A hybrid post-quantum digital signature scheme for the Ethereum virtual machine

Jernej Tonejc[1] Naïri Usher[1]

[1]Flare Research
{jernej,nairi}@flare.network

July 5, 2022

Version 1.0

Abstract

The Ethereum Virtual Machine (EVM) relies on secure digital signatures in order to authenticate transactions. Today, these are implemented using elliptic curve cryptography. Quantum computers, and more precisely Shor's algorithm, threaten the integrity of elliptic curve digital signatures and thus endanger the security of the EVM. National security agencies advise the move to post quantum cryptography standards and schemes. We propose a modification to the EVM transaction types. Specifically, we introduce a new transaction type which in addition to the elliptic curve signature includes a post-quantum signature (deterministic CRYSTALS-Dilithium Level 2), together with the hash of both signatures. This enables quantum resistant digital signatures whilst preserving the speed and efficiency of the EVM.

1 Introduction

Classical cryptography relies on mathematical problems which current, classical computers cannot solve in order to achieve secure communication between various parties in presence of malicious adversaries. Its application to distributed networks via Transport Layer Security (TLS) has allowed for the development of the secure internet. More broadly, it underpins much of modern technology such as the TLS protocol, SSH and more. These rely on *public key cryptography*, which enables the generation of two keys, termed a public and a private key for communication. Examples of such schemes are Rivest–Shamir–Adleman (RSA) [1], and schemes based on elliptic curves (EC) [2], e.g. Elliptic-curve Diffie–Hellman (ECDH).

Yet, *quantum computers* are known to pose a serious and critical threat to cryptographic protocols and thus to the security of systems currently widely used [3]. A quantum computer is a computational device exploiting the laws of quantum physics, as opposed to classical physics. These have been subject of research for the past three decades and constitute an area of active research and development [4]. In 1981, Richard Feynman suggested harnessing quantum physics in order to simulate quantum systems [5]. Indeed, these are systems whose complexity is such that computing their properties on current classical computers is as yet intractable. Instead, the idea was to build devices with quantum systems as their building blocks in order to natively simulate the system of interest. Over the next decade, this idea was formalized until, in 1994, Shor's factorization algorithm provided an efficient Fourier transform [6]. This development meant that both RSA and EC cryptosystems were no longer secure, and that given access to a quantum computer, these could be compromised. Given their central role in cryptography, this would be a cryptographic nightmare.

Currently, no quantum computer capable of compromising internet security is available. Indeed, building a quantum computer is not a straightforward task. In 2001, 5 qubits could factor the number 15 [7]. Since then, academic institutions, established companies and start-ups seek to develop a quantum computer capable of producing meaningful output. Today, qubit counts of 127, 128 and 80 have been achieved respectively by IBM [8], MIT [9] and Rigetti [10]. In contrast, 20 million noisy qubits would be required for Shor's factorization algorithm to be applied to meaningful problem instances [11, 12].

Nonetheless, the critical role of both RSA and EC cryptography means that both governments and companies are actively looking for new standards. *Post-quantum (PQ) cryptography* refers to classical cryptography for which no efficient quantum algorithms have (as yet) been devised. The National Institute of Standards and Technology (NIST) has recognised the need for new standards of public key cryptography, and has launched a competition for the development of new post-quantum cryptography standards [13]. The competition has proceeded in rounds, yielding candidates for both key exchange algorithms and digital signatures. More generally, security agencies are calling for enterprises and organisations that develop and implement cryptographic products to prepare for migration to PQ standards [14, 15, 16].

A digital signature scheme is a fundamental cryptographic primitive that is used to protect authenticity and integrity of communication and fulfills a role akin to physical signatures. A pair of keys — one private and one public — is generated: the private (signing) key is used to sign the document, whereas the public key is available to anyone wishing to verify the signature and ascertain the author of the message. Traditionally, the public key is linked to the identity of the signer via certificates and their authenticity is guaranteed by a certificate authority, which must be trusted. In contrast, in a decentralised setting, such as for blockchains in general and EVM in particular, there is no trusted certificate authority that would link individual identities to the corresponding public keys. Instead, a public key is connected to the source address of a transaction via a hash (see Section 2.1.2 for details), thus providing a trustless proof of authenticity. In the case of EVM, the signature scheme (ECDSA) is used in a way that allows recovering the public key from the signature itself. This in turn allows the computation of the originating address of a transaction. The ownership of the funds at an address is thus linked directly to the ability to produce valid signatures for that address.

A digital signature scheme has five parameters that are relevant for the implementation: signature size, public key size, private key size, signature generation time and signature verification time. Ideally, novel post-quantum encryption standards would seamlessly (or almost) replace current standards, thereby minimising any disruption. However, the current candidates for post-quantum digital signatures suffer from trade-offs in terms of public/private key size, signature size, signature time and verification time, as summarised in Table 1. Depending on the context used, this could cause a significant issue.

The EVM currently implements digital signatures using an EC cryptography scheme, and as such, is vulnerable to quantum threat. Indeed, an attacker with access to a quantum computer could break the digital signature and impersonate someone, and thus for example drain funds. In the context of the EVM, the signature size must be small as every transaction must include it, whereas the verification must be fast as signatures are verified by nodes on the network. Hence, developing PQ digital signature schemes is an important and active area of current research (e.g. BPQS [18] is a patented scheme based on hashes of digital signatures and one-time signatures).

The remainder of the paper presents an update to the Ethereum transaction types. Its purpose is to upgrade the EVM to PQ signatures whilst preserving its efficiency. Section 2 gives

Name	Signature size	Public/Private key size	Verification time	Signature time
NIST P-256	64	64/32	1	1
RSA-2048	256	512/256	0.2	25
Dilithium2	1320	2420/2272	0.3	2.5
Falcon512	897	666/1280	0.3	5
Rainbow I	66	157800/101200	0.1	2.4
SPHINCS$^+$-128s	7856	32/64	1.7	3000
Picnic-L1-full	32061	34/17	21	60
GeMMS128	33	352190/13440	21	60

Table 1: Comparison of current digital signature schemes and NIST PQ candidates, [17]. The sizes are in bytes. Both signature and verification time are relative to NIST P-256, an Elliptic Curve Digital Signature Algorithm (ECDSA) with prime curve P-256. For RSA, the public exponent in practice needs only 3 bytes but could be as big as the modulus itself. We consider Level 1 and Level 2 parameters for the PQ schemes as these correspond to the classical security level of the NIST P-256 curve.

an overview of the existing EVM system. Section 3 introduces a new transaction type which in addition to the ECDSA signature includes a PQ signature (deterministic CRYSTALS-Dilithium Level 2 [19, 20]), together with the hash of both signatures. This hybrid approach is in line with suggested mandatory practices put forth by government agencies [16]. It furthermore enables a smooth transition, is backward compatible and ensures stronger security, as PQ signatures have not been scrutinised as much as their classical counterparts. Transactions of type 1, as introduced in Ethereum Improvement Proposal (EIP) 2930 [21] are further extended (note that legacy transactions are not considered, but could be if EIP-2930 is not widely adopted). This scheme allows an upgrade of the EVM to PQ digital signatures. Section 4 outlines the next steps.

2 Status Quo

In this section, the current EVM system is reviewed, with a focus on transaction types, representations, and crucially how these are signed and stored. Additionally, a summary of the current discussion on Ethereum Magicians' forum (a forum for the crypto community to discuss EIPs and technical difficulties of Ethereum ecosystem) and potentially related EIPs[1] is included.

2.1 Overview of EVM

This section is a short summary of the Ethereum Yellow Paper [22], introducing the notation that is relevant to the proposal. Throughout the proposal, bold symbols (e.g. $\mathbf{T}$) represent variables that can hold sets, lists, vectors, whereas the non-bold symbols represent scalar values or single objects. When a variable appears as an index of some other variable (e.g. T_{x})), it represents the corresponding property of the object (in the example given, the type of the transaction). The length of a list or the size of a set is denoted by $\|\cdot\|$ (e.g. since $B_{\mathbf{T}}$ denotes the transactions in block B, $\|B_{\mathbf{T}}\|$ is the number of transactions in block B). The concatenation of byte arrays is denoted by a dot $(\cdot)$. The encoding of an arbitrary object x is denoted by $\texttt{RLP}(x)$ and is done using the recursive length prefix encoding, as defined in

[1]Ethereum Improvement Protocol

Appendix B of the Ethereum Yellow Paper.

Ethereum, taken as a whole, can be viewed as a transaction-based state machine: we begin with a genesis state and incrementally execute transactions to morph it into some current state. A valid state transition is one which comes about through a transaction. Formally

$$\sigma_{t+1} \equiv \Upsilon(\sigma_t, T),$$

where Υ is the Ethereum state transition function that allows components to carry out arbitrary computation, T is a transaction, while σ allows components to store arbitrary state between transactions (σ_t corresponds to the state of the system after t successive transactions and σ_0 is the genesis state). Transactions are collated into blocks; blocks are chained together using a cryptographic hash as a means of reference.

2.1.1 Transactions

A **transaction** (formally, T) is a single cryptographically-signed instruction constructed by an actor externally to the scope of Ethereum. The sender of a transaction cannot be a contract. EIP-2718 [23] introduced the notion of different transaction types. As of the Berlin version of the protocol, there are two transaction types: 0 (legacy) and 1 (EIP-2930 [21]). Further, there are two subtypes of transactions: those which result in message calls and those which result in the creation of new accounts with associated code (known informally as 'contract creation'). All transaction types specify a number of common fields:

- **type**: EIP-2718 transaction type; formally T_{x}.
- **nonce**: A scalar value equal to the number of transactions sent by the sender; formally T_{n}.
- **gasPrice**: A scalar value equal to the number of Wei to be paid per unit of gas for all computation costs incurred as a result of the execution of this transaction; formally T_{p}.
- **gasLimit**: A scalar value equal to the maximum amount of gas that should be used in executing this transaction. This is paid up-front, before any computation is done and may not be increased later; formally T_{g}.
- **to:** The 160-bit address of the message call's recipient or, for a contract creation transaction, $\varnothing$, used here to denote the byte string of length 0; formally T_{t}.
- **value**: A scalar value equal to the number of Wei to be transferred to the message call's recipient or, in the case of contract creation, as an endowment to the newly created account; formally T_{v}.
- **r**, **s**: Values corresponding to the signature of the transaction and used to determine the sender of the transaction; formally T_{r} and T_{s}. See Appendix A for the definition.

EIP-2930 (type 1) transactions also have

- **accessList**: List of access entries to warm up (i.e. make readily accessible for the computation when executing the transaction); formally $T_{\mathbf{A}}$. Each access list entry E is a tuple of an account address and a list of storage keys: $E \equiv (E_{\mathrm{a}}, E_{\mathbf{s}})$.
- **chainId**: Chain ID; formally T_{c}. Must be equal to the network chain ID β.

- **yParity**: Signature Y parity; formally T_{y}. Recall that in an ECDSA signature, r is the X-coordinate of a point. To reconstruct the public key from the signature, the Y-coordinate is also needed. Since the equation of the curve is known, the parity of the Y-coordinate is enough to recover Y.

Legacy transactions do not have an **accessList** ($T_{\mathbf{A}} = ()$), while **chainId** and **yParity** for legacy transactions are combined into a single value

- **w**: A scalar value encoding Y parity and possibly chain ID; formally T_{w}. $T_{\mathrm{w}} = 27 + T_{\mathrm{y}}$ or $T_{\mathrm{w}} = 2\beta + 35 + T_{\mathrm{y}}$ (see EIP-155 [24]).

Additionally, contract creation transactions contain

- **init**: An unlimited size byte array specifying the EVM-code for the account initialisation procedure; formally $T_{\mathbf{i}}$.

In contrast, a message call transaction contains

- **data**: An unlimited size byte array specifying the input data of the message call; formally $T_{\mathbf{d}}$.

A given transaction T can thus be represented as

$$L_{\mathrm{T}}(T) = \begin{cases} \left[T_{\mathrm{n}}, T_{\mathrm{p}}, T_{\mathrm{g}}, T_{\mathrm{t}}, T_{\mathrm{v}}, \mathbf{p}, T_{\mathrm{w}}, T_{\mathrm{r}}, T_{\mathrm{s}}\right] & \text{if } T_{\mathrm{x}} = 0, \\ \left[T_{\mathrm{c}}, T_{\mathrm{n}}, T_{\mathrm{p}}, T_{\mathrm{g}}, T_{\mathrm{t}}, T_{\mathrm{v}}, \mathbf{p}, T_{\mathbf{A}}, T_{\mathrm{y}}, T_{\mathrm{r}}, T_{\mathrm{s}}\right] & \text{if } T_{\mathrm{x}} = 1, \end{cases} \tag{1}$$

where

$$\mathbf{p} = \begin{cases} T_{\mathbf{i}} & \text{if } T_{\mathrm{t}} = \varnothing, \\ T_{\mathbf{d}} & \text{otherwise.} \end{cases} \tag{2}$$

2.1.2 Signing transactions

Transactions are signed using recoverable ECDSA signatures, using the elliptic curve SECP-256k1. It is assumed that the sender has a valid private key p_{r}, which is a randomly selected positive integer (represented as a byte array of length 32 in big-endian form) in the range $[1, \texttt{secp256k1n} - 1]$. We furthermore assume the existence of functions `ECDSAPUBKEY`, `ECDSASIGN` and `ECDSARECOVER`

$$\texttt{ECDSAPUBKEY}(p_{\mathrm{r}}) \to p_{\mathrm{u}}, \tag{3}$$

$$\texttt{ECDSASIGN}(e, p_{\mathrm{r}}) \to (v, r, s), \tag{4}$$

$$\texttt{ECDSARECOVER}(e, v, r, s) \to p_{\mathrm{u}}, \tag{5}$$

where p_{u} is the public key, e is the hash $h(T)$ of the transaction (defined below), v is the recovery identifier (i.e. the Y parity from above). See Appendix A for the definition of these three functions.

For a given private key p_{r}, the corresponding Ethereum address $A(p_{\mathrm{r}})$ (a 160-bit value) is defined as the rightmost 160-bits of the Keccak-256 hash of the corresponding ECDSA public key

$$A(p_{\mathrm{r}}) = \texttt{KEC}\left(\texttt{ECDSAPUBKEY}(p_{\mathrm{r}})\right)_{96\ldots255}. \tag{6}$$

The message hash, $h(T)$, to be signed is the Keccak-256 hash of the transaction. As not all of the transaction fields get hashed, we introduce the function L_{X} that depends on the transaction type and returns the fields to hash:

$$L_X(T) = \begin{cases} (T_n, T_p, T_g, T_t, T_v, \mathbf{p}) & \text{if } T_x = 0 \text{ and } T_w \in \{27, 28\}, \\ (T_n, T_p, T_g, T_t, T_v, \mathbf{p}, \beta, (), ()) & \text{if } T_x = 0 \text{ and } T_w \in \{2\beta + 35, 2\beta + 36\}, \\ (T_c, T_n, T_p, T_g, T_t, T_v, \mathbf{p}, T_{\mathbf{A}}) & \text{if } T_x = 1, \end{cases} \tag{7}$$

With the help of this function we can define the hash of the transaction as

$$h(T) = \begin{cases} \mathtt{KEC}(\mathtt{RLP}(L_X(T))) & \text{if } T_x = 0, \\ \mathtt{KEC}(T_x \cdot \mathtt{RLP}(L_X(T))) & \text{if } T_x = 1. \end{cases} \tag{8}$$

The signed transaction $G(T, p_r)$ is the same as the transaction T itself, except with three fields updated as

$$(T_y, T_r, T_s) = \mathtt{ECDSASIGN}(h(T), p_r). \tag{9}$$

Note that the public key does not need to be included in the transaction as it can be recovered from the signature.

2.1.3 Blocks

The **block** in EVM consists of three components:

- a collection of relevant pieces of information (known as the block header), H,
- information corresponding to the comprised transactions, $\mathbf{T}$, and
- a set of other block headers $\mathbf{U}$ that are known to have a parent identical to the present block's parent's parent (such blocks are known as ommers).

The block header of a block B is denoted by B_H. The other two components in the block are simply a list of ommer block headers, denoted by $B_{\mathbf{U}}$ and a series of the transactions, denoted by $B_{\mathbf{T}}$.

The block header contains several pieces of information. For this proposal, the only relevant part is **transactionsRoot**, the Keccak 256-bit hash of the root node of the trie[2] structure populated with each transaction in the transaction list portion of the block; formally H_t, defined as

$$H_t = \mathtt{TRIE}(\{p_T(k, B_{\mathbf{T}}[k] : k = 0, 1, \ldots, \|B_{\mathbf{T}}\| - 1\}), \tag{10}$$

where

$$p_T(k, T) = \begin{cases} \big(\mathtt{RLP}(k), \mathtt{RLP}(L_T(T))\big), & \text{if } T_x = 0, \\ \big(\mathtt{RLP}(k), (T_x) \cdot \mathtt{RLP}(L_T(T))\big), & \text{if } T_x = 1. \end{cases} \tag{11}$$

Formally, the block B can be referred to as

$$B \equiv (B_H, B_{\mathbf{T}}, B_{\mathbf{U}}) \tag{12}$$

and its serialisation is

$$L_B(B) = \Big(L_H(B_H), \tilde{L}_T^*(B_{\mathbf{T}}), L_H^*(B_{\mathbf{U}})\Big), \tag{13}$$

where

$$\tilde{L}_T(T) = \begin{cases} L_T(T) & \text{if } T_x = 0, \\ (T_x) \cdot \mathtt{RLP}(L_T(T)) & \text{if } T_x = 1, \end{cases} \tag{14}$$

[2] A trie is a modified Merkle Patricia tree, see Appendix D in the Ethereum Yellow Paper [22].

and the star * denotes that the transformation is applied to each element of the list. The definition of L_{H} is identical to the definition in [22] and is not repeated here as it is not affected by this proposal.

A block is valid if and only if it satisfies several conditions: it must be internally consistent with the ommer and transaction block hashes and the given transactions $B_{\mathbf{T}}$, when executed in order on the base state σ (derived from the final state of the parent block), result in a new state as represented in the block header. It is assumed that any transactions that get executed first pass the initial tests of intrinsic validity. These include:

1. The transaction is well-formed RLP, with no additional trailing bytes;
2. *the transaction signature is valid;*
3. the transaction nonce is valid (equivalent to the sender account's current nonce);
4. the sender account has no contract code deployed;
5. the gas limit is no smaller than the intrinsic gas, g_0, used by the transaction; and
6. the sender account balance contains at least the cost, v_0, required in up-front payment.

For this proposal, only the meaning of point 2 needs to be changed.

2.2 Existing Proposals

2.2.1 Magician's forum posts

A search for *quantum* yields results hinting at the necessity of implementing PQ signatures, without offering any concrete solution.

The question of allowing users freedom in choosing their signatures has been discussed [25], illustrating that the community is aware of the importance to upgrade to PQ signatures. Yet, no concrete proposal was found. The current consensus seems to rely on NIST yielding a replacement with similar time and storage requirements. One approach to this is to introduce account abstraction to the EVM [26, 27]. The current system requires users to have an externally owned account (EOA), which allows for a specific transaction and signature type. In contrast, account abstraction would use smart contract wallets to store funds, thereby allowing for increased types of both transaction [28] and signatures to be used. However, implementing the contract wallets in a way that guarantees security is a non-trivial task and could potentially lead to security issues for other smart contracts since the assumptions on the caller are no longer identical across all possible contract wallets (as opposed to the EOAs, where the only assumption is that the caller is in the possession of the private key that corresponds to the account address).

Alternatively, ZK-STARKs [29] are known to be quantum resistant, and thus offer potential to be used for post quantum cryptography [30]. Nevertheless, they are not a drop-in replacement for ECDSA. Instead, the correctness of the state transition between blocks would be verifiable using this scheme, with potentially lower complexity than re-running all the transactions by each validator separately.

The last result [31] proposes to increase the address size, however, that does not address the problem of quantum computers breaking the ECDSA signatures.

2.2.2 Related EIPs

The EIPs mentioning the word *quantum* in a relevant sense are: EIP-101 [32], EIP-1011 [33], EIP-2333 [34], EIP-2938 [35] and EIP-4844 [36]. Of these, EIP-2333 introduces a key derivation scheme for BLS signatures with a Lamport signature as a one-off backup, and EIP-101, EIP-1011 and EIP-2938 introduce various versions of account abstraction. None offer a concrete solution to the quantum threat. EIP-4844 introduces a way of including additional data into transactions which does not get added to the blockchain. The proposal does not mention post-quantum signatures, and instead discusses using versioned hashes that allow moving to a new version without having to break the compatibility on the chain (and mentions quantum-safety as one of the reasons for moving to a new version). The idea of having some data as part of the transaction that gets later discarded is similar to our proposal.

3 PQ Proposal

In this section, the quantum resistant proposal is presented, introducing a new transaction type requiring both an ECDSA signature as well as a PQ signature and discussing the signing and storage of transactions. Secondly, the required code modifications are discussed as well as a fallback scheme, and a comparison of both resources.

3.1 Proposal

Current guidelines for industry from national security agencies suggest adopting a hybrid approach to public key cryptography, that is, new schemes encompassing both a classical scheme such as ECDSA and RSA, as well as an algorithm believed to be quantum resistant. Given the increase in resource requirements of quantum schemes, the additional cost of a hybrid scheme is relatively low whilst guaranteeing security at least as good as today.

From Table 1 it follows that most PQ digital signature schemes are not appropriate for EVM due to the large signatures or keys or long signature and verification times. The only two schemes that offer reasonable size and performance are Dilithium2 and Falcon512. For this proposal we choose Dilithium2, specifically, *deterministic CRYSTALS-Dilithium Level 2* signature scheme (dCDL2) [19, 20]. dCDL2 is chosen over Falcon as it is easier to correctly implement and is less susceptible to side channel attacks [37].

We introduce a new transaction type 'Q' (decimal 81, hexadecimal `0x51`) that includes an ECDSA signature and a PQ signature. Transactions of this type have several additional fields in comparison to type 1 transactions. First, it contains fields related to the PQ signature scheme i.e. to the dCDL2 signature:

- **pqSignC**, **pqSignZ**, **pqSignH**: Values corresponding to the dCDL2 signature of the transaction; formally T_{qc}, $T_{\mathbf{qz}}$ and $T_{\mathbf{qh}}$,

and to the dCDL2 public key:

- **pqKeyRho**, **pqKeyT**: The public key for dCDL2; formally T_{qr} and $T_{\mathbf{qt}}$.

Second, a joint hash of the ECDSA and dCDL2 signatures that allows for a compact storage of the transactions within the blocks:

- **signHash**: The hash of the ECDSA and dCDL2 signature; formally T_{h},

where signature hash T_{h} for a type 81 transaction is computed as

$$T_{\text{h}} = \texttt{KEC}(\texttt{RLP}((T_{\text{y}}, T_{\text{r}}, T_{\text{s}}, T_{\text{qc}}, T_{\mathbf{qz}}, T_{\mathbf{qh}}))). \tag{15}$$

3.1.1 Transaction representation for submission and storage

We modify the representation equ. (1) of a transaction as used to store the transaction in a block by adding a new case corresponding to the post-quantum (i.e. $T_{\mathrm{x}} = 81$) case:

$$L_{\mathrm{T}}(T) = \begin{cases} \left[T_{\mathrm{n}}, T_{\mathrm{p}}, T_{\mathrm{g}}, T_{\mathrm{t}}, T_{\mathrm{v}}, \mathbf{p}, T_{\mathrm{w}}, T_{\mathrm{r}}, T_{\mathrm{s}}\right] & \text{if } T_{\mathrm{x}} = 0, \\ \left[T_{\mathrm{c}}, T_{\mathrm{n}}, T_{\mathrm{p}}, T_{\mathrm{g}}, T_{\mathrm{t}}, T_{\mathrm{v}}, \mathbf{p}, T_{\mathbf{A}}, T_{\mathrm{y}}, T_{\mathrm{r}}, T_{\mathrm{s}}\right] & \text{if } T_{\mathrm{x}} = 1, \\ \left[T_{\mathrm{c}}, T_{\mathrm{n}}, T_{\mathrm{p}}, T_{\mathrm{g}}, T_{\mathrm{t}}, T_{\mathrm{v}}, \mathbf{p}, T_{\mathbf{A}}, T_{\mathrm{y}}, T_{\mathrm{r}}, T_{\mathrm{s}}, T_{\mathrm{h}}\right] & \text{if } T_{\mathrm{x}} = 81. \end{cases} \tag{16}$$

Note that the difference between type 1 and type 81 transactions is the addition of a single 32 byte field T_{h}.

The definition of the function $L_{\mathrm{X}}(T)$ in equ. (7), representing the transaction data that gets signed, is expanded to cover the case $T_{\mathrm{x}} = 81$

$$L_{\mathrm{X}}(T) = \begin{cases} (T_{\mathrm{n}}, T_{\mathrm{p}}, T_{\mathrm{g}}, T_{\mathrm{t}}, T_{\mathrm{v}}, \mathbf{p}) & \text{if } T_{\mathrm{x}} = 0 \text{ and } T_{\mathrm{w}} \in \{27, 28\}, \\ (T_{\mathrm{n}}, T_{\mathrm{p}}, T_{\mathrm{g}}, T_{\mathrm{t}}, T_{\mathrm{v}}, \mathbf{p}, \beta, (), ()) & \text{if } T_{\mathrm{x}} = 0 \text{ and } T_{\mathrm{w}} \in \{2\beta + 35, 2\beta + 36\}, \\ (T_{\mathrm{c}}, T_{\mathrm{n}}, T_{\mathrm{p}}, T_{\mathrm{g}}, T_{\mathrm{t}}, T_{\mathrm{v}}, \mathbf{p}, T_{\mathbf{A}}) & \text{if } T_{\mathrm{x}} \in \{1, 81\}. \end{cases} \tag{17}$$

We introduce a new function $L_{\mathrm{Q}}(T)$ that encodes the transaction for submission to the network

$$L_{\mathrm{Q}}(T) = \begin{cases} L_{\mathrm{T}}(T) & \text{if } T_{\mathrm{x}} \in \{0, 1\}, \\ L_{\mathrm{T}}(T) \cdot \left[T_{\mathrm{qc}}, T_{\mathbf{qz}}, T_{\mathbf{qh}}, T_{\mathrm{qr}}, T_{\mathbf{qt}}\right] & \text{if } T_{\mathrm{x}} = 81. \end{cases} \tag{18}$$

Observe that the transaction submitted to the network is identical to the one later stored within the block for type 0 and type 1 transactions, but differs for type 81: the submitted transaction additionally includes the dCDL2 signature and the public key

$$(T_{\mathrm{qc}}, T_{\mathbf{qz}}, T_{\mathbf{qh}}, T_{\mathrm{qr}}, T_{\mathbf{qt}}). \tag{19}$$

Since ECDSA signatures have a recoverable public key property, the ECDSA signature itself is enough to derive the sender's address. For the dCDL2 signatures, the public key cannot be derived from the signature itself. Therefore transactions of type 81 have to include the public key as well as the signature when submitted to the network. The reason for not including these fields when storing the transactions in blocks is their total size and is explained in detail in section 3.4.

We assume the existence of functions PQCDPUBKEY and PQCDSIGN

$$\texttt{PQCDPUBKEY}(p_\zeta) \rightarrow (p_\rho, p_{\mathbf{t_1}}), \tag{20}$$
$$\texttt{PQCDSIGN}(e, p_\zeta) \rightarrow (\tilde{c}, \mathbf{z}, \mathbf{h}), \tag{21}$$

where $(p_\rho, p_{\mathbf{t_1}})$ is the dCDL2 public key, p_ζ is the dCDL2 private key, e is the hash $h(T)$ of the transaction (as defined in equ. (8)). See Appendix B for the definition of these functions. We define functions PUBKEY, SIGN and RECOVER as a combination of the corresponding ECDSA and dCDL2 functions:

$$\texttt{PUBKEY}(p_{\mathrm{r}}, p_\zeta) \rightarrow (\texttt{ECDSAPUBKEY}(p_{\mathrm{r}}), \texttt{PQCDPUBKEY}(p_\zeta)), \tag{22}$$
$$\texttt{SIGN}(e, p_{\mathrm{r}}, p_\zeta) \rightarrow (\texttt{ECDSASIGN}(e, p_{\mathrm{r}}), \texttt{PQCDSIGN}(e, p_\zeta)), \tag{23}$$
$$\texttt{RECOVER}(e, v, r, s, p_\rho, p_{\mathbf{t_1}}) \rightarrow (\texttt{ECDSARECOVER}(e, v, r, s), (p_\rho, p_{\mathbf{t_1}})). \tag{24}$$

For a given hybrid private key (p_r, p_ζ), the corresponding post-quantum-Ethereum address, denoted by $A(p_\mathrm{r}, p_\zeta)$ (a 160-bit value), is defined as the rightmost 160-bits of the Keccak-256 hash of the corresponding hybrid public key:

$$A(p_\mathrm{r}, p_\zeta) = \mathtt{KEC}\left(\mathtt{PUBKEY}(p_\mathrm{r}, p_\zeta)\right)_{96\ldots255}. \tag{25}$$

We redefine the signed transaction $G(T, (p_\mathrm{r}, p_\zeta))$ as the transaction T with six fields updated as

$$((T_\mathrm{y}, T_\mathrm{r}, T_\mathrm{s}), (T_\mathrm{qc}, T_\mathbf{qz}, T_\mathbf{qh})) = \mathtt{SIGN}(h(T), p_\mathrm{r}, p_\zeta). \tag{26}$$

3.1.2 Storage rationale

The additional transaction information in equ. (19) can be held until the finality is reached (e.g., 6-10 blocks). After that, only the **signHash** data is stored on the blockchain in addition to the data that is stored for transactions of type 0 or 1.

This saves space (as the dCDL2 signature and public key are significantly larger than the ECDSA signature) and offers the same level of security as the transactions cannot be modified due to the chaining of the block hashes. This is equivalent to the fact that revealing the private key of some past transaction cannot be used to revert that transaction, so the signature itself does not need to be stored permanently (it is only needed to validate the transaction before it is included in a block). Furthermore, since dCDL2 is the deterministic version of the CRYSTALS-Dilithium signature scheme, the owner of the corresponding private key can always reproduce the signature for an older transaction, thus reconstructing the whole transaction information $L_\mathrm{Q}(T)$ sent to the network, based only on the information stored on the blockchain. Due to this property, the legitimate sender can always prove the validity of a past transaction.

3.2 Code Modifications

In order to support the new transaction type, the EVM code as well as the client code must be updated. On the EVM side, the second initial test of intrinsic validity of a transaction has to be updated to take both ECDSA signature as well as the dCDL2 signature into account. In addition, the **signHash** field T_h has to be validated. The serialisation of the transactions for the inclusion into the Merkle tree also has to be done as described in equ. (16).

The clients (wallets) need to be able to submit the new transaction type. In particular, they must implement function $L_\mathrm{Q}(T)$ as defined by equ. (18) and the PQ signature algorithm CRYSTALS-Dilithium Level 2. In addition, they have to be able to compute the new address as defined in equ. (25).

Due to the difference between the submitted transaction data $L_\mathrm{Q}(T)$ and the stored transaction data $L_\mathrm{T}(T)$, the full nodes need a way of discarding the extra information (19) that is present in $L_\mathrm{Q}(T)$, as soon as a transaction is considered confirmed. The extra information cannot be discarded earlier as the transaction might need to be included in some later block in case of reorder. This pruning of the information can be done automatically as new blocks are added (e.g. pruning the blocks that are at $\mathtt{block\ height} - 10$) or periodically as a form of garbage collection (e.g. daily).

3.3 Fallback

The scheme allows for a fall-back: if a client does not support the new transaction scheme, it can submit the classical transaction with just the ECDSA signature, i.e. $L_\mathrm{T}(T)$ as defined in equ. (16) for types 0 and 1. It will be up to the full nodes to decide how long they want to accept these transactions. Since there is no way of checking whether an address is a new or an old type of address (for the system it plays no role - the sender's address of a transaction must be computable from the public keys that are associated with (one or two) signatures), there is no way of forcing the users to use the new addresses. Because the old addresses are not associated with the PQ keys, those will always have to be performed with just an ECDSA signature. However, a decision could be made that new transactions have to go to new addresses and so transfer of funds from the addresses that were created after a certain date (e.g. 2023-01-01) must include an ECDSA and a dCDL2 signature. In other words: any address that was created after the cutoff date can only appear as the sender in transactions of type 81.

3.4 Resource comparison

Introducing the new transaction type will have an impact on the space requirements for storing the additional transaction information and block processing time for miners and validators when validating new transactions.

Let $\mathbb{N}$ denote the non-negative integers and $\mathbb{B}$ byte sequences of arbitrary length. Let

$$\mathbb{N}_k = \{n \in \mathbb{N} : n < 2^k\} \quad \text{and} \quad \mathbb{B}_k = \{x \in \mathbb{B} : \|x\| = k\}. \tag{27}$$

Transaction size. For transactions of type 0 or 1, we have

$$T_{\mathrm{x}}, T_{\mathrm{y}} \in \{0,1\}; T_{\mathrm{c}} = \beta \in \mathbb{N}_{256}; T_{\mathrm{n}}, T_{\mathrm{p}}, T_{\mathrm{g}}, T_{\mathrm{v}}, T_{\mathrm{w}}, T_{\mathrm{r}}, T_{\mathrm{s}} \in \mathbb{N}_{256}; T_{\mathrm{t}} \in \mathbb{B}_{20}, T_{\mathbf{i}}, T_{\mathbf{d}} \in \mathbb{B}. \tag{28}$$

Since all the data in a transaction gets encoded using the `RLP`-encoding, the encoded objects are only slightly bigger. For the purpose of estimating the size of a typical transaction, we can assume that each of the quantities takes 32 bytes of space, except for T_{x} and T_{y}, which are one byte each, T_{t}, which is 20 bytes and the arbitrary length fields $T_{\mathbf{i}}, T_{\mathbf{d}}$ and $T_{\mathbf{A}}$. Assuming that a typical transaction is not a contract creation transaction, that $T_{\mathbf{d}}$ contains four 32-byte parameters and that $T_{\mathbf{A}}$ is empty, we obtain a transaction size of about 400 bytes. This matches well with the observed average transactions sizes on `etherscan.io`. The ECDSA signature on the SECP-256k1 curve has size 64 bytes and thus represents 16% of the total transaction size on average.

The proposed dCDL2 signature is 21 times larger than an ECDSA signature. The public key of the dCDL2 signature is 38 times larger than an ECDSA signature. Since the dCDL2 signature does not allow public key recovery, the public key must also be included in the transaction. This roughly corresponds to the addition of PQ signature components about sixty times greater than the current signature components (i.e. 3740 bytes), resulting in transactions that have on average 4140 bytes. The signatures and the PQ public key represent 92% of the transaction size.

Since the PQ signature and the corresponding public key do not get stored to the block chain and we only store an additional hash value of the two signatures, the increase in size of the stored data is about 8% for a typical transaction. Thus, given cryptographically relevant quantum computers, adding PQ signatures significantly improves the security of the EVM-based block chains, and so it can be argued that this increase in transaction size is acceptable given today's hardware capacities.

Signature time. The dCDL2 signature takes 2.5-times longer than a current ECDSA signature. As the two signature computations are independent, they can be performed in parallel. The transaction is signed by the user before being submitted. Furthermore, once submitted, there is a delay in the network as the user must wait for their transaction to be included in the block. Since the average block time for Ethereum is about 13 seconds, the additional time needed to compute the dCDL2 signature is negligible compared to the network delay.

Verification time. The verification time for a dCDL2 signature is 0.3 times the time for the ECDSA verification. Since the two verifications are independent of each other and can be performed in parallel, there is no delay in verifying the additional signature. Moreover, verifying the signature is a minor part of checking the validity of a transaction, so even the computational overhead is negligible.

4 Conclusion

The PQ-EVM proposes an efficient and simple solution to the quantum threat for the EVM via the addition of a PQ signature. Recall that n-bit classical (quantum) security means that it would take a classical (quantum) computer 2^n operations to break. Currently, 80-bit security is considered safe. ECDSA using the SECP-256k1 curve has 128-bit classical security and about 30-bit quantum security, i.e. it is not quantum resistant. In contrast, dCDL2 has 123-bit classical security and 112-bit quantum security, and is thus considered safe in the presence of both classical and quantum computers.

References

[1] Ronald L Rivest, Adi Shamir, and Leonard M Adleman. "A method for obtaining digital signatures and public key cryptosystems". In: *Secure communications and asymmetric cryptosystems*. Routledge, 2019, pp. 217–239.

[2] Neal Koblitz. "Elliptic curve cryptosystems". In: *Mathematics of computation* 48.177 (1987), pp. 203–209.

[3] Tiago M Fernandez-Carames and Paula Fraga-Lamas. "Towards post-quantum blockchain: A review on blockchain cryptography resistant to quantum computing attacks". In: *IEEE access* 8 (2020), pp. 21091–21116.

[4] John Preskill. "Quantum computing 40 years later". In: *arXiv preprint arXiv:2106.10522* (2021).

[5] Richard P Feynman et al. "Simulating physics with computers". In: *Int. j. Theor. phys* 21.6/7 (1982).

[6] Peter W Shor. "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer". In: *SIAM review* 41.2 (1999), pp. 303–332.

[7] Lieven MK Vandersypen et al. "Experimental realization of Shor's quantum factoring algorithm using nuclear magnetic resonance". In: *Nature* 414.6866 (2001), pp. 883–887.

[8] *IBM newsroom*. https://newsroom.ibm.com/2021-11-16-IBM-Unveils-Breakthrough-127-Qubit-Quantum-Processor. Accessed: 2022-03-31.

[9] *MIT news*. https://news.mit.edu/2020/scaling-quantum-chip-0708. Accessed: 2022-03-31.

[10] *Rigetti annoucement*. https://investors.rigetti.com/news-releases/news-release-details/rigetti-computing-announces-commercial-availability-80-qubit. Accessed: 2022-03-31.

[11] Craig Gidney and Martin Ekerå. "How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits". In: *Quantum* 5 (2021), p. 433.

[12] German Federal office for Information Security. *Status of quantum computer development*. Version 1.2. 2020. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/Studien/Quantencomputer/P283_QC_Studie-V_1_2.pdf?__blob=publicationFile&v=1.

[13] *Post Quantum Cryptography*. https://csrc.nist.gov/projects/post-quantum-cryptography. Accessed: 2022-03-31.

[14] William Barker and Murugiah Souppaya. *[Project Description] Migration to Post-Quantum Cryptography (Draft)*. Tech. rep. National Institute of Standards and Technology, 2021.

[15] *Quantum-safe cryptography – fundamentals, current developments and recommendations*. https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/Brochure/quantum-safe-cryptography.pdf?__blob=publicationFile&v=4. Accessed: 2022-05-23.

[16] *ANSSI VIEWS ON THE POST-QUANTUM CRYPTOGRAPHY TRANSITION*. https://www.ssi.gouv.fr/en/publication/anssi-views-on-the-post-quantum-cryptography-transition/. Accessed: 2022-05-23.

[17] *Sizing Up Post-Quantum Signatures*. https://blog.cloudflare.com/sizing-up-post-quantum-signatures/. Accessed: 2022-04-10.

[18] Konstantinos Chalkias et al. "Blockchained post-quantum signatures". In: *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. IEEE. 2018, pp. 1196–1203.

[19] Léo Ducas et al. "Crystals-dilithium: A lattice-based digital signature scheme". In: *IACR Transactions on Cryptographic Hardware and Embedded Systems* (2018), pp. 238–268.

[20] Shi Bai et al. "CRYSTALS-Dilithium. Algorithm Specifications and Supporting Documentation (Version 3.1)". In: *NIST Post-Quantum Cryptography Project, Round 3 Submission* (2021). URL: https://pq-crystals.org/dilithium/data/dilithium-specification-round3-20210208.pdf.

[21] Vitalik Buterin and Martin Swende. *EIP-2930: Optional access lists*. https://eips.ethereum.org/EIPS/eip-2930. Accessed: 2022-05-09.

[22] Gavin Wood et al. "Ethereum: A secure decentralised generalised transaction ledger". In: *Ethereum project yellow paper* Berlin Version.d77a387 (2022), pp. 1–41.

[23] Micah Zoltu. *EIP-2718: Typed Transaction Envelope*. https://eips.ethereum.org/EIPS/eip-2718. Accessed: 2022-05-09.

[24] Vitalik Buterin. *EIP-155: Simple replay attack protection*. https://eips.ethereum.org/EIPS/eip-155. Accessed: 2022-05-09.

[25] *Experiments we should be running*. https://ethereum-magicians.org/t/experiments-we-should-be-running/2573/11. Accessed: 2022-05-09.

[26] *We should be moving beyond EOAs, not enshrining them even further (EIP 3074-related)*. https://ethereum-magicians.org/t/we-should-be-moving-beyond-eoas-not-enshrining-them-even-further-eip-3074-related/6538. Accessed: 2022-05-09.

[27] *Implementing account abstraction as part of eth1.x*. https://ethereum-magicians.org/t/implementing-account-abstraction-as-part-of-eth1-x/4020. Accessed: 2022-05-09.

[28] *EIP-2718: Typed Transaction Envelope.* https://ethereum-magicians.org/t/eip-2718-typed-transaction-envelope/4355. Accessed: 2022-05-09.

[29] Eli Ben-Sasson et al. "Scalable, transparent, and post-quantum secure computational integrity". In: *Cryptology ePrint Archive* (2018).

[30] *Q + A on ETH 2.0 - Serenity session at Paris Council 2019.* https://ethereum-magicians.org/t/q-a-on-eth-2-0-serenity-session-at-paris-council-2019/2861. Accessed: 2022-05-09.

[31] *Increasing address size from 20 to 32 bytes.* https://ethereum-magicians.org/t/increasing-address-size-from-20-to-32-bytes/5485. Accessed: 2022-05-09.

[32] Vitalik Buterin. *EIP-101: Serenity Currency and Crypto Abstraction.* https://eips.ethereum.org/EIPS/eip-101. Accessed: 2022-05-09.

[33] Danny Ryan and Chih-Cheng Liang. *EIP-1011: Hybrid Casper FFG.* https://eips.ethereum.org/EIPS/eip-1011. Accessed: 2022-05-09.

[34] Carl Beekhuizen. *EIP-2333: BLS12-381 Key Generation.* https://eips.ethereum.org/EIPS/eip-2333. Accessed: 2022-05-09.

[35] Vitalik Buterin et al. *EIP-2938: Account Abstraction.* https://eips.ethereum.org/EIPS/eip-2938. Accessed: 2022-05-09.

[36] Vitalik Buterin et al. *EIP-4844: Shard Blob Transactions.* https://eips.ethereum.org/EIPS/eip-4844. Accessed: 2022-05-09.

[37] Emre Karabulut and Aydin Aysu. "Falcon Down: Breaking Falcon Post-Quantum Signature Scheme through Side-Channel Attacks". In: *2021 58th ACM/IEEE Design Automation Conference (DAC).* IEEE. 2021, pp. 691–696.

[38] Standards for Efficient Cryptography Group. "SEC 1: Elliptic Curve Cryptography, Version 2.0." In: (Mar. 2009).

[39] Standards for Efficient Cryptography Group. "SEC 2: Recommended Elliptic Curve Domain Parameters, Version 2.0". In: (Jan. 2010).

A ECDSA signatures

This appendix describes the ECDSA signature scheme as used in EVM. Note that the description of the signature and verification steps below omits some of the checks that need to be performed, in order to ensure that the signature is secure. These checks have been omitted here for clarity. Full description, including the omitted checks, can be found in [38, 39].

Let $\mathcal{C}$ be the elliptic curve SECP-256k1 defined by the equation

$$y^2 = x^3 + 7, \tag{29}$$

over the finite field $\mathbb{F}_p$ with

$$\begin{aligned} p &= 115792089237316195423570985008687907853269984665640564039457584007908834671663 \\ &= 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1. \end{aligned}$$

The group of points on this elliptic curve is cyclic of order `secp256k1n` $= n$ with

$$n = 115792089237316195423570985008687907852837564279074904382605163141518161494337.$$

Let G be a generator of this group, defined as

$$G = (55066263022277343669578718895168534326250603453777594175500187360389116729240,$$
$$32670510020758816978083085130507043184471273380659243275938904335757337482424).$$

Next, we define three functions `ECDSAPUBKEY`, `ECDSASIGN` and `ECDSARECOVER`, for respectively key generation, signing and public key recovery.

Key generation. The sender generates the private key p_r by randomly selecting a positive integer (represented as a byte array of length 32 in big-endian form) in the range $[1, \texttt{secp256k1n} - 1]$. The function `ECDSAPUBKEY` is defined as

$$\texttt{ECDSAPUBKEY}(p_\mathrm{r}) = p_\mathrm{u} = p_\mathrm{r} G,$$

where p_u denotes the public key.

Signing. To sign a message m whose hash is e, an entity A with the private key p_r proceeds as follows:

1. Select a random or pseudorandom integer k, $1 \leqslant k \leqslant n - 1$.
2. Compute $kG = (x_1, y_1)$ and interpret x_1 as an integer.
3. Let $v = y_1 \bmod 2$.
4. Compute $r = x_1 \bmod n$.
5. Compute $k^{-1} \bmod n$.
6. Interpret e as an integer and compute $s = k^{-1}(e + p_\mathrm{r} r) \bmod n$.
7. A's signature for the message m is $(v, r, s) = \texttt{ECDSASIGN}(e, p_\mathrm{r})$.

Public key recovery. To recover the public key p_u from the signature data (e, v, r, s), proceed as follows:

1. Using r and v, recover the point $R = kG = (x_1, y_1)$ by setting $x_1 = r$ and solving the elliptic curve equ. (29) for y_1, using v to determine which of the two possible values is correct.[3]
2. Compute $r^{-1} \bmod n$.
3. Compute $p_\mathrm{u} = \texttt{ECDSARECOVER}(e, v, r, s) = r^{-1}(sR - eG)$.

To verify A's signature (v, r, s) on message m with hash e, perform the following:

1. Verify that r and s are integers in the interval $[1, n - 1]$.
2. Recover the public key of A via $p_\mathrm{u} = \texttt{ECDSARECOVER}(e, v, r, s)$.
3. Compute $w = s^{-1} \bmod n$.
4. Compute $u_1 = ew \bmod n$, $u_2 = rw \bmod n$.
5. Compute $X = u_1 G + u_2 p_\mathrm{u}$.
6. Interpret the x-coordinate x_1 of X as an integer and compute $z = x_1 \bmod n$.
7. Accept the signature if and only if $z = r$.

[3]Note that the value of k cannot be recovered.

B dCDL2 signatures

This appendix describes the dCDL2 signature scheme as used in this paper. Note that the description of the signature and verification steps below involves functions that are defined in [20] but are not repeated here for simplicity. The pseudo-code also does not describe the optimisations that can be performed to speed up the operations. The descriptions are taken from Figure 4 in [20], with explicit parameter values as defined for CRYSTALS-Dilithium Level 2.

Let H denote the `SHAKE-256` extendable output function. Specifically, $H(M, d)$ generates an output of length d bits from a message M.

Let $q = 8380417 = 2^{23} - 2^{13} + 1$ (a prime number) and let R_q be the polynomial ring $\mathbb{Z}_q[X]/(X^{256} + 1)$. For an element $a \in \mathbb{Z}_q$ we define $\|a\|_\infty$ as

$$\|a\|_\infty = |a \bmod {}^{\pm}q|,$$

where $a \bmod {}^{\pm}q$ is the unique element a' in the range $-\frac{q}{2} < a' \leqslant \frac{q}{2}$. Similarly, for a polynomial $p = a_0 + a_1X + \cdots + a_{n-1}X^{n-1} \in R_q$ we define

$$\|p\|_\infty = \max_i \|a_i\|_\infty.$$

Furthermore, let

$$S_\eta = \{w \in R_q : \|w\|_\infty \leqslant \eta\}, \tag{30}$$

$$\tilde{S}_\eta = \{w \bmod {}^{\pm}2\eta : w \in R_q\}. \tag{31}$$

Note that the sets S_η and $\tilde{S}_\eta$ are very similar, except that $\tilde{S}_\eta$ does not contain any polynomials with $-\eta$ coefficients. Let B_τ denote the set of elements of R_q that have τ coefficients that are either -1 or 1 and the rest are 0.

Next, we define functions `PQCDPUBKEY`, `PQCDSECKEY` and `PQCDSIGN`, for respectively public/private key generation and signing.

Key generation. The sender generates the keys by first randomly selecting a 256-bit number $p_\zeta \in \mathbb{B}_{32}$ and then expanding it using H to generate the various components of the actual private key:

1. Randomly select $p_\zeta \in \mathbb{B}_{32}$.
2. $(\rho, \rho', K) = H(p_\zeta, 1024) \in \mathbb{B}_{32} \times \mathbb{B}_{64} \times \mathbb{B}_{32}$.
3. Let $\mathbf{A} = \texttt{ExpandA}(\rho) \in R_q^{4\times 4}$.
4. Let $(\mathbf{s}_1, \mathbf{s}_2) = \texttt{ExpandS}(\rho') \in S_2^4 \times S_2^4$.
5. Compute $\mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$.
6. $(\mathbf{t}_0, \mathbf{t}_1) = \texttt{Power2Round}_q(\mathbf{t}, 13)$.
7. $tr = H(\rho \cdot \mathbf{t}_1, 256) \in \mathbb{B}_{32}$, where $\cdot$ denotes the concatenation of the byte arrays.
8. The public key is $(p_\rho, p_{\mathbf{t}_1}) = (\rho, \mathbf{t}_1) = \texttt{PQCDPUBKEY}(p_\zeta)$, the expanded private key is $sk = (\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0) = \texttt{PQCDSECKEY}(p_\zeta)$.

Signing. To sign a message m with hash e, an entity A with the private key p_ζ proceeds as follows:

1. Compute the expanded private key $sk = \texttt{PQCDSECKEY}(p_\zeta)$.

2. Let $\mathbf{A} = \texttt{ExpandA}(\rho) \in R_q^{4\times 4}$.

3. Compute $\mu = H(tr \cdot e, 512) \in \mathbb{B}_{64}$.

4. Let $\kappa = 0$, $(\mathbf{z}, \mathbf{h}) = \perp$.

5. Compute $\rho' = H(K \cdot \mu, 512) \in \mathbb{B}_{64}$ (Note: this makes the signature deterministic).

6. While $(\mathbf{z}, \mathbf{h}) = \perp$, repeat steps 6a – 6g:

   (a) Let $\mathbf{y} = \texttt{ExpandMask}(\rho', \kappa) \in \tilde{S}^4_{131072}$.

   (b) Compute $\mathbf{w} = \mathbf{Ay}$.

   (c) Let $\mathbf{w}_1 = \texttt{HighBits}(\mathbf{w}, 190464)$.

   (d) Let $\tilde{c} = H(\mu \cdot \mathbf{w}_1, 256) \in \mathbb{B}_{32}$ and $c = \texttt{SampleInBall}(\tilde{c}) \in B_{39}$.

   (e) Compute $\mathbf{z} = \mathbf{y} + c\mathbf{s}_1$ and $r_0 = \texttt{LowBits}_q(\mathbf{w} - x\mathbf{s}_2, 190464)$.

   (f) If $\|z\|_\infty \geqslant 130994$ or $\|r_0\|_\infty \geqslant 95154$, then $(\mathbf{z}, \mathbf{h}) = \perp$, else compute $\mathbf{h} = \texttt{MakeHint}_q(-c\mathbf{t}_0, \mathbf{w} - c\mathbf{s}_2 + c\mathbf{t}_0, 190464)$. If now $\|c\mathbf{t}_0\|_\infty \geqslant 95232$ or the number of 1's in $\mathbf{h}$ is greater than 80, set $(\mathbf{z}, \mathbf{h}) = \perp$.

   (g) Let $\kappa = \kappa + 4$.

7. A's signature for the message m is $(\tilde{c}, \mathbf{z}, \mathbf{h}) = \texttt{PQCDSIGN}(e, p_\zeta)$.

To verify A's signature $(\tilde{c}, \mathbf{z}, \mathbf{h})$ on message m with hash e using the public key $(p_\rho, p_{\mathbf{t_1}})$, perform the following:

1. Generate $\mathbf{A} = \texttt{ExpandA}(p_\rho) \in R_q^{4\times 4}$

2. Compute $\mu = H(H(p_\rho \cdot p_{\mathbf{t_1}}, 256) \cdot e, 512) \in \mathbb{B}_{64}$.

3. Compute $c = \texttt{SampleInBall}(\tilde{c}) \in B_{39}$ and $\mathbf{w}'_1 = \texttt{UseHint}_q(\mathbf{h}, \mathbf{Az} - 8192c\mathbf{t}_1, 190464)$.

4. Accept the signature if $\|z\|_\infty < 130994$ and $\tilde{c} = H(\mu \cdot \mathbf{w}'_1, 256)$ and the number of 1's in $\mathbf{h}$ is at most 80.