Post-Quantum Zero-Knowledge Lineage and Authorization Proofs for Hierarchical Deterministic Wallets

Vincenzo Botta[1], Michal Pospieszalski[1], Emanuele Ragnoli[1], and Justus Ranvier[1]

[1] AmericanFortress, Jackson Hole, Wyoming , postquantum@americanfortress.io

Abstract

We present a post-quantum hierarchical deterministic wallet (HDW) signature scheme built on the BIP32-Ed25519 key derivation. The scheme is designed for compatibility with existing Ed25519 infrastructure: public keys are identical to those produced by any standard BIP32-Ed25519 wallet following the same derivation path, requiring no address migration or key re-registration. Post-quantum security is achieved by replacing the classical Ed25519 signing step with a ZK-STARK proof: the signer proves that it knows the master secret key satisfying the full BIP32-Ed25519 derivation chain and that the public key A_n was honestly derived from that witness. We present two variants: a base scheme Σ in which the HMAC tag is part of the public signature, and a modified scheme Σ' in which the tag is hidden entirely inside the STARK witness. We further describe a split-proof architecture that separates the derivation proof (generated once per key pair) from the signing proof (generated once per message), enabling parallel proof generation and constant signing cost independent of derivation depth at verification time. We implement Σ' in Rust using RISC Zero as the STARK backend and report benchmarks.

American Fortress, Jackson Hole, Wyoming, USA.

Contents

1 Introduction

HDW standardised in the BIP32 specification and its Ed25519 adaptation BIP32-Ed25519, are an important key management infrastructure in blockchain systems. A single seed deterministically generates an entire tree of key pairs, allowing a user to derive fresh public keys for every transaction while retaining the ability to recover the full wallet from the seed alone. The security of BIP32-Ed25519 is currently based on the discrete logarithm problem that can be broken in polynomial time by Shor's algorithm on a sufficiently large quantum computer.

The development of quantum computers will eventually require changes to blockchain protocols, especially in the design of digital signatures. However, modifying blockchain protocols is complex and typically requires long coordination and deployment times. For this reason, it is useful to consider approaches that can be adopted immediately, without introducing major protocol changes.

Such approaches may involve a moderate loss of efficiency, but can provide protection against quantum adversaries. This allows existing systems to maintain an acceptable level of security until it becomes practical to transition to fully post-quantum signature schemes or other long-term solutions at the protocol level that will also require the complete change of the addresses of the users.

We construct a post-quantum signature scheme for BIP32-Ed25519 hierarchical deterministic wallets that achieves exactly this goal. The public key $A_n = k_{L_n} \cdot B$ produced by our scheme is identical to the one produced by a standard BIP32-Ed25519 wallet following the same seed and derivation path. Signing no longer produces a classical Ed25519 signature; instead, the signer produces a ZK-STARK proof certifying knowledge of a seed x_0 and derived key material (k_{L_n}, k_{R_n}) such that the full BIP32-Ed25519 derivation chain from x_0 to A_n is satisfied and a valid signature tag for the message M is correctly computed from k_{R_n}. The verifier checks the STARK proof against the public statement (M, A_n) without learning anything about the seed or private keys.

1.1 Contributions

We make the following contributions.

1. We define the HMAC-ZK signature scheme Σ and its hidden-tag variant Σ', both instantiating the HDW framework of Section 4. We prove hierarchical EUF-CMA security for both schemes (Theorems 2, 6) against quantum polynomial-time adversaries.
2. We decompose the monolithic NP relation $\mathcal{R}$ into two independent sub-relations: a derivation relation $\mathcal{R}_{\mathsf{deriv}}$, proved once per key pair and reused across all signatures, and a signing relation $\mathcal{R}_{\mathsf{sign}}$, proved once per message.
3. Unlike post-quantum HDW proposals based on lattice or isogeny assumptions [1, 2, 3, 4], our scheme preserves the BIP32-Ed25519 public key format exactly. No address migration or key re-registration is required even if the signature verification requires checking a STARK proof in place of a classical Ed25519 signature.
4. **Implementation and benchmarks.** We implement Σ' in Rust using RISC Zero as the STARK backend.

1.2 Limitations and Open Problems

The main practical limitation of the scheme is signing time: proving takes on the order of minutes to hours depending on derivation depth, which makes the current implementation unsuitable for interactive use cases. This cost is intrinsic to the use of HMAC-SHA512 inside the STARK circuit. Replacing `HMAC-SHA512` with a ZK-friendly hash function such as Poseidon [5] would reduce proving time substantially, but would break BIP32-Ed25519 compatibility by producing different public keys.

1.3 Technical Overview

A BIP32-Ed25519 wallet derives its entire key hierarchy deterministically from a single 256-bit seed x_0. The root step hashes the seed with SHA-512 and applies a set of bit-level tweaks to produce the root extended key $k_0 = k_{L_0} \| k_{R_0}$ and chain code $c_0 = \mathsf{SHA256}(\texttt{0x01} \| x_0)$. Each subsequent child step at depth i computes $z_i = \mathsf{HMAC\text{-}SHA512}(c_{i-1}, \texttt{0x02} \| A_{i-1} \| \langle i_i \rangle_4)$, splits the 512-bit output into a 224-bit left half $z_L^{(i)}$ and a 256-bit right half $z_R^{(i)}$, and updates $k_{L_i} = k_{L_{i-1}} + 8 \cdot z_L^{(i)} \pmod{2^{256}}$, $k_{R_i} = k_{R_{i-1}} + z_R^{(i)} \pmod{2^{256}}$. The public key at depth i is $A_i = k_{L_i} \cdot B$, where B is the generator. A_i can be recomputed from public data alone.

Making BIP32-Ed25519 post-quantum. Our core idea is to leave the public keys A_i entirely unchanged and replace only the signing step. Instead of producing a signature the signer produces a ZK-STARK proof π attesting the knowledge of (x_0, k_{L_i}, k_{R_i}) satisfying the NP relation $\mathcal{R}$: the full BIP32-Ed25519 derivation chain from x_0 to A_i is correctly executed, and the message tag $S = \mathsf{HMAC\text{-}SHA512}(k_{R_i}, M)$ is honestly computed from the chain-derived k_{R_i}. The verifier checks π against the public statement (M, S, A_i) without learning anything about x_0 or the private key material. We call this scheme Σ. Notice that this technique keeps the derived keys identical to the originally generated keys.

Hiding the tag. In the base scheme Σ the tag S appears in the public statement. Since S is pseudorandom to any party not holding k_{R_i}, it carries no information, but its presence is unnecessary. The modified scheme Σ' moves S entirely into the private witness and adds the condition $h = \mathsf{SHA256}(k_{R_i} \| M)$ to generate the proof. The verifier now receives only the proof π against the minimal statement (M, A_i).

Split-proof architecture. The monolithic STARK circuit for $\mathcal{R}$ grows linearly with derivation depth n, which would make every signing operation expensive. We decouple the two sources of cost by splitting $\mathcal{R}$ into two independent sub-relations linked by a public hash $\mathsf{com}_{k_R} = \mathsf{SHA256}(k_{R_n})$:

- $\mathcal{R}_{\text{deriv}}$: certifies that A_n was honestly derived from a seed and that com_{k_R} is the hash of the chain-derived k_{R_n}. This proof π_{deriv} is generated *once per key pair* and reused for every subsequent signature that ues the same public key.
- $\mathcal{R}_{\text{sign}}$: certifies, for a given message M, that the signer holds a k_{R_n} consistent with com_{k_R} and that $h = \mathsf{SHA256}(k_{R_n} \| M)$ is correctly computed. This proof π_{sign} involves only two SHA-256 invocations and is entirely independent of n.

At verification time, both proofs are checked against the shared public value com_{k_R}. Notice that thistechnique can e used also to precompute the derivations proofs atgeneration time and then each derivation proof is used when needed to generate a signature.

Implementation and performance. We implement Σ' in Rust using RISC Zero as the STARK backend. The guest program checks all five conditions of $\mathcal{R}'$ and commits only (M, A_n, n) to the public journal; the host invokes the external prover in succinct mode to produce a constant-size receipt of 218.4 KB regardless of derivation depth. Key generation completes in under 1 ms at all measured depths. Proof generation grows linearly in n at approximately 61.7 seconds per unit of depth, ranging from $\approx$ 200 s at $n = 2$ to $\approx$ 9,138 s at $n = 147$. Verification is constant at 9–10 ms across all depths, and peak RAM does not exceed 8.4 MB, confirming that the bottleneck is computational rather than memory-bound.

2 Related Works

Signature in blockchains and deterministic wallets The scheme whose key derivation we build upon is BIP32-Ed25519, introduced by Khovratovich and Law [6] as an adaptation of BIP32 to the Edwards25519 group. Unlike classical BIP32, which operates over secp256k1 and admits public child-key derivation via linear scalar addition, BIP32-Ed25519 multiplies the child increment by 8 to remain within the cofactor-cleared subgroup, and restricts the left HMAC output to 28 bytes to prevent scalar overflow across derivation depth.

The formal cryptographic treatment of hierarchical deterministic wallets (HDW) was initiated by Das et al. [7], who introduced a security model for deterministic wallets, proposed a construction and proved it secure under the discrete-logarithm assumption. Das et al. [8] subsequently tightened the analysis, providing exact security bounds for BIP32 and addressing gaps in the original reduction. The access-hierarchy model that we follow, in which derivation keys propagate transitively to descendants and security is defined via a hierarchical EUF-CMA experiment, is due to Di Luzio et al. [9]. Erwig and Riahi [10] extended the HDW framework to support adaptor signatures. Further work has studied lightweight variants targeting IoT devices [11] and stealth-address extensions from lattice assumptions [12].

Our signature scheme relies on a ZK-STARK for zero-knowledge and knowledge soundness in the quantum era. The ZK-STARK proof system was introduced by Ben-Sasson, Bentov, Horesh, and Riabzev [13] as a scalable, transparent, and post-quantum secure computational system based solely on collision-resistant hash functions. The FRI protocol underlying STARK was introduced in the same work and later analyzed. An alternative lightweight proof system is Ligero [14], also hash-based and transparent, which underpins the Ligetron zkVM used as a proof-of-concept engine in related work on post-quantum EdDSA migration [15, 16]. They observe that EdDSA's deterministic seed-to-key mapping enables the seed to serve as a zero-knowledge witness, and construct a post-quantum secure dual-mode signature protocol that authorizes a fresh lattice-based verification key under the original Ed25519 address without any address change. Their implementation using the Ligetron zkVM reports a proving time of 6.2 seconds and a proof size of 5.4 MB at current benchmarks. ZK-friendly hash functions such as Poseidon [5] have been proposed to reduce STARK circuit complexity; however, adopting them would break compatibility with the BIP32-Ed25519 standard.

A complementary approach to [15] in the Bitcoin setting is the QSB scheme [17], which constructs quantum-safe Bitcoin transactions within legacy Script constraints by replacing the ECDSA commitment with a hash-to-sig puzzle whose security relies solely on the preimage resistance of RIPEMD-160.

The first post-quantum deterministic wallet was proposed by Alkadri et al. [1], who instantiate a rerandomizable-key-based construction using qTESLA, establishing security notions of unlinkability and unforgeability in the quantum random oracle model. Das et al. [2] construct efficient post-quantum threshold wallets from isogeny-based signatures with rerandomizable keys, while Shaw and Dutta [3] provide a compact post-quantum deterministic wallet from isogeny-based signatures following the same path. Most recently, Deegan et al. [4] present two post-quantum HD wallet constructions supporting non-hardened public key derivation: one based on ML-DSA (hardened derivation only) and one based on Raccoon-G (non-hardened derivation), the latter being the first to recover BIP32's full public key derivation functionality with provable security under standard lattice assumptions. A concurrent work by Seck and Roux-Langlois [18] targets a Dilithium-based construction for Bitcoin-style chains.

The security of our construction in the post-quantum setting relies on the hardness of inverting SHA-512 against Grover's algorithm and on the PRF security of HMAC-SHA-512. Our work follows the line of post-quantum cryptographic research. Ring-LWE-based encryption and signatures, as

introduced by Lyubashevsky et al. [19, 20], underpin the CRYSTALS-Dilithium/ML-DSA and Falcon signature schemes that appear in alternative post-quantum wallet proposals. Castryck et al [21] proposed CSIDH, a commutative group action from supersingular isogenies, provides a further post-quantum primitive with compact keys and has been used as a building block in isogeny-based HD wallet designs. Gajland et al. [22] proposed SWOOSH, a lattice-based non-interactive key exchange, demonstrates that lattice assumptions can support Diffie-Hellman-style non-interactive protocols, used in the public derivation functionality of HD wallets.

3 Notation and Primitives

3.1 Notation

Let $\lambda \in \mathbb{N}$ denote the *security parameter*. All algorithms implicitly receive 1^λ as input unless stated otherwise. A function $f : \mathbb{N} \to \mathbb{R}_{\geq 0}$ is *negligible*, written $f(\lambda) = \mathsf{negl}(\lambda)$, if for every polynomial p there exists λ_0 such that $f(\lambda) < 1/p(\lambda)$ for all $\lambda \geq \lambda_0$. We write $[n] = \{1, 2, \ldots, n\}$ for a positive integer n, and $[0 : n] = \{0, 1, \ldots, n-1\}$. For a finite set S, we write $x \xleftarrow{\$} S$ to denote that x is chosen uniformly at random from S. We abbreviate $\{0,1\}^* = \bigcup_{n \geq 0} \{0,1\}^n$ for the set of all finite bit-strings. For a bit-string $s \in \{0,1\}^*$ and indices $i \leq j$, we write $s[i : j]$ for the sub-string of bits at positions $i, i+1, \ldots, j-1$. A probabilistic polynomial-time (PPT) algorithm is a randomised algorithm whose running time is bounded by a polynomial in λ. A quantum polynomial-time (QPT) algorithm is a quantum algorithm whose running time is polynomial in λ. An adversary $\mathcal{A}$ is called *PPT* (resp. *QPT*) if it runs in probabilistic (resp. quantum) polynomial time. Let $\mathbb{G}$ be the prime-order subgroup of the Edwards25519 elliptic curve, with prime order ℓ and generator $B \in \mathbb{G}$. We write scalars in $\mathbb{Z}_\ell$ and group elements in $\mathbb{G}$. Scalar multiplication is written $k \cdot P$ for $k \in \mathbb{Z}_\ell$, $P \in \mathbb{G}$. All integers are encoded in little-endian byte order. For a non-negative integer v and a byte-length ℓ, we write $\langle v \rangle_\ell$ for the ℓ-byte little-endian encoding of v. Bit i of a byte-string is bit $i \bmod 8$ of byte $\lfloor i/8 \rfloor$, counting bit 0 as the least significant bit of byte 0.

3.2 Computational Assumption

Definition 1 (Discrete Logarithm (DL) in $\mathbb{G}$)**.** *The* discrete logarithm problem *in* $\mathbb{G}$ *is hard if for all PPT adversaries* $\mathcal{A}$*:*

$$\Pr\big[\mathcal{A}(A) = k \;\big|\; k \xleftarrow{\$} \mathbb{Z}_\ell,\ A := k \cdot B\big] \leq \mathsf{negl}(\lambda).$$

Shor's algorithm [23] solves the discrete logarithm problem in quantum polynomial time. Therefore the DL assumption does *not* hold against QPT adversaries[1].

3.3 Cryptographic Hash Functions

Definition 2 (Collision-resistant hash function (CRHF))**.** *A function family* $H = \{H_k : \{0,1\}^* \to \{0,1\}^n\}_k$ *is* collision resistant *if for all PPT adversaries* $\mathcal{A}$*:*

$$\Pr\big[\mathcal{A}(k) = (x, x') \ \wedge\ x \neq x' \ \wedge\ H_k(x) = H_k(x')\big] \leq \mathsf{negl}(\lambda).$$

Definition 3 (Preimage resistance)**.** *A hash function* $H : \{0,1\}^* \to \{0,1\}^n$ *is* preimage resistant *if for all PPT adversaries* $\mathcal{A}$*:*

$$\Pr\big[\mathcal{A}(H(x)) = x' \ \wedge\ H(x') = H(x) \;\big|\; x \xleftarrow{\$} \{0,1\}^n\big] \leq \mathsf{negl}(\lambda).$$

[1] This is the fundamental motivation for the present work: we replace every component of the signing process whose security rests on Definition 1 with a post-quantum alternative.

Quantum preimage resistance (QPre). We say a hash function $H : \{0,1\}^* \to \{0,1\}^n$ is *quantum preimage resistant* if no QPT adversary can find a preimage with non-negligible probability. Grover's algorithm [24] can find a preimage of any n-bit hash function using only $O(2^{n/2})$ quantum queries, compared to the $O(2^n)$ queries required classically. As a result, an n-bit hash provides only $n/2$ bits of quantum preimage security. Concretely, SHA-512 truncated to 256 bits offers 2^{128} quantum preimage security, and SHA-256 offers 2^{128} quantum preimage security.

3.4 Pseudorandom Functions and HMAC-SHA-512

Definition 4 (Pseudorandom function (PRF))**.** *A function family $F = \{F_k : \mathcal{X} \to \mathcal{Y}\}_{k \in \mathcal{K}}$ is a* secure PRF *if for all PPT adversaries $\mathcal{A}$:*

$$\left|\Pr[\mathcal{A}^{F_k(\cdot)} = 1 \mid k \xleftarrow{\$} \mathcal{K}] - \Pr[\mathcal{A}^{f(\cdot)} = 1 \mid f \xleftarrow{\$} \mathcal{Y}^{\mathcal{X}}]\right| \leq \mathsf{negl}(\lambda).$$

Post-quantum PRF security. When the adversary is QPT and may query F_k in quantum superposition, the standard PRF definition is strengthened to the *quantum-access PRF* (qPRF) model [**?**]. We say F is a *post-quantum secure PRF* if no QPT adversary with quantum oracle access to F_k can distinguish it from a random function with better than negligible advantage. HMAC instantiated with a post-quantum secure compression function is conjectured to satisfy this stronger notion [**?**].

Instantiation: HMAC-SHA-512. We use:

$$\mathsf{HMAC\text{-}SHA512} : \{0,1\}^{256} \times \{0,1\}^* \longrightarrow \{0,1\}^{512},$$

defined as $\mathsf{HMAC\text{-}SHA512}(k, m) = \mathsf{SHA512}\big((k \oplus \mathsf{opad}) \parallel \mathsf{SHA512}((k \oplus \mathsf{ipad}) \| m)\big)$, where opad and ipad are the standard HMAC padding constants [**?**]. We rely on the following assumption throughout.

Definition 5 (Post-quantum PRF security of HMAC-SHA-512)**.** *The family $\{\mathsf{HMAC\text{-}SHA512}(k, \cdot)\}_{k \in \{0,1\}^{256}}$ is a post-quantum secure PRF: for all QPT adversaries $\mathcal{A}$ with quantum oracle access,*

$$\left|\Pr[\mathcal{A}^{\mathsf{HMAC\text{-}SHA512}(k,\cdot)} = 1 \mid k \xleftarrow{\$} \{0,1\}^{256}] - \Pr[\mathcal{A}^{f(\cdot)} = 1 \mid f \xleftarrow{\$} (\{0,1\}^{512})^{\{0,1\}^*}]\right| \leq \mathsf{negl}(\lambda).$$

3.5 ZK-STARKs

A *non-interactive zero-knowledge argument* (NIZK) for an NP relation $\mathcal{R} \subseteq \mathcal{X} \times \mathcal{W}$ is a tuple $\Pi = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{VerifyProof})$ of algorithms where:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$: generates public parameters pp.
- $\mathsf{Prove}(\mathsf{pp}, x, w) \to \pi$: given a statement $x \in \mathcal{X}$ and witness $w \in \mathcal{W}$ with $(x, w) \in \mathcal{R}$, produces a proof π.
- $\mathsf{VerifyProof}(\mathsf{pp}, x, \pi) \to \{0,1\}$: outputs 1 iff π is a valid proof for x.

We require Π to satisfy the following four properties against QPT adversaries.

Perfect completeness. For all $(x, w) \in \mathcal{R}$ and all $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$:

$$\Pr\big[\mathsf{VerifyProof}(\mathsf{pp}, x, \mathsf{Prove}(\mathsf{pp}, x, w)) = 1\big] = 1.$$

Post-quantum knowledge soundness. For every QPT prover $\mathcal{P}^*$ and every $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, if $\mathcal{P}^*(\mathsf{pp})$ outputs (x, π) with $\mathsf{VerifyProof}(\mathsf{pp}, x, \pi) = 1$, then there exists a QPT extractor Ext such that:

$$\Pr\big[(x,\ \mathsf{Ext}^{\mathcal{P}^*}(\mathsf{pp}, x)) \in \mathcal{R}\big] \geq 1 - \mathsf{negl}(\lambda).$$

Simulation extractability. For every QPT adversary $\mathcal{A}$ given access to a simulation oracle $\mathcal{S}(\mathsf{pp}, \cdot)$ that outputs simulated proofs for arbitrary statements, if $\mathcal{A}$ outputs an accepting proof π^* for a statement x^* that was not previously queried to $\mathcal{S}$, then there exists a QPT extractor Ext such that:

$$\Pr\big[(x^*,\ \mathsf{Ext}^{\mathcal{A}}(\mathsf{pp}, x^*)) \in \mathcal{R}\big] \geq 1 - \mathsf{negl}(\lambda).$$

Statistical zero-knowledge. There exists a PPT simulator $\mathcal{S}$ such that for all $(x, w) \in \mathcal{R}$ and all (possibly computationally unbounded) verifiers $\mathcal{V}^*$:

$$\Big|\Pr[\mathcal{V}^*(\mathsf{pp}, x, \mathsf{Prove}(\mathsf{pp}, x, w)) = 1] - \Pr[\mathcal{V}^*(\mathsf{pp}, x, \mathcal{S}(\mathsf{pp}, x)) = 1]\Big| \leq \mathsf{negl}(\lambda).$$

Instantiation: ZK-STARK. We instantiate Π with a *ZK-STARK* [13], a *transparent* (no trusted setup) and post-quantum secure argument system whose security reduces solely to the collision resistance of the underlying hash function. The FRI [13] low-degree test underpins the STARK's succinctness and post-quantum soundness. In all security arguments below, the STARK may be replaced by any NIZK satisfying the four properties above without affecting the conclusions.

Remark 1 (Proof size and verification.)**.** *In the succinct (recursive) mode of the RISC Zero backend used in our implementation, the STARK receipt has* constant *size (218.4 KB) and verification time (9–10 ms) regardless of the depth of the BIP32-Ed25519 derivation chain proved inside the circuit; see Section 11 for details.*

3.6 Digital Signatures and Hierarchical Deterministic Wallets

Digital signatures. A *digital signature scheme* is a triple $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ of PPT algorithms:

- $\mathsf{Gen}(1^\lambda) \to (\mathsf{sk}, \mathsf{pk})$: generates a signing key sk and verification key pk.
- $\mathsf{Sign}(\mathsf{sk}, M) \to \sigma$: signs message $M \in \mathcal{M}$ under sk.
- $\mathsf{Verify}(\mathsf{pk}, M, \sigma) \to \{0, 1\}$: outputs 1 iff σ is a valid signature on M under pk.

Definition 6 (EUF-CMA)**.** *A signature scheme Σ is* existentially unforgeable under chosen-message attack *(EUF-CMA) if for all PPT adversaries $\mathcal{A}$:*

$$\Pr\left[\begin{array}{l|l} \mathsf{Verify}(\mathsf{pk}, M^*, \sigma^*) = 1 & (\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}(1^\lambda) \\ \wedge\ M^* \notin \{M_1, \ldots, M_q\} & (M^*, \sigma^*) \leftarrow \mathcal{A}^{\mathsf{Sign}(\mathsf{sk}, \cdot)}(\mathsf{pk}) \\ & M_1, \ldots, M_q \textit{ queried to } \mathsf{Sign} \end{array}\right] \leq \mathsf{negl}(\lambda).$$

The scheme is post-quantum EUF-CMA secure *if the same bound holds for all QPT adversaries $\mathcal{A}$ that may issue signing queries classically.*

Ed25519. Ed25519 [**?**] is a Schnorr-type signature scheme over the Edwards25519 group $\mathbb{G}$. The signing key is a scalar $\mathsf{sk} = k_L \in \mathbb{Z}_\ell$ and the verification key is $\mathsf{pk} = A = k_L \cdot B \in \mathbb{G}$. Its security (EUF-CMA in the random-oracle model) rests on the discrete logarithm assumption (Definition 1), and is therefore broken by Shor's algorithm.

Hierarchical deterministic wallets. A *hierarchical deterministic wallet* (HDW) [**?**] over a DAG access hierarchy $G = (V, E)$ is a tuple $\Pi_{\mathsf{HDW}} = (\mathsf{Set}, \mathsf{DPub}, \mathsf{DPriv}, \mathsf{Sign}, \mathsf{Verify})$ whose algorithms are described in full in Section **??** below. Informally, a single seed S deterministically generates signing and verification keys for every node $v \in V$, and a party holding the derivation key d_i of node v_i can derive the signing key of any descendant $v_j \in \mathsf{Desc}(v_i)$. The security notion is *hierarchical EUF-CMA* (Definition 8): an adversary that corrupts a subset of nodes (together with all their descendants) and queries signing oracles at uncorrupted nodes cannot forge a signature at any uncorrupted node on a fresh message.

4 Security Model of Hierarchical Deterministic Wallet

We follow the definition of [9] for hierarchical deterministic wallets (HDW). HDW is composed by five algorithms $(\mathsf{Set}, \mathsf{DPub}, \mathsf{DPriv}, \mathsf{Sign}, \mathsf{Verify})$, where: (1) Set deterministically instantiates the wallet by generating the public parameters pp and a set of derivation keys d_i, one for each node $v_i \in V$ of the hierarchy; (2) DPriv and DPub are responsible for the derivation of signing and public keys: DPriv derives the signing key sk_j of a node v_j (a descendant of v_i) using the derivation key d_i associated to v_i, while DPub derives the corresponding public key pk_j using only the public parameters pp; (3) Sign and Verify are the standard signing and verification algorithms of a digital signature scheme.

Concretely, let $G = (V, E)$ be a directed acyclic graph (DAG) representing an access hierarchy. We define the set of descendants $\mathsf{Desc}(v_i) = \{v_j \mid v_i \succcurlyeq v_j\}$ of node v_i to be the set of nodes v_j such that there exists a directed path from v_i to v_j in G. A hierarchical deterministic wallet $\Pi = (\mathsf{Set}, \mathsf{DPub}, \mathsf{DPriv}, \mathsf{Sign}, \mathsf{Verify})$, defined over a seed space $\mathcal{S}$ and message space $\mathcal{M}$, consists of the following algorithms.

- $\mathsf{Set}(1^\lambda, G, S)$: The deterministic setup algorithm takes as input a security parameter, an access graph $G = (V, E)$, and an initial seed $S \in \mathcal{S}$, and outputs the public parameters pp and a set of derivation keys $\{d_i\}_{v_i \in V}$.
- $\mathsf{DPub}(\mathsf{pp}, v_i)$: The deterministic public derivation algorithm takes as input the public parameters pp and a target node v_i, and outputs the public key pk_i associated to node v_i.
- $\mathsf{DPriv}(\mathsf{pp}, d_i, v_i, v_j)$: The deterministic private derivation algorithm takes as input the public parameters pp, the derivation key d_i of node v_i, and a target node $v_j \in \mathsf{Desc}(v_i)$, and outputs the secret key sk_j associated to node v_j.
- $\mathsf{Sign}(\mathsf{sk}_i, M)$: The randomized signing algorithm takes as input a message $M \in \mathcal{M}$ and a secret key sk_i, and outputs a signature σ.
- $\mathsf{Verify}(\mathsf{pk}_i, M, \sigma)$: The deterministic verification algorithm takes as input a public key pk_i, a message M, and a signature σ, and outputs a decision bit $b \in \{0, 1\}$.

A hierarchical deterministic wallet is *correct* if any user can derive the private and public key of their descendants and create a valid signature on their behalf.

Definition 7 (Correctness of HDW)**.** *A hierarchical deterministic wallet*

$$\Pi = \big(\mathsf{Set},\ \mathsf{DPub},\ \mathsf{DPriv},\ \mathsf{Sign},\ \mathsf{Verify}\big)$$

with seed space $\mathcal{S}$ and message space $\mathcal{M}$ is correct if for every DAG $G = (V, E)$, all $v_i, v_j \in V$ with $v_j \in \mathsf{Desc}(v_i)$, all $S \in \mathcal{S}$, and all $M \in \mathcal{M}$:

$$\Pr\big[\mathsf{Verify}(\mathsf{pk}_j,\ M,\ \mathsf{Sign}(\mathsf{sk}_j, M)) = 1\big] \ \geq\ 1 - \mathsf{negl}(\lambda),$$

where $(\mathsf{pp}, \{d_i\}_{v_i \in V}) = \mathsf{Set}(1^\lambda, G, S)$, $\mathsf{sk}_j = \mathsf{DPriv}(\mathsf{pp}, d_i, v_i, v_j)$, and $\mathsf{pk}_j = \mathsf{DPub}(\mathsf{pp}, v_j)$.

The security of a hierarchical deterministic wallet draws inspiration from existential unforgeability of signature schemes. We allow an adversary $\mathcal{A}$ to corrupt an arbitrary number of users in the hierarchy—by corrupting a user, $\mathcal{A}$ implicitly corrupts all her descendants as well. In addition, $\mathcal{A}$ has access to a signing oracle that returns signatures on arbitrary messages from any uncorrupted node. We challenge $\mathcal{A}$ to forge a signature for a fresh message on behalf of an uncorrupted node.

Definition 8 (Hierarchical EUF-CMA of HDW)**.** *A hierarchical deterministic wallet* Π *is* hierarchically existentially unforgeable under chosen-message attacks *if for every DAG* $G = (V, E)$ *and all PPT adversaries* $\mathcal{A}$*:*

$$\Pr\big[G^{\mathrm{heuf}}_{\Pi,\mathcal{A}}(\lambda, G) = 1\big] \;\leq\; \mathsf{negl}(\lambda),$$

where the experiment $G^{\mathrm{heuf}}_{\Pi,\mathcal{A}}(\lambda, G)$ *is defined as follows.*

Setup. *The challenger samples* $S \xleftarrow{\$} \mathcal{S}$ *and runs* $(\mathsf{pp}, \{d_i\}_{v_i \in V}) = \mathsf{Set}(1^\lambda, G, S)$*. It gives* pp *to* $\mathcal{A}$*.*

Query. *The adversary* $\mathcal{A}$ *has access to the following oracles.*

- $\mathcal{O}_{\mathrm{Corr}}(v_i)$*: On input* $v_i \in V$*, the challenger returns* d_i *to* $\mathcal{A}$*. Let* Q_{Corr} *denote the set of corrupted nodes, including all their descendants* $\mathsf{Desc}(v_i)$*.*
- $\mathcal{O}_{\mathrm{Sign}}(M, v_i)$*: On input* $(M, v_i) \in \mathcal{M} \times V$*, the challenger computes* $\mathsf{sk}_i = \mathsf{DPriv}(\mathsf{pp}, d_0, v_0, v_i)$ *and returns* $\sigma \xleftarrow{\$} \mathsf{Sign}(\mathsf{sk}_i, M)$*. Let* Q_{Sign} *denote the set of pairs* (M, v_i) *queried to this oracle.*

Forgery. $\mathcal{A}$ *outputs* (v_i, M, σ)*. The experiment returns* 1 *if and only if* $\mathsf{Verify}(\mathsf{pk}_i, M, \sigma) = 1$ *where* $\mathsf{pk}_i = \mathsf{DPub}(\mathsf{pp}, v_i)$*,* $v_i \notin Q_{\mathrm{Corr}}$*, and* $(M, v_i) \notin Q_{\mathrm{Sign}}$*; otherwise it returns* 0*.*

5 Post Quantum Hierarchical Deterministic Wallet

5.1 BIP32-Ed25519 Key Derivation

We recall the BIP32-Ed25519 key derivation scheme [6] and instantiate it in terms of the DPub and DPriv algorithms of the HDW framework of Section 4. Let $n \geq 0$ be the chain depth. All bit indices follow the *little-endian* convention: bit i of a byte string is bit $i \bmod 8$ of byte $\lfloor i/8 \rfloor$, with bit 0 being the least significant.

Public and private data. At every depth i, the scheme maintains the following quantities.

- *Private:*
  - The seed $x_0 \in \{0,1\}^{256}$, from which the entire key hierarchy is deterministically derived. It is never exposed.
  - The extended key $k_i = k_{L_i} \| k_{R_i} \in \{0,1\}^{256} \times \{0,1\}^{256}$, where k_{L_i} is the signing scalar and k_{R_i} is the nonce seed. This is the derivation key $d_i \coloneqq k_i$ of the HDW.
- *Public:*
  - The Ed25519 public key $A_i \coloneqq k_{L_i} \cdot B \in \mathbb{G}$ at each depth.
  - The chain code $c_i \in \{0,1\}^{256}$ at each depth, with root chain code $c_0 \coloneqq \mathsf{SHA256}(\mathtt{0x01} \| x_0)$.
  - The child index $i_n \in \{0, \ldots, 2^{31} - 1\}$ at each non-hardened node.

The public parameters are therefore $\mathsf{pp} \coloneqq (A_0, c_0)$, and the root derivation key is $d_0 \coloneqq k_0$. The seed x_0 and all extended keys $(k_1, \ldots, k_n)$ are private.

Root ($n = 0$) — used by Set. Given a seed $x_0 \in \{0,1\}^{256}$ (*private*), compute $\mathsf{SHA512}(x_0)$ and parse its 512-bit output as

$$k_{L_0}^{\text{raw}} \parallel k_{R_0} \;\coloneqq\; \mathsf{SHA512}(x_0), \qquad k_{L_0}^{\text{raw}},\, k_{R_0} \in \{0,1\}^{256}.$$

If the third-highest bit (bit 253) of $k_{L_0}^{\text{raw}}$ is set, discard x_0 and resample. Otherwise obtain k_{L_0} by the following bit-level adjustments:

- clear bits 0, 1, 2 (ensures $8 \mid k_{L_0}$ as an integer);
- clear bit 255;
- set bit 254.

Set the root extended private key $k_0 \coloneqq k_{L_0} \parallel k_{R_0} \in \{0,1\}^{512}$ (*private*) and the root public key $A_0 \coloneqq k_{L_0} \cdot B$ (*public*). We abbreviate these steps as $k_0 \coloneqq \mathsf{Tweak}(\mathsf{SHA512}(x_0))$. Derive the root chain code $c_0 \coloneqq \mathsf{SHA256}(\texttt{0x01} \| x_0) \in \{0,1\}^{256}$ (*public*). The root derivation key is $d_0 \coloneqq k_0$.

Child step ($n > 0$) — used by DPriv and DPub. Given the parent chain code $c_{n-1} \in \{0,1\}^{256}$ (*public*), the parent public key A_{n-1} (*public*), and a public child index $i_n \in \{0, \ldots, 2^{31}-1\}$, compute

$$z_n \;\coloneqq\; \mathsf{HMAC\text{-}SHA512}\big(c_{n-1},\, \texttt{0x02} \| A_{n-1} \| \langle i_n \rangle_4\big) \;\in\; \{0,1\}^{512},$$

where $\langle i_n \rangle_4$ denotes the 4-byte little-endian encoding of i_n. Extract from z_n:

$$\begin{aligned} z_L^{(n)} \;&\coloneqq\; z_n[0:224] \;\in\; \{0,1\}^{224}, \\ z_n[224:256] &\text{ (32 bits discarded)}, \\ z_R^{(n)} \;&\coloneqq\; z_n[256:512] \;\in\; \{0,1\}^{256}. \end{aligned}$$

Update the *private* extended key as

$$\begin{aligned} k_{L_n} \;&\coloneqq\; k_{L_{n-1}} + 8 \cdot z_L^{(n)} \pmod{2^{256}}, \\ k_{R_n} \;&\coloneqq\; k_{R_{n-1}} + z_R^{(n)} \pmod{2^{256}}. \end{aligned}$$

Derive the child chain code

$$c_n \;\coloneqq\; \mathsf{HMAC\text{-}SHA512}\big(c_{n-1},\, \texttt{0x03} \| A_{n-1} \| \langle i_n \rangle_4\big)[32:] \;\in\; \{0,1\}^{256} \quad (\textit{public}).$$

The child derivation key is $d_n \coloneqq k_n$, where $k_n \coloneqq k_{L_n} \| k_{R_n}$ (*private*). The child public key is

$$A_n \;\coloneqq\; k_{L_n} \cdot B \;=\; A_{n-1} + 8 \cdot z_L^{(n)} \cdot B \;\in\; \mathbb{G} \quad (\textit{public}).$$

Instantiation of DPriv and DPub.

- $\mathsf{DPriv}(\mathsf{pp},\, d_i,\, v_i,\, v_n)$: parse $d_i = k_i = k_{L_i} \| k_{R_i}$. For each step $j = i+1, \ldots, n$, retrieve the public index i_j from pp, compute $z_j = \mathsf{HMAC\text{-}SHA512}(c_{j-1},\, \texttt{0x02} \| A_{j-1} \| \langle i_j \rangle_4)$, and update k_{L_j}, k_{R_j}, c_j as above. Return $\mathsf{sk}_n \coloneqq k_n$ and $d_n \coloneqq k_n$.
- $\mathsf{DPub}(\mathsf{pp},\, v_n)$: parse $\mathsf{pp} = (A_0,\, c_0)$. For each step $j = 1, \ldots, n$, retrieve the public index i_j from pp, compute $z_j = \mathsf{HMAC\text{-}SHA512}(c_{j-1},\, \texttt{0x02} \| A_{j-1} \| \langle i_j \rangle_4)$, and update $A_j = A_{j-1} + 8 \cdot z_L^{(j)} \cdot B$ and c_j as above. Return $\mathsf{pk}_n \coloneqq A_n$. *This computation uses only public data $(A_0, c_0, i_1, \ldots, i_n)$; no private key material is required.*

Remark 2 (Public derivability)**.** *The child index i_n is public, consequently,* DPub *is a fully public computation: any party holding (A_0, c_0) and the sequence of child indices $(i_1, \ldots, i_n)$ can derive A_n without knowledge of any private key material. The signing nonce k_{R_n} remains private as it is never published and can only be recovered by a party holding the seed x_0.*

Remark 3 (Discarded bits in the child step)**.** *The* HMAC-SHA512 *output z_n is 512 bits (64 bytes). Only 480 of those bits are used: the first 28 bytes ($z_L^{(n)}$, 224 bits) and the last 32 bytes ($z_R^{(n)}$, 256 bits). The intervening 4 bytes at positions* 224–255 *are structurally discarded. This truncation follows the BIP32-Ed25519 specification: restricting $z_L^{(n)}$ to 28 bytes ensures that the accumulated scalar k_{L_n} does not wrap around the curve order $\ell \approx 2^{252}$ after any number of derivation steps up to the protocol limit of 2^{20}.*

5.2 The NP Relation for the ZK-STARK

The NP relation $\mathcal{R}$ checked by the STARK is defined as follows.

$$\mathcal{R} \coloneqq \left\{ \big((M,\, S,\, A_n),\ \mathbf{w}\big) \;\middle|\; \mathbf{w} = (x_0,\, k_{L_n},\, k_{R_n}) \ \text{ and (i)–(v) hold} \right\},$$

where $x_0 \in \{0,1\}^{256}$, $k_{L_n}, k_{R_n} \in \{0,1\}^{256}$, and the public indices $(i_1, \ldots, i_n) \in \{0, \ldots, 2^{31}-1\}^n$ are read from pp:

(i) **Root.**

$$\begin{aligned} k_0 &\coloneqq \mathsf{Tweak}(\mathsf{SHA512}(x_0)),\\ k_0 &= k_{L_0} \| k_{R_0} \in \{0,1\}^{256} \times \{0,1\}^{256},\\ c_0 &\coloneqq \mathsf{SHA256}(\mathtt{0x01} \| x_0). \end{aligned}$$

(ii) **Child steps.** For $i = 1, \ldots, n$, compute $z_i \coloneqq \mathsf{HMAC\text{-}SHA512}(c_{i-1},\, \mathtt{0x02} \| A_{i-1} \| \langle i_i \rangle_4)$, write $z_i = z_L^{(i)} \| z_R^{(i)}$ with $z_L^{(i)} \in \{0,1\}^{224}$, $z_R^{(i)} \in \{0,1\}^{256}$, and set

$$k_{L_i} \;\coloneqq\; k_{L_{i-1}} + 8 \cdot z_L^{(i)} \pmod{2^{256}}, \qquad k_{R_i} \;\coloneqq\; k_{R_{i-1}} + z_R^{(i)} \pmod{2^{256}},$$

$$c_i \;\coloneqq\; \mathsf{HMAC\text{-}SHA512}(c_{i-1},\, \mathtt{0x03} \| A_{i-1} \| \langle i_i \rangle_4)[32:].$$

(iii) **Binding.** $k_{L_n} = k_{L_n}^{\mathsf{derived}}$ and $k_{R_n} = k_{R_n}^{\mathsf{derived}}$, i.e. the values supplied in the witness equal those produced by the derivation chain.

(iv) **Public key.** $A_n = k_{L_n} \cdot B$.

(v) **Signature.** $S = \mathsf{HMAC\text{-}SHA512}(k_{R_n},\, M)$.

The public statement is $\mathbf{x} = (M, S, A_n) \in \{0,1\}^* \times \{0,1\}^{512} \times \mathbb{G}$. The witness is $\mathbf{w} = (x_0,\, k_{L_n},\, k_{R_n})$.

5.3 Signature Scheme

We instantiate the HDW of Section 4 with the following HMAC-ZK signature scheme

$$\Sigma = \big(\mathsf{Set},\, \mathsf{DPub},\, \mathsf{DPriv},\, \mathsf{Sign},\, \mathsf{Verify}\big).$$

$\mathsf{Set}(1^\lambda, G, x_0)$: Run $\mathsf{pp} \leftarrow \mathsf{STARK.Setup}(1^\lambda)$. Execute the BIP32-Ed25519 root derivation of Section 5.1 on seed x_0 to obtain $k_0 = (k_{L_0} \| k_{R_0})$, $A_0 = k_{L_0} \cdot B$, and root chain code $c_0 =$

$\mathsf{SHA256}(\texttt{0x01}\|x_0)$. Output public parameters $\mathsf{pp} := (A_0,\, c_0,\, (i_1, \ldots, i_n))$ and root derivation key $d_0 := k_0$.

$\mathsf{DPub}(\mathsf{pp},\, v_n)$: Parse $\mathsf{pp} = (A_0,\, c_0,\, (i_1, \ldots, i_n))$. Execute the BIP32-Ed25519 child steps of Section 5.1 using the public indices $(i_1, \ldots, i_n)$ to obtain A_n. Output $\mathsf{pk}_n := (\mathsf{pp},\, A_n)$.

$\mathsf{DPriv}(\mathsf{pp},\, d_i,\, v_i,\, v_n)$: Parse $d_i = k_i = (k_{L_i}\|k_{R_i})$ and $\mathsf{pp} = (A_0,\, c_0,\, (i_1, \ldots, i_n))$. Execute the BIP32-Ed25519 child steps of Section 5.1 from depth i to depth n using the public indices $(i_{i+1}, \ldots, i_n)$ to obtain $k_n = (k_{L_n}\|k_{R_n})$. Output $\mathsf{sk}_n := (\mathsf{pp},\, k_n)$ and set $d_n := k_n$.

$\mathsf{Sign}(\mathsf{sk}_n,\, M)$:

Parse $\mathsf{sk}_n = (\mathsf{pp},\, k_n)$ and extract k_{L_n} and k_{R_n} from k_n. Compute $S := \mathsf{HMAC\text{-}SHA512}(k_{R_n},\, M)$. Let the public statement and private witness be

$$\mathbf{x} := (M,\, S,\, A_n), \qquad \mathbf{w} := (x_0,\, k_{L_n},\, k_{R_n}).$$

Compute

$$\pi \;\leftarrow\; \mathsf{STARK.Prove}(\mathsf{pp},\, \mathbf{x},\, \mathbf{w}).$$

Output $\sigma := (S,\, \pi)$.

$\mathsf{Verify}(\mathsf{pk}_n,\, M,\, \sigma)$: Parse $\mathsf{pk}_n = (\mathsf{pp},\, A_n)$ and $\sigma = (S,\, \pi)$. Output 1 if and only if

$$\mathsf{VerifyProof}(\mathsf{pp},\, (M,\, S,\, A_n),\, \pi) = 1.$$

6 Security of the Scheme Presented in Section 5.1

6.1 Correctness

Theorem 1 (Correctness)**.** *For all honestly generated key pairs* $(\mathsf{sk}, \mathsf{vk})$ *and all messages* $M \in \{0,1\}^*$*,*

$$\mathsf{Verify}\big(\mathsf{vk},\; M,\; \mathsf{Sign}(\mathsf{sk}, M)\big) = 1.$$

Proof sketch. Let $\sigma = (S, \pi) = \mathsf{Sign}(k_n, M)$. By construction, $S = \mathsf{HMAC\text{-}SHA512}(k_{R_n}, M)$ and the witness $\mathbf{w} = (x_0,\, k_{L_n},\, k_{R_n})$ satisfies:

1. $k_0 := \mathsf{Tweak}(\mathsf{SHA512}(x_0))$ and $c_0 := \mathsf{SHA256}(\texttt{0x01}\|x_0)$ by the root step of key derivation,
2. for $i = 1, \ldots, n$, the child-step recurrences hold by construction of Set and DPriv,
3. $k_{L_n} = k_{L_n}^{\mathsf{derived}}$ and $k_{R_n} = k_{R_n}^{\mathsf{derived}}$ by construction,
4. $A_n = k_{L_n} \cdot B$ by definition of key generation,
5. $S = \mathsf{HMAC\text{-}SHA512}(k_{R_n}, M)$ by the signing algorithm,
6. therefore $\big((M, S, A_n),\, \mathbf{w}\big) \in \mathcal{R}$.

By STARK completeness, $\mathsf{VerifyProof}(\mathsf{pp}, (M, S, A_n), \pi) = 1$, hence Verify accepts. □

6.2 Unforgeability

We consider existential unforgeability under chosen-message attack (EUF-CMA). The adversary $\mathcal{A}$ receives $\mathsf{vk} = A_n$, may query $\mathsf{Sign}(\mathsf{sk}, \cdot)$ on polynomially many messages $M_1, \ldots, M_q$, and wins if it outputs a valid forgery (M^*, σ^*) with $M^* \notin \{M_1, \ldots, M_q\}$.

Theorem 2 (EUF-CMA)**.** *Assume* HMAC-SHA512 *is a post-quantum secure PRF,* QPre *holds for* SHA-512*, and* STARK *is a simulation extractable zero-knowledge and post-quantum knowledge sound proof system. The scheme is EUF-CMA secure against QPT adversaries at the* 2^{128}*-quantum-query level, for all depths* $n \geq 0$*.*

Proof. We proceed via a sequence of games.

Game G_0 (real game). The signing oracle computes $S_i = \mathsf{HMAC\text{-}SHA512}(k_{R_n}, M_i)$ and $\pi_i \leftarrow \mathsf{STARK.Prove}(\mathsf{pp}, (M_i, S_i, A_n), \mathbf{w})$, where $\mathbf{w} = (x_0, k_{L_n}, k_{R_n})$.

Game G_1 (simulated proofs). Replace each proof $\pi_i \leftarrow \mathsf{STARK.Prove}(\mathsf{pp}, (M_i, S_i, A_n), \mathbf{w})$ with $\pi_i \leftarrow \mathcal{S}(\mathsf{pp}, (M_i, S_i, A_n))$. Any QPT distinguisher between G_0 and G_1 breaks post-quantum zero-knowledge of STARK. Hence $\big|\Pr[\mathcal{A} \text{ wins } G_0] - \Pr[\mathcal{A} \text{ wins } G_1]\big|$ is negligible.

Game G_2 (PRF replaced by random function). Replace $\mathsf{HMAC\text{-}SHA512}(k_{R_n}, \cdot)$ with a uniformly random function $f : \{0,1\}^* \to \{0,1\}^{512}$, returning $S_i = f(M_i)$. Any QPT distinguisher between G_1 and G_2 breaks PRF security of HMAC, so $\big|\Pr[\mathcal{A} \text{ wins } G_1] - \Pr[\mathcal{A} \text{ wins } G_2]\big|$ is negligible.

Analysis of G_2. In G_2, proofs are simulated and S-values are independent random function outputs; hence $\mathcal{A}$ receives no information about k_{R_n}. By knowledge soundness, if a QPT $\mathcal{A}$ produces an accepting proof for a fresh statement (M^*, S^*, A_n), there exists an extractor that extracts a witness $\mathbf{w}^* = (x_0^*, k_{L_n}^*, k_{R_n}^*)$ satisfying all conditions (i)–(v) of $\mathcal{R}$. We analyse the two cases separately.

Case $n = 0$. Condition (i) requires $k_{L_0}^* \| k_{R_0}^* = \mathsf{Tweak}(\mathsf{SHA512}(x_0^*))$, so both $k_{L_0}^*$ and $k_{R_0}^*$ are uniquely determined by x_0^*. Condition (iii) requires $k_{L_0}^* = k_{L_0}^{\mathsf{derived}}$ and $k_{R_0}^* = k_{R_0}^{\mathsf{derived}}$, which at depth $n = 0$ are exactly the values from condition (i). Condition (iv) requires $k_{L_0}^* \cdot B = A_0 = k_{L_0} \cdot B$, so $k_{L_0}^* = k_{L_0}$. Hence finding x_0^* satisfying conditions (i) and (iv) jointly requires $\mathsf{SHA512}(x_0^*)[0{:}256] = k_{L_0}$, a preimage attack on the left 256 bits of $\mathsf{SHA512}$ requiring 2^{128} quantum queries by QPre. Since $k_{R_0}^*$ is determined by the same x_0^*, no QPT adversary can independently choose it to satisfy condition (v). Hence $\Pr[\mathcal{A} \text{ wins } G_2]$ is negligible.

Case $n \geq 1$. Condition (i) requires $k_{L_0}^* \| k_{R_0}^* = \mathsf{Tweak}(\mathsf{SHA512}(x_0^*))$ and $c_0^* = \mathsf{SHA256}(\mathtt{0x01} \| x_0^*)$, so the entire chain is rooted at x_0^*. Condition (ii) requires the child-step recurrence to hold for $i = 1, \ldots, n$, using the public indices $(i_1, \ldots, i_n)$. This means $k_{L_n}^{\mathsf{derived}}$ and $k_{R_n}^{\mathsf{derived}}$ are both fully determined by x_0^*. Condition (iii) then requires $k_{L_n}^* = k_{L_n}^{\mathsf{derived}}$ and $k_{R_n}^* = k_{R_n}^{\mathsf{derived}}$. Any valid witness must have $k_{R_n}^*$ equal to $k_{R_n}^{\mathsf{derived}}$, which is the value produced by running the full derivation chain from x_0^*. Therefore the adversary cannot choose $k_{R_n}^*$ independently of x_0^* with non-negligible probability.

Condition (iv) requires $k_{L_n}^* \cdot B = A_n$, so a QPT adversary may obtain the actual k_{L_n} via Shor's algorithm from A_n. However, knowing k_{L_n} provides no information on $k_{R_n}^*$: since $k_{R_n}^*$ is locked to x_0^* via conditions (i)–(iii), and $k_{L_n}^{\mathsf{derived}} = \mathsf{Tweak}(\mathsf{SHA512}(x_0^*))[0{:}256]$, recovering any valid x_0^* from k_{L_n} still requires inverting the left 256 bits of $\mathsf{SHA512}$, which costs 2^{128} quantum queries by QPre. Without finding such x_0^*, the adversary cannot determine $k_{R_n}^{\mathsf{derived}}$, and hence cannot satisfy condition (v) with non-negligible probability, since k_{R_n} is never exposed in G_2. Therefore $\Pr[\mathcal{A} \text{ wins } G_2]$ is negligible.

Combining the games probability and the analysis of G_2 the probaility $\Pr[\mathcal{A} \text{ wins } G_0]$ is negligible. □

6.3 Correctness of the HDW Instantiation

Theorem 3 (Correctness of the HDW Instantiation)**.** *The scheme in Section 5.3, with algorithms* Set, DPub, DPriv, Sign, *and* Verify, *is a correct hierarchical deterministic wallet in the sense of Definition 7.*

Proof. Fix any DAG $G = (V, E)$, seed $x_0 \in \{0,1\}^{256}$, nodes $v_i, v_n \in V$ with $v_n \in \mathsf{Desc}(v_i)$, and message $M \in \{0,1\}^*$. Let $(\mathsf{pp}, \{d_j\}_{v_j \in V}) = \mathsf{Set}(1^\lambda, G, x_0)$, so $\mathsf{pp} = (A_0, c_0, (i_1, \ldots, i_n))$ and $d_0 = k_0$.

Key consistency. We show $\mathsf{DPub}(\mathsf{pp}, v_n) = A_n$ and $\mathsf{DPriv}(\mathsf{pp}, d_i, v_i, v_n) = k_n$ with $A_n = k_{L_n} \cdot B$.

DPriv parses $d_i = k_i = k_{L_i} \| k_{R_i}$ and applies the child steps of Section 5.1 for $j = i+1, \ldots, n$, computing at each step $z_j := \mathsf{HMAC\text{-}SHA512}\big(c_{j-1}, \mathtt{0x02} \| A_{j-1} \| \langle i_j \rangle_4\big)$, and updating k_{L_j}, k_{R_j}, c_j accordingly. It returns $\mathsf{sk}_n = k_n = k_{L_n} \| k_{R_n}$.

DPub parses $\mathsf{pp} = (A_0, c_0, (i_1, \ldots, i_n))$ and applies the same child steps for $j = 1, \ldots, n$, computing the identical z_j values (since c_{j-1}, A_{j-1}, and i_j are the same in both executions), and updating $A_j = A_{j-1} + 8 \cdot z_L^{(j)} \cdot B$. It returns $\mathsf{pk}_n = A_n$.

Since $k_{L_n} \cdot B = (k_{L_i} + 8 \sum_{j=i+1}^{n} z_L^{(j)}) \cdot B = A_i + 8 \sum_{j=i+1}^{n} z_L^{(j)} \cdot B = A_n$, the public key returned by DPub equals $k_{L_n} \cdot B$, the public key implicit in sk_n.

Signature validity. By Theorem 1, for any sk_n and M, $\mathsf{Verify}(\mathsf{pk}_n, M, \mathsf{Sign}(\mathsf{sk}_n, M)) = 1$. Combined with the key consistency above, we obtain

$$\Pr\big[\mathsf{Verify}(\mathsf{DPub}(\mathsf{pp}, v_n),\ M,\ \mathsf{Sign}(\mathsf{DPriv}(\mathsf{pp}, d_i, v_i, v_n), M)) = 1\big] \ \geq\ 1 - \mathsf{negl}(\lambda),$$

which is exactly the correctness condition of Definition 7. □

6.4 Hierarchical EUF-CMA of HDW

Theorem 4 (Hierarchical EUF-CMA of the HDW Instantiation)**.** *The scheme* $\Sigma = (\mathsf{Set}, \mathsf{DPub}, \mathsf{DPriv}, \mathsf{Sign}, \mathsf{Verify})$ *of Section 5.3 is hierarchically existentially unforgeable under chosen-message attacks in the sense of Definition 8.*

Proof. The hierarchical EUF-CMA security of Σ is a direct consequence of the single-key EUF-CMA security established in Theorem 2.

Suppose $\mathcal{A}$ wins $G^{\mathrm{heuf}}_{\Sigma,\mathcal{A}}(\lambda, G)$ with non-negligible probability, outputting a forgery (v^*, M^*, σ^*) such that $\mathsf{Verify}(\mathsf{DPub}(\mathsf{pp}, v^*), M^*, \sigma^*) = 1$, $v^* \notin Q_{\mathrm{Corr}}$, and $(M^*, v^*) \notin Q_{\mathrm{Sign}}$.

Since $v^* \notin Q_{\mathrm{Corr}}$, neither v^* nor any of its ancestors in G were corrupted. Therefore the derivation key $d_{v^*} = k_{v^*}$, and in particular the signing key $\mathsf{sk}_{v^*} = k_{v^*}$, were never revealed to $\mathcal{A}$.

The view of $\mathcal{A}$ in the hierarchical experiment is identical to the view of a standard EUF-CMA adversary against the single-key scheme of Section 5.3 at verification key $A_{v^*} = \mathsf{DPub}(\mathsf{pp}, v^*)$: signing oracle queries (M_i, v^*) at the target node correspond exactly to single-key signing oracle queries at A_{v^*}; signing queries at any other node $v_j \neq v^*$ are answered independently from x_0 and carry no information about sk_{v^*}; and corruption queries reveal derivation keys of nodes outside the ancestor chain of v^*, none of which expose sk_{v^*}. Consequently, (M^*, σ^*) with $\mathsf{Verify}(A_{v^*}, M^*, \sigma^*) = 1$ and $M^* \notin \{M_i : (M_i, v^*) \in Q_{\mathrm{Sign}}\}$ is a valid EUF-CMA forgery against the single-key scheme at A_{v^*}. By Theorem 2, this occurs with at most negligible probability, contradicting our assumption. □

7 Hiding the Tag

In the scheme of Section 5.3, the tag $S = \mathsf{HMAC\text{-}SHA512}(k_{R_n}, M)$ is included as a plaintext component of the public statement $\mathbf{x} = (M, S, A_n)$. We observe that S need not be disclosed: since S is the output of a PRF keyed by k_{R_n}, it is computationally indistinguishable from a uniformly random string to any PPT/QPT party not holding k_{R_n}, and its inclusion in the public statement carries no additional computational content beyond what A_n already provides. Nonetheless, we show that the scheme can be modified so that the prover certifies possession of a valid tag without revealing it, by moving S entirely into the private witness and replacing the public statement with (M, A_n). This demonstrates the flexibility of the ZK-STARK approach and may be of independent interest in contexts where minimising the public statement is desirable.

7.1 Modified NP Relation

We replace the NP relation $\mathcal{R}$ of Section 5.2 with the following relation. The tag S and an auxiliary hash h are kept private; the public statement carries no tag-related value:

$$\mathcal{R}' \coloneqq \left\{ \big((M,\, A_n),\ \mathbf{w}'\big) \;\middle|\; \mathbf{w}' = (x_0,\, k_{L_n},\, k_{R_n},\, h) \ \text{ and (i')–(v') hold} \right\},$$

where $x_0 \in \{0,1\}^{256}$, $k_{L_n}, k_{R_n} \in \{0,1\}^{256}$, $h \in \{0,1\}^{256}$, and the public indices $(i_1, \ldots, i_n) \in \{0, \ldots, 2^{31}-1\}^n$ are read from pp:

(i′) **Root.**

$$\begin{aligned} k_0 &\coloneqq \mathsf{Tweak}(\mathsf{SHA512}(x_0)),\\ k_0 &= k_{L_0} \| k_{R_0} \in \{0,1\}^{256} \times \{0,1\}^{256},\\ c_0 &\coloneqq \mathsf{SHA256}(\mathtt{0x01} \| x_0). \end{aligned}$$

(ii′) **Child steps.** For $i = 1, \ldots, n$, compute $z_i \coloneqq \mathsf{HMAC\text{-}SHA512}(c_{i-1},\, \mathtt{0x02} \| A_{i-1} \| \langle i_i \rangle_4)$, write $z_i = z_L^{(i)} \| z_R^{(i)}$ with $z_L^{(i)} \in \{0,1\}^{224}$, $z_R^{(i)} \in \{0,1\}^{256}$, and set

$$k_{L_i} \coloneqq k_{L_{i-1}} + 8 \cdot z_L^{(i)} \pmod{2^{256}}, \qquad k_{R_i} \coloneqq k_{R_{i-1}} + z_R^{(i)} \pmod{2^{256}},$$

$$c_i \coloneqq \mathsf{HMAC\text{-}SHA512}(c_{i-1},\, \mathtt{0x03} \| A_{i-1} \| \langle i_i \rangle_4)[32:].$$

(iii′) **Binding.** $k_{L_n} = k_{L_n}^{\mathsf{derived}}$ and $k_{R_n} = k_{R_n}^{\mathsf{derived}}$, i.e. the values supplied in the witness equal those produced by the derivation chain.

(iv′) **Public key.** $A_n = k_{L_n} \cdot B$.

(v′) **Signature.** $h = \mathsf{SHA256}(k_{R_n} \,\|\, M)$.

The public statement is $\mathbf{x}' = (M,\, A_n) \in \{0,1\}^* \times \mathbb{G}$. The witness is $\mathbf{w}' = (x_0,\, k_{L_n},\, k_{R_n},\, h)$, where $h = \mathsf{SHA256}(k_{R_n} \| M)$ is an auxiliary value used internally to bind k_{R_n} to the message inside the circuit. The verifier learns only that a valid derivation chain exists for A_n and that the prover holds the corresponding k_{R_n}.

Remark 4 (Role of h)**.** *The value $h = \mathsf{SHA256}(k_{R_n} \| M)$ is a purely internal witness component that allows the circuit to verify the binding between k_{R_n} (derived in conditions (i′)–(iii′)) and the message M, without requiring the full HMAC computation to appear in the public statement. In particular, h is* not *output as part of the signature and is never seen by the verifier; its sole role is to enable constraint (v′) to be checked inside the STARK arithmetic circuit.*

7.2 Modified Signature Scheme

The modified scheme $\Sigma' = (\mathsf{Set}, \mathsf{DPub}, \mathsf{DPriv}, \mathsf{Sign}', \mathsf{Verify}')$ differs from Σ of Section 5.3 only in Sign' and Verify'.

$\mathsf{Sign}'(\mathsf{sk}_n,\, M)$: Parse $\mathsf{sk}_n = (\mathsf{pp},\, k_n)$ and extract k_{L_n} and k_{R_n} from k_n. Compute $h \coloneqq \mathsf{SHA256}(k_{R_n} \| M)$. Let $\mathbf{x}' \coloneqq (M,\, A_n)$ be the public statement and $\mathbf{w}' \coloneqq (x_0,\, k_{L_n},\, k_{R_n},\, h)$ be the private witness. Compute

$$\pi \leftarrow \mathsf{STARK.Prove}(\mathsf{pp},\, \mathbf{x}',\, \mathbf{w}').$$

Output $\sigma' \coloneqq \pi$.

$\mathsf{Verify}'(\mathsf{pk}_n,\, M,\, \sigma')$: Parse $\mathsf{pk}_n = (\mathsf{pp},\, A_n)$ and $\sigma' = \pi$. Output 1 if and only if

$$\mathsf{VerifyProof}(\mathsf{pp},\, (M,\, A_n),\, \pi) = 1.$$

8 Security of the Scheme Presented in Section 7

8.1 Correctness

Theorem 5 (Correctness of Σ')**.** *For all honestly generated key pairs* $(\mathsf{sk}_n, \mathsf{pk}_n)$ *and all messages* $M \in \{0,1\}^*$,

$$\mathsf{Verify}'\big(\mathsf{pk}_n,\ M,\ \mathsf{Sign}'(\mathsf{sk}_n, M)\big) = 1.$$

Proof sketch. Let $\sigma' = \pi = \mathsf{Sign}'(k_n, M)$. By construction, $h = \mathsf{SHA256}(k_{R_n} \| M)$, so the witness $\mathbf{w}' = (x_0, k_{L_n}, k_{R_n}, h)$ satisfies all conditions (i′)–(v′) of $\mathcal{R}'$: conditions (i′)–(iv′) hold by key derivation and definition of A_n (same argument as Theorem 1), and condition (v′) holds by the definition of h. By STARK completeness, $\mathsf{VerifyProof}(\mathsf{pp}, (M, A_n), \pi) = 1$, hence Verify' accepts. □

8.2 EUF-CMA Security of Σ'

Theorem 6 (EUF-CMA of Σ')**.** *Assume* HMAC-SHA512 *is a post-quantum secure* PRF*,* QPre *holds for* SHA-512*,* SHA-256 *is preimage-resistant, and* STARK *has post-quantum zero-knowledge and knowledge soundness. The scheme* Σ' *is* EUF-CMA *secure against* QPT *adversaries at the* 2^{128}*-quantum-query level, for all depths* $n \geq 0$*.*

Proof. We follow the same game sequence as Theorem 2.

Game G_0 (real game). The signing oracle, on input M_i, computes $h_i = \mathsf{SHA256}(k_{R_n} \,\|\, M_i)$ and produces $\pi_i \leftarrow \mathsf{STARK.Prove}(\mathsf{pp}, (M_i, A_n), \mathbf{w}'_i)$, where $\mathbf{w}'_i = (x_0, k_{L_n}, k_{R_n}, h_i)$.

Game G_1 (simulated proofs). Replace each π_i with $\pi_i \leftarrow \mathcal{S}(\mathsf{pp}, (M_i, A_n))$. Any QPT distinguisher between G_0 and G_1 breaks post-quantum zero-knowledge of STARK, so $|\Pr[\mathcal{A} \text{ wins } G_0] - \Pr[\mathcal{A} \text{ wins } G_1]|$ is negligible.

Analysis of G_1. In G_1, all proofs are simulated; the oracle responses contain no information about k_{R_n}, h_i, or any witness component. Suppose $\mathcal{A}$ outputs a forgery (M^*, A_n, π^*) with $M^* \notin \{M_1, \ldots, M_q\}$ accepted by the verifier. By STARK knowledge soundness, there exists a QPT extractor recovering $\mathbf{w}^{*\prime} = (x_0^*, k_{L_n}^*, k_{R_n}^*, h^*)$ satisfying all conditions (i′)–(v′) of $\mathcal{R}'$. Conditions (i′)–(iv′) are identical to conditions (i)–(iv) of $\mathcal{R}$; the hardness argument of Theorem 2 applies verbatim, requiring 2^{128} quantum queries to invert the left 256 bits of SHA-512. Hence $\Pr[\mathcal{A} \text{ wins } G_1]$ is negligible, and combining all the probabilities, $\Pr[\mathcal{A} \text{ wins } G_0]$ is negligible. □

Remark 5 (Hierarchical EUF-CMA)**.** *The hierarchical EUF-CMA security of* Σ' *in the sense of Definition 8 follows from Theorem 6 by the same reduction used in Section 6 for* Σ*: the view of an adversary winning the hierarchical experiment at an uncorrupted node* v^* *reduces directly to a single-key EUF-CMA forgery against* Σ' *at verification key* A_{v^*}*.*

Remark 6 (Binding strength of condition (v′))**.** *In the original relation* $\mathcal{R}$*, condition (v) bound the tag to the message via* $S = \mathsf{HMAC\text{-}SHA512}(k_{R_n}, M)$*, whose binding relied on the PRF security of* HMAC-SHA512*. In* $\mathcal{R}'$*, condition (v′) instead uses* $h = \mathsf{SHA256}(k_{R_n} \| M)$*, which achieves binding by preimage resistance alone. This substitution is sound because* k_{R_n} *is the output of a PRF-based derivation chain rooted at the secret seed* x_0*: it is computationally indistinguishable from a uniformly random 256-bit string to any adversary who does not know* x_0*. Consequently,* k_{R_n} *acts as an unpredictable salt in the SHA-256 input, and finding any* $(k_{R_n}^*, M^*)$ *satisfying* $\mathsf{SHA256}(k_{R_n}^* \| M^*) = h$ *without knowledge of* x_0 *requires inverting SHA-256, which costs* 2^{128} *quantum queries by preimage resistance. The two conditions therefore achieve the same binding guarantee via different primitives, and the additional assumption on SHA-256 preimage resistance in Theorem 6 is both necessary and sufficient for this argument.*

9 Split-Proof Architecture and Parallel Proofs

The monolithic relation $\mathcal{R}$ of Section 5.2 can be replaced by two independent sub-relations, each proved by a separate STARK and linked at verification time via the public commitment $\mathsf{com}_{k_R} = \mathsf{SHA256}(k_{R_n})$. The derivation proof π_{deriv} certifies that A_n was honestly derived from a seed and that com_{k_R} is the SHA-256 hash of the resulting k_{R_n}; it is generated once per key pair and reused for every signature. The signing proof π_{sign} certifies, for a given message M, that the signer knows a k_{R_n} consistent with the public com_{k_R} and that the signature hash h was honestly computed; it is generated for every signing request. Because $\mathcal{R}_{\mathsf{deriv}}$ and $\mathcal{R}_{\mathsf{sign}}$ share no witness material, the two proofs can be generated concurrently on independent machines (or processes), reducing end-to-end latency.

9.1 Derivation Relation $\mathcal{R}_{\mathsf{deriv}}$

The derivation relation captures the statement that a public key A_n was honestly derived from a seed via the BIP32-Ed25519 chain of Section 5.1, and that com_{k_R} is the SHA-256 hash of the chain-derived right half k_{R_n}:

$$\mathcal{R}_{\mathsf{deriv}} \coloneqq \left\{ \big((A_n,\, \mathsf{com}_{k_R}),\ \mathbf{w}_1 \big) \;\middle|\; \mathbf{w}_1 = (x_0,\, k_{L_n},\, k_{R_n}) \text{ and (D-i)–(D-v) hold} \right\}.$$

Public statement: $(A_n, \mathsf{com}_{k_R}) \in \mathbb{G} \times \{0,1\}^{256}$.
Private witness: $\mathbf{w}_1 = (x_0, k_{L_n}, k_{R_n})$. The public indices $(i_1, \ldots, i_n)$ and all intermediate chain values are read from pp or recomputed internally during the relation check, as in Section 5.1.

(D-i) **Root.** $k_0 \coloneqq \mathsf{Tweak}(\mathsf{SHA512}(x_0)), \quad k_0 = k_{L_0} \| k_{R_0} \in \{0,1\}^{256} \times \{0,1\}^{256}$.

(D-ii) **Child steps.** For $i = 1, \ldots, n$, retrieve the public index i_i from pp and compute $z_i \coloneqq \mathsf{HMAC\text{-}SHA512}(c_{i-1},\, \mathtt{0x02} \| A_{i-1} \| \langle i_i \rangle_4)$, writing $z_i = z_L^{(i)} \| z_{\mathsf{mid}}^{(i)} \| z_R^{(i)}$ with $z_L^{(i)} \in \{0,1\}^{224}$, $z_{\mathsf{mid}}^{(i)} \in \{0,1\}^{32}$ discarded, $z_R^{(i)} \in \{0,1\}^{256}$, and set

$$k_{L_i} \coloneqq k_{L_{i-1}} + 8 \cdot z_L^{(i)} \pmod{2^{256}}, \qquad k_{R_i} \coloneqq k_{R_{i-1}} + z_R^{(i)} \pmod{2^{256}},$$

$$c_i \coloneqq \mathsf{HMAC\text{-}SHA512}(c_{i-1},\, \mathtt{0x03} \| A_{i-1} \| \langle i_i \rangle_4)[32:].$$

(D-iii) **Binding.** $k_{L_n} = k_{L_n}^{\mathsf{derived}}$ and $k_{R_n} = k_{R_n}^{\mathsf{derived}}$, i.e. both halves supplied in the witness equal those produced by the derivation chain.

(D-iv) **Public key.** $A_n = k_{L_n} \cdot B$. This constraint is checked inside the relation; k_{L_n} remains part of the private witness and is never exposed.

(D-v) **Commitment.** $\mathsf{com}_{k_R} = \mathsf{SHA256}(k_{R_n})$.

Remark 7 (Security of the public commitment)**.** *The value $\mathsf{com}_{k_R} = \mathsf{SHA256}(k_{R_n})$ appears in the public statement without weakening security: an adversary seeing com_{k_R} still cannot recover k_{R_n} without inverting SHA-256, which requires 2^{128} quantum queries by preimage resistance. The relation enforces via conditions (D-iii) and (D-v) that both k_{L_n} and k_{R_n} are the correct outputs of the derivation chain rooted at x_0, and that com_{k_R} is their correct hash, so neither can be freely chosen by a forger independently of A_n.*

9.2 Signing Relation $\mathcal{R}_{\mathsf{sign}}$

The signing relation captures the statement that the signer knows a right half k_{R_n} consistent with the publicly committed value com_{k_R}, and that the signature hash h was honestly computed from k_{R_n} and the message M:

$$\mathcal{R}_{\mathsf{sign}} \coloneqq \Big\{ \big((M,\, A_n,\, \mathsf{com}_{k_R}),\, \mathbf{w}_2 \big) \;\Big|\; \mathbf{w}_2 = (k_{R_n},\, h) \text{ and (S-i)–(S-ii) hold} \Big\}.$$

Public statement: $(M,\, A_n,\, \mathsf{com}_{k_R}) \in \{0,1\}^* \times \mathbb{G} \times \{0,1\}^{256}$.
Private witness: $\mathbf{w}_2 = (k_{R_n},\, h) \in \{0,1\}^{256} \times \{0,1\}^{256}$.

(S-i) **Binding to derivation.** $\mathsf{SHA256}(k_{R_n}) = \mathsf{com}_{k_R}$. This links k_{R_n} to the one certified by π_{deriv} without re-executing the derivation chain.
(S-ii) **Signature hash.** $h = \mathsf{SHA256}(k_{R_n} \| M)$.

9.3 Combined Relation and Verification

The full scheme relation is obtained by composing $\mathcal{R}_{\mathsf{deriv}}$ and $\mathcal{R}_{\mathsf{sign}}$ via the shared public value com_{k_R}:

$$\mathcal{R} \coloneqq \Big\{ \big((M,\, A_n,\, \mathsf{com}_{k_R}),\, \mathbf{w} \big) \;\Big|\; \mathbf{w} = (\pi_{\mathsf{deriv}},\, \pi_{\mathsf{sign}}) \text{ and (C-i)–(C-ii) hold} \Big\}.$$

Public statement: $(M,\, A_n,\, \mathsf{com}_{k_R}) \in \{0,1\}^* \times \mathbb{G} \times \{0,1\}^{256}$.
Witness: $\mathbf{w} = (\pi_{\mathsf{deriv}},\, \pi_{\mathsf{sign}})$.

(C-i) **Derivation proof.** $\mathsf{STARK.Verify}\big(\mathcal{R}_{\mathsf{deriv}},\ (A_n,\, \mathsf{com}_{k_R}),\ \pi_{\mathsf{deriv}}\big) = 1$, certifying that $\big((A_n,\, \mathsf{com}_{k_R}),\, \mathbf{w}_1\big) \in \mathcal{R}_{\mathsf{deriv}}$ for some $\mathbf{w}_1$.
(C-ii) **Signing proof.** $\mathsf{STARK.Verify}\big(\mathcal{R}_{\mathsf{sign}},\ (M,\, A_n,\, \mathsf{com}_{k_R}),\ \pi_{\mathsf{sign}}\big) = 1$, certifying that $\big((M,\, A_n,\, \mathsf{com}_{k_R}),\, \mathbf{w}_2\big) \in \mathcal{R}_{\mathsf{sign}}$ for some $\mathbf{w}_2$.

9.4 Parallel Proof Generation

Proof	Inputs	Frequency
π_{deriv}	x_0	Once per key pair
π_{sign}	$M,\, k_{R_n}$	Once per message

Figure 1: Lifecycle of the two proofs.

Because $\mathcal{R}_{\mathsf{deriv}}$ and $\mathcal{R}_{\mathsf{sign}}$ share no witness material, their proofs are *fully independent* and can be generated in parallel or separately. The only shared public value is $\mathsf{com}_{k_R} = \mathsf{SHA256}(k_{R_n})$, which is available to both provers once the derivation chain has been evaluated from x_0.

Parallel protocol (at signing time):

1. The prover loads the pre-computed receipt π_{deriv} and reads com_{k_R} from its public statement $(A_n,\, \mathsf{com}_{k_R})$.
2. The prover generates π_{sign} for the public statement $(M,\, A_n,\, \mathsf{com}_{k_R})$ with private witness $\mathbf{w}_2 = (k_{R_n},\, h)$, where $h \coloneqq \mathsf{SHA256}(k_{R_n} \| M)$.

3. The verifier receives $(\pi_{\mathsf{deriv}}, \pi_{\mathsf{sign}})$ and checks conditions (C-i) and (C-ii) of the combined relation $\mathcal{R}$ against the public statement $(M, A_n, \mathsf{com}_{k_R})$.

The main cost at signing time is dominated by π_{sign} alone, whose trace length is determined by two SHA-256 invocations (conditions (S-i) and (S-ii)), and is independent of the derivation depth n.

10 Recursive Decomposition of $\mathcal{R}_{\mathsf{deriv}}$

The derivation relation $\mathcal{R}_{\mathsf{deriv}}$ requires proving correctness of an n-step HMAC chain in a single circuit, whose size grows linearly in n. For large n this is undesirable. We show that $\mathcal{R}_{\mathsf{deriv}}$ admits a recursive decomposition into $n+1$ smaller relations (one root relation and n step relations), such that a proof of the full chain is obtained by composing proofs of the individual steps.

The root relation establishes the base case of the derivation:

$$\mathcal{R}_{\mathsf{root}} \coloneqq \left\{ \left((A_0, \mathsf{com}_{k_R}^{(0)}),\ \mathbf{w}_0 \right) \;\middle|\; \mathbf{w}_0 = (x_0, k_{L_0}, k_{R_0}) \text{ and (B-i)–(B-iii) hold} \right\},$$

where:

(B-i) **Root derivation.** $k_0 = k_{L_0} \| k_{R_0} \coloneqq \mathsf{Tweak}(\mathsf{SHA512}(x_0))$.
(B-ii) **Public key.** $A_0 \coloneqq k_{L_0} \cdot B$.
(B-iii) **Commitment.** $\mathsf{com}_{k_R}^{(0)} \coloneqq \mathsf{SHA256}(k_{R_0})$.

Each step relation extends the chain by one HMAC step, taking the previous proof as part of its witness. The public index i_i at each depth is read from pp, as in Section 5.1:

$$\mathcal{R}_{\mathsf{step}}^{(i)} \coloneqq \left\{ \left((A_i, \mathsf{com}_{k_R}^{(i)}, A_{i-1}, \mathsf{com}_{k_R}^{(i-1)}),\ \mathbf{w}_i \right) \;\middle|\; \mathbf{w}_i = (k_{L_{i-1}}, k_{R_{i-1}}, \pi_{i-1}) \text{ and (R-i)–(R-v) hold} \right\},$$

where:

(R-i) **Previous step.** π_{i-1} is a valid proof of $\mathcal{R}_{\mathsf{root}}$ (for $i = 1$) or $\mathcal{R}_{\mathsf{step}}^{(i-1)}$ (for $i > 1$), with public statement $(A_{i-1}, \mathsf{com}_{k_R}^{(i-1)})$.
(R-ii) **Witness consistency.** $k_{L_{i-1}} \cdot B = A_{i-1}$ and $\mathsf{SHA256}(k_{R_{i-1}}) = \mathsf{com}_{k_R}^{(i-1)}$. This binds the plaintext key halves in the witness to the values certified by π_{i-1}.
(R-iii) **HMAC child step.** Retrieve the public index i_i from pp and compute

$$z^{(i)} \coloneqq \mathsf{HMAC\text{-}SHA512}\big(c_{i-1},\ \mathtt{0x02} \| A_{i-1} \| \langle i_i \rangle_4\big),$$

then set

$$k_{L_i} \coloneqq k_{L_{i-1}} + 8 \cdot z^{(i)}[0:224] \pmod{2^{256}}, \qquad k_{R_i} \coloneqq k_{R_{i-1}} + z^{(i)}[256:512] \pmod{2^{256}},$$

$$c_i \coloneqq \mathsf{HMAC\text{-}SHA512}(c_{i-1},\ \mathtt{0x03} \| A_{i-1} \| \langle i_i \rangle_4)[32:].$$

(R-iv) **Public key.** $A_i \coloneqq k_{L_i} \cdot B$.
(R-v) **Updated commitment.** $\mathsf{com}_{k_R}^{(i)} \coloneqq \mathsf{SHA256}(k_{R_i})$.

Remark 8 (Recursive composition and proof caching)**.** *The decomposition into $\mathcal{R}_{\mathsf{root}}$ and $\mathcal{R}_{\mathsf{step}}^{(i)}$ is an engineering optimization that reduces per-step circuit size from $O(n)$ to $O(1)$. Its soundness follows directly from the knowledge soundness of the underlying STARK applied inductively at each step,*

with the commitment chain $\mathsf{com}_{k_R}^{(0)}, \ldots, \mathsf{com}_{k_R}^{(n)}$ *enforcing consistency across steps. This recursive composition can be implemented in RISC Zero, which supports it natively via its recursion API.*

A further practical benefit is proof caching. *Because* $\mathcal{R}_{\mathsf{step}}^{(i)}$ *takes* π_{i-1} *as a public witness component, the proof at each depth* i *can be cached and reused across all derivations that share the same prefix* $(i_1, \ldots, i_i)$*. Concretely, a wallet maintaining a checkpoint* $(\pi_{\mathsf{step}}^{(i)}, A_i, \mathsf{com}_{k_R}^{(i)}, c_i)$ *at depth* i *can derive any descendant key at depth* $i + k$ *by running only* k *additional STARK proofs (one per step, independently of* i*). No private key material appears in the checkpoint, so it can be stored on untrusted media without compromising the seed* x_0 *or any derived private key. This reduces the amortised proving cost in a typical HD wallet tree (where most keys share long common prefixes) to a small constant number of STARK proofs per fresh address.*

11 Implementation and Performance

11.1 Implementation

We implemented the scheme Σ' of Section 7.2 in Rust using the RISC Zero zkVM[2] as the underlying STARK backend. The implementation is structured as three crates. The `core` crate is a library shared between the host and the guest; it provides the BIP32-Ed25519 key derivation of Section 5.1 (via `HMAC-SHA512` from the `hmac`/`sha2` crates and scalar arithmetic over Ed25519 via `curve25519-dalek`) and the relation checker `check_relation`, which verifies all five conditions (i′)–(v′) of $\mathcal{R}'$. The `methods` crate compiles the guest program to the RISC Zero instruction set; the guest reads the pair $(\mathbf{x}', \mathbf{w}') = ((M, A_n), (x_0, k_{L_n}, k_{R_n}, h))$ from the private input channel, calls `check_relation`, and commits only (M, A_n, n) to the public journal. The `host` crate implements Gen, Sign', and Verify': Sign' invokes the RISC Zero external prover in succinct mode, producing a constant-size STARK receipt; Verify' deserialises the receipt, calls `receipt.verify(image_id)`, and checks that the journal's public key and message match the expected values.

11.2 Performance

We used the following hardware to execute the performance evaluation.

Component	Specification
CPU	16 cores / 32 threads, 4.5 GHz
RAM	64 GB DDR5-5600 ECC
OS	Ubuntu (Linux)

We measured three operations at derivation depths $n \in \{2, 7, 12, \ldots, 147\}$:

- Gen: key generation.
- Sign': STARK proof generation.
- Verify': receipt verification.

For each depth we record wall-clock proving time, verification time, the RISC Zero guest cycle breakdown (derivation chain, Ed25519 scalar multiplication, SHA-256 hash, journal commit), total and user cycle counts, STARK segment count, peak RAM, and CPU utilisation during proving. The

[2] https://github.com/risc0/risc0

signature size is the size of the serialised STARK receipt, which is constant at 218.4 KB across all depths (succinct/constant-size STARK).

Table 1 reports key generation time, proving time, verification time, total and user cycle counts, STARK segment count, peak RAM, and CPU utilisation. Table 2 gives the per-depth guest cycle breakdown for a representative subset of depths.

Depth n	**Gen** (ms)	**Prove** (s)	**Verify** (ms)	**Total cycles**	**User cycles**	**Segs**	**RAM (MB)**	**CPU (%)**
2	<1	199.7	9	4 194 304	3 504 965	4	6.6	2863.4
7	<1	502.0	10	10 485 760	9 424 925	10	7.9	2880.5
12	<1	807.7	9	16 777 216	15 344 939	16	8.0	2877.7
17	<1	1111.0	9	23 068 672	21 264 899	22	8.0	2877.2
22	<1	1425.0	9	29 425 664	27 184 859	29	8.0	2877.1
27	<1	1735.7	9	35 913 728	33 104 819	35	8.0	2878.9
32	<1	2039.9	9	42 205 184	39 024 779	41	8.0	2878.1
37	<1	2357.3	9	48 758 784	44 944 739	47	8.1	2878.1
42	<1	2663.6	9	55 050 240	50 864 699	53	8.1	2877.6
47	<1	2984.3	9	61 865 984	56 784 659	59	8.1	2879.1
52	<1	3291.9	9	68 157 440	62 704 619	65	7.9	2879.3
57	<1	3595.1	9	74 448 896	68 624 579	71	7.9	2877.9
62	<1	3891.5	9	80 740 352	74 544 539	77	7.9	2878.6
67	<1	4194.5	9	87 031 808	80 464 499	83	7.9	2878.7
72	<1	4492.4	9	93 323 264	86 384 459	89	7.9	2879.4
77	<1	4807.0	9	99 745 792	92 304 419	96	7.9	2878.3
82	<1	5118.0	9	106 168 320	98 224 379	102	7.9	2878.8
87	1	5439.7	9	112 721 920	104 144 405	108	8.0	2878.2
92	<1	5753.2	10	119 013 376	110 064 365	114	8.0	2879.1
97	<1	6048.4	9	125 304 832	115 984 325	120	8.0	2878.3
102	1	6380.8	9	132 120 576	121 904 287	126	8.2	2878.0
107	1	6690.7	9	138 412 032	127 824 247	132	8.3	2878.9
112	1	7000.3	9	144 703 488	133 744 207	138	8.4	2878.8
117	1	7299.9	9	150 994 944	139 664 167	144	8.4	2879.2
122	1	7596.1	9	157 286 400	145 584 127	150	8.4	2878.8
127	1	7917.4	9	163 708 928	151 504 087	157	8.4	2878.3
132	1	8222.1	9	170 131 456	157 424 047	163	8.4	2878.8
137	1	8521.1	9	176 685 056	163 344 007	169	8.4	2879.2
142	1	8815.1	9	182 976 512	169 263 967	175	8.4	2878.4
147	1	9137.9	9	189 267 968	175 183 927	181	8.4	2878.6

Table 1: Performance of Σ' at varying derivation depths ($n = 2$ to 147, steps of 5). Gen time is wall-clock time for BIP32-Ed25519 key generation; Prove time is wall-clock time for `sign::5_stark_prove`. CPU utilisation is averaged over all 32 logical threads during proving (maximum 3200 %). RAM reports peak resident-set size.

Key generation time. Gen completes in under 1 ms at all measured depths up to $n = 97$, and in 1 ms for $n \geq 102$, confirming that the BIP32-Ed25519 key derivation cost is entirely negligible relative to STARK proof generation at every depth.

Proving time. Signing time is dominated by the STARK proof-generation step and grows linearly with n. Across 30 measured depths from $n = 2$ to $n = 147$, a linear fit yields $\mathsf{Prove}(n) \approx 61.7 \cdot n + 69.5$ seconds, with residuals below 7 s throughout, confirming that the relationship is very tightly linear. Concretely, proving time grows from approximately 200 s at $n = 2$ to approximately 9 138 s (≈2.5

Depth n	derive_chain	public_key (Ed25519)	compute_h (SHA-256)	env::commit	Grand total
2	2 388 169	1 061 134	4 751	5 924	3 467 449
7	8 307 284	1 061 134	4 751	5 924	9 386 589
12	14 226 399	1 061 134	4 751	5 924	15 305 787
22	26 064 629	1 061 134	4 751	5 924	27 144 067
32	37 902 859	1 061 134	4 751	5 924	38 982 347
52	61 579 319	1 061 134	4 751	5 924	62 658 907
72	85 255 779	1 061 134	4 751	5 924	86 335 467
97	114 851 354	1 061 134	4 751	5 924	115 931 195
122	144 446 929	1 061 134	4 751	5 924	145 526 897
147	174 042 504	1 061 134	4 751	5 924	175 122 597

Table 2: Guest cycle breakdown at selected depths. The `derive_chain` cost grows linearly with n at $\approx 1.18 \times 10^6$ cycles per unit of n; the Ed25519 scalar multiplication (`public_key_from_scalar`) and SHA-256 (`compute_h`) costs are depth-independent constants of 1 061 134 and 4 751 cycles respectively.

hours) at $n = 147$. This reflects the linear growth of the `derive_chain` cycle count, which increases at $\approx 1.18 \times 10^6$ cycles per unit of n. By contrast, the Ed25519 scalar multiplication contributes a fixed 1 061 134 cycles and the SHA-256 hash a fixed 4 751 cycles, both entirely independent of depth (Table 2).

Parallelism and CPU utilisation. RISC Zero's prover uses Rayon for data-parallel work-stealing across available threads. On our 32-thread machine (3 200 % theoretical maximum), CPU utilisation during `stark_prove` is stable across the entire depth range, averaging ≈2 878 % (≈89.9 % of maximum, corresponding to ≈28.8 active threads) from $n = 7$ upward. The shallow case $n = 2$ records a slightly lower 2 863 % (≈89.5 %), but the difference is small and consistent with early-stage segment scheduling before the trace is large enough to fully saturate the thread pool. Across the extended depth range, CPU utilisation is essentially flat, indicating that Rayon's work-stealing scheduler reaches a stable operating regime early and maintains it regardless of derivation depth or segment count. The persistent ≈10 % gap from the theoretical maximum is attributable to inter-segment synchronisation overhead inherent to RISC Zero's segmented execution model, but its cause has not been further isolated.

Prover throughput. The user-cycles-per-second metric is nearly constant across the full depth range, rising modestly from ≈17 600 cycles/s at $n = 2$ to ≈19 200 cycles/s at $n = 147$. The slight improvement at greater depths reflects larger traces providing proportionally more arithmetic work per synchronisation event, marginally improving efficiency. The near-flatness of throughput confirms that proving time is essentially proportional to cycle count throughout, with no significant regime change as depth increases.

Verification time. Verification is constant at 9–10 ms across all 30 measured depths, confirming the succinct, constant-size property of the STARK receipt independently of derivation depth. The isolated 10 ms reading at $n = 7$ and $n = 92$ is consistent with OS scheduling noise rather than any circuit-size or depth-dependent effect.

Memory. Peak resident-set RAM grows from 6.6 MB at $n = 2$ to a stable 8.4 MB from $n = 112$ onward, and does not exceed 8.4 MB at any measured depth. This confirms that the proving

bottleneck is computational rather than memory-bound. No swap activity was observed at any depth, and the low memory footprint means that the prover can run on commodity hardware without specialised memory configurations.

Paging overhead. RISC Zero segments the execution trace into power-of-two-sized chunks. The segment count grows from 4 at $n = 2$ to 181 at $n = 147$, and the associated paging cycle overhead grows from ≈689 000 to ≈14 084 000 cycles. The paging fraction of total cycles starts at 16.4 % at $n = 2$, then stabilises at approximately 7.5 % for all $n \geq 7$ and remains flat through $n = 147$. This indicates that paging cost scales proportionally to trace size once the trace is large enough to amortise the fixed per-segment overhead, and does not grow disproportionately at greater depths.

Signature size. The serialised STARK receipt is constant at 218.4 KB regardless of derivation depth, consistent with the succinct mode of the RISC Zero prover.

12 Split-Proof Performance

We report benchmarks for the split-proof architecture of Section 9.

12.1 Performance

We measured derivation proof generation (`split_deriv`) at depths $n \in \{1, \ldots, 10\}$ and signing proof generation (`split_sign`) at depths $n \in \{2, \ldots, 9\}$.

Derivation proofs ($\mathcal{R}_{\text{deriv}}$). Table 3 reports results for `split_deriv`. Proving time grows roughly linearly. All receipts serialise to a constant 218.5 KB.

Table 3: Performance of $\mathcal{R}_{\text{deriv}}$ (`split_deriv`) at depths $n = 1$–10.

Depth n	Cycles	Prove (s)	Size (KB)
1	2 293 157	427.7	218.5
2	3 477 177	674.1	218.5
3	4 661 097	842.9	218.5
4	5 845 117	579.8	218.5
5	7 029 187	609.8	218.5
6	8 213 007	679.8	218.5
7	9 396 977	790.4	218.5
8	10 581 047	872.9	218.5
9	11 764 917	920.9	218.5
10	12 948 915	682.9	218.5

Signing proofs ($\mathcal{R}_{\text{sign}}$). Table 4 reports results for `split_sign`. Because $\mathcal{R}_{\text{sign}}$ checks only two SHA-256 invocations.

Verification. The combined verification step checks both π_{deriv} and π_{sign} against the shared public value $com_{k_R} = \text{SHA256}(k_{R_n})$. As shown in Table 4, signing-proof verification is constant at 18.9–19.7 ms across all depths.

Table 4: Performance of $\mathcal{R}_{\text{sign}}$ (`split_sign`) at depths $n = 2$–9.

Depth n	Cycles	Prove (s)	Size (KB)	Verify (ms)
2	35 473	9.7	218.8	19.0
3	35 423	9.4	218.8	18.9
4	35 473	9.4	218.8	19.7
5	35 573	9.4	218.8	19.0
6	35 423	9.5	218.8	19.2
7	35 423	9.4	218.8	19.0
8	35 523	9.4	218.8	19.1
9	35 423	9.3	218.8	18.9

13 End-to-End User Flow on Blockchains

13.1 Deployment on UTXO Chains

The scheme Σ' is applicable to UTXO-based chains such as Bitcoin, subject to two protocol-level prerequisites. First, the chain must adopt Edwards25519 as its signature curve. Second, verification of the STARK receipt π requires a new script opcode, analogous to `OP_CHECKSIG` for classical signatures, that validates a STARK receipt against a public statement. On Bitcoin this would require a soft fork introducing such an opcode. Chains with flexible upgrade paths may require a fast path.

Once these prerequisites are met, the split-proof architecture of Section 9 maps cleanly onto the UTXO model: π_{deriv} is computed once per address. Each spend provides only $(\pi_{\text{deriv}}, \pi_{\text{sign}})$ and the classical signature σ_{classic}.

The main open problem for UTXO deployment is the efficient on-chain storage of the Σ' proof. The current STARK receipt size is 218.4 KB (Table 1), which is large relative to a typical Bitcoin transaction.

A possible direction to analyze in order to reduce the on-chain footprint is the construction of a succinct attestation. Concretely, the outer circuit would take the full STARK receipt π as a witness and certify that $\text{STARK.Verify}(\mathcal{R}', (M, A_n), \pi) = 1$. Two possible approaches worth investigating are: instantiating the outer proof system with a Poseidon-FRI-based STARK, or using a hash-based transparent argument such as Ligero. Both retain post-quantum knowledge soundness under the collision resistance of the underlying hash function and require no trusted setup. Replacing the FRI Merkle hash with a ZK-friendly function such as Poseidon would reduce the outer circuit complexitys. The efficient instantiation of any such outer argument and its integration into the UTXO transaction format remain open problems that we leave for future work.

13.2 End-to-End User Flow via the AmericanFortress Protocol

We describe how Σ' integrates with the AmericanFortress Protocol (AFP) signaling infrastructure [25] to deliver post-quantum-secured funds on the sending side without requiring any change to the user workflow. The key observation is that Σ' produces public keys identical to those of a standard BIP32-Ed25519 wallet: a sender's spending address is therefore simultaneously a classical Ed25519 address and a Σ' post-quantum address, with no additional key material or separate derivation path required.

The AFP signaling layer [25] permits posting signals as encrypted metadata in on-chain transactions, so neither party needs to be online simultaneously. Bob's wallet recovers all incoming signals

when it next comes online by scanning the chain and decrypting with his AFP private key. The AFP signaling layer provides cryptographic unlinkability between signals and their recipients: third-party observers cannot associate any signal with a specific transacting party or determine whether a Σ' descriptor is present.

13.3 Send Phase

1. **Address derivation.** Alice's wallet computes Bob's stealth receive address via the AFP Public Paycode mechanism [25]. Using Bob's publicly available Paycode and an EC Diffie-Hellman operation, Alice derives a fresh one-time address A_n^{recv} that is unique to the Alice-Bob pair. This stealth address prevents blockchain graph analysis from linking multiple incoming payments to Bob's identity, as third-party observers cannot compute A_n^{recv} from Bob's Paycode alone.

   Alice's own spending address, from which she sends the funds, is a standard BIP32-Ed25519 derived key $A_n^{\text{Alice}} = k_{L_n} \cdot B$. Because Σ' preserves the BIP32-Ed25519 public key format exactly, this address simultaneously admits two spending proofs:

   - σ_{classic}: a standard Ed25519 signature, valid on any BIP32-Ed25519-compatible chain;
   - π: a Σ' STARK proof certifying post-quantum knowledge of the seed and full derivation chain underlying A_n^{Alice}.

2. **Signaling.** Alice's wallet posts a single encrypted signal to the blockchain as transaction metadata following the AFP on-chain signaling protocol [25]. The signal is encrypted under Bob's public key and carries Alice's Public Paycode.

3. **On-chain settlement.** Alice broadcasts the funding transaction, spending from A_n^{Alice} and sending to A_n^{recv}, in the standard way adding π to the transaction content.

13.4 Receive Phase

1. **Signal recovery.** When Bob's wallet comes online, it scans the transaction pool on the blockchain and attempts decryption of all signals using his AFP private key.

2. **Stealth address reconstruction.** For each successfully decrypted signal, Bob's wallet retrieves Alice's Public Paycode and, using his own private key material, reconstructs the stealth receive address A_n^{recv} via the same mechanism. Bob also derives the corresponding private key, which he will need to spend the received funds.

13.5 Spend Phase

Alice spending from her own wallet. When Alice spends from her BIP32-Ed25519 derived address A_n^{Alice}, her wallet produces a *dual-mode* spending proof.

1. **Classical signature.** Alice's wallet produces a standard Ed25519 signature σ_{classic} under A_n^{Alice}, satisfying existing node validation rules on any chain that has not yet adopted Σ' verification.

2. **Post-quantum proof.** Alice's wallet produces the Σ' STARK proof π certifying knowledge of the seed and full BIP32-Ed25519 derivation chain underlying A_n^{Alice}, as described in Section 7.2. If the split-proof architecture of Section 9 is employed, π_{deriv} is precomputed once at wallet

initialisation time and only π_{sign} is generated at spend time, reducing per-transaction proving cost.

3. **Submission and verification.** Alice submits $(\sigma_{\text{classic}}, \pi)$ to the network. Nodes that support Σ' verification check both components, and nodes that do not yet support it fall back to σ_{classic} alone. This ensures backward compatibility until all nodes switch to a postquantum secure solution.

Alice spending from a stealth address. Extending post-quantum protection to Bob's spend from a stealth address A_n^{recv} requires a modification to the NP relation R'. Concretely, one would augment the public statement with the tweak s and replace condition (iv') with the relaxed binding $A_n^{\text{recv}} = (k_{L_n} + s) \cdot B$, leaving all other conditions (i')–(iii') and (v') unchanged. The intuition is that finding any (x_0, s) satisfying this relaxed condition for a given A_n^{recv} remains hard: once x_0 is fixed, k_{L_n} is uniquely determined by the BIP32-Ed25519 derivation chain, so an adversary wishing to forge a valid witness for a target A_n^{recv} must still invert SHA-512 to recover a valid x_0. Whether this relaxed relation admits a clean formal security proof, and whether it introduces any subtle interaction with the AFP Diffie-Hellman derivation of s, requires further investigation.

14 Conclusion

The advent of large-scale quantum computers will pose a concrete threat to the security of blockchain wallets based on classical elliptic-curve cryptography. BIP32-Ed25519 hierarchical deterministic wallets are secure under the discrete logarithm problem and will therefore require migration to post-quantum alternatives. This migration is complicated by the fact that changing the underlying signature scheme typically entails changing public keys and addresses, which requires coordination across users, wallets, and blockchain protocols.

In this work we have shown that it is possible to take a meaningful first step toward post-quantum security without breaking compatibility with existing BIP32-Ed25519 infrastructure. By replacing the classical Ed25519 signing step with a ZK-STARK proof of knowledge of the seed and derived key material, our scheme Σ' produces public keys that are identical to those of a standard BIP32-Ed25519 wallet following the same derivation path. No address migration or key re-registration is required, and verification remains efficient regardless of derivation depth.

The main practical limitation of the current implementation is proving cost: signing requires on the order of minutes to hours depending on derivation depth, which makes the scheme unsuitable for interactive use cases where a fresh signature must be produced for each transaction. This cost is intrinsic to the use of HMAC-SHA512 inside the STARK circuit and to the current state of general-purpose zkVM proving hardware. Two directions may alleviate this in future work. First, the split-proof architecture of Section 9 separates the derivation proof, which depends only on the seed and can be computed once when the wallet is initialised, from the signing proof, which depends only on the message and SHA-256 invocations and is independent of derivation depth. Precomputing the derivation proof at key-generation time would reduce the per-transaction proving cost to a small, depth-independent circuit, making the scheme significantly more practical on current hardware. Second, replacing HMAC-SHA512 with a ZK-friendly hash function such as Poseidon [5] would reduce circuit complexity substantially and bring proving times into a practical range. This direction comes at the cost of BIP32-Ed25519 compatibility: wallets using a different hash function produce different public keys and require address migration, but may be acceptable in settings where compatibility with the existing key infrastructure is not a hard requirement.

We also encourage blockchain ecosystems currently based on the secp256k1 curve, most notably Bitcoin and Ethereum, to consider migration to Edwards curves such as Edwards25519. The scheme presented in this work is specific to the BIP32-Ed25519 key derivation and cannot be directly applied to secp256k1-based wallets without a curve migration. Edwards curves offer well-studied security properties, efficient and constant-time arithmetic, and a growing ecosystem of cryptographic tooling. A migration from secp256k1 to an Edwards curve would enable the post-quantum upgrade path described in this work.

Finally, the scheme provides a natural bridge to a fully post-quantum wallet infrastructure when blockchain protocols eventually adopt native post-quantum signature schemes. At that point, the seed x_0 can be used as a zero-knowledge witness to authenticate the transition: following the approach of Baldimtsi et al. [15], a ZK proof can bind each existing Ed25519 address independently to a fresh post-quantum verification key without revealing x_0. The resulting post-quantum wallet hierarchy can then be instantiated as a full post-quantum HDW, for example following Deegan et al. [4].

References

[1] Nabil Alkeilani Alkadri, Poulami Das, Andreas Erwig, Sebastian Faust, Juliane Krämer, Siavash Riahi, and Patrick Struck. Deterministic wallets in a quantum world. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020: 27th Conference on Computer and Communications Security*, pages 1017–1031. ACM Press, November 2020.

[2] Poulami Das, Andreas Erwig, Michael Meyer, and Patrick Struck. Efficient post-quantum secure deterministic threshold wallets from isogenies. In Jianying Zhou, Tony Q. S. Quek, Debin Gao, and Alvaro A. Cárdenas, editors, *ASIACCS 24: 19th ACM Symposium on Information, Computer and Communications Security*. ACM Press, July 2024.

[3] Surbhi Shaw and Ratna Dutta. Post-quantum secure compact deterministic wallets from isogeny-based signatures with rerandomized keys. *Theoretical Computer Science*, 1035:115127, 2025.

[4] Conor Deegan, James Fitzwater, Kamil Doruk Gur, and David Nugent. Lattice HD wallets: Post-quantum BIP32 hierarchical deterministic wallets from lattice assumptions. Cryptology ePrint Archive, Paper 2026/380, 2026.

[5] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021: 30th USENIX Security Symposium*, pages 519–535. USENIX Association, August 2021.

[6] Dmitry Khovratovich and Jason Law. Bip32-ed25519: Hierarchical deterministic keys over a non-linear keyspace. In *2017 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 27–31, 2017.

[7] Poulami Das, Sebastian Faust, and Julian Loss. A formal treatment of deterministic wallets. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, *ACM CCS 2019: 26th Conference on Computer and Communications Security*, pages 651–668. ACM Press, November 2019.

[8] Poulami Das, Andreas Erwig, Sebastian Faust, Julian Loss, and Siavash Riahi. The exact security of BIP32 wallets. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021: 28th Conference on Computer and Communications Security*, pages 1020–1042. ACM Press, November 2021.

[9] Adriano Di Luzio, Danilo Francati, and Giuseppe Ateniese. Arcula: A secure hierarchical deterministic wallet for multi-asset blockchains. In Stephan Krenn, Haya Shulman, and Serge Vaudenay, editors, *CANS 20: 19th International Conference on Cryptology and Network Security*, volume 12579 of *Lecture Notes in Computer Science*, pages 323–343. Springer, Cham, December 2020.

[10] Andreas Erwig and Siavash Riahi. Deterministic wallets for adaptor signatures. In Vijayalakshmi Atluri, Roberto Di Pietro, Christian Damsgaard Jensen, and Weizhi Meng, editors, *ESORICS 2022: 27th European Symposium on Research in Computer Security, Part II*, volume 13555 of *Lecture Notes in Computer Science*, pages 487–506. Springer, Cham, September 2022.

[11] Chenghe Dong, Jianhong Zhang, Zongyi Lv, and Ruxuan Zhang. Lightweight hierarchical deterministic wallet supporting stealth address for iot. In *2023 IEEE 22nd International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 387–393, 2023.

[12] Xin Yin and Zhen Liu. A secure hierarchical deterministic wallet with stealth address from lattices. *Theoretical Computer Science*, 1009:114672, 2024.

[13] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Report 2018/046, 2018.

[14] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 2087–2104. ACM Press, October / November 2017.

[15] Foteini Baldimtsi, Konstantinos Chalkias, Arnab Roy, and Mahdi Sedaghat. Post-quantum readiness in EdDSA chains. Cryptology ePrint Archive, Paper 2025/1368, 2025.

[16] Kostas Kryptos Chalkias. Poster: Post-quantum readiness in eddsa chains. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, CCS '25, page 4809–4811, New York, NY, USA, 2025. Association for Computing Machinery.

[17] Avihu Mordechai Levy. Quantum-safe Bitcoin transactions without softforks. Cryptology ePrint Archive, 2026. Available at https://eprint.iacr.org/2026/xxx.

[18] Michel Seck and Adeline Roux-Langlois. Towards post-quantum bitcoin blockchain using dilithium signature. *IACR Communications in Cryptology (CiC)*, 2(3):3, 2025.

[19] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. In Henri Gilbert, editor, *Advances in Cryptology – EUROCRYPT 2010*, volume 6110 of *Lecture Notes in Computer Science*, pages 1–23. Springer, Berlin, Heidelberg, May / June 2010.

[20] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. On ideal lattices and learning with errors over rings. *J. ACM*, 60(6), November 2013.

[21] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. CSIDH: An efficient post-quantum commutative group action. In Thomas Peyrin and Steven Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018, Part III*, volume 11274 of *Lecture Notes in Computer Science*, pages 395–427. Springer, Cham, December 2018.

[22] Phillip Gajland, Bor de Kock, Miguel Quaresma, Giulio Malavolta, and Peter Schwabe. SWOOSH: Efficient lattice-based non-interactive key exchange. In Davide Balzarotti and Wenyuan Xu, editors, *USENIX Security 2024: 33rd USENIX Security Symposium*. USENIX Association, August 2024.

[23] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th Annual Symposium on Foundations of Computer Science*, pages 124–134. IEEE Computer Society Press, November 1994.

[24] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *28th Annual ACM Symposium on Theory of Computing*, pages 212–219. ACM Press, May 1996.

[25] Michal "Mehow" Pospieszalski and Jakub Zurawinski. $afe af: User-friendly privacy-preserving protocol for compliant cryptocurrency transactions. White paper, MatterFi Inc., Wyoming, USA, 2025. Accessed: 2026-05-06.