



# poqeth

## Efficient, post-quantum signature verification on Ethereum

Ruslan Kysil  
Eötvös Loránd University  
Budapest, Hungary  
[ruslan.ilesik@gmail.com](mailto:ruslan.ilesik@gmail.com)

Péter Kutas  
Eötvös Loránd University  
Budapest, Hungary  
University of Birmingham  
Birmingham, United Kingdom  
[kutasp@gmail.com](mailto:kutasp@gmail.com)

István András Seres  
Eötvös Loránd University  
Budapest, Hungary  
[seresistvanandras@gmail.com](mailto:seresistvanandras@gmail.com)

Nándor Kelecsényi  
Eötvös Loránd University  
Budapest, Hungary  
[kelecsenyi@protonmail.com](mailto:kelecsenyi@protonmail.com)

### Abstract

This work explores the application and efficient deployment of (standardized) post-quantum (PQ) digital signature algorithms in the blockchain environment. Specifically, we implement and evaluate four PQ signatures in the Ethereum Virtual Machine: W-OTS<sup>+</sup>, XMSS, SPHINCS<sup>+</sup>, and MAYO. We focus on optimizing the gas costs of the verification algorithms as that is the signature schemes' only algorithm executed on-chain, thus incurring financial costs (transaction fees) for the users. Hence, the verification algorithm is the signature schemes' main bottleneck for decentralized applications.

We examine two methods to verify post-quantum digital signatures on-chain. Our practical performance evaluation shows that full on-chain verification is often prohibitively costly. Naysayer proofs (FC'24) allow a novel optimistic verification mode. We observe that the Naysayer verification mode is generally the cheapest, at the cost of additional trust assumptions. We release our implementation called **poqeth** as an open-source library.

### ACM Reference Format:

Ruslan Kysil, István András Seres, Péter Kutas, and Nándor Kelecsényi. 2025. poqeth Efficient, post-quantum signature verification on Ethereum. In *ACM Asia Conference on Computer and Communications Security (ASIA CCS '25)*, August 25–29, 2025, Hanoi, Vietnam. ACM, New York, NY, USA, 17 pages. <https://doi.org/10.1145/3708821.3736193>

## 1 Introduction

Issuing cryptocurrency transactions is one of the most widely used and important digital signature applications. For instance, Bitcoin [42] settles roughly 300,000 transactions per day [29]. Furthermore, billions of dollars worth of crypto assets are transferred daily

on cryptocurrencies such as Bitcoin or Ethereum [57]. Each transaction is considered valid if digitally signed with either ECDSA [36] or the Schnorr signature algorithm [52]. At the time of writing, no major blockchain supports verifying post-quantum (PQ) secure digital signature algorithms (DSA) natively.

Several post-quantum secure signature schemes [10, 24, 50] and key encapsulation mechanisms have recently been standardized [40] by the National Institute of Standards and Technology (NIST) and the Internet Engineering Task Force (IETF) [33]. These successful standardization processes and the looming threat of quantum computers [53] compel practitioners to switch from pre-quantum to post-quantum signature schemes. Therefore, the Internet is bound to transition from pre-quantum signatures (e.g., RSA [51], ECDSA [36], or Schnorr [52]) to post-quantum secure signature schemes. A growing body of literature is already evaluating the cost profile of the newly standardized PQ signature schemes in various Internet protocols, e.g., the TLS protocol [44]. However, a similar line of work has yet to be developed to support the post-quantum transitioning of cryptocurrencies and blockchain applications.

### 1.1 Post-quantum security and cryptocurrencies

The problem of saving crypto assets from adversaries with access to quantum computers has been addressed in the literature [55]. Stewart et al.'s proposed solution for Bitcoin can be instantiated with any post-quantum digital signature algorithm. Interestingly, the purposefully limited Bitcoin scripting language allows one to encode<sup>1</sup> the verification circuit of Lamport signatures [32]. The resulting transaction will be roughly the same size as a regular Bitcoin transaction; therefore, there is a negligible additional cost incurred by verifying the Lamport signatures on Bitcoin. Hence, we consider the problem of efficiently verifying post-quantum secure digital signatures on Bitcoin solved. We review related work in more detail in Section 7. Therefore, in this work, we focus on enabling Ethereum, the second-largest cryptocurrency by market capitalization and the largest by transaction volume, to allow its users to issue post-quantum secure transactions.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

ASIA CCS '25, Hanoi, Vietnam

© 2025 Copyright held by the owner/author(s). Publication rights licensed to ACM.  
ACM ISBN 979-8-4007-1410-8/25/08  
<https://doi.org/10.1145/3708821.3736193>

<sup>1</sup>See Ethan Heilman's Bitcoin script for the hash-based Lamport signature verification: <https://groups.google.com/g/bitcoinddev/c/mR53go5gHlk>.

| EVM Opcode   | Gas cost |
|--------------|----------|
| ADDMOD       | 8        |
| MULMOD       | 8        |
| KECCAK256    | 36       |
| SSTORE       | 20,000   |
| SLOAD        | 2,100    |
| MSTORE       | 3        |
| MLOAD        | 3        |
| CALLDATALOAD | 3        |

**Table 1: Gas costs of important opcodes in the EVM [57]. Opcodes for hashing, temporal memory, and arithmetic operations are orders of magnitude cheaper than writing or reading permanent storage (SSTORE/SLOAD).**

## 1.2 The right PQ signature for Ethereum

Blockchains and their computing environments have unique requirements and limitations, e.g., permanent storage is extremely expensive, as the data stored on the blockchain are replicated and stored by the entire network, potentially by thousands of nodes [57]. In particular, when selecting post-quantum digital signatures for Ethereum, one must consider the specifics of the quasi-Turing-complete Ethereum Virtual Machine (EVM): the computational costs of executing EVM opcodes [47, 57], cf. Table 1.

First and foremost, for digital signature schemes, we primarily focus on the efficiency of the verification algorithm in the blockchain context, as only signature verification happens on-chain. Key generation and signing could be even slower, as they are executed off-chain. Thus, they do not incur any financial costs to users. Ethereum smart contracts are compiled to EVM bytecode, and the smart contract bytecode is executed on-chain. Ethereum’s computation environment measures the cost of each executed opcode by its so-called gas value, see Table 1. We enlist four characteristics of the EVM that largely influenced our choices in the implementation and evaluation of PQ digital signature algorithms.

**Permanent vs. transient storage** In Ethereum, reading and writing permanent storage (i.e., the blockchain) is expensive (SSTORE, SLOAD), especially compared to (modular) arithmetic (e.g., ADDMOD) or hashing operations (KECCAK256). Thus, signature schemes with large public keys were ruled out from further consideration, e.g., Dilithium [24]. Since signatures are only temporarily stored in the transaction CALLDATA, in the blockchain context, we favour small public keys even at the expense of larger signature sizes.

**Computational efficiency vs. gas pricing** In the EVM, each bytecode operation is assigned a gas cost, which does not always correlate directly with the actual computational resources required [47]. For example, invoking a cryptographic hash function costs 36 gas (KECCAK256), while performing a modular multiplication on 256-bit integers costs 8 gas (MULMOD). This results in an interesting observation: five multiplications are more expensive than a single hash operation, even though hash functions are typically considered more computationally intensive. Consequently, cryptographic operations, such as hash-based signatures, often

deemed computationally expensive, can be surprisingly efficient and cheap when implemented on Ethereum.

**Signer’s state** In some applications, stateful signatures are considered burdensome to deploy since the signer must keep state (typically just a counter as in XMSS [33]). Note that the verifier in the blockchain context can track this simple state since the verifier is implemented as a smart contract, i.e., a trusted party with a transparent state. This observation makes stateful signatures a perfect fit for blockchain applications since the signer can always read its simple state from the immutable, public blockchain, thus obviating the common synchronization issues of stateful signature schemes.

**Lack of native floating point arithmetic** The EVM is a stack machine with a 256-bit word size: it natively only supports modular arithmetic for integers less than  $2^{256}$ . Custom-made smart contract libraries may support larger integers (e.g., for verifying RSA signatures) or floating point arithmetic, though typically at the expense of moderate gas costs. For instance, there exist floating point libraries for Ethereum smart contracts.<sup>2</sup> Still, they incur gas costs (i.e.,  $\sim 600$  gas for addition) that are two orders of magnitude more expensive than their natively supported integer arithmetic counterparts, cf. ADDMOD, MULMOD in Table 1.

Considering the aforementioned characteristics and limitations, we decided to implement and evaluate an on-chain deployment of the following four post-quantum digital signatures in the EVM.

**W-OTS<sup>+</sup>** [32, 34]. Our first choice is the Winternitz-type one-time signature scheme by H ulsing. W-OTS<sup>+</sup> is an essential building block of standardized (IETF and NIST) hash-based signatures such as XMSS and SPHINCS+. Unlike Lamport signatures, it also offers a time-memory tradeoff we explore and evaluate in Section 4.1.

**XMSS** [31, 33]. As argued above, stateful signatures are practical in the blockchain context, as a smart contract can easily keep the state for the signer. Therefore, the first many-time signature scheme (though with finite signing capabilities) we consider is the stateful XMSS signature scheme that the IETF is currently standardizing.

**SPHINCS+** [3, 10]. Arguably, the most important stateless hash-based signature scheme that NIST has recently standardized is SPHINCS+. It offers small public keys, moderate signature sizes, and efficient verification.

**MAYO** [13, 14]. To diversify the trust assumptions our implemented schemes use, we consider a multivariate-quadratic-based signature scheme that offers one of the most compact signatures (321 bytes at NIST security level 1) with short public keys (1168 bytes at NIST sec. level 1). MAYO has been submitted to the latest round of NIST’s post-quantum digital signature standardization call and advanced to the second round at the time of writing.

## 1.3 Our contributions

In this work, we make the following three contributions.

- **On-chain verification.** We implement and evaluate the verification algorithm of four post-quantum digital signature algorithms (W-OTS<sup>+</sup>, XMSS, SPHINCS+, and MAYO) in

<sup>2</sup>See: <https://github.com/abdk-consulting/abdk-libraries-solidity>.

| Signature<br>Mode | W-OTS <sup>+</sup> | 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 |

**Table 2: On-chain vs. Naysayer verification gas costs in the EVM with the most efficient parameter settings for each considered PQ signature scheme. See Table 7 for the verification costs denominated in US dollars and Appendix D for PQ migration costs.**

the EVM. We suggest optimal parameter choices and other design decisions for a gas-efficient on-chain deployment.

- **Naysayer verification.** We implement and evaluate an efficient, recently proposed way of optimistically verifying proofs, in particular, PQ signatures. The so-called Naysayer verification [54] mode offers significant speedups compared to full verification. We recall the Naysayer proofs in Section 2.3. To our knowledge, this is the first publicly available implementation of the recently proposed Naysayer proof paradigm in the context of post-quantum secure digital signature schemes. We observe in Table 2 that currently, on Ethereum, the Naysayer verification mode offers the most efficient post-quantum signature scheme deployment.
- We release a proof-of-concept implementation, an extendable library we call **poqeth**,<sup>3</sup> is available under the MIT licence at <https://github.com/ruslan-ilesik/poqeth>.

The remainder of this paper is organized as follows. In Section 2, we recall the pertinent preliminaries on Ethereum, the EVM, smart contracts, and Naysayer proofs. In Section 3, we introduce our system model and (trust) assumptions. The on-chain deployment of hash-based signatures is presented in Section 4. We detail the deployment of the MAYO signature scheme for the EVM in Section 5. We design Naysayer proof systems for more NIST candidate PQ signature schemes in Appendix C. We review related work in Section 7. Our paper concludes with open problems and future directions for post-quantum transitioning in Ethereum in Section 8.

## 2 Preliminaries

This section introduces the notations used, the pertinent background knowledge on the Ethereum Virtual Machine (EVM), account abstraction in the EVM, and Naysayer proofs.

### 2.1 Notations

The security parameter is denoted as  $\lambda$ . Uniformly at random sampling an element  $x$  from a set  $S$  is denoted as  $x \in_R S$  or  $x \xleftarrow{\$} S$ . The Hamming weight of a binary string  $m \in \{0, 1\}^k$  is denoted as  $|m|_1$ . Vectors of strings or group/field elements are typeset in bold, e.g.,  $\mathbf{r} = (r_1, \dots, r_l)$  and  $\mathbf{r}[i]$  is the  $i$ th element of  $\mathbf{r}$ . The Merkle tree vector commitment of a vector  $\mathbf{r}$ , i.e., the Merkle tree’s root hash, is denoted as  $\mathcal{MT}(\mathbf{r}) = \text{com}_{\mathbf{r}}$ . We denote the membership proof (i.e., the authentication path) for an element  $r_i \in \mathbf{r}$  as  $\pi_{\mathbf{r}, r_i}$ . The Merkle membership verification algorithm has the following syntax  $\mathcal{MT}.Vrfy(\text{com}_{\mathbf{r}}, i, r_i, \pi_{\mathbf{r}, r_i}) \rightarrow \{0, 1\}$ . We use similar notations for other vector commitments, such as XMSS trees or L-trees.

<sup>3</sup>**poqeth** (/ˈpɑː.kət/): the library’s name alludes to the fast, small, pocket-sized post-quantum signature verification algorithms we implemented and evaluated.

### 2.2 The Ethereum Virtual Machine

The EVM is a quasi-Turing-complete<sup>4</sup> stack machine with 1024 stack size and a word size of 256 bits. The formal semantics of the EVM’s various opcodes are defined in [57]. Numerous opcodes exist for logical operations, modular arithmetic, and contract calls, among others, cf. Table 1. Three main programming languages compile to EVM bytecode: Solidity, Vyper, and Yul. In this work, we implement the benchmarked algorithms in Solidity. Currently, there are two types of accounts: externally owned accounts (EOAs) whose behaviour is determined by a secret-public key pair and contract accounts that do not possess a secret-public key. Rather, their behaviour is determined by some (immutable) EVM byte code. At the time of writing, each valid Ethereum transaction must come with the sender’s ECDSA signature. The EVM only supports the Keccak-256, SHA-256, and RIPEMD-160 hash functions natively.

*Account abstraction.* The Ethereum Request for Comments (ERC-4337) proposal, often called “account abstraction”, aims to unify EOAs and contract accounts [19]. More importantly, ERC-4337 would allow users to authenticate transactions with any digital signature scheme they choose. In particular, Ethereum users could sign their transactions using PQ digital signature algorithms, e.g., SPHINCS+ or MAYO. This opens up the possibility of endowing Ethereum with post-quantum security. These “non-native” signatures would be verified on-chain by custom smart contracts running in the EVM. Thus, the motivation for our work is to evaluate the efficiency of verifying PQ digital signatures in the EVM.

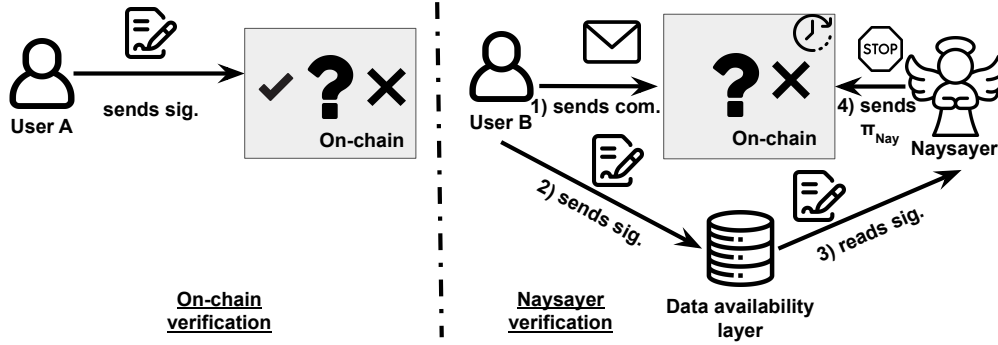
### 2.3 Naysayer proofs

A Naysayer proof system  $\Pi_{\text{nay}}$  [54] for a proof system  $\Pi$  allows one to efficiently prove with a naysayer proof  $\pi_{\text{nay}}$  that a certain proof  $\pi$  is incorrect for an NP relation  $\mathcal{R}$ . Naysayer proofs are useful when they can be verified much more efficiently than verifying  $\pi$ . We recall the formal definition of a *naysayer proof*:

*Definition 2.1 (Naysayer proof).* Given a non-interactive proof system  $\Pi = (\text{Setup}, \text{Prove}, \text{Verify})$  for some NP relation  $\mathcal{R}$ , the corresponding naysayer proof system  $\Pi_{\text{nay}}$  is a tuple of PPT algorithms  $(\text{NSetup}, \text{NProve}, \text{Naysay}, \text{VerifyNay})$  defined as follows:

- $\text{NSetup}(1^\lambda, 1_{\text{nay}}^\lambda) \rightarrow (\text{crs}, \text{crs}_{\text{nay}})$ : Given security parameters  $1^\lambda$  and  $1_{\text{nay}}^\lambda$  for the proof systems  $\Pi$  and  $\Pi_{\text{nay}}$ , respectively, output common reference strings  $\text{crs}$  and  $\text{crs}_{\text{nay}}$ . Note this algorithm might use private randomness.
- $\text{NProve}(\text{crs}, x, w) \rightarrow \pi$ : Given a statement  $x$  and witness  $w$  such that  $(x, w) \in \mathcal{R}$ , compute  $\pi' \leftarrow \Pi.\text{Prove}(\text{crs}, x, w)$  and

<sup>4</sup>The amount of computation (the `block_gas_limit` parameter) in a single block is upper bounded by the protocol. However, in principle, it may be raised arbitrarily high.



**Figure 1: The two signature verification modes we consider in this work. In the on-chain verification paradigm (left), a smart contract verifies the correctness of users’ signatures. In the Naysayer verification paradigm (right), on-chain only a commitment to the full signature is stored. The full signature is stored off-chain in a data availability layer. Naysayers can prove to an on-chain smart contract *the incorrectness* of the provided user signature by sending a Naysayer proof  $\pi_{\text{Nay}}$  to the verifier contract. The signature is deemed valid if no correct Naysayer proof has been submitted before a pre-set deadline.**

com a commitment to the evaluation trace of  $\Pi$ .  $\text{Verify}(\text{crs}, x, \pi') \rightarrow \text{output } \pi := (\text{com}, \pi')$ .

$\text{Naysay}(\text{crs}_{\text{Nay}}, (x, \pi), \text{Aux}_{\text{Nay}}) \rightarrow \pi_{\text{Nay}}$ : Given a statement  $x$ ,  $\pi = (\text{com}, \pi')$  where  $\pi'$  is a corresponding (potentially invalid) proof in proof system  $\Pi$ , and auxiliary information  $\text{Aux}_{\text{Nay}}$ , output a naysayer proof  $\pi_{\text{Nay}}$  disputing  $\pi'$ .

$\text{VerifyNay}(\text{crs}_{\text{Nay}}, (x, \pi), \pi_{\text{Nay}}) \rightarrow \{0, \perp\}$ : Given a pair of statement and a proof  $(x, \pi = (\text{com}, \pi'))$  and a naysayer proof  $\pi_{\text{Nay}}$  disputing  $\pi$ , output a bit indicating whether evidence against  $\pi$  is sufficient to reject  $\pi$  (0) or inconclusive ( $\perp$ ).

A naysayer proof system must satisfy correctness and soundness. Intuitively,  $\Pi_{\text{Nay}}$  satisfies correctness if every incorrect proof  $\pi$  can be proven to be incorrect with an appropriate naysayer proof  $\pi_{\text{Nay}}$ . On the other hand, soundness dictates that a correct proof  $\pi$  must be impossible to proven to be incorrect by a  $\pi_{\text{Nay}}$  with non-negligible probability. For formal definitions, the reader is referred to Appendix A.

### 3 System Model

This section describes the two signature verification modes’ system models, we apply in this work. In both models, key generation and the signing algorithms are run off-chain.

#### 3.1 On-chain signature verification

A smart contract running in the EVM verifies the full PQ secure signature on-chain. These custom-made signature verification smart contracts have standardized interfaces, as suggested in the ERC-4337 proposal. In principle, ERC-4337 compliant smart contracts allow the verification of any user-chosen digital signature scheme. This verification mode solely assumes the existential unforgeability of the applied signature scheme under chosen message attacks (EUF-CMA). The signature is only temporarily stored in the CALLDATA, and after verification (possibly another contract handles application-specific state changes, e.g., updating token balances), it is not available to the contract anymore. Most blockchain consensus algorithms assume synchrony. However, the on-chain

verification mode even works in an asynchronous network communication model, i.e., post-quantum signed transactions eventually reach the Ethereum peer-to-peer network. We expect on-chain verification to be moderately costly for most post-quantum signature algorithms, cf. Sections 4 and 5. Expensive on-chain verification motivates the following optimistic verification mode.

#### 3.2 Naysayer signature verification

The Naysayer verification mode aims to minimize the gas costs of the on-chain verification mode. Therefore, verifying a PQ signature is broken into two steps.

**Storing signature on-chain** The signer stores the signature on-chain, but the contract does not verify the correctness of the signature. Rather, it optimistically accepts it. In practice, we do not store the full signature on-chain as it would be gas inefficient. Rather, only the signature’s Merkle commitment (32 bytes) is stored in the contract’s permanent storage (which can be deleted after the challenge period). Note that the full signature is provided by the signer but the Merkle commitment to it must be computed (and stored) by the on-chain contract. If a malicious signer would provide a faulty commitment, then it would not be possible to naysay efficiently the incorrect signature. On the other hand, if the verifier is tasked to provide the Merkle commitment of the full signature, then the verifier could provide a faulty commitment, essentially accusing the signer of sending an invalid signature. Therefore, we must resort to computing the Merkle commitment of the signature by the on-chain smart contract. We assume that the full signature is available for inspection off-chain to everyone for naysaying.

**Challenge period** During a pre-defined challenge period (e.g., 1 hour), anyone could challenge the correctness of the Merkle-committed PQ signature by sending naysayer proof to the on-chain contract. If  $\pi_{\text{Nay}}$  is verified, the signature is deemed incorrect. If no verified  $\pi_{\text{Nay}}$  arrives in the contract during the challenge period, the signature is accepted as valid.

The Naysayer verification paradigm introduces massive cost savings compared to on-chain signature verification. First, in the happy path, when the signature is correct, the PQ signature does not even need to be verified on-chain. Second, in the sad path, when the signature is incorrect, one only needs to verify a naysayer proof  $\pi_{\text{nay}}$  whose verification cost is typically significantly lower than that of verifying the entire signature on-chain, cf. Sections 4 and 5.

However, the Naysayer verification mode introduces additional trust assumptions. In particular, we assume a synchronous communication model; otherwise, no  $\pi_{\text{nay}}$  could arrive at the blockchain on time if the adversary can arbitrarily delay messages. We assume existential honesty, i.e., there is at least one honest party who monitors the correctness of PQ signatures and naysays when necessary. Finally, we assume that the underlying blockchain is censorship-resistant [56]. Typically, as our performance evaluation shows in Sections 4 and 5, Naysayer signature verification is more gas efficient than on-chain signature verification.

*Resorting to full on-chain signature verification.* We do not consider that the on-chain and Naysayer verification modes are mutually exclusive. Specifically, we acknowledge that certain applications could apply full on-chain signature verification even when no Naysayer proof  $\pi_{\text{nay}}$  has been received on-chain. This approach may make sense in a regime where signatures are faulty more often than not. An extensive cost- and game-theoretic analysis could evaluate this choice by modeling the probability that a signature is incorrect. We leave to future work the efficiency evaluation of the full signature vs. Naysayer verification modes depending on the probability that signatures are faulty.

## 4 Hash-based signature scheme verification in the EVM

This section studies implementation challenges and optimal parameter sets for hash-based signature schemes in the EVM. We focus on W-OTS<sup>+</sup>, XMSS, and SPHINCS+. We evaluate both on-chain and Naysayer verifications. We implement every scheme with Keccak-256, which is currently the most secure choice of the three available hash functions of the EVM. It is possible to implement other hash functions in the EVM; however, they likely incur at least one order of magnitude more gas for *every hashing operation*. Hence, we slightly deviate from the NIST and IETF standards for gas efficiency.

### 4.1 The W-OTS<sup>+</sup> signature scheme

We start with W-OTS<sup>+</sup> as it is a core building block also in the XMSS and SPHINCS+ signature schemes. We quickly recall the W-OTS<sup>+</sup> hash-based signature scheme [32]. We stylize parts of the signature scheme irrelevant to our verification-focused discussion. The W-OTS<sup>+</sup> signature scheme with message length  $m$  is parametrized by the Winternitz parameter  $w \in \mathbb{N}$ , determining the time-memory tradeoff. We set the message length (in bits) to be  $m = 256$ . All messages are written in base  $w$ . Additional parameters  $l, l_1, l_2$ , where  $l = l_1 + l_2 = \left\lceil \frac{m}{\log(w)} \right\rceil + \left\lceil \frac{\log(l_1(w-1))}{\log(w)} \right\rceil$  depend on  $w$ . W-OTS<sup>+</sup> uses the following chaining function:

$$c_k^i(x, \mathbf{r}) := h_k(c_k^{i-1}(x, \mathbf{r}) \oplus r_i), \quad (1)$$

where  $h_k : \{0, 1\}^\lambda \rightarrow \{0, 1\}^\lambda$  is a cryptographic (non-compressing) hash function, for  $k \in \mathcal{K}_\lambda$  with a keyspace  $\mathcal{K}_\lambda$  and  $\mathbf{r}$  is a vector of random bit-strings, i.e.,  $\mathbf{r} = \{r_k\}_{k=1}^i \in \{0, 1\}^{i\lambda}$ . We let  $c_k^0(x, \mathbf{r}) := x$ .

*Definition 4.1 (W-OTS<sup>+</sup> signature scheme).* The W-OTS<sup>+</sup> scheme consists of the following three efficient algorithms.

**KeyGen**( $1^\lambda$ )  $\rightarrow$  (sk, pk,  $\mathbf{r}, k$ ). The secret key sk = (sk<sub>1</sub>, ..., sk<sub>l</sub>) consists of  $l$  random bit strings. The randomization vector  $\mathbf{r}$  consists of  $w - 1$  random bit strings, i.e.,  $\mathbf{r} = (r_1, \dots, r_{w-1})$ . A key  $k \in \mathcal{K}$  is randomly sampled. The public verification key pk is computed as

$$\text{pk} := (\text{pk}_0, \text{pk}_1, \dots, \text{pk}_l) = ((\mathbf{r}, k), c_k^{w-1}(\text{sk}_1, \mathbf{r}), \dots, c_k^{w-1}(\text{sk}_l, \mathbf{r})). \quad (2)$$

**Sign**( $M, \text{sk}, \mathbf{r}$ )  $\rightarrow \sigma$ . The W-OTS<sup>+</sup> signature is

$$\sigma = (\sigma_1 \dots \sigma_l) = (c_k^{b_1}(\text{sk}_1, \mathbf{r}), \dots, c_k^{b_l}(\text{sk}_l, \mathbf{r})),$$

where  $b_i$  is the  $i$ th  $w$ -ary word of  $M||C = (b_1, \dots, b_l)$ .  $C$  is a

checksum computed as  $C := \sum_{i=1}^{l_1} (w - 1 - M_i)$ .  $C$ 's length in  $w$ -ary is at most  $l_2$ , as  $C \leq l_1 w$ .

**Vrfy**( $M, \sigma, \text{pk}$ )  $\rightarrow \{0, 1\}$ . The algorithm checks  $\forall i \in [1, l]$ :

$$\text{pk}_i \stackrel{?}{=} c_k^{w-1-b_i}(\sigma_i, \mathbf{r}). \quad (3)$$

If the comparison holds  $\forall i \in [1, l]$ , returns 1, otherwise 0.

| $w$               | 4    | 8    | 16   | 256  |
|-------------------|------|------|------|------|
| $l (= l_1 + l_2)$ | 136  | 90   | 67   | 34   |
| pk (in bytes)     | 4352 | 2880 | 2144 | 1088 |
| #Hash in Verify   | 272  | 360  | 536  | 4352 |

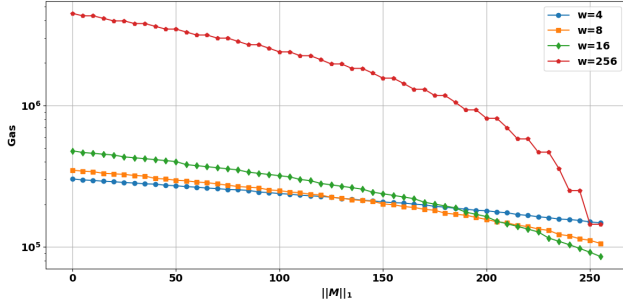
**Table 3: Public key size-hashing tradeoff in the W-OTS<sup>+</sup> scheme.** For a Winternitz parameter  $w$ , the public key pk consists of  $l$  elements of bit-length  $m (= 256)$ . The last row reports the average number of hashing operations (i.e.,  $\frac{l(w-1)}{2}$ ) in the verification algorithm. Shorter public keys incur an increased average number of hashing operations in the verification algorithm. As we shall see, the EVM favours larger public keys than an increased number of hashing operations.

*On-chain verification performance evaluation.* We observe in Equation (3) that the verification cost primarily depends on the Winternitz parameter  $w$ , the length  $l$  of the public key pk, and the signed message's Hamming weight  $\|M\|_1$ . Larger  $w$  yields shorter public keys at the cost of increased average number of hashing in the verification algorithm, see Table 3. Similarly, larger Hamming weight messages reduce the concrete computational complexity of the verification algorithm. This tradeoff is observed in Table 3 and we evaluate this tradeoff in Figure 2.

We apply the hash-and-sign paradigm, i.e., the signer signs  $H(M)$ . Thus, the signer could slightly decrease the verification gas costs by mauling the message in a way that results in a higher Hamming weight digest  $H(M)$ . This optimization was first observed in the blockchain context by Baldimtsi et al. [5]. Since  $\|H(M)\|_1$

| Mode \ w | 4       | 8       | 16      | 256       |
|----------|---------|---------|---------|-----------|
| On-chain | 222 114 | 223 478 | 272 355 | 1 971 090 |
| Naysayer | 174 448 | 139 821 | 126 095 | 237 908   |

**Table 4: W-OTS<sup>+</sup> average verification gas costs in the two verification modes for  $w \in \{4, 8, 16, 256\}$ . In each cell, we report weighted gas costs, where we weight the measured verification gas costs for all different Hamming weight messages, cf. Figure 2, by the probability that a uniformly random message  $m \in \mathcal{M} = \{0, 1\}^{256}$  has the given Hamming weight.**



**Figure 2: On-chain verification gas costs (note log-scaled y-axis) for W-OTS<sup>+</sup> for different Hamming weight messages  $\|M\|_1$  (x-axis) and Winternitz parameter  $w \in \{4, 8, 16, 256\}$ .**

is binomially distributed, i.e.,  $\|H(M)\|_1 \sim \text{Binom}(256, 0.5)$ , most messages will have  $\approx 100 - 150$  ones when signing 256-bit hash digests. More precisely,  $\mathbb{E}_M[\|H(M)\|_1] = 128$ . In this regime,  $w = 4$  is the optimal parameter choice. In the most common case of  $w = 4$ ,  $\|H(M)\|_1 = 128$ , we observe a 295 309 verification gas cost. Table 4 demonstrates that the W-OTS<sup>+</sup> on-chain verification is most gas-efficient for  $w = 4$ . As W-OTS<sup>+</sup> is a one-time signature scheme, the public key is not read from the permanent blockchain state; rather, it is sent to the contract as part of the CALldata, which is the cheapest (transient) storage. Therefore, the public key size does not significantly affect the on-chain verification cost. However, a larger  $w$  incurs more hashing operations, cf. Table 3. Put differently, hashing operations are the real bottleneck in W-OTS<sup>+</sup> on-chain verification. This phenomenon causes verification gas costs increase monotonically as a function of  $w$ .

**4.1.1 W-OTS<sup>+</sup> Naysayer verification mode.** We see in Equation (3) that checking the correctness of a W-OTS<sup>+</sup> signature  $\sigma = (\sigma_1, \dots, \sigma_l)$  needs to finish *all of the hash chains*, i.e.,  $\forall i \in [1, l]$ . However, if the W-OTS<sup>+</sup> signature is incorrect, then one only needs to check Equation (3) for a single index  $i$ . This observation lends itself to a simple Naysayer protocol. After signing, the signer stores  $\mathcal{MT}(\sigma, \text{pk}, M) = \text{com}_{\sigma, \text{pk}}$ . We assume that  $\sigma, \text{pk}, M$  are available to anyone for subsequent naysaying (e.g., they are stored at a data availability layer).

*The W-OTS<sup>+</sup> naysayer proof.* The W-OTS<sup>+</sup>  $\pi_{\text{nay}}$  contains all the necessary elements that allow a naysayer verifier to check Equation (3) for the faulty index  $i$ . Formally, let

$$\pi_{\text{nay}} := (\sigma_i, \text{pk}_i, \mathbf{r}, b_i, \pi_{(\sigma, \text{pk}), \text{pk}_i}, \pi_{(\sigma, \text{pk}), \sigma_i}, \pi_{(\sigma, \text{pk}), \mathbf{r}}, \pi_{(\sigma, \text{pk}), b_i}),$$

i.e., the naysayer prover opens  $\text{com}_{\sigma, \text{pk}}$ , and provides the contract with the faulty  $\sigma_i, \text{pk}_i, \mathbf{r}, b_i$  and accompanying Merkle proofs  $\pi_{(\sigma, \text{pk}), \text{pk}_i}, \pi_{(\sigma, \text{pk}), \sigma_i}, \pi_{(\sigma, \text{pk}), \mathbf{r}}, \pi_{(\sigma, \text{pk}), b_i}$  with respect to the Merkle commitment  $\text{com}_{\sigma, \text{pk}}$ .

*The W-OTS<sup>+</sup> naysayer verifier.* The contract verifies the Merkle opening proofs  $\pi_{(\sigma, \text{pk}), \text{pk}_i}, \pi_{(\sigma, \text{pk}), \sigma_i}, \pi_{(\sigma, \text{pk}), \mathbf{r}}, \pi_{(\sigma, \text{pk}), b_i}$  for  $\text{pk}_i, \sigma_i, \mathbf{r}, b_i$  with respect to  $\text{com}_{\sigma, \text{pk}}$ . Finally, the naysayer verifier checks for index  $i$ :  $\text{pk}_i \stackrel{?}{=} c_k^{w-1-b_i}(\sigma_i, \mathbf{r})$ . More formally:

$$\begin{aligned} & \mathcal{MT}.\text{Vrfy}(\text{com}_{\sigma, \text{pk}}, i, \text{pk}_i, \pi_{(\sigma, \text{pk}), \text{pk}_i}) \wedge \\ & \mathcal{MT}.\text{Vrfy}(\text{com}_{\sigma, \text{pk}}, i + l, \sigma_i, \pi_{(\sigma, \text{pk}), \sigma_i}) \wedge \\ & \mathcal{MT}.\text{Vrfy}(\text{com}_{\sigma, \text{pk}}, 0, \mathbf{r}, \pi_{(\sigma, \text{pk}), \mathbf{r}}) \wedge \\ & \mathcal{MT}.\text{Vrfy}(\text{com}_{\sigma, \text{pk}}, 2l + i, b_i, \pi_{(\sigma, \text{pk}), b_i}) \wedge \\ & \text{pk}_i \neq c_k^{w-1-b_i}(\sigma_i, \mathbf{r}) \end{aligned} \quad (4)$$

If all checks pass, the naysayer verifier outputs 0 (the W-OTS<sup>+</sup> signature  $\sigma$  is invalid) or  $\perp$  (inconclusive). We prove the security of the W-OTS<sup>+</sup> Naysayer proof system in Theorem B.1.

We evaluate and contrast the gas cost consumption of the Naysayer verifier with that of the full on-chain W-OTS<sup>+</sup> verifier in Table 4. We observe major cost savings when deploying the optimistic Naysayer verification, e.g., for  $w = 4$ , the Naysayer verifier shaves off 21.46% from the on-chain verification gas cost. Interestingly, the naysayer verifier is most efficient when  $w = 16$ . For larger  $w$ , the public key and the signature are shorter, resulting in smaller Merkle authentication paths in Equation (4). Although for  $w = 256$  we have the shortest signatures and public keys, the hash evaluations in checking  $\text{pk}_i \stackrel{?}{=} c_k^{w-1-b_i}(\sigma_i, \mathbf{r})$  dominates the Naysayer verifier cost rendering  $w = 16$  the optimal choice for the W-OTS<sup>+</sup>-Naysayer.

## 4.2 Extended Merkle signature scheme (XMSS)

The XMSS stateful hash-based signature scheme was introduced in 2011 by Buchmann, Dahmen, and Hülising [18] and is currently being standardized in the RFC8391 [31]. NIST has also approved specific variants of XMSS [39]. In the following, we are following the IETF XMSS standard's recommendations. XMSS only supports signatures for  $2^h$  messages, where  $h$  is the height of an XMSS tree  $\mathcal{XT}$ . An XMSS tree is a binary hash tree just like a Merkle tree but it also applies bit masks for deriving the parent nodes. More formally, the  $i$ th node  $n_{i,j}$  on the  $j$ th level of the tree (the root is on level  $h$ ) is obtained as:

$$n_{i,j} := H(n_{2i,j-1} \oplus b_{l,j} \| n_{2i+1,j-1} \oplus b_{r,j}), \quad (5)$$

using bitmasks  $(b_{l,j} || b_{r,j}) \in_R \{0, 1\}^{2^m}$ . Note that XMSS trees admit similar logarithmic-sized membership proofs (sometimes authentication paths) as those in Merkle trees.

The XMSS tree  $\mathcal{X}\mathcal{T}$  during key generation commits to  $2^h$  W-OTS<sup>+</sup> public keys  $\text{pk}_W := (\text{pk}_{W_1}, \dots, \text{pk}_{W_{2^h}})$ , and the tree's root  $\text{com}_{\text{pk}_W}$  is part of the XMSS scheme's public key. Furthermore, the  $i$ th XMSS tree leaf  $\text{pk}_{W_i}$  commits to the  $i$ th W-OTS<sup>+</sup> public keys' elements  $(\text{pk}_{W_{i0}}, \dots, \text{pk}_{W_{il}})$  using L-trees. The L-tree is a binary tree. Since  $l + 1$  might not be a power of two, a node that has no right sibling is lifted to a higher level of the L-tree until it becomes the right sibling of another node. In this construction, the same hash function as above but new bitmasks are used, see Equation (5). The bitmasks are the same for each of those  $2^h$  trees. As the L-trees have  $\lceil \log(l) \rceil$  height,  $\lceil \log(l) \rceil$  additional bitmasks are used. The XMSS public key  $\text{pk}$  contains the bitmasks  $\mathbf{b}$  and the root of the XMSS tree  $\text{com}_{\text{pk}_W}$ . Roughly, the XMSS signer for counter  $\text{cnt} = i$  provides a W-OTS<sup>+</sup> signature on a message  $M$  with  $\text{sk}_{W_i}$  and an authentication proof for  $\text{pk}_{W_i}$ , i.e., the  $i$ th W-OTS<sup>+</sup> public key. The verification entails checking the W-OTS<sup>+</sup> signature on  $M$ , recomputing  $\text{pk}'_{W_i}$  and comparing it to  $\text{pk}_{W_i}$ . Finally, the verifier checks the authentication path for the claimed  $\text{pk}_{W_i}$ .

Next, we recall the formal definition of the XMSS algorithms with some stylizing simplifications.

*Definition 4.2 (The XMSS algorithms).* The XMSS scheme consists of the following efficient algorithms.

**KeyGen**( $1^\lambda$ )  $\rightarrow$  (sk, pk). Let  $\text{sk} := \text{seed} \in_R \{0, 1\}^\lambda$  and generate  $\forall i \in [1, 2^h], j \in [0, l] : \text{pk}_{W_{ij}} := H(\text{seed} || i || j)$ . Let  $\text{pk}_{W_i} := \mathcal{L}([\text{pk}_{W_{i0}}, \dots, \text{pk}_{W_{il}}])$ . Similarly, let  $\text{com}_{\text{pk}_W} := \mathcal{X}\mathcal{T}([\text{pk}_{W_1}, \dots, \text{pk}_{W_{2^h}}])$ . Let  $\mathbf{b}$  denote the applied bitmasks. Finally,  $\text{pk} := (\text{com}_{\text{pk}_W}, \mathbf{b})$ . Return (sk, pk).

**Sign**( $M, \text{sk}, \text{cnt}$ )  $\rightarrow \sigma$ . The signature is  $\sigma = (i, \sigma_{W_i}, \pi_{\text{pk} \rightarrow \text{pk}_{W_i}})$ , where  $i$  is a counter,  $\sigma_{W_i}$  is a W-OTS<sup>+</sup> signature on  $M$  under  $\text{pk}_{W_i}$ , while  $\pi_{\text{pk}_W \rightarrow \text{pk}_{W_i}}$  is the XMSS authentication path for  $\text{pk}_{W_i}$ . Return  $\sigma$ .

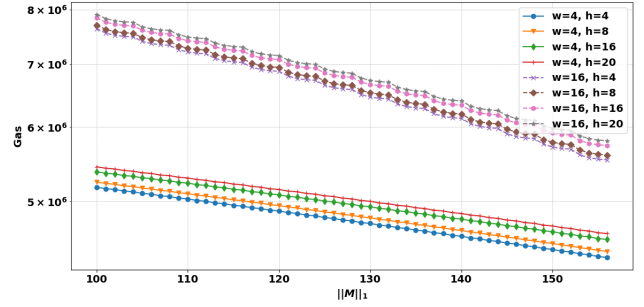
**Vrfy**( $M, \sigma, \text{pk}, \text{cnt}$ )  $\rightarrow \{0, 1\}$ . Parse  $\sigma$  as  $(i, \sigma_{W_i}, \pi_{\text{pk}_W \rightarrow \text{pk}_{W_i}})$ .

Compute  $\text{pk}_{W_{ij}} := c_k^{w-1-b_i}(\sigma_{W_i}[j], r)$ , where  $b_i$  is the  $j$ th w-ary word of  $M$ . The verifier recomputes the W-OTS<sup>+</sup> public key, i.e.,  $\text{pk}'_{W_i} := \mathcal{L}([\text{pk}_{W_{i0}}, \dots, \text{pk}_{W_{il}}])$  and checks

$$\begin{aligned} \text{cnt} &\stackrel{?}{=} i \wedge \text{pk}'_{W_i} \stackrel{?}{=} \pi_{\text{pk}_W \rightarrow \text{pk}_{W_i}}[0] \wedge \\ \mathcal{X}\mathcal{T}.\text{Vrfy}(\text{com}_{\text{pk}_W}, i, \text{pk}_{W_i}, \pi_{\text{pk}_W \rightarrow \text{pk}_{W_i}}) &\stackrel{?}{=} 1. \end{aligned} \quad (6)$$

If all the above checks are satisfied in Equation (6),  $\text{cnt} := \text{cnt} + 1$ , and return 1, otherwise 0.

*On-chain verification performance evaluation.* We evaluated the on-chain verification of XMSS for parameter sets  $(w, h) \in \{4, 16\} \times \{4, 8, 16, 20\}$ . The Winternitz parameters  $w = \{4, 16\}$  are recommended by the IETF XMSS standard. The choice of  $w$  is also supported by our measurements above in Section 4.1 as W-OTS<sup>+</sup> produces the most efficient verifications with Winternitz parameters  $w = \{4, 16\}$  for on-chain and Naysayer verifications, respectively. We chose  $h \in \{4, 8, 16, 20\}$  as a parameter set for  $h$  since it can support a wide range of real-world applications, i.e., max number



**Figure 3: On-chain verification gas costs (y-axis) for XMSS with messages of different Hamming weight (x-axis) for XMSS tree heights  $h \in \{4, 8, 16, 20\}$  and IETF-standardized Winternitz parameters  $w = 4$  and  $w = 16$ .**

of signatures<sup>5</sup> is up to  $2^{20}$ . Overall, we see two major trends in on-chain verification gas costs, see Figure 3.

First, XMSS is more efficient with Winternitz parameter  $w = 4$  than  $w = 16$  for all parameters  $h$  and  $\|H(M)\|_1$ , see Figure 3. The most efficient XMSS parameter set is  $(w, h) = (4, 4)$ , resulting in 4363623 gas (averaged out across all message Hamming weights). Second, an interesting trend for all the 8 measured  $(w, h)$  pairs is that the height of the Merkle tree has less effect on the verification gas cost than the message's Hamming weight. More precisely, for  $(w, h) = (4, 8)$  the verification gas cost is 5178583, while for  $(w, h) = (4, 20)$ , it is more expensive by 5.12%, i.e., 5443842 gas (both for messages of 100 Hamming weight). While  $(w, h) = (4, 4)$  and Hamming weight 156, we observe a 15.82% decrease compared to  $(w, h, \|M\|_1) = (4, 4, 100)$ , i.e., 4359189 gas. As in W-OTS<sup>+</sup>, we observe a linear relationship between  $\|M\|_1$  and verification gas cost, cf. Figure 3. We characterize this linear relationship using linear regression. We find that for  $(w, h) = (16, 20)$  the gas cost can be well approximated as  $7874574 - 37585x$  for Hamming weight  $x \in [0, 256]$ . Similarly, we found for  $(w, h) = (4, 20)$  a linear relationship  $5440598 - 14683x$ . Both the higher slope and intercept for  $w = 16$  can be explained by the larger W-OTS<sup>+</sup> public keys.

**4.2.1 XMSS Naysayer verification mode.** In our XMSS naysayer scheme, for the  $i$ th signature, the signer commits to the vector  $\mathbf{v} := (\{\sigma_{W_i}, \text{pk}_{W_{ij}}\}_{j=0}^l, \pi_{\text{pk}_W \rightarrow \text{pk}_{W_i}}^*)$ , i.e., computes  $\text{com}_{\mathbf{v}} := \mathcal{M}\mathcal{T}(\mathbf{v})$  and uploads  $\text{com}_{\mathbf{v}}$  to the blockchain. Note that  $\pi_{\text{pk}_W \rightarrow \text{pk}_{W_i}}^*$  is a modified authentication path that for each original authentication path element also contains its sibling node, see Figure 4. We explain the necessity of this later in Section 4.2.1. The verification of an XMSS signature  $\sigma = (i, \sigma_{W_i}, \pi_{\text{pk}_W \rightarrow \text{pk}_{W_i}})$  could fail due to one of the following three reasons.

- The W-OTS<sup>+</sup> signature  $\sigma_{W_i}$  is faulty, cf. Equation (3). This is naysayed precisely as in Section 4.1.1.
- The L-tree computation is faulty: the L-tree root of the W-OTS<sup>+</sup> public key does not match the one attached in  $\pi_{\text{pk}_W \rightarrow \text{pk}_{W_i}}^*$ . This fault is naysayed by opening  $\forall i \in [1, l] :$

<sup>5</sup>Note that in our evaluation, we disregard the key generation and signing algorithms' computational cost. For larger  $h$ , we observed significant XMSS key generation time.

| Mode \ $h$                             | 4         | 8         | 16        | 20        |
|----------------------------------------|-----------|-----------|-----------|-----------|
| On-chain ( $w=4$ )                     | 4 363 623 | 4 429 140 | 4 560 331 | 4 626 005 |
| On-chain ( $w=16$ )                    | 5 541 759 | 5 608 202 | 5 741 245 | 5 807 858 |
| Naysayer L-tree ( $w=4$ )              | 2 541 579 | 2 541 564 | 2 541 952 | 2 541 959 |
| Naysayer W-OTS <sup>+</sup> ( $w=4$ )  | 716 794   | 716 811   | 716 776   | 716 794   |
| Naysayer $\mathcal{XT}$ ( $w=4$ )      | 596 137   | 596 094   | 596 906   | 596 905   |
| Naysayer L-tree ( $w=16$ )             | 1 550 368 | 1 550 741 | 1 551 161 | 1 551 148 |
| Naysayer W-OTS <sup>+</sup> ( $w=16$ ) | 636 797   | 636 794   | 636 818   | 636 814   |
| Naysayer $\mathcal{XT}$ ( $w=16$ )     | 593 007   | 593 780   | 594 572   | 594 559   |

Table 5: XMSS on-chain and Naysayer verification gas costs averaged over all Hamming weight messages for Winternitz parameters  $w \in \{4, 16\}$ . The Naysayer types correspond to the three XMSS verification faults described in Section 4.2.1.

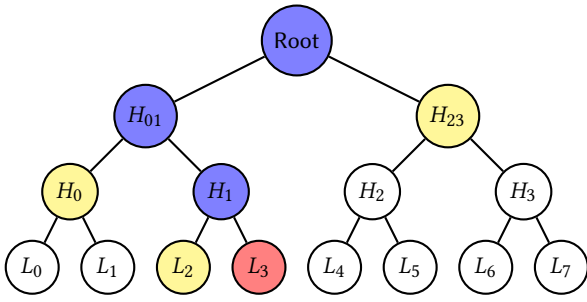


Figure 4: Naysayer proofs for (Merkle) tree authentication paths. For concreteness, we stick to Merkle trees but these observations also hold for L-trees or XMSS trees. In a Merkle tree, each node consists of the hash of its children. In this toy example, the prover committed to  $v := (v_0, v_1, \dots, v_7)$ , i.e., let  $L_i := H(v_i)$ . When the prover wants to show that  $v_3$  is committed in the Merkle tree at index 3, it provides the copath of the leaf  $L_3$  (in red), i.e.,  $\pi_{v,v_3} := (L_2, H_0, H_{23})$  (the yellow nodes in the tree). In the corresponding Naysayer proof system of the Merkle authentication path for  $v_3$ , the prover also includes the full verification trace, i.e., the blue nodes, i.e.,  $\{L_2, L_3, H_0, H_1, H_{01}, H_{23}, \text{Root}\}$ . Then  $\pi_{\text{nay}}$  would point to (one of) the verification trace’s failing hashing operation.

$\text{pk}_{W_i}$  public keys and the entire L-tree is recomputed. The L-tree’s root is compared to the one in stored in  $\pi_{\text{pk}_W \rightarrow \text{pk}_{W_i}}$ .

- The XMSS authentication path  $\pi_{\text{pk}_W \rightarrow \text{pk}_{W_i}}$  is faulty. Next, we detail how this can be naysayed in practice.

*Naysaying tree authentication paths.* The XMSS naysayer prover may need to naysay about the incorrectness of one of the authentication paths in a signature. Generally, we observe that for any tree-based authentication proof system, the prover should commit not only to the element  $v_i$  to be authenticated and its authentication path  $\pi_{v,v_i}$  but also to all the sibling nodes of the nodes in the authentication path  $\pi_{v,v_i}$ , i.e., the entire verification trace. This is illustrated in a toy example in Figure 4. We denote the set of augmented authentication path as  $\pi_{v,v_i}^*$ . Therefore, the prover Merkle-commits to all the elements in  $\pi_{v,v_i}^*$ , the authentication path

$\pi_{v,v_i}$  and its co-path and obtains Root. If the authentication path is faulty, then it means that a naysayer prover needs to open the Merkle commitment Root committing to the verification trace at three appropriately chosen places (i.e., where the hashing operation fails in the authentication path)  $v_i, v_j, v_k$  and show that

$$\begin{aligned}
& \mathcal{MT}.\text{Vrfy}(\text{com}_v, i, v_i, \pi_{v,v_i}) \\
& \wedge \mathcal{MT}.\text{Vrfy}(\text{com}_v, j, v_j, \pi_{v,v_j}) \\
& \wedge \mathcal{MT}.\text{Vrfy}(\text{com}_v, k, v_k, \pi_{v,v_k}) \\
& \wedge H(v_i \| v_j) \neq H(v_k).
\end{aligned} \tag{7}$$

Security of the XMSS Naysayer proof system is proven in Theorem B.2. On-chain verification costs, unsurprisingly, increase monotonically as a function of  $h$ ,  $w = 4$  being the more efficient choice for the Winternitz parameter. However, for the Naysayer verification mode,  $w = 16$  outperforms  $w = 4$ . This is because W-OTS<sup>+</sup> with  $w = 16$  has smaller public keys resulting in smaller L-trees. Among the three types of XMSS Naysayer proofs, recomputing the L-tree is the most expensive, while proving the incorrectness of an authentication path (cf. Equation (7)) in the XMSS tree is the most efficient. We observe that the height  $h$  of the XMSS tree has minimal impact on Naysayer’s verification gas costs. Finally, we note that XMSS Naysayer for every evaluated parameter is at least  $2.67\times$  less efficient than W-OTS<sup>+</sup> on-chain verification, see Table 4.

### 4.3 SPHINCS+

SPHINCS+ [10] is the only hash-based signature scheme recently standardized by NIST and, currently, the only NIST-standardized signature implemented in our **poqeth** library.

The SPHINCS+ signature is a tree-based stateless signature we briefly review. For further details, we refer to [10]. SPHINCS+ extends XMSS in the following sense: it builds a tree of trees, a so-called hypertree, cf. Figure 5. SPHINCS+ applies a binary tree where the W-OTS<sup>+</sup> public keys sign a few-time signature (FTS) scheme’s public key instead of the message directly, as in XMSS. Finally, the message is signed by the FORS FTS scheme. The FORS public key consists of  $k$  binary hash trees of height  $a$ . The FORS signature scheme can sign messages of  $k \cdot a$  bits. Each leaf node in a tree is used to sign the tree’s root below. For computational

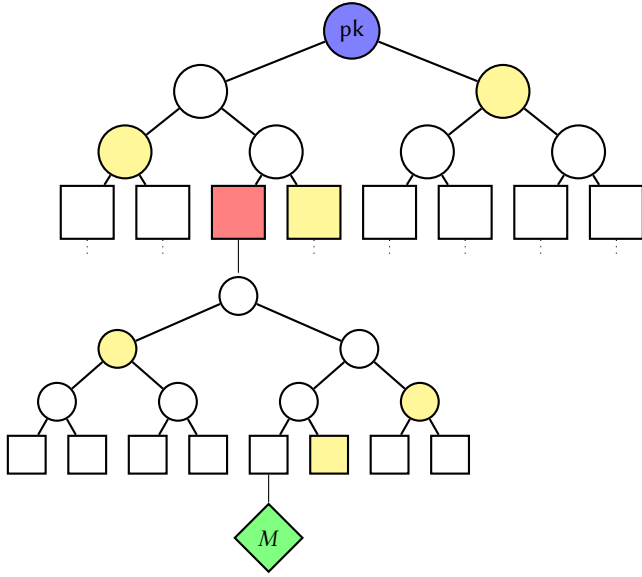


Figure 5: An illustration of a (small) SPHINCS structure for parameters  $(h, d) = (6, 2)$ . Circled nodes are hash nodes. Squared nodes represent OTS nodes (i.e., W-OTS<sup>+</sup>), and the diamond node denotes a few-time signature (FTS) node. The public key is the root of the hypertree. The message  $M$  is signed by the FORS FTS scheme. W-OTS<sup>+</sup> public keys are authenticated by including the yellow nodes in the signature.

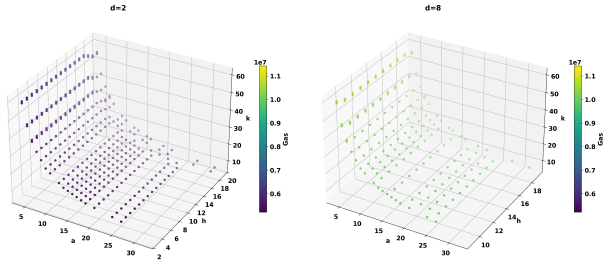


Figure 6: SPHINCS+ on-chain verification gas costs for fixed  $d$  parameters, i.e.,  $d \in \{2, 8\}$  and for varying  $h$  and  $a$  parameters.

efficiency, all leaf nodes of all intermediate trees are deterministically generated W-OTS<sup>+</sup> public keys that do not depend on any of the trees below them. Hence, the hypertree structure is purely virtual and never computed in full. During key generation, only the topmost subtree is calculated to derive the SPHINCS+ public key. Let  $h$  denote the height of the total tree, while the intermediate layers are denoted as  $d$ , and we set  $h' = \frac{h}{d}$ . For a given set of SPHINCS+ parameters  $(h, d, a, k)$  the bit length of the message that can be signed is computed as

$$m = \left\lfloor \frac{k \cdot a + 7}{8} \right\rfloor + \left\lfloor \frac{h - \left\lfloor \frac{h}{d} \right\rfloor + 7}{8} \right\rfloor + \left\lfloor \frac{\left\lfloor \frac{h}{d} \right\rfloor + 7}{8} \right\rfloor. \quad (8)$$

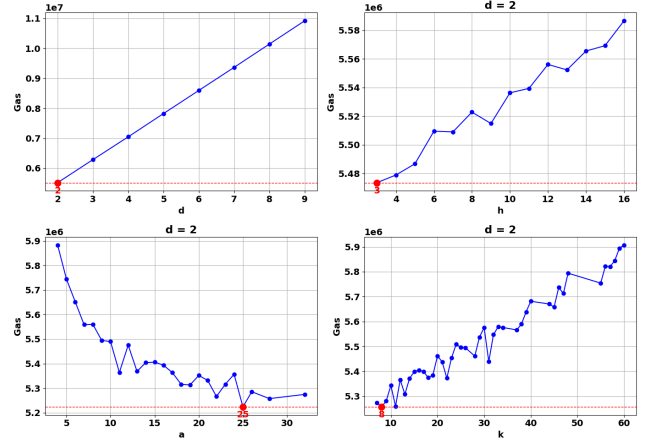


Figure 7: Impact of each SPHINCS+ parameter on on-chain verification costs when all other parameters remain constant.

We observe in Figure 7 that parameter  $d$  has the largest effect on verification gas costs when all other parameters remain fixed. For our SPHINCS+ verifier implementation to match the NIST security level 1 and to be able to sign  $m = 256$  bit messages, the most gas-efficient parameter set was  $(h, d, a, k) = (63, 10, 12, 15)$ . Applying the on-chain verification mode with the Winternitz parameters  $w = 4$  and 16, we measured the 11617690 and 13376297 gas, respectively. A more thorough analysis of the SPHINCS+ parameter space is conducted in Appendix E.

|                  | $w$        |            |
|------------------|------------|------------|
| Mode             | 4          | 16         |
| On-chain         | 11 617 690 | 13 376 297 |
| Naysay FORS      | 788 788    | 785 630    |
| Naysay FORS Hash | 1 360 472  | 1 343 606  |
| Naysay WOTS      | 1 233 214  | 1 238 559  |
| Naysay WOTS Hash | 1 451 263  | 1 310 754  |
| Naysay HT        | 693 721    | 690 587    |

Table 6: SPHINCS+ average verification gas costs for NIST security level 1, i.e., parameters  $(h, d, a, k) = (63, 10, 12, 15)$ .

*Naysaying SPHINCS+ signatures.* There are five places where the verification of SPHINCS+ signatures can fail. These errors correspond to five types of SPHINCS+ Naysayer proofs that we evaluate in Table 6. First, a hashing operation might fail at one of the intermediary nodes at one of the hypertrees. Second and third, W-OTS<sup>+</sup> or FORS signatures might be faulty. In the Merkle commitment, we commit to the hashes of W-OTS<sup>+</sup> and FORS public keys instead of the full public keys. The last two types of faults correspond to these hash computations being faulty. We prove the security of the SPHINCS+ Naysayer proof system in Theorem B.3.

## 5 MAYO: A multivariate-quadratic signature scheme in the EVM

Multivariate signature schemes usually follow a full domain hash approach based on surjective trapdoor one-way functions. Thus, creating a multivariate signature usually amounts to being able to generate a multivariate quadratic map  $P : \mathbb{F}_q^n \rightarrow \mathbb{F}_q^m$  with a trapdoor that allows the signer to find pre-images of  $P$ .

Patarin's Unbalanced Oil and Vinegar (UOV) [38] scheme is a classic example. Here, we briefly describe UOV using a formulation established in [12]. Let  $P : \mathbb{F}_q^n \rightarrow \mathbb{F}_q^m$  be a multivariate quadratic (MQ) map that vanishes on a secret linear space  $O$  of dimension  $m$  called the oil space. Then, knowing  $O$ , one can find pre-images of  $P$  in the following fashion. Assume we are looking for a pre-image of  $b \in \mathbb{F}_q^m$  and take a random vector  $r \in \mathbb{F}_q^n$ . We look for a pre-image in the form  $r + o$  where  $o \in O$ . Now, one has that

$$P(r + o) = P(r) + P(o) + P'(r, o)$$

where  $P'$  is the bilinear map associated to  $P$  (often called the polar form). Now when one tries to solve  $P(r + o) = b$ , then this reduces to solving the linear system  $P'(r, o) = b - P(r)$ , which by design has  $m$  equations and  $m$  variables. If this does not have a solution, then one samples a new  $r$ . Generating a system with an oil space as a trapdoor is relatively easy. One starts with a system where every quadratic form has a matrix with an  $m \times m$  bottom right corner with all 0 entries and then masks it with a secret linear transformation (note that the other entries of the initial matrices are also secret).

Even though UOV is an established scheme in the MQ literature, its public keys are large, especially in the context of the EVM. The main idea of MAYO is to utilize this formalism and start out with a system with a small oil space and then whip it up into a larger system with a larger oil space using a pseudorandom generator.

Let  $P : \mathbb{F}_q^n \rightarrow \mathbb{F}_q^m$  be a quadratic map. Let  $O$  be a linear subspace of dimension  $o$  such that for every  $u \in O$ , one has that  $P(u) = 0$ . Let  $E_{i,j}$  be a set of fixed matrices (that have the property that any non-trivial linear combination of them has rank  $m$ ), and then one can build the map  $P^* : \mathbb{F}_q^{kn} \rightarrow \mathbb{F}_q^m$  in the following fashion

$$P^*(x_1, \dots, x_k) = \sum_{i=1}^k E_{ii}P(x_i) + \sum_{i < j} E_{ij}P'(x_i, x_j).$$

Observe that  $\forall u \in O^k : P^*(u) = 0$ , implying that if  $ko > m$ , one can compute pre-images of any vector in the same fashion as in UOV. MAYO generates the public key in the following fashion to save on the size of the public key. Pick a random  $o \times (n - o)$  matrix  $O$  and let  $O$  be the row-space of the matrix  $(O|_O)$ . Now, every quadratic polynomial can be represented by an upper triangular matrix

$$\begin{pmatrix} P_i^{(1)} & P_i^{(2)} \\ 0 & P_i^{(3)} \end{pmatrix}$$

where  $P_i^{(1)}$  is a square matrix of size  $n - o$ ,  $P_i^{(3)}$  is a square matrix of size  $o$  and  $P_i^{(2)}$  is a rectangular matrix of size  $(n - o) \times o$ . Using a small random seed  $P_i^{(1)}$  and  $P_i^{(2)}$  can be generated pseudorandomly, and then, solving a linear equation, one can find a suitable  $P_i^{(1)}$  to ensure that the system vanishes on  $O$ . Next, we summarize the above discussion using the following definition.

*Definition 5.1 (The MAYO signature scheme).* MAYO takes as input system parameters  $q, n, m, o, k$  and the matrices  $E_{i,j} \in M_m(\mathbb{F}_q)$ .

**KeyGen**( $1^\lambda$ )  $\rightarrow$  (sk, pk). Generate a random oil space  $O$  of dimension  $o$ . seed  $\xleftarrow{\$} \{0, 1\}^\lambda$ . Using the seed, generate  $P_i^{(1)}$  and  $P_i^{(2)}$  for every  $i$  and compute  $P_i^{(3)}$  to ensure  $P = \begin{pmatrix} P_i^{(1)} & P_i^{(2)} \\ 0 & P_i^{(3)} \end{pmatrix}$  vanishes on  $O$ . Then sk = (seed,  $O$ ) and pk = (seed,  $P_i^{(3)}$ ).

**Sign**( $M$ , sk,  $r$ )  $\rightarrow \sigma$ . Compute  $P^*(x_1, \dots, x_k) = \sum_{i=1}^k E_{ii}P(x_i) + \sum_{i < j} E_{ij}P'(x_i, x_j)$ . Let  $s$  be a random salt and Compute  $r = H(M||s) \in \mathbb{F}_q^m$  for a hash function  $H : \{0, 1\}^* \rightarrow \mathbb{F}_q^m$ . Solve  $P^*(u) = r$  as discussed above. Return the signature  $\sigma = (s, u)$ .

**Vrfy**( $M$ ,  $\sigma$ , pk)  $\rightarrow \{0, 1\}$ . This algorithm verifies  $P^*(u) \stackrel{?}{=} H(M||s)$ . If the comparison holds, return 1; otherwise, 0.

*MAYO on-chain verification performance evaluation.* The two most important implementation decisions we had to make were choosing the finite field  $\mathbb{F}_q$  and, second, choosing the appropriate pseudorandom generator (PRG) for whipping up the MQ system that comprises the public key.

**The MAYO finite field  $\mathbb{F}_q$**  Many publicly available

open-source MAYO implementations<sup>6</sup> apply the choice  $q = 16$ , that is  $\mathbb{F}_q = GF(2^4)$ . We refrained from this choice as the EVM does not support finite field arithmetic natively in Galois fields with non-prime cardinality. Specifically, implementing  $\mathbb{F}_{16}$  arithmetic (especially multiplication) efficiently would have been challenging as it requires the storage of the multiplication table in memory for every signature verification. Therefore, we chose  $q = 31$  over  $q = 16$  because of its much more (gas-)efficient arithmetic (coming from the original MAYO paper [13]).

**An efficient PRG** Most MAYO deployments choose the AES-128-CTR symmetric cipher to whip the public key up to the final MQ system. Since the EVM does not support natively this cipher, we decided to apply the natively supported KECCAK-256 hash function for whipping the public key up.

*MAYO parameter selection.* Our implementation targets NIST security level 1. As mentioned in Section 1.2, we prefer parameter sets that result in shorter public keys even at the expense of larger signatures as signatures are only stored in cheap, temporary CALLDATA, while public keys must be stored and read from the expensive permanent storage. We only considered the MAYO variant without leakage (the original MAYO paper has versions with even smaller signatures but we chose parameter sets where one can prove that signatures leak no information). Thus, we implemented the MAYO signature scheme with the parameter set  $(q, m, n, o, k) = (31, 60, 62, 6, 10)$  that achieves NIST security level 1. We measured a 938 752 492 gas cost for verifying a single MAYO signature on-chain. We attribute this high gas cost to two main factors. First, as we do not store the expanded public key map ( $\approx 115$ KB) in

<sup>6</sup>See: <https://github.com/mjosaarinen/mayo-py>.

permanent storage,<sup>7</sup> the verifier smart contract needs to expand the entire public key map from the seed for every signature verification. Second, linear operations (e.g., polynomial evaluation) take up 84% of the gas cost. These linear operations could be significantly sped up if the EVM supported single instruction, multiple data (SIMD) operations. Future support for SIMD operations is under consideration and discussion [21]. Additionally, we acknowledge that our implementation is unoptimized as we store each 5-bit coefficient on an entire EVM word. Packing more  $\mathbb{F}_{31}$  elements into a single EVM word is a possible optimization we have not explored yet. Thus, we expect that gas cost for MAYO signature verification can be improved with a more careful EVM implementation.

*Naysaying MAYO signatures.* Looking at Definition 5.1, we see that the MAYO signature verification check  $P^*(u) \stackrel{?}{=} H(M||s)$  can fail if (at least) one row, i.e., a multivariate equation  $p_i(u)$  in  $P^*(u)$  is computed incorrectly. We expect significant speedups for the Naysayer verifier compared to on-chain verification as the Naysayer verifier needs to check only a single multivariate quadratic equation instead of checking all  $m$  of them ( $m = 68$  at NIST security level 1). Specifically, first, the signer commits to the vector  $\mathbf{v} := (\text{pk}, \sigma, M)$  and stores a Merkle commitment  $\text{com}_{\mathbf{v}} := \mathcal{MT}(\mathbf{v})$  on-chain.

If naysaying the MAYO signature is necessary, then for the  $i$ th ( $i \in [1, m]$ ) equation, the Naysayer prover opens the commitment  $\text{com}_{\mathbf{v}}$  by sending Merkle authentication paths to provide  $M, s, u$  and the coefficients of  $p_i(\cdot)$ . Let  $t_i$  denote the  $i$ th  $\lceil \log(q) \rceil$  word of  $H(M||s)$ . Then the Naysayer verifier checks  $p_i(u) \stackrel{?}{=} t_i$ . We prove the security of the MAYO Naysayer proof system in Theorem B.4. We found that the MAYO Naysayer verifier burns 107 634 064 gas, a major improvement to the on-chain MAYO verifier.

## 6 Naysayer proofs for more NIST PQ signature candidates

As we saw in Sections 4 and 5, at the time of writing, for PQ signatures, the Naysayer verification mode is significantly more efficient than on-chain verification in the EVM. Motivated by this, in this section, we describe systematically Naysayer proof systems for various NIST candidate PQ signature schemes. We focus on signature schemes that are still competing in the second round of the second NIST call for PQ signatures.<sup>8</sup> This section aims to provide *high-level ideas* of turning usual verification into Naysayer verification and to analyze which signatures might be best suited for the EVM. We leave the implementation and evaluation of the proposed Naysayer proof systems to future work.

Note that one of the stated goals of the additional signature call by NIST is to *standardize special-purpose signature schemes*, and we believe that being EVM-friendly could be useful information for the standardization process.

### 6.1 Code-based signatures

In NIST's second standardization round for PQ signatures, only LESS [15] and CROSS [4] remain as code-based signature schemes.

<sup>7</sup>Storing the entire MAYO public key map in permanent storage and reading it from there during signature verification would incur exorbitant gas costs.

<sup>8</sup>The complete list of competing PQ signatures can be found at <https://csrc.nist.gov/Projects/pqc-dig-sig/round-2-additional-signatures>.

**6.1.1 LESS.** LESS is based on the code equivalence problem. In this setting, one is given two linear codes  $C_0, C_1$  and their generator matrices  $G_0, G_1$ . One has to find an  $n \times n$  matrix  $Q$  such that  $G_1 = G_0Q$  and  $Q$  has precisely one non-zero element in every row and every column (such a matrix is called a monomial matrix).

The LESS scheme uses the usual framework to build a graph isomorphism-type identification scheme with 1-bit challenges and applies the Fiat-Shamir transform [28]. Every generator matrix for every code in the protocol is stored in reduced row echelon form. In one iteration of the interactive version of the protocol, one has that the verification equation is  $H(G_{\text{chal}}\text{resp}) \stackrel{?}{=} \text{commit}$ . When building a signature scheme using Fiat-Shamir the challenge is obtained as a hash of the commitment and the public key.

Thus, one has to do the following operations:

- Matrix multiplication, recomputing the challenge.
- Checking that  $\text{resp}$  is a monomial matrix.
- Compute the reduced row echelon form of  $G_{\text{chal}}\text{resp}$ .
- Checking the verification equation.

For a security parameter  $\lambda$ , this is executed  $\lambda$  times; for the Naysayer proof, it is enough to provide one faulty instance. The most costly step here is the reduced row echelon form computation. The signer could provide two transformation matrices  $A, B$  that transform  $G_{\text{chal}}\text{resp}$  into reduced row echelon form to speed this step up. Then what has to be checked is that  $AG_{\text{chal}}\text{resp}B$  is in row echelon form, which is significantly faster as it is enough to show a particular row of  $A$  and column of  $B$  that violates the reduced row echelon form. This shows that slightly increasing the signature size makes Naysayer verification significantly faster. The clear drawback of LESS is the large public key, which could be mitigated by only storing a hash of the public key and adding the actual public key to the signature (thus adding one extra hash to verification).

**6.1.2 CROSS.** CROSS is based on the restricted syndrome decoding problem where  $H$  is the parity check matrix of a linear code,  $H\mathbf{e} = \mathbf{y}$  and  $\mathbf{y}$  is given and one has to find  $\mathbf{e}$  restricted to a certain subgroup. Describing the entire CROSS protocol is more involved. Still, it is a 5-pass identification protocol where the last challenge is a bit (thus, similarly to LESS, one has to iterate  $\lambda$  times and use Fiat-Shamir). The verification check consists of<sup>9</sup>

- Two hashes, vector addition, matrix-vector multiplication.
- Application of a certain transformation  $\sigma \in G$  where  $G$  is some group. However, Section 3.11 of the aforementioned specification describes this as just a simple vector addition.

All these operations are fast. The Naysayer prover points to the incorrect iteration (out of  $\lambda$  iterations). CROSS has 38-byte public keys (for NIST level 1), which is suitable for Ethereum applications.

We describe additional Naysayer proof systems for more second round signature candidates in Appendix C.

## 7 Related work

As early as 2010, Nakamoto contemplated the potential consequences of quantum computing on Bitcoin's security in a BitcoinTalk post [43]. The state-of-the-art protocol to rescue Bitcoin in the face of a quantum computing threat was proposed by Stewart

<sup>9</sup>The verification check is described in detail in Figure 2, in the latest CROSS specification available at [https://www.cross-crypto.com/CROSS\\_Specification\\_v1.2.pdf](https://www.cross-crypto.com/CROSS_Specification_v1.2.pdf).

et al. [55], which was later improved in [35]. These protocols apply a slow commit-delay-reveal structure, where users spend their pre-quantum unspent transaction outputs (UTXOs) into a post-quantum secure UTXO. After a delay period, users can reveal (i.e., spend) their post-quantum public keys previously committed, accompanied by a PQ signature. Hence, transitioning a pre-quantum UTXO into a post-quantum secure one requires a Bitcoin transaction. Recently, the cost of transitioning all unspent bitcoins to PQ safe addresses à la Stewart et al. was estimated to last at least  $\approx 76.16$  days, *assuming no other transactions* are processed in Bitcoin, causing effectively a two-month-long downtime in Bitcoin [49].

Various applied cryptographic solutions were proposed to help the post-quantum transitioning of cryptocurrencies. Bonneau proposed a cryptocurrency design called Fawkescoin that only applies symmetric-key primitives [17]. Thus, he effectively showed that a quantum-secure cryptocurrency can be built with little to no overhead compared to cryptocurrencies using public-key primitives. Chaum et al. [20] extend the current cryptocurrency wallet design by creating an ECDSA-compatible signature scheme with a PQ secure fallback utilizing W-OTS<sup>+</sup>. Their technical invention is to use ECDSA secret keys and derive from those the W-OTS<sup>+</sup> public keys. Hence, if ECDSA is broken, W-OTS<sup>+</sup> could still be used to authenticate Bitcoin or other cryptocurrency transactions in a backwards-compatible manner. Giechaskiel et al. analyze Bitcoin’s security in the presence of broken cryptographic primitives [30].

Ethereum currently mandates each valid transaction to be signed by the pre-quantum ECDSA algorithm [57]. Account abstraction, a planned protocol upgrade, allows users to authenticate transactions by any signature scheme they choose [19]. These signatures will be verified by on-chain contracts and open the possibility of transitioning Ethereum to a post-quantum world. To our knowledge, there is no systematic study implementing and benchmarking the gas costs of verifying (standardized) PQ signatures in the EVM. Our work fills this gap. The only efforts we know of are EIP-7592 and EIP-7619 (Ethereum Improvement Proposal) and [2], which propose the addition of a precompile contract to the EVM that allows users to verify Falcon signatures [46, 48]. Allende et al. observe that verifying Falcon signatures in the EVM would cost 500 million gas on average [2]. Therefore, EIP-7592 and EIP-7619 solicit a protocol change that allows “natively” verifying Falcon signatures in the EVM by adding a new precompile contract. The protocol heavily subsidizes the gas costs of precompile contracts; i.e. if these EIPs are accepted, one could verify Falcon signatures on Ethereum for 1200 gas (less than the gas cost of verifying ECDSA signatures). However, as we argued in Section 1.2, our selected and benchmarked signature algorithms are better suited to the EVM than Falcon.

## 8 Conclusion and Future Directions

In this work, we implemented and evaluated **poqeth**, an open-source library that enables the efficient verification of PQ signatures on Ethereum, the currently most popular public blockchain. The technical discussions around the PQ transition of cryptocurrencies are still in their infancy, with many open questions remaining.

*Precompile contracts in the EVM for PQ signature verification.* This work shows that full verification of PQ signatures is moderately costly on Ethereum main net, see Table 7. Thus, a likely impact of

our paper is that the Ethereum community will amend the EVM to enable natively verifying PQ digital signature algorithms through precompile contracts. This approach would be similar to the one advocated in EIP-7619 for Falcon signatures [46]. However, the question of how PQ signature verification should be part of the EVM is open. Shall the EVM support precompile contracts for full signature verification? Or perhaps the EVM should stay modular and shall only provide precompiles for specific subroutines of these verification algorithms that form the bottleneck of the verification algorithms (e.g., verifying Merkle authentication paths, or evaluating polynomials, multivariate-quadratic equation systems over finite fields etc.). We leave these timely questions for future work.

|          | Sig. | W-OTS <sup>+</sup> | XMSS  | SPHINCS <sup>+</sup> | MAYO      |
|----------|------|--------------------|-------|----------------------|-----------|
| Mode     |      |                    |       |                      |           |
| On-chain |      | 2.76               | 54.31 | 144.58               | 12 411.01 |
| Naysayer |      | 1.57               | 7.40  | 8.63                 | 1339.48   |

**Table 7: On-chain vs. Naysayer verification USD costs with the most efficient parameter settings for the four considered signature schemes. Verification gas costs are taken from Table 2. ETH/USD exchange price and gas prices were taken on October 27, 2024; that is, at the time of writing, 1 ether was worth 2489 USD, while the median gas price was 5.08 Gwei.**

*Succinct proofs of signature verification.* A promising approach would be to prove the correct verification of PQ signatures instead of verifying them directly in the EVM. For example, a computationally powerful prover could collect numerous user transactions and the PQ signatures attached to them and issue a succinct PQ-secure STARK [8] proof attesting to the validity of all these signatures. (zk)STARKs and its many variants have already successfully been deployed on Ethereum in production. The EVM allows the efficient verification of STARK proofs ( $\approx 5$  million gas). Note that this gas cost could be amortized over multiple PQ signatures. We leave the evaluation and benchmarking of this approach with more standardized NIST PQ signatures for future work. A similar line of research has recently been initiated: see [23, 37] and references therein.

*Benchmarking more post-quantum signature algorithms.* Future work may evaluate and extend our benchmarks to other post-quantum signature schemes. In particular, following Naysayer proof systems described in Appendix C we expect that CROSS [4] and the MPC-in-the-head signatures will admit the most significant speed-ups using Naysayer proofs. Future work could implement these schemes and evaluate their efficiency.

*Naysayer prover/verifier efficiency tradeoffs.* Observe that the naysayer verifier efficiency can be highly optimized by committing to the entire EVM-level computation trace of signature verification. This observation would lead to highly efficient Naysayer verifiers, which only need to verify the incorrectness of a single EVM opcode. This strategy comes at the cost of increased signer time, as the signer must now commit to a much larger computation trace. We leave the evaluation of this optimization to future work.

**Acknowledgements.** We thank Noemi Glaeser and Antonio Sanso for insightful discussions on Naysayer proofs and PQ signatures. We are grateful to Alex Beregszászi for reviewing an earlier version of this paper. This research was supported by the Ministry of Culture and Innovation and the National Research, Development, and Innovation Office within the Quantum Information National Laboratory of Hungary (Grant No. 2022-2.1.1-NL-2022-00004) and by the grant “EXCELLENCE-151343”. Péter Kutas is supported by the János Bolyai Research Scholarship of the Hungarian Academy of Sciences and is also partly supported by EPSRC through grant number EP/V011324/1.

## References

- [1] Gora Adj, Stefano Barbero, Emanuele Bellini, Andre Esser, Luis Rivera-Zamarripa, Carlo Sanna, Javier Verbel, and Floyd Zweyding. 2024. MiRiH: Efficient Post-Quantum Signatures from MinRank in the Head. *IACR Transactions on Cryptographic Hardware and Embedded Systems* 2024, 2 (2024), 304–328.
- [2] Marcos Allende, Diego López León, Sergio Cerón, Adrián Pareja, Erick Pacheco, Antonio Leal, Marcelo Da Silva, Alejandro Pardo, Duncan Jones, David J Worrall, et al. 2023. Quantum-resistance in blockchain networks. *Scientific Reports* 13, 1 (2023), 5664.
- [3] Jean-Philippe Aumasson, Daniel J. Bernstein, Ward Beullens, Christoph Dobraunig, Maria Eichlseder, Scott Fluhrer, Stefan-Lukas Gazdag, Andreas Hülsing, Panos Kampanakis, Stefan Kölbl, Tanja Lange, Martin M. Lauridsen, Florian Mendel, Ruben Niederhagen, Christian Rechberger, Joost Rijneveld, Peter Schwabe, and Bas Westerbaan. [n. d.]. SPHINCS+ Submission to the NIST post-quantum project, v.3.1. <https://sphincs.org/data/sphincs+-r3.1-specification.pdf>.
- [4] Marco Baldi, Sebastian Bitzer, Alessio Pavoni, Paolo Santini, Antonia Wachter-Zeh, and Violetta Weger. 2024. Zero knowledge protocols and signatures from the restricted syndrome decoding problem. In *IACR International Conference on Public-Key Cryptography*. Springer, 243–274.
- [5] Foteini Baldimtsi, Konstantinos Chalkias, Panagiotis Chatzigiannis, and Mahimna Kelkar. 2022. Truncator: time-space tradeoff of cryptographic primitives. *Cryptology ePrint Archive* (2022).
- [6] Andrea Basso, Luca De Feo, Pierrick Dartois, Antonin Leroux, Luciano Maino, Giacomo Pope, Damien Robert, and Benjamin Wesolowski. 2024. SQISign2D-West: The Fast, the Small, and the Safer. *Cryptology ePrint Archive*, Paper 2024/760. <https://eprint.iacr.org/2024/760> <https://eprint.iacr.org/2024/760>.
- [7] Carsten Baum, Lennart Braun, Cyprien Delpech de Saint Guilhem, Michael Klooß, Christian Majenz, Shibam Mukherjee, Sebastian Ramacher, Christian Rechberger, Emmanuela Orsini, Lawrence Roy, et al. 2023. *FAEST: algorithm specifications*. Technical Report. Technical report, National Institute of Standards and Technology.
- [8] Eli Ben-Sasson, Iddo Bentov, Yimon Horeh, and Michael Riabzev. 2018. Scalable, transparent, and post-quantum secure computational integrity. *Cryptology ePrint Archive* (2018).
- [9] Ryad Benadjila, Thibault Feneuil, and Matthieu Rivain. 2024. MQ on my mind: Post-quantum signatures from the non-structured multivariate quadratic problem. In *2024 IEEE 9th European Symposium on Security and Privacy (EuroS&P)*. IEEE, 468–485.
- [10] Daniel J Bernstein, Andreas Hülsing, Stefan Kölbl, Ruben Niederhagen, Joost Rijneveld, and Peter Schwabe. 2019. The SPHINCS+ signature framework. In *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*. 2129–2146.
- [11] Slim Bettaieb, Loïc Bidoux, Victor Dyseryn, Andre Esser, Philippe Gaborit, Mukul Kulkarni, and Marco Palumbi. 2024. PERK: compact signature scheme based on a new variant of the permuted kernel problem. *Designs, Codes and Cryptography* (2024), 1–27.
- [12] Ward Beullens. 2021. Improved cryptanalysis of UOV and rainbow. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 348–373.
- [13] Ward Beullens. 2021. MAYO: practical post-quantum signatures from oil-and-vinegar maps. In *International Conference on Selected Areas in Cryptography*. Springer, 355–376.
- [14] Ward Beullens, Fabio Campos, Sofia Celi, Basil Hess, and Matthias J. Kannwischer. [n. d.]. MAYO Specification Document. <https://csrc.nist.gov/csrc/media/Projects/pqc-dig-sig/documents/round-1/spec-files/mayo-spec-web.pdf>.
- [15] Jean-François Biasse, Giacomo Micheli, Edoardo Persichetti, and Paolo Santini. 2020. LESS is more: code-based signatures without syndromes. In *Progress in Cryptology-AFRICACRYPT 2020: 12th International Conference on Cryptology in Africa, Cairo, Egypt, July 20–22, 2020, Proceedings 12*. Springer, 45–65.
- [16] Loïc Bidoux, Jesús-Javier Chi-Domínguez, Thibault Feneuil, Philippe Gaborit, Antoine Joux, Matthieu Rivain, and Adrien Vinçotte. 2025. RYDE: a digital signature scheme based on rank syndrome decoding problem with MPC-in-the-Head paradigm. *Designs, Codes and Cryptography* (2025), 1–36.
- [17] Joseph Bonneau. 2014. Fawkescoin: A cryptocurrency without public-key cryptography (transcript of discussion). In *Cambridge International Workshop on Security Protocols*. Springer, 359–370.
- [18] Johannes Buchmann, Erik Dahmen, and Andreas Hülsing. 2011. XMSS—a practical forward secure signature scheme based on minimal security assumptions. In *Post-Quantum Cryptography: 4th International Workshop, PQCrypto 2011, Taipei, Taiwan, November 29–December 2, 2011. Proceedings 4*. Springer, 117–129.
- [19] Vitalik Buterin, Yoav Weiss, Dror Tirosh, Shahaf Nacson, Alex Forshtat, Kristof Gazso, and Tjaden Hess. 2021. ERC-4337: account abstraction using alt mempool. Technical Report. Ethereum Improvement Proposals.
- [20] David Chaum, Mario Larangeira, Mario Yaksetig, and William Carter. 2021. W-OTS+ up my sleeve! a hidden secure fallback for cryptocurrency wallets. In *International Conference on Applied Cryptography and Network Security*. Springer, 195–219.
- [21] Greg Colvin. 2017. EIP-616: SIMD Operations for the EVM. <https://eips.ethereum.org/EIPS/eip-616> (2017).
- [22] Luca De Feo, David Kohel, Antonin Leroux, Christophe Petit, and Benjamin Wesolowski. 2020. SQISign: compact post-quantum signatures from quaternions and isogenies. In *Advances in Cryptology-ASIACRYPT 2020: 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7–11, 2020, Proceedings, Part I 26*. Springer, 64–93.
- [23] Justin Drake, Dmitry Khovratovich, Mikhail Kudinov, and Benedikt Wagner. 2025. Hash-Based Multi-Signatures for Post-Quantum Ethereum. *Cryptology ePrint Archive* (2025).
- [24] Léo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, Peter Schwabe, Gregor Seiler, and Damien Stehlé. 2018. Crystals-dilithium: A lattice-based digital signature scheme. *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2018), 238–268.
- [25] Léo Ducas, Eamonn W Postlethwaite, Ludo N Pulles, and Wessel van Woerden. 2022. Hawk: Module LIP makes lattice signatures fast, compact and simple. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 65–94.
- [26] Max Duparc and Tako Boris Fouotsa. 2024. SQIPrime: A dimension 2 variant of SQISignHD with non-smooth challenge isogenies. *Cryptology ePrint Archive*, Paper 2024/773. <https://eprint.iacr.org/2024/773> <https://eprint.iacr.org/2024/773>.
- [27] Thibault Feneuil, Antoine Joux, and Matthieu Rivain. 2022. Syndrome decoding in the head: Shorter signatures from zero-knowledge proofs. In *Annual International Cryptology Conference*. Springer, 541–572.
- [28] Amos Fiat and Adi Shamir. 1986. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the theory and application of cryptographic techniques*. Springer, 186–194.
- [29] Evangelos Georgiadis. 2019. How many transactions per second can bitcoin really handle? Theoretically. *Cryptology ePrint Archive* (2019).
- [30] Ilias Giechaskiel, Cas Cremers, and Kasper B Rasmussen. 2016. On bitcoin security in the presence of broken cryptographic primitives. In *Computer Security-ESORICS 2016: 21st European Symposium on Research in Computer Security, Heraklion, Greece, September 26–30, 2016, Proceedings, Part II 21*. Springer, 201–222.
- [31] A Huelsing, D Butin, S Gazdag, J Rijneveld, and A Mohaisen. 2018. RFC 8391: XMSS: eXtended Merkle Signature Scheme. <https://datatracker.ietf.org/doc/html/rfc8391>.
- [32] Andreas Hülsing. 2013. W-OTS+—shorter signatures for hash-based signature schemes. In *Progress in Cryptology-AFRICACRYPT 2013: 6th International Conference on Cryptology in Africa, Cairo, Egypt, June 22–24, 2013. Proceedings 6*. Springer, 173–188.
- [33] Andreas Hülsing, Denis Butin, Stefan Gazdag, Joost Rijneveld, and Aziz Mohaisen. 2018. XMSS: eXtended Merkle signature scheme. Technical Report.
- [34] Andreas Hülsing. 2017. WOTS+ – Shorter Signatures for Hash-Based Signature Schemes. *Cryptology ePrint Archive*, Paper 2017/965. <https://doi.org/10.1007/978-3-642-38553-7>.
- [35] Dragos I Ilie, William J Knottenbelt, and Iain D Stewart. 2020. Committing to quantum resistance, better: A speed-and-risk-configurable defence for bitcoin against a fast quantum computing attack. In *Mathematical Research for Blockchain Economy: 1st International Conference MARBLE 2019, Santorini, Greece*. Springer, 117–132.
- [36] Don Johnson, Alfred Menezes, and Scott Vanstone. 2001. The elliptic curve digital signature algorithm (ECDSA). *International journal of information security* 1 (2001), 36–63.
- [37] Irakliy Khaburzaniya, Konstantinos Chalkias, Kevin Lewi, and Harjasleen Malvai. 2022. Aggregating and thresholdizing hash-based signatures using STARKs. In *Proceedings of the 2022 ACM on Asia Conference on Computer and Communications Security*. 393–407.
- [38] Aviad Kipnis, Jacques Patarin, and Louis Goubin. 1999. Unbalanced oil and vinegar signature schemes. In *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 206–222.
- [39] Larry Marks and Evan Clendening. 2019. Stateful Hash-Based Signatures. *Public Comments on Misuse Resistance* (2019).

- [40] Dustin Moody. 2022. Status report on the third round of the NIST post-quantum cryptography standardization process. *NIST technical report* (2022).
- [41] Kohei Nakagawa and Hiroshi Onuki. 2024. SQIsign2D-East: A New Signature Scheme Using 2-dimensional Isogenies. *Cryptology ePrint Archive*, Paper 2024/771. <https://eprint.iacr.org/2024/771> <https://eprint.iacr.org/2024/771>.
- [42] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. *Satoshi Nakamoto* (2008).
- [43] Satoshi Nakamoto. 2010. Major Meltdown. In *Post on Bitcoin forum*, Vol. 2. 19. <https://bitcointalk.org/index.php?topic=202.msg2133#msg2133>
- [44] Christian Paquin, Douglas Stebila, and Goutam Tamvada. 2020. Benchmarking post-quantum cryptography in TLS. In *Post-Quantum Cryptography: 11th International Conference, PQCrypto 2020, Paris, France, April 15–17, 2020, Proceedings 11*. Springer, 72–91.
- [45] Jacques Patarin. 1997. The oil and vinegar signature scheme. In *Presented at the Dagstuhl Workshop on Cryptography September 1997*.
- [46] Erick Pacheco Pedraza. 2024. EIP-7619: Precompile Falcon512 generic verifier. Technical Report. Ethereum Improvement Proposals.
- [47] Daniel Perez and Benjamin Livshits. 2019. Broken metre: Attacking resource metering in EVM. *arXiv preprint arXiv:1909.07220* (2019).
- [48] Kuo Po-Chun, Cheng Chen-Mou, and Tam Chris. 2024. EIP-7592: Precompile for Falcon Signature Verification. Technical Report. Ethereum Improvement Proposals.
- [49] Jamie J Pont, Joseph J Kearney, Jack Moyler, and Carlos A Perez-Delgado. 2024. Downtime Required for Bitcoin Quantum-Safety. *arXiv preprint arXiv:2410.16965* (2024).
- [50] Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. 2020. Falcon. *Post-Quantum Cryptography Project of NIST* (2020).
- [51] Ronald L Rivest, Adi Shamir, and Leonard Adleman. 1978. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM* 21, 2 (1978), 120–126.
- [52] Claus-Peter Schnorr. 1990. Efficient identification and signatures for smart cards. In *Advances in Cryptology—CRYPTO’89 Proceedings 9*. Springer, 239–252.
- [53] Aaronson Scott. 2024. *Quantum Computing: Between Hope and Hype*. Technical Report. The University of Texas at Austin.
- [54] István András Seres, Noemi Glaeser, and Joseph Bonneau. 2023. Naysayer proofs. *Cryptology ePrint Archive* (2023).
- [55] Iain Stewart, Daniel Ilie, Alexei Zamyatin, Sam Werner, MF Torshizi, and William J Knottenbelt. 2018. Committing to quantum resistance: A slow defence for Bitcoin against a fast quantum computing attack. *Royal Society open science* 5, 6 (2018), 180410.
- [56] Anton Wahrstätter, Jens Ernstberger, Aviv Yaish, Liyi Zhou, Kaihua Qin, Taro Tsuchiya, Sebastian Steinhorst, Davor Svetinovic, Nicolas Christin, Mikolaj Barczeniewicz, et al. 2024. Blockchain censorship. In *Proceedings of the ACM on Web Conference 2024*. 1632–1643.
- [57] Gavin Wood et al. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper* 151, 2014 (2014), 1–32.

## A Extended Preliminaries

We recall the formal security definitions of a naysayer proof system.

**Definition A.1 (Naysayer correctness).** Given a proof system  $\Pi$ , a naysayer proof system  $\Pi_{\text{nay}}$  is *correct* if, for all honestly generated  $\text{crs}, \text{crs}_{\text{nay}}$ , all statements  $x$ , and all invalid proofs  $\pi$ , Naysay outputs a valid naysayer proof  $\pi_{\text{nay}}$ , that is, the following probability is 1:

$$\Pr \left[ \begin{array}{l} \text{VerifyNay}(\text{crs}_{\text{nay}}, (x, \pi), \pi_{\text{nay}}) \\ = 0 \end{array} \mid \begin{array}{l} (\text{crs}, \text{crs}_{\text{nay}}) \\ \vdash \text{NSetup}(1^\lambda, 1_{\text{nay}}^\lambda) \wedge \\ \text{Verify}(\text{crs}, x, \pi) = 0 \wedge \\ \pi_{\text{nay}} \\ \vdash \text{Naysay}(\text{crs}_{\text{nay}}, (x, \pi), \\ \text{Aux}_{\text{nay}}) \end{array} \right]$$

**Definition A.2 (Naysayer soundness).** Given a proof system  $\Pi$ , a naysayer proof system  $\Pi_{\text{nay}}$  is *sound* if, for all PPT adversaries  $\mathcal{A}$  and for all  $x, \text{Aux}$ , honestly generated  $\text{crs}, \text{crs}_{\text{nay}}$ , and correct proofs  $\pi$ ,  $\mathcal{A}$  produces a verifying naysayer proof  $\pi_{\text{nay}}$  with at most negligible probability, that is, the following probability is at most

negligible in the security parameter  $\lambda_{\text{nay}}$ :

$$\Pr \left[ \begin{array}{l} \text{VerifyNay}(\text{crs}_{\text{nay}}, (x, \pi), \pi_{\text{nay}}) = 0 \end{array} \mid \begin{array}{l} (\text{crs}, \text{crs}_{\text{nay}}) \\ \vdash \text{NSetup}(1^\lambda, 1_{\text{nay}}^\lambda) \wedge \\ \text{Verify}(\text{crs}, x, \pi) = 1 \wedge \\ \pi_{\text{nay}} \\ \vdash \mathcal{A}(\text{crs}_{\text{nay}}, (x, \pi), \\ \text{Aux}_{\text{nay}}) \end{array} \right]$$

## B Security proofs of the proposed Naysayer proof systems

In this section, we prove that all four proposed Naysayer proof systems for the studied signature schemes satisfy correctness (cf. Definition A.1), and soundness (cf. Definition A.2). Recall that in all our Naysayer proof systems the Merkle commitment to the signature and the accompanying auxiliary data, e.g., message, public key, other intermediate data necessary for verification, is computed by a trusted third party, i.e., an on-chain smart contract. Thus, we can trust the correctness of the on-chain com.

### B.1 W-OTS<sup>+</sup>

**THEOREM B.1 (W-OTS<sup>+</sup> NAYSAYER).** *The W-OTS<sup>+</sup> Naysayer proof system, cf. Section 4.1.1, satisfies correctness and soundness assuming the Merkle (vector) commitment scheme’s membership proofs correctness property and its position binding property.*

**Correctness.** If a W-OTS<sup>+</sup> signature  $\sigma$  committed by com is incorrect, then it means that the verification equation, cf. Equation (3), fails for at least one index in  $[1, l]$ . Let  $i \in [1, l]$  denote the smallest such index for which  $\text{pk}_i \neq c_k^{w-1-b_i}(\sigma_i, \mathbf{r})$ . Recall, the W-OTS<sup>+</sup> Naysayer proof is

$$\pi_{\text{nay}} = (\sigma_i, \text{pk}_i, \mathbf{r}, b_i, \pi_{(\sigma, \text{pk}), \text{pk}_i}, \pi_{(\sigma, \text{pk}), \sigma_i}, \pi_{(\sigma, \text{pk}), \mathbf{r}}, \pi_{(\sigma, \text{pk}), b_i})$$

That is, the W-OTS<sup>+</sup> Naysayer provides  $\sigma_i, \text{pk}_i, \mathbf{r}, b_i$  and accompanying Merkle authentication paths. Assuming the correctness of the Merkle authentication path proof system, the verifier always rejects invalid signatures, since by assumption for index  $i : \text{pk}_i \neq c_k^{w-1-b_i}(\sigma_i, \mathbf{r})$ .

**Soundness.** Naysayer soundness dictates that it should not be possible with non-negligible probability to submit a valid naysayer proof for a valid message, signature, public key triplet  $(m, \sigma, \text{pk})$ . Assume towards contradiction that there exist a triplet  $(m, \sigma, \text{pk})$  and a corresponding valid Naysayer proof  $\pi_{\text{nay}}$  for index  $i$ . By assumption of  $\sigma$  being a correct signature on  $m$  for  $\text{pk}$ , particularly, it also holds for index  $i$  that  $\text{pk}_i = c_k^{w-1-b_i}(\sigma_i, \mathbf{r})$ . Therefore, it must be the case that malicious W-OTS<sup>+</sup> naysayer managed to open the Merkle vector commitment to different values  $\text{pk}_i^*, \sigma_i^*$  and/or  $\mathbf{r}^*$  for which  $\text{pk}_i^* = c_k^{w-1-b_i}(\sigma_i^*, \mathbf{r}^*)$ . But this is a contradiction to the position-binding property of the Merkle vector commitment.

### B.2 XMSS

**THEOREM B.2 (XMSS NAYSAYER).** *The XMSS Naysayer proof system, cf. Section 4.2.1, satisfies correctness and soundness assuming the Merkle (vector) commitment scheme’s membership proofs correctness property and its position binding property.*

*Correctness.* XMSS signature verification can fail at three different places: for an XMSS signature  $\sigma_i$  corresponding to the counter  $i$ , the WOTS<sup>+</sup> signature  $\sigma_{W_i}$  may be invalid, the L-tree computation of the WOTS<sup>+</sup> public key  $\text{pk}_{W_i}$  may fail or the XMSS authentication path  $\pi_{\text{pk} \rightarrow \text{pk}_{W_i}}$  might be invalid. The correctness of the WOTS<sup>+</sup> Naysayer proof system has been proved in Appendix B.1. Therefore, we focus only on proving the Naysayer correctness of the latter two causes for an invalid XMSS signature. If  $\sigma_i$  is faulty because of an error in the L-tree computation then as outlined in Section 4.2.1, the XMSS Naysayer prover opens the Merkle commitment to  $\sigma_i$  at all indices corresponding to the L-tree elements and the on-chain Naysayer verifier recomputes the L-tree's root to obtain  $\text{pk}_{W_i}^*$ . The Naysayer prover must provide the contract with the  $i$ th public key  $\text{pk}_{W_i}$  and a correct Merkle authentication path  $\pi_{\text{pk} \rightarrow \text{pk}_{W_i}}$  for  $\text{pk}_{W_i}$ . This allegedly faulty  $\text{pk}_{W_i}^*$  can be compared to the public key  $\text{pk}_{W_i}$ . If the Merkle tree membership proofs satisfy correctness then for a faulty signature  $\sigma_i$  for which the L-tree computation is incorrect, it must be the case that  $\text{pk}_{W_i}^* \neq \text{pk}_{W_i}$ .

If the XMSS signature  $\sigma_i$  for counter  $i$  is invalid because the Merkle authentication  $\pi_{\text{pk} \rightarrow \text{pk}_{W_i}}$  of  $\sigma_i$  is invalid, then the Naysayer prover opens the commitment  $\text{com}_{\sigma_i}$  so that it provides a Naysayer proof consisting of an augmented authentication path for  $\text{pk}_{W_i}$  as illustrated in Figure 4. The augmented Merkle authentication path consists of two hashes at each level of the Merkle tree enabling the Naysayer verifier to verify the incorrectness of the Merkle authentication path.

*Soundness.* If  $\text{com}_{\sigma_i}$  is a Merkle vector commitment to a valid XMSS signature  $\sigma_i$ , and a malicious Naysayer prover could prove that  $\sigma_i$  is in fact an invalid signature, then it would imply that the malicious Naysayer prover managed to open the Merkle commitment to different values, hence, violating the position binding property of the Merkle tree.

### B.3 SPHINCS+

**THEOREM B.3 (SPHINCS+ NAYSAYER).** *The SPHINCS+ Naysayer proof system, cf. Section 4.3, satisfies correctness and soundness assuming the Merkle (vector) commitment scheme's membership proofs correctness property and its position binding property.*

*Correctness.* A SPHINCS+ signature might fail due to five different reasons, see Section 4.3. The correctness of the WOTS<sup>+</sup> Naysayer proof system has been shown in Appendix B.1. The remaining four failure cases are essentially proving the incorrectness of a Merkle authentication path whose Naysayer proof system has been shown to be correct in Appendix B.2.

*Soundness.* If a correct SPHINCS+ signature can be proven to be incorrect, then it means that the malicious Naysayer prover had managed to open the Merkle commitment to the SPHINCS+ signature to different values which contradicts to the position binding property of the Merkle vector commitment.

### B.4 MAYO

**THEOREM B.4 (MAYO NAYSAYER).** *The MAYO Naysayer proof system, cf. Section 5, satisfies correctness and soundness assuming the*

*Merkle (vector) commitment scheme's membership proofs correctness property and its position binding property.*

*Correctness.* Recall that in the MAYO Naysayer proof system the signer commits to the vector  $\mathbf{v} := (\text{pk}, \sigma, M)$  and stores the Merkle commitment  $\mathcal{MT}(\mathbf{v}) := \text{com}_{\mathbf{v}}$  on-chain. If a MAYO signature  $\sigma = (s, u)$  is an invalid signature, then it implies that there exists an index  $i$  for which the  $i$ th equation does not hold, i.e.,  $p_i(u) \neq H(M||s)$ . The Naysayer prover opens the Merkle commitment  $\text{com}_{\mathbf{v}}$  to provide the Naysayer verifier with the values  $\sigma (= (s, u))$ ,  $M$  and the coefficients of  $p_i(\cdot)$ . Due to the correctness of the Merkle authentication paths, the Naysayer verifier can check  $p_i(u) \stackrel{?}{=} H(M||s)$ .

*Soundness.* Assume towards contradiction that a correct MAYO signature  $\sigma$  is naysayed correctly with a  $\pi_{\text{nay}}$ . Then, it means that the Naysayer prover was able to prove that for some index  $i$  we have  $p_i(u) \neq H(M||s)$ . This is only possible if the malicious Naysayer prover managed to open the Merkle vector commitment for a different value, at least for one index. This contradicts the position binding property of the Merkle tree.

## C Additional Naysayer proof systems

### C.1 Multivariate signatures

Multivariate signature schemes follow a similar template as MAYO, cf. Section 5. Thus, the corresponding Naysayer proofs are similar in spirit. UOV is an interesting conservative choice [45], and verification here is slightly simpler than in MAYO, as there is no public key whipping procedure. However, the large public key makes it an unlikely candidate for Ethereum applications.

### C.2 SQIsign and Hawk

**C.2.1 SQIsign.** We believe SQIsign [22] could be well suited to the blockchain environment due to its small public key and signatures. The verification algorithm of SQIsign is rather costly (even with recent speed-ups in the newer versions [6, 26, 41]). However, it could potentially be more efficient using the Naysayer paradigm. One particular idea could be the following. In the identification system version of [22], the prover's response is a certain isogeny of degree  $2^k$ . The kernel of this isogeny is not rational, so it is cut up into chunks for which the kernel is rational. Thus, it can be represented by some elliptic curves and certain points on those curves. Finally, the verifier can check if the isogenies corresponding to those kernels are between the correct elliptic curves. To avoid forgeries, one also has to double-check some cyclicity conditions. One potential idea to turn SQIsign friendly for Naysayer proofs is to write the response isogeny as a chain of 2-isogenies. Thus, if a chain of isogenies is incorrect, a naysayer only has to prove that two curves are not 2-isogenous in the chain. This can be accomplished by evaluating a polynomial of degree 2 in two variables. If the cyclicity condition is not satisfied, one can simply submit a 2-torsion basis of a curve and check that the isogeny kills the 2-torsion, which requires two point doublings on elliptic curves. The drawback of this construction is that it slightly increases the signature size.

When applying Fiat-Shamir, one has to check the correctness of the challenge. This is usually rather straightforward; however, in SQIsign, the challenge is an isogeny that has to be recomputed from

a hash and checked during verification. Thus, to keep Naysayer verification fast, one has to do the same trick, namely writing the challenge isogeny into the signature as a chain of 2-isogenies. Here, there are two ways to cheat. One could use a fake chain of 2-isogenies, which can be checked similarly as discussed. The other is to use an isogeny whose kernel does not correspond to the one obtained by the hash in Fiat-Shamir. In this case, one has to recompute this isogeny until it diverges from the correct one.

In edge cases, this could lead to recomputing most of the challenge isogeny, which could be slow. Without this check, a forger observes an honest signature and can easily return the same signature for a different message (as the challenge would not be correct). However, for such an attack, the fake and the correct isogeny should diverge quite quickly since for a given message  $m_1$ , we have to find a message  $m_2$  such that output hashes are equal at many places. Thus, for most signatures, Naysayer verification remains fast.

**C.2.2 HAWK.** The HAWK signature scheme [25] relies on a structured version of the lattice isomorphism problem. However, this is only relevant to key generation as the verification algorithm is similar to FALCON [50]. Thus, there is no real need to discuss this separately, as the main benefit of HAWK versus Falcon is a simpler signing procedure, but that might not be so relevant in this context.

### C.3 MPC in-the-head

There are a large number of signatures following the MPC-in-the-head paradigm in the second round: MQOM [9], Mirath [1], SDitH [27], Ryde [16], FAEST [7] and PERK [11]. We do not deal with the specifics of all the schemes, but we describe the MPC-in-the-head paradigm, and a high-level method to build Naysayer proofs.

The general framework is that one is given a hard-to-invert function  $F$ ,  $x$  is secret, and  $F(x) = y$  is public. Again, an identification scheme is turned into a signature scheme using Fiat-Shamir. First, one can create  $n$  shares of the secret  $x$ , denoted by  $x_1, \dots, x_n$ . Then, one commits to these shares in the commitment phase. Furthermore, one takes an MPC protocol for the function  $F$  with inputs  $x_1, \dots, x_n$  where the parties locally compute value  $\alpha_i$  and then broadcast these values from which anyone can compute the shared function locally again. In the commitment phase, the prover sends these  $\alpha_i$  to the verifier. The challenge is an index  $i$ , and the prover responds with all the  $x_j$  where  $j \neq i$ .

The verifier checks if the commitments are correct and whether  $F(x)$  computed from the  $\alpha_i$  is indeed  $y$ . Completeness and zero knowledge of this protocol follow as the MPC protocol does not reveal any information on  $x_i$ , and the commitment hides  $x_i$  as well. One could cheat by designing an MPC protocol that outputs  $y$ , but for a single  $i$ , one of the commitments to  $x_i$  is not suitable. Thus, a malicious prover succeeds with probability  $1/n$ . In usual applications,  $n$  has to be chosen carefully. The larger  $n$ , the more costly the MPC protocol, but the identification protocol needs fewer iterations. Fewer iterations imply a smaller signature size. Thus, proposals usually try to balance size, signing and verification speed.

In a Naysayer proof, we only want to optimize for verification speed as public keys are small by design. Thus, the generic idea is to keep  $n$  small ( $n = 2$  or  $n = 3$  should suffice), as then if there is an error, the verification only has to run an MPC protocol for a low number of parties and check a low number of commitments.

## D Ethereum’s PQ migration cost

In this section, we provide loose lower bounds for the post-quantum migration costs in Ethereum, akin to those in [49] calculated for Bitcoin. At the time of writing (November 27th, 2024), there are  $\approx 210$  million unique externally owned accounts (EOA) on the Ethereum mainnet that had sent at least one transaction.<sup>10</sup> Since these addresses sent at least one valid transaction, their ECDSA public key is publicly exposed. Hence, in the presence of a plausible quantum computing threat, a quantum attacker could drain the cryptocurrency stored at these addresses. Thus, each of these Ethereum addresses must send a transaction to an address protected by a PQ secure public key. Currently, the block gas limit on Ethereum is 30 million gas, and a simple value transfer transaction costs 21000 gas. For the sake of simplicity, let us assume that a transaction with a PQ signature consumes 21000 gas as well. Therefore, at least 4 410 000 000 000 gas needs to be burnt to complete Ethereum’s PQ migration. This computation requires 147 000 full Ethereum blocks. At the time of writing, the block time interval in Ethereum is 12 seconds. Hence, Ethereum’s PQ transition would require at least  $\approx 20.42$  days of downtime, effectively. This downtime can be eased by trading off full blocks to a longer PQ transition period. Even with this approach, the Ethereum blockchain’s throughput would dramatically decrease.

## E Additional performance measurements for SPHINCS+

In this section, we provide additional measurements for the on-chain verification gas costs of SPHINCS+ for various parameter sets. Recall that a SPHINCS+ implementation needs to specify a parameter set  $(h, d, a, k)$ , where  $h$  denotes the total height of the SPHINCS+ hypertree,  $d$  denotes the number of intermediate layers, while  $a$  and  $k$  are parameters of the FORS signature scheme allowing us to sign messages with  $k \cdot a$  bits. Figure 8 depicts SPHINCS+ verification gas costs when one parameter is fixed, and the other vary for 256-bit messages.

<sup>10</sup>See: <https://dune.com/queries/2282489/3740508>.

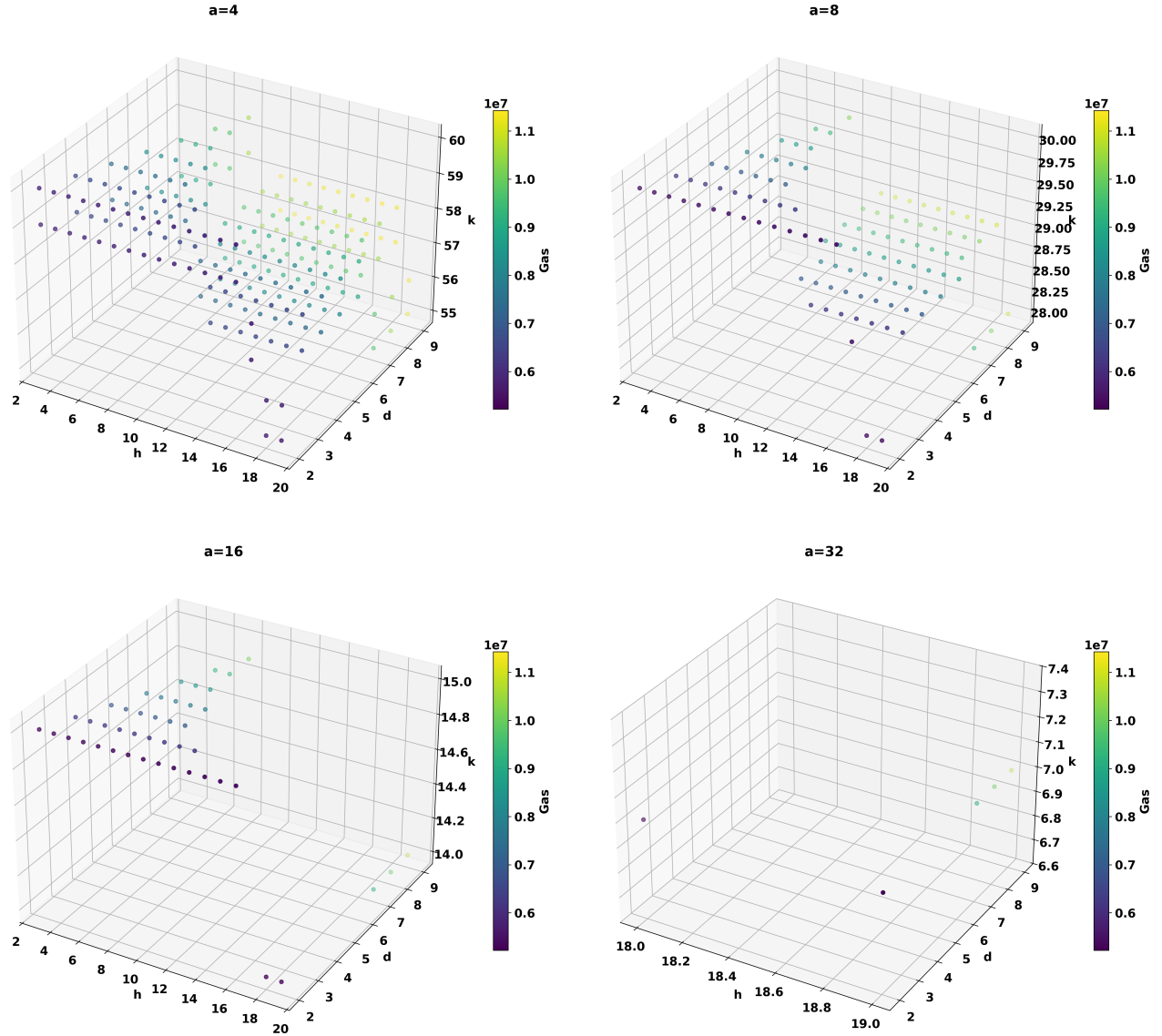


Figure 8: SPHINCS+ on-chain verification gas costs for fixed  $a$  parameters ( $a \in \{4, 8, 16, 32\}$ ) with varying  $h$  and  $d$  parameters. Note that we only evaluate the parameters that enable the signing of 256-bit messages, cf. Equation (8).