Post Quantum Ethereum Era

Implementation results and foresight

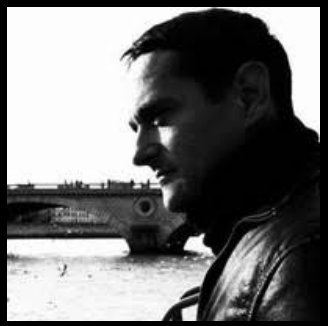
Renaud Dubois

May, 27th, 2025

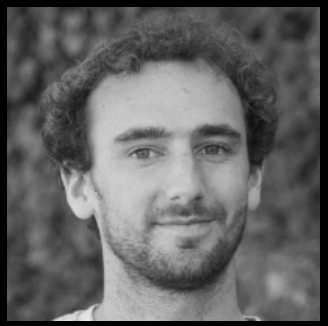
Summary

Casting

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



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



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

Missions

Bring deep expertise and innovation to every challenge on the whole security chain

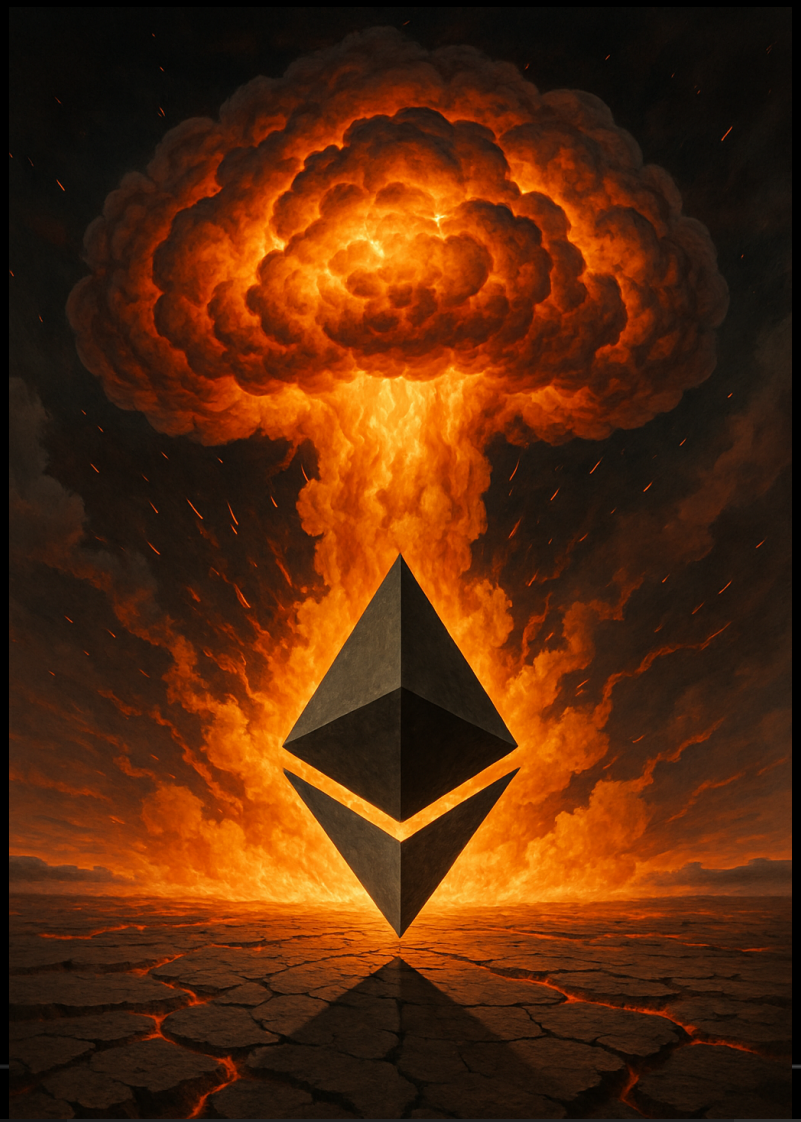
- user end (be it secure enclaves, hardware wallets)
- back end (TEE, HSMs)
- on-chain (smart contracts).

Blog

Github

What is Quantum Apocalypse ?

Shor algorithm solves factorization and discrete logarithm problems.



What is Quantum Apocalypse ?

Ethereum components at risk:

- EoA private keys (notably using ECDSA)
- BLS signatures in consensus
- Data Availability Sampling (leveraging KZG commitments)

ZKNOX roadmap is to solve the first and second points.

Candidates

IACR has our back.

Complexity Metric	Falcon-512	Dilithium2	SPHINCS+	GeMSS
Public Key Size (Bytes)	897	1312	32	33
Private Key Size (Bytes)	7553	2528	32	64
Signature Size (Bytes)	666	2420	17088	21952

Table: **NIST Level 1 Complexities of Selected PQ Signature Candidates**

FALCON appears as the more suited for onchain complexities but is known for its signer integration to be a challenge.

Pleasant feature: FALCON has an recover version, which is similar to ecrecover functionning (recover public key from signature)

Progressive Roadmap

Verifier (On chain side)

- Step1: use Account Abstraction (EIP-7702+7579/4337) with full solidity to experiment ✓
- Step2: integrate RIP into nodes
- Step3: accept as EIP
- Final: remove eoA (EIP-7701/EIP-7560)

At first, solidity enables experiments on cheap L2s.

Signer (User side)

- Step1: Hot wallet
- Step2: Hardware wallet

Verifier: Implementation Results

Function	Description	Gas Cost
`TETRATION_ethfalcon.verify`	EVM Friendly	24M
`ZKNOX_falcon.verify`	NIST	7M
`ZKNOX_ethfalcon.verify`	EVM Friendly	1.8M
`ZKNOX_epervier.verify`	Recover EVM friendly	1.9M

Table: Gas cost of ZKNOX FALCON verification functions

- In order to reduce gas cost, ETHFALCON and EPERVIER use Keccak instead of SHAKE, without compromising security.
- EPERVIER is a recover version which reduces complexity by tweaking public key representation, avoiding NTT inversion.
- Optimal NTT, fully Yuled

Verifier: Implementation Results

Function	Description	Gas Cost
TETRATION_ethdilithium.verify	EVM Friendly	40M
ZKNOX_dilithium.verify	NIST	13.5M
ZKNOX_ethdilithium.verify	EVM Friendly	6.6M

Table: Gas cost of ZKNOX DILITHIUM verification functions

Verifier: Conclusions

- FALCON x4 cheaper than DILITHIUM
- Straightforward to implement in Node, just follow NIST recommendations (see our Geth Fork)
- EVM friendly versions are preferable for step1
- NTT could be standardized rather than one or the other for more flexibility: EIP-7885
- Vectorization is very efficient to speed up NTT (EVMMAX experimentations), expected 80% gain.
- EVMMAX $\geq$ NTT, but harder to implement in Core.

Signers: constraints

Scheme	Operation	RAM Consumption (KiB)	Code Size (KiB)
Falcon	Key Generation	40-64	30-50
	Signing	40-64	30-50
Dilithium	Key Generation	≥ 7	15-25
	Signing	≥ 7	15-25

- Hot Wallet constraints are trivial.
- Hardware wallets limitations makes FALCON non trivial to integrate
- Vendors should make their homework, onchain constraints are prioritary

How to solve RAM limitation ?

Signers: limitless HW using Oblivious RAM

- Oblivious RAM extends device RAM by using host memory as an extension
- Requires confidentiality and integrity cipher of host memory
- While a long knowm mechanism, no Hardware Wallet implements it
- Should be in the firmware

Alternative is using an EVM wrapping syscalls, easier DX but slower.

Signers: Conclusions

- On chain constraints are prioritary. Ethereum shall lead the decision, not HW limitations.
- Oblivious RAM, plus other tricks can reduce dramatically implementation constraints
- Stay Tuned for more results

Proving FALCON/DILITHIUM

Need

- Allow FALCON/DILITHIUM integration into ZKEVM
- Allow PQ-Private Payments (PQ-Railgun)
- Solution for BLS replacement is Batching with snarks/starks.

Second one could use a dedicated scheme (LABRADOR).

Constraints

- FALCON and DILITHIUM operates on non native fields (Babybear, STwo)
- Non Native Fields require x30 more constraints (source: Bandersnatch vs P256 in Gnark, incoming paper)

ZKNOX aim: minimal tweak to obtain ZK friendly version

ZK friendly FALCON/DILITHIUM

Obvious tweaks

- Replace Hash function by a ZK friendly one
- Provide hints (see epervier definition) for easy batch inversion
- Accumulate several NTT steps, reduce once

Harder tweaks

- Starks fields are going shorter (no accumulation trick)
- Replacing fields is less trivial than for ECC (pick prime order curve and twist)
- FALCON is prone to overstretch attacks for ZK fields
- Estimator for S2/Babybear on dilithium provides 20% slowdown factor (WIP)

ZK Conclusion

- DILITHIUM is more ZK-friendly (and MPC too)
- Still experimental phase, spec very scheme sensitive

Conclusion

- Easy solutions for EOA migration
- EIP-7885 (NTT) to reduce gas costs, also reduce STARK proof cplx.
- Some efforts for HW signers, vendors will lobby for DILITHIUM
- Shall we tweak the proving scheme (LABRADOR) or the signature algorithm ?
- Privacy and Batching still very experimental, require STARKS with private input

ZKNOX would gladly help Wallets/Dapps to integrate its framework.

Questions ?

Blog

Github