PCC: Proof-Carrying Credentials for Enhanced Anonymity and Post-Quantum Delegation

Zixuan Wang*, ZHENG Tianyu†, Yuan Lu‡, Bin Xiao*
*The Hong Kong Polytechnic University
24052881r@connect.polyu.hk, b.xiao@polyu.edu.hk
†Department of Computing, The Hong Kong Polytechnic University
tian-yu.zheng@connect.polyu.hk
‡Agency for Science, Technology and Research (A*STAR)
lu_yuan@a-star.edu.sg

***Abstract*—Delegatable Anonymous Credentials (DACs) let an authority be delegated through a chain of intermediaries and later exercised anonymously, as in DeFi credit delegation. Because presentation is the operation a holder repeats at every use, its cost is what matters in practice; yet existing DACs build this presentation from the delegation chain itself. We call this limitation *prover awareness*: to present, a holder must possess and process delegation-lineage material whose shape mirrors the chain. It has two faces: its size makes the presentation cost grow with the delegation depth, and its content exposes the holder to delegation-topology metadata that, in incentive-driven and compliance-aware settings, enables lineage-based policy vetoes, strategic disintermediation, and de facto censorship in otherwise permissionless networks.**

We formalize the privacy face as *Enhanced Delegation Anonymity* (EDA), a leakage-parametrized two-observer notion combining verifier-side anonymity with holder-side topology obliviousness, the latter given both as a simulation definition and as an indistinguishability game over paths with identical allowed holder leakage. Because both faces share one cause, the chain-structured credential, a single change of representation removes them together: our framework *Proof-Carrying Credentials* (PCC) replaces the explicit chain with a single recursively updated proof-carrying state via Non-Uniform Incrementally Verifiable Computation (NIVC), making the presentation independent of the delegation depth and hiding the topology at once, while preserving restricted delegation and expressive presentation. Unlike prior constant-size DACs tied to bilinear pairings, PCC is hardness-agnostic and admits an explicit post-quantum-compatible instantiation in the QROM from transparent FRI-based recursion and hash-based signatures.

On consumer-grade hardware, the Halo2 (non-post-quantum) instantiation keeps the verifier-facing proof constant at 7.344 KB with 35–36 ms verification across all tested depths, and a 128-bit Plonky2 (post-quantum) instantiation keeps a constant 157.9 KB proof and 3.8 ms verification across depths $L \in \{1, 2, 4, 8, 16\}$, at 1.03 s per delegation step. Holder-side topology hiding and depth-independent presentation are thus achievable at practical recursive-proving cost.

1. Introduction

Transitive authorization is becoming increasingly important in decentralized systems [1]–[3]. Modern applications rely on delegation rather than outright transfer of authority: users delegate wallet operations to agents, organizations delegate access rights across services, and DeFi protocols [4], [5] delegate financial capabilities while retaining ownership of the underlying assets. Aave's credit delegation illustrates this pattern: a depositor can delegate borrowing capacity to another party without transferring the collateral [6], decoupling the right to act from ownership of the asset. Although current deployments often expose only shallow delegation interfaces, the same pattern naturally extends to settings where capabilities are restricted, re-delegated, and eventually exercised by downstream services across administrative or economic domains.

Delegatable Anonymous Credentials (DACs) [7]–[15] are the natural cryptographic primitive for realizing such transitive authorization. A DAC lets a holder derive a restricted downstream credential from an existing one, and lets the final presenter prove authorization without revealing unnecessary information about the credential or the delegation path.

Need for efficient DAC presentations In a DAC system, the presentation (or showing) is the use-time operation: after a credential is issued or delegated, every exercise of the delegated capability requires the presenter to convince a verifier that the action is authorized and restriction-preserving [11], [12]. Unlike one-time issuance or delegation, this cost is incurred on every use. It may be borne by the holder directly, by wallet software interfacing with decentralized applications [16]–[19], or by an outsourced proving service when zero-knowledge generation is expensive [20]–[23]; in each case it governs the latency, communication, and verification overhead of exercising delegated authority. Presentation efficiency is therefore not an implementation detail but a precondition for anonymous delegation to be used in practical decentralized applications. This burden compounds with delegation depth: when a presentation grows with the length of the delegation chain, each additional hop raises the price of every presentation.

Despite the importance of presentation efficiency, no existing DAC achieves a presentation whose cost is in-

dependent of the delegation depth. For early theoretical constructions [7]–[9], the credential is an explicit chain, so the presentation proof inherently grows with the delegation depth. Even for recent schemes [11]–[13], [15] that are expected to be practical, improving concrete efficiency, strengthening anonymity, or compressing the credential to constant size, the showing remains conditioned on the delegation chain, and unsurprisingly, when the chain is deep, it still carries delegation-dependent material (commitment vectors or witnesses) whose size grows with the depth; relaxing privacy only improves constants. Moreover, this path-structured material even leaks the delegation topology in addition to its cost, because its size and shape track the delegation path; as such, the holder or an outsourced prover learns the chain depth and per-hop structure even when the verifier-facing presentation is zero-knowledge. Given that, it becomes essential to make DAC presentations both efficient and topology-hiding, even for deep delegation chains.

TABLE 1. COMPARISON OF TYPICAL DAC SCHEMES.

Scheme	Ref.	Anon. †	Topo. ‡	PQE §	|Pre| ¶	|Cred|
BCC+09	[8]	$\mathcal{S}$	✗	✗	$O(L\lambda)$	$O(L)$
P-UC-DAC	[11]	$\mathcal{W}$	✗	✗	$O(L)$	$O(L)$
Mercurial-DAC	[10]	$\mathcal{W}$	✗	✗	$O(L)$	$O(L)$
TDAC	[14]	$\mathcal{W}_L$	✗	✗	$O(n)$	$O(1)$
EQS-DAC	[12]	$\mathcal{S}$	✗	✗	$O(L)$	$O(1)$
Mercurial-DAC*	[13]	$\mathcal{S}$	✗	✗	$O(L)$	$O(L)$
FDAAC-CR	[15]	$\mathcal{S}$	✗	✗	$O(L)$	$O(1)$
PCC (Ours)	–	$\mathcal{S}$	✓	✓	$\boldsymbol{O(1)}$	$O(1)$

† **Anonymity**: $\mathcal{S}$ = strong anonymity; $\mathcal{W}$ = weak anonymity, protecting only verifier-side presentation privacy; $\mathcal{W}_L$ = level-conditioned weak anonymity, i.e., verifier-side anonymity only among users at the same delegation level L.

‡ **Topo.**: hides delegation topology, including chain depth and per-hop structure, from the holder/prover.

§ **PQE**: post-quantum-compatible instantiation under explicit QROM assumptions.

¶ |Pre| and |Cred| report asymptotic growth mainly with respect to the delegation depth L. Attribute/policy-dependent public inputs or witnesses are not counted unless explicitly shown; the full presentation transcript may grow with their encoding. λ is the security parameter and n denotes the show-protocol witness size when applicable.

Why do we need topology hiding? Consider a multi-tier lending service inspired by Aave's credit delegation [24]. A collateral provider passes borrowing authority through intermediaries to a manager arranging loans. The same manager may receive the same rights from the same direct intermediary via two different upstream paths. Even at equal public disclosure and permitted local leakage, an explicit credential can reveal the upstream counterparties and depth to the manager or its prover. Derecho illustrates how provenance affects transaction acceptance [25]: here, surplus lineage combined with external records could suggest a link to a sensitive jurisdiction and lead the manager to decline an otherwise eligible loan. Required risk checks and disclosures remain mandatory. In a tiered fee-sharing arrangement, learning the depth or identities of upstream intermediaries can also let the manager seek a shorter channel or bypass a counterparty to retain more fees. The privacy target is the additional upstream information that the holder's credential and proving interface would otherwise expose.

1.1. Practical Obstacles to Efficient DAC Presentations

Nevertheless, designing a DAC with an efficient presentation is challenging: to prove an action authorized, the protocol must account for every delegation step and every restriction accumulated along the chain, yet doing so without paying a depth-dependent cost or leaking the chain's structure is far from obvious. Conventional designs settle this by letting the prover carry per-step credential material, trading away exactly the efficiency and topology privacy we seek.

Practicality issues of earlier theoretical DACs. DACs have been studied for nearly two decades, but the early constructions were primarily of theoretical interest due to their path-dependent proof complexity. The first DAC construction of Chase and Lysyanskaya [7] established that anonymous credentials could be delegated, but it relied on general non-interactive zero-knowledge proofs for complex NP statements; as later observed by Belenkiy et al. [8], this led to credential-chain representations whose size grows super-polynomially with the delegation depth and hence could not support non-constant-depth delegation in practice. Belenkiy et al. [8] made a substantial theoretical improvement by introducing randomizable proofs and reducing the proof size and proving time for a credential at depth L to $O(L\lambda)$, where λ is the security parameter. However, this still leaves both the proof and the prover's work linear in the delegation depth. Malleable-signature-based DACs [9] further refined the security definitions and obtained stronger context-hiding guarantees, but their resulting credentials still scale linearly with the number of delegations. Thus, while these works established the feasibility and formal foundations of strongly private delegation, their authorization evidence remained explicitly path-structured.

This path dependence is the main practicality obstacle in the early theoretical line. Each delegation contributes proof, signature, or transformation material that must be randomized, transformed, or re-proved when the downstream credential is presented. Consequently, zero-knowledge hides the chain from the verifier but does not eliminate the chain-shaped material that the prover must carry and process. More recent constructions improve on this foundational line in divergent ways, some relaxing privacy for concrete efficiency, others strengthening privacy with algebraic credentials such as mercurial signatures [10], [13], threshold credentials [14], or equivalence-class based schemes [15], as Table 1 summarizes, none removes the depth dependence from the presentation.

P-UC-DAC: practicality by exposing lineage. P-UC-DAC [11] is a major step toward practical DACs: a UC-secure construction with attributes whose pairing-based presentations avoid general-purpose zero-knowledge proofs. It obtains this efficiency through a presentation-centric privacy target—the presentation hides the credential chain from the verifier, but the delegation interface hands the delegatee the full chain, including the signatures, public keys, and attributes of all previous levels. The credential thus remains path-structured: a level-L

credential is L links, and a presentation proves possession of every link back to the issuer. P-UC-DAC therefore leaves the presentation depth-dependent and exposes the full delegation lineage to the downstream holder.

EQS-DAC: strong anonymity, depth-dependent showings. More recently, EQS-DAC [12] strengthens privacy by replacing explicit credential chains with SPSEQ-UC signatures over updatable set commitments: each delegation appends one set commitment to a signed commitment vector, and public randomization with key switching makes delegations and showings unlinkable. This avoids general-purpose zero-knowledge proofs and keeps the credential signature constant-size. Yet the chain is not removed from the presentation: the showing carries a commitment vector of length $\ell = \Theta(L)$, costing $(\ell+5)|\mathbb{G}_1|+|\mathbb{G}_2|+|\mathbb{Z}_p|$, and the holder additionally stores opening randomness linear in the chain. Thus EQS-DAC strengthens anonymity over P-UC-DAC, but the showing remains $\Theta(L)$ and the delegation depth stays visible in the holder's own state, so even strong verifier-side anonymity does not hide the topology from the prover.

TDAC: closest to depth-independence. TDAC [14] comes closest to our target: it compresses the delegated credential into a short decryption key and avoids carrying an explicit delegation-chain witness, so its showing cost is parameterized by the number of attributes rather than the delegation depth. This independence is not genuine, however. The protocol is level-dependent and anonymity holds only among users at the same level, so the level, i.e. the depth, is exposed by construction; and when delegation depth is encoded through accumulated attributes, the showing cost grows with the number of delegated attributes, reintroducing the dependence indirectly.

Can a DAC presentation be efficient and topology-hiding at once? Across these constructions, the obstacle is the same: the prover holds and proves over material whose size and shape mirror the delegation chain, simultaneously inflating the presentation cost and leaking the chain's topology. Our key observation is that these are not two independent defects but two symptoms of one cause, and can therefore be removed together. Carrying the accumulated authorization in a single depth-independent object, rather than a chain-structured credential, would make both the presentation cost and the exposed topology independent of the delegation depth.

Facing this obstacle in state-of-the-art DACs, we are asking the following question:

Can strong anonymity DACs achieve depth-independent presentation and topology-hiding?

1.2. Our Contribution

We answer this question affirmatively with PCC (Proof-Carrying Credentials), a generic DAC framework whose presentation is depth-independent and topology-hiding while preserving strong anonymity and restricted delegation. The idea is in the name: rather than carrying the delegation chain, a PCC credential carries a *proof* of it. Using Non-Uniform Incrementally Verifiable Computation (NIVC), PCC compiles the delegation lineage into a single recursively updatable proof-carrying state that each delegation step extends in place, so a holder retains only its terminal state, local attributes, and a constant-size recursive proof rather than a verifiable list of upstream ancestors or intermediate delegation signatures. This one representational change removes both limitations at once: a final presentation extends the same state with a presentation predicate, producing a verifier-facing proof whose size and verification cost are independent of the delegation depth, while the holder's local view no longer exposes delegation topology. Unlike prior constant-size DACs tied to bilinear pairings, PCC is structurally hardness-agnostic and admits an explicit post-quantum-compatible instantiation. In sum, our contributions are:

- **New two-observer privacy notion, Enhanced Delegation Anonymity (EDA), that adds holder-side topology obliviousness to strong anonymity.** We identify *prover awareness*: across the DAC design space, a holder must process chain-shaped lineage material to present. Its topology leakage is the privacy face of the same defect that drives depth-dependent presentation cost, so we formalize it as *Enhanced Delegation Anonymity* (EDA): a leakage-parametrized two-observer notion pairing *strong anonymity* over the verifier statement stmt_V with *holder-side topology obliviousness* over the holder view View_H, the latter relative to an explicit allowed-leakage function $\ell_H(\mathsf{path})$. Parameterizing on ℓ_H casts the informal question of how much structure a holder must learn into the standard simulation/indistinguishability paradigm: we give topology obliviousness as both a simulation definition and an equivalent indistinguishability game over paths with identical allowed leakage. This is orthogonal to identity anonymity: even BCC+09 [8], whose strong anonymity hides delegator identities and unlinkably re-randomizes presentations, fails it, since hiding *who* delegated does not stop the holder from retaining $O(L)$ lineage and learning the depth L. Moreover, the formulation is of independent interest for any delegation or provenance system whose intermediaries should not learn upstream structure, e.g., wallet delegation and service-mediated authorization chains.
- **Generic recursive DAC framework, Proof-Carrying Credentials (PCC), that compiles a delegation chain into a single, recursively updatable proof-carrying state, making both presentation cost and exposed topology independent of the delegation depth.** Following our observation, a PCC holder carries the accumulated authorization as a single proof-carrying state rather than the explicit per-hop history (each level's signature, public key, and attribute set). We build this over Non-Uniform Incrementally Verifiable Computation (NIVC), whose non-uniformity lets one recursion switch among the heterogeneous issuance, delegation, and presentation predicates. Each delegation folds one hop into the state by re-verifying the carried proof and checking the new transition, so the holder keeps only its terminal state, local attributes, and an $O(1)$ recursive proof, not the chain. Validating a depth-L chain thus reduces to verifying one recursive proof, and a final presentation extends the state with a presentation predicate to yield a verifier-facing proof whose size and verification cost are independent of depth. This

single change collapses the $O(L)$ credential and presentation of prior schemes to $O(1)$ and, since the holder no longer holds the chain, delivers EDA's holder-side topology obliviousness as a by-product.

- **Co-designed transition predicates and a depth-hiding verification interface that harvests depth-independent, topology-hiding presentation while still preserving restricted delegation, expressive policies, and standard DAC security.** Collapsing the chain into one opaque proof (Contribution 2) must not sacrifice what makes a DAC useful: enforcement of the accumulated restrictions and support for expressive presentation policies, which verifier-only zero-knowledge cannot deliver [10], [13]. We co-design a predicate family $\Phi = \{\phi_{\mathsf{issue}}, \phi_{\mathsf{del}}, \phi_{\mathsf{pre}}\}$ enforcing every policy inside the recursive step: ϕ_{del} checks attribute monotonicity $A_{i+1} \subseteq A_i$ and per-hop authorization (restricted delegation), and ϕ_{pre} enforces selective disclosure $A_{\mathsf{pre}} \preceq A_L$ (expressive policies). Compatibility with topology hiding requires a *depth-hiding* NIVC interface: the verifier checks only the root and terminal states, the recursive proof, and a public terminal condition, leaving the depth and selector sequence proof-internal. This wrapper adds no hardness assumption beyond the NIVC, only packaging each transition and invoking the NIVC prover over Φ; a wrapper reduction shows it inherits completeness, simulation-extractability, and zero-knowledge, hence correctness, delegation unforgeability, and EDA. PCC thus harvests depth-independence and topology hiding without surrendering restricted delegation, expressiveness, or security, and remains generic over any depth-hiding NIVC.
- **The first DAC to combine a constant-size presentation and credential with a post-quantum-compatible instantiation, implemented over both conventional and post-quantum recursive backends, with an extensive depth-scaling evaluation.** Because PCC is generic over any depth-hiding NIVC (Contribution 3), we instantiate it in two regimes: a Halo2 backend for the conventional setting and a transparent post-quantum one combining Plonky2-style FRI recursion over Goldilocks, Poseidon hashing, and a hash-based (SPHINCS$^+$-style) signature, analyzed in the QROM without pairings or a trusted setup. We measure verifier-facing proof size, verification latency, and the per-delegation cost across depths $L \in \{1, 2, 4, 8, 16\}$, isolating recursion's role through monolithic-versus-recursive Halo2 and Fractal-versus-Plonky2 comparisons. The proof size and verification time stay constant across all tested depths (7.344 KB / $\sim$36 ms for Halo2; 157.9 KB / 3.8 ms for 128-bit Plonky2); the only depth-dependent work is a per-hop delegation step, paid by the delegator and itself stable across depths (e.g., 1.03 s post-quantum), confirming that depth-independence stems from recursion, not the backend. These results show, for the first time, that holder-side topology hiding and depth-independent presentation are simultaneously achievable at practical recursive-proving cost, including under an explicit post-quantum-compatible instantiation.

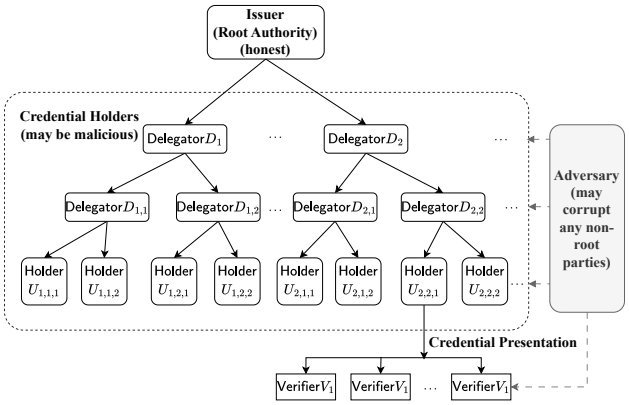


Figure 1. System and threat model. The root authority is honest, while non-root credential holders and verifiers may be malicious and collude. Credential holders may act as delegators, delegatees, or presenters across the issuance, delegation, and presentation phases.

2. Models and Technical Overview

This section sets up the problem and previews our solution before the formal treatment in Section 3. We first fix the system and threat model together with the privacy goal (Section 2.1), then state the central design challenge and the techniques PCC uses to meet it (Section 2.2), and finally introduce the notation and the single non-standard primitive our construction relies on (Section 2.3).

2.1. System and Threat Model

System and roles. We consider a hierarchical credential system with a **Root Authority**, intermediate **Users**, and external **Verifiers**; a user acts as *delegator*, *delegatee*, or *presenter* across the *issuance*, *delegation*, and *presentation* phases (Figure 1). The Root Authority publishes an authenticated root anchor $\mathsf{RootAnchor}$ that verifiers authenticate before accepting presentations; the initial credential state is trusted only once certified by a root issuance signature and verified inside the recursive proof.

Threat model. A PPT adversary may adaptively corrupt users, obtain their keys and credentials, collude, share material, attempt unauthorized delegations, and forge presentations for unauthorized attributes; verifiers may also be malicious. The adversary never obtains $\mathsf{sk}_{\mathsf{root}}$ and cannot make a verifier accept an unauthenticated root anchor, and issuance is answered only under the fixed $\mathsf{RootAnchor}$.

Privacy goal. We target *Enhanced Delegation Anonymity* (EDA), which constrains two observers. A *verifier*, seeing only the presentation transcript, learns nothing beyond the authorized statement and the intentional disclosure (verifier-side anonymity). A *holder*, whose local view comprises the credential, proving interface, and presentation, is not forced to learn delegation-topology metadata beyond an explicit allowed leakage $\ell_H(\mathsf{path})$: beyond ℓ_H it cannot cryptographically verify remote ancestors, intermediate credentials or signatures, or the depth L (holder-side topology obliviousness). EDA hides neither application-required disclosures nor the immediate delegation edge, and out-of-band leakage is out of scope. We formalize both notions in Section 3.2.

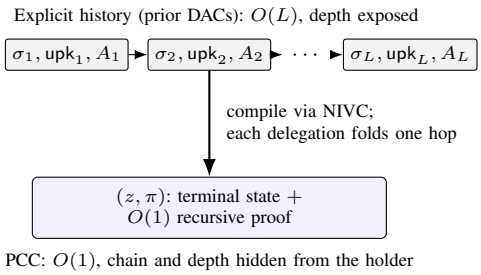


Figure 2. The key representational change. Prior DACs require the holder to carry and prove the explicit delegation chain, so both its size and the holder's view grow with the depth L. PCC compiles the same lineage into a single, recursively updated proof-carrying state (z, π): each delegation folds one hop into the state, and the holder retains only the terminal state and a constant-size recursive proof, hiding the chain and the depth.

2.2. Challenges and Our Techniques

Realizing EDA with a depth-independent presentation raises three obstacles, which we address in turn.

Challenge I: a depth-independent, topology-hiding presentation. A holder must prove that its authority traces back to the root, yet neither the verifier-facing presentation nor the holder's own view may grow with or reveal the delegation depth. The obvious attempt does not suffice: a zero-knowledge presentation computed over an explicitly stored chain hides the transcript from the verifier, but the holder still stores and proves the per-hop lineage, so its view exposes the depth and the presentation cost grows with L.

Our approach. We compile the lineage into a single, recursively updated proof-carrying state (Figure 2): each delegation folds one hop into the state by re-verifying the carried proof and checking the new transition, so a holder keeps only its terminal state and a constant-size recursive proof, never the chain. We obtain this from incrementally verifiable computation and equip it with a depth-hiding verification interface (Section 2.3), under which the verifier checks only the root and terminal states, the recursive proof, and a public terminal condition, while the depth and the step sequence stay internal to the proof.

Challenge II: preserving delegation semantics without the chain. Once the explicit lineage is gone, the recursion must still enforce the restrictions accumulated along the path, support expressive presentation policies, and remain unforgeable; moreover it must do so across heterogeneous operations, since issuance, delegation, and presentation enforce different checks.

Our approach. We use a non-uniform variant of incrementally verifiable computation (Definition 6) with a co-designed predicate family $\{\phi_{\mathsf{issue}}, \phi_{\mathsf{del}}, \phi_{\mathsf{pre}}\}$ that enforces every policy inside the recursive step: the delegation predicate checks attribute narrowing and per-hop authorization (restricted delegation), and the presentation predicate enforces selective disclosure (expressive policies), with the concrete relations given in our construction (Section 4.2). Because the construction merely invokes the recursive prover over this family, it adds no hardness assumption beyond the underlying scheme and inherits its completeness, simulation-extractability, and zero-knowledge, hence correctness, delegation unforgeability, and EDA.

Challenge III: a uniform and practical recursive step. Recursion is the dominant cost, and supporting heterogeneous predicates naively would require three large, independent step circuits. We observe, however, that the three predicates act on a common state $z = \mathsf{Com}(\mathsf{upk}, A; r)$ and are assembled from the same in-circuit gadgets: commitment opening, Poseidon hashing [26], hash-based signature verification, and attribute-set relations. They differ in logic, which is precisely why non-uniformity is needed to switch among them, yet they are homogeneous in composition, so the step circuits share a common gadget set and each is dominated by the same Poseidon core.

Our approach. Together with the per-step cost of the recursive scheme, this co-design keeps the marginal cost of one delegation uniform across the three operations and independent of the prior depth, rather than scaling with the chain; we treat this as a property of the predicate family rather than a separate optimization, and report the concrete per-step costs in Section 6.

2.3. Notation and Primitives

We write $[n] = \{1, \ldots, n\}$, use $\mathsf{negl}(\lambda)$ for negligible functions, and $\approx_c$ for computational indistinguishability. We use an EUF-CMA signature scheme Σ, binding/hiding commitments ($c \leftarrow \mathsf{Com}(m; r)$), and domain-separated hashes $H_{\mathsf{state}}, H_{\mathsf{sig}}, H_{\mathsf{FS}}, H_{\mathsf{pre}}$ with fixed tags. The only non-standard primitive is the following depth-hiding NIVC abstraction.

Non-Uniform IVC. The only non-standard primitive PCC relies on is a depth-hiding non-uniform IVC (NIVC): a recursive proof system $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Prove}, \mathsf{Verify})$ for a family of transition predicates $\mathsf{F} = \{\mathsf{F}_1, \ldots, \mathsf{F}_k\}$ over a shared public instance ρ, which in PCC contains the root anchor $\mathsf{RootAnchor}$. Each Prove step folds one transition: from a selector j, the current and next states z_i, z_{i+1}, a witness w_{i+1}, and the previous proof π_i, it outputs a *constant-size* proof π_{i+1} attesting that the prior recursion is valid and $\mathsf{F}_j(\rho, z_i, z_{i+1}, w_{i+1}) = 1$; the base step starts from the empty proof $\perp$, and root authenticity is enforced by ϕ_{issue} using the root verification key carried in ρ. We assume the standard completeness, simulation-extractability, and zero-knowledge guarantees; the full syntax and these properties appear in Appendix A.1 (Definition 6).

Depth-Hiding Verification Interface. Some standard presentations of IVC/NIVC expose the step count or selector sequence to the verifier. PCC, in contrast, requires a depth-hiding verification interface: the verifier checks only the root state, the terminal state, the recursive proof, and a public terminal condition. The actual delegation depth and the non-terminal selector sequence remain hidden inside the recursive proof. If a concrete backend requires a fixed recursion horizon, PCC can be compiled for a public upper bound $L_{\max}$ and padded with dummy transitions; in that case, $L_{\max}$ is public leakage, but the actual delegation depth remains hidden.

3. Delegatable Anonymous Credentials

In this section, we formally define the syntax and security model for Delegatable Anonymous Credentials (DAC). We model DAC as a hierarchical system where users can issue, delegate, and present credentials.

3.1. Syntax of DAC

Here, Cred denotes an abstract credential object; the syntax does not prescribe whether delegation history is stored explicitly or compressed into a derived proof object. Let $\mathcal{A}$ denote the universe of attributes, and let $\preceq$ denote an application-defined authorization preorder on $\mathcal{A}$; set inclusion is a canonical special case. All verification is relative to an authenticated root anchor RootAnchor. The RootAnchor binds the root verification key $\mathsf{ipk}_{\mathsf{root}}$ (together with the public parameters); verifiers trust RootAnchor as the root of authority. A DAC scheme Π_{DAC} consists of the following tuple of polynomial-time algorithms:

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$: On input of the security parameter λ, this algorithm outputs the public parameters pp. We assume pp is an implicit input to all subsequent algorithms.
- $\mathsf{KeyGen}(\mathsf{pp}) \to (\mathsf{sk}, \mathsf{pk})$: A randomized algorithm that generates a secret key sk and a corresponding public key pk for a user or issuer. For the root authority, we write its key pair as $(\mathsf{isk}_{\mathsf{root}}, \mathsf{ipk}_{\mathsf{root}}) \leftarrow \mathsf{KeyGen}(pp)$
- $\mathsf{Issue}(\mathsf{isk}_{\mathsf{root}}, \mathsf{RootAnchor}, \mathsf{pk}_{\mathsf{user}}, A_{\mathsf{root}}) \to \mathsf{Cred}_0$: The root issuance algorithm. It takes as input the root's secret key, the authenticated root anchor, the target user's public key, and an initial attribute set A_{root}. It outputs a level-0 credential Cred_0.
- $\mathsf{Delegate}(\mathsf{sk}_{\mathsf{parent}}, \mathsf{Cred}_{\mathsf{parent}}, \mathsf{pk}_{\mathsf{child}}, A_{\mathsf{child}}) \to \mathsf{Cred}_{\mathsf{child}} \cup \{\bot\}$: The delegation algorithm. It takes as input a delegator's secret key, a credential $\mathsf{Cred}_{\mathsf{parent}}$ authorizing an attribute set A_{parent}, a delegatee's public key, and a delegated attribute set A_{child}. It enforces monotonicity: if $A_{\mathsf{child}} \preceq A_{\mathsf{parent}}$, it outputs a new credential $\mathsf{Cred}_{\mathsf{child}}$; otherwise, it outputs $\bot$.
- $\mathsf{Prove}(\mathsf{sk}_{\mathsf{user}}, \mathsf{Cred}, m, A_{\mathsf{pre}}) \to \pi$: The presentation algorithm. It takes as input a user's secret key, a valid credential Cred, a context message m, and a set of attributes A_{pre} to disclose. It outputs a zero-knowledge proof π.
- $\mathsf{Verify}(\mathsf{RootAnchor}, m, A_{\mathsf{pre}}, \pi) \to \{0, 1\}$: The deterministic verification algorithm. It validates the proof π against the authenticated root anchor RootAnchor, the message m, and the disclosed attributes A_{pre}. It outputs 1 for valid proofs and 0 otherwise. For simplicity, we write the public statement as $(\mathsf{RootAnchor}, m, A_{\mathsf{pre}})$; application-specific context can be folded into m or the disclosed statement without affecting the definitions below.

3.2. Security Definitions

We separate authorization integrity from the two privacy properties of Enhanced Delegation Anonymity (EDA). The integrity game adapts the authorized-lineage and attribute-soundness formulations of Belenkiy et al. [8, Sec. 3.1], Blömer and Bobolz [27, Defs. D.1, D.3], and Mir et al. [12, App. D.3].

Definition 1 (Correctness)**.** *For any honestly generated, restriction-preserving credential of authorization A under* RootAnchor*, a presentation for an eligible request (ctx, A_{pre}) with $A_{\mathsf{pre}} \preceq A$ verifies except with negligible probability. In a bounded signature implementation, the signers must have available signing capacity.*

Definition 2 (Delegation Unforgeability)**.** *The challenger sets up one authenticated root* RootAnchor*, gives the adversary the public parameters, and retains the root signing key. Honest signing keys in this experiment are dedicated to the instance. The adversary adaptively accesses the following oracles. A handle h names a recorded credential, u names a registered holder, and invalid requests return $\bot$.*

- ***New.*** *Generate an honest holder u and return its public key, excluding already registered keys and the root key. The adversary can also register its own public keys subject to the same uniqueness requirement.*
- ***Issue and Delegate.*** *Issue eligible rights to a recipient, or delegate from a credential handle of an uncorrupted parent to a recipient, subject to attribute narrowing. Record the exact signed message and its transition data, and return a new handle. Give the token to the adversary if the recipient is adversary-controlled.*
- ***Corrupt.*** *Release a non-root holder's secret, signer state and recorded credentials, and classify it as adversary-controlled thereafter.*
- ***Show.*** *For a valid request to an uncorrupted holder's credential handle, return a presentation and log the complete canonical verifier statement x in $\mathcal{L}_{\mathsf{show}}$.*

Let $\mathsf{AuthCl}_{\mathcal{A}}$ be the downward closure under $\preceq$ of rights in credentials delivered to adversary-controlled recipients or exposed through corruption. The adversary wins if it outputs an accepting presentation under the fixed root for disclosure A^* with $x^* \notin \mathcal{L}_{\mathsf{show}}$ and $A^* \notin \mathsf{AuthCl}_{\mathcal{A}}$. Freshness excludes replays with different proof bytes. The verifier matches the public statement to the expected request (which can include a fresh nonce). The winning probability must be negligible.

To formalize privacy notations, let $\rho = \mathsf{RootAnchor}$ be fixed and let ctx encode the public presentation context, including any policy. The challenger gives the adversary honestly generated public parameters. In each game it samples a uniform bit b after the adversary has chosen the candidates and obtained its auxiliary view u, which is shared between both worlds. Security means negligible advantage $|\Pr[b' = b] - 1/2|$ for each PPT adversary's guess b'.

Definition 3 (Verifier-Side Anonymity)**.** *The adversary supplies two valid paths under ρ and their presentation inputs (or references to challenger-held inputs), both authorizing the same public statement stmt_V. In PCC this is $(\rho, z_{\mathsf{root}}, z_{\mathsf{pre}}, \tau_{\mathsf{pre}})$, determined by $(\rho, ctx, A_{\mathsf{pre}})$. Any intentional application disclosures must also match; PCC itself outputs neither a holder key nor a pseudonym. The challenger honestly produces a fresh presentation from path b and gives its full serialized public transcript to the adversary, who guesses b. The auxiliary view u can include candidate histories, credentials, and authorization keys, but not the challenge prover's fresh coins.*

This game adapts left–right DAC anonymity (e.g., Mir et al. [12, App. D.3, Fig. 6]) to standalone presentation; it does not model anonymity of the delegation interaction. Application disclosures can themselves identify or link presentations.

Definition 4 (Holder-Side Topology Obliviousness)**.** *The adversary supplies two valid upstream histories, possibly*

adversarially prepared, each extendable by an honest final handoff to the same holder H. The candidates must agree on the allowed leakage

$$\ell_H = (\rho, \ell_{\mathsf{loc}}, \ell_{\mathsf{pre}}, \ell_{\mathsf{edge}}, \ell_{\mathsf{sys}}).$$

Here ℓ_{loc} contains H's own keys and terminal attributes; ℓ_{pre}, common presentation inputs; ℓ_{edge}, exposed immediate-edge information; and ℓ_{sys}, public parameters and any fixed public depth bound. It does not directly include the actual depth or remote lineage. The challenger runs the final handoff for branch b with fresh randomness and honestly runs the presentation. The adversary, observing as H, receives the full view View_H: the credential, local proving inputs, visible coins and API metadata, and the presentation. It then guesses b. Its auxiliary view u can include both upstream histories and their secrets, but not the handoff prover's private coins for this challenge or messages revealing the selection outside the returned view. Physical side channels are out of scope.

In PCC the view includes $\mathsf{PCCred}_H = (\rho, \mathsf{Token}_H)$ with $\mathsf{Token}_H = (z_H, \pi_H, \mathsf{upk}_H, A_H, r_H)$, and uses the presentation witness $w_{\mathsf{pre}} = (\mathsf{usk}_H, ctx, A_{\mathsf{pre}}, A_H, r_H)$. The game tests the full view, including proof serialization and parser metadata. A simulator of the joint view from the common leakage and auxiliary view suffices for this game.

Enhanced Delegation Anonymity A DAC satisfies EDA if it meets both definitions. Hiding identities does not imply hiding topology: Belenkiy et al. [8, Sec. 1] hide upstream identities but carry a chain whose size grows with depth. Our holder-side guarantee does not hide disclosures in ℓ_H or prevent an informed sender from deliberately revealing the selected route.

4. Proof-Carrying Credentials

We introduce *Proof-Carrying Credentials* (PCC), a recursive DAC structure that compiles delegation validity into a proof-carrying state. Section 4.1 reviews a representative signature-and-NIZK baseline whose explicit history makes both the validation data and the exposed lineage grow with the delegation depth; Section 4.2 replaces this history with a single recursively updated state built over Non-Uniform IVC (NIVC) [28], giving a constant-size verifier-facing object for paths of arbitrary depth [29] that hides remote delegation topology. The PCC wrapper inherits the completeness, simulation-extractability, and zero-knowledge of the underlying NIVC, yielding correctness, delegation unforgeability, and Enhanced Delegation Anonymity at the DAC layer.

4.1. A Signature-and-NIZK Baseline

We first recall the explicit-history pattern underlying representative signature-and-NIZK DAC constructions, including P-UC-DAC [11]. In such a construction, a level-L holder receives a credential containing the entire delegation path:

$$\mathsf{Cred}_L = \left(\{(\sigma_i, \mathsf{upk}_i, A_i)\}_{i=1}^{L}, \mathsf{usk}_L\right), \tag{1}$$

where each σ_i authorizes the transition from upk_{i-1} to upk_i and binds the delegated attribute set A_i.

To present the credential, the holder proves in zero knowledge that the stored path is valid. Setting $\mathsf{upk}_0 = \mathsf{ipk}_{\mathsf{root}}$, the presentation relation contains the following depth-dependent checks:

$$\Gamma_i := \begin{bmatrix} \Sigma.\mathsf{Verify}(\mathsf{upk}_{i-1}, (\mathsf{upk}_i, A_i), \sigma_i) = 1 \\ \wedge\, A_i \preceq A_{i-1} \end{bmatrix},$$

$$\mathsf{R}_{\mathsf{hist}} = \left\{ (x, w) : \begin{array}{l} \bigwedge_{i=1}^{L} \Gamma_i \wedge \mathsf{upk}_L = \mathsf{PK}(\mathsf{usk}_L) \\ \qquad \wedge\, A_{\mathsf{pre}} \preceq A_L \end{array} \right\}.$$

Here the public statement includes the issuer key and the disclosed presentation statement, while the witness contains the path, the holder's secret key, and the attribute sequence.

This construction hides the delegation path from the verifier through zero knowledge, but the holder must still store and process the full chain $\{(\sigma_i, \mathsf{upk}_i, A_i)\}_{i=1}^{L}$ to present, so its local view stays explicitly history-dependent and retains cryptographically verifiable lineage. PCC eliminates this explicit history by replacing the path with a recursively updated proof-carrying state.

4.2. PCC Construction

Our proof-carrying credential construction leverages Non-Uniform Incrementally Verifiable Computation (NIVC), recalled in Definition 6, to realize a non-interactive recursive delegation model. The non-uniform feature is necessary because the recursive computation must support heterogeneous transition types: whereas standard IVC repeatedly applies a single transition function, our construction must switch among issuance, delegation, and presentation predicates. By running NIVC, the validity of the delegation path is accumulated into a recursively updated proof state carried by the credential. This enables a holder to present the credential together with a proof that the delegated authorization is valid with respect to a lineage rooted at the issuer.

We define three state-transition predicates, $\Phi = \{\phi_{\mathsf{issue}}, \phi_{\mathsf{del}}, \phi_{\mathsf{pre}}\}$, for PCC. These predicates encode the core security policies and validity constraints enforced by the respective step circuits at each transition. Before defining the predicates, we formalize the state and witness structures.

- **State (z_i):** The state at step i is a commitment $z_i = \mathsf{Com}(\mathsf{upk}_i, A_i; r_i)$ to the current delegatee's verification key upk_i and authorized attribute set A_i. The public initial state is derived from the authenticated anchor as $z_{\mathsf{root}} := H_{\mathsf{state}}(\mathsf{tag}_{\mathsf{issue}}, \mathsf{RootAnchor})$.
- **Witness (w_{i+1}):** Write $o_i = (\mathsf{upk}_i, A_i, r_i)$ for the opening of z_i. The witness contains the private inputs for the transition:

$$w = \begin{cases} (\sigma_{\mathsf{root}}, o_0, \mathsf{meta}_{\mathsf{iss}}) & \text{issue}, \\ (\sigma_i, o_{i+1}, o_i, \mathsf{meta}_{\mathsf{del}}) & \text{del}, \\ (\mathsf{usk}_i, ctx, A_{\mathsf{pre}}, A_i, r_i) & \text{pre}. \end{cases}$$

The Issuance Predicate The predicate ϕ_{issue} initializes the recursive lineage. The issuer verification key is resolved from the authenticated anchor and bound to the fixed instance ρ. It checks

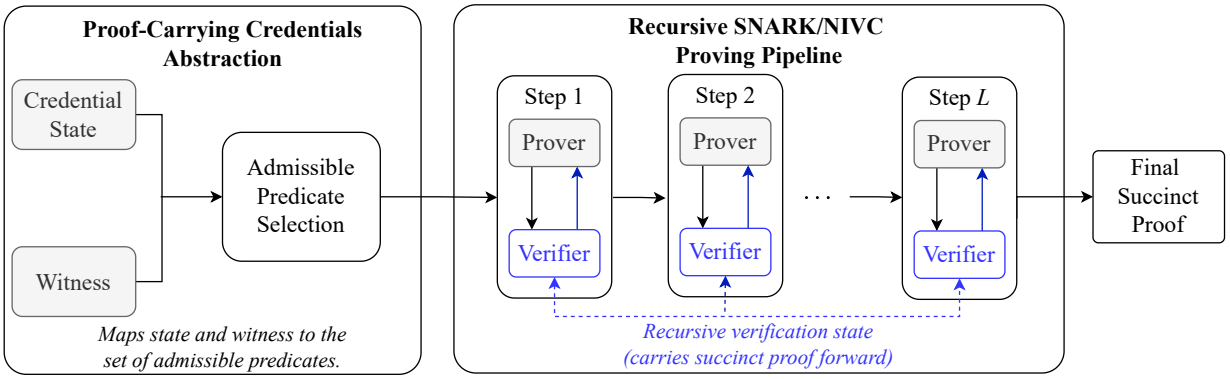


Figure 3. PCC workflow based on NIVC.

that $z_{\mathsf{root}} = H_{\mathsf{state}}(\mathsf{tag}_{\mathsf{issue}}, \mathsf{RootAnchor})$, that $z_0 = \mathsf{Com}(\mathsf{upk}_0, A_0; r_0)$, and that

$$\Sigma.\mathsf{Verify}(\mathsf{ipk}_{\mathsf{root}}, H_{\mathsf{sig}}(\mathsf{tag}_{\mathsf{issue}}, z_0, A_0, \mathsf{meta}_{\mathsf{iss}}), \sigma_{\mathsf{root}}) = 1.$$

The Delegation Predicate The predicate ϕ_{del} governs the extension of the delegation path (function selector $j = \mathsf{del}$). It takes as public inputs the current state commitment z_i and the next state commitment z_{i+1}, along with the witness w_{i+1}. The step circuit accepts the transition if and only if $\phi_{\mathsf{del}}(\rho, z_i, z_{i+1}, w_{i+1}) = 1$. The constraints are defined as follows:

- *Consistency:* The witness w_{i+1} must correctly open the commitments z_i and z_{i+1} to reveal the underlying states (upk_i, A_i) and $(\mathsf{upk}_{i+1}, A_{i+1})$.
- *Attribute Hierarchy Policy:* It enforces the monotonicity of authority by verifying that the delegated attributes are a valid subset of the delegator's attributes: $A_{i+1} \subseteq A_i$.
- *Transition Authorization:* It verifies that the transition from z_i to z_{i+1} is cryptographically authorized by the owner of the current state. This is enforced by checking that σ_i (included in w_{i+1}) is a valid signature on the domain-separated delegation message M_i^{del} under upk_i:

$$\begin{aligned} M_i^{\mathsf{del}} &:= H_{\mathsf{sig}}\big(\mathsf{tag}_{\mathsf{del}}, z_i, z_{i+1}, A_{i+1}, \mathsf{meta}_{\mathsf{del}}\big), \\ &\Sigma.\mathsf{Verify}(\mathsf{upk}_i, M_i^{\mathsf{del}}, \sigma_i) = 1. \end{aligned} \tag{2}$$

The Presentation Predicate For the terminal presentation step, we write $i = L$ and use $(\mathsf{upk}_L, A_L, \mathsf{usk}_L)$ to denote the terminal holder state. The predicate ϕ_{pre}, corresponding to selector $j = \mathsf{pre}$, finalizes a credential into a context-bound presentation. It takes the terminal credential-state commitment z_L as the current state and a public presentation state z_{pre} as the next state. The function outputs 1 if and only if:

- **Credential-state consistency.** The witness opens the terminal credential state z_L to $(\mathsf{upk}_L, A_L, r_L)$.
- **Presentation binding.** The output state is the deterministic public binding $z_{\mathsf{pre}} = H_{\mathsf{pre}}(\mathsf{pre}, ctx, A_{\mathsf{pre}})$, where ctx is the presentation context and A_{pre} is the disclosed attribute set.
- **Ownership proof.** The circuit checks $\mathsf{KeyRel}(\mathsf{upk}_L, \mathsf{usk}_L) = 1$ (Definition 8).
- **Selective disclosure logic.** The disclosed attributes are authorized by the terminal credential: $A_{\mathsf{pre}} \preceq A_L$.

Based on the NIVC primitive and the predicates defined above, we now formally construct the Proof-Carrying Credentials scheme Π_{PCC}.

Remark 1. *The integrity of the historical path leading to z_i (lineage validation) is guaranteed by the underlying NIVC recursive verification mechanism and is implicitly checked before executing this step.*

Definition 5 (Proof-Carrying Credentials Scheme)**.** *The Proof-Carrying Credentials scheme is a tuple of polynomial-time algorithms*

$$\Pi_{\mathsf{PCC}} = (\mathsf{PCC.Setup}, \mathsf{PCC.KeyGen}, \mathsf{PCC.Derive}, \mathsf{PCC.Verify})$$

instantiated over the NIVC primitive (Definition 6) and the predicate set $\Phi = \{\phi_{\mathsf{issue}}, \phi_{\mathsf{del}}, \phi_{\mathsf{pre}}\}$:

- $\mathsf{PCC.Setup}(1^\lambda) \to \mathsf{pp}$*: The generator runs the NIVC setup* $\mathsf{NIVC.Setup}(1^\lambda)$ *to produce public parameters* pp.
- $\mathsf{PCC.KeyGen}(\mathsf{pp}) \to (\mathsf{pk}, \mathsf{vk})$*: The indexer takes the parameters and the specific logic description Φ to generate the proving and verification keys:* $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{NIVC.KeyGen}(\mathsf{pp}, \Phi)$
- ***Auxiliary subroutine*** $\mathsf{PCC.Derive}^*$*:* *Given* $(\mathsf{pk}, j, \rho, z_{\mathsf{root}}, z_i, z_{i+1}, w_{i+1}, \pi_i)$*, this internal subroutine updates the recursive proof from state z_i to state z_{i+1} under the selected predicate $j \in \{\mathsf{issue}, \mathsf{del}, \mathsf{pre}\}$. It invokes the underlying NIVC prover as*

$$\pi_{i+1} \leftarrow \mathsf{NIVC.Prove}(\mathsf{pk}, j, \rho, z_{\mathsf{root}}, z_i, z_{i+1}, w_{i+1}, \pi_i).$$

It returns the updated recursive proof π_{i+1}.
In delegation mode, $j = \mathsf{del}$ and z_{i+1} is the next credential-state commitment. In presentation mode, $j = \mathsf{pre}$ and $z_{i+1} = z_{\mathsf{pre}}$ is the public presentation-binding state.
- ***Wrapper Syntax.*** *To streamline the DAC lifecycle, we introduce the high-level wrapper* PCC.Derive*, which encapsulates witness construction and automates predicate selection. Here and below,* Token_i *denotes the credential token carried at step i.*
- $\mathsf{PCC.Derive}(\rho, z_{\mathsf{root}}, \mathsf{Token}_i, \mathsf{args}; \mathsf{sk}) \to \mathsf{Token}'$ *:*
The wrapper selects the transition predicate according to args*. In issuance mode $\mathsf{Token}_i = \perp$; otherwise it parses $\mathsf{Token}_i = (z_i, \pi_i, \mathsf{upk}_i, A_i, r_i)$.*
  - ***Issuance mode.*** *If* args *encode* $(\mathsf{upk}_0, A_0, \mathsf{meta}_{\mathsf{iss}})$*, sample r_0, set $z_0 := \mathsf{Com}(\mathsf{upk}_0, A_0; r_0)$, and compute*

$$\begin{aligned} M_{\mathsf{iss}} &:= H_{\mathsf{sig}}(\mathsf{tag}_{\mathsf{issue}}, z_0, A_0, \mathsf{meta}_{\mathsf{iss}}), \\ \sigma_{\mathsf{root}} &\leftarrow \Sigma.\mathsf{Sign}(\mathsf{isk}_{\mathsf{root}}, M_{\mathsf{iss}}), \\ w_0 &:= (\sigma_{\mathsf{root}}, \mathsf{upk}_0, A_0, r_0, \mathsf{meta}_{\mathsf{iss}}). \end{aligned}$$

Then compute

$$\pi_0 \leftarrow \mathsf{PCC.Derive}^*(\mathsf{pk}, \mathsf{issue}, \rho, z_{\mathsf{root}}, z_{\mathsf{root}}, z_0, w_0, \bot),$$

and output

$$\mathsf{Token}_0 := (z_0, \pi_0, \mathsf{upk}_0, A_0, r_0).$$

– ***Delegation mode.*** *If* args *encode delegation parameters* $(\mathsf{upk}_{i+1}, A_{i+1}, \mathsf{meta}_{\mathsf{del}})$*, set* $j = \mathsf{del}$*, sample* r_{i+1}*, and compute* $z_{i+1} := \mathsf{Com}(\mathsf{upk}_{i+1}, A_{i+1}; r_{i+1})$*. Define the delegation message and signature as*

$$M_i^{\mathsf{del}} := H_{\mathsf{sig}}(\mathsf{tag}_{\mathsf{del}}, z_i, z_{i+1}, A_{i+1}, \mathsf{meta}_{\mathsf{del}}),$$
$$\sigma_i \leftarrow \Sigma.\mathsf{Sign}(\mathsf{usk}_i, M_i^{\mathsf{del}}).$$

Construct the delegation witness

$$w_{i+1} := (\sigma_i, o_{i+1}, o_i, \mathsf{meta}_{\mathsf{del}}),$$

and compute

$$\pi_{i+1} \leftarrow \mathsf{PCC.Derive}^*(\mathsf{pk}, \mathsf{del}, \rho, z_{\mathsf{root}}, z_i, z_{i+1}, w_{i+1}, \pi_i).$$

Output $\mathsf{Token}_{i+1} := (z_{i+1}, \pi_{i+1}, \mathsf{upk}_{i+1}, A_{i+1}, r_{i+1})$.

– ***Presentation mode.*** *If* args *encode presentation parameters* (ctx, A_{pre})*, set* $j = \mathsf{pre}$ *and compute*

$$z_{\mathsf{pre}} := H_{\mathsf{pre}}(\mathsf{pre}, ctx, A_{\mathsf{pre}}).$$

Construct the presentation witness

$$w_{i+1} := (\mathsf{usk}_i, ctx, A_{\mathsf{pre}}, A_i, r_i),$$

and compute

$$\pi_{i+1} \leftarrow \mathsf{PCC.Derive}^*(\mathsf{pk}, \mathsf{pre}, \rho, z_{\mathsf{root}}, z_i, z_{\mathsf{pre}}, w_{i+1}, \pi_i).$$

Output the final presentation token

$$\mathsf{Token}_{\mathsf{pre}} := (z_{\mathsf{pre}}, \pi_{i+1}, ctx, A_{\mathsf{pre}}).$$

- ***Verification.*** *Given* $(\mathsf{vk}, \rho, \mathsf{Token}_{\mathsf{pre}})$*, parse*

$$\mathsf{Token}_{\mathsf{pre}} = (z_{\mathsf{pre}}, \pi_{\mathsf{pre}}, ctx, A_{\mathsf{pre}}).$$

Check ctx *and* A_{pre} *against the expected request and set* $z_{\mathsf{root}} := H_{\mathsf{state}}(\mathsf{tag}_{\mathsf{issue}}, \rho)$*. Recompute the expected presentation state* $\widehat{z}_{\mathsf{pre}} := H_{\mathsf{pre}}(\mathsf{pre}, ctx, A_{\mathsf{pre}})$ *and rejects if* $\widehat{z}_{\mathsf{pre}} \neq z_{\mathsf{pre}}$*. It then forms the public terminal condition* $\tau_{\mathsf{pub}} := (\mathsf{tag} = \mathsf{pre}, ctx, A_{\mathsf{pre}})$ *and checks*

$$b \leftarrow \mathsf{NIVC.Verify}(\mathsf{vk}, \rho, z_{\mathsf{root}}, z_{\mathsf{pre}}, \pi_{\mathsf{pre}}, \tau_{\mathsf{pub}}).$$

The verifier accepts iff $b = 1$*. The verifier is not given the delegation depth, intermediate credentials, intermediate states, or the non-terminal selector sequence.*

We require Π_{PCC} to satisfy completeness, simulation extractability, and zero-knowledge. For delegation steps, we write the concrete credential at step i as $\mathsf{PCCred}_i := (\rho, \mathsf{Token}_i)$, where $\mathsf{Token}_i = (z_i, \pi_i, \mathsf{upk}_i, A_i, r_i)$. For presentation, the final token exposes the public projection (z_{pre}, π) together with the presentation context encoded in args.

4.3. Security Analysis

The representation by itself does not imply privacy or integrity. The recursive verifier must enforce the execution language issue del* for a carried credential and issue del* pre for a showing. In particular, the base proof is accepted only by the issuance branch, every successor extends an accepting credential under the same root, and a presentation cannot be used as a predecessor. Appendix A.1 specifies the proof interface. We assume canonical input encodings, binding state commitments, and collision resistance for the signed and presentation transcripts.

Theorem 1 (Conditional Security of PCC)**.** *Let the backend satisfy completeness, adaptive auxiliary-input zero knowledge and fresh-statement trace extraction after simulated showings (Definitions 6 and 7). Let the authorization signature be EUF-CMA secure, and let the ownership relation satisfy Definition 8. Under the binding conditions above, the induced DAC is correct and delegation-unforgeable. Auxiliary-input zero knowledge of the presentation proof yields verifier-side anonymity for a fixed public statement. With credential-proof zero knowledge and a canonical holder representation apart from the credential proof bytes determined by the allowed leakage, PCC also satisfies holder-side topology obliviousness at an honest final handoff.*

Proof overview. Completeness follows by induction over authorized transitions. In the integrity game, replace honest showings by simulations, then extract from a fresh accepting output. Match the trace from the root against the signing ledger. A first unmatched transition under an honest signer yields a signature forgery or a binding violation. Reaching an adversarial holder through recorded edges puts the final disclosure in the authorized closure. Otherwise the extracted terminal secret for a recorded honest holder enables a fresh signature under that holder's key. For privacy, the verifier proof is simulated from its public statement. The holder proof is simulated from its fixed credential statement; the showing is then generated using that proof and the available terminal witness. The reductions appear in Appendix B.2 and Appendix B.3. □

5. Post-Quantum Instantiation of PCC

In this section, we present a concrete post-quantum instantiation of PCC. The generic PCC framework is structurally hardness-agnostic: it yields post-quantum security provided that the underlying signature scheme and recursive proof system remain secure against quantum adversaries. Our goal here is therefore not to redefine PCC, but to instantiate it with primitives that simultaneously support post-quantum security and efficient recursive proving.

To maximize circuit efficiency in the concrete realization, we prioritize primitives that minimize arithmetic complexity inside the recursive proof system. Our primary instantiation combines Plonky2-style FRI-based recursion, Poseidon hashing, and a circuit-oriented hash-based authorization scheme. This preserves the generic DAC interface of Section 4. The 100- and 128-bit settings reported in the original evaluation are nominal parameter targets, not security levels established by this analysis.

5.1. Cryptographic Primitives and Parameters

Our post-quantum-compatible instantiation combines transparent recursive proving with circuit-friendly authorization checks: Goldilocks arithmetic ($p = 2^{64} - 2^{32} + 1$, FRI over $\mathbb{F}_p^2$) enables efficient Plonky2-style FRI recursion, Poseidon provides arithmetization-friendly hashing, and a Poseidon-based, SPHINCS$^+$-style [30] hash-based signature authorizes issuance and delegation without pairing- or discrete-log assumptions. We retain Fractal-style transparent recursion as a comparison point in Section 6. The full primitive stack, parameters, and security rationale (Table 4) are deferred to Appendix A.2, under whose explicit QROM assumptions the instantiation is post-quantum-compatible.

5.2. Protocol Design of PQDAC

We instantiate a concrete post-quantum DAC, Π_{PQDAC}, as the generic Π_{PCC} of Definition 5 run over the primitive stack of Table 4: its algorithms are exactly $\mathsf{PCC.Setup}/\mathsf{PCC.KeyGen}/\mathsf{PCC.Derive}/\mathsf{PCC.Verify}$ with Σ and the NIVC instantiated post-quantumly, and user keys are signature keypairs ($\mathsf{usk} := \mathsf{sk}_{\mathsf{sig}}$, $\mathsf{upk} := \mathsf{pk}_{\mathsf{sig}}$). The full algorithmic specification is given in Figure 7. The protocol is parameterized by $\Phi = \{\phi_{\mathsf{issue}}, \phi_{\mathsf{del}}, \phi_{\mathsf{pre}}\}$, where ϕ_{issue} authenticates root issuance, ϕ_{del} governs recursive delegation, and ϕ_{pre} governs context-bound presentation. A credential evolves through three phases:

- **Setup and issuance.** Setup compiles Φ into $(\mathsf{pk}, \mathsf{vk})$; the root authority invokes $\mathsf{PCC.Derive}$ in issuance mode under $\mathsf{RootAnchor}$, producing $\mathsf{PCCred}_0 = (\rho, \mathsf{Token}_0)$ with $\rho := \mathsf{RootAnchor}$.
- **Recursive delegation.** $\mathsf{Delegate}$ invokes $\mathsf{PCC.Derive}$ in delegation mode on $(\mathsf{PCCred}_i, \mathsf{upk}_{i+1}, A_{i+1}, \mathsf{meta}_{\mathsf{del}}, \mathsf{usk}_i)$, yielding Token_{i+1} with a constant-size verifier-facing proof component.
- **Presentation.** $\mathsf{Present}$ invokes $\mathsf{PCC.Derive}$ in presentation mode for $\mathsf{ctx} := (m, \mathsf{Policy})$ and A_{pre}; Verify checks $\mathsf{Token}_{\mathsf{pre}}$ against $\mathsf{RootAnchor}$, from which z_{root} is derived.

The realization inherits security only if it implements the interfaces in Theorem 1. In particular, security of isolated FRI/Fiat–Shamir proofs does not establish extraction of a full recursive trace after simulated showings.

Theorem 2 (Conditional Delegation Unforgeability)**.** *If the realization satisfies the integrity premises of Theorem 1, it is delegation-unforgeable (Definition 2).*

Theorem 3 (Conditional Enhanced Delegation Anonymity)**.** *If the realization satisfies the privacy premises of Theorem 1, it satisfies EDA.*

Theorem 4 (Conditional Post-Quantum Security)**.** *The same conclusions hold against quantum polynomial-time adversaries with classical protocol queries when all interfaces of Theorem 1 hold in the same quantum security model. Appendix B.1 specifies the assumptions and their scope.*

6. Evaluation

6.1. Experimental Setup

Environment and metrics. All benchmarks were executed on a desktop workstation with an AMD Ryzen 7 9800X3D processor, 32 GB of RAM, and a 1 TB SSD, running Ubuntu 22.04 LTS. Unless otherwise stated, measurements were collected in single-threaded mode over 10 independent trials, and we report the mean. We evaluate three metrics: final verifier-facing proof size, final verification latency, and delegation time per recursive step. For recursive DACs, the per-step delegation time is the marginal online cost of extending an existing credential by one hop, rather than the cumulative cost of reconstructing the full delegation chain. All proof sizes reported in this section refer to the final object delivered to the verifier.

Workloads. We evaluate delegation depths up to $L = 16$, which is conservative relative to typical transitive-authorization deployments: DeFi credit delegation often uses small hop counts, while SSI, enterprise IAM, and service-mediated authorization chains rarely exceed single-digit or low double-digit depth. For the Halo2 experiments, we use a workload with 20 attributes and evaluate both selective disclosure and richer validity predicates. For the post-quantum experiments, we report results at 100-bit and 128-bit security. Unless otherwise stated, compared systems use matched delegation depth, attribute-set size, and presentation-policy complexity.

Baselines. For the non-post-quantum evaluation, we compare against **P-UC-DAC** [11], the closest prior practical proof-based DAC baseline. To isolate the role of recursion, we also compare **Halo2 Monolithic** with **Halo2 Recursive** under the same backend and matched workload: the former encodes accumulated lineage-validation checks directly in the final presentation circuit, whereas the latter carries lineage validity recursively through the PCC state and exports a single verifier-facing proof object. For the post-quantum setting, we compare a non-recursive Fractal-style baseline, a recursive Fractal baseline, and our Plonky2-based recursive PCC. We do not use EQS-DAC as a primary experimental baseline because it belongs to a different design line that avoids general proof-system machinery; we retain it as a qualitative comparison point in Table 1.

Evaluation goals. This section answers four questions:

- **RQ1:** Compared with the closest prior practical proof-based DAC baseline, does PCC remove depth-dependent verifier-facing presentation growth while incurring only a moderate constant recursive overhead?
- **RQ2:** On the Halo2 backend, does the PCC turn recursive lineage carrying into a constant-size verifier-facing proof interface?
- **RQ3:** Does the post-quantum instantiation remain practical at deployment-realistic delegation depths?
- **RQ4:** Under matched FRI configurations, how do PCC's presentation latency and peak resident memory vary with delegation depth compared with exact-depth and fixed-bound explicit-history baselines?

6.2. Performance of the Non-Post-Quantum Instantiation (Halo2-based PCC)

We first evaluate the non-post-quantum instantiation of PCC built over Halo2 [31]. This part of the evaluation serves two purposes. First, it compares PCC against **P-UC-DAC**, the closest prior practical proof-based DAC baseline. Second, it contrasts two Halo2 realizations: **Halo2 Monolithic**, where the presentation circuit explicitly grows with the delegation lineage, and **Halo2 Recursive**, where lineage validity is carried through the PCC state and the exported verifier-facing presentation object is independent of the tested delegation depth. The key question is whether the recursive PCC architecture removes the depth-dependent verifier-facing cost observed in the monolithic baseline.

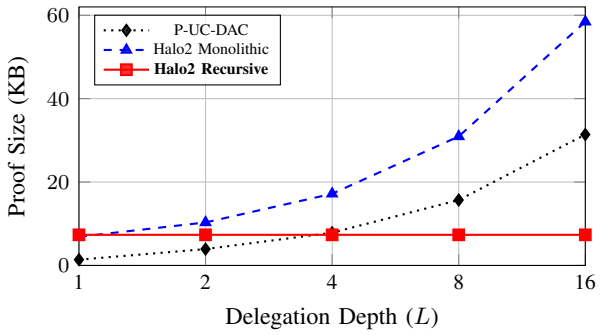


Figure 4. Proof size vs. delegation depth. The monolithic Halo2 baseline exposes depth-dependent verifier-facing proof material, whereas Recursive Halo2 carries lineage validity through the PCC state and exports a constant-size presentation object across the tested depths.

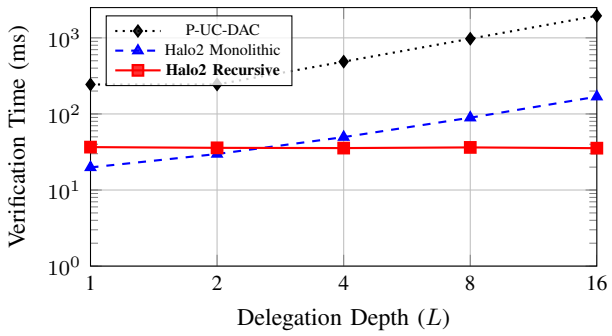


Figure 5. Verification time vs. delegation depth. Monolithic Halo2 incurs depth-dependent verification cost because lineage-validation material grows with the delegation chain. Recursive Halo2 exports a depth-independent verifier-facing presentation object, yielding approximately constant verification time across tested depths.

6.2.1. Comparison. We begin with an external comparison against P-UC-DAC [11]. This baseline is the most relevant prior practical point of comparison for our non-post-quantum setting because, like PCC, it belongs to the proof-based DAC line rather than to the more recent signature-only constructions that explicitly avoid general zero-knowledge machinery.

Figure 4 and Figure 5 show the verifier-facing scaling behavior of the non-PQ instantiation. P-UC-DAC and monolithic Halo2 expose depth-dependent presentation material, causing proof size and verification latency to grow with L. Recursive Halo2 instead carries lineage validity through the PCC state and exports a constant-size verifier-facing proof: 7.344 KB and approximately 35–36 ms verification across all tested depths. Table III shows that the marginal online delegation cost is also stable across depths, confirming that each recursive update depends only on the previous proof and the current transition witness.

Answer to RQ1. PCC removes the depth-dependent verifier-facing growth of the closest proof-based baseline and additionally hides the holder-side topology, trading a moderate constant recursive overhead for a proof interface whose size and verification cost are independent of the delegation depth.

6.2.2. Delegation Cost. To complement the verifier-facing measurements, we also report the online cost of a single delegation step. Table 2 summarizes the delegation time per step together with the final verification cost and the final proof size.

TABLE 2. ONLINE DELEGATION-STEP COST AND FINAL VERIFIER-FACING PROOF METRICS FOR HALO2-BASED PCC.

Depth	Delegation (s)	Verify (ms)	Proof (KB)
1	4.697	36.655	7.344
2	4.368	35.855	7.344
4	4.664	35.600	7.344
8	4.785	36.319	7.344
16	4.752	35.499	7.344

The *delegation time per step* remains stable across all tested depths, staying within a narrow range of 4.368–4.785 s. Likewise, the final verification cost varies only within 35.499–36.655 ms across the tested depths, consistent with the intended depth-independent verifier-facing interface. This confirms that the online marginal cost of extending the credential by one additional delegation step does not increase with prior delegation depth. Together with the constant 7.344 KB verifier-facing proof and the approximately constant 35–36 ms verification cost, this shows that Halo2-based PCC supports incremental credential evolution without exposing a depth-dependent proof interface to the verifier.

We further verify that this depth-independence is not an artifact of a simple disclosure policy: across simple ($\mathcal{L}_{\mathsf{sig}}$), range ($\mathcal{L}_{\mathsf{range}}$), and membership ($\mathcal{L}_{\mathsf{memb}}$) presentation predicates implemented inside ϕ_{pre}, PCC keeps the final verifier-facing proof at 7.344 KB, so richer presentation logic changes circuit content without reintroducing depth-dependent growth (Table 5, Appendix A.4).

Answer to RQ2. Under matched workloads, recursive Halo2 keeps a constant 7.344 KB proof and 35 -36 ms verification across all tested depths (per-step delegation stable at 4.368–4.785 s), whereas the monolithic variant grows with L; the depth-independence is therefore a property of the PCC architecture, not a Halo2 artifact.

6.3. Performance of the Post-Quantum Instantiation (Plonky2-based PCC)

We next evaluate the scalability of our post-quantum (PQ) instantiation. This part of the evaluation is archi-

TABLE 3. **PQ ARCHITECTURAL COMPARISON AT DEPLOYMENT-REALISTIC DELEGATION DEPTHS.**

Sec	Depth	Circuit Complexity			Prover Memory (MB)			Plonky2 Metrics		
		Base	Frac.	Plonky2	Base	Frac.	Plonky2	Prove/Step (s)	Verify (s)	Final Proof (KB)
100	1	8,960†	48,960†	4,096♣	1.71	9.34	8.00	0.600	0.0029	125.2
	2	17,920†	48,960†	8,192♣	3.42	9.34	16.00	0.600	0.0029	125.2
	4	35,840†	48,960†	8,192♣	6.84	9.34	16.00	0.600	0.0029	125.2
	8	71,680†	48,960†	8,192♣	13.67	9.34	16.00	0.600	0.0029	125.2
	16	143,360†	48,960†	8,192♣	27.34	9.34	16.00	0.600	0.0029	125.2
128	1	8,960†	48,960†	4,096♣	1.71	9.34	8.00	1.030	0.0038	157.9
	2	17,920†	48,960†	8,192♣	3.42	9.34	16.00	1.030	0.0038	157.9
	4	35,840†	48,960†	8,192♣	6.84	9.34	16.00	1.030	0.0038	157.9
	8	71,680†	48,960†	8,192♣	13.67	9.34	16.00	1.030	0.0038	157.9
	16	143,360†	48,960†	8,192♣	27.34	9.34	16.00	1.030	0.0038	157.9

Notes:

† Complexity in **R1CS constraints**, used for the Fractal-based baselines.

♣ Complexity in **polynomial trace degree** (N), used for the Plonky2 instantiation.

- **Prove/Step** reports the cost of one additional recursive delegation step, not the total cost of constructing an entire depth-L chain from scratch.
- We report only 100-bit and 128-bit settings because these are the security levels used in our concrete PQ evaluation.

tectural rather than lineage-comparative: unlike the non-post-quantum setting, there is no directly comparable prior DAC baseline built on recursive post-quantum proof machinery. We therefore compare three PQ architectural variants:

- **Baseline (Fractal-Lin):** a non-recursive Fractal-style instantiation [32], [33] in which signatures and state are aggregated linearly along the delegation chain;
- **Fractal (Fractal-Rec):** a recursive Fractal instantiation over R1CS, included as a functional recursive baseline for IVC-style delegation; and
- **Plonky2 (Ours):** our primary recursive PCC instantiation, using a Plonkish arithmetization over the Goldilocks field to optimize concrete proving efficiency.

We evaluate depths up to $L = 16$, which already exceeds realistic deployment depths for transitive-authorization systems while still exercising the regime in which recursive closing should have stabilized.

Table 3 compares non-recursive Fractal, recursive Fractal, and Plonky2-based PCC at 100-bit and 128-bit security. The non-recursive baseline grows linearly with L in constraints and memory, while both recursive designs remove this depth-dependent growth. In the 128-bit Plonky2 instantiation, PCC maintains a 157.9 KB final proof, 3.8 ms verification, and 1.03 s per recursive delegation step across all tested depths. These results support the main claim that PCC achieves depth-independent verifier-facing presentation with practical post-quantum recursive proving costs.

Answer to RQ3. At 128-bit security, Plonky2-based PCC stays constant at 157.9 KB, 3.8 ms verification, and 1.03 s per delegation step across all teste depths, while the non-recursive Fracta baseline grows linearl (143,360 constraints, 27.34 MB at $L = 16$); the cost is thus shifted to incremental delegation and remains practical at deployment-realistic depths.

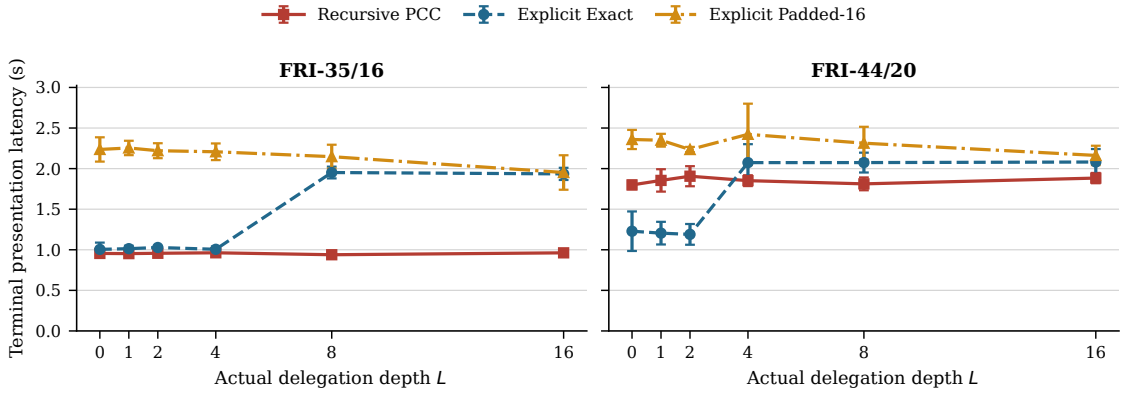
Summary and limitations. Overall, the experiments above support the claims tested by RQ1–RQ3: PCC improves the verifier-facing scaling profile over proof-based DAC baselines, the depth independence comes from recursive lineage carrying rather than a backend artifact, and the post-quantum instantiation remains practical at realistic delegation depths. The evaluation is limited to prototype implementations on a single workstation and delegation depths up to $L = 16$; it does not measure network effects, side-channel leakage, revocation mechanisms, or multi-party deployment overhead. These issues are outside the current cryptographic threat model and require separate system-level evaluation.

6.4. Presentation Latency and Memory under Matched FRI Configurations

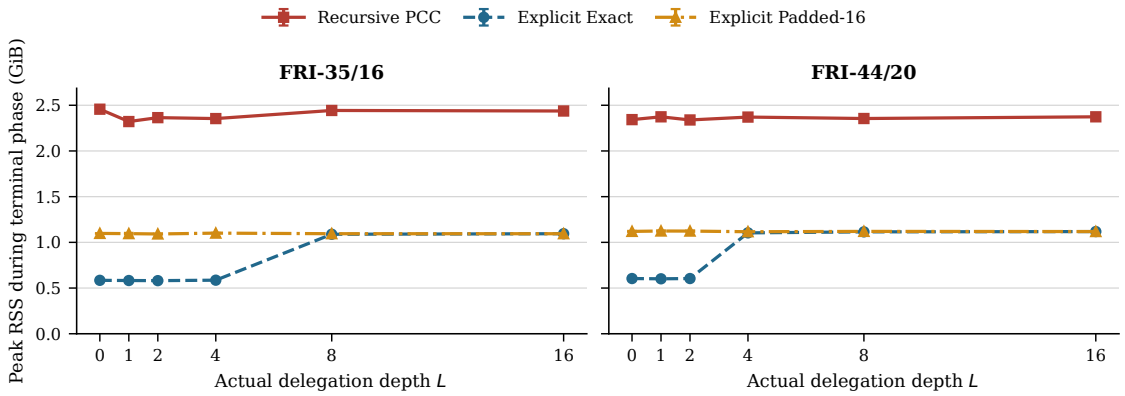
To isolate the effect of the credential representation from the backend, we compare recursive PCC with two explicit-history baselines in a separate FRI realization with a dedicated presentation circuit. Explicit-Exact checks the actual chain, whereas Explicit-Padded-16 checks a fixed 16-slot relation including inactive slots. Within each profile, all systems use the same Goldilocks/Poseidon stack, hash-based authorization, eight-bit policy and public showing inputs. We run five independent processes per system, profile and depth $L \in \{0, 1, 2, 4, 8, 16\}$ (180 runs), at 16 threads on an AMD Ryzen 7 9800X3D with 31.15 GiB of RAM under Windows. Presentation timing excludes circuit construction, issuance and delegation. The profiles denote FRI query and proof-of-work settings, not certified security levels.

Figure 6(a) shows that PCC presentation latency is nearly flat in depth: 0.939–0.963 s for FRI-35/16 and 1.799–1.906 s for FRI-44/20. Across the tested depths, PCC's mean latency is 50.7–57.7% and 12.9–23.7% lower than Padded-16, respectively. Exact is faster at the three shallowest depths under FRI-44/20. Figure 6(b) shows the cost of the recursive state: PCC uses about 2.1–2.2× the peak RSS of Padded-16. These are separate process-level measurements, not direct comparisons to the timings or memory figures in Table 3.

Answer to RQ4. PCC's presentation latency and peak resident memory stay nearly flat across the measured depths. Its mean presentation latency is lower than



(a) Terminal presentation latency.



(b) Absolute peak terminal resident memory (RSS).

Figure 6. Matched FRI measurements across actual delegation depths (16 threads, five independent processes per cell). The latency bars are two-sided Student-t 95% confidence intervals for the mean.

Padded-16's in both profiles, but its peak resident memory is higher; Explicit-Exact is faster at shallow depths under FRI-44/20.

7. Related Work

We focus on recursive proof systems and proof-carrying mechanisms most directly related to PCC; Section 1 discusses the DAC design space and anonymity trade-offs.

Incrementally Verifiable Computation (IVC) was formalized by Valiant [28]. Halo [34] introduced accumulation-based recursion without trusted setup, and Nova [35] improved recursive proving through folding. SuperNova [36] generalized this line to Non-Uniform IVC (NIVC) for heterogeneous step functions, while HyperNova [37] further improves folding-based recursion. PCC uses this abstraction for separate issuance, delegation, and presentation predicates. These systems provide the recursive-proof substrate, but do not define a DAC abstraction, address delegated authorization, or model holder-visible topology leakage.

PCC is also related to proof-carrying data (PCD): Chiesa and Tromer introduced PCD for attaching proofs to data flowing through distributed computations [38], and subsequent work connected PCD to accumulation schemes [39]. PCC differs in both object and privacy goal: its carried proof certifies credential lineage and authorization validity while hiding remote delegation topology from the terminal holder.

Transparent and post-quantum proof systems form our concrete backend. STARKs and FRI-based proof systems provide transparent post-quantum computational integrity [40], [41]; Fractal applies this line to recursion [32], and Plonky2 improves concrete efficiency via Plonkish arithmetization over the Goldilocks field [42]. We use Poseidon for arithmetization-friendly hashing [26]. Derecho [25] also uses proof-carrying techniques for privacy-preserving disclosures, but it does not address delegated authorization or the holder-side topology-hiding requirement captured by EDA.

8. Conclusion and Future Work

We introduced Enhanced Delegation Anonymity (EDA), which pairs verifier-side anonymity with holder-side topology hiding, and Proof Carrying Credentials (PCC), a recursive DAC framework that compiles delegation lineage into a depth-independent proof-carrying state without sacrificing restricted delegation or expressive presentation. Our Halo2 and Plonky2 instantiations achieve this at practical proving cost, including a post-quantum-compatible QROM instantiation. Future work includes revocation, threshold delegation, and broader recursive backends.

References

[1] A. Augusto, R. Belchior, M. Correia, A. Vasconcelos, L. Zhang, and T. Hardjono, "Sok: Security and privacy of blockchain interop-

erability," in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024, pp. 3840–3865.

[2] M. Bartoletti, J. H.-y. Chiang, and A. L. Lafuente, "Sok: lending pools in decentralized finance," in *International Conference on Financial Cryptography and Data Security*. Springer, 2021, pp. 553–578.

[3] "Defillama - defi dashboard and crypto analytics." https://defillama.com/, 2025.

[4] S. Werner, D. Perez, L. Gudgeon, A. Klages-Mundt, D. Harz, and W. Knottenbelt, "Sok: Decentralized finance (defi)," in *Proceedings of the 4th ACM Conference on Advances in Financial Technologies*, 2022, pp. 30–46.

[5] K. Qin, L. Zhou, P. Gamito, P. Jovanovic, and A. Gervais, "An empirical study of defi liquidations: Incentives, risks, and instabilities," in *Proceedings of the 21st ACM internet measurement conference*, 2021, pp. 336–350.

[6] "Defi's largest lending network." https://aave.com/, 2022.

[7] M. Chase and A. Lysyanskaya, "On signatures of knowledge," in *Annual International Cryptology Conference*. Springer, 2006, pp. 78–96.

[8] M. Belenkiy, J. Camenisch, M. Chase, M. Kohlweiss, A. Lysyanskaya, and H. Shacham, "Randomizable proofs and delegatable anonymous credentials," in *Annual international cryptology conference*. Springer, 2009, pp. 108–125.

[9] M. Chase, M. Kohlweiss, A. Lysyanskaya, and S. Meiklejohn, "Malleable signatures: New definitions and delegatable anonymous credentials," in *2014 IEEE 27th computer security foundations symposium*. IEEE, 2014, pp. 199–213.

[10] E. C. Crites and A. Lysyanskaya, "Delegatable anonymous credentials from mercurial signatures," in *Cryptographers' Track at the RSA Conference*. Springer, 2019, pp. 535–555.

[11] J. Camenisch, M. Drijvers, and M. Dubovitskaya, "Practical uc-secure delegatable credentials with attributes and their application to blockchain," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 683–699.

[12] O. Mir, D. Slamanig, B. Bauer, and R. Mayrhofer, "Practical delegatable anonymous credentials from equivalence class signatures," *Proceedings on Privacy Enhancing Technologies*, 2023.

[13] S. Griffy, A. Lysyanskaya, O. Mir, O. Perez Kempner, and D. Slamanig, "Delegatable anonymous credentials from mercurial signatures with stronger privacy," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2024, pp. 296–325.

[14] O. Mir, D. Slamanig, and R. Mayrhofer, "Threshold delegatable anonymous credentials with controlled and fine-grained delegation," *IEEE Transactions on Dependable and Secure Computing*, vol. 21, no. 4, pp. 2312–2326, 2024.

[15] M. Xie, P. Ju, Y. Zhao, Z. L. Jiang, M. H. Au, J. Fang, Y. Yu, and X. Wang, "Fdaac-cr: Practical delegatable attribute-based anonymous credentials with fine-grained delegation management and chainable revocation," *IEEE Transactions on Information Forensics and Security*, 2025.

[16] C. F. Torres, F. Willi, and S. Shinde, "Is your wallet snitching on you? an analysis on the privacy implications of web3," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 769–786.

[17] K. Li, J. Chen, X. Liu, Y. R. Tang, X. Wang, and X. Luo, "As strong as its weakest link: How to break blockchain dapps at rpc service." in *NDSS*, 2021.

[18] Y. Yu, T. Sharma, S. Das, and Y. Wang, "" don't put all your eggs in one basket": How cryptocurrency users choose and secure their wallets," in *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*, 2024, pp. 1–17.

[19] Z. Zhou, T. Sharma, L. Emano, S. Das, and Y. Wang, "Iterative design of an accessible crypto wallet for blind users," in *Nineteenth Symposium on Usable Privacy and Security (SOUPS 2023)*, 2023, pp. 381–398.

[20] A. Chiesa, R. Lehmkuhl, P. Mishra, and Y. Zhang, "Eos: Efficient private delegation of {zkSNARK} provers," in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 6453–6469.

[21] Y. Hu, P. Mishra, X. Wang, J. Xie, K. Yang, Y. Yu, and Y. Zhang, "{DFS}: Delegation-friendly {zkSNARK} and private delegation of provers," in *34th USENIX Security Symposium (USENIX Security 25)*, 2025, pp. 2065–2084.

[22] Y. Yang, Y. Cheng, K. Wang, X. Li, J. Sun, J. Shen, X. Dong, Z. Cao, G. Yang, and R. H. Deng, "Siniel: Distributed privacy-preserving zksnark," 2025.

[23] K. Abbaszadeh, H. Hafezi, J. Katz, and S. Meiklejohn, "Single-server private outsourcing of zk-snarks," in *2026 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2026, pp. 3075–3090.

[24] Aave, "Aave V3 credit delegation," https://aave.com/docs/aave-v3/guides/credit-delegation.

[25] J. Beal and B. Fisch, "Derecho: privacy pools with proof-carrying disclosures," in *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, 2024, pp. 3197–3211.

[26] L. Grassi, D. Khovratovich, C. Rechberger, A. Roy, and M. Schofnegger, "Poseidon: A new hash function for {Zero-Knowledge} proof systems," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 519–535.

[27] J. Blömer and J. Bobolz, "Delegatable attribute-based anonymous credentials from dynamically malleable signatures," in *Applied Cryptography and Network Security*, 2018. [Online]. Available: https://eprint.iacr.org/2018/340

[28] P. Valiant, "Incrementally verifiable computation or proofs of knowledge imply time/space efficiency," in *Theory of Cryptography Conference (TCC)*. Springer, 2008, pp. 1–18.

[29] A. Chiesa, Z. Guan, S. Samocha, and E. Yogev, "Security bounds for proof-carrying data from straightline extractors," in *Theory of Cryptography Conference*. Springer, 2024, pp. 464–496.

[30] D. J. Bernstein, A. Hülsing, S. Kölbl, R. Niederhagen, J. Rijneveld, and P. Schwabe, "The SPHINCS+ signature framework," Cryptology ePrint Archive, Paper 2019/1086, 2019. [Online]. Available: https://eprint.iacr.org/2019/1086

[31] zcash, "Solidity verifier generator for halo2 proof with kzg polynomial commitment scheme on bn254." https://github.com/privacy-scaling-explorations/halo2, 2021.

[32] A. Chiesa, D. Ojha, and N. Spooner, "Fractal: Post-quantum and transparent recursive proofs from holography," in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2020, pp. 769–793.

[33] "libiop: a c++ library for iop-based zksnarks," https://github.com/scipr-lab/libiop, 2019.

[34] S. Bowe, J. Grigg, and D. Hopwood, "Halo: Recursive proof composition without a trusted setup," *Cryptology ePrint Archive*, 2019.

[35] A. Kothapalli, S. Setty, and I. Tzialla, "Nova: Recursive zero-knowledge arguments from folding schemes," in *Annual International Cryptology Conference*. Springer, 2022, pp. 359–388.

[36] A. Kothapalli and S. Setty, "Supernova: Proving universal machine executions without universal circuits," *Cryptology ePrint Archive*, 2022.

[37] A. Koth and S. Setty, "Hypernova: Recursive arguments for customizable constraint systems," in *Annual International Cryptology Conference*. Springer, 2024, pp. 345–379.

[38] A. Chiesa and E. Tromer, "Proof-carrying data and hearsay arguments from signature cards." in *ICS*, vol. 10, 2010, pp. 310–331.

[39] B. Bünz, A. Chiesa, P. Mishra, and N. Spooner, "Proof-carrying data from accumulation schemes," *Cryptology ePrint Archive*, 2020.

[40] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev, "Scalable, transparent, and post-quantum secure computational integrity," *Cryptology ePrint Archive*, 2018.

[41] E. Ben, I. Bentov, Y. Horesh, and M. Riabzev, "Fast reed-solomon interactive oracle proofs of proximity," in *45th international colloquium on automata, languages, and programming (icalp 2018)*. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, pp. 14–1.

[42] zkevm, "Plonky2 is a snark implementation based on techniques from plonk and fri. it is the successor of plonky, which was based on plonk and halo." https://github.com/0xPolygonZero/plonky2, 2024.

[43] B. Abdolmaleki, M. Campanelli, Q. Dao, and H. Khoshakhlagh, "On the simulation-extractability of proof-carrying data," Cryptology ePrint Archive, Paper 2025/2037, 2025. [Online]. Available: https://eprint.iacr.org/2025/2037

[44] M. Chen, A. Chiesa, T. Gur, J. O'Connor, and N. Spooner, "Proof-carrying data from arithmetized random oracles," IACR Cryptology ePrint Archive, Report 2023/587, 2023.

Appendix A. Supplementary Protocol Details

A.1. NIVC: Full Syntax and Security Properties

Definition 6 (PCC-Compatible Non-Uniform Incrementally Verifiable Computation)**.** *A PCC-compatible Non-Uniform Incrementally Verifiable Computation scheme for a set of k polynomial-time, deterministic state-transition verification functions $\mathsf{F} = \{\mathsf{F}_1, \mathsf{F}_2, \ldots, \mathsf{F}_k\}$ is a tuple of four polynomial-time algorithms $(\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Prove}, \mathsf{Verify})$ with the following syntax: We write ρ for public instance data shared by all transition predicates; in PCC, ρ includes the authenticated root anchor* RootAnchor.

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$*: A probabilistic algorithm that takes a security parameter λ and outputs public parameters* pp*. These parameters are assumed to be implicitly available to all other algorithms.*
- $\mathsf{KeyGen}(\mathsf{pp}, \mathsf{F}) \to (\mathsf{pk}, \mathsf{vk})$*: A deterministic algorithm that takes the public parameters* pp *and a description of the transition-function family* F *to produce a prover key* pk *and a verifier key* vk*.*
- $\mathsf{Prove}(\mathsf{pk}, j, \rho, z_0, z_i, z_{i+1}, w_{i+1}, \pi_i) \to \pi_{i+1}$*: A probabilistic algorithm for an incremental step. It takes the prover key* pk*, a function selector $j \in \{1, \ldots, k\}$, the public instance data ρ, the initial state z_0, the current state z_i, the next state z_{i+1}, a non-deterministic transition witness w_{i+1}, and the previous recursive proof π_i. It outputs an updated proof π_{i+1} of constant size, attesting that the previous recursive computation is valid and that the current transition satisfies $\mathsf{F}_j(\rho, z_i, z_{i+1}, w_{i+1}) = 1$. For the base step, π_0 is taken to be a distinguished empty proof $\perp$, which only initializes recursion and does not authenticate the initial credential state. In PCC, root authenticity is enforced by the ϕ_{issue} transition using the root verification key contained in ρ.*
- $\mathsf{Verify}(\mathsf{vk}, \rho, z_0, z_n, \pi_n, \tau_{\mathsf{pub}}) \to \{0, 1\}$*: A deterministic algorithm that takes the verifier key* vk*, the public instance data ρ, the initial public state z_0, the final public state z_n, the final recursive proof π_n, and a public terminal condition τ_{pub}. It outputs 1 if π_n certifies the existence of a valid recursive execution from z_0 to z_n satisfying τ_{pub}, and outputs 0 otherwise.*

The public recursive statement is $x = (\rho, z_{\mathsf{root}}, z_{\mathsf{out}}, \tau_{\mathsf{pub}})$. The terminal conditions are typed: τ_{cred} denotes an issue followed by zero or more delegations; τ_{pre} denotes such a credential trace followed by one presentation. These are conditions on the entire execution, not just on its last selector. In particular, issue is the only base transition, and a presentation proof is not a credential predecessor. The actual length, non-terminal selectors, intermediate states, and witnesses are not public inputs.

Definition 7 (PCC-compatible proof security)**.** *In addition to completeness for the typed execution language (Definition 9), the NIVC interface requires the following properties in a common setup and oracle environment.*

- *Adaptive auxiliary-input zero knowledge. One stateful simulator produces proof objects for true statements x, without the private penultimate state, selector history, or witnesses. Replacing honest proofs by these simulations remains indistinguishable in the presence of the games' permitted auxiliary data and oracle responses, including honestly produced credential proofs when simulating presentations. For holder privacy the simulated credential must also support the public presentation-proving operation using the fixed local witness.*
- *Fresh-statement extraction after simulated showings. For an accepting output (x^*, π^*) with x^* outside the log of simulated statements, an extractor obtains a finite typed lineage with its transition witnesses for that output, except with negligible failure. This must hold in the presence of genuinely produced credential proofs and permitted system-oracle responses. The extractor may use the backend's simulation/extraction state and retained honest credential witnesses. It must operate on the same accepting output and signer transcript, without the external signing challenge secret, resetting external state, or adding signing queries.*

The trace has polynomial length, and the extraction work is expected polynomial in the security parameter and the experiment's polynomial trace horizon, including locally generated updates. This horizon is a proof parameter, not a serialized depth input. These are explicit recursive composition requirements; ordinary soundness or one-step knowledge soundness does not by itself give them [29], [43].

A.2. Primitive Details for the Post-Quantum Instantiation

Field Arithmetic We choose the 64-bit Goldilocks prime field $\mathbb{F}_p$ with modulus $p = 2^{64} - 2^{32} + 1$. This specific prime is chosen fundamentally for its hardware alignment: unlike larger fields that require multi-precision arithmetic, Goldilocks operations fit natively within 64-bit CPU registers and support aggressive SIMD vectorization, thereby maximizing the prover's arithmetic throughput. Furthermore, the modulus satisfies $p \equiv 1 \pmod{2^{32}}$, ensuring high 2-adicity to support the efficient Number Theoretic Transforms (NTT) required for polynomial commitments. The algebraic circuit is defined over $\mathbb{F}_p$ for efficiency, while FRI challenges are sampled from the quadratic extension $\mathbb{F}_p^2$. The size of that challenge space is not a bound on the protocol's soundness error or an end-to-end post-quantum security level. **Arithmetization-Friendly Hash Function** A critical component of our system is the

TABLE 4. PRIMITIVE STACK FOR THE POST-QUANTUM-COMPATIBLE PCC INSTANTIATION.

Component	Instantiation	Role / Parameter Choice
Algebraic layer		
Field arithmetic	Goldilocks $\mathbb{F}_p$, $p = 2^{64} - 2^{32} + 1$; FRI over $\mathbb{F}_p^2$	64-bit native arithmetic; high 2-adicity; extension-field challenges.
Hash function	Poseidon / Goldilocks; capacity $c = 4$, width $w = 12$	State binding, Merkle authentication, FS hashing in-circuit.
Cryptographic primitives		
Issuance and delegation signature	Poseidon-based hash-based sig. (SPHINCS$^+$-style)	PQ root issuance and delegation authorization; circuit-friendly verification.
Recursive proof system	Plonky2-style FRI recursion (Fractal-compatible)	Transparent recursion; no pairings; no trusted setup.
Protocol interfaces		
Oracle domains	H_{state}, H_{sig}, H_{FS}, H_{pre}	Domain separation for state, signatures, FS, presentation binding.

domain-separated hash layer implementing H_{state}, H_{sig}, H_{pre}, and H_{FS}. We instantiate H using Poseidon hash [26], which is designed for low algebraic complexity.

- **Configuration:** We instantiate the Poseidon hash function over the 64-bit Goldilocks field. We configure the sponge with capacity $c = 4$ field elements and state width $w = 12$. These are parameters, not an estimate of concrete classical or quantum attack cost.
- **Efficiency:** Unlike SHA-3, which relies on bitwise operations that incur prohibitive overhead in arithmetic circuits, Poseidon is designed natively for algebraic execution. Furthermore, our use of the 64-bit Goldilocks field allows for vectorized arithmetic (e.g., AVX-512) on modern CPUs. This design significantly reduces the number of constraints per round and enables orders-of-magnitude faster proof generation compared to schemes operating over 256-bit prime fields.

Post-Quantum Signature Scheme (Σ) To instantiate the signature scheme Σ used for root issuance and delegation authorization, we use a Plonky2-native hash-based signature construction. Structurally analogous to SPHINCS+ [30], our construction is tailored for arithmetic circuit efficiency by exclusively employing the Poseidon hash function over the Goldilocks field. In this scheme, the verification key is defined as the root of a Merkle tree, where leaves correspond to one-time signing keys (instantiated via Poseidon-based W-OTS). Consequently, a valid signature consists of a one-time signature and the Merkle authentication path linking the leaf to the root. This design allows the verification logic to be realized as a highly efficient native gadget within the recursive circuit, incurring negligible overhead compared to generic post-quantum signatures. The analysis assumes EUF-CMA of the signing implementation, including safe management of one-time signing leaves. The structural analogy is not an identification with standardized SPHINCS$^+$ or a proof of stateless security.

Post-Quantum NIVC The realization of post-quantum NIVC primarily relies on recursive arguments constructed from the Fast Reed-Solomon Interactive Oracle Proofs of Proximity (FRI) protocol. Fractal [32] pioneered this direction by instantiating the recursive accumulation paradigm over Rank-1 Constraint Systems (R1CS), demonstrating for the first time that recursive composition is feasible without relying on quantum-vulnerable bilinear pairings. Building upon this foundation, Plonky2 [42] significantly reduced the recursion overhead by adopting Plonkish arithmetization and operating over the 64-bit Goldilocks field, which enables highly efficient, vectorized field arithmetic on modern CPUs. A defining advantage of these FRI-based constructions is their use of hash functions without a trusted setup. Their post-quantum proof properties remain conditional on the assumptions in Appendix B.1. While our PCC allows for flexible instantiation with any FRI-based recursive system, we prioritize Plonky2 in our primary construction to leverage its specialized circuit optimizations. However, to provide a comprehensive comparative analysis of circuit complexity and scalability, we implement and evaluate both the Fractal-based and Plonky2-based instantiations in Section 6.

A.3. Detailed construction of the PQDAC Protocol

Figure 7 gives the full algorithmic specification of Π_{PQDAC}, the post-quantum instantiation summarized in Section 5.2. It makes explicit the witness and argument construction that the wrapper PCC.Derive encapsulates at each phase: Setup compiles Φ into the proving and verification keys; KeyGen derives user keys as post-quantum signature keypairs; and Issue, Delegate, and Present are single PCC.Derive calls in issuance, delegation, and presentation mode, respectively, each extending the carried token by one transition. Verify reduces to a single PCC.Verify check against the authenticated RootAnchor. All cryptographic logic is inherited from the generic Π_{PCC} (Definition 5); the only instantiation-specific choices are the primitives of Table 4.

A.4. Presentation-Policy Complexity

Table 5 reports the final verifier-facing proof size under presentation predicates of increasing complexity. It is a policy-complexity sanity check rather than a fully controlled cross-system microbenchmark: the PCC column reports measured Halo2 proof sizes, whereas the prior columns summarize representative proof-size contributions reported or implied by prior constructions. Across all tested policy classes, PCC keeps the final proof at 7.344 KB.

Appendix B. PQDAC Security

B.1. Assumptions and QROM Model

The proofs below require the typed NIVC interface in Definition 7, EUF-CMA security of Σ, binding state commitments, and collision resistance of the hashes on

The PQDAC Protocol

Setup$(1^\lambda) \to (\mathsf{pp}, \mathsf{pk}, \mathsf{vk})$:
1. Generate generic public parameters for the underlying PCC scheme: $\mathsf{pp} \leftarrow \mathsf{PCC.Setup}(1^\lambda)$.
2. Define $\Phi = \{\phi_{\mathsf{issue}}, \phi_{\mathsf{del}}, \phi_{\mathsf{pre}}\}$ as the recursive relation logic for issuance, delegation, and presentation.
3. Compile the proving and verification keys using the PCC indexer: $(\mathsf{pk}, \mathsf{vk}) \leftarrow \mathsf{PCC.KeyGen}(\mathsf{pp})$.
4. Return $(\mathsf{pp}, \mathsf{pk}, \mathsf{vk})$.

KeyGen$(\mathsf{pp}) \to (\mathsf{usk}, \mathsf{upk})$**:**
1. Parse public parameters pp to retrieve the security parameter λ and the underlying finite field $\mathbb{F}$.
2. Run the standard key generation algorithm for the post-quantum signature component: $(\mathsf{sk}_{\mathsf{sig}}, \mathsf{pk}_{\mathsf{sig}}) \leftarrow \Sigma.\mathsf{KeyGen}(1^\lambda)$.
3. Set the user secret key: $\mathsf{usk} := \mathsf{sk}_{\mathsf{sig}}$.
4. Set the user public key: $\mathsf{upk} := \mathsf{pk}_{\mathsf{sig}}$.
5. Return $(\mathsf{usk}, \mathsf{upk})$.

Issue$(\mathsf{RootAnchor}, \mathsf{upk}_0, A_0; \mathsf{isk}_{\mathsf{root}}) \to \mathsf{PCCred}_0$**:**
1. Set $\rho := \mathsf{RootAnchor}$ and $z_{\mathsf{root}} := H_{\mathsf{state}}(\mathsf{tag}_{\mathsf{issue}}, \rho)$.
2. Define issuance arguments: $\mathsf{args}_{\mathsf{iss}} := (\mathsf{upk}_0, A_0, \mathsf{meta}_{\mathsf{iss}})$.
3. Invoke the PCC wrapper in issuance mode:
$$\mathsf{Token}_0 \leftarrow \mathsf{PCC.Derive}(\rho, z_{\mathsf{root}}, \bot, \mathsf{args}_{\mathsf{iss}}; \mathsf{isk}_{\mathsf{root}}).$$
4. Construct the initial credential:
$$\mathsf{PCCred}_0 := (\rho, \mathsf{Token}_0).$$
5. Return PCCred_0.

Delegate$(\mathsf{PCCred}_i, \mathsf{upk}_{i+1}, A_{i+1}; \mathsf{usk}_i) \to \mathsf{PCCred}_{i+1}$**:**
1. Parse input credential PCCred_i as (ρ, Token_i) and set $z_{\mathsf{root}} := H_{\mathsf{state}}(\mathsf{tag}_{\mathsf{issue}}, \rho)$.
2. Define $\mathsf{args}_{i+1} := (\mathsf{upk}_{i+1}, A_{i+1}, \mathsf{meta}_{\mathsf{del}})$. Invoke the PCC recursive wrapper to delegate authority to U_{i+1}:
$$\mathsf{Token}_{i+1} \leftarrow \mathsf{PCC.Derive}(\rho, z_{\mathsf{root}}, \mathsf{Token}_i, \mathsf{args}_{i+1}; \mathsf{usk}_i).$$
3. This step recursively validates Token_i and enforces the delegation predicate ϕ_{del}.
4. Construct the new delegated credential: $\mathsf{PCCred}_{i+1} := (\rho, \mathsf{Token}_{i+1})$.
5. Return PCCred_{i+1}.

Present$(\mathsf{PCCred}_n, \mathsf{usk}_n, \mathsf{Policy}, m, A_{\mathsf{pre}}) \to \Pi$**:**
1. Set $z_{\mathsf{root}} := H_{\mathsf{state}}(\mathsf{tag}_{\mathsf{issue}}, \rho)$ and parse PCCred_n as (ρ, Token_n).
2. Define presentation context $\mathsf{ctx} := (m, \mathsf{Policy})$ and $\mathsf{args}_{\mathsf{pre}} := (\mathsf{ctx}, A_{\mathsf{pre}})$.
3. Invoke the PCC wrapper in presentation mode:
$$\mathsf{Token}_{\mathsf{pre}} \leftarrow \mathsf{PCC.Derive}(\rho, z_{\mathsf{root}}, \mathsf{Token}_n, \mathsf{args}_{\mathsf{pre}}; \mathsf{usk}_n).$$
4. Parse $\mathsf{Token}_{\mathsf{pre}} = (z_{\mathsf{pre}}, \pi_{\mathsf{pre}}, \mathsf{ctx}, A_{\mathsf{pre}})$.
5. Return presentation $\Pi := (z_{\mathsf{pre}}, \pi_{\mathsf{pre}}, m, \mathsf{Policy}, A_{\mathsf{pre}})$.

Verify$(\mathsf{vk}, \mathsf{RootAnchor}, \Pi) \to \{0, 1\}$**:**
1. Parse Π as $(z_{\mathsf{pre}}, \pi_{\mathsf{pre}}, m, \mathsf{Policy}, A_{\mathsf{pre}})$.
2. Set $\rho := \mathsf{RootAnchor}$, reconstruct $\mathsf{ctx} := (m, \mathsf{Policy})$, and form $\mathsf{Token}_{\mathsf{pub}} := (z_{\mathsf{pre}}, \pi_{\mathsf{pre}}, \mathsf{ctx}, A_{\mathsf{pre}})$.
3. Invoke the PCC verifier relative to the authenticated root anchor: $b \leftarrow \mathsf{PCC.Verify}(\mathsf{vk}, \rho, \mathsf{Token}_{\mathsf{pub}})$.
4. Return bit b.

Figure 7. Detailed construction of the PQDAC Protocol instantiated over the generic PCC. The left column describes the system setup and issuance phases, while the right column details the delegation, presentation, and verification processes.

TABLE 5. IMPACT OF PRESENTATION-POLICY COMPLEXITY ON FINAL PROOF SIZE. PRIOR COLUMNS ARE LITERATURE-BASED ESTIMATES ILLUSTRATING QUALITATIVE SCALING; PCC REPORTS MEASURED HALO2 FINAL PROOF SIZE.

Policy	Logic	P-UC-DAC [11]	Mercurial [13]	Ours
Simple	$\mathcal{L}_{\mathsf{sig}}$	≈ 0.5 KB	≈ 0.2 KB	7.344 KB
Medium	$\mathcal{L}_{\mathsf{range}}$	≈ 1.5 KB[a]	≈ 1.5 KB[a]	7.344 KB
Complex	$\mathcal{L}_{\mathsf{memb}}$	≈ 20 KB[b]	≈ 20 KB[b]	7.344 KB

[a] Range-proof contribution to the presentation size.
[b] Set-membership-proof contribution to the presentation size.

canonically encoded signed and public statements. In the QROM version the attacker has quantum access to the modeled hash interfaces, but the signing and DAC service queries are classical. We distinguish the domains H_{state}, H_{sig}, H_{FS}, and H_{pre} for the state binding, signature operations, Fiat–Shamir transcript, and presentation binding, respectively. The simulated and real proofs must be compatible with the same setup and oracle environment, including the interactions with honestly produced credential proofs and signing responses. Any Fiat–Shamir programming and its loss are part of the assumed proof interface, not consequences of domain separation.

Modeling a transcript hash as a quantum-accessible random oracle does not automatically provide a circuit for evaluating an ideal oracle inside a recursive proof; arithmetic random-oracle models distinguish these interfaces [44]. The role of in-circuit concrete Poseidon must be justified separately in any reduction to an ideal-oracle model. Neither ordinary FRI soundness nor the extension-field and sponge parameters alone establish the extraction and zero-knowledge assumptions for the evaluated backend. The conclusion below is conditional and does not assign a concrete quantum security level.

B.2. Delegation Unforgeability

Definition 8 (Usable terminal signing witness)**.** *When the presentation predicate accepts a secret-key witness* usk *as corresponding to the opening's committed key* upk*, that witness must enable the reduction to sign a fresh digest that verifies under* upk*. For an ordinary signature key pair this is the key relation with signing correctness. If the signing interface requires additional state, the terminal witness or the relation must permit recovery of sufficient usable state without the honest holder's secret. A check on a cached public-key field alone would not suffice.*

Proof of Theorem 2.

Use the honest-signing ledger of Definition 2. First replace each successful honest Show proof with a simulation

for its true canonical statement $x = (\rho, z_{\mathsf{root}}, z_{\mathsf{pre}}, \tau_{\mathsf{pre}})$. The handles, signing calls, ledger entries, claimed attributes, and public context checks are unchanged. Soundness of the handle checks and honest credential evolution ensures the simulated statements are true. Adaptive auxiliary-input zero knowledge bounds this change even when the adversary has real credential tokens and other oracle outputs. The same canonical statement is recorded in the show log in both games, so a change of proof bytes is not a fresh statement.

On a fresh accepting presentation, use extraction in the simulated experiment to obtain a finite trace with one issue, zero or more delegations, and one presentation, for the same trusted root, public context, and disclosed authorization A^*. The extractor is tied to the accepting output and the signing transcript. The predicate at the end opens the predecessor commitment, extracts its signing-key witness, and enforces the authorization restriction. When adjacent witnesses give different openings of the same commitment, the reduction can output a binding violation. Canonical context and attribute encodings and the presentation binding ensure that the inspection concerns the same accepted public statement.

Scan the trace from the root, matching each signed transition to the ledger entry for the signing key. Initially the signer is the honest root. While the signer is honest, if its extracted signed digest is absent from its signing ledger, the trace contains an EUF-CMA forgery. If the digest is present, compare the structured message and openings with the ledger record. A different structured message with that digest gives a collision in H_{sig}, and different openings of the same state commitment give a binding violation. Otherwise the entry fixes the matched transition and its registered destination, so the scan can continue from this state. The comparison is of signed messages, not signature bytes.

If a matched transition first reaches an adversary-controlled holder, the corresponding authorization is in the authorized closure. This includes credentials independently obtained by holders before a later corruption, as well as those exposed or delivered thereafter. Every later delegation and the final presentation preserve the authorization restriction, so transitivity puts A^* in the closure, contrary to the winning condition. Thus the scan needs no ledger entries for transitions performed after reaching an adversarial key.

If there is no missing signature and no binding violation, the only remaining case is that the trace reaches an uncorrupted, registered honest holder. The extracted terminal witness permits a signature on a digest that has not been signed for that holder (Definition 8), again yielding EUF-CMA. This case is essential: merely identifying a fully authorized lineage does not show that the adversary could present as its terminal holder.

For each signature case the reduction guesses the implicated honestly generated verification key (including the root for issuance) and embeds its EUF-CMA challenge at that registration. It uses the challenger for that key's authorized signing calls and aborts if that key is corrupted. Credential proofs use public keys, state openings, and the already obtained signatures; the guessed key's secret is not needed to generate them. For honest showings under the guessed key, the reduction already uses simulated proofs for true statements, so it does not need this secret until extraction. Its guess succeeds with inverse polynomial probability, because the implicated key is still honest. Signing after terminal extraction is performed with the extracted witness, not with the EUF-CMA oracle. Extraction cannot add queries to the signing ledger or rewind the external signer. For q_{key} candidate keys the bound is of the form

$$\mathsf{Adv}^{\mathsf{uf}}(\mathcal{A}) \le \epsilon_{\mathsf{zk}} + \epsilon_{\mathsf{ext}} + \epsilon_{\mathsf{bind}} + q_{\mathsf{key}}\mathsf{Adv}^{\mathsf{euf}}_{\Sigma}(\mathcal{B}) + \mathsf{negl}(\lambda).$$

Here ϵ_{bind} covers commitment and canonical hashed statement binding, and the negligible term includes honest proof generation failures. The argument concerns the public authorization of a fresh presentation. In this construction the public presentation has no token-specific identifier on which to base a separate ownership test. □

B.3. Enhanced Delegation Anonymity

Lemma 1 (Verifier-side anonymity)**.** *If the presentation proofs satisfy the zero-knowledge interface of Definition 7 and all non-proof parts of the observable presentation are determined by its public statement and allowed auxiliary information, then PCC satisfies Definition 3.*

Proof. For either admissible path, the presentation proof has the same statement

$$x_{\mathsf{pre}} = (\rho, z_{\mathsf{root}}, H_{\mathsf{pre}}(\mathsf{pre}, ctx, A_{\mathsf{pre}}), \tau_{\mathsf{pre}}(ctx, A_{\mathsf{pre}})).$$

The verifier does not receive the predecessor commitment, predecessor proof, or terminal key in the native presentation. The two worlds are indistinguishable from the same experiment that simulates the fresh proof using only x_{pre} in the common setup. Auxiliary inputs permitted by the game, including known proof objects, remain available in this hybrid. Any extra application fields must be matched as required by the game; no pseudonym is added by the PCC wrapper. □

Lemma 2 (Holder-side topology obliviousness)**.** *If credential proofs satisfy Definition 7 and the token and proving interfaces expose no extra topology-dependent metadata, then PCC satisfies Definition 4.*

Proof. The challenge fixes the same terminal key pair, authorization, public parameters, presentation inputs, and allowed incoming-edge information. Couple the honest final handoffs by using the same freshly sampled opening randomness r_H in both experiments; each marginal has the prescribed honest distribution. The terminal commitment $z_H = \mathsf{Com}(\mathsf{upk}_H, A_H; r_H)$ is then common. The honestly delivered token contains $(\rho, z_H, \pi_H, \mathsf{upk}_H, A_H, r_H)$, hence its only field still dependent on the choice of remote path is the credential proof π_H and any metadata attached to it. The proof may nevertheless be correlated with auxiliary information about the prefixes, so we use auxiliary-input zero knowledge for an honestly generated handoff, at the common statement $x_H = (\rho, z_{\mathsf{root}}, z_H, \tau_{\mathsf{cred}})$.

Replace only π_H by a simulation for x_H. The holder's own presentation is then obtained by actually running the prescribed PCC.Derive on the simulated accepting predecessor, the same local opening and secret, and fresh holder

presentation randomness. This is efficient post-processing in the common setup and oracle environment: the wrapper needs no predecessor witness beyond the delivered proof. It preserves the correlations among the credential bytes, the actual inputs to the holder's proving invocation, the holder's coins if observable, and the resulting presentation. In particular, we do not simulate the presentation independently of its observable proving inputs. Both hybrid views now depend only on the common terminal data, allowed auxiliary information, and x_H, so they are indistinguishable. This does not hide credentials or proof-generation traces that a colluding predecessor independently discloses. Any length, verifier-data identifier, parser field, or other exposed metadata that varies with remote depth would violate the stated interface condition. □

Proof of Theorem 3. These two lemmas establish the two privacy experiments in the stated leakage and oracle model. □

B.4. Post-Quantum Security in the QROM

Proof of Theorem 4. Assume the same interface, binding, usable signing witness, and signature-security hypotheses against quantum polynomial-time adversaries in the common model of Appendix B.1. The integrity reduction uses the extractor for the observed accepting output and does not implement a classical rewind of an external quantum adversary or signer. The anonymity reductions use computational indistinguishability and the honest presentation operation as post-processing. Thus the same reductions apply with the corresponding quantum advantages. This is a statement about preservation of the assumed properties, not an instantiation proof for the measured FRI and Poseidon parameters. □

Appendix C. Wrapper-Level Security Games for PCC

The wrapper forwards recursive steps to NIVC.Prove. Its security interface concerns the recursive public statement $x = (\rho, z_{\mathsf{root}}, z_{\mathsf{out}}, \tau_{\mathsf{pub}})$, distinct from the holder's full token or the prover's private inputs. We use the typed terminal conditions $\tau_{\mathsf{cred}} = (\mathsf{tag} = \mathsf{cred})$ and $\tau_{\mathsf{pre}}(ctx, A_{\mathsf{pre}}) = (\mathsf{tag} = \mathsf{pre}, ctx, A_{\mathsf{pre}})$.

Definition 9 (Valid PCC lineage)**.** *A lineage for x is a finite list of states $(s_0, \ldots, s_n)$, selectors $(j_1, \ldots, j_n)$ and witnesses $(w_1, \ldots, w_n)$. It starts with $s_0 = z_{\mathsf{root}} = H_{\mathsf{state}}(\mathsf{tag}_{\mathsf{issue}}, \rho)$, ends with $s_n = z_{\mathsf{out}}$, and every step verifies $\phi_{j_i}(\rho, s_{i-1}, s_i, w_i) = 1$ under the same ρ. For a credential terminal condition the selectors have the form* issue del**; for a presentation terminal condition they have the form* issue del* pre*. In the latter case, the presentation witness uses the public (ctx, A_{pre}) and the last state is $H_{\mathsf{pre}}(\mathsf{pre}, ctx, A_{\mathsf{pre}})$. At each non-base step the prior proof is checked for the credential terminal condition and the same predecessor state. The depth is bounded by the experiment's proof horizon, not exposed to the verifier.*

Definition 10 (Wrapper-level completeness)**.** *An honestly invoked* PCC.Derive* *accepts its valid base or recursive inputs and yields an accepting proof for the corresponding typed statement, except with negligible probability. The base case uses only an issue step from z_{root} with predecessor $\perp$. The other steps extend a credential proof by delegation or close it by presentation.*

Definition 11 (Wrapper-level simulation extractability)**.** *Consider the wrapper experiment with access to honest credential generation and simulated proofs for adaptively chosen true statements. Log each simulated canonical public statement x in Q_{sim}. The probability of an accepting output (x^*, π^*) with $x^* \notin Q_{\mathsf{sim}}$ for which the extractor does not return a valid lineage for that output is negligible. The extractor and auxiliary-oracle restrictions are those of Definition 7. The log is over statements, not over proof encodings; in particular, the premise is not a claim about an altered proof of an already simulated statement.*

Definition 12 (Wrapper-level zero knowledge)**.** *In the real experiment a query supplies a valid wrapper proving input for x and receives the honest proof. In the simulation experiment the same admissibility checks are applied, and the proof is simulated using only x and the simulator's state. Invalid queries return $\perp$ in both experiments. No efficient adversary distinguishes the experiments, including the permitted auxiliary input and system responses, with non-negligible advantage. The simulated proof is returned in the same observable interface and layout as the real proof; private mode and predecessor-state data are not simulator inputs.*

Theorem 5 (Security of the PCC wrapper)**.** *For a PCC-compatible NIVC satisfying Definition 7 for the typed execution language,* PCC.Derive* *and the corresponding verifier preserve wrapper-level completeness, simulation extractability, and zero knowledge.*

Proof. A valid wrapper input supplies the selected transition witness and the prior credential proof, or the designated empty proof for issuance. It invokes the same honest NIVC prover on that input, so completeness is preserved. The wrapper verifier checks the public root, terminal type and presentation binding, then invokes the NIVC verifier on exactly x. The extraction experiment forwards the same true statements to the backend simulator and logs those statements. Any fresh wrapper acceptance is therefore a fresh backend acceptance for the same x, and the returned typed lineage verifies all PCC steps. For zero knowledge, replace the backend answers by its simulations of these same statements, keeping the wrapper's public checks and deterministic packaging. Under the auxiliary-input and layout hypotheses this preserves the adversary's view. The wrapper does not derive typed recursive extraction or auxiliary-input zero knowledge from plain soundness. □