poqeth: Efficient, Post-Quantum Signature Verification on Ethereum

Seungmin Kim
INFONET

Journal Club Summary
July 7, 2026

Paper: Ruslan Kysil, István András Seres, Péter Kutas, and Nándor Kelecsényi, "poqeth: Efficient, Post-Quantum Signature Verification on Ethereum," ASIA CCS 2025.

Overview

poqeth asks whether Ethereum smart contracts can verify post-quantum signatures at a reasonable cost.

- **Target:** W-OTS+, XMSS, SPHINCS+, and MAYO verifiers implemented in Solidity/EVM.
- **Comparison:** full on-chain verification vs. optimistic, Naysayer-style verification.
- **Metric:** gas cost under the ordinary EVM execution model.

The central question is:

> How expensive is post-quantum signature verification inside Ethereum smart contracts, and how much can the cost be reduced by replacing full on-chain verification with optimistic/Naysayer verification?

1 Introduction

The paper starts from the fact that cryptocurrency transactions rely heavily on digital signatures. **Bitcoin and Ethereum currently use pre-quantum signatures such as ECDSA or Schnorr. If large quantum computers become practical, these systems will need to move to post-quantum signatures.**

Bitcoin has a limited scripting language, but researchers have still tested simple post-quantum schemes such as Lamport signatures. Ethereum is more flexible: smart contracts can define their own signature checks. **The practical question is how much gas these checks use.**

The paper focuses on verification because it is the only part of the signature scheme that must run on-chain. Key generation and signing happen off-chain, so they do not directly determine the user's transaction fee. For Ethereum, the important factors are the operations used by the verifier, the amount of input data, storage use, and total gas.

1.1 Post-Quantum Signatures Under the Ethereum Cost Model

A fast signature scheme on a normal computer is not always cheap on Ethereum. Permanent storage is expensive, while transaction input data and temporary memory are cheaper. Also, `KECCAK256` is available as a native opcode, which makes hash-based signatures relatively attractive.

The following table gives the cost intuition used throughout the paper. Arithmetic and memory operations are cheap, hashing is moderately cheap, and persistent storage is much more expensive.

Opcode	Gas
`ADDMOD`	8
`MULMOD`	8
`KECCAK256`	36
`MSTORE/MLOAD`	3
`SLOAD`	2,100
`SSTORE`	20,000

From this perspective, the paper evaluates four signature schemes. One notable omission is Falcon. **The paper excludes Falcon because the EVM does not natively support floating-point arithmetic and software emulation would be expensive.**

The next table explains why the paper compares these four schemes rather than treating all post-quantum signatures as a single category. Each family stresses a different part of the EVM cost model.

Scheme	Family	Reason for inclusion
W-OTS+	Hash-based one-time	A core building block of XMSS and SPHINCS+, and well matched to Keccak-based computation.
XMSS	Stateful hash-based	Signer state can be managed by a smart contract, which fits naturally with blockchain state.
SPHINCS+	Stateless hash-based	A NIST-standardized post-quantum signature scheme that does not require signer state.
MAYO	Multivariate quadratic	Provides a different trade-off from hash-based schemes, with relatively compact signatures and public keys.

The paper makes three main contributions:

- it implements four post-quantum signature verifiers in the EVM and measures their gas costs;
- it compares full on-chain verification with Naysayer verification;
- it releases the implementation as the poqeth library.

2 System Model

The paper compares two modes; key generation and signing happen off-chain in both. **Naysayer mode normally accepts the signature and checks only one small step when a monitor reports an error.**

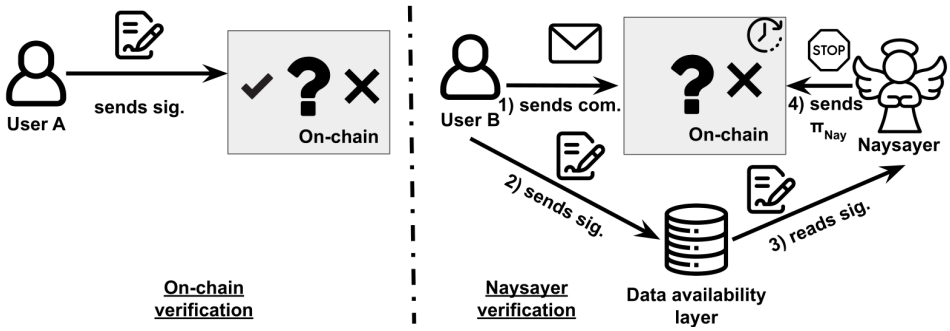

Figure 1: The two verification modes studied in the paper: full on-chain verification and Naysayer verification.

Full on-chain mode. The contract receives the signature in calldata and runs the complete verifier immediately. **This mode needs no outside monitor, but the contract pays for every hash, Merkle step, or polynomial operation.**

Naysayer mode. The contract stores only a Merkle root for the signature, while monitors download and check the full signature off-chain. The root is only 32 bytes, but the full signature must remain available or monitors cannot find an error. The signature is accepted after a challenge period unless someone proves an error. This saves gas when signatures are valid, but it assumes:

- the challenge period is long enough for a transaction to arrive,
- at least one honest monitor checks each signature,
- and attackers cannot block challenge transactions.

Protocol flow.

1. The submitter posts a Merkle root for the signature and locks some money.
2. Monitors check the full signature during the challenge period.
3. A monitor that finds an error reveals the values for one wrong step.
4. The contract checks that step; a valid challenge rejects the signature, while no challenge means acceptance.

Naysayer verification does not remove the work: it moves full verification off-chain and puts only a small error proof on-chain.

3 Hash-Based Signature Scheme Verification in the EVM

This is the core section of the paper. The authors implement W-OTS+, XMSS, and SPHINCS+ in the EVM and compare full on-chain verification with Naysayer verification. All three implementations use `KECCAK256`. **The goal is not to keep the exact hash choices of the standards, but to evaluate what is efficient under the EVM's native cost model.**

3.1 W-OTS+

W-OTS+ is a one-time hash-based signature scheme and a building block for both XMSS and SPHINCS+. **W-OTS+ reveals one hash-chain value for each digit of the message digest.** Its main parameter is the Winternitz parameter w. This parameter creates a trade-off between public key size and hash chain length.

The role of w. A larger w means fewer hash chains, but each chain becomes longer. For example, $w = 16$ uses about twice as many chains as $w = 256$, while their maximum lengths are 15 and 255. **Thus, increasing w makes the signature and public key shorter but increases the amount of hashing.**

What the verifier computes. Write the base-w message digits as b_i and the signature elements as σ_i. For each chain, the verifier reconstructs a public-key element

$$pk'_i = H^{\,w-1-b_i}(\sigma_i).$$

Verification succeeds when all reconstructed chain values, including checksum chains, produce the expected public key.

A small example. A digit b_i is one base-w digit of the message digest, not the whole message. For $w = 16$, every digit is between 0 and 15. If $b_i = 5$, the signer reveals the chain value

$$\sigma_i = H^5(sk_i),$$

and the verifier hashes it another $16 - 1 - 5 = 10$ times to reach $H^{15}(sk_i) = pk_i$.

The attacker must guess a 256-bit chain value, not one of the 16 digit values. Because revealed values can be hashed forward, a W-OTS+ key must be used only once.

Hash-chain computation is the main bottleneck: $w = 4$ is best for full verification, while $w = 16$ is best for Naysayer mode because it checks only one faulty chain.

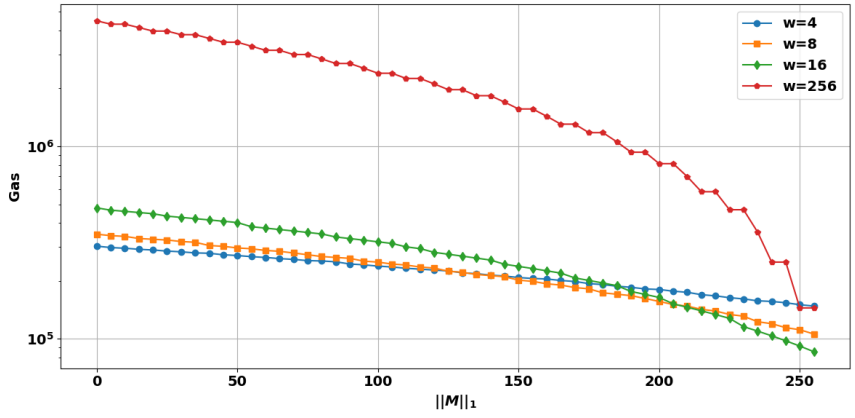

Figure 2: W-OTS+ on-chain gas. The message and w change the number of hash operations.

3.2 XMSS

XMSS is a stateful hash-based signature scheme that organizes W-OTS+ public keys in a Merkle tree. The signer must update a counter so that each one-time key is used only once; a smart contract can manage this public state.

XMSS puts many W-OTS+ public keys in a Merkle tree and proves a path from the selected key to the public root.

A small example. In the right-hand figure, red L_3 is the reconstructed W-OTS+ leaf and yellow L_2, H_0, and H_{23} form its Merkle path. The verifier computes the blue nodes:

$$H_1 = H(L_2 \parallel L_3), \qquad H_{01} = H(H_0 \parallel H_1), \qquad Root = H(H_{01} \parallel H_{23}).$$

The result must equal the public root. If H_{01} is wrong, a Naysayer reveals only H_0, H_1, and H_{01}.

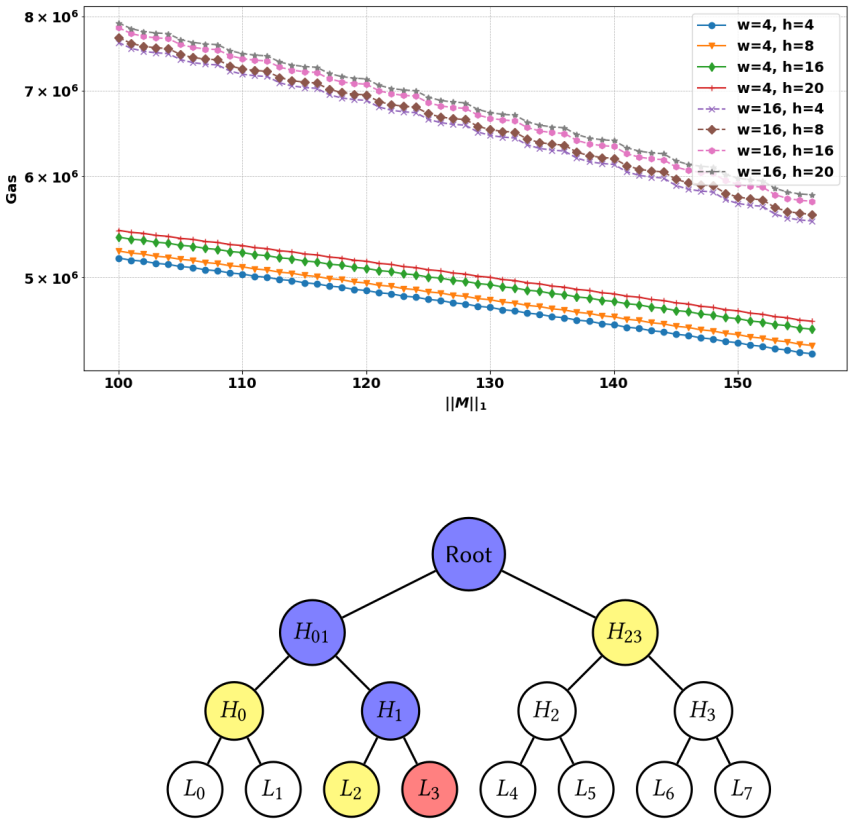

Figure 3: XMSS gas results (top) and the local Merkle-path check used by a Naysayer (bottom).

In the gas graph, changing the tree height moves the curves only slightly. Most of the gap comes from the W-OTS+ and L-tree work that must be done before the Merkle path is checked. **For XMSS, the W-OTS+ and L-tree calculations cost more gas than increasing the Merkle tree height. XMSS needs several million gas in full on-chain mode, but Naysayer verification reduces this to roughly 600,000 gas.**

3.3 SPHINCS+

SPHINCS+ repeats an XMSS-like check across several layers so that the signer does not need to track state. Its verifier combines FORS, W-OTS+, and several Merkle-tree layers.

Where the work comes from. FORS turns the message digest into a root value. The verifier then checks this root through d layers, each with another W-OTS+ verification and Merkle path.

A small example. Read the figure from the green message value M at the bottom to the blue public key pk at the top. FORS creates R_0, and each W-OTS+ plus yellow Merkle path produces the next root:

$$M \xrightarrow{\text{FORS}} R_0 \xrightarrow[\text{yellow path}]{\text{W-OTS+}} R_1 \xrightarrow[\text{yellow path}]{\text{W-OTS+}} pk.$$

A Naysayer challenges one wrong FORS, W-OTS+, or Merkle-path step instead of checking the full route.

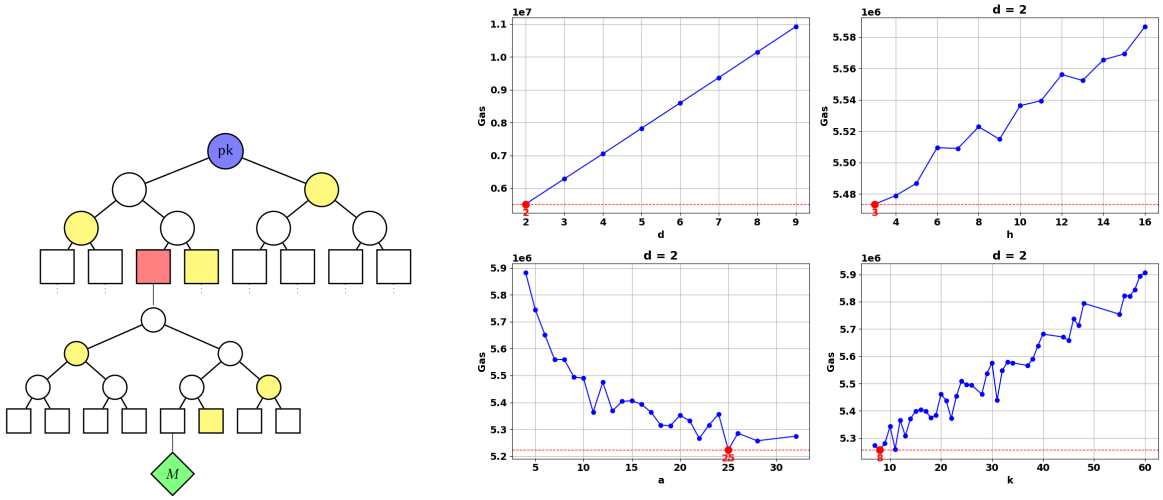

Figure 4: SPHINCS+ structure from M to pk (left) and the effect of its parameters on verification gas (right).

Here, h is the total hypertree height, d is the number of layers, and a and k control the FORS trees. The right-hand plots show that increasing d repeatedly adds W-OTS+ work, while the other parameters mainly change Merkle and FORS path lengths. The paper's best parameter set is $(h, d, a, k) = (63, 10, 12, 15)$. The number of layers d affects gas the most because each layer adds another W-OTS+ check. **Full SPHINCS+ verification costs 11,617,690 gas, while the cheapest Naysayer check costs about 690,000 gas.**

4 MAYO: A Multivariate-Quadratic Signature Scheme in the EVM

MAYO puts signature values into many quadratic equations and checks that every equation matches the message. The paper tests whether its small signatures and public keys also lead to lower Ethereum cost.

Current standardization status. On May 14, 2026, NIST advanced MAYO to the third round of its Additional Digital Signatures process. It is still a candidate, not a final standard.

For the EVM, the paper calculates modulo 31 using F_{31} and replaces AES-CTR public-key expansion with a `KECCAK256`-based generator.

What the verifier computes. The verifier rebuilds the public equations from a seed and checks

$$P(s) \stackrel{?}{=} t,$$

where s comes from the signature and t from the message. The chosen parameters are $(q, m, n, o, k) = (31, 60, 62, 6, 10)$. Thus, verification produces 60 equation outputs from 10 vectors of 62 field elements, and rebuilds an expanded public map of about 115KB. For example, assume the toy polynomial $P_1(x_1, x_2) = x_1^2 + 2x_1x_2 + 3x_2$ over F_{31}, with coefficients $(1, 2, 3)$. Then

$$P_1(2, 4) = 2^2 + 2(2)(4) + 3(4) = 4 + 16 + 12 = 32 \equiv 1 \pmod{31}.$$

A real signature must satisfy many such equations; one failing equation is enough for a Naysayer proof.

Full on-chain MAYO verification costs 938,752,492 gas. Polynomial-related operations use about 84% of this gas because the EVM has no SIMD support. The code also stores each 5-bit coefficient in a full 256-bit word instead of packing several coefficients together. The short public-key seed saves storage, but it forces the verifier to rebuild a large public map for every signature. Likewise, using a whole EVM word for one small coefficient wastes memory and data movement. **Checking one failing equation in Naysayer mode reduces the cost to 107,634,064 gas, but this is still very expensive.**

In summary, MAYO has small keys and signatures and fits Naysayer proofs well, but full on-chain verification is not practical without major optimization or SIMD-like EVM support.

5 Applying Naysayer Proofs to Other Candidates

The authors also discuss other NIST candidates, but do not implement them. Hash-based schemes can expose one wrong Merkle or hash-chain step, while equation-based schemes can expose one failing equation. **The key question is whether one small wrong step is enough to prove that a signature is invalid.** These are design sketches rather than measured implementations, so their practical gas savings remain future work.

6 Conclusion and Future Directions

The gas totals are easier to understand when we identify the most expensive operation in each verifier.

Scheme	Main full-verification work	What one challenge checks
W-OTS+	Repeated hash-chain iterations	One incorrect chain or chain step
XMSS	W-OTS+ reconstruction and L-tree compression	One wrong W-OTS+, L-tree, or Merkle-path step
SPHINCS+	FORS plus repeated W-OTS+ and hypertree paths	One faulty FORS, W-OTS+, hash, or hypertree step
MAYO	Rebuilding and checking quadratic equations	One equation that does not match

Signature size alone does not predict Ethereum cost: we must also ask whether verification uses cheap EVM operations and whether one small proof can show an error.

The paper's conclusions can be summarized as follows.

- Post-quantum signature verification on Ethereum is possible, but full on-chain verification is expensive.
- Hash-based schemes are relatively well matched to the EVM because of its Keccak-friendly cost model.
- Naysayer verification provides large gas savings, but it adds liveness and honest-naysayer assumptions.
- MAYO has the advantage of compact signatures, but polynomial evaluation is currently too expensive in the EVM.

Limits of the Evaluation

The reported Naysayer gas numbers cover only the on-chain challenge, not the full system cost. The reported savings exclude monitor operation, full-signature availability, monitor rewards, and the delay of the challenge period. Security also depends on at least one honest monitor finding an error and sending a challenge before the deadline.

Three questions therefore remain open for a deployment:

- Who pays monitors when almost every submitted signature is valid?
- Can an attacker hide signature data or block challenge transactions until the deadline?
- Is the full system cheaper than native verification or a short zero-knowledge proof?

The paper leaves the monitor incentive and reward mechanism as an open problem.

Our comments. **First, excluding Falcon leaves an important comparison missing.** Falcon uses floating-point arithmetic mainly in key generation and signing, while verification is based on integer and modular operations such as hash-to-point, NTT-style polynomial multiplication, and a norm check. Pure EVM Falcon verification may still be very expensive, but the lack of native floating point is not by itself a strong reason to exclude it from a verification-focused study.

Second, the paper gives too little attention to repeated data cost. A public key may be stored once and reused, but every transaction carries a new signature. Large signatures therefore consume calldata and block space for every transaction, and Naysayer mode still requires the full signature to remain available to all monitors. A complete comparison should report signature size, public-key size, verification gas, and data cost together.

The gas results are still useful, but low challenge gas alone does not prove that the full system is practical.

The main results can be summarized as follows. This table is the best one-page summary of the experimental result. It shows that the optimistic/Naysayer architecture gives large savings for every scheme, but it does not make every scheme practical.

Mode	W-OTS+	XMSS	SPHINCS+	MAYO
On-chain	222,114	4,363,623	11,617,690	938,752,492
Naysayer	126,095	594,572	693,721	107,634,064
Reduction	43.2%	86.4%	94.0%	88.5%

Using an ETH price of 2,489 USD and a median gas price of 5.08 Gwei, the paper converts these costs to USD as follows. This conversion is not meant to be a permanent price estimate. It is useful because it turns abstract gas numbers into an application-level deployment cost.

Mode	W-OTS+	XMSS	SPHINCS+	MAYO
On-chain	$2.76	$54.31	$144.58	$12,411.01
Naysayer	$1.57	$7.40	$8.63	$1,339.48

One workaround is a native precompile, similar in spirit to the secp256r1 verifier first deployed as RIP-7212 and later specified for Ethereum L1 in EIP-7951. For post-quantum signatures, Ethereum could precompile either the full verifier or only expensive steps such as polynomial evaluation. This would avoid much of the Solidity cost, but it requires a protocol change and a new gas-pricing decision.

Future work also includes short proofs of correct verification, better reward rules for monitors, and migration plans for Ethereum accounts.