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

October 15, 2026

Abstract

In the blockchain setting, where 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. We present an orthogonal approach: keep the existing address format unchanged and replace the classical signing step with a NIZK proof of knowledge of the seed underlying the address (or of the private data of a hardened ancestor), where no step of the security argument relies on the discrete logarithm problem. We prove security against quantum adversaries in the quantum random oracle model, under an explicit assumption on the zero-knowledge and simulation extractability of the NIZK.

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 result to hierarchical deterministic wallets. Our main contribution is ZKPoSP (Zero-Knowledge Proof of Seed Provenance), a signature scheme that splits the proof into a derivation proof generated once per key pair and a signing proof generated once per message, so that the per-message proving cost does not depend on the derivation depth. We also introduce QBIP32, a key derivation scheme based on a keyed function HASH768 (instantiated with KMAC256) that produces the signing scalar, an explicit witness, and the chain code in a single call at hardened steps, for any elliptic curve group of prime order. As last contributioun we notice that after Q-day, once networks reject all non-post-quantum signatures, the signing scalars can be moved to the public statement reducing derivation proving time.

Contents

1 Introduction **3**
1.1 Contributions 5
1.2 Technical Overview 5

2 Related Work **8**

3 Preliminaries **9**
3.1 Notation 9
3.2 Discrete Logarithm 9

3.3 Random Oracle Models . . . 10
3.4 Hash Functions . . . 10
3.5 Pseudorandom Functions and Keyed Hash Functions . . . 11
3.6 Simulation Extractable NIZKs . . . 12
3.7 Digital Signatures . . . 13
3.8 Hierarchical Deterministic Wallets . . . 14
3.9 SLIP-10/Ed25519 Key Derivation . . . 16
3.10 BIP32 Key Derivation (secp256k1) . . . 17
3.11 BIP32-Ed25519 Key Derivation . . . 20

4 Monolithic Post-Quantum Signing for BIP32-Ed25519 22
4.1 Signature Scheme Σ . . . 22
4.2 Security of Σ . . . 23

5 ZKPoSP: Split-Proof Architecture 24
5.1 Derivation Relation $\mathcal{R}_{\mathsf{deriv}}$ . . . 25
5.2 Signing Relation $\mathcal{R}_{\mathsf{sign}}$ . . . 26
5.3 Combined Relation and Verification . . . 26
5.4 ZKPoSP for Non-Hardened BIP32-Ed25519 . . . 26

6 Can the Derivation Proof Be Shortened? 29
6.1 When Last-Step Certificates Can Be Used . . . 29
6.2 Motivation for QBIP32 . . . 30

7 QBIP32: A Universal Sponge-Based HD Wallet 31
7.1 The HASH768 Primitive . . . 31
7.2 Node Structure and Key Derivation . . . 31
7.3 NP Relations for Hardened Paths . . . 33
7.4 Pruned NP Relations . . . 35
7.5 Full-Path Relation . . . 36

8 Post-Q-Day Variant 37

9 Implementation and Benchmarks 38
9.1 Monolithic Proof: BIP32-Ed25519 Σ . . . 38
9.2 BIP44 Path Benchmarks . . . 39
9.3 Post-Q-Day Variant Benchmarks . . . 40
9.4 Summary . . . 43

10 Incremental Derivation and Recursion 43

11 Conclusion 44

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 HD wallet schemes in use is based 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. We think that it is preferrable an approach in whch the public key format remains unchanged and instead replace the classical signing step with a proof of knowledge of the seed used to create the wallet. This proof can be generated using a NIZK argument of knowledge (NIZKAoK) that is secure against quantum adversaries. This approach was proopsed 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. Our solution is compatible with existing infrastructure since public keys have the same format as those produced by any standard BIP32-Ed25519 or BIP32 wallet following the same type of derivation path.

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 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 witness at the hardened anchor, that is, a private value

of the parent of the anchor that is an input of the anchor step and cannot be computed from public data, even by a quantum adversary; this value then plays the role of the seed for the subtree below the parent. We characterise this for four schemes.

- SLIP-10/Ed25519, BIP32 secp256k1, and BIP32-Ed25519 have a witness for hardened keys only. Non-hardened keys are vulnerable to an attack caused by a variable that can be freely chosen by the adversary. For BIP32 secp256k1 and BIP32-Ed25519 the witness is a value w that the wallet computes with SHA-256, since in BIP32 secp256k1 the chain code is part of exported extended public keys and in BIP32-Ed25519 the right half k_R of a node can be computed from that of a non-hardened child; w changes no key or address.
- QBIP32 (introduced here) provides a 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 BIP32-Ed25519, BIP32 secp256k1, and QBIP32, the existence of a 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; for SLIP-10/Ed25519, where all levels of a BIP44 path are hardened, it covers only the last hardened step. 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. With our solution 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. In all our constructions, the derivation proof must establish a link between the secret signing scalar used in the underlying key derivation scheme and its corresponding public key, which requires proving a scalar multiplication inside the NIZK. This is necessary to maintain hidden the secret signing scalar, indeed 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 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. ZKPoSP replaces the classical signing step with a NIZK proof of knowledge of a seed, requiring no address migration or new key registration. It 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 against quantum adversaries in the quantum random oracle model, under an assumption on the NIZK (see the paragraph on the security model below).

2. 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 at hardened steps and provides a witness for any prime-order elliptic curve group (Section 7). This enables ZKPoSP to be instantiated over secp256k1, Ed25519, and any curve used in blockchain infrastructure.

3. Hardened-anchor construction. For BIP44 paths, a single NIZK covers the last hardened step plus the non-hardened suffix (empty for SLIP-10/Ed25519), giving a derivation proof independent of the hardened-prefix depth (Sections 6 and 7). The theorems are stated for the hardened-only and pruned variants; the security of the full variant follows by the same argument.

4. Implementation and benchmarks. All constructions are implemented in Rust using RISC Zero v5.0.0-rc.1 (Section 9). Signing proving time is approximately 12–13 seconds and verification time approximately 9–10 ms across all variants, depths, and curves; derivation proving for a full BIP44 path takes approximately 76 seconds for SLIP-10/Ed25519 and approximately 156–158 seconds for the other schemes.

1.2 Technical Overview

A BIP32-Ed25519 wallet derives its entire key hierarchy deterministically from a single 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. In the quantum random oracle model, 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), as explained in the paragraph on the security model below. For simplicity, the present construction is shown for non-hardened nodes only; the hardened

case is treated in Section 6, and 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 $h_w = \mathsf{SHA256}(w_n)$, where w_n is a witness key computed with SHA-256 from the seed along the derivation path: $\mathcal{R}_{\mathsf{deriv}}$ certifies that A_n was honestly derived from a seed and that h_w is the one-way hash of the chain-derived w_n; $\mathcal{R}_{\mathsf{sign}}$ certifies, for a given message M, that the signer holds a w_n consistent with h_w and that $h = \mathsf{SHA256}(w_n \| M)$ 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 the derivation step. For a non-hardened step the parent witness key is not an input of the step, so an adversary can choose a fake parent witness key and produce a passing proof for a value of its own choice. For a hardened step all private values of the parent are inputs of the step, and if the adversary cannot compute them, they play the role of the seed for the subtree below the parent. We call such a value a witness and characterise its existence across four schemes. SLIP-10/Ed25519 uses the chain code c_n as witness at hardened nodes. For BIP32 secp256k1 and BIP32-Ed25519 no value defined by the standard is suitable: in BIP32 secp256k1 every private scalar is recoverable from its public key via Shor's algorithm and the chain code is part of every extended public key, and in BIP32-Ed25519 the right half k_R of a node can be computed from that of a non-hardened child. We therefore add a witness w, computed with SHA-256 from the private parent values at each hardened step and extended by a one-way chain below it; it changes no key, address, or extended key.

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 $\mathsf{HASH768} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^{768}$ with 768-bit output, so that every derivation step explicitly produces a 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 witness increment at hardened steps (it is discarded at non-hardened steps), 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; at a non-hardened step, T_W is public, and $NW_{\mathsf{child}} \coloneqq \mathsf{SHA256}(NW_{\mathsf{par}} \| \langle x \rangle_4)$ instead, where x is the index of the child. 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 witness present by construction at every

[2] Concretely, the signing proof demonstrates knowledge of a secret value w_n such that $\mathsf{SHA256}(w_n)$ matches the value h_w fixed by the derivation proof, and that $\mathsf{SHA256}(w_n \| M)$ 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.

hardened node for any elliptic curve group, without any structural change to the derivation or proof system across curves. This halves the number of keyed-function calls 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 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, BIP32 secp256k1, and QBIP32 all admit a shortened derivation proof for hardened keys (for BIP32 secp256k1 and BIP32-Ed25519, if every child of the root is hardened, as in BIP44), 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 (for SLIP-10/Ed25519 all levels are hardened, so the proof covers only the last hardened step). 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 linkability to the seed at higher proving cost. The post-Q-day variant further removes all scalar multiplications from both variants by moving the needed signing scalars to the public statement, as discussed above, reducing derivation proving time substantially on the full BIP44 path.

Security model. No proof in this paper uses the discrete logarithm assumption or other assumptions known to be broken byt a quantum adversary. This alone does not prove security against quantum adversaries since an adversary can evaluate the hash functions in superposition. For this reason we proved our schemes in the QROM model under the following assumìtions: zero-knowledge and simulation extractability of the NIZK hold against quantum adversaries, with a straightline extractor. Known results come close to this assumption for hash-based SNARKs [CMS19, CDHZ25], but to our knowledge it has not been proved for STARKs.

Implementation and performance. We implement all constructions in Rust using RISC Zero as the zkVM backend, with 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 on $m/44'/0'/0'/0'/0'$), 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 keys generated with this protocol are secure only under the discrete logarithm assumption, which a quantum adversary can break. BIP32-Ed25519 [KL17] adapts BIP32 to the Edwards25519 group. Unlike BIP32, it 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 as the EdDSA signing nonce); a second call, with the same key and a different prefix byte, 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.

Das et al. [DFL19] introduced formal security notions and proved constructions secure under the discrete-logarithm assumption for deterministic wallets. Das et al. [DEF+21] subsequently tightened the analysis, providing exact security bounds for BIP32 and addressing gaps in the original reduction. We follow the the hierarchical EUF-CMA model defined by Di Luzio et al. [LFA20]. In their model the derivation keys propagate transitively to descendants and security is defined via a hierarchical experiment. 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. Seck and Roux-Langlois [SR25] construct DilithiumRK, a signature scheme with rerandomizable keys based on Dilithium, and derive deterministic wallets for Bitcoin from it. All of these proposals require address migration; our construction avoids this by preserving the BIP32 and BIP32-Ed25519 public key formats exactly.

An orthogonal line of work targets post-quantum security for existing address formats without migration. Sattath and Wyborski [SW23] introduce signature lifting: whenever a post-quantum one-way function appears somewhere along the path from secret key to public key, a deployed pre-quantum signature scheme can be converted into a post-quantum one under the same keypair. Their seed lifting instantiation applies this to HD-derived keys, using Picnic [CDG+17] to prove a single application of H. The resulting signature is a triple (σ, msk, P), where msk is the master key, P the derivation path, and σ a Picnic argument of knowledge of an H-preimage of msk; verification recomputes $\mathrm{Der}(msk, P)$ classically, outside the proof, and compares it with the on-chain public key. Because the derivation chain is checked in the clear, the soundness issues that motivate our witness analysis (Section 6) do not arise for their construction. Their proof is smaller, since Picnic certifies a single hash preimage rather than a derivation chain, but revealing msk lets a classical adversary recover every descendant scalar. The authors leave the removal of this exposure as an open problem; they post msk on-chain only if the spender fails to reveal in a commit-reveal protocol, and consider its use only in a post-quantum era, the same licence our post-Q-day variant relies on for exposing

signing scalars. Our construction instead keeps the derivation chain hidden inside the NIZK, except for the signing scalars of the post-Q-day variant, at the cost of a proof that scales with the chain rather than with a single hash evaluation.

A closer target for that same trade-off is Baldimtsi et al. [BCR25], who observe that EdDSA's deterministic seed-to-key mapping makes the seed itself a valid zero-knowledge witness for the public key, enabling a post-quantum proof of key ownership without changing the address format. Unlike seed lifting, their construction hides the witness (the seed) inside the proof rather than exposing it classically, 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 nonetheless the degenerate case $n = 0$ of the present framework: plain single-level EdDSA (RFC 8032) with no derivation chain, no hardened or non-hardened distinction, no split-proof architecture, and no hardened-anchor decomposition. Consigny [nic26] presents 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, the paper achieves 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 on a 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 b, we write $\langle v \rangle_b$ for the b-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 Discrete Logarithm

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 Models

Classical random oracles.

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 $\mathcal{O} : \{0,1\}^* \to \{0,1\}^{\ell_\mathcal{O}}$:

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

where $\mathcal{O}$ is sampled uniformly from the set of all functions $\{0,1\}^* \to \{0,1\}^{\ell_\mathcal{O}}$ 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 $\{0,1\}^{\ell_\mathcal{O}}$ and independent of all previous outputs.

We model SHA-256 and SHA-512 on fixed-length inputs as random oracles. We also model HMAC-SHA512 as a random oracle, with the 256-bit chain codes and the root keys `Bitcoin seed` and `ed25519 seed` as keys; HMAC with keys of a fixed length is indifferentiable from a random oracle when the underlying compression function is ideal [DRST12].

Quantum random oracles. A quantum adversary can evaluate a hash function on a superposition of inputs. In the quantum random oracle model (QROM) [BDF+11], the oracle $\mathcal{O}$ is a uniformly random function, but the adversary accesses it through the unitary $|x, y\rangle \mapsto |x, y \oplus \mathcal{O}(x)\rangle$. Honest parties and the challenger still query $\mathcal{O}$ classically. Several standard techniques of the classical ROM do not apply in the QROM: the reduction cannot record the queries of the adversary, cannot program the oracle lazily on fresh queries, and cannot rewind the adversary freely. A proof in the classical ROM therefore does not imply security in the QROM.

We use the following standard search bound, which follows from the compressed-oracle technique of Zhandry [Zha19]; we do not track the constant.

Lemma 3.3 (Search in the QROM)**.** *Let $\mathcal{O}$ be a random function, and for every input x let E_x be an event that depends only on $\mathcal{O}(x)$ and has probability at most δ. Then every algorithm that makes at most Q quantum queries to $\mathcal{O}$ outputs an x for which E_x holds with probability $O((Q+1)^2\delta)$. For classical queries the bound is $(Q+1)\,\delta$.*

We also model SHA-256, SHA-512, HMAC-SHA512, and HASH768 as quantum random oracles where we work in the QROM.

3.4 Hash Functions

Definition 3.4 (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 $H_s(x) \in \{0,1\}^{\ell_H(\lambda)}$.

Definition 3.5 (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 3.6 (Preimage resistance)**.** A hash function (Gen, H) is preimage resistant if for all PPT adversaries $\mathcal{A}$:

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

Grover's algorithm [Gro96] gives a quadratic speedup for preimage search: finding a preimage of an n-bit hash function takes $O(2^{n/2})$ evaluations on a quantum computer, compared to $O(2^n)$ classically. Hence SHA-256 and SHA-512 truncated to 256 bits give 128 bits of preimage security against quantum adversaries.

3.5 Pseudorandom Functions and Keyed Hash Functions

Definition 3.7 (Pseudorandom function (PRF))**.** Let $F : \{0,1\}^\lambda \times \mathcal{X} \to \mathcal{Y}$ be an efficient keyed function with domain $\mathcal{X}$ and range $\mathcal{Y}$. F is a pseudorandom function if for all PPT distinguishers $\mathcal{D}$:

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

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 $\mathcal{D}$.

We use $\mathsf{HMAC\text{-}SHA512}(k, m) = \mathsf{SHA512}\big((k \oplus \mathsf{opad}) \,\|\, \mathsf{SHA512}((k \oplus \mathsf{ipad})\|m)\big)$, where opad and ipad are the standard HMAC padding constants. Song and Yun [SY17] prove that HMAC is a secure PRF against quantum adversaries if its compression function is a secure PRF against quantum adversaries (for SHA-512 this is conjectured). In the QROM we model HMAC-SHA512 as a quantum random oracle; this is a heuristic, since the result of [DRST12] is classical and quantum indifferentiability is known only for some constructions [Zha19, Cza21].

Definition 3.8 (Keyed extensible-output function (keyed XOF))**.** Let $\ell_H : \mathbb{N} \to \mathbb{N}$ be a polynomially bounded output-length function. A keyed function $H : \{0,1\}^\lambda \times \{0,1\}^* \to \{0,1\}^{\ell_H(\lambda)}$ is a keyed extensible-output function if the following hold for all $k \xleftarrow{\$} \{0,1\}^\lambda$:

- (PRF) For any output length $L \le \ell_H(\lambda)$, the truncation $H(k,\cdot)[0:L]$ is a secure PRF against QPT adversaries.
- (Collision resistance) $H(k,\cdot)$ is collision resistant against QPT adversaries.
- (Preimage resistance) $H(k,\cdot)$ is preimage resistant against QPT adversaries.
- (Output independence) For any fixed d and $n = \ell_H(\lambda)/3$, 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 $\mathsf{HASH768} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^{768}$ 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:

$$\mathsf{HASH768}(k, d) \ :=\ \mathsf{KMAC256}(K{=}k,\ X{=}d,\ L{=}768,\ \mathit{Cust}{=}\text{“”}),$$

where L is the output length in bits, and *Cust* is a domain-separation string, that in our case is empty. Keys have $\lambda = 256$ bits, except in root resampling (Section 7.2), which uses a 768-bit key;

KMAC256 accepts keys of any length. KMAC256 is based on cSHAKE256 and inherits its security against QPT adversaries under the assumption that the sponge on the Keccak permutation behaves as a random function. The output independence property of Definition 3.8 holds for KMAC256 by the XOF security of cSHAKE256 and the fact that the three blocks occupy non-overlapping output positions. In the security proofs of Sections 6 and 7 we model $\mathsf{HASH768}$ as a random oracle; for KMAC256 this is justified by the indifferentiability of the sponge construction [BDPV08] and, against quantum adversaries, by its quantum indifferentiability [Cza21].

Remark 3.9 (Alternative instantiations of $\mathsf{HASH768}$). Any keyed XOF satisfying Definition 3.8 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. We model it as a random oracle, as stated above. 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. Root resampling needs 768-bit keys: KMAC256 and Poseidon2 accept them, while keyed BLAKE3 takes only 256-bit keys and needs a different resampling rule.

3.6 Simulation Extractable NIZKs

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

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

A simulator for Π is a pair $\mathsf{Sim} = (\mathsf{Sim}_H, \mathsf{Sim}_P)$ of PPT algorithms that share a state. $\mathsf{Sim}_H(\mathsf{pp}, q)$ answers a query q to the random oracle, and $\mathsf{Sim}_P(\mathsf{pp}, x)$ returns a proof π for x. The shared state is a table of input-output pairs, initially empty. On a query q, Sim_H returns the output stored for q; if there is none, it samples a uniform output, stores it, and returns it. To build a proof, Sim_P may store outputs of its choice for inputs that are not yet in the table (we say that Sim_P programs the random oracle), and Sim_H returns these outputs on later queries.

We require Π to satisfy the following properties, where all probabilities are also over the choice of H.

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

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

Knowledge soundness. There exists a PPT extractor Ext such that for every PPT prover $\mathcal{P}^*$:

$$\Pr\left[\begin{array}{l|l} \mathsf{NIZKVerify}^H(\mathsf{pp}, x, \pi) = 1 & \mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda) \\ \wedge\ (x, w) \notin \mathcal{R} & (x, \pi) \leftarrow \mathcal{P}^{*H}(\mathsf{pp}) \\ & w \leftarrow \mathsf{Ext}(\mathsf{pp}, x, L) \end{array}\right] \leq \mathsf{negl}(\lambda),$$

where L is the list of the queries of $\mathcal{P}^*$ to H with their answers.

Statistical zero-knowledge. There exists a simulator Sim such that for every $\mathcal{V}^*$, unbounded in time but making polynomially many queries to its oracles:

$$\left|\Pr\left[\mathcal{V}^{*H,\mathcal{P}(\mathsf{pp},\cdot,\cdot)}(\mathsf{pp})=1\right]-\Pr\left[\mathcal{V}^{*\mathsf{Sim}_H(\mathsf{pp},\cdot),\mathcal{S}(\mathsf{pp},\cdot,\cdot)}(\mathsf{pp})=1\right]\right|\leq \mathsf{negl}(\lambda),$$

where $\mathsf{pp}\leftarrow\mathsf{Setup}(1^\lambda)$, $\mathcal{P}(\mathsf{pp},x,w)$ returns $\mathsf{NIZKProve}^H(\mathsf{pp},x,w)$ and $\mathcal{S}(\mathsf{pp},x,w)$ returns $\mathsf{Sim}_P(\mathsf{pp},x)$ if $(x,w)\in\mathcal{R}$, and both return $\perp$ otherwise.

Simulation extractability. Let Sim be the simulator of the statistical zero-knowledge property. There exists a PPT extractor Ext such that for every PPT adversary $\mathcal{A}$:

$$\Pr\left[\begin{array}{l|l}\mathsf{NIZKVerify}^{\mathsf{Sim}_H(\mathsf{pp},\cdot)}(\mathsf{pp},x^*,\pi^*)=1 & \mathsf{pp}\leftarrow\mathsf{Setup}(1^\lambda)\\ \wedge\ x^*\notin Q & (x^*,\pi^*)\leftarrow\mathcal{A}^{\mathsf{Sim}_H(\mathsf{pp},\cdot),\mathsf{Sim}_P(\mathsf{pp},\cdot)}(\mathsf{pp})\\ \wedge\ (x^*,w^*)\notin\mathcal{R} & w^*\leftarrow\mathsf{Ext}(\mathsf{pp},x^*,L)\end{array}\right]\leq\mathsf{negl}(\lambda),$$

where Q is the set of the statements queried by $\mathcal{A}$ to Sim_P and L is the list of the queries of $\mathcal{A}$ to Sim_H with their answers.

We instantiate Π with a ZK-STARK [BBHR18], a transparent (no trusted setup) argument system with succinct proofs. In the ROM, Chiesa and Fenzi [CF24] prove that hash-based zkSNARKs obtained from interactive oracle proofs are UC-secure, which implies simulation extractability. Based on this evidence, we conjecture that the STARK of RISC Zero used in our implementation is simulation extractable; proving it is out of the scope of this paper.

The properties above are stated for PPT adversaries. For QPT adversaries we make the following assumption.

Assumption 3.10 (Zero-knowledge and simulation extractability in the QROM)**.** The NIZK Π is statistically zero-knowledge against every $\mathcal{V}^*$, unbounded in time, that makes polynomially many quantum queries to H (and Sim_H) and classical queries to the proof oracle; in this setting, Sim_H answers queries in superposition. The NIZK Π is also simulation extractable against QPT adversaries in the QROM, with a straightline extractor, that is, an extractor that does not rewind the adversary. In the experiment of simulation extractability, $\mathcal{A}$ queries H (and Sim_H) in superposition, while its queries to Sim_P stay classical. Since quantum queries cannot be recorded, Ext does not receive the list L: it answers the queries of $\mathcal{A}$ to Sim_H in its place and outputs w^* from its own state. The adversary may also have quantum access to other random oracles, independent of the oracle of the NIZK.

Assumption 3.10 is not proved, but there are many close results. Chiesa et al. [CMS19] prove that SNARGs obtained from IOPs with round-by-round soundness (as in STARKs) are sound in the QROM, and that the Micali construction keeps zero-knowledge and proof of knowledge in the QROM. Grilo et al. [GHHM21] bound the advantage of a quantum adversary in distinguishing a random oracle from an adaptively reprogrammed one, the technique used by Sim. Chiesa et al. [CDHZ25] prove security of such hash-based constructions in the QROM with an adaptive straightline quantum extractor, and Don et al. [DFMS22] give straightline extraction for hash-based commitments in the QROM. The ZK-STARK may be replaced by any NIZK with these properties.

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.

In this paper the message space is $\mathcal{M} = \{0,1\}^{256}$. An arbitrary message is first hashed with SHA-256, and the scheme signs the digest; by collision resistance of SHA-256, unforgeability for digests gives unforgeability for arbitrary messages.

Definition 3.11 (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 3.12 (EUF-CMA)**.** A signature scheme Σ is existentially unforgeable under chosen-message attack (EUF-CMA) if for all PPT adversaries $\mathcal{A}$:

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

We say that Σ is EUF-CMA secure against quantum adversaries if the same holds for all QPT adversaries $\mathcal{A}$ with quantum access to the random oracles; the signing oracle stays classical.

3.8 Hierarchical Deterministic Wallets

Following the framework of Di Luzio et al. [LFA20] we modified it as follows. We restrict the access hierarchy from a DAG to a tree, which matches the derivation structure of all HD wallets used in practice; we extend the derivation algorithms to 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 following the derivation rules according to the tree structure.

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.

The global public parameters pp are fixed at key generation. Each node v_i holds a *node state* st_i, which contains the public data needed for further derivation from v_i, including the public key pk_i; separately, the node has two private keys: a *derivation key* $d_i \in \mathcal{D}$, used to derive the keys of the children of v_i, and a *witness key* w_i, a private value bound to the seed. The witness key is computed at every derivation step, hardened and non-hardened, and is used for signing and for generating certificates. The signing key is $\mathsf{sk}_i := (d_i, w_i)$. 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 $\mathsf{sk}_i := (d_i, w_i)$.
- $\mathsf{DPub}(\mathsf{pp}, \mathsf{st}_i, v_i, v_j)$: given the node state st_i of the deepest hardened ancestor v_i of v_j (or of the root v_0 if v_j has no hardened ancestor), 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 after v_i 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.

Definition 3.13 (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 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 3.14 (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 (including v_0), computed by applying DPub iteratively from v_0. If v_0 has no non-hardened child, the chain code is removed from st_0 before it is given to $\mathcal{A}$. 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 $(M^*, v^*) \notin Q_{\mathrm{Sign}}$.

We say that $\Pi[F, F_{\mathsf{ver}}]$ is hierarchically EUF-CMA secure against quantum adversaries if the same holds for all QPT adversaries $\mathcal{A}$ with quantum access to the random oracles; the oracles $\mathcal{O}_{\mathrm{Corr}}$ and $\mathcal{O}_{\mathrm{Sign}}$ stay classical.

For the tree with the root as its only node, the experiment $G^{\mathrm{heuf}}_{\Pi,\mathcal{A}}(\lambda, T)$ is the EUF-CMA experiment of Definition 3.12 for the signature scheme whose key pair is $(\mathsf{sk}_0, \mathsf{pk}_0)$ and whose signature on M is (τ_0, σ). So hierarchical EUF-CMA implies EUF-CMA.

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. It is stored in the node state; the derivation relations take it as a private witness.

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 $\mathsf{st}_0 \coloneqq (K_0, \perp)$, where $\perp$ 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 $\mathsf{st}_{n-1} = (K_{n-1}, i_{n-1})$; take the index i_n of v_n from T; 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 $\mathsf{st}_{n-1} = (K_{n-1}, i_{n-1})$; take the index i_n of v_n from T; 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 computes $h_c \coloneqq \mathsf{SHA256}(c_n)$ and generates the signing proof π_{sign} for (M, h_c) as described in Section 5, and outputs $\sigma \coloneqq (\pi_{\mathsf{sign}}, h_c)$.
- $\mathsf{DCert}(\mathsf{pp}, d_h, w_h, \mathsf{st}_h, v_h, v_n, a, F)$: 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 private data $(k_{L_{n-1}}, c_{n-1})$ of the parent of v_n for last-step proofs, as described in Section 6.1), 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.
- $\mathsf{SignVerify}(\mathsf{pp}, K_n, M, \sigma)$: in classical mode, return 1 iff σ is a valid Ed25519 signature for M under K_n; in ZKPoSP mode, return 1 iff π_{sign} verifies for (M, h_c).
- $\mathsf{Verify}(\mathsf{pp}, K_n, \tau_n, M, \sigma, F_{\mathsf{ver}})$: 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 $\mathsf{SignVerify}(\mathsf{pp}, K_n, M, \sigma) = 1$, and the value h_c is the same in τ_n and σ.

Quantum vulnerability. Since $K_n = k_n \cdot B$, a QPT adversary recovers k_n from K_n with Shor's algorithm [Sho94], so SLIP-10/Ed25519 signatures are not unforgeable against quantum adversaries.

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 $\mathbb{G}_{\mathrm{secp}}$ 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.
  - The private key $k_i \in \mathbb{Z}_q$ at each depth, with $0 < k_i < q$. This is the derivation key $d_i \coloneqq k_i$.
  - The witness key $w_i \in \{0,1\}^{256}$ at each depth, defined below. It is not part of BIP32: the wallet computes it with SHA-256 from values it already holds, and it changes no private key, public key, address, or extended key.
- Node data:
  - The chain code $c_i \in \{0,1\}^{256}$ at each depth, stored in the node state. It is not part of the public key, but it is part of the extended public key (K_i, c_i), which wallets export, for example at the account level for watch-only use. From an exported (K_i, c_i) anyone computes the chain codes and public keys of all descendants of v_i reached by non-hardened steps; a QPT adversary also recovers k_i with Shor's algorithm and then computes all descendants, hardened or not.
- Public:
  - The compressed secp256k1 public key $K_i \coloneqq [k_i]G_{\text{secp}}$ 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. It is stored in the node state; the derivation relations take it as a private witness.

The global public parameters are $\mathsf{pp} \coloneqq K_0$, and the root signing key is $\mathsf{sk}_0 \coloneqq (k_0, w_0)$. The node state at depth i is $\mathsf{st}_i \coloneqq (K_i, c_i, i_i)$, which contains the public key, the chain code, and the child index used to derive v_i from its parent. We use two fixed tags, the ASCII strings $\mathsf{tag}_0 =$ `ZKPoSP-secp256k1 root` and $\mathsf{tag}_\mathsf{h} =$ `ZKPoSP-secp256k1 hardened`. The three uses of SHA-256 in the witness keys below are domain-separated: two of them start with distinct tags, and the third has a 36-byte input, shorter than both.

Root ($n = 0$). Given a seed x_0, compute $I \coloneqq \mathsf{HMAC\text{-}SHA512}(\text{"}\texttt{Bitcoin seed}\text{"}, x_0) \in \{0,1\}^{512}$ and parse $I = I_L \| I_R$ with $I_L, I_R \in \{0,1\}^{256}$. Set $k_0 \coloneqq I_L$ and $c_0 \coloneqq I_R$. If $I_L = 0$ or $I_L \geq q$, discard and resample. Set $K_0 \coloneqq [k_0]G_{\text{secp}}$ and $w_0 \coloneqq \mathsf{SHA256}(\mathsf{tag}_0 \| x_0)$. The root signing key is $\mathsf{sk}_0 \coloneqq (k_0, w_0)$, and the root node state is $\mathsf{st}_0 \coloneqq (K_0, c_0, \perp)$.

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}, parent node state $\mathsf{st}_{n-1} = (K_{n-1}, c_{n-1}, i_{n-1})$, and child index i_n, compute

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

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:

$$k_n \coloneqq (k_{n-1} + I_L) \bmod q, \qquad c_n \coloneqq I_R,$$
$$K_n \coloneqq [k_n]G_{\text{secp}} = K_{n-1} + [I_L]G_{\text{secp}}, \qquad w_n \coloneqq \mathsf{SHA256}(w_{n-1} \| \langle i_n \rangle_4).$$

The child signing key is $\mathsf{sk}_n \coloneqq (k_n, w_n)$, and the child node state is $\mathsf{st}_n \coloneqq (K_n, c_n, i_n)$.

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

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

where $\mathtt{ser}_{32}(k_{n-1})$ is the 32-byte big-endian encoding of k_{n-1}. Parse and update (k_n, c_n, K_n) identically to the non-hardened case, and set $w_n \coloneqq \mathsf{SHA256}(\mathsf{tag_h}\|\mathtt{ser}_{32}(k_{n-1})\|c_{n-1}\|\langle i_n\rangle_4)$. Because k_{n-1} is a private HMAC input, the child key is not publicly derivable without the parent private key. The child signing key is $\mathsf{sk}_n \coloneqq (k_n, w_n)$, and the child node state is $\mathsf{st}_n \coloneqq (K_n, c_n, i_n)$.

Derivation relation. Let v_h be the last hardened node on the path from the root to v_n (the anchor), so that $i_h \geq 2^{31}$ and $i_{h+1}, \ldots, i_n < 2^{31}$, and let v_{h-1} be its parent. The derivation relation $\mathcal{R}^{\mathsf{secp}}_{\mathsf{deriv}}$ has public statement $(K_n, h_w) \in \mathbb{G}_{\mathrm{secp}} \times \{0,1\}^{256}$ and private witness (k_{h-1}, c_{h-1}) together with the indices $i_h, \ldots, i_n$ and the intermediate values $\{k_j, c_j, w_j\}_{j=h}^{n}$, and it checks:

(BP-i) **Hardened anchor step.** $I = \mathsf{HMAC\text{-}SHA512}(c_{h-1},\, \mathtt{0x00}\|\mathtt{ser}_{32}(k_{h-1})\|\langle i_h\rangle_4)$, $k_h = (k_{h-1} + I_L) \bmod q$, $c_h = I_R$, and $w_h = \mathsf{SHA256}(\mathsf{tag_h}\|\mathtt{ser}_{32}(k_{h-1})\|c_{h-1}\|\langle i_h\rangle_4)$.

(BP-ii) **Non-hardened steps $j = h+1, \ldots, n$.** $K_{j-1} = [k_{j-1}]G_{\mathrm{secp}}$, $I = \mathsf{HMAC\text{-}SHA512}(c_{j-1},\, \mathtt{ser}_P(K_{j-1})\|\langle i_j\rangle_4)$, $k_j = (k_{j-1} + I_L) \bmod q$, $c_j = I_R$, and $w_j = \mathsf{SHA256}(w_{j-1}\|\langle i_j\rangle_4)$.

(BP-iii) **Leaf public key.** $K_n = [k_n]G_{\mathrm{secp}}$.

(BP-iv) **Hash binding.** $h_w = \mathsf{SHA256}(w_n)$.

For $n = h$ there is no non-hardened step, and this is the hardened-only relation. The signing relation is $\mathcal{R}_{\mathsf{sign}}$ of Section 5.2. If the path from the root to v_n contains no hardened node, the derivation relation instead starts from the seed: it computes the root step and w_0 from x_0, followed by (BP-ii) for $j = 1, \ldots, n$, (BP-iii) and (BP-iv).

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 $d_{n-1} = k_{n-1}$ and $\mathsf{st}_{n-1} = (K_{n-1}, c_{n-1}, i_{n-1})$; take the index i_n of v_n from T; apply the non-hardened child step; return $d_n \coloneqq k_n$, w_n, 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 $d_{n-1} = k_{n-1}$ and $\mathsf{st}_{n-1} = (K_{n-1}, c_{n-1}, i_{n-1})$; take the index i_n of v_n from T; apply the hardened child step; return $d_n \coloneqq k_n$, w_n, 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, computing only (K, c); 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 $d_{n-1} = k_{n-1}$ 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 computes $h_w \coloneqq \mathsf{SHA256}(w_n)$ and generates the signing proof π_{sign} for (M, h_w) as described in Section 5, and outputs $\sigma \coloneqq (\pi_{\mathsf{sign}}, h_w)$.
- $\mathsf{DCert}(\mathsf{pp}, d_h, w_h, \mathsf{st}_h, v_h, v_n, a, F)$: in classical mode, $F = \varepsilon$ and the certificate is empty; in ZKPoSP mode, $a \coloneqq (k_{h-1}, c_{h-1})$ is the private data of the parent of the anchor v_h (or $a \coloneqq x_0$ if the path has no hardened node), and F produces a proof π_{deriv} for $\mathcal{R}^{\mathsf{secp}}_{\mathsf{deriv}}$ with statement (K_n, h_w); the certificate is $\tau_n \coloneqq (\pi_{\mathsf{deriv}}, h_w)$.
- $\mathsf{DVerify}(\mathsf{pp}, K_n, \tau_n, F_{\mathsf{ver}})$: 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.
- $\mathsf{SignVerify}(\mathsf{pp}, K_n, M, \sigma)$: in classical mode, return 1 iff σ is a valid ECDSA or Schnorr signature for M under K_n; in ZKPoSP mode, return 1 iff π_{sign} verifies for (M, h_w).

- $\mathsf{Verify}(\mathsf{pp}, K_n, \tau_n, M, \sigma, F_{\mathsf{ver}})$: in classical mode, return $\mathsf{SignVerify}(\mathsf{pp}, K_n, M, \sigma)$. In ZKPoSP mode, return 1 iff $\mathsf{DVerify}(\mathsf{pp}, K_n, \tau_n, F_{\mathsf{ver}}) = 1$, $\mathsf{SignVerify}(\mathsf{pp}, K_n, M, \sigma) = 1$, and the value h_w is the same in τ_n and σ.

Quantum vulnerability and the witness key. Since $K_i = [k_i]G_{\mathrm{secp}}$, a QPT adversary recovers k_i from K_i with Shor's algorithm. The chain code is not a suitable witness key, because it is part of the extended public key. The witness key w_n is not part of any extended public key. At a hardened step it depends on k_{h-1} and c_{h-1}, so a QPT adversary can compute it if it knows the chain code of v_{h-1} or of one of its ancestors, since it recovers every private key with Shor's algorithm and then computes every hardened step below that ancestor. This happens if the root has a non-hardened child, since then c_0 is given in the game of Definition 3.14, or if an extended public key of one of these nodes has been exported.

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 $k_{L_i} \in \{0,1\}^{256}$ and the nonce seed $k_{R_i} \in \{0,1\}^{256}$, which together form the derivation key $d_i \coloneqq (k_{L_i}, k_{R_i})$ of the HDW.
  - The witness key $w_i \in \{0,1\}^{256}$, defined below. It is not part of BIP32-Ed25519: the wallet computes it with SHA-256 from values it already holds, and it changes no private key, public key, chain code, or address.
- 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 $c_0 \coloneqq \mathsf{SHA256}(\texttt{0x01} \| x_0)$. In the game of Definition 3.14, c_0 is given to the adversary only if the root has a non-hardened child.
  - 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. It is stored in the node state; the derivation relations take it as a private witness.

The global public parameters are $\mathsf{pp} \coloneqq A_0$ and the root signing key is $\mathsf{sk}_0 \coloneqq (d_0, w_0)$. 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, all extended keys $(k_{L_1}, k_{R_1}, \ldots, k_{L_n}, k_{R_n})$, and all witness keys are private. We use two fixed tags, the ASCII strings $\mathsf{tag}_0 = \texttt{ZKPoSP-ed25519 root}$ and $\mathsf{tag}_{\mathsf{h}} = \texttt{ZKPoSP-ed25519 hardened}$.

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

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

If the third-highest bit (bit 253) of $k_{L_0}^{\mathrm{raw}}$ 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 derivation

key $d_0 \coloneqq (k_{L_0}, k_{R_0})$ 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 $c_0 \coloneqq \mathsf{SHA256}(\mathtt{0x01} \| x_0) \in \{0,1\}^{256}$ (public) and the root witness key $w_0 \coloneqq \mathsf{SHA256}(\mathsf{tag}_0 \| x_0)$, and set $\mathsf{sk}_0 \coloneqq (d_0, w_0)$. 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 $d_{n-1} = (k_{L_{n-1}}, k_{R_{n-1}})$, parent witness key w_{n-1}, parent node state $\mathsf{st}_{n-1} = (A_{n-1}, c_{n-1}, i_{n-1})$, and child index i_n, compute

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

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):

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

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 witness key is $w_n \coloneqq \mathsf{SHA256}(w_{n-1} \| \langle i_n \rangle_4)$. The child derivation key is $d_n \coloneqq (k_{L_n}, k_{R_n})$, the child signing key is $\mathsf{sk}_n \coloneqq (d_n, w_n)$, and the child node state is $\mathsf{st}_n \coloneqq (A_n, c_n, i_n)$.

Hardened child step ($i_n \geq 2^{31}$). Given the parent derivation key $d_{n-1} = (k_{L_{n-1}}, k_{R_{n-1}})$, 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}, and A_n identically to the non-hardened case, and derive the child chain code $c_n \coloneqq \mathsf{HMAC\text{-}SHA512}(c_{n-1},\, \mathtt{0x01} \| k_{L_{n-1}} \| k_{R_{n-1}} \| \langle i_n \rangle_4)[32:]$. 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. The child witness key is $w_n \coloneqq \mathsf{SHA256}(\mathsf{tag_h} \| k_{L_{n-1}} \| k_{R_{n-1}} \| c_{n-1} \| \langle i_n \rangle_4)$. The child derivation key is $d_n \coloneqq (k_{L_n}, k_{R_n})$, the child signing key is $\mathsf{sk}_n \coloneqq (d_n, w_n)$, 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 $d_{n-1} = (k_{L_{n-1}}, k_{R_{n-1}})$ and $\mathsf{st}_{n-1} = (A_{n-1}, c_{n-1}, i_{n-1})$; take the index i_n of v_n from T; apply the non-hardened child step; return d_n, w_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 $d_{n-1} = (k_{L_{n-1}}, k_{R_{n-1}})$ and $\mathsf{st}_{n-1} = (A_{n-1}, c_{n-1}, i_{n-1})$; take the index i_n of v_n from T; apply the hardened child step; return d_n, w_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 $d_{n-1} = (k_{L_{n-1}}, k_{R_{n-1}})$ and $\mathsf{st}_{n-1} = (A_{n-1}, c_{n-1}, i_{n-1})$; apply the hardened child step; return A_n and $\mathsf{st}_n := (A_n, c_n, i_n)$.
- $\mathsf{Sign}(\mathsf{sk}_n, M)$: parse $\mathsf{sk}_n = ((k_{L_n}, k_{R_n}), w_n)$; produce a classical Ed25519 signature under k_{L_n} using k_{R_n} as the nonce seed. In ZKPoSP mode, Sign computes $h_w := \mathsf{SHA256}(w_n)$ and generates the signing proof π_{sign} for (M, h_w) as described in Section 5, and outputs $\sigma := (\pi_{\mathsf{sign}}, h_w)$.
- $\mathsf{DCert}(\mathsf{pp}, d_h, w_h, \mathsf{st}_h, v_h, v_n, a, F)$: in classical mode, $F = \varepsilon$ and the certificate is empty; in ZKPoSP mode, $a := x_0$ is the root seed for full-chain proofs (or the private data of the parent of the hardened anchor for pruned proofs, as described in Section 6.1), 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.
- $\mathsf{SignVerify}(\mathsf{pp}, A_n, M, \sigma)$: in classical mode, return 1 iff σ is a valid Ed25519 signature for M under A_n; in ZKPoSP mode, return 1 iff π_{sign} verifies for (M, h_w).
- $\mathsf{Verify}(\mathsf{pp}, A_n, \tau_n, M, \sigma, F_{\mathsf{ver}})$: 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 $\mathsf{SignVerify}(\mathsf{pp}, A_n, M, \sigma) = 1$, and the value h_w is the same in τ_n and σ.

Quantum vulnerability. Since $A_n = k_{L_n} \cdot B$, a QPT adversary recovers k_{L_n} from A_n with Shor's algorithm [Sho94], so BIP32-Ed25519 signatures are not unforgeable against quantum adversaries.

4 Monolithic Post-Quantum Signing for BIP32-Ed25519

As a didactic stepping stone toward the other constructions presented later, we show how 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. We call this scheme Σ. We notice that 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, in this section we restrict our analysis to non-hardened derivation paths.

In this scheme we bind the message to the derived key 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, 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:

$$\mathcal{R} := \left\{ \big((M,\, A_n),\, \mathbf{w}\big) \;\middle|\; \mathbf{w} = (x_0,\, i_1, \ldots, i_n,\, k_{L_n},\, k_{R_n},\, h) \text{ and (i)–(v) hold} \right\},$$

where $x_0 \in \{0,1\}^{256}$, $k_{L_n}, k_{R_n} \in \{0,1\}^{256}$, $h \in \{0,1\}^{256}$, and the indices $(i_1, \ldots, i_n)$ are part of the private witness:

(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: $(M, A_n) \in \mathcal{M} \times \mathbb{G}$. Witness: $\mathbf{w} = (x_0,\ i_1, \ldots, i_n,\ k_{L_n},\ k_{R_n},\ h)$.

$\Sigma = (\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ 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 $\mathbf{w} \coloneqq (x_0,\ i_1, \ldots, i_n,\ k_{L_n},\ k_{R_n},\ h)$; 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* $M \in \mathcal{M}$*:*

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

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 $\mathbf{w} = (x_0,\ i_1, \ldots, i_n,\ k_{L_n},\ k_{R_n},\ h)$ 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)**.** *Let SHA-256, SHA-512, and HMAC-SHA512 be modelled as quantum random oracles* $\mathcal{O}_{\mathsf{256}}$*,* $\mathcal{O}_{\mathsf{512}}$*, and* $\mathcal{O}_{\mathsf{H}}$*, and let* NIZK *satisfy Assumption 3.10 with zero-knowledge error* $\varepsilon_{\mathsf{zk}}$ *and extraction error* $\varepsilon_{\mathsf{ext}}$*. Then for every QPT adversary* $\mathcal{A}$ *that makes at most* Q *quantum oracle queries, including those made by the extractor, the probability that* $\mathcal{A}$ *wins the EUF-CMA game for* Σ *is at most*

$$\varepsilon_{\mathsf{zk}} + \varepsilon_{\mathsf{ext}} + O\big((Q+1) \cdot 2^{-128}\big) + O\big((Q + 2n + 3)^2 \cdot 2^{-224}\big).$$

A classical adversary is a special case of a quantum one, so the theorem also holds against PPT adversaries in the classical ROM.

Proof. **Game G_0 (real game).** The challenger runs $\mathsf{Gen}(1^\lambda)$ to obtain $(\mathsf{sk}_n, \mathsf{pk}_n)$ with $\mathsf{sk}_n = (x_0, k_{L_n}, k_{R_n})$, and gives $\mathsf{pk}_n = A_n$, pp, and the public chain $(A_0, \ldots, A_{n-1}, c_0, \ldots, c_{n-1})$ to $\mathcal{A}$. For each signing query M_i, the challenger computes $h_i := \mathcal{O}_{\mathsf{256}}(k_{R_n} \| M_i)$ and returns $\sigma_i \leftarrow \mathsf{NIZKProve}(\mathsf{pp}, (M_i, A_n), \mathbf{w}_i)$, where $\mathbf{w}_i = (x_0, i_1, \ldots, i_n, k_{L_n}, k_{R_n}, h_i)$. $\mathcal{A}$ wins if it outputs (M^*, σ^*) with $\mathsf{NIZKVerify}(\mathsf{pp}, (M^*, A_n), \sigma^*) = 1$ and $M^* \notin \{M_1, \ldots, M_q\}$.

Game G_1 (simulated proofs). As G_0, but each σ_i is produced by the simulator, $\sigma_i \leftarrow \mathsf{Sim}_P(\mathsf{pp}, (M_i, A_n))$, and the queries of $\mathcal{A}$ to the random oracle of NIZK are answered by Sim_H.

Lemma 4.3. $|\Pr[\mathcal{A} \textit{ wins } G_0] - \Pr[\mathcal{A} \textit{ wins } G_1]| \le \varepsilon_{\mathsf{zk}}$.

Proof. A distinguisher against the zero-knowledge property of NIZK runs $\mathcal{A}$, forwards each signing query M_i to its proof oracle with statement (M_i, A_n), forwards each query of $\mathcal{A}$ to the random oracle of NIZK to its random oracle, and answers all other oracle queries honestly; it simulates G_0 with the real prover and G_1 with the simulator. □

Bounding $\Pr[\mathcal{A}$ wins $G_1]$. By Lemma 4.3, it remains to show that $\Pr[\mathcal{A} \text{ wins } G_1] \le \varepsilon_{\mathsf{ext}} + O((Q+1)\cdot 2^{-128}) + O((Q+2n+3)^2 \cdot 2^{-224})$. Since $M^* \notin \{M_1, \ldots, M_q\}$, the statement (M^*, A_n) was never given to Sim_P. By Assumption 3.10, except with probability $\varepsilon_{\mathsf{ext}}$, the extractor outputs a witness $\mathbf{w}^* = (x_0^*, i_1^*, \ldots, i_n^*, k_{L_n}^*, k_{R_n}^*, h^*)$ for (M^*, A_n). Let n be the derivation depth, and let P^* be the $2n+2$ oracle inputs evaluated by conditions (i) and (ii) for $\mathbf{w}^*$, i.e., the two root inputs x_0^* and $\mathtt{0x01}\|x_0^*$, and two HMAC inputs per step. Let P° be the same inputs for the honest witness.

Case 1: $x_0^* = x_0$. The adversary output the seed. Its view depends on x_0 only through the oracle answers at x_0 and $\mathtt{0x01}\|x_0$, so it either guessed x_0, with probability 2^{-256}, or inverted a quantum random oracle. By [AHU19], it happens with probability $O((Q+1)\cdot 2^{-128})$.

Case 2: $x_0^* \neq x_0$. By condition (iv), $k_{L_n}^* \cdot B = A_n$ with $k_{L_n}^* = k_{L_{n-1}}^* + 8 \cdot z_{L_n}^*$, where $z_{L_n}^*$ comes from the answer at the last input of P^*. If this input equals the honest one, there should exist a derivation level in which the adversary inputs P^* with a value different from the honest one but obtaining the same chain code, finding a second preimage, which has probability at most 2^{-256} per input. Otherwise, its answer must equal the fixed residue of k_{L_n} modulo ℓ, which has probability at most 2^{-224} per input. In both cases Lemma 3.3 gives probability $O((Q+2n+3)^2 \cdot 2^{-224})$.

Summing the extraction error and the two cases bounds $\Pr[\mathcal{A}$ wins $G_1]$, and Lemma 4.3 gives the bound of the theorem. □

Remark 4.4 (Why starting from the seed). On a purely non-hardened path, the only value the adversary ignores is the seed x_0. The proof therefore ends either with the adversary outputting x_0 (Case 1) or it found a different preimage for a value output by the oracle (Case 2). Once a hardened step is present, the proof can start at the parent of the last hardened node instead (Section 6).

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 we restrict 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

$h_w = \mathsf{SHA256}(w_n)$, where w_n is the witness key of the node, which appears in the statement of both proofs to ensure that both refer to the same w_n.[3]

The witness key is not part of BIP32-Ed25519: the wallet computes it with SHA-256 from values it already holds, and it changes no private key, public key, chain code, or address. At the root, $w_0 \coloneqq \mathsf{SHA256}(\mathsf{tag}_0 \| x_0)$, where tag_0 is the ASCII string `ZKPoSP-ed25519 root`, and at a non-hardened step with index i_i, $w_i \coloneqq \mathsf{SHA256}(w_{i-1} \| \langle i_i \rangle_4)$. Unlike k_{R_i}, which a child obtains from its parent by adding a public tweak, w_i is obtained by a one-way function, so the witness key of a node does not reveal the witness keys of its parent or of its siblings.

The first part is the derivation proof π_{deriv} that proves that A_n is correctly derived from a seed following the BIP32-Ed25519 rules, and that h_w is the hash of the w_n produced by that derivation. This proof is generated once per key pair in the derivation tree and can be reused for every signature made with that pair.

The second part is the signing proof π_{sign} that proves that the signer holds a value w_n consistent with the public h_w, and that the signature hash h was computed correctly from w_n and the message M.

The derivation and signing procedures are independent, therefore 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.

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_w is the hash of the witness key w_n derived along the same path:

$$\mathcal{R}_{\mathsf{deriv}} \coloneqq \left\{ \big((A_n,\, h_w),\, \mathbf{w}_1 \big) \;\middle|\; \mathbf{w}_1 = (x_0,\, i_1, \ldots, i_n,\, k_{L_n},\, w_n) \text{ and (D-i)–(D-v) hold} \right\}.$$

Public statement: $(A_n,\, h_w) \in \mathbb{G} \times \{0,1\}^{256}$.
Private witness: $\mathbf{w}_1 = (x_0,\, i_1, \ldots, i_n,\, k_{L_n},\, w_n)$. The indices $(i_1, \ldots, i_n)$ are part of the private witness, so the proof does not reveal the derivation path.

(D-i) **Root.** $k_{L_0} \| k_{R_0} \coloneqq \mathsf{Tweak}(\mathsf{SHA512}(x_0))$, $c_0 \coloneqq \mathsf{SHA256}(\mathtt{0x01} \| x_0)$, $w_0 \coloneqq \mathsf{SHA256}(\mathsf{tag}_0 \| x_0)$.

(D-ii) **Child steps.** For $i = 1, \ldots, n$, take the index i_i from the witness 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

$$k_{L_i} \coloneqq k_{L_{i-1}} + 8 \cdot z_L^{(i)} \pmod{2^{256}}, \qquad w_i \coloneqq \mathsf{SHA256}(w_{i-1} \| \langle i_i \rangle_4),$$
$$c_i \coloneqq \mathsf{HMAC\text{-}SHA512}(c_{i-1},\, \mathtt{0x03} \| A_{i-1} \| \langle i_i \rangle_4)[32:].$$

(D-iii) **Binding.** $k_{L_n} = k_{L_n}^{\mathsf{derived}}$ and $w_n = w_n^{\mathsf{derived}}$, that is, both values 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.** $h_w = \mathsf{SHA256}(w_n)$.

These conditions enforce that both k_{L_n} and w_n are correctly derived from x_0, and that h_w is the correct hash of w_n, so neither can be freely chosen by a forger independently of A_n.

[3] Publishing h_w does not weaken security since the adversary cannot recover w_n from h_w except with negligible probability.

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

The signing relation captures the statement that the signer knows a witness key w_n consistent with the hash binding h_w, and that the signature hash h was honestly computed from w_n and the message M:

$$\mathcal{R}_{\mathsf{sign}} \coloneqq \left\{ \big((M,\, h_w),\, \mathbf{w}_2 \big) \;\middle|\; \mathbf{w}_2 = (w_n,\, h) \text{ and (S-i)–(S-ii) hold} \right\}.$$

Public statement: $(M,\, h_w) \in \mathcal{M} \times \{0,1\}^{256}$.
Private witness: $\mathbf{w}_2 = (w_n,\, h) \in \{0,1\}^{256} \times \{0,1\}^{256}$.

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

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_w, which must be identical in both proofs:

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

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

(C-i) **Derivation proof.** $\mathsf{NIZKVerify}\big(\mathsf{pp},\, (A_n,\, h_w),\, \pi_{\mathsf{deriv}}\big) = 1$, certifying that $\big((A_n,\, h_w),\, \mathbf{w}_1\big) \in \mathcal{R}_{\mathsf{deriv}}$ for some $\mathbf{w}_1$.
(C-ii) **Signing proof.** $\mathsf{NIZKVerify}\big(\mathsf{pp},\, (M,\, h_w),\, \pi_{\mathsf{sign}}\big) = 1$, certifying that $\big((M,\, h_w),\, \mathbf{w}_2\big) \in \mathcal{R}_{\mathsf{sign}}$ for some $\mathbf{w}_2$, where h_w is the same value appearing in condition (C-i).

The verifier additionally checks that the value h_w in the derivation proof statement is equal to the h_w in π_{sign}'s public statement binding the signing proof to the derivation proof. In practice, the verifier does not compare two values: it takes h_w from the statement of π_{deriv} and uses the same value as public input when it checks π_{sign}, as in the Verify algorithm of Section 5.4.

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, extended with the witness key w_i.

$\mathsf{Set}(1^\lambda, T, S)$ runs as in Section 3.11 with seed $S = x_0$: it computes $k_{L_0} \| k_{R_0} \coloneqq \mathsf{Tweak}(\mathsf{SHA512}(x_0))$, $c_0 \coloneqq \mathsf{SHA256}(\mathtt{0x01} \| x_0)$, and $w_0 \coloneqq \mathsf{SHA256}(\mathsf{tag}_0 \| x_0)$, then derives $(k_{L_i}, k_{R_i}, c_i, w_i)$ 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 $d_i \coloneqq (k_{L_i}, k_{R_i})$, the witness key w_i, and the signing key $\mathsf{sk}_i \coloneqq (d_i, w_i)$.

The value k_{R_i} is used for hardened derivation and for classical Ed25519 signatures, and plays no role in the ZKPoSP proofs. 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 $\mathsf{sk}_i = (d_i, w_i)$ and st_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 $(d_0, w_0, \mathsf{st}_0, a) = ((k_{L_0}, k_{R_0}), w_0, \mathsf{st}_0, x_0)$.

$\mathsf{DCert}(\mathsf{pp}, d_0, w_0, \mathsf{st}_0, v_0, v_n, a, F)$: parse $d_0 = (k_{L_0}, k_{R_0})$, $a = x_0$. 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}, w_n). Compute $h_w \coloneqq \mathsf{SHA256}(w_n)$ and $\pi_{\mathsf{deriv}} \leftarrow \mathsf{NIZKProve}(\mathsf{pp}, (A_n, h_w), (x_0, i_1, \ldots, i_n, k_{L_n}, w_n))$. Output $\tau_n \coloneqq (\pi_{\mathsf{deriv}}, h_w)$.

$\mathsf{DVerify}(\mathsf{pp}, \mathsf{pk}_n, \tau_n, F_{\mathsf{ver}})$: parse $\mathsf{pk}_n = A_n$, $\tau_n = (\pi_{\mathsf{deriv}}, h_w)$. 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 $\mathsf{NIZKVerify}(\mathsf{pp}, (A_n, h_w), \pi_{\mathsf{deriv}}) = 1$.

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

$\mathsf{SignVerify}(\mathsf{pp}, \mathsf{pk}_n, M, \sigma)$: parse $\sigma = (\pi_{\mathsf{sign}}, h_w)$. 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 $\mathsf{NIZKVerify}(\mathsf{pp}, (M, h_w), \pi_{\mathsf{sign}}) = 1$.

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

Theorem 5.1 (Correctness of ZKPoSP)**.** *Let* NIZK *be perfectly complete (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 3.13: 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 $\mathsf{sk}_j = ((k_{L_j}, k_{R_j}), w_j)$. By construction of Sign, $\sigma \coloneqq \mathsf{Sign}(\mathsf{sk}_j, M) = (\pi_{\mathsf{sign}}, h_w)$, where $h_w \coloneqq \mathsf{SHA256}(w_j)$, $h \coloneqq \mathsf{SHA256}(w_j \| M)$, and $\pi_{\mathsf{sign}} \leftarrow \mathsf{NIZKProve}(\mathsf{pp}, (M, h_w), (w_j, h))$ under $\mathcal{R}_{\mathsf{sign}}$. By construction of DCert, $\tau_j = (\pi_{\mathsf{deriv}}, h_w)$ with $\pi_{\mathsf{deriv}} \leftarrow \mathsf{NIZKProve}(\mathsf{pp}, (A_j, h_w), (x_0, i_1, \ldots, i_j, k_{L_j}, w_j))$ under $\mathcal{R}_{\mathsf{deriv}}$, using the same h_w, since DCert is invoked with $d_0 = (k_{L_0}, k_{R_0})$, w_0, anchor witness $a = x_0$, and applies DPriv along the purely non-hardened path from v_0 to v_j.

The witness $\mathbf{w}_1 = (x_0, i_1, \ldots, i_j, k_{L_j}, w_j)$ 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, (D-iv) holds since $A_j = k_{L_j} \cdot B$ by definition of key generation, and (D-v) holds by construction of h_w. The witness $\mathbf{w}_2 = (w_j, h)$ satisfies conditions (S-i)–(S-ii) of $\mathcal{R}_{\mathsf{sign}}$: (S-i) holds since both occurrences of h_w are identical by construction, and (S-ii) holds by construction of h.

By the completeness property of NIZK (Section 3.6), $\mathsf{NIZKVerify}(\mathsf{pp}, (A_j, h_w), \pi_{\mathsf{deriv}}) = 1$ and $\mathsf{NIZKVerify}(\mathsf{pp}, (M, h_w), \pi_{\mathsf{sign}}) = 1$. 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 $\sigma = (\pi_{\mathsf{sign}}, h_w)$ and $\tau_j = (\pi_{\mathsf{deriv}}, h_w)$ carry the same h_w by construction, the equality check $h_w = h'_w$ in Verify holds trivially. Therefore all three conditions of Verify hold. □

Theorem 5.2 (Hierarchical EUF-CMA of ZKPoSP)**.** *Let SHA-256, SHA-512, and HMAC-SHA512 be modelled as quantum random oracles $\mathcal{O}_{\mathsf{256}}$, $\mathcal{O}_{\mathsf{512}}$, and $\mathcal{O}_{\mathsf{H}}$, and let* NIZK *satisfy Assumption 3.10 with zero-knowledge error $\varepsilon_{\mathsf{zk}}$ and extraction error $\varepsilon_{\mathsf{ext}}$. Let $T = (V, E)$ be a tree in which every node is non-hardened, with depth at most n. Then for every QPT adversary $\mathcal{A}$ that makes at most q_S signing queries and at most Q quantum oracle queries, including those made by the extractor, the probability that $\mathcal{A}$ wins $G^{\mathrm{heuf}}_{\Pi,\mathcal{A}}(\lambda, T)$ of Definition 3.14 for ZKPoSP for BIP32-Ed25519 is at most*

$$2\varepsilon_{\mathsf{zk}} + \varepsilon_{\mathsf{ext}} + O\big((Q+1)\sqrt{n+4}\cdot 2^{-128}\big) + O\big((Q+2|V|+2n+3)^2\cdot 2^{-224}\big).$$

Proof. **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. Since we are assuming that all nodes are non-hardened, $\mathcal{A}$ receives at setup $\mathsf{st}_i = (A_i, c_i, i_i)$ for each node. $\mathcal{O}_{\mathrm{Corr}}(v_i)$ returns $(d_i, w_i, \mathsf{st}_i)$, and $\mathcal{O}_{\mathrm{Sign}}(M, v_i)$ returns (A_i, τ_i, σ_i). $\mathcal{A}$ wins if it outputs $(v^*, M^*, \sigma^*, \tau^*)$ with $\mathsf{Verify}(\mathsf{pp}, \mathsf{pk}_{v^*}, \tau^*, M^*, \sigma^*, F_{\mathsf{ver}}) = 1$, $v^* \notin Q_{\mathrm{Corr}}$, and $(M^*, v^*) \notin Q_{\mathrm{Sign}}$.

Game G_1 (simulated derivation and signing proofs). As G_0, but every $\pi_{\mathsf{deriv},i}$ and $\pi_{\mathsf{sign},i}$ returned by $\mathcal{O}_{\mathrm{Sign}}$ is produced by Sim_P for the corresponding statement, and the queries of $\mathcal{A}$ to the random oracle of the NIZK are answered by Sim_H.

Lemma 5.3. $|\Pr[\mathcal{A}\ \textit{wins}\ G_0] - \Pr[\mathcal{A}\ \textit{wins}\ G_1]| \le 2\varepsilon_{\mathsf{zk}}$.

Proof. Identical argument to Lemma 4.3, applied once to the proofs for $\mathcal{R}_{\mathsf{deriv}}$ and once to the proofs for $\mathcal{R}_{\mathsf{sign}}$; each application covers all q_S signing queries. □

Bounding $\Pr[\mathcal{A}$ wins $G_1]$. By Lemma 5.3, it remains to bound $\Pr[\mathcal{A}$ wins $G_1]$.

Given $\sigma^* = (\pi^*_{\mathsf{sign}}, h^*)$ and $\tau^* = (\pi^*_{\mathsf{deriv}}, h^{*\prime})$, it holds that Verify accepts, that $h^* = h^{*\prime}$, $\mathsf{NIZKVerify}(\mathsf{pp}, (A_{v^*}, h^*), \pi^*_{\mathsf{deriv}}) = 1$ and $\mathsf{NIZKVerify}(\mathsf{pp}, (M^*, h^*), \pi^*_{\mathsf{sign}}) = 1$. Let $h_{v^*} \coloneqq \mathcal{O}_{\mathsf{256}}(w_{v^*})$, where w_{v^*} is the witness key of v^* output by Set. Distinct nodes have distinct witness keys w_v, distinct values $h_v = \mathcal{O}_{\mathsf{256}}(w_v)$, and distinct public keys, except with probability $O(|V|^2 \cdot 2^{-224})$.

Case 1: $h^* = h_{v^*}$. The statement (M^*, h_{v^*}) was never given to Sim_P, since $(M^*, v^*) \notin Q_{\mathrm{Sign}}$ and no other node has the value h_{v^*}. By Assumption 3.10, except with probability $\varepsilon_{\mathsf{ext}}$, the extractor outputs a witness (w^*, h) for (M^*, h_{v^*}), and by condition (S-i), $\mathcal{O}_{\mathsf{256}}(w^*) = h_{v^*}$. If $w^* = w_{v^*}$, the adversary output the witness key of v^*. Let $\mathcal{I}$ be the set of oracle inputs that are $\mathtt{0x01}\|x_0$ or $\mathsf{tag}_0\|x_0$, or that start with x_0 or with the witness key of v^* or of one of its ancestors; a fixed input lies in $\mathcal{I}$ with probability at most $(n+4)\cdot 2^{-256}$. Since no ancestor of v^* was corrupted, the corrupted witness keys and all other values seen by the adversary depend on w_{v^*} only through the oracle answers on $\mathcal{I}$. So the adversary either guessed w_{v^*}, with probability 2^{-256}, or found it from these answers, that is, inverted a quantum random oracle. By [AHU19], this has probability $O((Q+1)\sqrt{n+4}\cdot 2^{-128})$. If $w^* \neq w_{v^*}$, the adversary found a second preimage of the fixed value h_{v^*}, which has probability at most 2^{-256} per input, and Lemma 3.3, counting the $|V|$ honest witness keys as extra queries, gives probability $O((Q+|V|+1)^2 \cdot 2^{-256})$.

Case 2: $h^* \neq h_{v^*}$. The statement (A_{v^*}, h^*) was never given to Sim_P. By Assumption 3.10, except with probability $\varepsilon_{\mathsf{ext}}$, the extractor outputs a witness $(x^*_0, \mathbf{i}^*, k^*_{L_{v^*}}, w^*)$ for (A_{v^*}, h^*), where $\mathbf{i}^*$ are its indices. If $x^*_0 = x_0$, the adversary output the seed; the oracle inputs that depend on x_0 are in $\mathcal{I}$, so this has the probability bound of Case 1. So $x^*_0 \neq x_0$, and the argument of Case 2 in the proof of Theorem 4.2 applies to the path from the root to v^*: the adversary found a second preimage of a chain code of the honest chain, or an oracle answer equal to the fixed residue of $k_{L_{v^*}}$ modulo ℓ. Lemma 3.3, counting the $2|V|$ honest HMAC inputs and the root inputs as extra queries, gives probability $O((Q+2|V|+2n+3)^2 \cdot 2^{-224})$.

Summing all these probabilities with Lemma 5.3 gives the bound of the theorem. □

6 Can the Derivation Proof Be Shortened?

The proving times reported in Section 9.1 grow linearly with the derivation depth n. For this reason, in this section we show a new derivation proof, which we call last-step proof, that proves the derivation for a single parent-to-child step, and we show when it is secure.

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:

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

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: $(d_{n-1},\, w_{n-1},\, \mathsf{st}_{n-1},\, i_n,\, d_n,\, w_n)$, the parent and child derivation and witness keys and the child index, together with the parent node state (which contains the parent chain code in the schemes that use one).

(L-i) **Derivation step.** (d_n, w_n) is correctly derived from $(d_{n-1}, w_{n-1}, \mathsf{st}_{n-1})$ and the 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.

6.1 When Last-Step Certificates Can Be Used

A last-step certificate can be used in an HDW instantiation only if the resulting instantiation achieves hierarchical EUF-CMA (Definition 3.14). For non-hardened steps this is false, and we give a counterexample for each scheme. For a hardened step, let a be the private data of the parent v_{n-1} containing its derivation key, its witness key, and its chain code. If the adversary cannot compute a, then a plays the role of the seed for the subtree of v_{n-1} and every private key, chain code, and witness key of this subtree is computed from a using the derivation procedure. The proof of Theorem 5.2 then applies with a in place of the seed (Theorem 6.1).

Non-hardened steps. Let v_n be a non-hardened node whose parent v_{n-1} is reached from the root by non-hardened steps only, so that the adversary receives at setup the node state st_{n-1}, which contains the public key and the chain code of v_{n-1}. A QPT adversary recovers the derivation key d_{n-1} from the public key of v_{n-1} with Shor's algorithm, and computes the keyed hash of the step from the chain code and the public key of v_{n-1}, which gives d_n. The parent witness key is not an input of this keyed hash, so the adversary picks any $w^*_{n-1} \xleftarrow{\$} \{0,1\}^{256}$, sets $w^*_n = \mathsf{SHA256}(w^*_{n-1} \| \langle i_n \rangle_4)$ and $h^* = \mathsf{SHA256}(w^*_n)$, and proves $\mathcal{R}_{\mathsf{last}}$ for (pk_n, h^*) with witness $(d_{n-1}, w^*_{n-1}, \mathsf{st}_{n-1}, i_n, d_n, w^*_n)$. Since it knows w^*_n, it also proves the signing relation for (M^*, h^*) for any message M^*. It outputs v_n, M^*, the signing proof, and the derivation proof with h^*: Verify accepts, and v_n was neither corrupted nor queried, so the adversary wins with probability $1 - \mathsf{negl}(\lambda)$ without any corruption or signing query. This counterexample applies to BIP32 secp256k1 and BIP32-Ed25519, with the witness keys of Sections 3.10 and 5, and to QBIP32 (Section 7).

Hardened steps.

Theorem 6.1 (Last-step certificates for hardened nodes)**.** *Let SHA-256, SHA-512, and HMAC-SHA512 be modelled as quantum random oracles, and let* NIZK *satisfy Assumption 3.10 with zero-knowledge error $\varepsilon_{\mathsf{zk}}$ and extraction error $\varepsilon_{\mathsf{ext}}$. Consider BIP32 secp256k1, BIP32-Ed25519, or SLIP-10/Ed25519, and let $T = (V, E)$ be a tree of depth at most n in which every child of the root is*

hardened and every node with a last-step certificate is hardened. Then for every QPT adversary $\mathcal{A}$ that makes at most q_S signing queries and at most Q quantum oracle queries, including those made by the extractor, the probability that $\mathcal{A}$ wins $G^{\mathrm{heuf}}_{\Pi,\mathcal{A}}(\lambda, T)$ of Definition 3.14 with last-step certificates is at most

$$2\varepsilon_{\mathsf{zk}} + \varepsilon_{\mathsf{ext}} + O\big((Q+1)\sqrt{n+4}\cdot 2^{-128}\big) + O\big((Q+2|V|+2n+3)^2\cdot 2^{-224}\big).$$

Proof. Identical argument to Theorem 5.2, with the private data a of the parent of v^* in place of the seed x_0. Since every child of the root is hardened, no chain code is given at setup, and since $v^* \notin Q_{\mathrm{Corr}}$, no ancestor of v^* is corrupted, so a is never given to $\mathcal{A}$. A corrupted node reveals only its own chain code and those of its descendants, so $\mathcal{A}$ does not know the chain code of any ancestor of v^*. A QPT adversary can recover the derivation key in a with Shor's algorithm, but the chain code and, for BIP32-Ed25519, the value $k_{R_{n-1}}$ remain unknown, and they are inputs of the keyed hash of the step. For BIP32-Ed25519, $k_{R_{n-1}}$ cannot be recovered from a corrupted non-hardened child either, since this requires the chain code of the parent. In Case 1, the set $\mathcal{I}$ also contains the inputs of the step from the parent of v^* and of the steps above it. In Case 2, if the extracted parent data equals a and the extracted index is that of v^*, then the extracted witness key is that of v^* and $h^* = h_{v^*}$, which contradicts this case; otherwise the argument of Case 2 of Theorem 4.2 applies to the step from the parent, with probability at most 2^{-224} per query for the leaf scalar. □

The same holds when the certificate covers the hardened step and a non-hardened suffix below it, since every value of the suffix is also computed from a. For BIP32 secp256k1 and BIP32-Ed25519, the theorem needs every child of the root to be hardened. If the root has a non-hardened child, $\mathcal{A}$ receives c_0 and recovers the root derivation key with Shor's algorithm (for BIP32-Ed25519, it also recovers k_{R_0} by corrupting the non-hardened child), and then it computes every hardened step below the root, including the witness keys, so it can forge for every hardened node. In BIP44 paths every child of the root is hardened therefore no additional information is exposed. In SLIP-10/Ed25519 every child of the root is hardened, since all nodes are hardened, and there are no extended public keys.

6.2 Motivation for QBIP32

For the existing schemes, last-step certificates are secure for hardened keys only, and for BIP32 secp256k1 and BIP32-Ed25519 only if every child of the root is hardened. The reason for this condition is the input of a hardened step. In BIP32 secp256k1 and BIP32-Ed25519, the witness key of a hardened node is computed from the parent's derivation key and chain code (and k_R for BIP32-Ed25519). A QPT adversary recovers the derivation key with Shor's algorithm, so the step is protected only by the chain code (and k_R), and these values are public as soon as the root has a non-hardened child, or an extended public key of an ancestor is exported. In practice, this means that the security of a hardened key depends on how the rest of the wallet is used.

QBIP32 removes this dependence. A hardened step takes the parent witness key NW_{par} as input, and NW is never public: it is derived from the seed through one-way steps, and a corrupted non-hardened child reveals only $\mathsf{SHA256}(NW_{\mathsf{par}} \| \langle x \rangle_4)$. So last-step certificates for hardened keys stay secure even if the root has non-hardened children and extended public keys of ancestors are exported (Section 7.3).

QBIP32 has three further advantages. The witness key is an output of the derivation itself, computed in the same call as the signing scalar and the chain code, while for BIP32 secp256k1 and BIP32-Ed25519 it is an extra value outside the standard, which existing wallets do not compute. Each step uses a single call to a keyed extensible-output function, instantiated with KMAC256 (a

sponge-based construction defined in NIST SP 800-185 [Nat16]), instead of the two HMAC calls of BIP32-Ed25519. Finally, the same derivation logic gives both hardened and non-hardened derivation on secp256k1, Ed25519, and any other prime-order curve, while SLIP-10 has no non-hardened derivation on Ed25519 and BIP32-Ed25519 exists only for Ed25519. The cost is that QBIP32 is a new standard: its keys differ from those of existing wallets, so users must migrate, unlike with ZKPoSP over the existing schemes.

Table 1: Last-step certificates by scheme and key type. "No" means that there is a counterexample to hierarchical EUF-CMA; "Yes" means that hierarchical EUF-CMA holds; for BIP32 secp256k1 and BIP32-Ed25519 this requires that every child of the root is hardened (Theorem 6.1). QBIP32 is defined in Section 7.

Scheme	**Key type**	**Last-step only?**	**Reason**
BIP32 secp256k1	Non-hardened	No	w_{n-1} not a step input (Section 6.1)
BIP32 secp256k1	Hardened	Yes	parent data unknown (Thm. 6.1)
SLIP-10/Ed25519	Hardened	Yes	parent data unknown (Thm. 6.1)
BIP32-Ed25519	Non-hardened	No	w_{n-1} not a step input (Section 6.1)
BIP32-Ed25519	Hardened	Yes	parent data unknown (Thm. 6.1)
QBIP32	Non-hardened	No	NW_{par} not a step input (Section 7.3)
QBIP32	Hardened	Yes	parent data unknown (Section 7.3)

7 QBIP32: A Universal Sponge-Based HD Wallet

7.1 The HASH768 Primitive

As defined in Section 3.5, $\mathsf{HASH768} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^{768}$ 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 3.9.

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 keyed-function calls 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 (witness, private), NC (chain code, public). In the HDW framework of Section 3.8, the derivation key is $d_i := NS_i$, the witness key is $w_i := NW_i$, and the signing key is $\mathsf{sk}_i := (NS_i, NW_i)$. 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 D be the domain-separation string `"QBIP-32: secp256k1"` or `"QBIP-32:  ed25519"` depending on the underlying curve. Compute:

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

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 $\mathsf{HASH768}$ call with the same D (this requires $\mathsf{HASH768}$ to accept 768-bit keys, as KMAC256 does). This loop terminates within a bounded number of iterations with overwhelming probability.

Acceptance criteria for a child. A tweak is accepted if, for secp256k1, $T_S < q$ and $NS_{\mathsf{child}} \neq 0$, and, for Ed25519, $NS_{\mathsf{child}} \bmod \ell \neq 0$, where ℓ is the order of the base point.

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):

$$\begin{aligned}
NS_{\mathsf{child}} &\coloneqq \begin{cases} (NS_{\mathsf{par}} + T_S) \bmod q & \text{(secp256k1)}, \\ NS_{\mathsf{par}} + 8 \cdot T_{S,28} & \text{(Ed25519, plain integer addition)}, \end{cases} \\
NW_{\mathsf{child}} &\coloneqq \begin{cases} (NW_{\mathsf{par}} + T_W) \bmod 2^{256} & \text{(hardened step)}, \\ \mathsf{SHA256}(NW_{\mathsf{par}} \| \langle x \rangle_4) & \text{(non-hardened step)}, \end{cases} \\
NC_{\mathsf{child}} &\coloneqq T_C,
\end{aligned}$$

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 $K_{\mathsf{child}} = NS_{\mathsf{child}} \cdot G$. At a non-hardened step T_W is publicly computable, so NW_{child} is computed with a one-way function instead, and T_W is not used: the witness key of a node does not reveal the witness key of its parent or of its siblings. At a hardened step, recovering NW_{par} from NW_{child} requires T_W, which depends on the private NS_{par} and NW_{par}.

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 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. The proofs of this section use one property of f: for every fixed NS_{par} and every fixed $K \in \mathbb{G}$, a uniform T_S gives $f(NS_{\mathsf{par}}, T_S) \cdot G = K$ with probability at most 2^{-224}. The two instantiations above have this property, since T_S enters as a uniform value added to NS_{par} (for Ed25519, $8 \cdot T_{S,28}$, with 224 uniform bits). For a new curve, the guarantees of QBIP32 hold if f has this property, with 2^{-224} replaced by the bound of f.

7.3 NP Relations for Hardened Paths

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 $h_w = \mathsf{SHA256}(w_n)$ 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.

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

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 $\mathcal{R}^{\mathsf{hard}}_{\mathsf{sign}}$.

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

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 Certificates

The results of Section 6.1 hold also for QBIP32. For a non-hardened step, NW_{par} is not an input of the HASH768 call, so the counterexample of Section 6.1 applies, with NS and NW in place of d and w. For a hardened step, the private data $a = (NS_{\mathsf{par}}, NW_{\mathsf{par}}, NC_{\mathsf{par}})$ of the parent are all inputs of the HASH768 call. A QPT adversary recovers NS_{par} with Shor's algorithm, and NC_{par} may be public, but NW_{par} is not part of any public value, so a plays the role of the seed as in Theorem 6.1.

Theorem 7.2 (Hierarchical EUF-CMA of QBIP32 hardened-only)**.** *Let* HASH768 *and SHA-256 be modelled as quantum random oracles, and let* NIZK *satisfy Assumption 3.10 with zero-knowledge error $\varepsilon_{\mathsf{zk}}$ and extraction error $\varepsilon_{\mathsf{ext}}$. Let $T = (V, E)$ be a tree in which every node except the root is hardened, with depth at most n. Then for every QPT adversary $\mathcal{A}$ that makes at most q_S signing queries and at most Q quantum oracle queries, including those made by the extractor, the probability that $\mathcal{A}$ wins $G^{\mathrm{heuf}}_{\Pi,\mathcal{A}}(\lambda, T)$ of Definition 3.14 for QBIP32 hardened-only is at most*

$$2\varepsilon_{\mathsf{zk}} + \varepsilon_{\mathsf{ext}} + O\big((Q+1)\sqrt{n+2} \cdot 2^{-128}\big) + O\big((Q + |V| + 2)^2 \cdot 2^{-224}\big).$$

Proof. Identical argument to Theorem 5.2, with NW in place of w, h_{nw} in place of h_w, and HASH768 in place of HMAC-SHA512. In Case 1, $\mathcal{I}$ is the set of oracle inputs whose key is the seed, or whose data starts with $NS_{i-1} \| NW_{i-1}$ for a step i of the path from the root to v^*, or that contain NW_{v^*} as their first or last 32 bytes. Since no ancestor of v^* was corrupted, the corrupted nodes reveal

values that depend on the secrets of this path only through oracle answers on $\mathcal{I}$: at a hardened step, $NW_{\mathsf{child}} - T_W = NW_{\mathsf{par}}$, but T_W is the answer at an input that contains $NS_{\mathsf{par}} \| NW_{\mathsf{par}}$. In Case 2, the statement (K_{v^*}, h^*) was never given to Sim_P, and the extractor outputs a witness of $\mathcal{R}^{\mathsf{hard}}_{\mathsf{deriv}}$ for it. If its parent values are those of the challenger and its index is that of v^*, then $h^* = h_{nw}$ of v^*, which contradicts this case. Otherwise the argument of Case 2 of Theorem 4.2 applies to the step from the parent, and Lemma 3.3, counting the $|V|$ HASH768 inputs of the challenger as extra queries, gives probability $O((Q + |V| + 2)^2 \cdot 2^{-224})$. □

7.4 Pruned NP Relations

The pruned relations cover derivation paths containing a hardened prefix followed by a non-hardened suffix. As in Section 6.1, 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. In this subsection, index 0 denotes the anchor, not the root. Let $h_{nw} = \mathsf{SHA256}(NW_{\mathsf{leaf}})$.

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

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

(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** $j = 1, \ldots, k$. $K_{j-1} = NS_{j-1} \cdot G$; $(T_S \| T_W \| T_C) = \mathsf{HASH768}(NC_{j-1},\ K_{j-1} \| \langle x_j \rangle_4)$; $NS_j = f(NS_{j-1}, T_S)$, $NW_j = \mathsf{SHA256}(NW_{j-1} \| \langle x_j \rangle_4)$, $NC_j = T_C$.

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

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

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

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

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

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

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

Pruned HDW instantiation. The DCert, DVerify, Sign, SignVerify, and Verify algorithms are identical in structure to the hardened-only case, with $\mathcal{R}^{\mathsf{hard}}_{\mathsf{deriv}}$ 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 7.3 (Hierarchical EUF-CMA of QBIP32 pruned)**.** *Under the assumptions of Theorem 7.2, let $T = (V, E)$ be a tree of depth at most n in which every path from the root consists of hardened nodes followed by non-hardened nodes, and the children of the root are hardened. Then for every QPT adversary $\mathcal{A}$ that makes at most q_S signing queries and at most Q quantum oracle queries, including those made by the extractor, the probability that $\mathcal{A}$ wins $G^{\mathrm{heuf}}_{\Pi,\mathcal{A}}(\lambda, T)$ of Definition 3.14 for QBIP32 pruned is at most*

$$2\varepsilon_{\mathsf{zk}} + \varepsilon_{\mathsf{ext}} + O\big((Q+1)\sqrt{n+2}\cdot 2^{-128}\big) + O\big((Q+2|V|+2)^2\cdot 2^{-224}\big).$$

Proof. Identical argument to Theorem 7.2, with the pruned relations in place of the hardened-only ones. In Case 1, $\mathcal{I}$ also contains the inputs that start with NW_j for a non-hardened step j of the suffix, and the inputs that contain NW_{leaf} as their last 32 bytes. At a non-hardened step the witness key of the child is $\mathsf{SHA256}(NW_{\mathsf{par}} \| \langle x \rangle_4)$, the answer at an input in $\mathcal{I}$, so a corrupted node below v^*, or a corrupted sibling of v^*, reveals nothing more about NW_{v^*} than these answers. In Case 2, every value of the suffix is computed from the private data of the parent of the anchor, as in Theorem 6.1. □

7.5 Full-Path Relation

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 $c_{\mathsf{seed}} = \mathsf{SHA256}(r \| S)$, 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 $\mathcal{R}^{\mathsf{full\text{-}path}}_{\mathsf{deriv}}$.

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

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

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.** $c_{\mathsf{seed}} = \mathsf{SHA256}(r \| S)$.

(F-ii) **Root generation.** $(NS_0 \| NW_0 \| NC_0) = \mathsf{HASH768}(S,\, D)$, 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** $i = p{+}1, \ldots, p{+}t$**.** $K_{i-1} = NS_{i-1} \cdot G$; $(T_S \| T_W \| T_C) = \mathsf{HASH768}(NC_{i-1}, K_{i-1} \| \langle x_i \rangle_4)$; $NS_i = f(NS_{i-1}, T_S)$, $NW_i = \mathsf{SHA256}(NW_{i-1} \| \langle x_i \rangle_4)$, $NC_i = T_C$.

(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 $a = (S, r)$, and the tuple $(p, t, c_{\mathsf{seed}})$ added to the public statement of the derivation certificate.

Remark 7.4 (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 designed to resist quantum adversaries: the witness key (NW for QBIP32, w for BIP32-Ed25519 and BIP32-secp256k1, the chain code for SLIP-10/Ed25519) is computationally hidden, and no step of the signing security argument depends on the hardness of the discrete logarithm problem (see Section 3.3 for what is proved and what is conjectured). However, under the current threat model the leaf scalar NS_{leaf} and the scalars of its ancestors must be kept secret, because a classical adversary 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 should be rejected by the network, regardless of its classical validity. Under this post-Q-day model, exposing these scalars carries no additional risk, since an adversary who learns them cannot produce an accepted signature without the witness key anyway. This model shift enables a significant reduction in prover workload. The verifier recomputes $K_i = NS_i \cdot G$ externally for the leaf and for every node whose public key is an input of a non-hardened step; this operation is more efficient than checking the multiplication inside the NIZK and adds no measurable overhead to verification time. Hardened steps do not use any public key, so the scalars of the other nodes are not needed. 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, all scalar multiplications can be removed from the derivation proof and moved to an external verifier check. We write NS throughout for the signing scalar of a node; the concrete instantiation is k_L for BIP32-Ed25519, k for BIP32-secp256k1, the SLIP-10 secret key for SLIP-10/Ed25519, and NS 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 every scalar multiplication is removed from the circuit: condition (P-iii) and the computations $K_{j-1} = NS_{j-1} \cdot G$ in (P-ii) (or (F-v) and the computations $K_{i-1} = NS_{i-1} \cdot G$ in (F-iv) in the full-path case). The corresponding scalars and public keys are moved into the public statement, and the verifier checks $K = NS \cdot G$ for each of them outside the NIZK. The same transformation applies to the analogous split relations of BIP32-Ed25519, BIP32-secp256k1, and SLIP-10/Ed25519: in every case the scalar multiplications in the derivation relation are the public-key computations of the leaf and of the parents of the non-hardened steps, and moving these scalars into the public statement removes them. For SLIP-10/Ed25519, which has no non-hardened steps, only the leaf

scalar is moved. For the pruned relation, where index 0 is the anchor and indices $1, \ldots, k$ are the non-hardened suffix, the statement is $(K_0, \ldots, K_k, NS_0, \ldots, NS_k, h_{nw})$ with $K_k = K_{\mathsf{leaf}}$; for the full-path relation, where index 0 is the root, it is $(p, t, K_p, \ldots, K_{p+t}, NS_p, \ldots, NS_{p+t}, h_{nw}, c_{\mathsf{seed}})$ with $K_{p+t} = K_n$. The scheme-specific binding hash (h_{nw} for QBIP32, h_w for BIP32-Ed25519 and BIP32-secp256k1, h_c for SLIP-10/Ed25519) is unchanged from the pre-Q-day relation. The signing proof is unchanged, since it does not involve scalar multiplication: $\mathcal{R}_{\mathsf{sign}}^{\mathsf{pruned}}$ in the pruned case and $\mathcal{R}_{\mathsf{sign}}^{\mathsf{hard}}$ in the full-path case.

Post-Q-day verifier.

1. Verify the derivation proof: check $\mathsf{NIZKVerify}(\mathcal{R}_{\mathsf{deriv}}^{\mathsf{qday}}, (K_0, \ldots, K_k, NS_0, \ldots, NS_k, h_{nw}), \pi_{\mathsf{deriv}}) = 1$.
2. Verify the signing proof: check $\mathsf{NIZKVerify}(\mathcal{R}_{\mathsf{sign}}^{\mathsf{pruned}}, (M, h_{nw}), \pi_{\mathsf{sign}}) = 1$.
3. Check cross-proof consistency: confirm that h_{nw} matches across both public statements.
4. Check the public keys classically: confirm $K_j = NS_j \cdot G$ for $j = 0, \ldots, k$, and $K_k = K_{\mathsf{leaf}}$.
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 $\mathcal{R}_{\mathsf{sign}}^{\mathsf{hard}}$, the indices $p, \ldots, p+t$, and K_n in place of $\mathcal{R}_{\mathsf{sign}}^{\mathsf{pruned}}$, the indices $0, \ldots, k$, 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.

9 Implementation and Benchmarks

All constructions are implemented in Rust with RISC Zero v5.0.0-rc.1. All proving times are measured on a dedicated OVH server (AMD Ryzen 9 9950X3D, 16 cores / 32 threads, 64 GB RAM, no GPU, Ubuntu). Proof size is about 219 KB and verification time is about 9–10 ms in every configuration, since both are fixed by the RISC Zero succinct receipt format, so the tables omit them.

The implementation follows the relations of Sections 5, 6, and 7 with one difference in the witness key, and we use it to estimate the cost of the constructions. In BIP32-Ed25519 the implemented witness key is the right half k_{R_n}, and in QBIP32 the implemented non-hardened update is $NW_{\mathsf{child}} = NW_{\mathsf{par}} + T_W \bmod 2^{256}$ instead of $\mathsf{SHA256}(NW_{\mathsf{par}} \| \langle x \rangle_4)$. In this construction the adversary that corrupts a node at level i can trivially reconstruct the private keys of the non-hardened ancestors. The implemented schemes are therefore secure in a weaker model, in which corrupting a node also corrupts every node connected to it by non-hardened steps. The schemes of this paper differ from the implemented ones by one SHA-256 evaluation per non-hardened step in the derivation circuit, and the signing circuit is the same.

9.1 Monolithic Proof: BIP32-Ed25519 Σ

The monolithic scheme Σ of Section 4 proves the whole chain from the seed in one STARK (Tables 2 and 3). Proving time grows linearly, $\mathsf{Prove}(n) \approx 61.7 \cdot n + 69.5$ seconds, from about 200 seconds at $n = 2$ to about 9138 seconds at $n = 147$, and peak RAM stays below 8.4 MB. The `derive_chain` component dominates, with about 1.18×10^6 cycles per level. This linear growth is what the split and pruned constructions remove.

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.

9.2 BIP44 Path Benchmarks

We benchmark the split constructions on the BIP44 path $m/44'/0'/0'/0/0$ and its prefixes. Derivation proving is a one-time cost per key pair, and sign proving is the cost per transaction. In Tables 5 to 8, the first row, whose leaf is the last hardened node, uses the hardened-only relations, and the other rows use the pruned relations.

SLIP-10/Ed25519 proves only the last hardened step, so derivation takes about 76 seconds at every depth (Table 4). BIP32-Ed25519, BIP32-secp256k1, and QBIP32 on both curves take about 156–158 seconds (Tables 5 to 8). On RISC Zero the proving cost is set by the number of padded segments, and these four circuits fall in the same segment configuration, so the single HASH768 call of QBIP32 gives no measurable advantage over the two HMAC calls of BIP32-Ed25519. Whether it does on a backend with native support for the Keccak permutation is left open; the contribution of QBIP32 is one derivation procedure for every prime-order curve.

QBIP32 full-path proves the whole chain from the seed, so derivation grows with the path length, up to 440 seconds for Ed25519 and 476 seconds for secp256k1 at the full BIP44 path (Tables 9 and 10). The measured circuit omits the seed commitment (F-i), which adds one SHA-256 evaluation.

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 $h_c = \mathsf{SHA256}(c_n)$.

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), measured with the hash binding $h_c = \mathsf{SHA256}(c_n)$ instead of $h_w = \mathsf{SHA256}(w_n)$. Row one implements $\mathcal{R}^{\mathsf{hard}}_{\mathsf{deriv}} + \mathcal{R}^{\mathsf{hard}}_{\mathsf{sign}}$; the rest implement $\mathcal{R}^{\mathsf{pruned}}_{\mathsf{deriv}} + \mathcal{R}^{\mathsf{pruned}}_{\mathsf{sign}}$.

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 $\mathcal{R}^{\mathsf{pruned}}_{\mathsf{deriv}} + \mathcal{R}^{\mathsf{pruned}}_{\mathsf{sign}}$.

9.3 Post-Q-Day Variant Benchmarks

In the post-Q-day variant of Section 8 the verifier computes all scalar multiplications, so the derivation cost depends only on the hash calls, and the two curves give the same times. QBIP32

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 $\mathcal{R}^{\mathsf{pruned}}_{\mathsf{deriv}} + \mathcal{R}^{\mathsf{pruned}}_{\mathsf{sign}}$.

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 $\mathcal{R}^{\mathsf{full\text{-}path}}_{\mathsf{deriv}} + \mathcal{R}^{\mathsf{hard}}_{\mathsf{sign}}$.

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 $\mathcal{R}^{\mathsf{full\text{-}path}}_{\mathsf{deriv}} + \mathcal{R}^{\mathsf{hard}}_{\mathsf{sign}}$.

and BIP32-Ed25519 take about 21 seconds when the leaf is the anchor and about 40 seconds with a non-hardened suffix, because the hash calls of the suffix move the circuit to a larger segment size (Tables 11, 12, and 13). BIP32-secp256k1, with one HMAC call per step, and SLIP-10/Ed25519 stay at about 21 seconds at every depth (Tables 14 and 15).

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.

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 all public-key computations removed from the circuit and the needed left-half scalars 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 all public-key computations removed from the circuit and the needed signing scalars moved into the public statement; the verifier computes $K = k \cdot G$ externally.

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

Table 16 collects the times at the full BIP44 path. Since the server is used only for proving, we convert its flat price of \$250 per month into $\$9.65 \times 10^{-5}$ per second to obtain the costs. Sign proving takes about 12.5–12.7 seconds in every scheme, that is, about 0.12 ¢ per transaction; its circuit consists of two SHA-256 evaluations with the RISC Zero accelerator. Derivation is a one-time cost per key pair, off the transaction path, from 0.21 ¢ for the fastest post-Q-day variants to 4.59 ¢ for QBIP32 full-path on secp256k1. Even in the pruned variants derivation is the largest cost, which makes it the natural target for incremental proving across derivation levels.

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 $m/44'/0'/0'/0/0$ ($m/44'/0'/0'/0'/0'$ for SLIP-10/Ed25519, where all levels are hardened). 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.

10 Incremental Derivation and Recursion

The derivation proof is the largest cost of the system (Section 9.4), and in the full-path variant it grows with the length of the path. Its cost can be reduced if the proof for level i builds on the proof for level $i-1$ instead of re-deriving the chain from the seed.

Recursion. In the simplest form, each level has its own proof, which proves only the step from level $i-1$ to level i. Consecutive proofs are linked by one-way hashes of the values they share, as π_{deriv} and π_{sign} are linked by h_w. Each proof is cheap, but the verifier checks one proof per level, and the size of the certificate grows linearly with the depth. In full STARK recursion, the proof

π_{i-1} is part of the witness of the proof for level i, which verifies π_{i-1} inside the circuit and proves one more derivation step. The result is a single receipt of constant size at every depth. The prover pays the cost of verifying π_{i-1} inside the circuit at each level, so the saving depends on the cost of this verification compared with the cost of one derivation step; we leave this measurement open.

Folding. Folding schemes prove the repeated application of the same step function incrementally, with roughly constant cost per step. The derivation step is the same function at every hardened level, so it is a natural fit. This solution is subject to the following obstacle: RISC Zero does not expose its arithmetization, so folding would need a zkVM that supports incrementally verifiable computation, or a native circuit for the derivation relation in a folding framework.

Delegated proving. The derivation proofs can be generated as a service: one or more servers, powerful enough to generate the derivation proofs, compute them for the users and return them. A server can also keep the proofs of the upper levels of a wallet and extend them when a new key is derived. The witness contains private values of the wallet, and a server that learns them can produce signing proofs for every key of the corresponding subtree, so the witness must be hidden from the servers. Possible solutions are secure multi-party computation, in which a set of servers computes the proof on a secret-shared witness [OB22, GGJ+23, CLMZ23], and techniques based on homomorphic encryption, in which a single untrusted server computes the proof on a hidden witness; Garg, Goel, and Wang [GGW24] obtain such a delegation from FRI on hidden values. Both solutions must be evaluated, to understand whether they are practical for the derivation relations of this paper.

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 a complete game-based reduction for BIP32-Ed25519 against QPT adversaries in the quantum random oracle model, under an assumption on the NIZK, 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 $h_w = \mathsf{SHA256}(w_n)$, 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 secure for hardened keys and not secure for non-hardened keys in every scheme (for BIP32 secp256k1 and BIP32-Ed25519, if every child of the root is hardened). Then we introduced QBIP32 unifying the constructions: this protocol makes the witness NW an explicit output of every $\mathsf{HASH768}$ call at hardened steps (at non-hardened steps it is updated with SHA-256), and one derivation scheme works uniformly across any prime-order curve[4]. 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 needed signing scalars to the public statement, removing all in-circuit scalar multiplications, under the

[4]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.

network-level assumption that, after Q-day, networks reject all non-post-quantum signatures. 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 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 ≈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. Section 10 discusses possible directions to reduce it: incremental derivation via recursion and folding, and delegated proving. We also plan to pursue the following strategies. First, we plan to evaluate RISC Zero's GPU-accelerated prover; this could reduce proving time without any change to the relations themselves, though the fixed per-segment overhead identified in Section 9.2 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. Second, 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. Third, 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 9.2 rather than by the guest program's exact cycle count. Fourth, 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.

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.

Acknowledgements

This work is the subject of pending patent applications owned by AmericanFortress. No license, express or implied, to any patent rights is granted by the publication of this paper.

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.

[AHU19] Andris Ambainis, Mike Hamburg, and Dominique Unruh. Quantum security proofs using semi-classical oracles. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part II*, volume 11693 of *Lecture Notes in Computer Science*, pages 269–295. Springer, Cham, August 2019.

[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.

[BDF+11] Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. Random oracles in a quantum world. In Dong Hoon Lee and Xiaoyun Wang, editors, *Advances in Cryptology – ASIACRYPT 2011*, volume 7073 of *Lecture Notes in Computer Science*, pages 41–69. Springer, Berlin, Heidelberg, December 2011.

[BDPV08] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. On the indifferentiability of the Sponge construction. In Nigel P. Smart, editor, *Advances in Cryptology – EUROCRYPT 2008*, volume 4965 of *Lecture Notes in Computer Science*, pages 181–197. Springer, Berlin, Heidelberg, April 2008.

[CDG+17] Melissa Chase, David Derler, Steven Goldfeder, Claudio Orlandi, Sebastian Ramacher, Christian Rechberger, Daniel Slamanig, and Greg Zaverucha. Post-quantum zero-knowledge and signatures from symmetric-key primitives. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 1825–1842. ACM Press, October / November 2017.

[CDHZ25] Alessandro Chiesa, Zijing Di, Zihan Hu, and Yuxi Zheng. How to prove post-quantum security for succinct non-interactive reductions. Cryptology ePrint Archive, Report 2025/2166, 2025.

[CF24] Alessandro Chiesa and Giacomo Fenzi. zkSNARKs in the ROM with unconditional UC-security. In Elette Boyle and Mohammad Mahmoody, editors, *TCC 2024: 22nd Theory of Cryptography Conference, Part I*, volume 15364 of *Lecture Notes in Computer Science*, pages 67–89. Springer, Cham, December 2024.

[CLMZ23] Alessandro Chiesa, Ryan Lehmkuhl, Pratyush Mishra, and Yinuo Zhang. Eos: Efficient private delegation of zkSNARK provers. In Joseph A. Calandrino and Carmela Troncoso, editors, *USENIX Security 2023: 32nd USENIX Security Symposium*, pages 6453–6469. USENIX Association, August 2023.

[CMS19] Alessandro Chiesa, Peter Manohar, and Nicholas Spooner. Succinct arguments in the quantum random oracle model. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019: 17th Theory of Cryptography Conference, Part II*, volume 11892 of *Lecture Notes in Computer Science*, pages 1–29. Springer, Cham, December 2019.

[Cza21] Jan Czajkowski. Quantum indifferentiability of SHA-3. Cryptology ePrint Archive, Report 2021/192, 2021.

[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.

[DFMS22] Jelle Don, Serge Fehr, Christian Majenz, and Christian Schaffner. Online-extractability in the quantum random-oracle model. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022, Part III*, volume 13277 of *Lecture Notes in Computer Science*, pages 677–706. Springer, Cham, May / June 2022.

[DRST12] Yevgeniy Dodis, Thomas Ristenpart, John P. Steinberger, and Stefano Tessaro. To hash or not to hash again? (In)differentiability results for H^2 and HMAC. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 348–366. Springer, Berlin, Heidelberg, August 2012.

[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.

[GGJ+23] Sanjam Garg, Aarushi Goel, Abhishek Jain, Guru-Vamsi Policharla, and Sruthi Sekar. zkSaaS: Zero-knowledge SNARKs as a service. In Joseph A. Calandrino and Carmela Troncoso, editors, *USENIX Security 2023: 32nd USENIX Security Symposium*, pages 4427–4444. USENIX Association, August 2023.

[GGW24] Sanjam Garg, Aarushi Goel, and Mingyuan Wang. How to prove statements obliviously? In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology – CRYPTO 2024, Part X*, volume 14929 of *Lecture Notes in Computer Science*, pages 449–487. Springer, Cham, August 2024.

[GHHM21] Alex B. Grilo, Kathrin Hövelmanns, Andreas Hülsing, and Christian Majenz. Tight adaptive reprogramming in the QROM. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2021, Part I*, volume 13090 of *Lecture Notes in Computer Science*, pages 637–667. Springer, Cham, December 2021.

[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.

[OB22] Alex Ozdemir and Dan Boneh. Experimenting with collaborative zk-SNARKs: Zero-knowledge proofs for distributed secrets. In Kevin R. B. Butler and Kurt Thomas, editors, *USENIX Security 2022: 31st USENIX Security Symposium*, pages 4291–4308. USENIX Association, August 2022.

[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.

[SR25] Michel Seck and Adeline Roux-Langlois. Towards post-quantum bitcoin blockchain using dilithium signature. *IACR Communications in Cryptology (CiC)*, 2(3):3, 2025.

[SW23] Or Sattath and Shai Wyborski. Protecting quantum procrastinators with signature lifting: A case study in cryptocurrencies. Cryptology ePrint Archive, Report 2023/362, 2023.

[SY17] Fang Song and Aaram Yun. Quantum security of NMAC and related constructions - PRF domain extension against quantum attacks. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part II*, volume 10402 of *Lecture Notes in Computer Science*, pages 283–309. Springer, Cham, August 2017.

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

[Zha19] Mark Zhandry. How to record quantum queries, and applications to quantum indifferentiability. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part II*, volume 11693 of *Lecture Notes in Computer Science*, pages 239–268. Springer, Cham, August 2019.