantum signature
  3. Fetches the ECDSA public key (or Ethereum address)
  4. Verifies the ECDSA signature using `ecrecover`

Hybrid 4337

- MLDSA Expanded public key stored on-chain (20512B) per user
- Hybrid contract:
  1. Fetches the MLDSA public key
  2. Verifies the Post-Quantum signature
  3. Fetches the ECDSA public key (or Ethereum address)
  4. Verifies the ECDSA signature using `ecrecover`
  5. Accepts only if **both** signatures are valid.

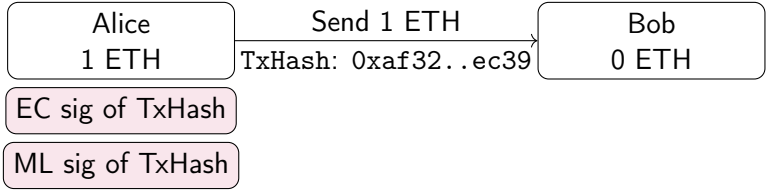
Hybrid 4337

- MLDSA Expanded public key stored on-chain (20512B) per user
- Hybrid contract:
  1. Fetches the MLDSA public key
  2. Verifies the Post-Quantum signature
  3. Fetches the ECDSA public key (or Ethereum address)
  4. Verifies the ECDSA signature using `ecrecover`
  5. Accepts only if **both** signatures are valid.



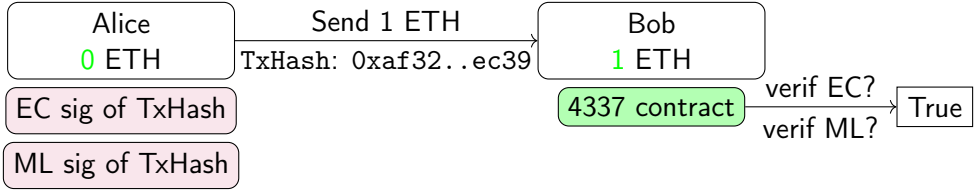
Hybrid 4337

- MLDSA Expanded public key stored on-chain (20512B) per user
- Hybrid contract:
  1. Fetches the MLDSA public key
  2. Verifies the Post-Quantum signature
  3. Fetches the ECDSA public key (or Ethereum address)
  4. Verifies the ECDSA signature using `ecrecover`
  5. Accepts only if **both** signatures are valid.



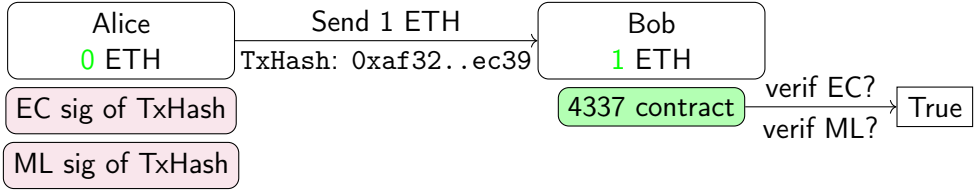
Hybrid 4337

- MLDSA Expanded public key stored on-chain (20512B) per user
- Hybrid contract:
  1. Fetches the MLDSA public key
  2. Verifies the Post-Quantum signature
  3. Fetches the ECDSA public key (or Ethereum address)
  4. Verifies the ECDSA signature using `ecrecover`
  5. Accepts only if **both** signatures are valid.



Hybrid 4337

- MLDSA Expanded public key stored on-chain (20512B) per user
- Hybrid contract:
  1. Fetches the MLDSA public key
  2. Verifies the Post-Quantum signature
  3. Fetches the ECDSA public key (or Ethereum address)
  4. Verifies the ECDSA signature using `ecrecover`
  5. Accepts only if **both** signatures are valid.



Thanks to Alex and Shahaf from 4337 team for the integration help :-)

Conclusion and perspectives

- ✓ Hardware implementation of MLDSA and MLDSAETH,

Conclusion and perspectives

- ✓ Hardware implementation of MLDSA and MLDSAETH,
- ✓ Hybrid 4337 account with ECDSA and MLDSA,

Conclusion and perspectives

- ✓ Hardware implementation of MLDSA and MLDSAETH,
- ✓ Hybrid 4337 account with ECDSA and MLDSA,
- ☐ BIP32 compliant secret derivation,

Conclusion and perspectives

- ✓ Hardware implementation of MLDSA and MLDSAETH,
- ✓ Hybrid 4337 account with ECDSA and MLDSA,
- ☐ BIP32 compliant secret derivation,
- ☐ Universal Ethereum app enables FN-DSA and ML-DSA on other HW,

Conclusion and perspectives

- ✓ Hardware implementation of MLDSA and MLDSAETH,
- ✓ Hybrid 4337 account with ECDSA and MLDSA,
- ☐ BIP32 compliant secret derivation,
- ☐ Universal Ethereum app enables FN-DSA and ML-DSA on other HW,
- ☐ More comparisons with FN-DSA,

Conclusion and perspectives

- ✓ Hardware implementation of MLDSA and MLDSAETH,
- ✓ Hybrid 4337 account with ECDSA and MLDSA,
- ☐ BIP32 compliant secret derivation,
- ☐ Universal Ethereum app enables FN-DSA and ML-DSA on other HW,
- ☐ More comparisons with FN-DSA,
- ☐ Experiments on L2 possible but gas is expensive.

Conclusion and perspectives

- ✓ Hardware implementation of MLDSA and MLDSAETH,
- ✓ Hybrid 4337 account with ECDSA and MLDSA,
- ☐ BIP32 compliant secret derivation,
- ☐ Universal Ethereum app enables FN-DSA and ML-DSA on other HW,
- ☐ More comparisons with FN-DSA,
- ☐ Experiments on L2 possible but gas is expensive.

Thank you for your attention.