Provenance Proofs: Linkable Zero-Knowledge Derivation Relations for Hierarchical Deterministic Wallets

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

[1]AmericanFortress, Jackson Hole, Wyoming

November 4, 2026

Abstract

ZKPoSP proves that a public key or address was honestly derived from a hidden wallet seed, without revealing the seed or the private derivation state. This paper studies relationships among several derived values: whether public keys or addresses, possibly on different curves and different blockchains, share a hidden origin, which is the seed or an anchor of the wallet, and who else can recognize this relation.

We give a model for such *provenance proofs*, with derivation proofs, which carry a link value, and provenance proofs, which show that several derivation statements share an origin. We define correctness, derivation soundness, provenance soundness, non-frameability and unlinkability. We give four constructions, which differ in who can link: only the user, with a joint proof or with commitments linked later; anyone within a linking context, with a deterministic tag, which can also act as a nullifier; or a designated authority, with an encryption of that tag. We prove, in sketch form, that the constructions satisfy the five properties under the security of the ZKPoSP derivation relations, the random-oracle model, and the zero knowledge and simulation extractability of the proof system.

We describe an application to cross-chain transfers from Bitcoin to Solana, in which the payout address must come from the same wallet as the source address, so that an attacker cannot substitute the destination. We report Plonky3 benchmarks of the derivation proofs on which the constructions are built.

Contents

1 Introduction **2**

2 Preliminaries **3**
- 2.1 Derivation nodes and hash bindings 4
- 2.2 Hash functions and commitments 4
- 2.3 Non-interactive zero-knowledge arguments 4
- 2.4 Public-key encryption 4
- 2.5 What we use from ZKPoSP 4

3 Model **5**
- 3.1 Overview 5
- 3.2 Syntax 5
- 3.3 Properties 6

4 Constructions **8**
4.1 Common part 8
4.2 Form 1: joint provenance proofs 8
4.3 Form 1 with deferred linkage 9
4.4 Form 2: publicly linkable tags 9
4.5 Form 3: authority-linkable tags 10
4.6 Extensions outside the model 10

5 Security **11**
5.1 Assumptions on the derivation schemes 11
5.2 Security of the constructions 11

6 Application: cross-chain transfers **15**

7 Performance **17**

8 Related work and discussion **17**

9 Conclusion **18**

1 Introduction

ZKPoSP [2] proves that an existing public key or address was honestly derived from a hidden wallet seed, without revealing the seed or the private derivation state, and without changing the public-key or address format. The proof certifies either a complete derivation from the seed, or a pruned derivation that starts at a hardened anchor.

ZKPoSP proves one public value at a time. A wallet, however, controls many related values: addresses on several blockchains, payment codes, identity keys, credential keys. Often a verifier needs to know that several of these values belong to the same wallet, without learning any secret, the derivation paths, or the other keys of the wallet. The values may be derived under different schemes, for example on secp256k1 for Bitcoin and on Ed25519 for Solana, and then they have no common node: what they share is the seed. We call the seed, and any anchor fixed by the user, an *origin*, and we call a zero-knowledge proof that several values share one origin a *provenance proof*.

Showing that values share an origin raises a second question: who else may recognize the relation, and when. A user may want two services to be unable to link its proofs, while it still wants one service to link the proofs of one operation, or a regulator to recognize its activity. ZKPoSP already notes that a full-path proof can carry a fixed value computed from the seed, so that all proofs of a wallet can be recognized [2, Sec. 7.5]. That observation is informal, and it gives one choice only: every proof is linked to every other. This paper gives a model in which the user controls this choice, four constructions that realize different choices, and their security.

The model. In our model a user has two kinds of proofs. A *derivation proof* shows that one public value is derived from an origin, and is bound to the operation it is used for. Its statement may carry a *link value*, which decides who, besides the user, can link it to other derivation proofs. A *provenance proof* shows that several derivation statements share one origin. We require five properties: correctness; derivation soundness, so an accepted derivation proof is for a value really derived from an origin; provenance soundness, so an accepted provenance proof is for values that really share an origin; non-frameability, so nobody can use an honest user's values or proofs to build

proofs the user did not make, or to link a foreign value to the user; and unlinkability, so proofs of different users cannot be told apart, except where the construction links them on purpose.

The constructions. The four constructions differ in the link value, and so in who can link. In *Form 1* there is no link value, and the user links values with one provenance proof that proves all their derivations again. In *Form 1 with deferred linkage*, each derivation proof carries a fresh commitment to its origin, and the user can later link any chosen proofs with a short proof on the commitments, without repeating the derivations. In *Form 2*, each derivation proof carries a tag computed from the origin and a linking context, so anyone can link proofs made in the same context, while proofs made in different contexts stay unlinkable; the context can identify a service, or a single operation, in which case the tag also prevents a second use. In *Form 3*, the tag of Form 2 is encrypted under the key of an authority, so only the authority can link. In all forms, derivation proofs are NIZK arguments of knowledge built from the derivation relations of ZKPoSP.

Motivating applications. These choices of linkability make several applications possible. A sender can check that a fresh payment address of a recipient comes from the same wallet as the recipient's long-term identifier; the joint proof of Form 1 gives this check for one payment only, without creating an identifier that links the recipient's payments. An institution can show an auditor that selected withdrawal keys come from one custody origin, without revealing its key hierarchy, and it can show different keys to different auditors without letting them link their views. A regulated service can let only a regulator recognize repeated deposits from the same certified wallet since with Form 3, every other party sees unrelated proofs. A wallet that replaces a compromised anchor can prove that the new anchor comes from the same recovery origin as the old one, so the parties that trusted the old anchor can trust the new one without a new onboarding. A cross-chain service can pay out only to an address that belongs to the same wallet as the deposit address, even if the two chains use different curves, which stops an attacker from substituting the payout address. We describe the last application in more detail, as an example of how the constructions are used.

Contributions.

1. A model for provenance proofs (Section 3), with its procedures and five properties: correctness, derivation soundness, provenance soundness, non-frameability and unlinkability. The model covers origins in one tree and a common seed across several schemes.

2. Four constructions (Section 4), which give different choices of who can link, from nobody but the user to anyone within a linking context.

3. An application to cross-chain transfers from Bitcoin to Solana (Section 6), with an integration design for a typical instant-exchange service.

4. Plonky3 benchmarks of the derivation proofs on which all constructions are built (Section 7).

2 Preliminaries

We follow the notation of ZKPoSP [2].

2.1 Derivation nodes and hash bindings

A derivation node is written $N = (NS, NW, NC)$, where NS is the secret scalar, NW is a witness component, and NC is the chain code. The corresponding public key is $\mathrm{Pub}(NS)$, where Pub is the public-key map of the scheme, for example $NS \cdot G$ on an elliptic-curve group. Child nodes are computed from parent material and a child index by the keyed derivation function of the scheme: HMAC-SHA512 in BIP32 [14], SLIP-10 and BIP32-Ed25519, and HASH768 in QBIP32 [2]. We call a node *hidden* when it appears only inside a proof's witness.

The component NW can serve as a *quantum-safe witness*: a private value bound to the seed by a one-way function and not recoverable from the public key, even by an adversary that can compute discrete logarithms. ZKPoSP exposes a hash of such private material in the public statement of its derivation relations, for example $h_{nw} = \mathrm{SHA256}(NW_n)$ in the hardened QBIP32 relation or $h_{cw} = \mathrm{SHA256}(NC_{\mathrm{leaf}} \| NW_{\mathrm{leaf}})$ in the pruned one. We call it the *hash binding* and write it h when the scheme-specific form is not relevant.

2.2 Hash functions and commitments

We work in the random-oracle model (ROM), as ZKPoSP does. For a message $m \in \{0,1\}^*$ and randomness $r \xleftarrow{\$} \{0,1\}^{256}$ we define $\mathsf{Com}(m; r) := \mathrm{SHA256}(m \| r)$. In the ROM, Com is a statistically hiding and computationally binding commitment scheme.

2.3 Non-interactive zero-knowledge arguments

We use a NIZK $\Pi = (\mathrm{Setup}, \mathsf{NIZKProve}, \mathsf{NIZKVerify})$ with the properties defined in ZKPoSP [2]: perfect completeness, knowledge soundness, statistical zero knowledge, and simulation extractability.

We instantiate Π with a FRI-based STARK built with Plonky3, with zero knowledge obtained through a hiding polynomial commitment. The state of the art gives evidence that STARKs are simulation extractable: Chiesa and Fenzi show that hash-based SNARKs obtained from interactive oracle proofs in the ROM are UC-secure, which implies simulation extractability [5]. Based on this evidence, we conjecture that our Plonky3 STARK is simulation extractable; proving it is out of the scope of this paper. Of the properties of Section 3, only non-frameability uses simulation extractability (Section 5.2).

2.4 Public-key encryption

A public-key encryption scheme $(\mathrm{KeyGen}, \mathrm{Enc}, \mathrm{Dec})$ encrypts a message x under a public key pk with randomness a, written $\mathrm{Enc}_{pk}(x; a)$. We require correctness, that is, decryption under the matching secret key sk returns x for every x and every a, and IND-CPA security [6], that is, no efficient adversary that chooses two messages of the same length can tell which of them was encrypted with more than negligible advantage.

2.5 What we use from ZKPoSP

ZKPoSP defines derivation relations for BIP32-secp256k1, BIP32-Ed25519, SLIP-10/Ed25519 and QBIP32. The full derivation relation R_{deriv} certifies a complete derivation from the seed to the target value. The hardened relation $R^{\mathrm{hard}}_{\mathrm{deriv}}$ certifies a derivation along hardened steps. The pruned relation $R^{\mathrm{pruned}}_{\mathrm{deriv}}$ starts at the parent of the last hardened node, certifies that hardened step and the non-hardened suffix, and does not repeat the derivation from the seed.

Pruning is sound only when the node where the proof starts holds private material bound to the seed and not recoverable from public information. ZKPoSP shows that a proof that starts from a node reached by a non-hardened step is unsound, and that a proof whose first step is hardened is sound. Every derivation predicate in this paper starts at the seed or at such a hardened anchor.

The full-path relation $R_{\text{deriv}}^{\text{full-path}}$ of ZKPoSP adds a value $c_{\text{seed}} = \text{SHA256}(r \| seed)$ to the public statement, with r sampled once per wallet and reused in every full-path proof, so that all proofs of one wallet carry the same value. The same construction applies at any hardened ancestor. In the terminology of this paper this is a Form 2 tag with one constant linking context.

3 Model

3.1 Overview

The purpose of the model is to let a user convince a verifier that several public keys or addresses belong to the same wallet, without revealing any secret key, the derivation paths, or any other key of the wallet, and to let the user control who can recognize this relation. The keys may belong to different blockchains and be derived under different derivation schemes. The model captures the procedures that such a system needs and the security properties it must satisfy, independently of how the proofs are implemented.

A user holds one secret. A derivation scheme turns this secret into a tree of nodes. The secret determines the root of the tree, and each node determines its children, one for each child index. Every node contains a key pair, with a secret part and a public part. A node is identified by its path, that is, the sequence of child indices from the root to it. The same secret can be used with several derivation schemes, for example one for Bitcoin and one for Solana, and then it determines one tree, with its own root, for each scheme. The user may also fix an intermediate node as an *anchor*; for all derivations that start from an anchor, we consider the anchor as their root, and paths are taken from it. We call the secret and the anchors *origins*; an origin is always secret and is never revealed by any proof. The model has two kinds of proofs. A *derivation proof* shows that one public value is derived from an origin, and is bound to the operation it is used for. Its statement may also carry a *link value*, which determines who, besides the user, can link the proof to other derivation proofs, and how. A *provenance proof* shows that several derivation statements share one origin.

For example, a user who moves value from Bitcoin to Solana gives a derivation proof for its Bitcoin address and one for its Solana address, and the service checks a provenance proof before it pays out. A user who proves two addresses to two different services may want the services to be unable to link the two proofs.

We want five properties. *Correctness*: proofs produced honestly are accepted. *Derivation soundness*: an accepted derivation proof is for a value that is really derived from some origin. *Provenance soundness*: an accepted provenance proof is for values that really share an origin. *Non-frameability*: nobody can use an honest user's values or proofs to create an accepted proof that the honest user did not create, or to link a foreign value to the honest user. *Unlinkability*: derivation proofs of different users cannot be told apart, except where the instantiation lets them be linked on purpose.

3.2 Syntax

We write σ for a derivation scheme. Each scheme specifies which child indices are *hardened*. The public output of a node is a pair (Y, h), where Y is the public value used by applications, such as

a public key or an address, and h is additional public data, possibly empty. A provenance proof system for a family of derivation schemes consists of the following procedures.

- $\mathsf{GlobalSetup}(1^\lambda) \to \mathrm{pp}$ generates the public parameters shared by all users and verifiers, including any auxiliary data that an instantiation needs.
- $\mathsf{UserSetup}(\mathrm{pp}) \to S$ generates the secret of a user.
- $\mathsf{SetAnchor}(\mathrm{pp}, \sigma, S, p) \to N_a$ returns the anchor N_a, which is the node at path p in the tree that the scheme σ derives from S. The *origins of* S are S itself and the anchors returned by SetAnchor on S.
- $\mathsf{Derive}(\mathrm{pp}, \sigma, \rho, p) \to (N, Y, h)$ returns the node N at path p from the origin ρ, and its public output (Y, h).
- $\mathsf{LinkKeyGen}(\mathrm{pp}) \to (lpk, lsk)$ generates a linking key pair for a party that is allowed to link proofs; instantiations that do not use such keys return $(\bot, \bot)$.
- $\mathsf{DerivProve}(\mathrm{pp}, \rho, \sigma, p, ctx, lc, lpk) \to (x, \pi, st)$ produces a derivation proof for the value at path p from the origin ρ. The statement is $x = (ctx, \sigma, Y, h, \ell)$, where ctx describes the operation the proof is for, (Y, h) is the public output, and ℓ is the link value, computed from ρ, the linking context lc and the key lpk, or $\ell = \bot$ if the instantiation uses no link value. The prover keeps the private state st for later provenance proofs. It returns $\bot$ if ρ is an anchor and p is empty or starts with a non-hardened index.
- $\mathsf{DerivVerify}(\mathrm{pp}, x, \pi) \to \{0, 1\}$ verifies a derivation proof.
- $\mathsf{ProvProve}(\mathrm{pp}, (x_i)_{i\in[k]}, \rho, (p_i, st_i)_{i\in[k]}) \to \pi_{\mathrm{prov}}$ produces a provenance proof that the derivation statements $x_1, \ldots, x_k$ share the origin ρ. An instantiation may return $\pi_{\mathrm{prov}} = \bot$ when the link values alone establish this.
- $\mathsf{ProvVerify}(\mathrm{pp}, (x_i)_{i\in[k]}, \pi_{\mathrm{prov}}, lsk) \to \{0, 1\}$ verifies a provenance proof. The key lsk is $\bot$ unless the instantiation restricts linking to the holder of a linking key.
- $\mathsf{PubLink}(\mathrm{pp}, lc, lc') \to \{0, 1\}$ is a public deterministic procedure that returns 1 if two derivation statements from the same origin, produced with linking contexts lc and lc', can be linked by anyone from their link values alone, without any key.

We say that (Y, h) is *derived from* an origin ρ under σ along p if $\mathsf{Derive}(\mathrm{pp}, \sigma, \rho, p)$ returns a node with public output (Y, h) and, when ρ is an anchor, p is non-empty and starts with a hardened index. The condition on anchors makes this notion non-trivial: without it, every public output would be derived from its own node along the empty path.

The model does not fix the link value or the way ProvVerify and PubLink use it; these choices distinguish the instantiations of Section 4.

3.3 Properties

Definition 3.1 (Correctness)**.** *For every* pp *output by* GlobalSetup *and every* S *output by* UserSetup*,* DerivVerify *accepts every output of* DerivProve *on an origin of* S *that is not* $\bot$*. If* $x_1, \ldots, x_k$ *are produced by* DerivProve *from one origin* ρ *of* S *with the same linking context* lc *and the same key* lpk*, then* ProvVerify *accepts the output of* ProvProve *on them, under the secret key that matches* lpk*, or under* $\bot$ *if* $lpk = \bot$*.*

The other properties are defined by games between a challenger and a PPT adversary. All procedures are public and all parties use the same pp. The challenger creates the honest users with UserSetup and keeps their secrets; the adversary can create its own users and its own linking keys. The games use the following oracles, parametrized by an origin ρ or by a secret S of an honest user.

- $\mathcal{O}_\rho^{\mathrm{Deriv}}(\sigma, p, ctx, lc, lpk)$ returns a derivation proof (x, π) from $\mathsf{DerivProve}(\mathrm{pp}, \rho, \sigma, p, ctx, lc, lpk)$, keeps the inputs and the private state, and adds x to a list Q.
- $\mathcal{O}_\rho^{\mathrm{Prov}}((x_i)_{i\in[k]})$, for statements returned by $\mathcal{O}_\rho^{\mathrm{Deriv}}$ with the same linking context and the same key, returns $\mathsf{ProvProve}(\mathrm{pp}, (x_i)_{i\in[k]}, \rho, (p_i, st_i)_{i\in[k]})$, where p_i and st_i are the path and the private state that $\mathcal{O}_\rho^{\mathrm{Deriv}}$ kept for x_i.
- $\mathcal{O}_S^{\mathrm{Corr}}(\sigma, p)$, for a path p whose last index is hardened, returns the node at p in the tree that σ derives from S, and marks its subtree as corrupted.
- $\mathcal{O}^{\mathrm{Key}}()$ runs LinkKeyGen, returns lpk, and keeps (lpk, lsk) in a list K.

A statement in Q is *honest* if its origin and its node are not in a corrupted subtree. We say that ProvVerify *accepts* if it outputs 1 under $\perp$ or under the secret key of an entry of K.

Definition 3.2 (Derivation soundness)**.** *The challenger runs* GlobalSetup*. The adversary receives* pp*. It outputs a derivation statement $x = (ctx, \sigma, Y, h, \ell)$ and a proof π. It wins if* DerivVerify *accepts (x, π) and (Y, h) is not derived from any origin under σ. The system is derivation sound if every PPT adversary wins with negligible probability.*

Definition 3.3 (Provenance soundness)**.** *The challenger runs* GlobalSetup*. The adversary receives* pp *and access to $\mathcal{O}^{\mathrm{Key}}$. It outputs derivation statements $x_1, \ldots, x_k$ with proofs, and a provenance proof π_{prov}. It wins if* DerivVerify *accepts every derivation proof,* ProvVerify *accepts π_{prov}, and the public outputs of $x_1, \ldots, x_k$ are not all derived from one origin. The system is provenance sound if every PPT adversary wins with negligible probability.*

Definition 3.4 (Non-frameability)**.** *The challenger runs* GlobalSetup *and creates one honest user with secret S. The adversary receives* pp *and access to $\mathcal{O}_S^{\mathrm{Corr}}$, to $\mathcal{O}_\rho^{\mathrm{Deriv}}$ and $\mathcal{O}_\rho^{\mathrm{Prov}}$ for every origin ρ of S, and to $\mathcal{O}^{\mathrm{Key}}$. It outputs derivation statements with proofs, and possibly a provenance proof. It wins if* DerivVerify *accepts every derivation proof, and either some statement not in Q has the public value of an honest statement, or* ProvVerify *accepts the provenance proof on a list that contains an honest statement and a statement not in Q whose public value is not in a corrupted subtree. The system is non-frameable if every PPT adversary wins with negligible probability.*

The first case covers an honest value used in a new context, for example a substituted payout address. The second case covers a foreign value linked to an honest statement. Neither case covers a replay of an honest statement with its proof; applications prevent it by making *ctx* unique per operation or by recording link values.

Definition 3.5 (Unlinkability)**.** *The challenger runs* GlobalSetup*, creates two honest users with secrets S_0, S_1, and samples $b \xleftarrow{\$} \{0,1\}$. The adversary receives* pp *and access to $\mathcal{O}_\rho^{\mathrm{Deriv}}$ and $\mathcal{O}_\rho^{\mathrm{Prov}}$ for every origin ρ of S_0 and of S_1, and to $\mathcal{O}^{\mathrm{Key}}$. Once, it chooses an origin position (the secret, or the anchor at a given path) and inputs for $\mathcal{O}^{\mathrm{Deriv}}$, and receives the derivation proof computed on these inputs from that origin of S_b. It outputs a bit b'. It wins if $b' = b$ and it did not link the challenge on purpose, that is, it did not query any origin of S_0 or S_1 for the node of the challenge (the same scheme and the same position in the tree, whatever origin is used), did not use a linking*

context that PubLink *links to the challenge one, did not use a linking key outside K, and did not ask for a provenance proof that contains the challenge. The system is unlinkable if every PPT adversary wins with probability at most* $1/2$ *plus a negligible function.*

Unlinkability concerns proofs only: if an application publishes the same public value in two operations, for example by reusing an address, the two operations are linked by that value regardless of the proofs.

4 Constructions

We give four instantiations of the model of Section 3. In all of them, a derivation proof is a NIZK argument of knowledge, produced with the proof system Π of Section 2.3, for a relation built from the derivation relations of ZKPoSP [2]. In Form 1 the relation is the ZKPoSP derivation relation itself. In the other forms it also checks that the link value is computed from the same origin. The idea of such a link value was introduced informally in ZKPoSP [2, Sec. 7.5]; here we develop it, with a commitment, a tag that depends on a linking context, or an encrypted tag as link value. In Form 1 and in its deferred variant, the provenance proof is also a NIZK argument of knowledge with Π, for a relation over several derivation statements. In Forms 2 and 3 no provenance proof is needed: ProvProve returns $\perp$, and ProvVerify checks the link values directly. The instantiations differ in the link value ℓ, in the relations they prove, and in how ProvVerify and PubLink work.

4.1 Common part

GlobalSetup runs the setup of the NIZK Π of Section 2.3 and outputs its parameters as pp. UserSetup samples a seed S. SetAnchor and Derive compute nodes and public outputs with the derivation functions of the scheme σ, for example BIP32 or QBIP32. We write $\mathcal{D}_\sigma(\rho, p, Y, h) = 1$ if (Y, h) is derived from the origin ρ under σ along p, as defined in Section 3.2. The base derivation relation is

$$R_{\text{deriv}} := \big\{ \big((ctx, \sigma, Y, h), (\rho, p) \big) \;\big|\; \mathcal{D}_\sigma(\rho, p, Y, h) = 1 \big\}.$$

It is the derivation relation of ZKPoSP [2]: the full-path relation when ρ is the seed, and the pruned relation when ρ is an anchor. The value ctx is the auxiliary public value of the ZKPoSP derivation relation, which ZKPoSP uses to link a derivation proof to a signing proof [2]; in this work it identifies the operation the proof is for.

We encode seeds, nodes and linking contexts as bit strings with a fixed encoding encode$(\cdot)$, where different values always give different bit strings. We write $\mathsf{Com}(\rho; a)$ for the commitment $\mathsf{Com}(\text{encode}(\rho); a)$. Forms 2 and 3 use a deterministic tag function: for a linking context lc and an origin ρ, $H_{\text{tag}}(lc, \rho) := \text{SHA256}(\text{encode}(lc) \| \text{encode}(\rho))$.

4.2 Form 1: joint provenance proofs

Form 1 lets the user show that several keys share one origin, while nothing in the derivation statements can be used by anyone else to link them. Derivation statements carry no link value, so different derivation proofs cannot be compared, and only the user can link them. To link some keys, the user gives one provenance proof that proves again all their derivations, with the same origin as witness in every derivation. This is the simplest form, but the provenance proof repeats all the derivation work.

- LinkKeyGen returns $(\perp, \perp)$, and DerivProve ignores lc and lpk.

- DerivProve outputs $x = (ctx, \sigma, Y, h, \bot)$, a proof π for R_{deriv}, and $st = \bot$.
- ProvProve outputs a proof for the relation

$$R_{\text{prov}} := \big\{ \big((x_i)_{i\in[k]}, (\rho, (p_i)_{i\in[k]}) \big) \;\big|\; \mathcal{D}_{\sigma_i}(\rho, p_i, Y_i, h_i) = 1 \text{ for every } i \in [k] \big\},$$

  and ProvVerify verifies it.
- PubLink always returns 0.

4.3 Form 1 with deferred linkage

This variant lets the user prove each key when it is used, possibly at different times and to different verifiers, and decide only later which proofs to link. Nobody else can link the proofs, and the later link proof does not repeat the derivations. To obtain this, each derivation proof also proves a fresh commitment to its origin. The commitments hide the origin, so different proofs cannot be compared. To link some of them, the user proves that their commitments open to the same origin, which only the user can do, since only the user knows the randomness.

- LinkKeyGen returns $(\bot, \bot)$, and DerivProve ignores lc and lpk.
- DerivProve samples $a \xleftarrow{\$} \{0,1\}^{256}$, sets $C := \mathsf{Com}(\rho; a)$, outputs $x = (ctx, \sigma, Y, h, C)$, a proof π for

$$R_{\text{com}} := \big\{ \big((ctx, \sigma, Y, h, C), (\rho, p, a) \big) \;\big|\; \mathcal{D}_\sigma(\rho, p, Y, h) = 1 \;\wedge\; C = \mathsf{Com}(\rho; a) \big\},$$

  and $st = a$.
- ProvProve outputs a proof for the relation

$$R_{\text{link}} := \big\{ \big((C_i)_{i\in[k]}, (\rho, (a_i)_{i\in[k]}) \big) \;\big|\; C_i = \mathsf{Com}(\rho; a_i) \text{ for every } i \in [k] \big\},$$

  where C_i is the link value of x_i and $a_i = st_i$, and ProvVerify verifies it.
- PubLink always returns 0.

4.4 Form 2: publicly linkable tags

Form 2 lets anyone check, without any further proof, that derivation proofs made in the same linking context come from the same origin, for example all proofs of one operation or all proofs given to one service. Proofs made in different linking contexts stay unlinkable. To obtain this, each derivation proof carries a tag, which is a deterministic function of the linking context and the origin. The same origin and the same context always give the same tag, while different contexts give unrelated tags. The derivation proof also proves that the tag is computed from the same origin as the key, so a tag cannot be copied onto a key with a different origin.

- LinkKeyGen returns $(\bot, \bot)$, and DerivProve ignores lpk.
- DerivProve sets $\tau := H_{\text{tag}}(lc, \rho)$, outputs $x = (ctx, \sigma, Y, h, (lc, \tau))$, a proof π for

$$R_{\text{tag}} := \big\{ \big((ctx, \sigma, Y, h, (lc, \tau)), (\rho, p) \big) \;\big|\; \mathcal{D}_\sigma(\rho, p, Y, h) = 1 \;\wedge\; \tau = H_{\text{tag}}(lc, \rho) \big\},$$

  and $st = \bot$.
- ProvProve returns $\bot$, and ProvVerify accepts if all link values (lc_i, τ_i) are equal.
- PubLink(pp, lc, lc') returns 1 if and only if $lc = lc'$.

4.5 Form 3: authority-linkable tags

Form 3 gives the same linking as Form 2, but only to a designated authority, for example a regulator or an auditor; every other party can verify the proofs but cannot link them. To obtain this, the tag of Form 2 is encrypted under the public key of the authority. The derivation proof also proves that the ciphertext contains the correct tag. Since the encryption scheme is IND-CPA secure, nobody except the authority can tell whether two ciphertexts contain the same tag; the authority decrypts them and compares the tags.

- LinkKeyGen runs KeyGen of the encryption scheme.
- DerivProve samples randomness a and sets $ct := \mathrm{Enc}_{lpk}(H_{\mathrm{tag}}(lc, \rho); a)$. It outputs the statement $x = (ctx, \sigma, Y, h, (lc, lpk, ct))$, a proof π for

$$R^{\mathrm{auth}}_{\mathrm{tag}} := \left\{ \big((ctx, \sigma, Y, h, (lc, lpk, ct)), (\rho, p, a)\big) \;\middle|\; \begin{array}{l} \mathcal{D}_\sigma(\rho, p, Y, h) = 1, \\ ct = \mathrm{Enc}_{lpk}(H_{\mathrm{tag}}(lc, \rho); a) \end{array} \right\},$$

  and $st = \bot$.
- ProvProve returns $\bot$. ProvVerify accepts if $lsk \neq \bot$, all statements have the same lc and the same lpk, lsk is the secret key that matches this lpk, and $\mathrm{Dec}_{lsk}(ct_i)$ is the same for every i.
- PubLink always returns 0.

The public key is part of the statement, so every verifier checks that the ciphertext is addressed to the intended authority. To let several authorities link independently, the user includes one ciphertext for each authority, and each authority runs ProvVerify on its own ciphertexts. A deployment that must resist quantum adversaries should use an encryption scheme that is IND-CPA secure against them [11, 9, 1, 10], since a quantum adversary that breaks the encryption can decrypt the tags and link the proofs.

4.6 Extensions outside the model

We describe two extensions used by some of the applications in Section 1. They are not covered by the definitions of Section 3, and we argue their security only informally.

Selective disclosure over a committed set. Some applications need to prove that a value belongs to a set published by a third party, for example an approved set of credential keys. The set is represented by a Merkle tree with root TreeCommit whose leaves are commitments $c_j := \mathsf{Com}(N_j; r_j)$ to member nodes. A prover holding a committed node N_j shows, inside the same NIZK as the derivation of Y_j, that Y_j is derived from N_j and that c_j belongs to the tree at TreeCommit. An accepting proof must satisfy both the derivation and the authentication path, so a value cannot be attributed to the set without breaking the collision resistance of the tree hash. Zero knowledge hides N_j, r_j, the authentication path and the other leaves.

Continuity under rotation. A Form 2 tag may already be trusted by many verifiers when its anchor must be replaced. Let R_{rec} be a protected recovery origin, and let the current anchor N_a and its successor N_b be the nodes that Derive returns from R_{rec} along two fixed paths p_{active} and p_{next}. For a linking context lc, let $\mathrm{AnchorTag}_a := H_{\mathrm{tag}}(lc, N_a)$ and $\mathrm{AnchorTag}_b := H_{\mathrm{tag}}(lc, N_b)$.

The rotation proof is for the relation R_{cont} with statement $x = (lc, \text{AnchorTag}_a, \text{AnchorTag}_b)$ and witness $w = (R_{\text{rec}}, N_a, N_b)$:

$$R_{\text{cont}} := \left\{ (x, w) \;\middle|\; \begin{array}{l} N_a \text{ and } N_b \text{ are the nodes at } p_{\text{active}} \text{ and } p_{\text{next}} \text{ from } R_{\text{rec}}, \\ \text{AnchorTag}_a = H_{\text{tag}}(lc, N_a), \quad \text{AnchorTag}_b = H_{\text{tag}}(lc, N_b) \end{array} \right\}.$$

Knowledge soundness gives one R_{rec} from which both anchors are derived along the fixed paths, so an arbitrary descendant of R_{rec} cannot be presented as the successor. If N_a is compromised but R_{rec} stays secret, the attacker cannot authorize a new anchor, since a valid proof requires knowledge of R_{rec}. Zero knowledge hides R_{rec}, N_a and N_b.

5 Security

5.1 Assumptions on the derivation schemes

Let Π be the NIZK argument of knowledge of Section 2.3, used in all constructions of Section 4. Besides the properties of Π, of the commitment and of the encryption scheme, the security proofs use two properties of the derivation schemes, which we define here.

Derivation binding. Let S^* be the seed of an honest user. The adversary $\mathcal{A}$ can choose paths and receive the public outputs of the nodes at these paths. These nodes are in the trees that the derivation schemes derive from S^*. $\mathcal{A}$ can also corrupt nodes. For a path whose last index is hardened, it receives the node at that path, including its secret key. The subtree of that node is then considered corrupted. The derivation schemes are *derivation binding* if no PPT adversary can output public value Y^* of a node outside the corrupted subtrees, an origin ρ', a path p' and data h' such that (Y^*, h') is derived from ρ' along p'. In other words, corrupting a node gives $\mathcal{A}$ control only over the subtree of that node, and not over the rest of the tree. Outside the corrupted subtrees, $\mathcal{A}$ cannot prove that an honest key comes from an origin it knows. In particular, it cannot recover the honest seed or an honest anchor. The derivation schemes of ZKPoSP satisfy this property in the ROM [2, Sec. 6.1, Prop. 6.6, Prop. 6.8, Prop. 7.3]. The reason is that the first step from any origin is either the computation of the root node from the seed, or a hardened step. The output of this step is a random value, so the adversary cannot choose an origin that leads to a given key.

Output unlinkability. Let S_0, S_1 be two honest seeds. The public output of the node at a given position under S_0 cannot be distinguished from the public output of the node at the same position under S_1, even by an adversary that obtains the public outputs of all other nodes of both seeds. The derivation schemes of ZKPoSP satisfy this property in the ROM, since every child is computed with a random-oracle call keyed by a value that the adversary never sees: at a hardened step the call also takes the secret key of the parent, and at a non-hardened step it is keyed by the chain code of the parent, which is not part of any public output or proof in the game. If an extended public key, that is, a public key together with its chain code, is published, the non-hardened descendants of that node can be computed by anyone and become linkable, as in BIP32 itself; the unlinkability game does not reveal chain codes, so this case is outside the model.

5.2 Security of the constructions

We prove the security of each form separately. Each property uses a different part of Π. Derivation and provenance soundness need only knowledge soundness. Unlinkability needs only zero

knowledge, since its proof simulates proofs but never extracts. Non-frameability needs simulation extractability, because its proof must simulate the proofs of the honest user, without knowing the honest seed, and extract witnesses from the proofs of the adversary in the same game.

Theorem 5.1. *Assume that* Π *is complete, knowledge sound and zero knowledge, and that the derivation schemes are derivation binding and output unlinkable. Then Form 1 is correct, derivation sound, provenance sound and unlinkable. If* Π *is also simulation extractable, Form 1 is also non-frameable.*

Proof sketch. Correctness. Derivation proofs and provenance proofs produced honestly are accepted, by completeness of Π. Indeed, $\mathsf{DerivProve}$ uses the origin ρ and the path p, which are a valid witness for R_{deriv}; this is the completeness of the ZKPoSP derivation proofs [2]. Likewise, $\mathsf{ProvProve}$ uses the common origin ρ and the paths $p_1, \ldots, p_k$, which are a valid witness for R_{prov}.

Derivation soundness. If $\mathsf{DerivVerify}$ accepts a proof for $x = (ctx, \sigma, Y, h, \perp)$, knowledge soundness of Π gives a witness (ρ, p) with $\mathcal{D}_\sigma(\rho, p, Y, h) = 1$. So (Y, h) is derived from ρ, and the adversary wins only with negligible probability.

Provenance soundness. If $\mathsf{ProvVerify}$ accepts, knowledge soundness of Π for R_{prov} gives one origin ρ and paths $p_1, \ldots, p_k$ such that every public output (Y_i, h_i) is derived from ρ. So the public outputs share an origin, and the adversary wins only with negligible probability.

Non-frameability. The argument uses three properties. By zero knowledge, the honest proofs that $\mathcal{A}$ receives reveal nothing about the honest origins beyond the statements themselves. By simulation extractability, seeing these proofs does not help $\mathcal{A}$ produce an accepted proof for a new statement: every such proof comes with a witness, so $\mathcal{A}$ must know an origin and a path for it. By derivation binding, $\mathcal{A}$ cannot know an origin from which an honest public value outside the corrupted subtrees is derived. In the first winning case, $\mathcal{A}$ outputs an accepted statement $x \notin Q$ with the public value of an honest statement. So $\mathcal{A}$ knows an origin from which this honest value is derived, which contradicts derivation binding. In the second winning case, $\mathcal{A}$ outputs an accepted provenance proof on a list that contains an honest statement and a statement not in Q. This list was never answered by $\mathcal{O}_\rho^{\text{Prov}}$, so the provenance proof is for a new statement, and $\mathcal{A}$ knows one origin from which all public values of the list are derived, including the honest one. This again contradicts derivation binding.

Unlinkability. The challenge statement is $x^* = (ctx^*, \sigma, Y, h, \perp)$, and $\mathcal{A}$ must tell whether it comes from S_0 or from S_1. By zero knowledge, the proofs reveal nothing beyond the statements, so $\mathcal{A}$ can use only the statements. In Form 1 there is no link value, and ctx^* is chosen by $\mathcal{A}$, so only the public output (Y, h) of the challenge depends on the user. $\mathcal{A}$ did not ask for the node of the challenge under either seed, and did not ask for a provenance proof that contains the challenge. So, by output unlinkability, $\mathcal{A}$ cannot tell whether (Y, h) comes from S_0 or from S_1, and it wins with probability at most $1/2$ plus a negligible function. □

Theorem 5.2. *Assume that* Π *is complete, knowledge sound and zero knowledge, that SHA256 is a random oracle, and that the derivation schemes are derivation binding and output unlinkable. Then Form 1 with deferred linkage is correct, derivation sound, provenance sound and unlinkable. If* Π *is also simulation extractable, it is also non-frameable.*

Proof sketch. Correctness. Honest derivation proofs are accepted by the completeness of the ZKPoSP derivation proofs [2], since the additional check $C = \mathsf{Com}(\rho; a)$ holds by construction. Honest provenance proofs are accepted by completeness of Π, since $\mathsf{ProvProve}$ uses the common origin ρ and the randomness $a_1, \ldots, a_k$ of the commitments, which are a valid witness for R_{link}.

Derivation soundness. By knowledge soundness of Π, an accepted derivation proof comes with a witness (ρ, p, a) for R_{com}, which satisfies $\mathcal{D}_\sigma(\rho, p, Y, h) = 1$. So (Y, h) is derived from ρ, and the adversary wins only with negligible probability.

Provenance soundness. By knowledge soundness of Π, each accepted derivation proof comes with a witness (ρ_i, p_i, a'_i) such that $C_i = \mathsf{Com}(\rho_i; a'_i)$ and (Y_i, h_i) is derived from ρ_i. The accepted provenance proof comes with a witness for R_{link}, that is, one origin ρ and randomness a_i such that $C_i = \mathsf{Com}(\rho; a_i)$ for every i. By binding of Com, $\rho_i = \rho$ for every i, so all public outputs are derived from ρ.

Non-frameability. The argument uses four properties. By zero knowledge, the honest proofs that $\mathcal{A}$ receives reveal nothing beyond the statements. By hiding of Com, the honest commitments in these statements reveal nothing about the honest origins. By simulation extractability, every accepted proof that $\mathcal{A}$ produces for a new statement comes with a witness, so $\mathcal{A}$ must know it. By derivation binding, $\mathcal{A}$ cannot know an origin from which an honest public value outside the corrupted subtrees is derived. In the first winning case, $\mathcal{A}$ outputs an accepted statement not in Q with the public value of an honest statement. So $\mathcal{A}$ knows an origin from which this honest value is derived, which contradicts derivation binding. In the second winning case, the list contains an honest statement, whose commitment was computed with fresh randomness that $\mathcal{A}$ never sees. If the provenance proof is for a new list of commitments, simulation extractability shows that $\mathcal{A}$ knows an opening of this honest commitment. Otherwise the list of commitments was answered by $\mathcal{O}_\rho^{\text{Prov}}$, so the new statement carries an honest commitment, and simulation extractability on its derivation proof shows again that $\mathcal{A}$ knows an opening of it. In the ROM, opening an honest commitment requires its randomness, so this happens only with negligible probability.

Unlinkability. The challenge statement is $x^* = (ctx^*, \sigma, Y, h, C)$, and $\mathcal{A}$ must tell whether it comes from S_0 or from S_1. By zero knowledge, the proofs reveal nothing beyond the statements, so $\mathcal{A}$ can use only the statements. The context ctx^* is chosen by $\mathcal{A}$. The commitment C uses fresh randomness, and $\mathcal{A}$ did not ask for a provenance proof that contains the challenge, so, by hiding of Com, C reveals nothing about the user. $\mathcal{A}$ did not ask for the node of the challenge under either seed, so, by output unlinkability, it cannot tell whether (Y, h) comes from S_0 or from S_1. So $\mathcal{A}$ wins with probability at most $1/2$ plus a negligible function. □

Theorem 5.3. *Assume that* Π *is complete, knowledge sound and zero knowledge, that SHA256 is a random oracle, and that the derivation schemes are derivation binding and output unlinkable. Then Form 2 is correct, derivation sound, provenance sound and unlinkable. If* Π *is also simulation extractable, Form 2 is also non-frameable.*

Proof sketch. Correctness. Honest derivation proofs are accepted by the completeness of the ZK-PoSP derivation proofs [2], since the additional check $\tau = H_{\text{tag}}(lc, \rho)$ holds by construction. Statements produced from the same origin with the same linking context carry the same tag, so $\mathsf{ProvVerify}$ accepts.

Derivation soundness. By knowledge soundness of Π, an accepted derivation proof comes with a witness (ρ, p) for R_{tag}, which satisfies $\mathcal{D}_\sigma(\rho, p, Y, h) = 1$. So (Y, h) is derived from ρ, and the adversary wins only with negligible probability.

Provenance soundness. By knowledge soundness of Π, each accepted derivation proof comes with an origin ρ_i such that (Y_i, h_i) is derived from ρ_i and $\tau_i = H_{\text{tag}}(lc_i, \rho_i)$. Since $\mathsf{ProvVerify}$ accepts, all pairs (lc_i, τ_i) are equal, so $H_{\text{tag}}(lc, \rho_i) = H_{\text{tag}}(lc, \rho_1)$ for every i. The encoding is injective, so $\rho_i = \rho_1$ unless $\mathcal{A}$ found a collision of SHA256, which happens only with negligible probability.

Non-frameability. The argument uses four properties. By zero knowledge, the honest proofs that $\mathcal{A}$ receives reveal nothing beyond the statements. In the ROM, an honest tag $H_{\text{tag}}(lc, \rho^*)$ is

a random value, and it reveals nothing about the honest origin ρ^* unless $\mathcal{A}$ queries SHA256 on its input, which requires knowing ρ^*. By simulation extractability, every accepted proof that $\mathcal{A}$ produces for a new statement comes with a witness, so $\mathcal{A}$ must know it. By derivation binding, $\mathcal{A}$ cannot know an honest origin, or an origin from which an honest public value outside the corrupted subtrees is derived. In the first winning case, $\mathcal{A}$ outputs an accepted statement not in Q with the public value of an honest statement. So $\mathcal{A}$ knows an origin from which this honest value is derived, which contradicts derivation binding. In the second winning case, the new statement carries the same link value (lc, τ) as an honest statement, so, by simulation extractability, $\mathcal{A}$ knows an origin ρ' with $H_{\text{tag}}(lc, \rho') = \tau$. This is possible only if ρ' is the honest origin, which contradicts derivation binding, or if $\mathcal{A}$ found a collision of SHA256, which happens only with negligible probability.

Unlinkability. The challenge statement is $x^* = (ctx^*, \sigma, Y, h, (lc^*, \tau))$, and $\mathcal{A}$ must tell whether it comes from S_0 or from S_1. By zero knowledge, the proofs reveal nothing beyond the statements, so $\mathcal{A}$ can use only the statements. The context ctx^* and the linking context lc^* are chosen by $\mathcal{A}$. Since PubLink returns 1 exactly when the two linking contexts are equal, no other query of either user uses lc^*. So, in the ROM, the tag τ is a random value that $\mathcal{A}$ has never seen. $\mathcal{A}$ could recognize it only by querying SHA256 on its input, which requires knowing an honest origin and contradicts derivation binding. $\mathcal{A}$ did not ask for the node of the challenge under either seed, so, by output unlinkability, it cannot tell whether (Y, h) comes from S_0 or from S_1. So $\mathcal{A}$ wins with probability at most 1/2 plus a negligible function. □

Theorem 5.4. *Assume that* Π *is complete, knowledge sound and zero knowledge, that SHA256 is a random oracle, that the encryption scheme is correct and IND-CPA secure, and that the derivation schemes are derivation binding and output unlinkable. Then Form 3 is correct, derivation sound, provenance sound and unlinkable. If* Π *is also simulation extractable, Form 3 is also non-frameable.*

Proof sketch. Correctness. Honest derivation proofs are accepted by the completeness of the ZK-PoSP derivation proofs [2], since the additional check $ct = \text{Enc}_{lpk}(H_{\text{tag}}(lc, \rho); a)$ holds by construction. By correctness of the encryption scheme, the authority decrypts the same tag from every statement produced from the same origin with the same linking context and key, so ProvVerify accepts.

Derivation soundness. By knowledge soundness of Π, an accepted derivation proof comes with a witness (ρ, p, a) for $R^{\text{auth}}_{\text{tag}}$, which satisfies $\mathcal{D}_\sigma(\rho, p, Y, h) = 1$. So (Y, h) is derived from ρ, and the adversary wins only with negligible probability.

Provenance soundness. ProvVerify accepts only under the secret key of a key pair created by $\mathcal{O}^{\text{Key}}$, so the key is honest. By knowledge soundness of Π, each accepted derivation proof comes with an origin ρ_i such that (Y_i, h_i) is derived from ρ_i and $ct_i = \text{Enc}_{lpk}(H_{\text{tag}}(lc, \rho_i); a_i)$. By correctness of the encryption scheme, $\text{Dec}_{lsk}(ct_i) = H_{\text{tag}}(lc, \rho_i)$. Since ProvVerify accepts, these values are all equal, so $H_{\text{tag}}(lc, \rho_i) = H_{\text{tag}}(lc, \rho_1)$ for every i. The encoding is injective, so $\rho_i = \rho_1$ unless $\mathcal{A}$ found a collision of SHA256, which happens only with negligible probability.

Non-frameability. The argument uses five properties. By zero knowledge, the honest proofs that $\mathcal{A}$ receives reveal nothing beyond the statements. In the ROM, an honest tag $H_{\text{tag}}(lc, \rho^*)$ is a random value, and it reveals nothing about the honest origin ρ^* unless $\mathcal{A}$ queries SHA256 on its input, which requires knowing ρ^*. By simulation extractability, every accepted proof that $\mathcal{A}$ produces for a new statement comes with a witness, so $\mathcal{A}$ must know it. By correctness of the encryption scheme, a ciphertext decrypts to the tag it was computed from. By derivation binding, $\mathcal{A}$ cannot know an honest origin, or an origin from which an honest public value outside the corrupted subtrees is derived. In the first winning case, $\mathcal{A}$ outputs an accepted statement not in Q with the public value of an honest statement. So $\mathcal{A}$ knows an origin from which this honest value

is derived, which contradicts derivation binding. In the second winning case, ProvVerify accepts under an honest key, so the new statement carries a ciphertext that decrypts to an honest tag τ. By simulation extractability, $\mathcal{A}$ knows an origin ρ' and randomness a such that this ciphertext is $\mathrm{Enc}_{lpk}(H_{\mathrm{tag}}(lc, \rho'); a)$, so, by correctness of the encryption scheme, $H_{\mathrm{tag}}(lc, \rho') = \tau$. This is possible only if ρ' is the honest origin, which contradicts derivation binding, or if $\mathcal{A}$ found a collision of SHA256, which happens only with negligible probability.

Unlinkability. The challenge statement is $x^* = (ctx^*, \sigma, Y, h, (lc^*, lpk^*, ct))$, and $\mathcal{A}$ must tell whether it comes from S_0 or from S_1. By zero knowledge, the proofs reveal nothing beyond the statements, so $\mathcal{A}$ can use only the statements. The context ctx^*, the linking context lc^* and the key lpk^* are chosen by $\mathcal{A}$. The key lpk^* was created by $\mathcal{O}^{\mathrm{Key}}$, so $\mathcal{A}$ does not know its secret key, and, by IND-CPA security, the ciphertext ct reveals nothing about the tag it contains. $\mathcal{A}$ did not ask for the node of the challenge under either seed, so, by output unlinkability, it cannot tell whether (Y, h) comes from S_0 or from S_1. So $\mathcal{A}$ wins with probability at most $1/2$ plus a negligible function. □

The theorems do not cover replay of an honest statement with its proof, as noted after the definition of non-frameability. The extensions of Section 4.6 are argued informally in that section.

6 Application: cross-chain transfers

We describe one application of the constructions of Section 4. It uses Form 2 as it is, so its properties are those of Theorem 5.3.

Problem. A user holds a seed S from which a Bitcoin wallet and a Solana wallet are derived, with BIP32 over secp256k1 and SLIP-10 over Ed25519. The user wants to move value from Bitcoin to Solana through a bridge or an instant-exchange service. The service wants to pay out only to a Solana address that belongs to the same wallet as the Bitcoin address that made the deposit, without learning S, the derivation paths, or any other address of the user.

Why it is relevant. In an ordinary cross-chain transfer, the payout address is not related to the source address. An attacker who intercepts the request after the deposit can replace the payout address with its own, and the service cannot detect it. If the service pays out only to an address with the same origin as the source, such a substitution fails.

Construction used. We use Form 2 (Section 4.4), with one derivation proof for each chain, so each proof uses only one curve, and the two proofs of one transfer carry the same tag. The two addresses belong to the trees of two different schemes, so their only common origin is the seed, and both proofs start from it. We choose Form 2 for simplicity; the other forms can also be used, with the properties given by their theorems in Section 5.2. With Form 1 or its deferred variant, the user also gives a provenance proof that the two addresses share an origin. With Form 3, the service must hold the linking key.

Protocol. Let σ_1 be BIP32 over secp256k1 and σ_2 SLIP-10 over Ed25519. The user derives the Bitcoin address Y_1 from S along a path p_1 under σ_1, and the Solana address Y_2 from S along a path p_2 under σ_2. Let op be the Bitcoin outpoint that the deposit spends, controlled by Y_1, and let v be the amount. The operation context is $ctx := (\texttt{BTC}, \texttt{SOL}, op, v, Y_1, Y_2)$, and the linking context is $lc := ctx$, which is different for every transfer since an outpoint can be spent only once.

1. The user computes the two derivation proofs

   $(x_1, \pi_1) \leftarrow \mathsf{DerivProve}(\mathrm{pp}, S, \sigma_1, p_1, ctx, lc, \perp), \qquad (x_2, \pi_2) \leftarrow \mathsf{DerivProve}(\mathrm{pp}, S, \sigma_2, p_2, ctx, lc, \perp).$

   Both statements carry the same tag $\tau = H_{\mathrm{tag}}(lc, S)$.

2. The user sends Y_2 to the service, which records it as the payout address. The service gives the user a Bitcoin deposit address A_{dep} that it controls, and the user sends v from op to A_{dep}.

3. The user sends (x_1, π_1, x_2, π_2) and a reference to the deposit transaction to the service.

4. The service recomputes ctx from the deposit transaction and from the payout address it recorded in step 2, and checks that the deposit spends op, that op is controlled by Y_1, that v was sent to A_{dep}, and that the transaction has the required confirmations. It checks that the public values of x_1 and x_2 are Y_1 and the recorded payout address, that $\mathsf{DerivVerify}$ accepts both proofs with this ctx, that $\mathsf{ProvVerify}(\mathrm{pp}, (x_1, x_2), \perp, \perp) = 1$, and that τ was not recorded before.

5. If all checks pass, the service records τ and pays out to Y_2 on Solana.

Integration with an instant-exchange service. We show how the protocol can be attached to the API of a typical instant cross-chain exchange service, which we call Swapper X. This is a design only: no integration has been performed, and Swapper X is not a specific existing service. We model Swapper X as offering three commands.

- `CreateExchange(sourceCurrency, destinationCurrency, payoutAddress, metadata)` starts an exchange and returns an identifier and a deposit address; `metadata` is a short field that can carry a reference value.
- `CheckStatus(exchangeId)` returns the status of the exchange.
- `PushOrRefund(exchangeId, refundAddress?)` completes a pending exchange or refunds it, to the sending address or to a given refund address.

The integration adds a verification service, which runs step 4 of the protocol before the payout. The user calls `CreateExchange(BTC, SOL,` Y_2`, ref)`, where `ref` is a hash of the two proofs, since the proofs are too large for `metadata`, and it sends the statements and the proofs to the verification service. Swapper X returns the deposit address A_{dep}, and the user sends v from op to it. When `CheckStatus` shows that the deposit is confirmed, the exchange is paused before payout, and the verification service runs the checks of step 4, with the `payoutAddress` of the exchange as the recorded payout address. If all checks pass, Swapper X pays out to Y_2; otherwise the integration calls `PushOrRefund` to return the deposit. If the refund goes to a separate address Y_3, the user derives it from S under σ_1 before step 1, adds it to ctx, and gives a third derivation proof with this ctx and the same linking context, so that the refund also goes to the user's wallet. In this case the verification service also adds the `refundAddress` of the exchange to ctx when it recomputes it, and checks that the public value of the third proof is this `refundAddress`.

The integration does not change how Swapper X handles exchanges and payouts; it only adds one check before the payout. It lets the service enforce a policy that the source and destination addresses come from the same wallet, which closes the substitution attack without asking the user to reveal anything about its wallet.

7 Performance

Every construction of Section 4 contains derivation proofs, so the cost of a derivation proof is the basic cost of all of them. We report this cost here. The constructions also prove the link value (a commitment, a tag or an encrypted tag) or several derivations in one proof, and we did not measure these additions; the numbers below are therefore a lower bound on the cost of the constructions, not their exact cost.

Setup. We measured single derivation proofs on secp256k1, for one value on one derivation path, for BIP32 and for QBIP32 [2], in our Plonky3 implementation. The hash bindings use Keccak-256 in place of SHA-256. Benchmarks were run on an AMD Ryzen 9 9950X3D system with 16 physical cores and 32 hardware threads, 64 GB of RAM, and Ubuntu 24.04.4 LTS. The implementation was compiled with Rust 1.98.0-nightly and uses Plonky3 v0.6.0[1] with a hiding FRI commitment for zero knowledge.

Results. Table 1 reports prove time, verify time and proof size. A full-path proof starts from the seed and proves every step down to a node at the given depth; at depth 1 the path is `m/44'`, and at depth 5 it is `m/44'/0'/0'/0/0`. A pruned proof starts from an anchor, and proves the last hardened step and the non-hardened steps below it. For the QBIP32 full-path proof at depth 5 we made five runs, and all of them are within 2% of the value in the table.

Scheme	Relation	Depth	Prove (s)	Verify (ms)	Proof (MB)
QBIP32	full-path	1	2.380	196.352	3.758
QBIP32	full-path	2	2.446	206.467	4.305
QBIP32	full-path	3	2.437	213.786	4.708
QBIP32	full-path	4	2.874	261.782	6.265
QBIP32	full-path	5	3.220	309.189	7.820
QBIP32	pruned	4	2.736	203.720	3.319
QBIP32	pruned	5	3.118	234.457	4.196
BIP32	pruned	4	2.330	185.210	3.674
BIP32	pruned	5	2.896	226.894	4.997

Table 1: Cost of one derivation proof on secp256k1.

Most of the cost is the same for every path: a QBIP32 full-path proof at depth 1 already takes 2.4 seconds, and going from depth 1 to depth 5 adds about 0.8 seconds. This is expected for a STARK, where the trace is padded to a power of two and part of the work does not depend on the path. At the same depth, the pruned proof saves little proving time but gives a much smaller proof: for QBIP32 at depth 5 it takes 3.1 seconds in place of 3.2, with a proof of 4.2 MB in place of 7.8 MB. BIP32 and QBIP32 prove different relations, so their numbers should not be compared directly.

8 Related work and discussion

Hierarchical deterministic wallets. BIP32 [14] lets anyone derive the public keys below a non-hardened extended public key, but this reveals the chain code and the structure of the wallet,

[1] `https://github.com/Plonky3/plonky3`.

and it does not work for hardened steps. Provenance proofs show only the relation between selected keys, in zero knowledge, and they work for hardened and pruned derivations through the relations of ZKPoSP [2]. The fixed seed value of ZKPoSP [2, Sec. 7.5] is Form 2 with one constant linking context.

Linkable anonymous authentication. Linkable ring and group signatures [8, 13] let anyone recognize two signatures of the same hidden signer. Direct Anonymous Attestation [3] limits this linking to signatures with the same basename, which plays the role of our linking context, and event-oriented authentication [7] limits it to one event. CryptoNote [12] uses deterministic key images to detect a second spend of the same output. These systems link signatures made with the same key, while provenance proofs link different public keys through their hidden origin, possibly across schemes and curves.

Anonymous credentials. Anonymous credentials [4] let a user prove possession of a credential issued by an authority, without linking its presentations. Provenance proofs work on the derivation state of a wallet, not on issued attributes, and the two can be combined, for example to prove that a credential key and a wallet key come from the same origin.

Security against quantum adversaries. We do not analyse security against quantum adversaries; all our arguments are against PPT adversaries in the ROM. For the status of the ZKPoSP derivation relations against quantum adversaries, see [2].

9 Conclusion

We gave a model for provenance proofs, in which a user proves that several public keys come from one hidden origin, in one derivation tree or across schemes, and controls who else can recognize this relation. We gave four constructions that realize different choices of who can link: only the user, with Form 1 and its deferred variant; anyone within a linking context, with Form 2; or a designated authority, with Form 3. We also presented an application that shows how the constructions are used: a cross-chain service that pays out only to an address from the same wallet as the deposit.

Generative AI Usage

Generative-AI tools were used for editorial assistance in the preparation of this paper. All technical content was reviewed and validated by the authors.

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

[1] J. Bos, L. Ducas, E. Kiltz, T. Lepoint, V. Lyubashevsky, J. M. Schanck, P. Schwabe, G. Seiler, and D. Stehlé, "CRYSTALS-Kyber: A CCA-Secure Module-Lattice-Based KEM," in *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pp. 353–367, 2018.

[2] Vincenzo Botta, Michal Pospieszalski, Emanuele Ragnoli, Justus Ranvier. *ZKPoSP: Post-Quantum Zero-Knowledge Proofs for Hierarchical Deterministic Wallets*. Cryptology ePrint Archive, Paper 2026/1508, 2026. `https://eprint.iacr.org/2026/1508`

[3] E. Brickell, J. Camenisch, and L. Chen, "Direct Anonymous Attestation," in *Proceedings of the 11th ACM Conference on Computer and Communications Security (CCS)*, pp. 132–145, 2004.

[4] J. Camenisch and A. Lysyanskaya, "An Efficient System for Non-transferable Anonymous Credentials with Optional Anonymity Revocation," in *Advances in Cryptology – EUROCRYPT 2001*, LNCS 2045, 2001.

[5] A. Chiesa and G. Fenzi, "zkSNARKs in the ROM with Unconditional UC-Security," in *Theory of Cryptography – TCC 2024*, LNCS, pp. 67–89, 2024.

[6] S. Goldwasser and S. Micali, "Probabilistic Encryption," *Journal of Computer and System Sciences*, vol. 28, no. 2, pp. 270–299, 1984.

[7] P. Li, J. Lai, and Y. Wu, "Event-Oriented Linkable and Traceable Anonymous Authentication and Its Application to Voting," *Journal of Information Security and Applications*, vol. 60, Art. 102865, 2021.

[8] J. K. Liu, V. K. Wei, and D. S. Wong, "Linkable Spontaneous Anonymous Group Signature for Ad Hoc Groups," in *Information Security and Privacy*, ACISP 2004, LNCS 3108, pp. 325–335, 2004.

[9] V. Lyubashevsky, C. Peikert, and O. Regev, "On Ideal Lattices and Learning with Errors over Rings," *Journal of the ACM*, vol. 60, no. 6, pp. 1–35, 2013.

[10] National Institute of Standards and Technology, "Module-Lattice-Based Key-Encapsulation Mechanism Standard," Federal Information Processing Standards Publication 203 (FIPS 203), 2024.

[11] O. Regev, "On Lattices, Learning with Errors, Random Linear Codes, and Cryptography," *Journal of the ACM*, vol. 56, no. 6, pp. 1–40, 2009.

[12] N. van Saberhagen, "CryptoNote v2.0," 2013. `https://cryptonote.org/whitepaper.pdf`

[13] P. P. Tsang and V. K. Wei, "Short Linkable Ring Signatures for E-Voting, E-Cash and Attestation," in *Information Security Practice and Experience*, ISPEC 2005, LNCS 3439, 2005.

[14] P. Wuille, "BIP 32: Hierarchical Deterministic Wallets," Bitcoin Improvement Proposal 32, 2012.