ZKPoSP: Post-Quantum Zero-Knowledge Proofs for Hierarchical Deterministic Wallets

Vincenzo Botta Michal Pospieszalski Emanuele Ragnoli
Justus Ranvier

AmericanFortress, Jackson Hole, Wyoming
vincenzo@matterfi.com, mehow@matterfi.com, emanuele@matterfi.com, justus@matterfi.com

July 29, 2026

Abstract

Recent advances in quantum hardware, including Google's Willow processor, have substantially narrowed the timeline to cryptographically relevant quantum computers. In the blockchain setting, where addresses and key derivation standards such as BIP32, BIP44, and SLIP-10 are the dominant infrastructure for wallet management, a quantum computer running Shor's algorithm can recover any elliptic-curve private key from the corresponding public key, threatening every wallet in production today. Migrating to post-quantum signature schemes requires changing the public key format and forcing address migration across all participating networks, a significant problem for blockchain communities. We present an orthogonal approach: keep the existing address format entirely unchanged and instead replace the classical signing step with a NIZK proof of knowledge of the seed underlying the existing address, where security against quantum adversaries reduces to the conjectured quantum hardness of the underlying hash functions and the soundness of the NIZK against quantum adversaries, requiring no address migration or key registration.

We build on the observation of Baldimtsi et al. that EdDSA's deterministic seed-to-key mapping makes the seed a valid zero-knowledge witness for the public key, and extend their single-level result to the full hierarchical deterministic wallet setting. Starting from BIP32-Ed25519, we replace the classical signing step with a NIZK proof certifying knowledge of the root seed and the full derivation chain, and prove EUF-CMA security for the resulting scheme; post-quantum security is conjectured to hold since all underlying primitives depend only on the hardness of hash functions and the soundness of the NIZK against quantum adversaries.

The existing schemes each derive their quantum-safe witness as an incidental artifact of a curve-specific key format, and none provides a single derivation standard that works uniformly across curves. We therefore introduce QBIP32, a new key derivation scheme based on a keyed function HASH768 (instantiated with KMAC256) that produces the signing scalar, an explicit quantum-safe witness, and the chain code in a single call. QBIP32 is defined for any elliptic curve group of prime order with a fixed generator, requiring no structural change to the derivation or proof system across curves: the same construction covers secp256k1, Ed25519, and any future curve used in blockchain infrastructure. This universality stands in contrast to the BIP32-Ed25519 approach, which relies on the Ed25519 extended key format and has no analogue for other curves.

We then address the efficiency problem: a monolithic proof of the full derivation chain has cost growing linearly with derivation depth. Our main contribution is ZKPoSP (Zero-Knowledge Proof of Seed Provenance), a signature scheme conjectured secure against quantum adversaries that splits the proof into a derivation proof generated once per key pair and a signing proof generated once per message, reducing per-message proving cost to a constant independent of derivation

depth. We further characterise exactly when the derivation proof itself can be shortened to cover only the last step of the derivation rather than the full chain from the root seed, and identify the existence of a private value that is bound to the seed by a one-way function and not recoverable from the public key as the precise criterion. This criterion is met by hardened keys but not by non-hardened keys, a structural distinction common to all schemes: BIP32 secp256k1, SLIP-10/Ed25519, BIP32-Ed25519, and QBIP32 all admit a last-step derivation proof for hardened keys, while non-hardened keys require a proof reaching back to the last hardened ancestor. We exploit this for BIP44 paths to prove only one hardened step plus the non-hardened suffix in a single proof, allowing the root seed to be removed from the proving device once the anchor node is generated, while remaining in secure long-term storage.

Before Q-day, separating the derivation and signing proofs already reduces per-transaction cost to a constant independent of derivation depth. After Q-day, once networks reject all non-post-quantum signatures, the leaf scalar can be moved to the public statement, removing all elliptic-curve scalar multiplications from the proof circuit and reducing derivation proving time substantially.

We implement all constructions in Rust using RISC Zero as the NIZK backend, instantiate HASH768 with KMAC256, and report benchmarks for monolithic proofs, ZKPoSP across full BIP44 paths, and the post-Q-day variant. Signing proving time is constant at approximately 12–13 seconds and verification time is constant at approximately 9–10 ms across all variants and depths.

Contents

1 Introduction

Hierarchical deterministic wallets (HDW), standardised in BIP32 [Wui12] and its Ed25519 adaptation BIP32-Ed25519 [KL17], are the dominant 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 recovering the full wallet from the seed alone. The security of all used HD wallet schemes is based on on the discrete logarithm problem, which Shor's algorithm [Sho94] solves in polynomial time on a quantum machine.

Post-quantum signing without address migration. Lattice-based and hash-based post-quantum wallet schemes [ADE+20, DEMS24, SD25, DFGN26] change the public key format, forcing address migration and protocol updates across all participating blockchains and wallets. An preferable approach should leave the public key format entirely unchanged and instead replace the classical signing step with a NIZK proof of knowledge of the seed underlying the existing address (under the assumption that the specific NIZK soundness holds against quantum adversaries). This approach was pioneered by Baldimtsi et al. [BCR25] for EdDSA and forms the starting point of the present work for BIP32-Ed25519. The schemes presented in this paper support Ed25519, secp256k1, and any elliptic curve of prime order, and are designed for compatibility with existing infrastructure: public keys have the same format as those produced by any standard BIP32-Ed25519 or BIP32 wallet following the same type of derivation path, requiring no address migration or key registration.

The proving cost problem. To achieve a post-quantum signature for a key derived at depth n, the signer must convince the verifier that the public key was honestly derived from a secret seed following the prescribed derivation path. A natural idea would be to prove only the last parent-to-child derivation step, binding the child key to its immediate parent without re-deriving the entire path. This is however insufficient in general: for non-hardened keys, all inputs to the derivation function are public, so an adversary (with a quantum machine) can freely pick any fake parent state, compute the correct child outputs from public data alone, and produce a passing proof with no connection to the legitimate owner's seed. A sound proof must therefore cover the full derivation chain from the root seed to the signing key, establishing an unbroken link that cannot be forged. A monolithic NIZK proving this full chain has proving time growing linearly with derivation depth, reaching hours for deep paths. Our main contribution is a signature scheme that resolves this problem by splitting the proof into two independent parts: a derivation proof π_{deriv}, computed once per key pair and certifying that the public key was honestly derived from a secret seed, and a signing proof π_{sign}, computed once per message at a cost independent of derivation depth. We call this signature scheme ZKPoSP (Zero-Knowledge Proof of Seed Provenance).

The quantum-safe witness question. ZKPoSP requires π_{deriv} to cover the full derivation chain from the root seed, but in practice wallets rarely use purely non-hardened paths. The BIP44 standard, which is the dominant convention in blockchain infrastructure, prescribes paths of the form $m/44'/coin'/account'/change/index$ consisting of three hardened levels followed by two non-hardened levels. For hardened derivation steps, the inputs to the derivation function include private parent key material, so a fake parent state cannot be freely constructed even by a quantum adversary without inverting a one-way function. This raises the question: if the derivation path contains hardened nodes, can the derivation proof be shortened to start from the last hardened node rather than from the root seed, treating that node as a trusted anchor for the remaining non-hardened suffix? The answer depends on whether the derivation chain contains a quantum-safe witness at the

hardened anchor, that is, a private value that is bound to the seed by a one-way function and not recoverable from the public key even by a quantum adversary. We characterise this for four schemes.

- SLIP-10/Ed25519, BIP32 secp256k1, and BIP32-Ed25519 have a quantum-safe witness for hardened keys only. Non-hardened keys are vulnerable to an attack caused by a variable that can be frely chosen by the adversary.
- QBIP32 (introduced here) provides a quantum-safe witness for any prime-order elliptic curve group by construction, enabling shortened derivation proofs for hardened keys on secp256k1, Ed25519, and any other curve used in blockchain infrastructure, with no structural change to the derivation or proof system.

For SLIP-10/Ed25519, BIP32-Ed25519, and QBIP32, the existence of a quantum-safe witness at hardened nodes means that a single NIZK can cover the last hardened step plus the two non-hardened levels of a BIP44 path, with the last hardened node serving as the anchor. The derivation proof is therefore a fixed-size, depth-independent computation regardless of how deep the hardened prefix is, and the root seed no longer needs to reside on the proving device[1] once the anchor node has been generated and stored.

Two operational modes. A wallet holder using ZKPoSP can operate in two modes. In *classical mode*, the wallet uses existing Ed25519 or secp256k1 signatures directly, remaining fully compatible with current network infrastructure but providing no post-quantum security guarantee. In *post-quantum mode*, the classical signature is replaced by a ZKPoSP signature pair $(\pi_{\mathsf{deriv}}, \pi_{\mathsf{sign}})$: the derivation proof attests that the public key was honestly derived from a secret seed following the prescribed path, and the signing proof binds the message to the same derived key material without revealing any secret. The derivation proof is generated once per address, ahead of any transaction, and does not need to be repeated. The signing proof is generated once per message. Funds recorded on-chain today under existing addresses are not immediately at risk from this transition: no address migration is required, and the ZKPoSP signature is produced by the same wallet seed that controls the existing address.

The post-Q-day variant. The constructions above address *what* the derivation proof must certify. A separate cost question concerns *how expensive* that certification is: in all our constructions, the derivation proof must establish a link between the secret signing scalar used in the underlying BIP32 or SLIP-10 key derivation scheme and its corresponding public key, which requires proving a scalar multiplication inside the NIZK. This is necessary because the secret signing scalar must remain hidden: if it were exposed, a classical adversary could forge signatures directly without any quantum capability. Once a sufficiently large quantum computer exists, however, any classical signature should be rejected by the network regardless of its validity, so an adversary who learns the signing scalar gains nothing: they cannot produce an accepted signature outside a post-quantum scheme anyway. Under this post-Q-day model, there is no reason to hide the signing scalar, and it can be moved directly into the public statement of the derivation proof. The verifier then checks the scalar-to-public-key consistency classically outside the NIZK, removing all scalar multiplications from the proof circuit and reducing derivation proving time.

[1] I.e., the seed can be deleted from the device that generates proofs and retained in secure long-term storage (e.g., an offline backup or hardware security module), since it remains necessary to derive any new branch of keys.

1.1 Contributions

1. ZKPoSP: a post-quantum signature scheme for HD wallets. Our main contribution is ZKPoSP, a signature scheme for hierarchical deterministic wallets that replaces the classical signing step with a NIZK proof of knowledge of a seed, requiring no address migration or new key registration. ZKPoSP splits the proof into a derivation proof π_{deriv}, generated once per key pair, and a signing proof π_{sign}, generated once per message at a cost independent of derivation depth (Section 5). We prove EUF-CMA and hierarchical EUF-CMA security for ZKPoSP, working in the random oracle model for SHA-256 and SHA-512; post-quantum security is conjectured to hold since the proofs do not rely on any assumption known to be broken against quantum adversaries.

2. Monolithic construction and security proof for BIP32-Ed25519. As a didactic stepping stone toward ZKPoSP, we define a monolithic signature scheme Σ for BIP32-Ed25519 and prove EUF-CMA security via an explicit game-based reduction (Section 4).

3. Analysis of proof shortening. We characterise exactly when the derivation proof can be shortened to cover only the last derivation step rather than the full chain from the root seed, identifying the quantum-safe witness as the precise criterion (Section 6).

4. QBIP32. A new key derivation scheme based on a keyed extensible-output function $\mathsf{HASH768}$, instantiated with KMAC256, that produces scalar, witness, and chain code in a single call and provides a quantum-safe witness for any prime-order elliptic curve group without any structural change to the derivation or proof system (Section 7). This enables ZKPoSP to be instantiated over secp256k1, Ed25519, and any curve used in blockchain infrastructure.

5. Hardened-anchor construction and post-Q-day variant. For BIP44 paths, a single NIZK covers the last hardened step plus the non-hardened suffix, giving a fixed-size derivation proof independent of the hardened-prefix depth in the pruned variant (witness starts from the hardened anchor node); the full variant (witness starts from the root seed) instead re-derives the entire path, with proving cost growing with total path length. The post-Q-day variant further removes all scalar multiplications from the proof relation by moving the leaf scalar to the public statement, reducing derivation proving time once networks reject non-post-quantum signatures.

6. Implementation and benchmarks. All constructions are implemented in Rust using RISC Zero v5.0.0-rc.1, with $\mathsf{HASH768}$ instantiated as KMAC256 for QBIP32 and HMAC-SHA512 as the derivation MAC for BIP32-Ed25519, BIP32-secp256k1, and SLIP-10/Ed25519, on a 16-core machine at 4.5 GHz (Section 9). Signing proving time is constant at approximately 12–13 seconds and verification time is constant at approximately 9–10 ms across all variants, depths, and curves. Benchmarks are reported for monolithic proofs, ZKPoSP across full BIP44 paths (derivation proving approximately 76 seconds for SLIP-10/Ed25519 and approximately 156–158 seconds for BIP32-Ed25519, BIP32-secp256k1, and QBIP32 on both curves), the post-Q-day variant (approximately 21 seconds for BIP32-secp256k1 and SLIP-10/Ed25519, approximately 40 seconds for BIP32-Ed25519 and QBIP32 on both curves), and an analysis of three approaches to incremental derivation via hash-linking, recursion, and folding.

1.2 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 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 child step at depth i computes $z_i = \mathsf{HMAC\text{-}SHA512}(c_{i-1}, \texttt{0x02} \| A_{i-1} \| \langle i_i \rangle_4)$, where $\langle i_i \rangle_4$ denotes the 4-byte little-endian encoding of the child index i_i, 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 $k_{L_i} = k_{L_{i-1}} + 8 \cdot z_L^{(i)} \pmod{2^{256}}$ and $k_{R_i} = k_{R_{i-1}} + z_R^{(i)} \pmod{2^{256}}$. The public key at depth i is $A_i = k_{L_i} \cdot B$.

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 classical Ed25519 signature, the signer produces a NIZK argument of knowledge π attesting 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 an internal binding value $h = \mathsf{SHA256}(k_{R_i} \| M)$ links the chain-derived k_{R_i} to the message. The verifier checks π against the public statement (M, A_i) without learning anything about x_0 or the private key material. We call this scheme Σ. Working in the random oracle model for SHA-256 and SHA-512, security of Σ reduces to the simulation extractability property of the NIZK argument of knowledge (so that an adversary who has seen polynomially many simulated proofs cannot forge a proof for a fresh statement without knowing a valid witness); post-quantum security is conjectured to hold since neither the random oracle model nor the NIZK properties rely on any assumption known to be broken against quantum adversaries. For simplicity, the present construction is shown for non-hardened nodes only; the hardened case, together with the universal treatment across curves, is introduced with QBIP32 in Section 7.

Split-proof architecture. The monolithic NIZK proof for $\mathcal{R}$ grows linearly with derivation depth n. We decouple the two sources of cost by splitting $\mathcal{R}$ into two sub-relations linked by a public hash binding $h_{k_R} = \mathsf{SHA256}(k_{R_n})$: $\mathcal{R}_{\mathsf{deriv}}$ certifies that A_n was honestly derived from a seed and that h_{k_R} is the one-way hash of the chain-derived k_{R_n}; $\mathcal{R}_{\mathsf{sign}}$ certifies, for a given message M, that the signer holds a k_{R_n} consistent with h_{k_R} and that $h = \mathsf{SHA256}(k_{R_n} \| M)$ is correctly computed. The derivation proofs π_{deriv_i} (where i is the index of the derived address) can be precomputed as soon as the seed is known and reused for every subsequent signature under the same address; only π_{sign} needs to be generated at signing time, at a cost of two SHA-256 invocations independent of n.[2]

Shortening the derivation proof and QBIP32. The split-proof architecture still requires π_{deriv} to cover the full chain from the root seed. A natural question is whether this can be improved: rather than re-deriving the entire path, can the prover certify only a single parent-to-child step, treating the parent node as a trusted starting point? This would make the derivation proof a fixed-size computation independent of path depth. The answer depends on a structural property of the derivation chain: such a shortening is sound if and only if the derivation chain contains a private value that is bound to the seed by a one-way function and not recoverable from the public key even by a quantum adversary. If no such value exists, an adversary can freely choose any fake parent state, compute the child outputs from public data alone, and produce a passing proof with no connection to the legitimate seed. We call this value a quantum-safe witness and characterise its existence across four schemes. BIP32 secp256k1 has no quantum-safe witness for any key type: every private scalar is recoverable from its public key via Shor's algorithm. SLIP-10/Ed25519 and

[2]Concretely, the signing proof demonstrates knowledge of a secret value k_{R_n} such that $\mathsf{SHA256}(k_{R_n})$ matches h_{k_R} committed in the derivation proof, and that $\mathsf{SHA256}(k_{R_n} \| M)$ is correctly formed. By the zero-knowledge property of the NIZK argument of knowledge, no secret scalar or private key is ever revealed to the verifier. The verifier checks both proofs and confirms they refer to the same derived key without re-running any derivation or accessing any secret material.

BIP32-Ed25519 have k_{R_n} as a quantum-safe witness for hardened keys only, while for non-hardened keys for BIP32-Ed25519 all HMAC inputs are public and k_{R_n} is freely computable.

To provide a general solution that works for any underlying elliptic curve, including secp256k1, we introduce QBIP32, a new key derivation scheme built around a cryptographic hash function $\mathsf{HASH768} : \{0,1\}^{\lambda} \times \{0,1\}^{*} \to \{0,1\}^{768}$ with 768-bit output, so that every derivation step explicitly produces a quantum-safe witness as one of its outputs. At each derivation step, a single call to $\mathsf{HASH768}$ produces three independent 256-bit blocks: $\mathsf{HASH768}(k, d) = (T_S \,\|\, T_W \,\|\, T_C)$, where T_S is used to update the signing scalar, T_W is the explicit quantum-safe witness increment, and T_C becomes the new chain code. The child node update rule is:

$$NS_{\mathsf{child}} \coloneqq f(NS_{\mathsf{par}}, T_S), \qquad NW_{\mathsf{child}} \coloneqq (NW_{\mathsf{par}} + T_W) \bmod 2^{256}, \qquad NC_{\mathsf{child}} \coloneqq T_C,$$

where f encodes the scalar arithmetic convention of the underlying curve and is the only curve-specific component of the construction. For hardened derivation, the inputs to $\mathsf{HASH768}$ include the private parent values NS_{par} and NW_{par}, so the child witness NW_{child} cannot be computed without knowing them; NW is therefore a quantum-safe witness present by construction at every hardened node for any elliptic curve group, without any structural change to the derivation or proof system across curves. This halves the number of hash invocations per derivation step relative to BIP32-Ed25519, which requires two HMAC calls per step. It is important to notice that in the RISC Zero backend used in this paper, this reduction in primitive calls does not translate into a measured proving-time advantage (see Section 9.2).

Throughout this paper we instantiate $\mathsf{HASH768}$ with KMAC256, the keyed variant of SHA-3 defined in NIST SP 800-185. KMAC256 is based on the Keccak permutation, is FIPS-compliant, and is conjectured to retain its security properties against quantum adversaries.

Hardened-anchor construction and post-Q-day variant. The most common derivation standard in practice is BIP44, which prescribes paths of the form $m/44'/coin'/account'/change/index$ with exactly three hardened levels followed by two non-hardened levels. Since SLIP-10/Ed25519, BIP32-Ed25519, and QBIP32 all admit a shortened derivation proof for hardened keys, we can apply this idea concretely to BIP44: instead of re-deriving the full path from the root seed, the derivation proof covers only the last hardened step plus the two non-hardened levels in a single NIZK, with the last hardened node serving as a trusted anchor for the non-hardened suffix. The depth of the hardened prefix before the anchor is irrelevant to the proof, giving a fixed-size derivation proof regardless of how deep the hardened prefix is. The pruned variant starts the witness from the parent of the hardened anchor node, so the root seed is not needed at proving time. The full variant re-derives the entire path from the root seed inside the NIZK, providing a stronger binding guarantee at higher proving cost. The post-Q-day variant further removes all scalar multiplications from both variants by moving NS_n to the public statement, as discussed above, reducing derivation proving time substantially on the full BIP44 path.

Implementation and performance. We implement all constructions in Rust using RISC Zero as the zkVM backend, with $\mathsf{HASH768}$ instantiated as KMAC256 for QBIP32 and HMAC-SHA512 as the derivation MAC for BIP32-Ed25519, BIP32-secp256k1, and SLIP-10/Ed25519. For the hardened-anchor construction, derivation proving time on the complete BIP44 path $m/44'/0'/0'/0/0$ is approximately 156–158 seconds for the pruned variant of BIP32-Ed25519, BIP32-secp256k1, and QBIP32 on both curves (76 seconds for SLIP-10/Ed25519), and approximately 440–476 seconds for the full variant, the latter growing with total path length. The post-Q-day variant brings derivation proving time on the same path to approximately 40 seconds for BIP32-Ed25519 and for QBIP32 on

both curves, and to approximately 21 seconds for BIP32-secp256k1 and SLIP-10/Ed25519. Signing proving time is constant at approximately 12–13 seconds across all variants and depths; verification is constant at 9–10 milliseconds.

2 Related Work

BIP32 [Wui12] is the dominant HD wallet standard, operating over secp256k1 and supporting both hardened and non-hardened child-key derivation. Non-hardened derivation allows computing child public keys from a parent public key without knowledge of the parent private key, which is central to many wallet architectures but the secret key generated with this protocol are secure under the discrete logarithm problem that is reversible by a quantum adversary. BIP32-Ed25519 [KL17] adapts BIP32 to the Edwards25519 group; unlike BIP32, it multiplies the child increment by 8 to remain within the cofactor-cleared subgroup, restricts the left HMAC output to 28 bytes to prevent scalar overflow at depth, and uses HMAC-SHA512 twice: one call produces a single 512-bit output split between the left 224 bits, which update the signing scalar k_{L_i}, and the right 256 bits, which update the nonce seed k_{R_i} (used both as the EdDSA signing nonce and, at hardened nodes, as the quantum-safe witness); a second, independently keyed call derives the chain code c_i alone. SLIP-0010 [HR16] is a generalisation of BIP32 to multiple curves including Ed25519; its Ed25519 instantiation is hardened-only and derives each step from an HMAC-SHA512 call keyed by the parent chain code, with the chain code itself serving as a private value at hardened nodes.

The first rigorous security treatment of deterministic wallets is due to Das et al. [DFL19], who introduced formal security notions and proved constructions secure under the discrete-logarithm assumption. Das et al. [DEF$^+$21] subsequently tightened the analysis, providing exact security bounds for BIP32 and addressing gaps in the original reduction. The hierarchical EUF-CMA model that we follow, in which derivation keys propagate transitively to descendants and security is defined via a hierarchical experiment, is due to Di Luzio et al. [LFA20]. Erwig and Riahi [ER22] extended this framework to adaptor signatures.

The most direct post-quantum upgrade replaces the underlying signature scheme with a quantum-safe alternative, at the cost of changing the public key format and forcing address migration. Alkadri et al. [ADE$^+$20] proposed the first post-quantum deterministic wallet, instantiating a rerandomisable-key construction using qTESLA and establishing unlinkability and unforgeability notions in the quantum random oracle model. Das et al. [DEMS24] construct efficient post-quantum threshold wallets from isogeny-based signatures with rerandomisable keys, and Shaw and Dutta [SD25] provide a compact variant following the same isogeny-based approach. Deegan et al. [DFGN26] present two constructions that recover BIP32's full derivation functionality: one based on ML-DSA (hardened derivation only) and one based on Raccoon-G (non-hardened derivation), the latter being the first post-quantum wallet with provably secure public-key derivation under standard lattice assumptions. A concurrent work by Seck and Roux-Langlois targets a Dilithium-based construction for Bitcoin-style chains. All of these proposals require address migration; our construction avoids this by preserving the BIP32-Ed25519 public key format exactly.

An orthogonal line of work targets post-quantum security for existing address formats without migration. Baldimtsi et al. [BCR25] observe that EdDSA's deterministic seed-to-key mapping makes the seed a valid zero-knowledge witness for the public key, enabling a post-quantum proof of key ownership without changing the address format. Their construction targets plain single-level EdDSA (RFC 8032) with no derivation chain, using the Ligero proof system [AHIV17] via the Ligetron zkVM, and reports 6.2 seconds proving time and 5.4 MB proof size. Their setting is the degenerate case $n = 0$ of the present framework: no derivation chain, no hardened or non-hardened distinction, no

split-proof architecture, and no hardened-anchor decomposition. A complementary approach in the Bitcoin setting constructs quantum-safe transactions within legacy Script constraints by replacing the ECDSA commitment with a hash puzzle relying on RIPEMD-160 preimage resistance, avoiding any address format change. Consigny et al. [nic26] present SPHINCS$^-$, a family of EVM-optimised variants of SPHINCS+ (SLH-DSA) that replace SHAKE256 with the EVM-native `KECCAK256` opcode, enabling on-chain verification of stateless hash-based signatures at approximately 127K gas without any precompile or protocol change. By combining WOTS+C and FORS+C grinding with aggressively tuned parameter sets, they achieve signature sizes of approximately 3.7 KB and per-key budgets in the range 2^{14}–2^{20}, sufficient for typical wallet usage. Unlike the present work, SPHINCS$^-$ targets direct on-chain verification of a post-quantum signature rather than a zero-knowledge proof of seed ownership, and does not preserve the existing Ed25519 or secp256k1 address format.

3 Preliminaries

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. An adversary $\mathcal{A}$ is called PPT if it runs in probabilistic polynomial time; it is called QPT if it runs in polynomial time ona quantum machine. 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 unless stated otherwise. 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, except where a big-endian encoding is stated explicitly at the point of use (as in the index encodings of the SLIP-10, BIP32-secp256k1, and QBIP32 derivations). 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 Assumptions

Definition 3.1 (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 [Sho94] solves the discrete logarithm problem in polynomial time running on a quantum machine. Therefore the DL assumption does not hold against quantum adversaries.

3.3 Random Oracle Model

Definition 3.2 (Random Oracle Model)**.** A cryptographic scheme Π is secure in the Random Oracle Model (ROM) if for all PPT adversaries $\mathcal{A}$ with oracle access to a truly random function $\mathcal{O} : \{0,1\}^* \to \{0,1\}^\ell$:

$$\Pr[\mathcal{A}^{\mathcal{O}}(\cdot) \text{ breaks } \Pi] \leq \mathsf{negl}(\lambda),$$

where $\mathcal{O}$ is sampled uniformly from the set of all functions $\{0,1\}^* \to \{0,1\}^\ell$ at the start of the experiment, and all parties including the adversary access the hash function H only through oracle queries to $\mathcal{O}$. For any fresh input x not previously queried, $\mathcal{O}(x)$ is uniformly random in $\{0,1\}^\ell$ and independent of all previous outputs.

We model SHA-256 and SHA-512 as random oracles when used on fixed-length inputs, as is the case for the root-step computations in Sections 3.10 and 3.11. We prove our theorems in the ROM. The ROM is not used for HMAC-SHA512, which suffers from length-extension attacks under SHA's Merkle-Damgård structure and is instead treated as a concrete keyed primitive assumed to be a secure PRF (Definition 3.6).

3.4 Hash Functions

Definition 3.3 (Hash function)**.** A hash function is a pair of PPT algorithms (Gen, H) where $\mathsf{Gen}(1^\lambda)$ outputs a key s, and H takes as input a key s and a string $x \in \{0,1\}^*$ and outputs a string $H_s(x) \in \{0,1\}^{\ell(\lambda)}$.

Definition 3.4 (Collision-resistant hash function (CRHF))**.** A hash function (Gen, H) is collision resistant if for all PPT adversaries $\mathcal{A}$:

$$\Pr\big[s \leftarrow \mathsf{Gen}(1^\lambda);\ \mathcal{A}(s) = (x, x') \ \wedge\ x \neq x' \ \wedge\ H_s(x) = H_s(x')\big] \leq \mathsf{negl}(\lambda).$$

Definition 3.5 (Preimage resistance)**.** A hash function (Gen, H) is preimage resistant if for all PPT adversaries $\mathcal{A}$:

$$\Pr\big[s \leftarrow \mathsf{Gen}(1^\lambda);\ x \xleftarrow{\$} \{0,1\}^\lambda;\ \mathcal{A}(s, H_s(x)) = x' \ \wedge\ H_s(x') = H_s(x)\big] \leq \mathsf{negl}(\lambda).$$

We say a hash function (Gen, H) is quantum preimage resistant (QPre) if no quantum adversary running in polynomial time can find a preimage with non-negligible probability. Grover's algorithm [Gro96] achieves a quadratic speedup over classical preimage search: finding a preimage of an n-bit hash function requires $O(2^{n/2})$ evaluations compared to $O(2^n)$ classically. As a result, an n-bit hash provides only $n/2$ bits of preimage security. Concretely, SHA-512 truncated to 256 bits and SHA-256 both offer 2^{128} bits of preimage security under this attack.

3.5 Pseudorandom Functions and Keyed Hash Functions

Definition 3.6 (Pseudorandom function (PRF))**.** Let $F : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^*$ be an efficient keyed function. F is a pseudorandom function if for all PPT distinguishers $\mathcal{D}$:

$$\Big|\Pr[\mathcal{D}^{F_k(\cdot)}(1^\lambda) = 1 \mid k \xleftarrow{\$} \{0,1\}^\lambda] - \Pr[\mathcal{D}^{f(\cdot)}(1^\lambda) = 1 \mid f \xleftarrow{\$} \mathcal{Y}^{\mathcal{X}}]\Big| \leq \mathsf{negl}(\lambda),$$

where the first probability is over uniform choice of k and the randomness of $\mathcal{D}$, and the second is over uniform choice of f from all functions with the same domain and range, and the randomness of $\mathcal{D}$.

HMAC instantiated with a conjectured post-quantum secure compression function (such as SHA) is conjectured to be a secure PRF against quantum adversaries running in polynomial time. We use $\mathsf{HMAC\text{-}SHA512} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^{512}$, defined as $\mathsf{HMAC\text{-}SHA512}(k, m) = \mathsf{SHA512}\big((k \oplus \mathsf{opad}) \,\|\, \mathsf{SHA512}((k \oplus \mathsf{ipad}) \| m)\big)$, where opad and ipad are the standard HMAC padding constants. We rely on the assumption that HMAC-SHA-512 is a secure PRF against quantum adversaries running in polynomial time wherever the key to a derivation step is genuinely secret,

namely the hardened derivation steps of Sections 3.9, 3.10, and 3.11; for non-hardened steps the key is publicly computable and this assumption is not invoked, as discussed in the remark following Theorem 4.2.

Definition 3.7 (Keyed extensible-output function (keyed XOF))**.** Let $\ell : \mathbb{N} \to \mathbb{N}$ be a polynomially bounded output-length function. A keyed function $H : \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^{\ell(\lambda)}$ is a keyed extensible-output function if the following hold for all $k \xleftarrow{\$} \{0,1\}^\lambda$:

- (PRF) For any output length $L \leq \ell(\lambda)$, the truncation $H(k,\cdot)[0:L]$ is a secure PRF against QPT adversaries (Definition 3.6).
- (Collision resistance) $H(k,\cdot)$ is collision resistant against QPT adversaries (Definition 3.4).
- (Preimage resistance) $H(k,\cdot)$ is preimage resistant against QPT adversaries (Definition 3.5).
- (Output independence) For any fixed k and d, the sub-strings $H(k,d)[0:n]$, $H(k,d)[n:2n]$, $H(k,d)[2n:3n]$ are computationally independent in the sense that knowledge of any two provides no non-negligible advantage in predicting the third.

We define $\mathsf{HASH768} : \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^{768}$ as a keyed XOF with fixed 768-bit output, parsed as three consecutive 256-bit blocks:

$$\mathsf{HASH768}(k,d) \;=\; (T_S \| T_W \| T_C), \qquad T_S, T_W, T_C \in \{0,1\}^{256}.$$

Throughout this paper we instantiate $\mathsf{HASH768}$ using KMAC256, the keyed variant of SHA-3 defined in NIST SP 800-185 [Nat16]. Concretely:

$$\mathsf{HASH768}(k,d) \;\coloneqq\; \mathsf{KMAC256}(K{=}k,\ X{=}d,\ L{=}768,\ S{=}\text{“”}),$$

where L is the output length in bits, and S is a domain-separation string, that in our case is empty. KMAC256 is based on cSHAKE256 and inherits its security against QPT adversaries under the assumption that the Keccak permutation is quantum preimage resistant. The output independence property of Definition 3.7 holds for KMAC256 by the XOF security of cSHAKE256 and the fact that the three blocks occupy non-overlapping output positions. All benchmarks and implementation results in Section 9 use this instantiation.

Remark 3.8 (Alternative instantiations of $\mathsf{HASH768}$). Any keyed XOF satisfying Definition 3.7 may be used to instantiate $\mathsf{HASH768}$. Three families are of practical interest, each with different trade-offs. KMAC256 (primary, used in this paper) is standardised by NIST SP 800-185 [Nat16], based on the Keccak permutation, FIPS-compliant. Security relies solely on the preimage and collision resistance of Keccak. BLAKE3 [OANWO20] is a high-throughput XOF, not yet standardised by NIST. Security assumptions are comparable to SHA-3. Poseidon2 [GKR+21] over BN254 is an arithmetically efficient sponge designed for ZK circuits. The security arguments in this paper are stated against the abstract $\mathsf{HASH768}$ interface and apply to all three instantiations, subject to their respective security assumptions.

3.6 Simulation Extractable NIZKs

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

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$: generates public parameters pp.
- $\mathsf{NIZKProve}(\mathsf{pp}, x, w) \to \pi$: given $(x,w) \in \mathcal{R}$, produces a proof π.
- $\mathsf{NIZKVerify}(\mathsf{pp}, x, \pi) \to \{0,1\}$: outputs 1 iff π is valid for x.

We require Π to satisfy the following properties.

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

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

Knowledge soundness. For every PPT prover $\mathcal{P}^*$ and every $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, if $\mathcal{P}^*(\mathsf{pp})$ outputs (x, π) with $\mathsf{NIZKVerify}(\mathsf{pp}, x, \pi) = 1$, then there exists a PPT 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 PPT adversary $\mathcal{A}$ given access to a simulation oracle $\mathsf{Sim}(\mathsf{pp}, \cdot)$, if $\mathcal{A}$ outputs an accepting proof π^* for a statement x^* not previously queried to Sim, then there exists a PPT 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. For all (possibly unbounded) verifiers $\mathcal{V}^*$, there exists a PPT simulator Sim such that for all $(x, w) \in \mathcal{R}$:

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

We instantiate Π with a ZK-STARK [BBHR18], a transparent (no trusted setup) argument system with succinct proofs, conjectured to be sound against QPT adversaries. In all security arguments below, the ZK-STARK may be replaced by any NIZK conjectured to satisfy the properties above and is sound against quantum adversaries without affecting the conclusions.

3.7 Digital Signatures

A digital signature scheme is a triple $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$:

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

Definition 3.9 (Correctness of a signature scheme)**.** A signature scheme $\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ is correct if for all $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}(1^\lambda)$ and all $M \in \mathcal{M}$:

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

Definition 3.10 (EUF-CMA)**.** A signature scheme Σ is existentially unforgeable under chosen-message attack (EUF-CMA) if for all PPT adversaries $\mathcal{A}$:

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

3.8 Hierarchical Deterministic Wallets

We follow the framework of Di Luzio et al. [LFA20] and extend it in three directions: we restrict the access hierarchy from a DAG to a tree, which matches the derivation structure of all HD wallet standards we use in practice; we extend the derivation algorithms to capture the distinction between hardened and non-hardened child derivation; and we introduce a derivation certificate that allows any party to verify that a public key was honestly derived according to the tree structure, without requiring knowledge of any private key material.

Let $T = (V, E)$ be a rooted tree representing the key hierarchy, with root v_0. Each node $v_i \in V$ is labelled as either *hardened* or *non-hardened*, with the root v_0 treated as a special case. The set of descendants of v_i is $\mathsf{Desc}(v_i) = \{v_j \mid v_i \prec v_j\}$, that is, all v_j reachable from v_i in T.

We distinguish between three kinds of per-node material. Global public parameters pp, fixed once at key generation, contain scheme-specific data such as the root public key and curve parameters. Each node v_i holds a *node state* st_i, which carries the evolving per-node public data produced during derivation, can include the public key pk_i, and is passed as input to any subsequent derivation from v_i. Each node v_i additionally holds two private keys: a *derivation key* $d_i \in \mathcal{D}$, which is the private scalar used to compute child keys, and a *witness key* $w_i \in \mathcal{W}$, which is a private value bound to the seed by a one-way function and used in signing, in computing child witness keys, and in hardened public key derivation. The witness key must be propagated through every derivation step, both hardened and non-hardened, so that it is always available for signing and certificate generation. For signing purposes, we bundle d_i and w_i into a single signing key $\mathsf{sk}_i \coloneqq (d_i, w_i)$.

The scheme is parameterised by a pair (F, F_{ver}) where F is a certificate generation function and F_{ver} is its verification counterpart. We require the pair to satisfy the following completeness condition: for any honestly generated certificate $\tau_j = \mathsf{DCert}(\mathsf{pp}, d_h, w_h, \mathsf{st}_h, v_h, v_j, a, F)$, we have $F_{\mathsf{ver}}(\mathsf{pp}, \mathsf{pk}_j, \tau_j) = 1$.

An HDW $\Pi[F, F_{\mathsf{ver}}] = (\mathsf{Set}, \mathsf{DPub}, \mathsf{DPub}^*, \mathsf{DPriv}, \mathsf{DPriv}^*, \mathsf{DCert}, \mathsf{DVerify}, \mathsf{Sign}, \mathsf{SignVerify}, \mathsf{Verify})$ over seed space $\mathcal{S}$, derivation key space $\mathcal{D}$, witness key space $\mathcal{W}$, and message space $\mathcal{M}$ consists of:

- $\mathsf{Set}(1^\lambda, T, S)$: takes the key tree and seed $S \in \mathcal{S}$; outputs global public parameters pp, the root node state st_0, and for every $v_i \in V$ the derivation key $d_i \in \mathcal{D}$, the witness key $w_i \in \mathcal{W}$, and the signing key $\mathsf{sk}_i \coloneqq (d_i, w_i)$.
- $\mathsf{DPub}(\mathsf{pp}, \mathsf{st}_i, v_i, v_j)$: given the node state st_i of the deepest hardened ancestor v_i of v_j, outputs the public key pk_j and node state st_j by iteratively applying non-hardened derivation steps from v_i to v_j, using only public inputs. Returns $\perp$ if any node on the path from v_i to v_j is hardened.
- $\mathsf{DPub}^*(\mathsf{pp}, d_i, w_i, \mathsf{st}_i, v_i, v_j)$: given the derivation key d_i, witness key w_i, and node state st_i of parent v_i, outputs the public key pk_j and node state st_j of its hardened child v_j.
- $\mathsf{DPriv}(\mathsf{pp}, d_i, w_i, \mathsf{st}_i, v_i, v_j)$: given the derivation key d_i, witness key w_i, and node state st_i of parent v_i, outputs the derivation key d_j, witness key w_j, and node state st_j of its non-hardened child v_j. Returns $\perp$ if v_j is hardened.
- $\mathsf{DPriv}^*(\mathsf{pp}, d_i, w_i, \mathsf{st}_i, v_i, v_j)$: given the derivation key d_i, witness key w_i, and node state st_i of parent v_i, outputs the derivation key d_j, witness key w_j, and node state st_j of its hardened child v_j. Returns $\perp$ if v_j is non-hardened.
- $\mathsf{DCert}(\mathsf{pp}, d_h, w_h, \mathsf{st}_h, v_h, v_j, a, F)$: given the derivation key d_h, witness key w_h, and node state st_h of the deepest hardened ancestor v_h of v_j (or the root state st_0 if no hardened ancestor exists), an anchor witness a carrying any additional private material required by F to build the proof but not stored in the node key pair, and the certificate generation function F, outputs a derivation certificate τ_j attesting that pk_j was honestly derived from v_h according to the prescribed path in T.

- $\mathsf{DVerify}(\mathsf{pp}, \mathsf{pk}_j, \tau_j, F_{\mathsf{ver}})$: given the public key pk_j, a derivation certificate τ_j, and the certificate verification function F_{ver}, outputs 1 if $F_{\mathsf{ver}}(\mathsf{pp}, \mathsf{pk}_j, \tau_j) = 1$ and 0 otherwise.
- $\mathsf{Sign}(\mathsf{sk}_i, M)$: parses $\mathsf{sk}_i = (d_i, w_i)$ and outputs a signature σ.
- $\mathsf{SignVerify}(\mathsf{pp}, \mathsf{pk}_j, M, \sigma)$: given the public key pk_j, a message M, and a signature σ, outputs 1 iff the signature is valid for pk_j and M, and 0 otherwise. It does not check any derivation certificate.
- $\mathsf{Verify}(\mathsf{pp}, \mathsf{pk}_j, \tau_j, M, \sigma, F_{\mathsf{ver}})$: in the classical case (no certificate), outputs $\mathsf{SignVerify}(\mathsf{pp}, \mathsf{pk}_j, M, \sigma)$. In ZKPoSP mode, outputs 1 iff $\mathsf{DVerify}(\mathsf{pp}, \mathsf{pk}_j, \tau_j, F_{\mathsf{ver}}) = 1$ and $\mathsf{SignVerify}(\mathsf{pp}, \mathsf{pk}_j, M, \sigma) = 1$, and 0 otherwise.

Multi-level derivation is obtained by applying the appropriate single-step algorithm iteratively along the path from parent to descendant, passing the node state and keys produced at each step as input to the next.

Definition 3.11 (Correctness of HDW)**.** $\Pi[F, F_{\mathsf{ver}}]$ is correct if for every tree $T = (V, E)$, all $v_j \in V$, all $S \in \mathcal{S}$, and all $M \in \mathcal{M}$:

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

where $(\mathsf{pp}, \mathsf{st}_0, \{\mathsf{sk}_i\}) = \mathsf{Set}(1^\lambda, T, S)$, pk_j is included in st_j, and $\tau_j = \mathsf{DCert}(\mathsf{pp}, d_h, w_h, \mathsf{st}_h, v_h, v_j, a, F)$ where v_h is the deepest hardened ancestor of v_j and a is the anchor witness supplied by the caller.

The security game allows an adversary to corrupt nodes (obtaining their signing keys and node states, implicitly corrupting all descendants) and query a signing oracle at any uncorrupted node. The signing oracle returns both the signature and a derivation certificate, so the adversary can verify the consistency of the derived public key without learning any private key material beyond what is explicitly revealed.

Definition 3.12 (Hierarchical EUF-CMA of HDW)**.** $\Pi[F, F_{\mathsf{ver}}]$ is hierarchically EUF-CMA secure if for every tree $T = (V, E)$ and all PPT adversaries $\mathcal{A}$:

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

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

Setup. The challenger samples $S \xleftarrow{\$} \mathcal{S}$ and runs $(\mathsf{pp}, \mathsf{st}_0, \{\mathsf{sk}_i\}_{v_i \in V}) = \mathsf{Set}(1^\lambda, T, S)$. It gives pp to $\mathcal{A}$, together with st_j for every node v_j reachable from v_0 via a purely non-hardened path, computed by applying DPub iteratively from v_0. Node states of nodes with a hardened ancestor are not given to $\mathcal{A}$ at setup.

Query. $\mathcal{A}$ has access to:

- $\mathcal{O}_{\mathrm{Corr}}(v_i)$: returns $(\mathsf{sk}_i, \mathsf{st}_i)$, i.e. $(d_i, w_i, \mathsf{st}_i)$; adds v_i and all its descendants to Q_{Corr}. With $(d_i, w_i, \mathsf{st}_i)$, $\mathcal{A}$ may now compute the node states of all descendants of v_i by iterative application of DPub and DPub^*, their signing keys by iterative application of DPriv and DPriv^*, and their derivation certificates via $\mathsf{DCert}(\cdot, F)$.
- $\mathcal{O}_{\mathrm{Sign}}(M, v_i)$: computes $\tau_i = \mathsf{DCert}(\mathsf{pp}, d_h, w_h, \mathsf{st}_h, v_h, v_i, a_h, F)$ where v_h is the deepest hardened ancestor of v_i and a_h is the corresponding anchor witness, and returns $(\mathsf{pk}_{v_i}, \tau_i, \sigma)$ where $\sigma \xleftarrow{\$} \mathsf{Sign}(\mathsf{sk}_i, M)$; adds (M, v_i) to Q_{Sign}. $\mathcal{A}$ can verify the consistency of pk_{v_i} by checking $\mathsf{DVerify}(\mathsf{pp}, \mathsf{pk}_{v_i}, \tau_i, F_{\mathsf{ver}}) = 1$.

Forgery. $\mathcal{A}$ outputs $(v^*, M^*, \sigma^*, \tau^*)$. The challenger computes pk_{v^*} internally. The experiment returns 1 iff $\mathsf{Verify}(\mathsf{pp}, \mathsf{pk}_{v^*}, \tau^*, M^*, \sigma^*, F_{\mathsf{ver}}) = 1$, $v^* \notin Q_{\mathrm{Corr}}$, and $(M^*, v^*) \notin Q_{\mathrm{Sign}}$.

Remark 3.13 (Relationship to Di Luzio et al.). The original framework of Di Luzio et al. [LFA20] is recovered as the special case in which all nodes are non-hardened, DPub^*, DPriv^*, DCert, and $\mathsf{DVerify}$ are never invoked, the witness key w_i is absent, the node state is absorbed into pp, and all public keys are given to $\mathcal{A}$ at setup. In that special case Verify reduces to classical signature verification without a derivation certificate.

3.9 SLIP-10/Ed25519 Key Derivation

We recall the SLIP-10 [HR16] key derivation scheme instantiated over Ed25519 and instantiate it in terms of the HDW framework of Section 3.8. SLIP-10/Ed25519 is hardened-only: non-hardened child derivation is not defined for Ed25519 under this standard, and the chain code is never exposed since there is no public child key derivation. Let $n \geq 0$ be the chain depth and let ℓ be the Ed25519 group order.

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 private scalar $k_{L_i} \in \{0,1\}^{256}$, which is the derivation key $d_i \coloneqq k_{L_i}$ of the HDW.
  - The chain code $c_i \in \{0,1\}^{256}$, which is the witness key $w_i \coloneqq c_i$ of the HDW. The chain code is private in SLIP-10/Ed25519 since it is never needed for public key derivation and acts as the HMAC key binding the derivation chain.
- Public:
  - The Ed25519 public key $K_i \coloneqq (k_{L_i} \bmod \ell) \cdot B \in \mathbb{G}$ at each depth.
  - The child index $i_n \in \{2^{31}, \ldots, 2^{32}-1\}$ at each node; all indices are hardened.

The global public parameters are $\mathsf{pp} \coloneqq K_0$ and the root signing key is $\mathsf{sk}_0 \coloneqq (k_{L_0}, c_0)$. The node state at depth i is $\mathsf{st}_i \coloneqq (K_i, i_i)$, which contains the public key and the child index used to derive v_i from its parent. The seed x_0 and all private scalars and chain codes are private.

Root ($n = 0$). Given a seed $x_0 \in \{0,1\}^{256}$ (private), compute

$$k_{L_0} \| c_0 \;\coloneqq\; \mathsf{HMAC\text{-}SHA512}\big(\text{"}\mathtt{ed25519\,seed}\text{"},\; x_0\big) \;\in\; \{0,1\}^{512},$$

parsing the 512-bit output as k_{L_0} (left 32 bytes) and c_0 (right 32 bytes). The root public key is $K_0 \coloneqq (k_{L_0} \bmod \ell) \cdot B$ (public). The root signing key is $\mathsf{sk}_0 \coloneqq (k_{L_0}, c_0)$, and the root node state is $\mathsf{st}_0 \coloneqq (K_0, \bot)$, where $\bot$ indicates that the root has no parent index.

Hardened child step ($i_n \geq 2^{31}$). Given the parent derivation key $d_{n-1} = k_{L_{n-1}}$, parent witness key $w_{n-1} = c_{n-1}$, and hardened index i_n, compute

$$k_{L_n} \| c_n \;\coloneqq\; \mathsf{HMAC\text{-}SHA512}\big(c_{n-1},\; \mathtt{0x00} \| k_{L_{n-1}} \| \langle i_n \rangle_4\big) \;\in\; \{0,1\}^{512},$$

where $\langle i_n \rangle_4$ is the 4-byte big-endian encoding of i_n, parsing the output as k_{L_n} (left 32 bytes) and c_n (right 32 bytes). The child signing scalar is $k_n \coloneqq k_{L_n} \bmod \ell$, the child public key is $K_n \coloneqq k_n \cdot B$ (public), and the child node state is $\mathsf{st}_n \coloneqq (K_n, i_n)$. Because both $k_{L_{n-1}}$ and c_{n-1} are private, the child key is not publicly derivable. The child derivation key is $d_n \coloneqq k_{L_n}$ and the child witness key is $w_n \coloneqq c_n$, giving the child signing key $\mathsf{sk}_n \coloneqq (k_{L_n}, c_n)$.

Instantiation of the HDW algorithms.

- $\mathsf{DPriv}^*(\mathsf{pp}, d_{n-1}, w_{n-1}, \mathsf{st}_{n-1}, v_{n-1}, v_n)$: parse $d_{n-1} = k_{L_{n-1}}$, $w_{n-1} = c_{n-1}$, and $\mathsf{st}_{n-1} = (K_{n-1}, i_{n-1})$; retrieve index i_n from st_n; apply the hardened child step to obtain k_{L_n} and c_n; return $d_n \coloneqq k_{L_n}$, $w_n \coloneqq c_n$, and $\mathsf{st}_n \coloneqq (K_n, i_n)$.
- $\mathsf{DPub}^*(\mathsf{pp}, d_{n-1}, w_{n-1}, \mathsf{st}_{n-1}, v_{n-1}, v_n)$: parse $d_{n-1} = k_{L_{n-1}}$, $w_{n-1} = c_{n-1}$, and $\mathsf{st}_{n-1} = (K_{n-1}, i_{n-1})$; retrieve index i_n from st_n; apply the hardened child step to obtain k_{L_n} and c_n; return K_n and $\mathsf{st}_n \coloneqq (K_n, i_n)$. Note that DPub^* is an internal operation of the key owner, who provides (d_{n-1}, w_{n-1}) from their own private storage to compute the child public key without generating or exposing the full child private material.
- $\mathsf{DPub}(\mathsf{pp}, \mathsf{st}_i, v_i, v_j)$: returns $\perp$, since SLIP-10/Ed25519 does not support non-hardened derivation.
- $\mathsf{DPriv}(\mathsf{pp}, d_i, w_i, \mathsf{st}_i, v_i, v_j)$: returns $\perp$, since SLIP-10/Ed25519 does not support non-hardened derivation.
- $\mathsf{Sign}(\mathsf{sk}_n, M)$: parse $\mathsf{sk}_n = (k_{L_n}, c_n)$; produce a classical Ed25519 signature under $k_{L_n} \bmod \ell$. In ZKPoSP mode, Sign generates the signing proof π_{sign} as described in Section 5.
- $\mathsf{DCert}(\mathsf{pp}, d_h, w_h, \mathsf{st}_h, v_h, v_n, a, F)$: in classical mode, $F = \varepsilon$ and the certificate is empty; in ZKPoSP mode, $a \coloneqq x_0$ is the root seed, apply DPriv^* iteratively from v_h to v_n to obtain the witness material, then invoke F with a to produce the derivation certificate τ_n.
- $\mathsf{DVerify}(\mathsf{pp}, K_n, \tau_n, F_{\mathsf{ver}})$: in classical mode, F_{ver} always returns 1; in ZKPoSP mode, invoke $F_{\mathsf{ver}}(\mathsf{pp}, K_n, \tau_n)$ and return its output.
- $\mathsf{SignVerify}(\mathsf{pp}, K_n, M, \sigma)$: return 1 iff σ is a valid Ed25519 signature for M under K_n.
- $\mathsf{Verify}(\mathsf{pp}, K_n, \tau_n, M, \sigma, F_{\mathsf{ver}})$: in classical mode, return $\mathsf{SignVerify}(\mathsf{pp}, K_n, M, \sigma)$. In ZKPoSP mode, return 1 iff $\mathsf{DVerify}(\mathsf{pp}, K_n, \tau_n, F_{\mathsf{ver}}) = 1$ and $\mathsf{SignVerify}(\mathsf{pp}, K_n, M, \sigma) = 1$.

Quantum vulnerability and the role of the derivation certificate. The signing scalar $k_n = k_{L_n} \bmod \ell$ satisfies $K_n = k_n \cdot B$; a QPT adversary recovers k_n from K_n via Shor's algorithm. This breaks the classical security argument: in standard Ed25519, a valid signature implicitly proves knowledge of the secret scalar used to compute the public key. Once Shor's algorithm is available, any party can recover the secret scalar from the public key, so a valid signature no longer certifies that the signer is the legitimate owner of the key.

To restore unforgeability in the post-quantum setting, the signer must prove knowledge of the seed from which the entire derivation chain originates, since the seed is bound to the public key only through a chain of one-way hash functions and is not recoverable from any public value even by a quantum adversary. This is the role of the derivation certificate τ_n in ZKPoSP mode: rather than implicitly certifying key ownership through a classical signature, DCert produces a NIZK proof that K_n was honestly derived from a secret seed following the prescribed derivation path, establishing a sound link between the signer and the public key also in the presence of quantum machines.

The chain code $w_n = c_n$ is the right half of the HMAC output at depth n, whose computation requires both $k_{L_{n-1}}$ as the HMAC data input and c_{n-1} as the HMAC key. Even if a quantum adversary recovers $k_{L_{n-1}}$ via Shor from K_{n-1}, recovering c_n still requires knowledge of c_{n-1}, which is private and not derivable from any public value without inverting the HMAC chain back to the seed, which happens with negligible probability even against a quantum adversary by the preimage resistance of SHA-512 and Grover's algorithm [Gro96]. Therefore $w_n = c_n$ is a quantum-safe witness for every hardened key: it is private, bound to the seed, and not recoverable by any QPT adversary. This is the structural property that enables sound proofs for SLIP-10/Ed25519 hardened keys using only the data from the parent node, as analysed in Section 6.2.

3.10 BIP32 Key Derivation (secp256k1)

We recall the BIP32 key derivation scheme [Wui12] and instantiate it in terms of the HDW framework of Section 3.8. Let $n \geq 0$ be the chain depth. Let $\mathbb{G}_{\text{secp}}$ be the prime-order group of the secp256k1 elliptic curve, with generator G_{secp} and prime order $q \approx 2^{256}$. Scalars are elements of $\mathbb{Z}_q$; we write $[k]G_{\text{secp}}$ for scalar multiplication.

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

- Private:
  - The seed $x_0 \in \{0,1\}^{256}$, from which the entire key hierarchy is deterministically derived. It is never exposed and serves as the witness key $w_i := x_0$ at every node, as explained below.
  - The private key $k_i \in \mathbb{Z}_q$ at each depth, with $0 < k_i < q$. This is the derivation key $d_i := k_i$.
  - The chain code $c_i \in \{0,1\}^{256}$ at each depth. For hardened nodes, c_i is private but recoverable by a quantum adversary once k_{i-1} is known via Shor. For non-hardened nodes, c_i is public. In neither case does c_i serve as a quantum-safe witness independently of the seed.
- Public:
  - The compressed secp256k1 public key $K_i := [k_i]G_{\text{secp}}$ at each depth.
  - The child index $i_n \in \{0, \ldots, 2^{32}-1\}$ at each node. Indices $i_n \geq 2^{31}$ are hardened; indices $i_n < 2^{31}$ are non-hardened.

The global public parameters are $\mathsf{pp} := K_0$, and the root signing key is $\mathsf{sk}_0 := (k_0, x_0)$. The node state at depth i is $\mathsf{st}_i := (K_i, c_i, i_i)$, which contains the public key, the chain code, and the child index used to derive v_i from its parent. The seed x_0 and all private keys $(k_1, \ldots, k_n)$ are private.

Root ($n = 0$). Given a seed x_0, compute

$$I \;:=\; \mathsf{HMAC\text{-}SHA512}\big(\text{"}\texttt{Bitcoin seed}\text{"},\; x_0\big) \;\in\; \{0,1\}^{512}.$$

Parse $I = I_L \| I_R$ with $I_L, I_R \in \{0,1\}^{256}$. Set $k_0 := I_L \bmod q$ and $c_0 := I_R$. If $k_0 = 0$ or $k_0 \geq q$, discard and resample. Set $K_0 := [k_0]G_{\text{secp}}$. The root signing key is $\mathsf{sk}_0 := (k_0, x_0)$, and the root node state is $\mathsf{st}_0 := (K_0, c_0, \perp)$.

Non-hardened child step ($i_n < 2^{31}$). Given the parent derivation key $d_{n-1} = k_{n-1}$, parent witness key $w_{n-1} = x_0$, parent node state $\mathsf{st}_{n-1} = (K_{n-1}, c_{n-1}, i_{n-1})$, and child index i_n, compute

$$I \;:=\; \mathsf{HMAC\text{-}SHA512}\big(c_{n-1},\; \mathtt{ser}_P(K_{n-1}) \| \langle i_n \rangle_4\big) \;\in\; \{0,1\}^{512},$$

where $\mathtt{ser}_P(K_{n-1})$ is the 33-byte compressed encoding of K_{n-1} and $\langle i_n \rangle_4$ is the 4-byte big-endian encoding of i_n. Parse $I = I_L \| I_R$. If $I_L \geq q$ or $k_{n-1} + I_L \bmod q = 0$, the child key is invalid; skip this index. Otherwise:

$$k_n \;:=\; (k_{n-1} + I_L) \bmod q, \qquad c_n \;:=\; I_R, \qquad K_n \;:=\; [k_n]G_{\text{secp}} \;=\; K_{n-1} + [I_L]G_{\text{secp}}.$$

The child derivation key is $d_n := k_n$, the witness key is $w_n := x_0$, giving the child signing key $\mathsf{sk}_n := (k_n, x_0)$, and the child node state is $\mathsf{st}_n := (K_n, c_n, i_n)$.

Hardened child step ($i_n \geq 2^{31}$). Given the parent derivation key $d_{n-1} = k_{n-1}$, parent witness key $w_{n-1} = x_0$, parent node state $\mathsf{st}_{n-1} = (K_{n-1}, c_{n-1}, i_{n-1})$, and child index i_n, compute

$$I \;:=\; \mathsf{HMAC\text{-}SHA512}\big(c_{n-1},\; \mathtt{0x00} \| \mathtt{ser}_{32}(k_{n-1}) \| \langle i_n \rangle_4\big) \;\in\; \{0,1\}^{512},$$

where $\mathtt{ser}_{32}(k_{n-1})$ is the 32-byte big-endian encoding of k_{n-1}. Parse and update identically to the non-hardened case. Because k_{n-1} is a private HMAC input, the child key is not publicly derivable without the parent private key. The child derivation key is $d_n \coloneqq k_n$, the witness key is $w_n \coloneqq x_0$, giving the child signing key $\mathsf{sk}_n \coloneqq (k_n, x_0)$, and the child node state is $\mathsf{st}_n \coloneqq (K_n, c_n, i_n)$.

Instantiation of the HDW algorithms.

- $\mathsf{DPriv}(\mathsf{pp}, d_{n-1}, w_{n-1}, \mathsf{st}_{n-1}, v_{n-1}, v_n)$: parse $d_{n-1} = k_{n-1}$, $w_{n-1} = x_0$, and $\mathsf{st}_{n-1} = (K_{n-1}, c_{n-1}, i_{n-1})$; retrieve index i_n from st_n; apply the non-hardened child step; return $d_n \coloneqq k_n$, $w_n \coloneqq x_0$, and $\mathsf{st}_n \coloneqq (K_n, c_n, i_n)$.
- $\mathsf{DPriv}^*(\mathsf{pp}, d_{n-1}, w_{n-1}, \mathsf{st}_{n-1}, v_{n-1}, v_n)$: parse $d_{n-1} = k_{n-1}$, $w_{n-1} = x_0$, and $\mathsf{st}_{n-1} = (K_{n-1}, c_{n-1}, i_{n-1})$; retrieve index i_n from st_n; apply the hardened child step; return $d_n \coloneqq k_n$, $w_n \coloneqq x_0$, and $\mathsf{st}_n \coloneqq (K_n, c_n, i_n)$.
- $\mathsf{DPub}(\mathsf{pp}, \mathsf{st}_i, v_i, v_j)$: parse $\mathsf{st}_i = (K_i, c_i, i_i)$; apply the non-hardened child step iteratively from v_i to v_j; return K_j and $\mathsf{st}_j \coloneqq (K_j, c_j, i_j)$.
- $\mathsf{DPub}^*(\mathsf{pp}, d_{n-1}, w_{n-1}, \mathsf{st}_{n-1}, v_{n-1}, v_n)$: parse $d_{n-1} = k_{n-1}$, $w_{n-1} = x_0$, and $\mathsf{st}_{n-1} = (K_{n-1}, c_{n-1}, i_{n-1})$; apply the hardened child step; return K_n and $\mathsf{st}_n \coloneqq (K_n, c_n, i_n)$.
- $\mathsf{Sign}(\mathsf{sk}_n, M)$: parse $\mathsf{sk}_n = (k_n, w_n)$; produce a classical ECDSA or Schnorr signature under k_n. In ZKPoSP mode, Sign generates the signing proof π_{sign} as described in Section 5.
- $\mathsf{DCert}(\mathsf{pp}, d_h, w_h, \mathsf{st}_h, v_h, v_n, a, F)$: in classical mode, $F = \varepsilon$ and the certificate is empty; in ZKPoSP mode, $a \coloneqq x_0$ is the root seed, and F is invoked with a and the full derivation chain from the root to v_n as witness material to produce the derivation certificate τ_n. Since there is no intermediate quantum-safe witness other than the seed itself, the derivation proof must always cover the full chain from x_0 to v_n and cannot be shortened to a last step, as analysed in Section 6.1.
- $\mathsf{DVerify}(\mathsf{pp}, K_n, \tau_n, F_{\mathsf{ver}})$: in classical mode, F_{ver} always returns 1; in ZKPoSP mode, invoke $F_{\mathsf{ver}}(\mathsf{pp}, K_n, \tau_n)$ and return its output.
- $\mathsf{SignVerify}(\mathsf{pp}, K_n, M, \sigma)$: return 1 iff σ is a valid ECDSA or Schnorr signature for M under K_n.
- $\mathsf{Verify}(\mathsf{pp}, K_n, \tau_n, M, \sigma, F_{\mathsf{ver}})$: in classical mode, return $\mathsf{SignVerify}(\mathsf{pp}, K_n, M, \sigma)$. In ZKPoSP mode, return 1 iff $\mathsf{DVerify}(\mathsf{pp}, K_n, \tau_n, F_{\mathsf{ver}}) = 1$ and $\mathsf{SignVerify}(\mathsf{pp}, K_n, M, \sigma) = 1$.

Quantum vulnerability. Every private key $k_i \in \mathbb{Z}_q$ satisfies $K_i = [k_i]G_{\mathrm{secp}}$. A QPT adversary applies Shor's algorithm to recover k_i from K_i directly. For non-hardened nodes, the chain code c_i is public and all HMAC inputs are known, so the entire derivation is publicly reconstructible without any quantum capability. Moreover, k_{i-1} appears as a private HMAC input, but since a quantum adversary can recover k_{i-1} from K_{i-1} via Shor, all HMAC inputs become known and the entire derivation chain is quantum-reconstructible for all non-hardened nodes. The only value that a quantum adversary cannot recover in any case is the seed x_0 itself, since the connection between x_0 and K_0 passes through HMAC-SHA512 and is protected by its preimage resistance, which provides 2^{128} quantum security by Grover's algorithm [Gro96]. The seed x_0 therefore serves as the unique quantum-safe witness $w_i \coloneqq x_0$ at every node, and the derivation proof in ZKPoSP mode starts from the seed.

3.11 BIP32-Ed25519 Key Derivation

We recall the BIP32-Ed25519 key derivation scheme [KL17] and instantiate it in terms of the HDW framework of Section 3.8. 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 signing scalar $k_{L_i} \in \{0,1\}^{256}$, which is the derivation key $d_i \coloneqq k_{L_i}$ of the HDW.
  - The nonce seed $k_{R_i} \in \{0,1\}^{256}$, which is the witness key $w_i \coloneqq k_{R_i}$ of the HDW. For hardened nodes, k_{R_i} is a genuine quantum-safe witness since it depends on private HMAC inputs not recoverable by a quantum adversary. For non-hardened nodes, k_{R_i} is freely computable from public data and does not serve as a quantum-safe witness, as analysed below.
- 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 $c_0 \coloneqq \mathsf{SHA256}(\texttt{0x01} \| x_0)$.
  - The child index $i_n \in \{0, \ldots, 2^{32}-1\}$ at each node. Indices $i_n \geq 2^{31}$ are hardened; indices $i_n < 2^{31}$ are non-hardened.

The global public parameters are $\mathsf{pp} \coloneqq A_0$ and the root signing key is $\mathsf{sk}_0 \coloneqq (k_{L_0}, k_{R_0})$. The node state at depth i is $\mathsf{st}_i \coloneqq (A_i, c_i, i_i)$, which contains the public key, the chain code, and the child index used to derive v_i from its parent. The seed x_0 and all extended keys $(k_{L_1}, k_{R_1}, \ldots, k_{L_n}, k_{R_n})$ are private.

Root ($n = 0$). 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; clear bit 255; set bit 254. Set the root signing key with $\mathsf{sk}_0 \coloneqq (k_{L_0}, k_{R_0})$, and the root public key $A_0 \coloneqq k_{L_0} \cdot B$ (public). We abbreviate these steps as $k_{L_0} \| k_{R_0} \coloneqq \mathsf{Tweak}(\mathsf{SHA512}(x_0))$. Derive the root chain code $c_0 \coloneqq \mathsf{SHA256}(\texttt{0x01} \| x_0) \in \{0,1\}^{256}$ (public). The root node state is $\mathsf{st}_0 \coloneqq (A_0, c_0, \perp)$.

Non-hardened child step ($i_n < 2^{31}$). Given the parent derivation key $d_{n-1} = k_{L_{n-1}}$, parent witness key $w_{n-1} = k_{R_{n-1}}$, parent node state $\mathsf{st}_{n-1} = (A_{n-1}, c_{n-1}, i_{n-1})$, and child index i_n, 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 $\langle i_n \rangle_4$ is the 4-byte little-endian encoding of i_n. Extract from z_n:

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

Update the private keys:

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

Derive the child chain code (public):

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

The child public key is $A_n \coloneqq k_{L_n} \cdot B = A_{n-1} + 8 \cdot z_L^{(n)} \cdot B$ (public). The child derivation key is $d_n \coloneqq k_{L_n}$, the child witness key is $w_n \coloneqq k_{R_n}$, giving the child signing key $\mathsf{sk}_n \coloneqq (k_{L_n}, k_{R_n})$, and the child node state is $\mathsf{st}_n \coloneqq (A_n, c_n, i_n)$. Since all HMAC inputs are public in this step, $z_R^{(n)}$ is freely computable and $w_n = k_{R_n}$ does not serve as a quantum-safe witness.

Hardened child step ($i_n \geq 2^{31}$). Given the parent derivation key $d_{n-1} = k_{L_{n-1}}$, parent witness key $w_{n-1} = k_{R_{n-1}}$, parent node state $\mathsf{st}_{n-1} = (A_{n-1}, c_{n-1}, i_{n-1})$, and child index i_n, compute

$$z_n \coloneqq \mathsf{HMAC\text{-}SHA512}\big(c_{n-1},\ \mathtt{0x00} \| k_{L_{n-1}} \| k_{R_{n-1}} \| \langle i_n \rangle_4\big) \in \{0,1\}^{512}.$$

Extract $z_L^{(n)}$ and $z_R^{(n)}$ and update k_{L_n}, k_{R_n}, c_n, and A_n identically to the non-hardened case. Because $k_{L_{n-1}}$ and $k_{R_{n-1}}$ are private HMAC inputs, A_n cannot be derived without knowledge of the parent private key, and $w_n = k_{R_n}$ is a genuine quantum-safe witness. The child derivation key is $d_n \coloneqq k_{L_n}$, the child witness key is $w_n \coloneqq k_{R_n}$, giving the child signing key $\mathsf{sk}_n \coloneqq (k_{L_n}, k_{R_n})$, and the child node state is $\mathsf{st}_n \coloneqq (A_n, c_n, i_n)$.

Instantiation of the HDW algorithms.

- $\mathsf{DPriv}(\mathsf{pp}, d_{n-1}, w_{n-1}, \mathsf{st}_{n-1}, v_{n-1}, v_n)$: parse $d_{n-1} = k_{L_{n-1}}$, $w_{n-1} = k_{R_{n-1}}$, and $\mathsf{st}_{n-1} = (A_{n-1}, c_{n-1}, i_{n-1})$; retrieve index i_n from st_n; apply the non-hardened child step; return $d_n \coloneqq k_{L_n}$, $w_n \coloneqq k_{R_n}$, and $\mathsf{st}_n \coloneqq (A_n, c_n, i_n)$.
- $\mathsf{DPriv}^*(\mathsf{pp}, d_{n-1}, w_{n-1}, \mathsf{st}_{n-1}, v_{n-1}, v_n)$: parse $d_{n-1} = k_{L_{n-1}}$, $w_{n-1} = k_{R_{n-1}}$, and $\mathsf{st}_{n-1} = (A_{n-1}, c_{n-1}, i_{n-1})$; retrieve index i_n from st_n; apply the hardened child step; return $d_n \coloneqq k_{L_n}$, $w_n \coloneqq k_{R_n}$, and $\mathsf{st}_n \coloneqq (A_n, c_n, i_n)$.
- $\mathsf{DPub}(\mathsf{pp}, \mathsf{st}_i, v_i, v_j)$: parse $\mathsf{st}_i = (A_i, c_i, i_i)$; apply the non-hardened child step iteratively from v_i to v_j using only public data; return A_j and $\mathsf{st}_j \coloneqq (A_j, c_j, i_j)$.
- $\mathsf{DPub}^*(\mathsf{pp}, d_{n-1}, w_{n-1}, \mathsf{st}_{n-1}, v_{n-1}, v_n)$: parse $d_{n-1} = k_{L_{n-1}}$, $w_{n-1} = k_{R_{n-1}}$, and $\mathsf{st}_{n-1} = (A_{n-1}, c_{n-1}, i_{n-1})$; apply the hardened child step; return A_n and $\mathsf{st}_n \coloneqq (A_n, c_n, i_n)$.
- $\mathsf{Sign}(\mathsf{sk}_n, M)$: parse $\mathsf{sk}_n = (k_{L_n}, k_{R_n})$; produce a classical Ed25519 signature under k_{L_n} using k_{R_n} as the nonce seed. In ZKPoSP mode, Sign generates the signing proof π_{sign} as described in Section 5.
- $\mathsf{DCert}(\mathsf{pp}, d_h, w_h, \mathsf{st}_h, v_h, v_n, a, F)$: in classical mode, $F = \varepsilon$ and the certificate is empty; in ZKPoSP mode, $a \coloneqq x_0$ is the root seed for full-chain proofs (or the parent key pair of the hardened anchor for pruned proofs, as described in Section 7.4), apply DPriv^* iteratively from v_h to v_n to obtain the witness material, then invoke F with a to produce the derivation certificate τ_n.
- $\mathsf{DVerify}(\mathsf{pp}, A_n, \tau_n, F_{\mathsf{ver}})$: in classical mode, F_{ver} always returns 1; in ZKPoSP mode, invoke $F_{\mathsf{ver}}(\mathsf{pp}, A_n, \tau_n)$ and return its output.
- $\mathsf{SignVerify}(\mathsf{pp}, A_n, M, \sigma)$: return 1 iff σ is a valid Ed25519 signature for M under A_n.
- $\mathsf{Verify}(\mathsf{pp}, A_n, \tau_n, M, \sigma, F_{\mathsf{ver}})$: in classical mode, return $\mathsf{SignVerify}(\mathsf{pp}, A_n, M, \sigma)$. In ZKPoSP mode, return 1 iff $\mathsf{DVerify}(\mathsf{pp}, A_n, \tau_n, F_{\mathsf{ver}}) = 1$ and $\mathsf{SignVerify}(\mathsf{pp}, A_n, M, \sigma) = 1$.

Quantum vulnerability and the role of the derivation certificate. The signing scalar k_{L_n} satisfies $A_n = k_{L_n} \cdot B$; a QPT adversary recovers k_{L_n} via Shor's algorithm. This breaks the classical security argument: in standard Ed25519, a valid signature implicitly proves knowledge of the secret scalar used to compute the public key. Once Shor's algorithm is available, any party can recover the secret scalar from the public key, so a valid signature no longer certifies that the signer is the legitimate owner of the key.

To restore unforgeability in the post-quantum setting, the signer must prove knowledge of the seed from which the entire derivation chain originates, since the seed is bound to the public key only

through a chain of one-way hash functions and is not recoverable from any public value even by a quantum adversary. This is the role of the derivation certificate τ_n in ZKPoSP mode: rather than implicitly certifying key ownership through a classical signature, DCert produces a NIZK proof that A_n was honestly derived from a secret seed following the prescribed derivation path, establishing a sound link between the signer and the public key also in the presence of quantum machines.

For hardened keys, $w_n = k_{R_n}$ is a genuine quantum-safe witness: it is computed from $k_{L_{n-1}}$ and $k_{R_{n-1}}$ as private HMAC inputs, and recovering k_{R_n} from any public value requires inverting HMAC-SHA512 on private inputs, which happens with negligible probability even against a quantum adversary by the preimage resistance of SHA-512 and Grover's algorithm [Gro96]. For non-hardened keys, all HMAC inputs to z_n are public, so $z_R^{(n)}$ is freely computable and $w_n = k_{R_n}$ is not a quantum-safe witness; the derivation proof must therefore cover the full chain from the root seed in this case. This asymmetry is the structural property that enables sound last-step proofs for hardened keys but not for non-hardened keys, as analysed in Section 6.2.

4 Monolithic Post-Quantum Signing for BIP32-Ed25519

As a didactic stepping stone toward the more efficient constructions presented later, we show how to obtain a simple post-quantum signature scheme for BIP32-Ed25519 by replacing the classical signing step with a single monolithic NIZK proof covering the full derivation chain from the root seed to the signing key. This construction, which we call Σ, is not the main contribution of the paper: its proving cost grows linearly with derivation depth and it does not scale to practical wallet use. Its purpose is to isolate the core cryptographic argument and to establish the security reductions that will be reused in the more efficient constructions.

4.1 Signature Scheme Σ

The NIZK certifies knowledge of the full derivation chain from root seed to signing key, binding the message to the derived key material without revealing any private value. For clarity of exposition, we restrict throughout this section to purely non-hardened derivation paths; the general case involving mixed hardened and non-hardened steps requires a per-step type flag in the relation and is handled in Section 5.

Rather than including the HMAC tag $S = \mathsf{HMAC\text{-}SHA512}(k_{R_n}, M)$ in the public statement, we bind the message to the derived key material via an internal hash value $h = \mathsf{SHA256}(k_{R_n} \| M)$. Since k_{R_n} is derived from x_0 via a PRF chain and is computationally indistinguishable from a uniformly random 256-bit string to any adversary not knowing x_0, it acts as an unpredictable salt: finding any $(k^*_{R_n}, M^*)$ satisfying $\mathsf{SHA256}(k^*_{R_n} \| M^*) = h$ without knowledge of x_0 requires inverting SHA-256, which happens with negligible probability even against a quantum adversary by the preimage resistance of SHA-256 and Grover's algorithm [Gro96].

The NP relation certified by the NIZK is:

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

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 $(i_1, \ldots, i_n)$ are read from pp:

(i) **Root.** $k_0 \coloneqq \mathsf{Tweak}(\mathsf{SHA512}(x_0))$, $k_0 = k_{L_0} \| k_{R_0}$, $c_0 \coloneqq \mathsf{SHA256}(\mathtt{0x01} \| x_0)$.

(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 $z_i = z_L^{(i)} \| z_{\mathsf{mid}}^{(i)} \| z_R^{(i)}$ 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

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

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

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

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

(v) **Message binding.** $h \;=\; \mathsf{SHA256}(k_{R_n} \,\|\, M)$.

Public statement: $(M, A_n) \in \{0,1\}^* \times \mathbb{G}$. Witness: $\mathbf{w} = (x_0, k_{L_n}, k_{R_n}, h)$.

$\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ is a signature scheme for BIP32-Ed25519 defined as follows.

$\mathsf{Gen}(1^\lambda)$: sample $x_0 \xleftarrow{\$} \{0,1\}^{256}$; run $\mathsf{Set}(1^\lambda, T, x_0)$ as in Section 3.11 to obtain $(\mathsf{pp}, \mathsf{st}_0, \{(d_i, w_i)\})$; derive $(d_n, w_n, \mathsf{st}_n)$ along the prescribed path; set $\mathsf{sk}_n \coloneqq (x_0, k_{L_n}, k_{R_n})$, and $\mathsf{pk}_n \coloneqq A_n$. Output $(\mathsf{sk}_n, \mathsf{pk}_n)$.

$\mathsf{Sign}(\mathsf{sk}_n, M)$: parse $\mathsf{sk}_n = (x_0, k_{L_n}, k_{R_n})$; compute $h \coloneqq \mathsf{SHA256}(k_{R_n} \| M)$; set the public statement $\mathbf{x} \coloneqq (M, A_n)$ and the witness $\mathbf{w} \coloneqq (x_0, k_{L_n}, k_{R_n}, h)$; compute $\pi \leftarrow \mathsf{NIZKProve}(\mathsf{pp}, \mathbf{x}, \mathbf{w}, \mathcal{R})$; output $\sigma \coloneqq \pi$.

$\mathsf{Verify}(\mathsf{pp}, \mathsf{pk}_n, M, \sigma)$: parse $\mathsf{pk}_n = A_n$ and $\sigma = \pi$; output 1 iff $\mathsf{NIZKVerify}(\mathsf{pp}, (M, A_n), \pi) = 1$.

4.2 Security of Σ

Theorem 4.1 (Correctness of Σ)**.** *Let* NIZK *be a non-interactive argument of knowledge satisfying Section 3.6. Then for all key pairs* $(\mathsf{sk}_n, \mathsf{pk}_n)$ *produced by* $\mathsf{Gen}(1^\lambda)$ *and all* $M \in \{0,1\}^*$*:*

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

Proof. Let $\sigma = \pi = \mathsf{Sign}(\mathsf{sk}_n, M)$, where $\mathsf{sk}_n = (x_0, k_{L_n}, k_{R_n})$. By construction, $h = \mathsf{SHA256}(k_{R_n} \| M)$ and the witness $\mathbf{w} = (x_0, k_{L_n}, k_{R_n}, h)$ satisfies:

1. $k_{L_0} \| k_{R_0} \coloneqq \mathsf{Tweak}(\mathsf{SHA512}(x_0))$ and $c_0 \coloneqq \mathsf{SHA256}(\mathtt{0x01} \| x_0)$ by the root step of key derivation;
2. for $i = 1, \ldots, n$, the child-step recurrences hold by construction of Gen;
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. $h = \mathsf{SHA256}(k_{R_n} \| M)$ by the signing algorithm;
6. therefore $\big((M, A_n),\, \mathbf{w}\big) \in \mathcal{R}$.

By the completeness property of NIZK (Section 3.6), $\mathsf{NIZKVerify}(\mathsf{pp}, (M, A_n), \pi) = 1$ with probability $1 - \mathsf{negl}(\lambda)$, hence Verify accepts. □

Theorem 4.2 (EUF-CMA of Σ, restricted to non-hardened paths)**.** *Let* NIZK *be a non-interactive argument of knowledge satisfying Section 3.6. In the random oracle model for SHA-256 and SHA-512,* Σ *restricted to purely non-hardened derivation paths is EUF-CMA secure against PPT adversaries.*

Proof. Suppose for contradiction that there exists a PPT adversary $\mathcal{A}$ that wins the EUF-CMA game for Σ with non-negligible probability ε. That is, $\mathcal{A}$ makes signing queries $\mathcal{O}_{\mathsf{Sign}}(M_1), \ldots, \mathcal{O}_{\mathsf{Sign}}(M_q)$ and random oracle queries, and outputs a forgery (M^*, σ^*) such that $\mathsf{Verify}(\mathsf{pp}, \mathsf{pk}_n, M^*, \sigma^*) = 1$ and $M^* \notin \{M_1, \ldots, M_q\}$, with probability at least ε.

Game G_0 (real game). The challenger runs $\mathsf{Gen}(1^\lambda)$ to obtain $(\mathsf{sk}_n, \mathsf{pk}_n)$ where $\mathsf{sk}_n = (x_0, k_{L_n}, k_{R_n})$, $c_0 = \mathcal{O}_{\mathsf{256}}(\mathtt{0x01} \| x_0)$, and $k_{L_0} \| k_{R_0} = \mathsf{Tweak}(\mathcal{O}_{\mathsf{512}}(x_0))$, and gives pk_n to $\mathcal{A}$. Here $\mathcal{O}_{\mathsf{256}} : \{0,1\}^* \to \{0,1\}^{256}$ and $\mathcal{O}_{\mathsf{512}} : \{0,1\}^* \to \{0,1\}^{512}$ are the random oracles modelling SHA-256 and SHA-512 respectively. For each signing query M_i, the challenger computes:

$$h_i \coloneqq \mathcal{O}_{\mathsf{256}}(k_{R_n} \| M_i), \quad \mathbf{w}_i \coloneqq (x_0, k_{L_n}, k_{R_n}, h_i), \quad \pi_i \leftarrow \mathsf{NIZKProve}(\mathsf{pp}, (M_i, A_n), \mathbf{w}_i),$$

and returns $\sigma_i \coloneqq \pi_i$ to $\mathcal{A}$. By assumption, $\mathcal{A}$ outputs a forgery (M^*, σ^*) with probability ε.

Game G_1 (simulated proofs). The challenger is identical to G_0 except that for each signing query M_i, the proof is produced by the NIZK simulator $\mathcal{S}$ guaranteed by the zero-knowledge property of NIZK (Section 3.6):

$$\pi_i \leftarrow \mathcal{S}(\mathsf{pp}, (M_i, A_n)).$$

The response to each query M_i is $\sigma_i \coloneqq \pi_i$ as before.

Lemma 4.3. $|\Pr[\mathcal{A} \textit{ wins } G_0] - \Pr[\mathcal{A} \textit{ wins } G_1]| \leq \mathsf{negl}(\lambda)$.

Proof. Suppose $\mathcal{A}$ distinguishes G_0 from G_1 with non-negligible advantage δ. We construct a PPT distinguisher $\mathcal{D}$ against the zero-knowledge property of NIZK. $\mathcal{D}$ receives pk_n and has access to either a real prover oracle $\mathsf{NIZKProve}(\mathsf{pp}, \cdot, \cdot)$ or a simulator oracle $\mathcal{S}(\mathsf{pp}, \cdot)$. $\mathcal{D}$ runs $\mathcal{A}$, forwarding each signing query M_i to its oracle with statement (M_i, A_n) and returning the resulting proof π_i to $\mathcal{A}$. $\mathcal{D}$ answers all random oracle queries honestly. Since $\mathcal{D}$ perfectly simulates G_0 when given the real prover and G_1 when given the simulator, $\mathcal{D}$ distinguishes the two oracles with advantage δ, contradicting the zero-knowledge property of NIZK (Section 3.6). Hence $\delta \leq \mathsf{negl}(\lambda)$. □

Analysis of G_1. Since G_0 and G_1 are computationally indistinguishable by Lemma 4.3, $\mathcal{A}$ wins G_1 with probability at least $\varepsilon - \mathsf{negl}(\lambda)$, which is non-negligible. In G_1, $\mathcal{A}$ has seen only simulated proofs $\pi_1, \ldots, \pi_q$ produced by $\mathcal{S}$ for statements $(M_1, A_n), \ldots, (M_q, A_n)$. If $\mathcal{A}$ outputs a forgery (M^*, σ^*) with $\mathsf{NIZKVerify}(\mathsf{pp}, (M^*, A_n), \sigma^*) = 1$ and $M^* \notin \{M_1, \ldots, M_q\}$, then by simulation extractability of NIZK (Section 3.6), there exists a PPT extractor $\mathcal{E}$ that, given the same view as $\mathcal{A}$ including the simulated proofs and all random oracle queries, extracts a valid witness $\mathbf{w}^* = (x_0^*, k_{L_n}^*, k_{R_n}^*, h^*)$ satisfying all conditions (i)–(v) of $\mathcal{R}$ for the statement (M^*, A_n). We now argue that no PPT adversary can produce such a witness with non-negligible probability.

By condition (i), $k_{L_0}^* \| k_{R_0}^* = \mathsf{Tweak}(\mathcal{O}_{\mathsf{512}}(x_0^*))$ and $c_0^* = \mathcal{O}_{\mathsf{256}}(\mathtt{0x01} \| x_0^*)$. By conditions (ii)–(iv), the public chain $(A_1, \ldots, A_{n-1}, c_1, \ldots, c_{n-1})$ is fixed and given in pp, so every $z_L^{(i)}$ and $z_R^{(i)}$ in the recurrence is forced to take the unique value computed from this fixed public chain (up to the negligible probability of a chain-code or public-key collision), and condition (iv) requires $k_{L_n}^* \cdot B = A_n$, the fixed honest public key. Unrolling the recurrence, this means x_0^* must satisfy $c_0^* = c_0$ and $k_{L_0}^* \cdot B = A_0$, both fixed public targets determined by the honestly generated key. By condition (i), this means x_0^* must satisfy:

$$\mathcal{O}_{\mathsf{256}}(\mathtt{0x01} \| x_0^*) = c_0 \qquad \text{and} \qquad \mathsf{Tweak}(\mathcal{O}_{\mathsf{512}}(x_0^*))[0:256] \cdot B = A_0.$$

This is a preimage condition against the fixed values c_0 and A_0: the targets are already pinned by the honest key, and the adversary must find any x_0^* (whether or not $x_0^* = x_0$) mapping onto them. In the ROM, the only way to evaluate $\mathcal{O}_{\mathsf{256}}(\mathtt{0x01} \| x_0^*)$ or $\mathcal{O}_{\mathsf{512}}(x_0^*)$ is to query the respective oracle directly. Since x_0 is sampled uniformly at random and never revealed to $\mathcal{A}$, for each distinct query x made by $\mathcal{A}$, the probability that $\mathcal{O}_{\mathsf{256}}(\mathtt{0x01} \| x) = c_0$ is exactly 2^{-256}, independently of all other queries. After at most q_{RO} random oracle queries, the probability that any query hits c_0 is

at most $q_{\mathsf{RO}} \cdot 2^{-256}$, which is negligible for polynomial q_{RO}. Therefore x_0^* cannot be found with non-negligible probability.

Since x_0^* cannot be found with non-negligible probability, and condition (i) fixes $k_{R_0}^*$ as the right half of $\mathsf{Tweak}(\mathcal{O}_{\mathsf{512}}(x_0^*))$ simultaneously with $k_{L_0}^*$, $k_{R_n}^*$ is determined by the same unreachable x_0^* via the fixed public shift established by conditions (ii)–(iii); there is no way for $\mathcal{A}$ to supply a valid $k_{R_n}^*$ independently of finding x_0^*. Consequently condition (v), which requires $h^* = \mathcal{O}_{\mathsf{256}}(k_{R_n}^* \| M^*)$, cannot be satisfied with a correct $k_{R_n}^*$ except with negligible probability.

Therefore no PPT adversary can produce a valid witness $\mathbf{w}^*$ satisfying all conditions (i)–(v) of $\mathcal{R}$ with non-negligible probability, so $\Pr[\mathcal{A} \text{ wins } G_1] \leq \mathsf{negl}(\lambda)$, contradicting our assumption that ε is non-negligible. □

Remark 4.4 (Why the reduction must go all the way to the seed). On a purely non-hardened path, every chain code c_{i-1} and every intermediate value z_i is publicly computable from c_0 and the public chain, so no intermediate step of the derivation offers any hardness to reduce to. The only secret anywhere in the chain is the seed x_0 itself; consequently the proof is forced to bind $k_{L_n}^*$ and $k_{R_n}^*$ all the way back to a preimage condition on x_0, rather than stopping at some intermediate hardened-looking step. This is specific to purely non-hardened paths; once a hardened step is present, c_{i-1} at that step is genuinely secret and offers an earlier point at which the reduction can stop, as treated in Section 6.2.

Remark 4.5 (Post-quantum security of Σ). The security proof consumes only the NIZK properties of zero-knowledge and simulation extractability, with SHA-256 and SHA-512 modelled as random oracles. The discrete logarithm assumption is not used anywhere in the reduction: even if a QPT adversary recovers k_{L_n} from A_n via Shor's algorithm and inverts the public chain to obtain the fixed target k_{L_0}, finding x_0^* matching this target still requires querying $\mathcal{O}_{\mathsf{512}}$ on x_0^* and matching the output, which succeeds with probability at most $q_{\mathsf{RO}} \cdot 2^{-256}$ even for a QPT adversary making polynomially many queries. Post-quantum security is therefore conjectured to hold since the NIZK properties are conjectured to resist QPT adversaries.

5 ZKPoSP: Split-Proof Architecture

The relation $\mathcal{R}$ of Section 4.1 can be split into two separate relations, each proved by its own NIZK. For clarity of exposition we restrict, as in Section 4.1, to purely non-hardened derivation paths; the hardened case is treated in Section 6. The two proofs are linked at verification time through a single shared public value $h_{k_R} = \mathsf{SHA256}(k_{R_n})$, which appears in the statement of both proofs to ensure that both refer to the same k_{R_n}, namely the one produced by the derivation chain. Publishing h_{k_R} does not weaken security: an adversary seeing it cannot recover k_{R_n} without inverting SHA-256.

The first part is the derivation proof π_{deriv}. It confirms that the public key A_n was correctly derived from a seed following the BIP32-Ed25519 rules, and that h_{k_R} is the hash of the k_{R_n} produced by that derivation. This proof is generated once per key pair and reused for every signature made with that key.

The second part is the signing proof π_{sign}. It confirms that the signer holds a value k_{R_n} consistent with the public h_{k_R}, and that the signature hash h was computed correctly from k_{R_n} and the message M. This proof is generated once per message.

Because the derivation and signing procedures are independent, their proofs can be generated at different times. In particular, π_{deriv} can be computed for each key pair as soon as the seed x_0 is known, even before any message needs to be signed. Later, when a signature is needed, only π_{sign} must be generated, and the verifier checks both proofs together against the shared public value h_{k_R}.

5.1 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 non-hardened BIP32-Ed25519 chain of Section 3.11, and that h_{k_R} is the hash of the chain-derived right half k_{R_n}:

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

Public statement: $(A_n,\, h_{k_R}) \in \mathbb{G} \times \{0,1\}^{256}$.
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 3.11.

(D-i) **Root.** $k_{L_0} \| k_{R_0} \coloneqq \mathsf{Tweak}(\mathsf{SHA512}(x_0))$, $c_0 \coloneqq \mathsf{SHA256}(\mathtt{0x01} \| x_0)$.

(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 $k_{R_n} = k_{R_n}^{\mathsf{derived}}$, that is, 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) **Hash binding.** $h_{k_R} = \mathsf{SHA256}(k_{R_n})$.

Conditions (D-iii) and (D-v) enforce that both k_{L_n} and k_{R_n} are the correct outputs of the derivation chain rooted at x_0, and that h_{k_R} is the correct hash of k_{R_n}, so neither can be freely chosen by a forger independently of A_n.

5.2 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 h_{k_R}, and that the signature hash h was honestly computed from k_{R_n} and the message M:

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

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

(S-i) **Binding to derivation.** $\mathsf{SHA256}(k_{R_n}) = h_{k_R}$. This links k_{R_n} to the one certified by π_{deriv} without re-executing the derivation chain.

(S-ii) **Signature hash.** $h = \mathsf{SHA256}(k_{R_n} \| M)$.

5.3 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 h_{k_R}, which must be identical in both proofs:

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

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

(C-i) **Derivation proof.** $\mathsf{NIZKVerify}\big(\mathsf{pp},\, (A_n,\, h_{k_R}),\, \pi_{\mathsf{deriv}}\big) = 1$, certifying that $\big((A_n,\, h_{k_R}),\, \mathbf{w}_1\big) \in \mathcal{R}_{\mathsf{deriv}}$ for some $\mathbf{w}_1$.

(C-ii) **Signing proof.** $\mathsf{NIZKVerify}\big(\mathsf{pp},\, (M,\, h_{k_R}),\, \pi_{\mathsf{sign}}\big) = 1$, certifying that $\big((M,\, h_{k_R}),\, \mathbf{w}_2\big) \in \mathcal{R}_{\mathsf{sign}}$ for some $\mathbf{w}_2$, where h_{k_R} is the same value appearing in condition (C-i).

The verifier additionally checks cross-proof consistency: the value h_{k_R} extracted from π_{deriv}'s public statement must equal the value h_{k_R} in π_{sign}'s public statement, ensuring both proofs refer to the same derived key. This check binds the signing key to the derivation chain without re-executing it. Equivalently, and more directly, the verifier can dispense with an explicit comparison by using the h_{k_R} extracted from π_{deriv}'s public statement directly as the public input supplied to $\mathsf{NIZKVerify}$ when checking π_{sign}, so that consistency is enforced by the verifier's own data flow; this is the convention used in the Verify algorithm of Section 5.4. This shortcut is sound only if the derivation proof is verified first: the verifier must check $\mathsf{NIZKVerify}(\mathsf{pp}, (A_n, h_{k_R}), \pi_{\mathsf{deriv}}) = 1$ before forwarding the extracted h_{k_R} into the signing check, since an unverified π_{deriv} carries no guarantee that its h_{k_R} is the hash of a properly derived k_{R_n}. This check is enforced trivially for the honest protocol execution, since a single h_{k_R} is computed and reused for both proofs; its role is to rule out an adversary who supplies π_{deriv} and π_{sign} bound to different k_{R_n} values, as ruled out by the EUF-CMA reduction of Section 5.4.

Once π_{deriv} has been precomputed for a key pair, generating a signature requires only π_{sign}, whose trace length is fixed by the two SHA-256 invocations of conditions (S-i)–(S-ii) and is independent of the derivation depth n.

5.4 ZKPoSP for Non-Hardened BIP32-Ed25519

We instantiate ZKPoSP for non-hardened BIP32-Ed25519 within the HDW framework of Section 3.8, using Set, DPub, DPriv as in Section 3.11.

$\mathsf{Set}(1^\lambda, T, S)$ runs as in Section 3.11 with seed $S = x_0$: it computes $k_{L_0} \| k_{R_0} \coloneqq \mathsf{Tweak}(\mathsf{SHA512}(x_0))$ and $c_0 \coloneqq \mathsf{SHA256}(\mathtt{0x01} \| x_0)$, then derives (k_{L_i}, k_{R_i}, c_i) at each node along T via DPriv. It outputs:

- $\mathsf{pp} \coloneqq A_0$,
- $\mathsf{st}_0 \coloneqq (A_0, c_0, \perp)$,
- for every $v_i \in V$: the derivation key $d_i \coloneqq k_{L_i}$, the witness key $w_i \coloneqq k_{R_i}$, and the signing key $\mathsf{sk}_i \coloneqq (k_{L_i}, k_{R_i})$.

The seed x_0 is not part of any node key pair. It is supplied separately as the anchor witness $a \coloneqq x_0$ when calling DCert, allowing DCert to build the full-chain witness for $\mathcal{R}_{\mathsf{deriv}}$ without ever exposing x_0 through $\mathcal{O}_{\mathrm{Corr}}$, which returns only $\mathsf{sk}_i = (d_i, w_i) = (k_{L_i}, k_{R_i})$. Since every node in a non-hardened tree is non-hardened, the deepest hardened ancestor of any v_n is always the root v_0, so DCert is always invoked with $(d_0, w_0, \mathsf{st}_0, a) = (k_{L_0}, k_{R_0}, \mathsf{st}_0, x_0)$.

$\mathsf{DCert}(\mathsf{pp}, d_0, w_0, \mathsf{st}_0, v_0, v_n, a, F)$: parse $d_0 = k_{L_0}$, $w_0 = k_{R_0}$, $a = x_0$. Here $F \coloneqq \mathsf{NIZKProve}(\cdot,\cdot,\cdot)$ instantiated for the relation $\mathcal{R}_{\mathsf{deriv}}$ of Section 5.1. Apply DPriv iteratively along the non-hardened path from v_0 to v_n to obtain (k_{L_n}, k_{R_n}). Compute $h_{k_R} \coloneqq \mathsf{SHA256}(k_{R_n})$ and $\pi_{\mathsf{deriv}} \leftarrow \mathsf{NIZKProve}(\mathsf{pp}, (A_n, h_{k_R}), (x_0, k_{L_n}, k_{R_n}))$. Output $\tau_n \coloneqq (\pi_{\mathsf{deriv}}, h_{k_R})$.

$\mathsf{DVerify}(\mathsf{pp}, \mathsf{pk}_n, \tau_n, F_{\mathsf{ver}})$: parse $\mathsf{pk}_n = A_n$, $\tau_n = (\pi_{\mathsf{deriv}}, h_{k_R})$. Here $F_{\mathsf{ver}} \coloneqq \mathsf{NIZKVerify}(\cdot,\cdot,\cdot)$ instantiated for the relation $\mathcal{R}_{\mathsf{deriv}}$ of Section 5.1. Output 1 iff $\mathsf{NIZKVerify}(\mathsf{pp}, (A_n, h_{k_R}), \pi_{\mathsf{deriv}}) = 1$.

$\mathsf{Sign}(\mathsf{sk}_n, M)$: parse $\mathsf{sk}_n = (k_{L_n}, k_{R_n})$. Compute $h_{k_R} \coloneqq \mathsf{SHA256}(k_{R_n})$, $h \coloneqq \mathsf{SHA256}(k_{R_n} \| M)$, and $\pi_{\mathsf{sign}} \leftarrow \mathsf{NIZKProve}(\mathsf{pp}, (M, h_{k_R}), (k_{R_n}, h))$ instantiated for the relation $\mathcal{R}_{\mathsf{sign}}$ of Section 5.2. Output $\sigma \coloneqq (\pi_{\mathsf{sign}}, h_{k_R})$.

$\mathsf{SignVerify}(\mathsf{pp}, \mathsf{pk}_n, M, \sigma)$: parse $\sigma = (\pi_{\mathsf{sign}}, h_{k_R})$. Here $F_{\mathsf{ver}} \coloneqq \mathsf{NIZKVerify}(\cdot,\cdot,\cdot)$ instantiated for the relation $\mathcal{R}_{\mathsf{sign}}$ of Section 5.2. Output 1 iff $\mathsf{NIZKVerify}(\mathsf{pp}, (M, h_{k_R}), \pi_{\mathsf{sign}}) = 1$.

$\mathsf{Verify}(\mathsf{pp}, \mathsf{pk}_n, \tau_n, M, \sigma, F_{\mathsf{ver}})$: parse $\tau_n = (\pi_{\mathsf{deriv}}, h_{k_R})$ and $\sigma = (\pi_{\mathsf{sign}}, h'_{k_R})$. Output 1 iff $\mathsf{DVerify}(\mathsf{pp}, \mathsf{pk}_n, \tau_n, F_{\mathsf{ver}}) = 1$, $h_{k_R} = h'_{k_R}$, and $\mathsf{SignVerify}(\mathsf{pp}, \mathsf{pk}_n, M, \sigma) = 1$.

Theorem 5.1 (Correctness of ZKPoSP)**.** *Let* NIZK *be a non-interactive argument of knowledge satisfying Section 3.6, and let* $F \coloneqq \mathsf{NIZKProve}$ *for* $\mathcal{R}_{\mathsf{deriv}}$ *and* $F_{\mathsf{ver}} \coloneqq \mathsf{NIZKVerify}$ *for* $\mathcal{R}_{\mathsf{deriv}}$ *and* $\mathcal{R}_{\mathsf{sign}}$*. Then ZKPoSP for non-hardened BIP32-Ed25519 satisfies Definition 3.11: for every tree* $T = (V, E)$ *with all nodes non-hardened, all* $v_j \in V$*, all* $S \in \mathcal{S}$*, and all* $M \in \mathcal{M}$*,*

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

where $(\mathsf{pp}, \mathsf{st}_0, \{\mathsf{sk}_i\}) = \mathsf{Set}(1^\lambda, T, S)$*,* pk_j *is included in* st_j*, and* $\tau_j = \mathsf{DCert}(\mathsf{pp}, d_0, w_0, \mathsf{st}_0, v_0, v_j, x_0, F)$*, since* T *has no hardened nodes and the deepest hardened ancestor of* v_j *defaults to the root* v_0*, with anchor witness* $a \coloneqq x_0$*.*

Proof. Fix T, v_j, S, M as in the theorem statement, and let $\mathsf{sk}_j = (k_{L_j}, k_{R_j})$. By construction of Sign, $\sigma \coloneqq \mathsf{Sign}(\mathsf{sk}_j, M) = (\pi_{\mathsf{sign}}, h_{k_R})$, where $h_{k_R} \coloneqq \mathsf{SHA256}(k_{R_j})$, $h \coloneqq \mathsf{SHA256}(k_{R_j} \| M)$, and $\pi_{\mathsf{sign}} \leftarrow \mathsf{NIZKProve}(\mathsf{pp}, (M, h_{k_R}), (k_{R_j}, h))$ under $\mathcal{R}_{\mathsf{sign}}$. By construction of DCert, $\tau_j = (\pi_{\mathsf{deriv}}, h_{k_R})$ with $\pi_{\mathsf{deriv}} \leftarrow \mathsf{NIZKProve}(\mathsf{pp}, (A_j, h_{k_R}), (x_0, k_{L_j}, k_{R_j}))$ under $\mathcal{R}_{\mathsf{deriv}}$, using the same h_{k_R}, since DCert is invoked with $d_0 = k_{L_0}$, $w_0 = k_{R_0}$, anchor witness $a = x_0$, and applies DPriv along the purely non-hardened path from v_0 to v_j.

The witness $\mathbf{w}_1 = (x_0, k_{L_j}, k_{R_j})$ satisfies conditions (D-i)–(D-v) of $\mathcal{R}_{\mathsf{deriv}}$: (D-i)–(D-iii) hold by the honest derivation chain underlying Set and DPriv (Section 3.11), (D-iv) holds since $A_j = k_{L_j} \cdot B$ by definition of key generation, and (D-v) holds by construction of h_{k_R}. The witness $\mathbf{w}_2 = (k_{R_j}, h)$ satisfies conditions (S-i)–(S-ii) of $\mathcal{R}_{\mathsf{sign}}$: (S-i) holds since both occurrences of h_{k_R} are identical by construction, and (S-ii) holds by construction of h.

By the completeness property of NIZK (Section 3.6), $\mathsf{NIZKVerify}(\mathsf{pp}, (A_j, h_{k_R}), \pi_{\mathsf{deriv}}) = 1$ and $\mathsf{NIZKVerify}(\mathsf{pp}, (M, h_{k_R}), \pi_{\mathsf{sign}}) = 1$, each with probability $1 - \mathsf{negl}(\lambda)$, hence both jointly with probability $1 - \mathsf{negl}(\lambda)$ by a union bound. By definition of $\mathsf{DVerify}$, $\mathsf{DVerify}(\mathsf{pp}, \mathsf{pk}_j, \tau_j, F_{\mathsf{ver}}) = 1$. By definition of $\mathsf{SignVerify}$, $\mathsf{SignVerify}(\mathsf{pp}, \mathsf{pk}_j, M, \sigma) = 1$. Since $\sigma = (\pi_{\mathsf{sign}}, h_{k_R})$ and $\tau_j = (\pi_{\mathsf{deriv}}, h_{k_R})$ carry the same h_{k_R} by construction, the equality check $h_{k_R} = h'_{k_R}$ in Verify holds trivially. Therefore all three conditions of Verify hold simultaneously with probability $1 - \mathsf{negl}(\lambda)$. □

Theorem 5.2 (EUF-CMA of $(\mathsf{Sign}, \mathsf{SignVerify})$)**.** *Let* NIZK *be a non-interactive argument of knowledge satisfying Section 3.6. In the random oracle model for SHA-256 and SHA-512,* $(\mathsf{Sign}, \mathsf{SignVerify})$ *for non-hardened BIP32-Ed25519 is EUF-CMA secure (Definition 3.10) against PPT adversaries, treating* h_{k_R} *as a public parameter fixed by the honestly generated key pair.*

Proof. Suppose for contradiction that there exists a PPT adversary $\mathcal{A}$ that wins the EUF-CMA game for $(\mathsf{Sign}, \mathsf{SignVerify})$ with non-negligible probability ε. $\mathcal{A}$ is given $\mathsf{pk}_n = A_n$ and $h_{k_R} = \mathsf{SHA256}(k_{R_n})$, makes signing queries $M_1, \ldots, M_q$, and outputs a forgery (M^*, σ^*) with $\sigma^* = (\pi^*_{\mathsf{sign}}, h^*_{k_R})$ such that $\mathsf{SignVerify}(\mathsf{pp}, A_n, M^*, \sigma^*) = 1$ and $M^* \notin \{M_1, \ldots, M_q\}$, with probability at least ε.

Game G_0 (real game). The challenger runs $\mathsf{Set}(1^\lambda, T, S)$ to obtain $\mathsf{sk}_n = (k_{L_n}, k_{R_n})$ and $\mathsf{pk}_n = A_n$, computes $h_{k_R} \coloneqq \mathcal{O}_{256}(k_{R_n})$, and gives (A_n, h_{k_R}) to $\mathcal{A}$, where $\mathcal{O}_{256}$ models SHA-256 as a random oracle. For each signing query M_i, the challenger computes $h_i \coloneqq \mathcal{O}_{256}(k_{R_n} \| M_i)$, $\pi_{\mathsf{sign},i} \leftarrow \mathsf{NIZKProve}(\mathsf{pp}, (M_i, h_{k_R}), (k_{R_n}, h_i))$ under $\mathcal{R}_{\mathsf{sign}}$, and returns $\sigma_i \coloneqq (\pi_{\mathsf{sign},i}, h_{k_R})$. By assumption $\mathcal{A}$ outputs a forgery with probability ε.

Game G_1 (simulated proofs). Identical to G_0 except every $\pi_{\mathsf{sign},i}$ is produced by the NIZK simulator: $\pi_{\mathsf{sign},i} \leftarrow \mathcal{S}(\mathsf{pp}, (M_i, h_{k_R}))$.

Lemma 5.3. $|\Pr[\mathcal{A} \textit{ wins } G_0] - \Pr[\mathcal{A} \textit{ wins } G_1]| \le \mathsf{negl}(\lambda)$.

Proof. Identical argument to Lemma 4.3: a distinguisher $\mathcal{D}$ against the zero-knowledge property of NIZK for $\mathcal{R}_{\mathsf{sign}}$ simulates either game perfectly depending on which oracle it is given, achieving the same advantage as $\mathcal{A}$. □

Analysis of G_1. By Lemma 5.3, $\mathcal{A}$ wins G_1 with non-negligible probability $\varepsilon - \mathsf{negl}(\lambda)$. If $\mathcal{A}$ outputs (M^*, σ^*) with $\mathsf{SignVerify}(\mathsf{pp}, A_n, M^*, \sigma^*) = 1$ and $M^* \notin \{M_1, \ldots, M_q\}$, then $\sigma^* = (\pi^*_{\mathsf{sign}}, h^*_{k_R})$ with $\mathsf{NIZKVerify}(\mathsf{pp}, (M^*, h^*_{k_R}), \pi^*_{\mathsf{sign}}) = 1$. By simulation extractability of NIZK, a PPT extractor $\mathcal{E}$ extracts a witness $(k^*_{R_n}, h^*)$ satisfying (S-i)–(S-ii) of $\mathcal{R}_{\mathsf{sign}}$ for statement $(M^*, h^*_{k_R})$.

By (S-i), $\mathcal{O}_{256}(k^*_{R_n}) = h^*_{k_R}$. For the forgery to pass Verify's equality check $h_{k_R} = h'_{k_R}$, we need $h^*_{k_R} = h_{k_R} = \mathcal{O}_{256}(k_{R_n})$. Finding any $k^*_{R_n}$ such that $\mathcal{O}_{256}(k^*_{R_n}) = h_{k_R}$ means to find a preimage of a fixed target: in the ROM, each query hits h_{k_R} with probability 2^{-256}, so after at most q_{RO} queries, $\Pr[k^*_{R_n} \text{ found}] \le q_{\mathsf{RO}} \cdot 2^{-256}$, which is negligible.

Therefore no PPT adversary can produce a valid forgery with non-negligible probability, contradicting that $\varepsilon - \mathsf{negl}(\lambda)$ is non-negligible. □

Theorem 5.4 (Hierarchical EUF-CMA of ZKPoSP, restricted to non-hardened trees)**.** *Let* NIZK *be a non-interactive argument of knowledge satisfying Section 3.6. Let SHA-256 and SHA-512 be random oracles. ZKPoSP for BIP32-Ed25519 restricted to trees T in which every node $v_i \in V$ is non-hardened is hierarchically EUF-CMA secure (Definition 3.12) against PPT adversaries.*

Proof. Suppose for contradiction that there exists a PPT adversary $\mathcal{A}$ winning $G^{\mathrm{heuf}}_{\Pi,\mathcal{A}}(\lambda, T)$ with non-negligible probability ε, for some tree T with all nodes non-hardened. By Definition 3.12, $\mathcal{A}$ outputs $(v^*, M^*, \sigma^*, \tau^*)$ with $\sigma^* = (\pi^*_{\mathsf{sign}}, h^*_{k_R})$ and $\tau^* = (\pi^*_{\mathsf{deriv}}, h^{**}_{k_R})$ such that $\mathsf{Verify}(\mathsf{pp}, \mathsf{pk}_{v^*}, \tau^*, M^*, \sigma^*, F_{\mathsf{ver}}) = 1$, $v^* \notin Q_{\mathrm{Corr}}$, and $(M^*, v^*) \notin Q_{\mathrm{Sign}}$. Since Verify accepts, we have: (1) $\mathsf{NIZKVerify}(\mathsf{pp}, (A_{v^*}, h^{**}_{k_R}), \pi^*_{\mathsf{deriv}}) = 1$; (2) $h^{**}_{k_R} = h^*_{k_R}$; (3) $\mathsf{SignVerify}(\mathsf{pp}, \mathsf{pk}_{v^*}, M^*, \sigma^*) = 1$. By (2), we write $h^*_{k_R}$ for the common value throughout.

Game G_0 (real game). The challenger runs $\mathsf{Set}(1^\lambda, T, S)$ to obtain $(\mathsf{pp}, \mathsf{st}_0, \{\mathsf{sk}_i\}_{v_i \in V})$ and plays $G^{\mathrm{heuf}}_{\Pi,\mathcal{A}}(\lambda, T)$ honestly, using $x_0 = S$ as the anchor witness a for all $\mathcal{O}_{\mathrm{Sign}}$ queries. The seed x_0 is never output by Set, never returned by $\mathcal{O}_{\mathrm{Corr}}$ (which returns only $\mathsf{sk}_i = (k_{L_i}, k_{R_i})$), and never returned by $\mathcal{O}_{\mathrm{Sign}}$ (which returns $(\mathsf{pk}_{v_i}, \tau_i, \sigma_i)$ with no seed material). Therefore x_0 is information-theoretically hidden from $\mathcal{A}$ throughout the game. By assumption $\mathcal{A}$ wins with probability ε.

Game G_1 (simulated derivation and signing proofs). Identical to G_0 except every $\pi_{\mathsf{deriv},i}$ and $\pi_{\mathsf{sign},i}$ issued via $\mathcal{O}_{\mathrm{Sign}}$ is produced by the NIZK simulator $\mathcal{S}$ for the corresponding statement, rather than by $\mathsf{NIZKProve}$. By the zero-knowledge property of NIZK, applied twice (once for $\mathcal{R}_{\mathsf{deriv}}$ and

once for $\mathcal{R}_{\mathsf{sign}}$) and combined via a union bound over the polynomially many queries, $\mathcal{A}$ wins G_1 with probability at least $\varepsilon - \mathsf{negl}(\lambda)$, which is non-negligible.

Analysis of G_1. In G_1, $\mathcal{A}$ outputs a valid forgery with non-negligible probability. Since $\mathsf{NIZKVerify}(\mathsf{pp}, (A_{v^*}, h^*_{k_R}), \pi^*_{\mathsf{deriv}}) = 1$ and $(M^*, v^*) \notin Q_{\mathrm{Sign}}$, π^*_{deriv} is a proof for a statement not previously simulated at v^* with the same $h^*_{k_R}$. By simulation extractability of NIZK, a PPT extractor $\mathcal{E}$ extracts a witness $(x^*_0, k^*_{L_{v^*}}, k^*_{R_{v^*}})$ satisfying conditions (D-i)–(D-v) of $\mathcal{R}_{\mathsf{deriv}}$ for statement $(A_{v^*}, h^*_{k_R})$. By condition (D-iv), $k^*_{L_{v^*}} \cdot B = A_{v^*}$, the honestly generated public key at v^*. Unrolling conditions (D-i)–(D-iii) as in Theorem 4.2, x^*_0 must satisfy $\mathcal{O}_{256}(\texttt{0x01} \| x^*_0) = c_0$ against the fixed honest target c_0; since x_0 is never revealed to $\mathcal{A}$, this is a preimage condition against a uniformly random target, which succeeds with probability at most $q_{\mathsf{RO}} \cdot 2^{-256}$, negligible. Hence the extracted $k^*_{R_{v^*}}$ is the unique value satisfying $h^*_{k_R} = \mathcal{O}_{256}(k^*_{R_{v^*}})$ (by condition (D-v)), binding $h^*_{k_R}$ to a specific $k^*_{R_{v^*}}$ that $\mathcal{A}$ cannot choose freely.

Since $h^*_{k_R}$ is bound to $k^*_{R_{v^*}}$ as above, and $(M^*, v^*) \notin Q_{\mathrm{Sign}}$ means the signing oracle at v^* was never queried on M^*, the pair $(M^*, h^*_{k_R})$ was never produced by $\mathcal{O}_{\mathrm{Sign}}$. By Theorem 5.2, no PPT adversary can produce a valid $(\pi^*_{\mathsf{sign}}, h^*_{k_R})$ passing $\mathsf{SignVerify}$ for a fresh message under the fixed $h^*_{k_R}$, except with negligible probability.

Both the extraction of a valid π^*_{deriv} and the production of a valid σ^* fail except with negligible probability, so $\Pr[\mathcal{A} \text{ wins } G_1] \leq \mathsf{negl}(\lambda)$, contradicting that $\varepsilon - \mathsf{negl}(\lambda)$ is non-negligible. □

Remark 5.5 (Post-quantum security of ZKPoSP). The security proofs for Theorems 5.2 and 5.4 consume only the following assumptions: the zero-knowledge and simulation extractability properties of NIZK, and the preimage resistance of SHA-256 and SHA-512 modelled as random oracles. The discrete logarithm assumption is not used anywhere: even if a QPT adversary recovers k_{L_n} from A_n via Shor's algorithm, finding x^*_0 satisfying $\mathcal{O}_{256}(\texttt{0x01} \| x^*_0) = c_0$ still succeeds with probability at most $q_{\mathsf{RO}} \cdot 2^{-256}$. The NIZK is instantiated with a ZK-STARK, which is statistical zero-knowledge and is conjectured to satisfy simulation extractability against QPT adversaries. Post-quantum security of ZKPoSP is therefore conjectured to hold under the assumption that the ZK-STARK retains its properties against quantum adversaries and that SHA-256 and SHA-512 retain their preimage resistance.

6 Can the Derivation Proof Be Shortened?

The proving times reported in Section 9.1 grow linearly with derivation depth n. A natural question is whether the derivation proof can be shortened: instead of proving the full chain from the root seed, can the prover establish key ownership by proving only a single parent-to-child derivation step? We call this a last-step proof.

The answer depends on whether the derivation step involves a quantum-safe witness, that is, a private value that is bound to the seed by a one-way function and cannot be recovered from the public key even by a quantum adversary. If such a value exists and appears as an input to the last step, a last-step proof is sound. If all inputs to the last step are either public or quantum-recoverable, the proof is unsound: an adversary can freely choose a fake parent state, compute the step outputs from public data, and produce a passing proof with no connection to the legitimate owner's seed. When a sound last-step proof exists, the anchor witness a of Section 5 is instantiated as the private key pair (d_{n-1}, w_{n-1}) of the parent of the hardened anchor node, rather than the root seed x_0; this is the pruned variant of Section 7.4.

A last-step derivation proof for a child node v_n with parent v_{n-1} is a NIZK for the following

generic relation, instantiated differently for each scheme:

$$\mathcal{R}_{\mathsf{last}} \coloneqq \Big\{ \big((\mathsf{pk}_n,\, h_{\mathsf{bind}}),\, (d_{n-1},\, w_{n-1},\, d_n,\, w_n) \big) \;\Big|\; \text{(L-i)–(L-iii) hold} \Big\}.$$

Public statement: $(\mathsf{pk}_n,\, h_{\mathsf{bind}})$, where pk_n is the child public key and h_{bind} is a one-way hash of the child witness key w_n.
Private witness: $(d_{n-1},\, w_{n-1},\, d_n,\, w_n)$, the parent and child derivation and witness keys.

(L-i) **Derivation step.** (d_n, w_n) is correctly derived from (d_{n-1}, w_{n-1}) and the public child index i_n according to the prescribed derivation step of the scheme.
(L-ii) **Public key.** pk_n is the public key corresponding to d_n.
(L-iii) **Hash binding.** $h_{\mathsf{bind}} = H(w_n)$, where H is a one-way function specified by the instantiation.

The soundness or unsoundness of a last-step proof reduces to whether the parent witness w_{n-1} appearing in condition (L-i) is genuinely private, that is, not recoverable from any public value by a QPT adversary. If w_{n-1} is private, an adversary cannot freely choose it to construct a fake derivation; if it is public or quantum-recoverable, the adversary can pick any w^*_{n-1} and satisfy (L-i) without any connection to the legitimate seed.

6.1 BIP32 secp256k1

For non-hardened keys, all inputs to the HMAC call at depth n are public: the parent chain code c_{n-1}, the parent public key K_{n-1}, and the child index i_n. For hardened keys, c_{n-1} is private: it is the right half of the HMAC output at depth $n-1$, whose computation required k_{n-2} and c_{n-2} as private inputs, and is not published in the node state.

Proposition 6.1 (Unsoundness of last-step proofs for BIP32 secp256k1, non-hardened)**.** *For BIP32 secp256k1 non-hardened keys, last-step proofs are unsound. There exists a PPT adversary that produces a passing last-step proof with no connection to any legitimate seed.*

Proof. The adversary computes $I_L \| I_R = \mathsf{HMAC\text{-}SHA512}(c_{n-1},\, \mathtt{ser}_P(K_{n-1}) \| \langle i_n \rangle_4)$ from public data, picks any $k^*_{n-1} \xleftarrow{\$} \mathbb{Z}_q$, sets $k^*_n = (k^*_{n-1} + I_L) \bmod q$, and produces a valid last-step proof certifying (k^*_{n-1}, k^*_n). The proof passes because the relation checks only the recurrence, not the origin of k^*_{n-1}. □

Proposition 6.2 (Soundness of last-step proofs for BIP32 secp256k1, hardened)**.** *For BIP32 secp256k1 hardened keys, last-step proofs are sound, assuming HMAC-SHA512 is a secure PRF against QPT adversaries (Definition 3.6) and* NIZK *satisfies zero-knowledge and simulation extractability (Section 3.6).*

Proof. Suppose for contradiction that there exists a PPT adversary $\mathcal{A}$ that produces a valid last-step derivation proof π^*_{deriv} for statement $(\mathsf{pk}_n, h^*_{\mathsf{bind}})$ with non-negligible probability ε, given only the public values $(K_0, \ldots, K_n)$.

Game G_0 (real game). The challenger runs honest key generation to produce $(K_0, \ldots, K_n)$ using $\mathsf{HMAC\text{-}SHA512}(c_{n-1}, \cdot)$ at the hardened step at depth n, and gives $(K_0, \ldots, K_n)$ to $\mathcal{A}$. $\mathcal{A}$ outputs π^*_{deriv} with $\mathsf{NIZKVerify}(\mathsf{pp}, (K_n, h^*_{\mathsf{bind}}), \pi^*_{\mathsf{deriv}}) = 1$ with probability ε.

Game G_1 (simulated NIZK proofs). Identical to G_0 except any NIZK proof produced by the challenger internally is replaced by a simulated proof from $\mathcal{S}$. By the zero-knowledge property of

NIZK, $|\Pr[\mathcal{A}\text{ wins }G_0] - \Pr[\mathcal{A}\text{ wins }G_1]| \le \mathsf{negl}(\lambda)$, so $\mathcal{A}$ wins G_1 with non-negligible probability $\varepsilon - \mathsf{negl}(\lambda)$.

Game G_2 (random function replacing HMAC). Identical to G_1 except $\mathsf{HMAC\text{-}SHA512}(c_{n-1}, \cdot)$ at depth n is replaced by a truly random function $f(\cdot)$, so (k_n, c_n) are uniformly random and independent of c_{n-1}.

We construct a PPT adversary $\mathcal{B}$ against the PRF security of HMAC-SHA512. $\mathcal{B}$ receives a challenge oracle $\mathcal{O}$ that is either $\mathsf{HMAC\text{-}SHA512}(c_{n-1}, \cdot)$ for a secret key c_{n-1}, or a truly random function $f(\cdot)$. $\mathcal{B}$ runs honest key generation up to depth $n-1$, uses $\mathcal{O}$ at depth n to produce (k_n, c_n), sets $K_n = [k_n]G_{\text{secp}}$, and gives $(K_0, \ldots, K_n)$ to $\mathcal{A}$. If $\mathcal{A}$ outputs a valid π^*_{deriv}, $\mathcal{B}$ outputs 1; otherwise $\mathcal{B}$ outputs 0. Indeed, when $\mathcal{O} = \mathsf{HMAC\text{-}SHA512}(c_{n-1}, \cdot)$ the simulation is G_1; when $\mathcal{O} = f(\cdot)$ the simulation is G_2. If $|\Pr[\mathcal{A}\text{ wins }G_1] - \Pr[\mathcal{A}\text{ wins }G_2]|$ is non-negligible, $\mathcal{B}$ breaks PRF security with the same advantage, a contradiction. Hence $\mathcal{A}$ wins G_2 with non-negligible probability.

Analysis of G_2. In G_2, (k_n, c_n) are uniformly random and independent of c_{n-1}. By simulation extractability of NIZK, a PPT extractor $\mathcal{E}$ extracts a witness (k^*_{n-1}, c^*_{n-1}) satisfying the hardened recurrence, so $\mathsf{HMAC\text{-}SHA512}(c^*_{n-1}, \texttt{0x00}\|\mathtt{ser}_{32}(k^*_{n-1})\|\langle i_n\rangle_4)$ must equal the challenger's (k_n, c_n). But in G_2, $(k_n, c_n) = f(\texttt{0x00}\|\mathtt{ser}_{32}(k^*_{n-1})\|\langle i_n\rangle_4)$ for a truly random f, so finding (k^*_{n-1}, c^*_{n-1}) consistent with (k_n, c_n) succeeds with probability at most $q \cdot 2^{-256}$, that is negligible. This contradicts that $\mathcal{A}$ wins G_2 with non-negligible probability.

□

6.2 SLIP-10/Ed25519 and BIP32-Ed25519

SLIP-10/Ed25519 and BIP32-Ed25519 derive each hardened step from an HMAC-SHA512 call keyed by the parent chain code, and each carries a private quantum-safe witness at hardened nodes: the accumulating right half k_{R_n} for BIP32-Ed25519, and the chain code c_n (the right half of the HMAC output, which becomes the next step's HMAC key) for SLIP-10/Ed25519. The quantum-safe witness analysis applies to both, with the corresponding witness in each case.

Proposition 6.3 (Unsoundness of last-step proofs for BIP32-Ed25519, non-hardened)**.** *For BIP32-Ed25519 non-hardened keys, last-step proofs are unsound. There exists a PPT adversary that produces a passing last-step proof with no connection to any legitimate seed.*

Proof. For a non-hardened step at depth n, the HMAC output is:

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

where c_{n-1}, A_{n-1}, and i_n are all public, so z_n is fully computable by any party. In particular $z_L^{(n)} = z_n[0:224]$ and $z_R^{(n)} = z_n[256:512]$ are both publicly known.

We construct a QPT adversary $\mathcal{A}$ as follows. $\mathcal{A}$ recovers $k_{L_{n-1}}$ from the public key $A_{n-1} = k_{L_{n-1}} \cdot B$ and k_{L_n} from A_n via Shor's algorithm. $\mathcal{A}$ then picks any $k^*_{R_{n-1}} \xleftarrow{\$} \{0,1\}^{256}$ independently of the legitimate $k_{R_{n-1}}$, sets $k^*_{R_n} = k^*_{R_{n-1}} + z_R^{(n)} \bmod 2^{256}$. $\mathcal{A}$ computes $h^*_{\mathsf{bind}} = \mathsf{SHA256}(k^*_{R_n})$ and produces a NIZK proof π^* for $\mathcal{R}_{\mathsf{last}}$ with public statement $(A_n, h^*_{\mathsf{bind}})$ and private witness $(k_{L_{n-1}}, k^*_{R_{n-1}}, k_{L_n}, k^*_{R_n})$.

The proof π^* verifies because all relation conditions (L-i)–(L-iii) are satisfied: condition (L-i) holds since both scalar recurrences use the correct public tweaks $z_L^{(n)}$ and $z_R^{(n)}$ applied to the honestly recovered $k_{L_{n-1}}$ and the freely chosen $k^*_{R_{n-1}}$; condition (L-ii) holds since $k_{L_n} \cdot B = (k_{L_{n-1}} + 8 \cdot z_L^{(n)}) \cdot B = A_n$ by the honest non-hardened public key derivation formula; condition (L-iii) holds by construction of h^*_{bind}. The witness value $k^*_{R_{n-1}}$ is chosen uniformly at random and is

independent of the legitimate $k_{R_{n-1}}$ with overwhelming probability $1 - 2^{-256}$, so the proof has no connection to the legitimate seed. □

Proposition 6.4 (Soundness of last-step proofs for BIP32-Ed25519 and SLIP-10/Ed25519, hardened)**.** *For BIP32-Ed25519 and SLIP-10/Ed25519 hardened keys, last-step proofs are sound, assuming HMAC-SHA512 is a secure PRF against QPT adversaries (Definition 3.6) and* NIZK *satisfies zero-knowledge and simulation extractability (Section 3.6).*

Proof. The proof is identical in structure to that of Proposition 6.2, applied to each scheme's hardened HMAC-SHA512 call keyed by c_{n-1}. For BIP32-Ed25519 the PRF input is $\texttt{0x00} \| k_{L_{n-1}} \| k_{R_{n-1}} \| \langle i_n \rangle_4$ and the quantum-safe witness is the accumulated right half k_{R_n}; for SLIP-10/Ed25519 the PRF input is $\texttt{0x00} \| k_{L_{n-1}} \| \langle i_n \rangle_4$, the output is parsed as (k_{L_n}, c_n), and the quantum-safe witness is the chain code c_n. In G_2 the HMAC-SHA512 output is replaced by a truly random f, so the extractor recovering a witness satisfying the hardened recurrence must invert a random function to reproduce the challenger's output, which succeeds with probability at most $q \cdot 2^{-256}$, that is negligible. □

6.3 Motivation for QBIP32

The analysis above shows that all three existing schemes admit sound last-step proofs for hardened keys and unsound last-step proofs for non-hardened keys. This asymmetry is structural: for hardened derivation the HMAC key is a private chain code that is computationally bound to the seed and not recoverable by any QPT adversary, while for non-hardened derivation all HMAC inputs are public and the quantum-safe witness degenerates. The limitation on non-hardened keys is therefore inherent and cannot be resolved by modifying the proof system alone.

The remaining limitation of existing schemes is a lack of uniformity: each scheme has its own derivation structure, and the quantum-safe witness appears as an incidental product of that structure rather than as an explicit design choice. In particular, there is no single derivation standard that works across all prime-order elliptic curves and guarantees a quantum-safe witness by construction at every hardened node.

In Section 7 we introduce QBIP32, a unified key derivation scheme based on a keyed extensible-output function instantiated with KMAC256 (a sponge-based construction defined in NIST SP 800-185 [Nat16]). The quantum-safe witness is a first-class output of the derivation function rather than an artifact of a particular hash structure, and the same derivation logic applies uniformly to secp256k1, Ed25519, and any other curve used in blockchain infrastructure. The soundness and unsoundness of last-step proofs for QBIP32 follow the same pattern as the existing schemes and are stated formally in Section 7.

Table 1: Sufficiency of last-step proof by scheme and key type. QBIP32 is defined in Section 7.

Scheme	**Key type**	**Last-step only?**	**Reason**
BIP32 secp256k1	Non-hardened	No	All values publicly computable (Prop. 6.1)
BIP32 secp256k1	Hardened	Yes	c_{n-1} private, PRF-bound to seed (Prop. 6.2)
SLIP-10/Ed25519	Hardened	Yes	c_{n-1} private, PRF-bound to seed (Prop. 6.4)
BIP32-Ed25519	Non-hardened	No	$z_R^{(n)}$ publicly computable (Prop. 6.3)
BIP32-Ed25519	Hardened	Yes	$k_{R_{n-1}}$ private, PRF-bound to seed (Prop. 6.4)
QBIP32	Non-hardened	No	TW publicly computable (Section 7)
QBIP32	Hardened	Yes	NW_{par} private, PRF-bound to seed (Section 7)

7 QBIP32: A Universal Sponge-Based HD Wallet

7.1 The HASH768 Primitive

As defined in Section 3.5, $\mathsf{HASH768} : \{0,1\}^{\lambda} \times \{0,1\}^{*} \to \{0,1\}^{768}$ is a keyed extensible-output function with 768-bit output, instantiated with KMAC256 (a sponge-based construction defined in NIST SP 800-185 [Nat16]), parsed as three consecutive 256-bit blocks:

$$\mathsf{HASH768}(k, d) \;=\; (T_S \| T_W \| T_C), \qquad T_S, T_W, T_C \in \{0,1\}^{256}.$$

Alternative instantiations using BLAKE3 or Poseidon2 are discussed in Remark 3.8.

We note one structural advantage of QBIP32 over BIP32-Ed25519 independent of the chosen hash function: BIP32-Ed25519 requires two HMAC calls per derivation step (one for the child key, one for the child chain code), whereas QBIP32 requires a single HASH768 call that produces scalar, witness, and chain code simultaneously. This halves the number of hash invocations per derivation step regardless of the instantiation. Whether this reduction in primitive calls yields a corresponding reduction in proving time is backend-dependent: our RISC Zero benchmarks (Section 9.2) show no measurable advantage, since fixed per-segment STARK cost dominates over the exact hash call count.

7.2 Node Structure and Key Derivation

Each QBIP32 node holds three 256-bit values: NS (signing scalar, private), NW (quantum-safe witness, private, never published), NC (chain code, public). In the HDW framework of Section 3.8, the derivation key is $d_i \coloneqq NS_i$, the witness key is $w_i \coloneqq NW_i$, and the signing key is $\mathsf{sk}_i \coloneqq (NS_i, NW_i)$. The public key at each node is $K_i = NS_i \cdot G$, where G is the generator of the underlying curve.

Root generation. Given seed $S \in \{0,1\}^{256}$, let T be the domain-separation string `"QBIP-32: secp256k1"` or `"QBIP-32: ed25519"` depending on the underlying curve. Compute:

$$(NS_0,\, NW_0,\, NC_0) \;\coloneqq\; \mathsf{HASH768}\big(S,\, T\big).$$

Resampling. The root scalar NS_0 must satisfy the acceptance criteria of the underlying curve: for secp256k1, $0 < NS_0 < q$ where q is the curve order; for Ed25519, bit 253 of NS_0 (in little-endian) must be zero, and the BIP32-Ed25519 bit tweaks (clear bits 0, 1, 2, 255; set bit 254) are applied. If the criteria are not satisfied, replace S with the full 768-bit node $NS_0 \| NW_0 \| NC_0$ and repeat the HASH768 call with the same T. This loop terminates within a bounded number of iterations with overwhelming probability.

Hardened child derivation ($x \geq 2^{31}$). Given parent node $(NS_{\mathsf{par}}, NW_{\mathsf{par}}, NC_{\mathsf{par}})$ and hardened index x, compute:

$$(T_S,\, T_W,\, T_C) \;\coloneqq\; \mathsf{HASH768}\big(NC_{\mathsf{par}},\; NS_{\mathsf{par}} \,\|\, NW_{\mathsf{par}} \,\|\, \langle x \rangle_4\big),$$

where $\langle x \rangle_4$ is the 4-byte big-endian encoding of x. Both NS_{par} and NW_{par} are private inputs; the child tweak is therefore not publicly computable. If the resulting tweak fails the acceptance criteria, replace the tweak $(T_S \| T_W \| T_C)$ with $\mathsf{HASH768}(NC_{\mathsf{par}},\; T_S \| T_W \| T_C)$ and retry, keeping NC_{par} as the key.

Non-hardened child derivation ($x < 2^{31}$). Given parent node and non-hardened index x, let $K_{\mathsf{par}} = NS_{\mathsf{par}} \cdot G$ be the parent public key. Compute:

$$(T_S,\, T_W,\, T_C) \;\coloneqq\; \mathsf{HASH768}\big(NC_{\mathsf{par}},\; K_{\mathsf{par}} \,\|\, \langle x \rangle_4\big).$$

Since K_{par} and NC_{par} are both public, the tweak is publicly computable by any party holding the parent public parameters. If the resulting tweak fails the acceptance criteria, replace the tweak $(T_S \| T_W \| T_C)$ with $\mathsf{HASH768}(NC_{\mathsf{par}},\; T_S \| T_W \| T_C)$ and retry, as in the hardened case.

Child node update. Given tweak values (T_S, T_W, T_C):

$$\begin{aligned}
NS_{\mathsf{child}} &\coloneqq \begin{cases} (NS_{\mathsf{par}} + T_S) \bmod q & \text{(secp256k1)}, \\ NS_{\mathsf{par}} + 8 \cdot T_{S,28} & \text{(Ed25519, plain integer addition)}, \end{cases} \\
NW_{\mathsf{child}} &\coloneqq (NW_{\mathsf{par}} + T_W) \bmod 2^{256}, \\
NC_{\mathsf{child}} &\coloneqq T_C,
\end{aligned}$$

where $T_{S,28}$ denotes the value obtained by retaining only the 28 least significant bytes of T_S and zeroing the rest, following the BIP32-Ed25519 scalar truncation convention. The child public key is $K_{\mathsf{child}} = NS_{\mathsf{child}} \cdot G$.

Instantiation of the HDW algorithms.

- $\mathsf{Set}(1^\lambda, T, S)$: runs root generation and iterative child derivation along T to produce, for every $v_i \in V$: derivation key $d_i \coloneqq NS_i$, witness key $w_i \coloneqq NW_i$, signing key $\mathsf{sk}_i \coloneqq (NS_i, NW_i)$, and node state $\mathsf{st}_i \coloneqq (K_i, NC_i, i_i)$. Outputs $\mathsf{pp} \coloneqq K_0$, st_0, and $\{\mathsf{sk}_i\}_{v_i \in V}$.
- $\mathsf{DPriv}(\mathsf{pp}, d_i, w_i, \mathsf{st}_i, v_i, v_j)$: applies the non-hardened child derivation step; returns $d_j \coloneqq NS_j$, $w_j \coloneqq NW_j$, st_j.
- $\mathsf{DPriv}^*(\mathsf{pp}, d_i, w_i, \mathsf{st}_i, v_i, v_j)$: applies the hardened child derivation step; returns $d_j \coloneqq NS_j$, $w_j \coloneqq NW_j$, st_j.
- $\mathsf{DPub}(\mathsf{pp}, \mathsf{st}_i, v_i, v_j)$: applies non-hardened child derivation iteratively using only public data; returns K_j and st_j.
- $\mathsf{DPub}^*(\mathsf{pp}, d_i, w_i, \mathsf{st}_i, v_i, v_j)$: applies the hardened child derivation step; returns K_j and st_j.
- $\mathsf{Sign}(\mathsf{sk}_n, M)$: parse $\mathsf{sk}_n = (NS_n, NW_n)$; generates the signing proof π_{sign} as described in Section 7.3.
- $\mathsf{DCert}(\mathsf{pp}, d_h, w_h, \mathsf{st}_h, v_h, v_n, a, F)$: in classical mode, $F = \varepsilon$ and the certificate is empty; in ZK-PoSP mode, a is the anchor witness (the seed S for full-chain proofs, or $(NS_{\mathsf{par}}, NW_{\mathsf{par}}, NC_{\mathsf{par}})$ of the parent of the hardened anchor for pruned proofs), and F is invoked with a and the derivation chain to produce the certificate τ_n.
- $\mathsf{DVerify}(\mathsf{pp}, K_n, \tau_n, F_{\mathsf{ver}})$: in classical mode, F_{ver} always returns 1; in ZKPoSP mode, invokes $F_{\mathsf{ver}}(\mathsf{pp}, K_n, \tau_n)$.
- $\mathsf{SignVerify}(\mathsf{pp}, K_n, M, \sigma)$: verifies the signing proof σ for message M under K_n, as described in Section 7.3.
- $\mathsf{Verify}(\mathsf{pp}, K_n, \tau_n, M, \sigma, F_{\mathsf{ver}})$: in classical mode, returns $\mathsf{SignVerify}(\mathsf{pp}, K_n, M, \sigma)$; in ZKPoSP mode, returns 1 iff $\mathsf{DVerify}(\mathsf{pp}, K_n, \tau_n, F_{\mathsf{ver}}) = 1$ and $\mathsf{SignVerify}(\mathsf{pp}, K_n, M, \sigma) = 1$.

Remark 7.1 (Curve generality of QBIP32). QBIP32 is defined for any elliptic curve group $\mathbb{G}$ of prime order with a generator G. The derivation function, the quantum-safe witness NW, the chain code NC, and all proof relations in Section 7.3 are independent of the curve. The only curve-specific component is the scalar update rule for NS_{child}, which in general takes the form $NS_{\mathsf{child}} \coloneqq f(NS_{\mathsf{par}}, T_S)$ for a curve-dependent function f; the two instantiations shown above for

secp256k1 and Ed25519 are concrete examples of this pattern. Any other prime-order curve can be supported by defining the appropriate f without changing any other part of the derivation or proof system. Notice that no step of any proof in this section uses an algebraic property of f itself. Consequently, QBIP32's quantum-safe-witness and unforgeability guarantees hold for any deterministic f, and in particular do not depend on which curve-specific mechanism f uses. Extending QBIP32 to a new curve therefore reduces to correctly reuse that curve's own known-secure scalar update rule as f; QBIP32 neither strengthens nor weakens whatever guarantees that update rule already carries on its own.

7.3 NP Relations for Full Hardened Path Derivations

We define the proof relations for QBIP32 derivation paths consisting only of hardened steps, and instantiate the HDW algorithms of Section 3.8; the pruned case (a hardened prefix followed by a non-hardened suffix) is treated in Section 7.4. In both cases, the hash binding linking the derivation certificate to the signing proof uses $h_{nw} = \mathsf{SHA256}(NW_n)$ in place of $h_{k_R} = \mathsf{SHA256}(k_{R_n})$ used for BIP32-Ed25519. The structural role of the two values is identical: both are one-way hashes of a private value that is computationally bound to the seed and not recoverable from the public key by a QPT adversary.

Hardened-only derivation relation $\mathcal{R}^{\mathsf{hard}}_{\mathsf{deriv}}$. Only the last hardened step is proved. The witness starts from the parent of the target node.

$$\mathcal{R}^{\mathsf{hard}}_{\mathsf{deriv}} \coloneqq \left\{ \big((K_n,\ h_{nw}),\ (NS_{\mathsf{par}},\ NW_{\mathsf{par}},\ NC_{\mathsf{par}},\ NS_n,\ NW_n,\ x) \big) \;\middle|\; \text{(Q-i)–(Q-iv) hold} \right\}.$$

Public statement: $(K_n,\ h_{nw}) \in \mathbb{G} \times \{0,1\}^{256}$.
Private witness: $(NS_{\mathsf{par}},\ NW_{\mathsf{par}},\ NC_{\mathsf{par}},\ NS_n,\ NW_n,\ x)$.

(Q-i) **Hardened step.** $(T_S \| T_W \| T_C) = \mathsf{HASH768}(NC_{\mathsf{par}},\ NS_{\mathsf{par}} \| NW_{\mathsf{par}} \| \langle x \rangle_4)$.
(Q-ii) **Scalar update.** $NS_n = f(NS_{\mathsf{par}}, T_S), \quad NW_n = NW_{\mathsf{par}} + T_W \bmod 2^{256}$.
(Q-iii) **Public key.** $K_n = NS_n \cdot G$.
(Q-iv) **Hash binding.** $h_{nw} = \mathsf{SHA256}(NW_n)$.

Hardened-only signing relation $\mathcal{R}^{\mathsf{hard}}_{\mathsf{sign}}$.

$$\mathcal{R}^{\mathsf{hard}}_{\mathsf{sign}} \coloneqq \left\{ \big((M,\ h_{nw}),\ (NW_n,\ h) \big) \;\middle|\; h = \mathsf{SHA256}(M \| NW_n),\ h_{nw} = \mathsf{SHA256}(NW_n) \right\}.$$

Public statement: $(M,\ h_{nw})$.
Private witness: $(NW_n,\ h)$.

Hardened-only HDW instantiation.

$\mathsf{DCert}(\mathsf{pp}, d_h, w_h, \mathsf{st}_h, v_h, v_n, a, F)$: parse $a = (NS_{\mathsf{par}}, NW_{\mathsf{par}}, NC_{\mathsf{par}})$. Apply DPriv^* from v_h to v_n to obtain (NS_n, NW_n). Compute $h_{nw} \coloneqq \mathsf{SHA256}(NW_n)$ and $\pi_{\mathsf{deriv}} \leftarrow \mathsf{NIZKProve}(\mathsf{pp}, (K_n, h_{nw}), \mathbf{w}_{\mathsf{hard}})$ under $\mathcal{R}^{\mathsf{hard}}_{\mathsf{deriv}}$. Output $\tau_n \coloneqq (\pi_{\mathsf{deriv}}, h_{nw})$.

$\mathsf{DVerify}(\mathsf{pp}, K_n, \tau_n, F_{\mathsf{ver}})$: parse $\tau_n = (\pi_{\mathsf{deriv}}, h_{nw})$. Output 1 iff $\mathsf{NIZKVerify}(\mathsf{pp}, (K_n, h_{nw}), \pi_{\mathsf{deriv}}) = 1$.

$\mathsf{Sign}(\mathsf{sk}_n, M)$: parse $\mathsf{sk}_n = (NS_n, NW_n)$. Compute $h_{nw} \coloneqq \mathsf{SHA256}(NW_n)$, $h \coloneqq \mathsf{SHA256}(M \| NW_n)$, and $\pi_{\mathsf{sign}} \leftarrow \mathsf{NIZKProve}(\mathsf{pp}, (M, h_{nw}), (NW_n, h))$ under $\mathcal{R}^{\mathsf{hard}}_{\mathsf{sign}}$. Output $\sigma \coloneqq (\pi_{\mathsf{sign}}, h_{nw})$.

$\mathsf{SignVerify}(\mathsf{pp}, K_n, M, \sigma)$: parse $\sigma = (\pi_{\mathsf{sign}}, h_{nw})$. Output 1 iff $\mathsf{NIZKVerify}(\mathsf{pp}, (M, h_{nw}), \pi_{\mathsf{sign}}) = 1$.

$\mathsf{Verify}(\mathsf{pp}, K_n, \tau_n, M, \sigma, F_{\mathsf{ver}})$: parse $\tau_n = (\pi_{\mathsf{deriv}}, h_{nw})$ and $\sigma = (\pi_{\mathsf{sign}}, h'_{nw})$. Output 1 iff $\mathsf{DVerify}(\mathsf{pp}, K_n, \tau_n, F_{\mathsf{ver}}) = 1$, $h_{nw} = h'_{nw}$, and $\mathsf{SignVerify}(\mathsf{pp}, K_n, M, \sigma) = 1$.

Last-Step Proof Properties

The soundness and unsoundness results for last-step proofs of QBIP32 announced in Table 1 are stated here, now that both relations are available.

Proposition 7.2 (Unsoundness of last-step proofs for QBIP32, non-hardened)**.** *For QBIP32 non-hardened keys, last-step proofs are unsound. There exists a QPT adversary that produces a passing last-step proof with no connection to any legitimate seed.*

Proof. The proof is analogous to that of Proposition 6.3. For a non-hardened step, NC_{par} and K_{par} are both public, so $(T_S, T_W, T_C) = \mathsf{HASH768}(NC_{\mathsf{par}}, K_{\mathsf{par}} \| \langle x \rangle_4)$ is publicly computable, and in particular $T_W = NW_{\mathsf{child}} - NW_{\mathsf{par}} \bmod 2^{256}$ is known. The adversary recovers NS_{par} from K_{par} via Shor's algorithm, picks any $NW^*_{\mathsf{par}} \xleftarrow{\$} \{0,1\}^{256}$, sets $NW^*_{\mathsf{child}} = NW^*_{\mathsf{par}} + T_W \bmod 2^{256}$, and derives $NS^*_{\mathsf{child}} = f(NS_{\mathsf{par}}, T_S)$ from the public T_S. The resulting proof certifies $(NS_{\mathsf{par}}, NW^*_{\mathsf{par}}, NS^*_{\mathsf{child}}, NW^*_{\mathsf{child}})$ with NW^*_{par} independent of the legitimate NW_{par} with overwhelming probability. □

Proposition 7.3 (Soundness of last-step proofs for QBIP32, hardened)**.** *For QBIP32 hardened keys, last-step proofs are sound, assuming* HASH768 *is a secure PRF against QPT adversaries (Definition 3.6) and* NIZK *satisfies zero-knowledge and simulation extractability (Section 3.6).*

Proof. The proof follows the same G_0, G_1, G_2 structure as Proposition 6.2, with the following substitutions: the PRF is HASH768 keyed by NC_{par}, the PRF input is $NS_{\mathsf{par}} \| NW_{\mathsf{par}} \| \langle x \rangle_4$, and the output is $(NS_{\mathsf{child}}, NW_{\mathsf{child}}, NC_{\mathsf{child}})$ as in conditions (Q-i)–(Q-ii). In G_2 the output of HASH768 is replaced by a truly random function, so the extractor's witness $(NS^*_{\mathsf{par}}, NW^*_{\mathsf{par}})$ must invert a random function to reproduce the challenger's output, which succeeds with probability at most $q \cdot 2^{-256}$, negligible. □

Theorem 7.4 (EUF-CMA of (Sign, SignVerify) for QBIP32 hardened-only, sketch)**.** *Under the same assumptions as Proposition 7.3,* (Sign, SignVerify) *for QBIP32 hardened-only is EUF-CMA secure (Definition 3.10) against PPT adversaries.*

Proof sketch. The proof follows the same G_0, G_1, G_2 structure as Theorem 5.2, with HASH768 replacing HMAC-SHA512 as the PRF and h_{nw} replacing h_{k_R} as the hash binding. In G_2, h_{nw} is the output of a truly random function and is independent of NW_n. By simulation extractability, any forgery yields a witness (NW^*_n, h^*) satisfying conditions (Q-iv) and $\mathcal{R}^{\mathsf{hard}}_{\mathsf{sign}}$, but since h_{nw} is random and independent of any value the adversary can compute, this succeeds with probability at most $q \cdot 2^{-256}$, negligible. □

Theorem 7.5 (Hierarchical EUF-CMA of QBIP32 hardened-only, sketch)**.** *Under the same assumptions as Proposition 7.3, QBIP32 hardened-only is hierarchically EUF-CMA secure (Definition 3.12) against PPT adversaries.*

Proof sketch. The proof follows the same structure as Theorem 5.4. Since $v^* \notin Q_{\mathrm{Corr}}$ and $\mathcal{O}_{\mathrm{Corr}}$ returns only $\mathsf{sk}_i = (NS_i, NW_i)$, the anchor witness $a = (NS_{\mathsf{par}}, NW_{\mathsf{par}}, NC_{\mathsf{par}})$ is never revealed. After replacing honest proofs with simulated ones ($G_0 \to G_1$), simulation extractability applied to π^*_{deriv} yields a witness satisfying (Q-i)–(Q-iv). By Proposition 7.3, any valid extracted witness is computationally bound to the legitimate $(NS_{\mathsf{par}}, NW_{\mathsf{par}})$, so h^*_{nw} is bound to the legitimate NW_n. The signing forgery under the pinned h^*_{nw} is then ruled out by Theorem 7.4. □

7.4 Pruned NP Relations

The pruned relations cover derivation paths containing a hardened prefix followed by a non-hardened suffix. By Proposition 7.3, the hardened prefix does not need to be re-proved: the proof starts from the parent of the last hardened node (the anchor) and covers one hardened step plus the non-hardened suffix. Throughout this subsection we consider BIP44 paths of the form $m/44'/coin'/account'/change/index$ (three hardened levels followed by two non-hardened levels), which is the dominant convention in blockchain infrastructure; the same relations apply to any path typology consisting of a hardened prefix and a non-hardened suffix, with the anchor placed at the last hardened node. For BIP44 the anchor sits at depth 3 and the proof covers one hardened step plus $k = 2$ non-hardened steps: a fixed, small circuit whose size is independent of the number of hardened levels above the anchor.

Pruned derivation relation $\mathcal{R}^{\mathsf{pruned}}_{\mathsf{deriv}}$. The derivation proof covers the hardened anchor step (index x_h) plus $k \geq 1$ subsequent non-hardened steps (indices $x_1, \ldots, x_k$). The witness starts from the parent of the hardened anchor node; all hardened steps before the anchor are not re-proved inside the NIZK. Let $h_{cw} = \mathsf{SHA256}(NC_{\mathsf{leaf}} \| NW_{\mathsf{leaf}})$.

$$\mathcal{R}^{\mathsf{pruned}}_{\mathsf{deriv}} \coloneqq \left\{ \left((K_{\mathsf{leaf}},\ h_{cw}),\ \mathbf{w}_{\mathsf{pruned}} \right) \;\middle|\; \text{(P-i)–(P-iv) hold} \right\},$$

where $\mathbf{w}_{\mathsf{pruned}} = (NS_{\mathsf{par}},\ NW_{\mathsf{par}},\ NC_{\mathsf{par}},\ NS_0,\ NW_0,\ NC_0,\ x_h,\ \{K_{j-1},\ x_j,\ NS_j,\ NW_j,\ NC_j\}_{j=1}^{k})$.

Public statement: $(K_{\mathsf{leaf}},\ h_{cw}) \in \mathbb{G} \times \{0,1\}^{256}$.

(P-i) **Hardened anchor step.** $(T_S \| T_W \| T_C) = \mathsf{HASH768}(NC_{\mathsf{par}},\ NS_{\mathsf{par}} \| NW_{\mathsf{par}} \| \langle x_h \rangle_4)$, $NS_0 = f(NS_{\mathsf{par}}, T_S)$, $NW_0 = NW_{\mathsf{par}} + T_W$, $NC_0 = T_C$.

(P-ii) **Non-hardened steps** $j = 1, \ldots, k$. $K_{j-1} = NS_{j-1} \cdot G$; $(T_S \| T_W \| T_C) = \mathsf{HASH768}(NC_{j-1},\ K_{j-1} \| \langle x_j \rangle_4)$; $NS_j = f(NS_{j-1}, T_S)$, $NW_j = NW_{j-1} + T_W$, $NC_j = T_C$.

(P-iii) **Leaf public key.** $K_{\mathsf{leaf}} = NS_k \cdot G$.

(P-iv) **Hash binding.** $h_{cw} = \mathsf{SHA256}(NC_k \| NW_k)$.

Remark 7.6 (Hash binding in the pruned case). In the hardened-only case the hash binding uses only NW_n, since this is the sole quantum-safe witness value that needs to be linked across the two proofs. In the pruned case the binding uses both NC_{leaf} and NW_{leaf}, because the non-hardened suffix depends on the chain code and the signing proof must be bound to the specific leaf reached by that suffix. Combining both values in a single SHA-256 call prevents an adversary from substituting a different non-hardened path that happens to produce the same NW_{leaf}.

Pruned signing relation $\mathcal{R}^{\mathsf{pruned}}_{\mathsf{sign}}$.

$$\mathcal{R}^{\mathsf{pruned}}_{\mathsf{sign}} \coloneqq \left\{ \left((M,\ h_{cw}),\ (NW_{\mathsf{leaf}},\ NC_{\mathsf{leaf}},\ h) \right) \;\middle|\; \text{(PS-i)–(PS-ii) hold} \right\}.$$

Public statement: $(M,\ h_{cw})$.
Private witness: $(NW_{\mathsf{leaf}},\ NC_{\mathsf{leaf}},\ h)$.

(PS-i) **Hash binding.** $h_{cw} = \mathsf{SHA256}(NC_{\mathsf{leaf}} \| NW_{\mathsf{leaf}})$.

(PS-ii) **Signature hash.** $h = \mathsf{SHA256}(M \| NW_{\mathsf{leaf}})$.

Pruned HDW instantiation. The DCert, $\mathsf{DVerify}$, Sign, $\mathsf{SignVerify}$, and Verify algorithms are identical in structure to the hardened-only case, with h_{nw} replaced by h_{cw}, $\mathcal{R}^{\mathsf{hard}}_{\mathsf{deriv}}$ replaced by $\mathcal{R}^{\mathsf{pruned}}_{\mathsf{deriv}}$, $\mathcal{R}^{\mathsf{hard}}_{\mathsf{sign}}$ replaced by $\mathcal{R}^{\mathsf{pruned}}_{\mathsf{sign}}$, and $\mathbf{w}_{\mathsf{hard}}$ replaced by $\mathbf{w}_{\mathsf{pruned}}$ in the DCert call.

Theorem 7.7 (EUF-CMA of (Sign, SignVerify) for QBIP32 pruned, sketch)**.** *Under the same assumptions as Proposition 7.3,* (Sign, SignVerify) *for QBIP32 pruned is EUF-CMA secure (Definition 3.10) against PPT adversaries.*

Proof sketch. The proof follows the same structure as Theorem 7.4, with h_{cw} replacing h_{nw} as the hash binding and $\mathcal{R}_{\mathsf{sign}}^{\mathsf{pruned}}$ replacing $\mathcal{R}_{\mathsf{sign}}^{\mathsf{hard}}$. □

Theorem 7.8 (Hierarchical EUF-CMA of QBIP32 pruned, sketch)**.** *Under the same assumptions as Proposition 7.3, QBIP32 pruned is hierarchically EUF-CMA secure (Definition 3.12) against PPT adversaries.*

Proof sketch. The proof follows the same structure as Theorem 7.5. □

Proposition 7.9 (Seed deletion)**.** *The pruned construction requires retaining only the parent of the anchor node and the anchor node itself (approximately 192 bytes total). The root seed may be securely deleted after generating and storing these two nodes.*

Proof. All values at the non-hardened levels below the anchor are recomputable from the anchor node and public indices via the non-hardened child derivation procedure of Section 7.2. The root seed appears in neither of the two relations $\mathcal{R}_{\mathsf{deriv}}^{\mathsf{pruned}}$ and $\mathcal{R}_{\mathsf{sign}}^{\mathsf{pruned}}$: the derivation witness $\mathbf{w}_{\mathsf{pruned}}$ starts from the parent of the anchor node, and the signing witness contains only leaf values. □

7.5 Full-Path Relation with Seed Linkability

The pruned relation deliberately discards the connection to the root seed. In some applications the opposite is desired: proving that a leaf key was derived from a specific seed, for example to establish linkability among all keys generated from the same wallet. The full-path relation covers the entire derivation from the seed to the leaf (a hardened prefix of length p followed by a non-hardened suffix of length t) and adds a commitment to the seed in the public statement. The commitment is computed once at wallet setup as $c_{\mathsf{seed}} = \mathsf{SHA256}(r \| seed)$, where the randomness $r \xleftarrow{\$} \{0,1\}^{256}$ is sampled once and reused in every full-path proof for that seed. Reusing r is what provides linkability: all proofs for keys derived from the same seed carry the same commitment value c_{seed}, so a verifier can check that two leaves belong to the same wallet without learning the seed, while hiding holds because r is uniform and never revealed.

Full-path derivation relation $\mathcal{R}_{\mathsf{deriv}}^{\mathsf{full\text{-}path}}$.

$$\mathcal{R}_{\mathsf{deriv}}^{\mathsf{full\text{-}path}} := \left\{ \left((p,\, t,\, K_n,\, h_{nw},\, c_{\mathsf{seed}}),\ \mathbf{w}_{\mathsf{full\text{-}path}} \right) \;\middle|\; \text{(F-i)–(F-vi) hold} \right\},$$

where $\mathbf{w}_{\mathsf{full\text{-}path}} = (seed,\, r,\, \{x_i\}_{i=1}^{p+t},\, \{NS_i,\, NW_i,\, NC_i\}_{i=0}^{p+t})$.

Public statement: $(p,\, t,\, K_n,\, h_{nw},\, c_{\mathsf{seed}})$, where p is the number of hardened steps, t the number of non-hardened steps, $K_n \in \mathbb{G}$ the leaf public key, $h_{nw} \in \{0,1\}^{256}$ the hash binding, and $c_{\mathsf{seed}} \in \{0,1\}^{256}$ the seed commitment.

(F-i) **Commitment opening.** $c_{\mathsf{seed}} = \mathsf{SHA256}(r \| seed)$.

(F-ii) **Root generation.** $(NS_0 \| NW_0 \| NC_0) = \mathsf{HASH768}(seed,\, T)$, applying the resampling rule of Section 7.2 until NS_0 satisfies the acceptance criteria of the underlying curve.

(F-iii) **Hardened steps** $i = 1, \ldots, p$**.** $(T_S \| T_W \| T_C) = \mathsf{HASH768}(NC_{i-1},\ NS_{i-1} \| NW_{i-1} \| \langle x_i \rangle_4)$; $NS_i = f(NS_{i-1}, T_S)$, $\quad NW_i = NW_{i-1} + T_W \bmod 2^{256}$, $\quad NC_i = T_C$.

(F-iv) **Non-hardened steps** $i = p{+}1, \ldots, p{+}t$**.** $K_{i-1} = NS_{i-1} \cdot G$; $(T_S \| T_W \| T_C) = \mathsf{HASH768}(NC_{i-1}, K_{i-1} \| \langle x_i \rangle_4)$; $NS_i = f(NS_{i-1}, T_S)$, $NW_i = NW_{i-1} + T_W \bmod 2^{256}$, $NC_i = T_C$.

(F-v) **Leaf public key.** $K_n = NS_{p+t} \cdot G$.

(F-vi) **Hash binding.** $h_{nw} = \mathsf{SHA256}(NW_{p+t})$.

The hash binding uses $h_{nw} = \mathsf{SHA256}(NW_{p+t})$ as in the hardened-only case, so the signing relation is $\mathcal{R}^{\mathsf{hard}}_{\mathsf{sign}}$ unchanged, and the HDW instantiation is identical to the hardened-only case with $\mathcal{R}^{\mathsf{full\text{-}path}}_{\mathsf{deriv}}$ in place of $\mathcal{R}^{\mathsf{hard}}_{\mathsf{deriv}}$, the anchor witness $a = (seed, r)$, and the tuple $(p, t, c_{\mathsf{seed}})$ added to the public statement of the derivation certificate.

Remark 7.10 (Linkability anchored at a hardened ancestor). The same construction applies with any hardened ancestor in place of the seed: the commitment becomes $c_{\mathsf{anc}} = \mathsf{SHA256}(r \| NS_{\mathsf{anc}} \| NW_{\mathsf{anc}} \| NC_{\mathsf{anc}})$ and condition (F-ii) is replaced by starting the derivation at that node. This links all keys in the subtree rooted at the chosen ancestor, rather than all keys of the wallet.

8 Post-Q-Day Variant

The schemes described so far are already secure against quantum adversaries: the witness NW is computationally hidden, and signing security does not depend on the hardness of the discrete logarithm problem. However, under the current threat model the leaf scalar NS_{leaf} must be kept secret, because a classical adversary who obtains it can forge signatures directly without needing a quantum computer. Once Q-day arrives, this classical concern is eliminated by a change in the global signing model: any signature produced by a scheme that is not quantum-secure is rejected by the network, regardless of its classical validity. Under this post-Q-day model, exposing NS_{leaf} carries no additional risk, since an adversary who learns it cannot produce an accepted signature outside QBIP32 anyway. This model shift enables a significant reduction in prover workload. The verifier recomputes $K_i = NS_i \cdot G$ for each level externally; this operation is more efficient than checking the multiplication inside the NIZK and adds no measurable overhead to verification time. The same Q-day simplification applies uniformly to every hardened-anchor scheme in this paper: SLIP-10/Ed25519, BIP32-Ed25519, BIP32-secp256k1, and QBIP32. In each case, once a quantum computer capable of running Shor's algorithm is available, the scalar multiplication $K_{\mathsf{leaf}} = NS_{\mathsf{leaf}} \cdot G$ can be removed from the derivation proof and moved to a classical verifier check. We write NS_{leaf} throughout for the leaf signing scalar; the concrete instantiation is $k_{L,\mathsf{leaf}}$ for BIP32-Ed25519, k_{leaf} for BIP32-secp256k1, the SLIP-10 secret key at the hardened leaf for SLIP-10/Ed25519, and NS_{leaf} for QBIP32. The generator is G for secp256k1 and B for Ed25519 curves.

Post-Q-day derivation relation $\mathcal{R}^{\mathsf{qday}}_{\mathsf{deriv}}$. This is identical to $\mathcal{R}^{\mathsf{pruned}}_{\mathsf{deriv}}$ or $\mathcal{R}^{\mathsf{full\text{-}path}}_{\mathsf{deriv}}$ except that condition (P-iii) (or (F-v) in the full-path case) is removed from the circuit and replaced by a classical check: the verifier computes $K_{\mathsf{leaf}} = NS_{\mathsf{leaf}} \cdot G$ outside the NIZK after extracting NS_{leaf} from the public statement. The same transformation applies to the analogous split relations of BIP32-Ed25519, BIP32-secp256k1, and SLIP-10/Ed25519: in every case the leaf public-key check is the only scalar-multiplication constraint in the derivation relation, and moving the leaf scalar into the public statement removes it. **Modified public statement.** For the pruned relation, $(K_{\mathsf{leaf}}, NS_{\mathsf{leaf}}, h_{cw}) \in \mathbb{G} \times \{0,1\}^{256} \times \{0,1\}^{256}$; for the full-path relation, $(p, t, K_n, NS_n, h_{nw}, c_{\mathsf{seed}})$, with NS_n likewise moved into the statement. The scheme-specific binding hash (h_{cw} or h_{nw} for QBIP32, h_{k_R} for BIP32-Ed25519, h_c for BIP32-secp256k1 and SLIP-10/Ed25519) is unchanged from the pre-Q-day relation. The signing proof is unchanged, since it does not involve scalar multiplication: $\mathcal{R}^{\mathsf{pruned}}_{\mathsf{sign}}$ in the pruned case and $\mathcal{R}^{\mathsf{hard}}_{\mathsf{sign}}$ in the full-path case.

Post-Q-day verifier.

1. Verify the derivation proof: check $\mathsf{NIZKVerify}(\mathcal{R}^{\mathsf{qday}}_{\mathsf{deriv}}, (K_{\mathsf{leaf}}, NS_{\mathsf{leaf}}, h_{cw}), \pi_{\mathsf{deriv}}) = 1$.
2. Verify the signing proof: check $\mathsf{NIZKVerify}(\mathcal{R}^{\mathsf{pruned}}_{\mathsf{sign}}, (M, h_{cw}), \pi_{\mathsf{sign}}) = 1$.
3. Check cross-proof consistency: confirm that h_{cw} matches across both public statements.
4. Check the public key classically: confirm $K_{\mathsf{leaf}} = NS_{\mathsf{leaf}} \cdot G$.
5. Accept iff all four checks pass.

The steps above are stated for the pruned instantiation of QBIP32; the full-path case is identical with $\mathcal{R}^{\mathsf{hard}}_{\mathsf{sign}}$, h_{nw}, and K_n in place of $\mathcal{R}^{\mathsf{pruned}}_{\mathsf{sign}}$, h_{cw}, and K_{leaf}, and the corresponding instantiations for BIP32-Ed25519, BIP32-secp256k1, and SLIP-10/Ed25519 differ only in the scheme-specific binding hash and generator.

Remark 8.1 (Security assumption of the post-Q-day variant). The post-Q-day variant requires that after Q-day, any signature not produced by a post-quantum scheme is rejected by consensus. Before Q-day, the variant should not be used: a classical adversary who learns NS_{leaf} from the public statement can forge classical signatures for the corresponding address. The variant is presented here as a forward-looking construction.

9 Implementation and Benchmarks

All constructions are implemented in Rust using RISC Zero v5.0.0-rc.1, in five separate workspaces matching the scheme variants of this paper: `slip10-ed25519` (SLIP-10/Ed25519), `QBIP32` (hardened-only and pruned proof pairs), `QBIP32-full` (full-path relation), and the post-Q-day variants `slip10-ed25519-qday` and `QBIP32-qday`.

The hardware used for all benchmarks is: 16 cores / 32 threads at 4.5 GHz, 64 GB, running with Ubuntu operating system. Proof size is constant at approximately 219 KB and verification time is constant at approximately 9–10 ms across all configurations, depths, and path lengths reported below; this is a property of the RISC Zero succinct receipt format and is independent of circuit content, derivation depth, curve, and scheme variant. Verification columns are therefore omitted from the tables.

9.1 Monolithic Proof: BIP32-Ed25519 Σ

We first benchmark the baseline construction Σ, a single STARK that re-derives the entire chain from the root seed to depth n and binds the signature in one monolithic circuit, without any split into derivation and signing proofs. Table 2 reports proving time, verification time, cycle counts, and resource use at selected depths, and Table 3 breaks the guest cycle count down by component. Proving time grows linearly, $\mathsf{Prove}(n) \approx 61.7 \cdot n + 69.5$ seconds, from approximately 200 seconds at $n = 2$ to approximately 9138 seconds (approximately 2.5 hours) at $n = 147$, while verification stays constant at 9–10 ms, proof size at 218.4 KB, and peak RAM below 8.4 MB. As Table 3 shows, the `derive_chain` component dominates and grows at approximately 1.18×10^6 cycles per level, while the Ed25519 scalar multiplication and the SHA-256 signature hash are depth-independent constants. This linear blow-up is precisely what the split-proof and pruning constructions of the previous sections are designed to remove.

9.2 BIP44 Path Benchmarks

We now benchmark the split-proof schemes across the full BIP44 path $m/44'/0'/0'/0/0$ and its prefixes. In every table, derivation proving is the one-time cost per derived key pair and sign proving is the per-transaction cost.

Depth n	**Gen** (ms)	**Prove** (s)	**Total cycles**	**User cycles**	**Segs**	**RAM (MB)**	**CPU (%)**
2	<1	199.7	4 194 304	3 504 965	4	6.6	2863
7	<1	502.0	10 485 760	9 424 925	10	7.9	2881
12	<1	807.7	16 777 216	15 344 939	16	8.0	2878
22	<1	1425.0	29 425 664	27 184 859	29	8.0	2877
32	<1	2039.9	42 205 184	39 024 779	41	8.0	2878
52	<1	3291.9	68 157 440	62 704 619	65	7.9	2879
72	<1	4492.4	93 323 264	86 384 459	89	7.9	2879
97	<1	6048.4	125 304 832	115 984 325	120	8.0	2878
122	1	7596.1	157 286 400	145 584 127	150	8.4	2879
147	1	9137.9	189 267 968	175 183 927	181	8.4	2879

Table 2: Monolithic proof Σ for BIP32-Ed25519 at selected depths, implementing the full-chain relation from the root seed in a single STARK. This construction is a didactic baseline that motivates the split-proof architecture.

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
22	26 064 629	1 061 134	4 751	5 924	27 144 067
52	61 579 319	1 061 134	4 751	5 924	62 658 907
97	114 851 354	1 061 134	4 751	5 924	115 931 195
147	174 042 504	1 061 134	4 751	5 924	175 122 597

Table 3: Guest cycle breakdown for the monolithic proof. `derive_chain` grows at approximately 1.18×10^6 cycles per unit of n; Ed25519 scalar multiplication and SHA-256 are depth-independent constants.

Table 4 reports SLIP-10/Ed25519 (workspace `slip10-ed25519`), which implements the SLIP-10/Ed25519 hardened-only derivation and signing proofs—the split-proof construction of Section 5 specialised to the hardened SLIP-10 derivation step of Section 3.9, with the chain code c_n as the quantum-safe witness and hash binding $h_c = \mathsf{SHA256}(c_n)$. Since only the last hardened step is proved, the derivation time is constant at approximately 76 seconds regardless of prefix length.

Path	**Deriv prove (s)**	**Sign prove (s)**
$m/44'$	76.37	12.72
$m/44'/0'$	76.72	12.64
$m/44'/0'/0'$	76.29	12.65
$m/44'/0'/0'/0$	76.43	12.65
$m/44'/0'/0'/0/0$	76.48	12.68

Table 4: SLIP-10/Ed25519 hardened-only derivation and signing proofs, with hash binding $h_c = \mathsf{SHA256}(c_n)$.

Tables 5 and 6 report the split-proof construction of Section 5 applied to BIP32-Ed25519 and BIP32-secp256k1 respectively, both using HMAC-SHA512 as the derivation MAC. Row one in each table implements the hardened-only pair $\mathcal{R}^{\mathsf{hard}}_{\mathsf{deriv}} + \mathcal{R}^{\mathsf{hard}}_{\mathsf{sign}}$, and the remaining rows implement the

pruned pair $\mathcal{R}^{\mathsf{pruned}}_{\mathsf{deriv}} + \mathcal{R}^{\mathsf{pruned}}_{\mathsf{sign}}$. The quantum-safe witness is the accumulated right half k_{R_n} for BIP32-Ed25519 with hash binding $h_{k_R} = \mathsf{SHA256}(k_{R_n})$, and the chain code c_n for BIP32-secp256k1 with hash binding $h_c = \mathsf{SHA256}(c_n)$.

Path	**Deriv prove (s)**	**Sign prove (s)**
$m/44'/0'/0'$	156.69	12.50
$m/44'/0'/0'/0$	157.49	12.51
$m/44'/0'/0'/0/0$	157.53	12.53

Table 5: BIP32-Ed25519 split proof (HMAC-SHA512). Row one implements $\mathcal{R}^{\mathsf{hard}}_{\mathsf{deriv}} + \mathcal{R}^{\mathsf{hard}}_{\mathsf{sign}}$; the rest implement $\mathcal{R}^{\mathsf{pruned}}_{\mathsf{deriv}} + \mathcal{R}^{\mathsf{pruned}}_{\mathsf{sign}}$.

Path	**Deriv prove (s)**	**Sign prove (s)**
$m/44'/0'/0'$	157.34	12.49
$m/44'/0'/0'/0$	157.48	12.51
$m/44'/0'/0'/0/0$	157.64	12.51

Table 6: BIP32-secp256k1 split proof (HMAC-SHA512), with hash binding $h_c = \mathsf{SHA256}(c_n)$. Row one implements $\mathcal{R}^{\mathsf{hard}}_{\mathsf{deriv}} + \mathcal{R}^{\mathsf{hard}}_{\mathsf{sign}}$; the rest implement $\mathcal{R}^{\mathsf{pruned}}_{\mathsf{deriv}} + \mathcal{R}^{\mathsf{pruned}}_{\mathsf{sign}}$.

Tables 7 and 8 report QBIP32 (workspace `QBIP32`) for Ed25519 and secp256k1 respectively, both using KMAC256 as the HASH768 instantiation. In both, the first row (whose leaf is the last hardened node) implements the hardened-only pair $\mathcal{R}^{\mathsf{hard}}_{\mathsf{deriv}} + \mathcal{R}^{\mathsf{hard}}_{\mathsf{sign}}$ with $h_{nw} = \mathsf{SHA256}(NW_n)$, and the remaining rows implement the pruned pair $\mathcal{R}^{\mathsf{pruned}}_{\mathsf{deriv}} + \mathcal{R}^{\mathsf{pruned}}_{\mathsf{sign}}$ with $h_{cw} = \mathsf{SHA256}(NC_{\mathsf{leaf}} \| NW_{\mathsf{leaf}})$. For Ed25519 (Table 7) the derivation time is flat at approximately 156 seconds, and for secp256k1 (Table 8) it is likewise flat at approximately 157 seconds.

Path	**Deriv prove (s)**	**Sign prove (s)**
$m/44'/0'/0'$	156.39	12.53
$m/44'/0'/0'/0$	155.77	12.50
$m/44'/0'/0'/0/0$	155.93	12.53

Table 7: QBIP32 for Ed25519 (KMAC256). Row one implements $\mathcal{R}^{\mathsf{hard}}_{\mathsf{deriv}} + \mathcal{R}^{\mathsf{hard}}_{\mathsf{sign}}$; the rest implement $\mathcal{R}^{\mathsf{pruned}}_{\mathsf{deriv}} + \mathcal{R}^{\mathsf{pruned}}_{\mathsf{sign}}$.

Tables 5, 6, 7, and 8 show that the split-proof derivation time for BIP32-Ed25519, BIP32-secp256k1, QBIP32-Ed25519, and QBIP32-secp256k1 is essentially the same on the RISC Zero backend, clustering at approximately 156–158 seconds across the hardened last-step and pruned suffix configurations, while SLIP-10/Ed25519 (Table 4) is faster by approximately a factor of two at 76 seconds. The RISC Zero proving cost is dominated by the fixed per-padded-segment STARK cost rather than by the exact number of user cycles executed within each segment, and the

Path	Deriv prove (s)	Sign prove (s)
$m/44'/0'/0'$	157.47	12.52
$m/44'/0'/0'/0$	156.95	12.56
$m/44'/0'/0'/0/0$	157.13	12.56

Table 8: QBIP32 for secp256k1 (KMAC256). Row one implements $\mathcal{R}^{\mathsf{hard}}_{\mathsf{deriv}} + \mathcal{R}^{\mathsf{hard}}_{\mathsf{sign}}$; the rest implement $\mathcal{R}^{\mathsf{pruned}}_{\mathsf{deriv}} + \mathcal{R}^{\mathsf{pruned}}_{\mathsf{sign}}$.

derivation circuits for the four pruned split proofs land in comparable segment configurations, which quantises small guest-cycle differences away in the reported proving times. Under this condition the single-KMAC256-call structure of QBIP32, which replaces the two-call HMAC-SHA512 construction of BIP32-Ed25519 with one call to a keyed hash producing the signing scalar, the quantum-safe witness, and the chain code together, does not surface as a proving-time advantage over BIP32 variants on this backend.

The design contribution of QBIP32 is therefore not proving-time reduction in this backend but a unification of the derivation procedure across curves. The same single-call keyed hash instantiation of HASH768 defines derivation for secp256k1, Ed25519, and any future elliptic curve group of prime order with a fixed generator, in contrast to BIP32-Ed25519 whose two-call HMAC-SHA512 structure is tied to the Ed25519 extended-key format and does not extend to other curves. QBIP32 additionally produces the quantum-safe witness NW_n as a specific output of the derivation function rather than as an incidental byproduct of a particular hash construction, and keeps the chain code NC_n unconditionally secret as a keyed-hash input. Whether the single-call structure translates to a proving-time advantage on backends that provide a native representation of the Keccak permutation or of KMAC256 as a lookup, so that the sponge is not interpreted in software, is left as an implementation question.

Tables 9 and 10 report QBIP32 full-path (workspace `QBIP32-full`), which implements $\mathcal{R}^{\mathsf{full\text{-}path}}_{\mathsf{deriv}}$, re-deriving root generation, all hardened prefix steps, the anchor step, and the non-hardened suffix inside a single STARK, paired with $\mathcal{R}^{\mathsf{hard}}_{\mathsf{sign}}$. Because nothing is pruned, derivation proving grows with total path length, reaching 440 seconds for Ed25519 and 476 seconds for secp256k1 at the full BIP44 path. The benchmarked circuit corresponds to the full-path relation without the seed-commitment condition (F-i) of Section 7.5, which would add one SHA-256 evaluation.

Path	Deriv prove (s)	Sign prove (s)
$m/44'/0'/0'$	167.88	12.55
$m/44'/0'/0'/0$	316.24	12.55
$m/44'/0'/0'/0/0$	439.65	12.54

Table 9: QBIP32 full-path for Ed25519 (KMAC256), implementing $\mathcal{R}^{\mathsf{full\text{-}path}}_{\mathsf{deriv}} + \mathcal{R}^{\mathsf{hard}}_{\mathsf{sign}}$.

9.3 Post-Q-Day Variant Benchmarks

We finally benchmark the post-Q-day variant of Section 8, in which all scalar multiplications are removed from the STARK and the verifier computes $K_i = NS_i \cdot G$ externally. Sign prove time and

Path	Deriv prove (s)	Sign prove (s)
$m/44'/0'/0'$	199.56	12.60
$m/44'/0'/0'/0$	332.49	12.64
$m/44'/0'/0'/0/0$	476.07	12.58

Table 10: QBIP32 full-path for secp256k1 (KMAC256), implementing $\mathcal{R}^{\mathsf{full\text{-}path}}_{\mathsf{deriv}} + \mathcal{R}^{\mathsf{hard}}_{\mathsf{sign}}$.

all verification times are unchanged relative to the standard variants, since those proofs do not involve scalar multiplication, and the external scalar multiplications add no measurable overhead to the constant verification time.

Tables 11 and 12 report QBIP32 post-Q-day (workspace `QBIP32-qday`) for Ed25519 and secp256k1, both implementing $\mathcal{R}^{\mathsf{qday}}_{\mathsf{deriv}}$: the pruned relation with the leaf public-key check (P-iii) removed from the circuit and NS_{leaf} moved into the public statement. Derivation proving drops to approximately 21 seconds when the leaf is the anchor and to approximately 40 seconds once a non-hardened suffix is present, and the two curves show identical times because all elliptic-curve multiplications are now outside the STARK.

Path	Deriv prove (s)	Sign prove (s)
$m/44'$	21.43	12.55
$m/44'/0'$	21.46	12.54
$m/44'/0'/0'$	21.43	12.58
$m/44'/0'/0'/0$	39.72	12.58
$m/44'/0'/0'/0/0$	39.85	12.60

Table 11: QBIP32 post-Q-day for Ed25519, implementing $\mathcal{R}^{\mathsf{qday}}_{\mathsf{deriv}}$; the verifier computes $K_i = NS_i \cdot G$ externally.

Path	Deriv prove (s)	Sign prove (s)
$m/44'$	21.38	12.57
$m/44'/0'$	21.43	12.54
$m/44'/0'/0'$	21.51	12.57
$m/44'/0'/0'/0$	39.77	12.54
$m/44'/0'/0'/0/0$	39.75	12.59

Table 12: QBIP32 post-Q-day for secp256k1, implementing $\mathcal{R}^{\mathsf{qday}}_{\mathsf{deriv}}$; the verifier computes $K_i = NS_i \cdot G$ externally.

Tables 13 and 14 report the same post-Q-day transformation applied to BIP32-Ed25519 and BIP32-secp256k1: the leaf public-key check is removed from the circuit and the leaf signing scalar (the accumulated left half for BIP32-Ed25519, the private scalar for BIP32-secp256k1) is moved into

the public statement, with the verifier computing K externally. Once the scalar multiplication is out of the circuit, the residual proving cost is set by the derivation hash calls, and the difference between the two variants becomes visible. BIP32-Ed25519 uses two HMAC-SHA512 calls per non-hardened step (one for the scalar tweak, one for the new chain code) whereas BIP32-secp256k1 uses one, and the extra work pushes the BIP32-Ed25519 pruned circuit into a larger segment tier: BIP32-Ed25519 post-Q-day rises from approximately 21 seconds when the leaf is the anchor to approximately 40 seconds once a non-hardened suffix is present (matching QBIP32 post-Q-day), while BIP32-secp256k1 post-Q-day stays at approximately 21 seconds at every path length.

Path	**Deriv prove (s)**	**Sign prove (s)**
$m/44'/0'/0'$	21.42	12.54
$m/44'/0'/0'/0$	39.77	12.53
$m/44'/0'/0'/0/0$	39.72	12.52

Table 13: BIP32-Ed25519 post-Q-day, implementing the split relations with the leaf public-key check removed from the circuit and the leaf left-half scalar moved into the public statement; the verifier computes $K = k_L \cdot B$ externally.

Path	**Deriv prove (s)**	**Sign prove (s)**
$m/44'/0'/0'$	21.44	12.56
$m/44'/0'/0'/0$	21.46	12.58
$m/44'/0'/0'/0/0$	21.51	12.60

Table 14: BIP32-secp256k1 post-Q-day, implementing the split relations with the leaf public-key check removed from the circuit and the leaf signing scalar moved into the public statement; the verifier computes $K = k \cdot G$ externally.

Table 15 reports the corresponding SLIP-10/Ed25519 post-Q-day variant (workspace `slip10-ed25519-qday`), which removes $K_n = NS_n \cdot B$ from the SLIP-10 derivation relation and publishes NS_n; its constant approximately 21.6 seconds is tied with BIP32-secp256k1 post-Q-day for the fastest derivation configuration in the paper.

Path	**Deriv prove (s)**	**Sign prove (s)**
$m/44'$	21.53	12.67
$m/44'/0'$	21.58	12.70
$m/44'/0'/0'$	21.62	12.68
$m/44'/0'/0'/0$	21.70	12.71
$m/44'/0'/0'/0/0$	21.63	12.67

Table 15: SLIP-10/Ed25519 post-Q-day, implementing the SLIP-10 derivation relation with $K_n = NS_n \cdot B$ removed and NS_n published; the verifier recovers the key classically.

9.4 Summary Comparison and Discussion

Table 16 collects the derivation and sign proving times of all variants at the full BIP44 path $m/44'/0'/0'/0/0$. The fastest derivation configurations are BIP32-secp256k1 post-Q-day at 21.5 seconds and SLIP-10/Ed25519 post-Q-day at 21.6 seconds; among the standard (pre-Q-day) variants, SLIP-10/Ed25519 at 76.5 seconds is fastest, followed by the BIP32 and QBIP32 pruned variants which cluster at approximately 156–158 seconds. Sign proving is constant at approximately 12.5–12.7 seconds in every row by construction.

Scheme	Deriv prove (s)	Sign prove (s)	Deriv cost (¢)	Sign cost (¢/tx)
SLIP-10/Ed25519	76.5	12.7	0.74	0.12
BIP32-Ed25519 (pruned)	157.5	12.5	1.52	0.12
BIP32-secp256k1 (pruned)	157.6	12.5	1.52	0.12
QBIP32 Ed25519 (pruned)	155.9	12.5	1.50	0.12
QBIP32 secp256k1 (pruned)	157.1	12.6	1.52	0.12
QBIP32 full-path Ed25519	439.7	12.5	4.24	0.12
QBIP32 full-path secp256k1	476.1	12.6	4.59	0.12
QBIP32 post-Q-day Ed25519	39.9	12.6	0.38	0.12
QBIP32 post-Q-day secp256k1	39.8	12.6	0.38	0.12
BIP32-Ed25519 post-Q-day	39.7	12.5	0.38	0.12
BIP32-secp256k1 post-Q-day	21.5	12.6	0.21	0.12
SLIP-10/Ed25519 post-Q-day	21.6	12.7	0.21	0.12

Table 16: Summary comparison at the full BIP44 path $m/44'/0'/0'/0/0$. BIP32-Ed25519, BIP32-secp256k1, and SLIP-10/Ed25519 use HMAC-SHA512 as the derivation MAC; QBIP32 variants use KMAC256. Cost columns convert the measured times at the flat OVH cloud rate of $\$9.65 \times 10^{-5}$ per second ($250/month); derivation cost is one-time per key pair, signing cost is per transaction.

Derivation proving is a one-time cost per derived key pair, generated as soon as the seed is known and ahead of any transaction; it does not need to be repeated and is therefore not on the critical transaction path. Sign proving (≈12.5 s on this hardware) is the per-transaction cost; its circuit consists of two SHA-256 invocations using the RISC Zero hardware accelerator and is constant across all variants, depths, curves, and path lengths by construction.

All proving times above are measured on the dedicated OVH cloud server used throughout this work (AMD Ryzen 9 9950X3D, 16 cores / 32 threads, 64 GB RAM, no GPU, hosted in OVH's EU central region), billed at a flat $250 per month. Since this machine is used solely to generate proofs, we attribute its full monthly price to proving and convert it to a rate of $\$250/(720\,\text{h} \times 3600\,\text{s}) \approx \$9.65 \times 10^{-5}$ per second; the monthly price already bundles hardware, hosting, and power, so no separate energy accounting is required. Multiplying this rate by the measured proving times gives the cost columns of Table 16, expressed in US cents (¢). We stress the distinction between one-time and recurring cost: the derivation proof is generated once per key

pair, ahead of any transaction, so its cost is paid a single time for each derived key, whereas the signing proof is the only per-transaction cost. On this hardware the recurring per-transaction cost is constant at approximately 0.12 ¢ across every scheme, curve, and depth. The one-time activation of a quantum-safe address is approximately 0.21 ¢ for the cheapest post-Q-day configurations (BIP32-secp256k1 and SLIP-10/Ed25519), so a full end-to-end post-Q-day cycle (derivation, signing, and verification, the latter adding a negligible $\approx 10^{-4}$ ¢) costs under one third of a US cent.

Although derivation proving is off the critical transaction path, it is not free: in the full-path variant it grows with total path length (up to ≈440–476 s at the full BIP44 path), and even in the pruned variant it is the largest cost in the system. This makes the derivation proof the natural target for incremental proving, where the proof for one derivation level reuses work from the previous level rather than re-deriving from scratch.

10 Incremental Derivation and Recursion

All four proof configurations (SLIP-10/Ed25519, QBIP32 hardened-only, QBIP32 pruned, QBIP32 full-path) admit in principle some form of incremental derivation proof, where the proof for level i builds on the proof for level $i-1$ rather than re-deriving from scratch. Three concrete approaches can be identified, each with different trade-offs.

The first approach is hash-linked independent proofs: each derivation level i produces an independent STARK that proves only the single step from level $i-1$ to level i. The public statement at level i includes cryptographic hashes inherited from level $i-1$; the proof at level i proves that these hashes are one-way hashes of values in its witness and that the derivation step was performed correctly. This is the same technique already used to link $\mathcal{R}_{\mathsf{deriv}}$ and $\mathcal{R}_{\mathsf{sign}}$, extended across derivation levels. Each individual proof is cheap. The disadvantages are: the verifier must verify all d receipts in sequence, one per level; and since each RISC Zero succinct receipt occupies approximately 218 KB (Section 9), a path of depth d requires $d \times 218$ KB of storage.

The second approach is STARK recursion: the receipt π_{i-1} is embedded as a witness inside the proof for level i, which verifies π_{i-1} inside the circuit. The result is a single compact receipt regardless of depth. The derivation proof from the last hardened node at level $i-1$ and at level i share a large portion of the same computation (the difference is exactly one additional derivation step). However, in a STARK the prover still pays the full cost of verifying π_{i-1} inside the circuit at each level. Whether the net saving is positive depends on the relative cost of in-circuit receipt verification versus re-deriving one step, which requires empirical analysis.

The third approach is folding schemes, which are designed for exactly this structure: proving the repeated application of the same step function F incrementally, where each step folds the previous accumulator at roughly constant per-step cost. The derivation function $F(NS_{i-1}, NW_{i-1}, NC_{i-1}, x_i) \to (NS_i, NW_i, NC_i)$ is identical at every hardened step, making it a natural fit. However, two fundamental obstacles must be addressed. First, RISC Zero is a blackbox zkVM: the folding accumulator must be injected at the arithmetization level, which RISC Zero does not expose. Using folding with RISC Zero would require either a zkVM that natively supports incrementally verifiable computation or reimplementing the derivation relation as a native circuit in a folding-friendly framework, abandoning the zkVM model entirely. Second, the scheme must remain post-quantum secure. Traditional folding schemes rely on the hardness of the discrete logarithm problem over elliptic curves and are therefore broken by a quantum adversary. A plausible post-quantum folding direction today is FRI-based folding, which builds incrementally verifiable computation structures entirely on cryptographic hash-based commitments. This line of research is active but not yet mature, and no production-ready post-quantum folding scheme currently exists.

Remark 10.1 (Simulation extractability under composition). An important security requirement that applies to all three approaches is that the composed derivation proof must satisfy simulation extractability. Simulation extractability is necessary to prove the unforgeability of the full signing scheme: without it, an adversary could potentially combine a valid derivation receipt with a forged signing proof. When recursion or folding is introduced, the simulation extractability of the composed scheme must be re-established from the simulation extractability of the component proofs. This is a non-trivial open analysis item for all three approaches described above and is flagged as a prerequisite for any secure deployment of incremental derivation.

11 Conclusion

We have presented a post-quantum signing framework for hierarchical deterministic wallets that preserves the existing address format and replaces only the signing step with a NIZK proof of seed provenance. The monolithic scheme Σ establishes feasibility and gives the first complete game-based reduction for BIP32-Ed25519 against QPT adversaries, but has proving cost linear in derivation depth. ZKPoSP removes this cost from the signature generation by splitting the proof into a derivation proof, generated once per key pair, and a signing proof, generated once per message at the cost of two SHA-256 invocations; the two are linked by the one-way hash binding $h_{k_R} = \mathsf{SHA256}(k_{R_n})$, and ZKPoSP is proved EUF-CMA and hierarchically EUF-CMA secure. The derivation proof depends only on the seed and the derivation path and not on any transaction data. It is therefore not on the transaction critical path and can be generated at any time after the key pair is created and long before any transaction is signed. We then characterise exactly when the derivation proof can be shortened to a single last step: this is sound for hardened keys and unsound for non-hardened keys in every scheme. Then we introduced QBIP32 unifying the constructions: this protocol makes the quantum-safe witness NW an explicit output of every $\mathsf{HASH768}$ call, and one derivation scheme works uniformly across any prime-order curve[3]. For BIP44 paths this yields a fixed-size, depth-independent derivation proof in a pruned variant. The post-Q-day variant applies to every hardened-anchor scheme in this paper: it moves the leaf scalar to the public statement, removing all in-circuit scalar multiplications, under a network-level assumption that becomes vacuous after Q-day. On the full BIP44 path this cuts derivation proving from about 156 to 40 seconds for QBIP32 (both curves) and for BIP32-Ed25519, and from about 76 to 21 seconds for SLIP-10/Ed25519; BIP32-secp256k1 post-Q-day is the joint-fastest configuration at about 21 seconds, tied with SLIP-10/Ed25519 post-Q-day. All constructions are implemented in Rust over RISC Zero with KMAC256 as the $\mathsf{HASH768}$ instantiation for QBIP32 and HMAC-SHA512 for BIP32-Ed25519, BIP32-secp256k1, and SLIP-10/Ed25519. Signing proving is constant at approximately 12–13 seconds and verification at approximately 9–10 ms with proof size approximately 219 KB, across all variants, depths, and curves. In deployment, this means the cost paid at the moment of a transaction is only the constant $\approx$12.5 s signing proof; every derivation proof reported in Section 9 is a one-time expense that has already been paid long before that transaction is ever made.

The benchmarks of Section 9 show that derivation proving, although off the transaction critical path, remains the dominant cost in the system, and we plan to pursue several complementary strategies to reduce it further. First, as described in Section 10, the derivation proof can potentially be computed incrementally, so that deriving a new leaf reuses work already proven for a shared prefix rather than re-deriving it from scratch. Second, we plan to evaluate RISC Zero's GPU-accelerated

[3]In the case of BIP32-Ed25519 QBIP32 replaces the two HMAC calls per step with a single $\mathsf{HASH768}$ call, even if on the RISC Zero backend used this does not yield a measured proving-time advantage over BIP32-Ed25519 or BIP32-secp256k1. Indeed QBIP32's demonstrated contribution is curve-uniformity, not proving speed.

prover; this could reduce proving time without any change to the relations themselves, though the fixed per-segment overhead identified in Section 9.4 as the dominant cost for small circuits may not shrink proportionally, so the relative ordering between schemes observed in this paper may persist even under GPU acceleration. Third, we plan to investigate hardware- or precompile-level acceleration of KMAC256/Keccak-f inside RISC Zero, analogous to the existing SHA-256 accelerator used for the signing proof; native acceleration of the Keccak permutation could allow QBIP32's single-call structure to finally surface as a measured proving-time advantage over the two-HMAC-call schemes, rather than only a reduction in primitive-call count, at the cost of depending on RISC Zero's own roadmap or on maintaining a custom precompile. Fourth, the current Rust implementation is a proof of concept and has not been systematically optimised; profiling and reducing guest-side cycle count, memory access patterns, and redundant computation should lower proving time somewhat, although such gains are likely bounded by the fixed per-segment floor identified in Section 9.4 rather than by the guest program's exact cycle count. Fifth, we are exploring migrating away from a general-purpose zkVM entirely and implementing the derivation and signing relations as native circuits in Plonky3; this removes the zkVM's fixed per-segment overhead and gives fine-grained control over the arithmetization of KMAC256. This solution requires an audited circuit implementation that is at an earlier stage of maturity than the RISC Zero implementation reported here. Sixth, since the derivation proof is generated once per key pair and is not required at signing time, we plan to explore delegating its computation to a dedicated proving server or pool of servers, which would let wallets and lower-powered devices benefit from ZKPoSP without provisioning proving-grade hardware themselves. Since the derivation witness includes private seed or anchor material that must either be transmitted to the delegated server or processed inside a trusted execution environment, we need to design a delegation protocol that preserves the same privacy guarantees without being prohibitively costly for the server as well. One possible direction is multi-party computation (MPC), though this typically requires multiple non-colluding servers, whereas we prefer a single-server deployment model; an alternative is (verifiable) fully homomorphic encryption ((v)FHE), which would allow a single server to compute over encrypted witness material, but this technology is not yet mature enough for this setting.

Generative AI Usage

The writing of this paper was assisted by Claude to create a narrative to explain the research results. All AI-generated prose was manually validated and verified for correctness. No AI support was used to add the citations and to find the related works. None of this paper's research findings were AI-assisted.

References

[ADE+20] 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.

[AHIV17] 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.

[BBHR18] 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.

[BCR25] Foteini Baldimtsi, Konstantinos Chalkias, and Arnab Roy. Post-quantum readiness in EdDSA chains. Cryptology ePrint Archive, Report 2025/1368, 2025.

[DEF+21] 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.

[DEMS24] 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.

[DFGN26] 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.

[DFL19] 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.

[ER22] 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.

[GKR+21] 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.

[Gro96] 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.

[HR16] Jochen Hoenicke and Pavol Rusnak. SLIP-0010: Universal private key derivation from master private key. `https://github.com/satoshilabs/slips/blob/master/slip-0010.md`, April 2016.

[KL17] 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.

[LFA20] 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.

[Nat16] National Institute of Standards and Technology. SHA-3 Derived Functions: cSHAKE, KMAC, TupleHash, and ParallelHash. NIST Special Publication 800-185, National Institute of Standards and Technology, 2016.

[nic26] nicocsgy. SPHINCS minus: Efficient Stateless Post-Quantum Signature Verification on the EVM. `https://ethresear.ch/t/sphincs-minus-efficient-stateless-post-quantum-signature-verification-on-the-evm/25165`, 2026. Ethereum Research forum post, accessed 2026-06-23.

[OANWO20] Jack O'Connor, Jean-Philippe Aumasson, Samuel Neves, and Zooko Wilcox-O'Hearn. BLAKE3: One function, fast everywhere. `https://github.com/BLAKE3-team/BLAKE3-specs/blob/master/blake3.pdf`, 2020.

[SD25] Surbhi Shaw and Ratna Dutta. Post-quantum secure compact deterministic wallets from isogeny-based signatures with rerandomized keys. *Theoretical Computer Science*, 1035:115127, 2025.

[Sho94] 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.

[Wui12] Pieter Wuille. BIP 32: Hierarchical deterministic wallets. Bitcoin Improvement Proposal, February 2012.