

Delegatable Anonymous Credentials from Legacy Credentials using Recursive zk-SNARKs

Leandro Rometsch*

Hamburg University of Technology
Hamburg, Germany
leandro.rometsch@tuhh.de

Sebastian Faust

Technical University of Darmstadt
Darmstadt, Germany
sebastian.faust@tu-darmstadt.de

Philipp-Florens Lehwalder*

Technical University of Darmstadt
Darmstadt, Germany
philipp-florens.lehwalder@tu-darmstadt.de

Stefan Schulte

Technische Universität Berlin
Berlin, Germany
schulte@tu-berlin.de

Abstract

Digital identity systems are becoming increasingly prevalent, driven by regulatory efforts such as the European Digital Identity (EUDI) Wallet. Yet these systems usually do not achieve strong privacy guarantees as offered by anonymous credentials, since they rely on standardized curves and widespread signature schemes like ECDSA that are incompatible with pairing-based primitives underpinning most anonymous credential constructions. Recent work bridges this gap by retrofitting such *legacy credentials* with anonymous-credential properties via zero-knowledge proofs, requiring no issuer-side modifications. However, none of these constructions supports *delegation*: the ability for users to pass on a restricted credential derived from their own to another party, while all parties along the delegation chain maintain full anonymity. Delegatable anonymous credentials (DACs) provide exactly these guarantees, but are likewise incompatible with existing deployments.

We present the first DAC scheme built directly on top of legacy credentials. Our construction requires no issuance infrastructure changes, and yields constant-size credentials independent of delegation depth, with full unlinkability along the chain and selective attribute disclosure at every level. We instantiate it for credentials based on JSON Web Tokens (JWTs) signed with ECDSA using Plonky2 as the recursive proving backend, contributing a secure in-circuit JWT parser that closes vulnerabilities in prior work and a Plonky2 extension for cyclic recursion with per-step zero-knowledge. At 64 attributes, delegation takes 1.2 s and verification only 3.3 ms, with a constant proof size across all delegation levels.

CCS Concepts

• **Security and privacy** → **Privacy-preserving protocols**; *Pseudonymity, anonymity and untraceability; Digital signatures.*

Keywords

delegatable anonymous credentials, legacy credentials, zero-knowledge proofs, recursive zk-SNARKs, digital identity

1 Introduction

Regulatory efforts around the world [36, 67, 68, 78] are driving the adoption of digital identity systems, with the European Digital Identity (EUDI) Wallet [35] scheduled for deployment in late 2026.

In these systems, an issuer issues a credential to a user (the holder), who stores it on-device and presents it to a verifier to prove eligibility to access a service. Such credentials range from government-issued identity documents to employee cards and access-control entitlements, each binding issuer-attested attributes to the user. Privacy is a crucial requirement, as the European eIDAS 2.0 regulation mandates unlinkable credential presentations, ensuring that users cannot be tracked across uses even if the issuer and verifier collude.

In principle, such privacy guarantees can be achieved via anonymous credential (AC) schemes originally proposed by Chaum [28] and later realized by Camenisch and Lysyanskaya [20, 22, 23]. Instead of revealing the original credential, constructions typically have the user prove in zero-knowledge possession of a valid credential, while disclosing only a chosen subset of attributes or predicates over them. Unfortunately, many deployed digital identity systems, including in particular the current specification of the EUDI Wallet, do not achieve these strong privacy guarantees [15]. The main reason for this is that they rely on standardized signature schemes such as ECDSA that are widely available on secure hardware (e.g., for device-binding and standardization compliance), but are incompatible with pairing-based curves used in most AC schemes. Since standardization processes at bodies like NIST and update cycles of hardware security modules have long lead times, deployed systems are unlikely to migrate to native AC schemes requiring different cryptographic primitives in the near future.

To overcome this shortcoming, several recent works [6, 38, 40, 58, 66, 73] have therefore explored retrofitting *legacy credentials* with AC privacy properties via zero-knowledge proofs (ZKPs), in particular zero-knowledge succinct non-interactive arguments of knowledge (zk-SNARKs), over the legacy signature scheme. Crucially, this approach requires no modification to the credential issuance infrastructure and is therefore attractive for improving privacy without system-level changes.

A key feature missing from these constructions but crucial for real-world applications is *delegation*. It arises in any hierarchical setting, such as access control in an organization, where a manager passes a restricted subset of their permissions to a subordinate, who may restrict them further before passing them on. Without delegation, every such step requires contacting the issuer for a new credential. Delegatable anonymous credentials (DACs) allow for such delegation of responsibilities and have been intensively studied for two decades [1, 8, 14, 18, 26, 27, 32, 33, 45, 64, 71]. Beyond

*Both authors contributed equally to this work.

improving operational efficiency, DACs also enhance privacy, as verification does not reveal the identities of intermediate delegators. This is important in organizational hierarchies, where a verifier learning the chain would learn who delegated to whom, which allows identifying the holder’s role, branch, or supervisor even when the credential itself discloses no such attributes [56].

In this work, we present the first DAC scheme derived from legacy credentials, which we showcase for credentials in the form of JSON Web Tokens (JWTs) signed with ECDSA on the P-256 curve, an instantiation widely used in digital identity systems [35, 77]. Our construction keeps credential sizes and costs constant, independent of how often the credential has been delegated, while providing full anonymity for all holders. A verifier checks a presentation in only 3.3 ms, which suits low-latency settings.

At its core, the scheme uses *recursive zk-SNARKs*, a tool that has only recently become practical at the scale our construction requires. A base circuit parses and verifies the legacy credential once; a delegation circuit then recursively verifies the preceding proof and re-delegates the credential with an attribute subvector to the next user. Each delegated credential is itself a succinct ZKP that hides intermediate delegators, ensures unlinkable presentations, and reduces unforgeability to that of the underlying legacy signature scheme and to the soundness of the zk-SNARK scheme.

Contributions. We summarize the key contributions of our work:

- We propose a transformation of legacy credentials into constant-sized DACs with full unlinkability using recursive zk-SNARKs. We instantiate the legacy credential scheme based on JWTs with ECDSA. The implementation includes a windowed JWT parser that extracts variable-length claims, closing parsing vulnerabilities reported in prior work.
- We contribute the recursive-proof engineering that makes our scheme practical and modular. A wrapper circuit lets the delegation and presentation circuits be compiled once and reused across different credential types, while an addition to Plonky2 [69] enables cyclic recursion with per-step zero-knowledge, a property that any protocol releasing intermediate proofs requires.
- Our implementation, based on recursive zk-SNARKs, achieves a delegation time of 1.2 s, a presentation generation time of 0.5 s, a credential size of ≈ 147 KB, and a verification time of only 3.3 ms. These metrics remain constant across delegation levels and for up to 512 attributes.
- Lastly, we propose various extensions to our construction such as extending presentations with predicates or device binding, restricting delegations, or supporting revocation.

1.1 Related Work

Delegatable Anonymous Credentials. DACs, introduced by Chase and Lysyanskaya [27], let a holder derive a credential for another party from their own while possibly adapting the attribute set, and let that delegatee delegate further. Every holder presents anonymously to a verifier, who checks validity against the root issuer alone, and all intermediate delegators remain hidden.

Belenkiy et al. [8] gave the first practical instantiation from randomizable Groth-Sahai proofs [47], an approach shared by later constructions [26, 41], with credential sizes growing linearly per

delegation. Camenisch et al. [18] improved concrete efficiency using structure-preserving signatures, but gave up anonymity during delegation. Constructions based on mercurial signatures [32, 33, 64, 71], an extension of structure-preserving signatures on equivalence classes [48], instead publicly randomize signatures and keys and thereby avoid expensive ZKPs, and Mir et al. [64] removed the linear size growth with their SPSEQ-UC primitive. Schemes over equivalence-class signatures, however, achieve only a weak form of anonymity [7, 45], which allows a delegator to deanonymize their delegates. Griffy et al. [45] recently closed this gap with adversarial public-key class hiding for mercurial signatures, although their construction delegates only signing keys rather than attributes.

Like [14, 71], we adopt the *selective delegation* model, where each delegator can pass on only a subset of their own attributes.

All existing DAC constructions rely on pairing-friendly curves and would therefore require changes to existing issuance infrastructures, whereas our construction operates directly over widely deployed legacy signatures such as ECDSA.

Transforming Legacy Credentials into Anonymous Credentials. Rosenberg et al. [73] (zk-creds) built an AC toolkit over Groth16 [46], supporting complex access criteria and issuance from existing identity documents. They noted that issuance authority is delegatable but gave no formalization or construction, and recursion over Groth16 would be costly. Baldimtsi et al. [6] (zkLogin) also used Groth16, but over OpenID Connect JWTs [55] to authorize blockchain transactions. Their in-circuit parser inspires ours, which adds per-byte and structural constraints on the claim window that close the parsing attacks Celi et al. [25] later found.

Frigo and Shelat [40], Paquin et al. [66] (Crescent), and Kaviani and Setty [58] (Vega) targeted efficient ZKPs for deployed credential formats such as ISO mobile driving license (mDL) [54] and JWTs. Crescent and Vega precompute once over the legacy credential so that each presentation only re-randomizes the preprocessing material and adds an efficient online proof. Vega’s lookup-centric arithmetization reads fields at prover-supplied offsets instead of parsing them in-circuit, which suffices for ISO mDL, whose schema-fixed widths make every extent a circuit constant. A JSON string value instead ends at the first unescaped delimiter, so for JWT payloads the extent must itself be constrained to be minimal over every byte of the value region, which our parser does in-circuit (Section 5.3.2). Frigo and Shelat build on sumcheck [63] and Ligero [2] and require no preprocessing, but yield larger proofs and slower verification. None of these constructions supports efficient recursive proofs, which we require for delegation.

Recursive zk-SNARKs. In recursive SNARK composition [11, 12], a proof verifies another proof by encoding its verifier as a circuit. This is the standard route to incrementally verifiable computation (IVC) [80], where a single succinct proof attests to an arbitrarily long computation, and to its generalization to distributed settings, proof-carrying data (PCD) [31]. Under *cyclic* recursion, a proof verifies proofs of its own instance, so a single fixed circuit serves an arbitrary number of steps. Folding schemes [59–61] reach IVC at a lower per-step prover cost, but their intermediate states are not complete zero-knowledge arguments, which every delegation step of our construction requires and which likewise led Hofmann et al. [50] to rule out folding when aggregating remote-attestation

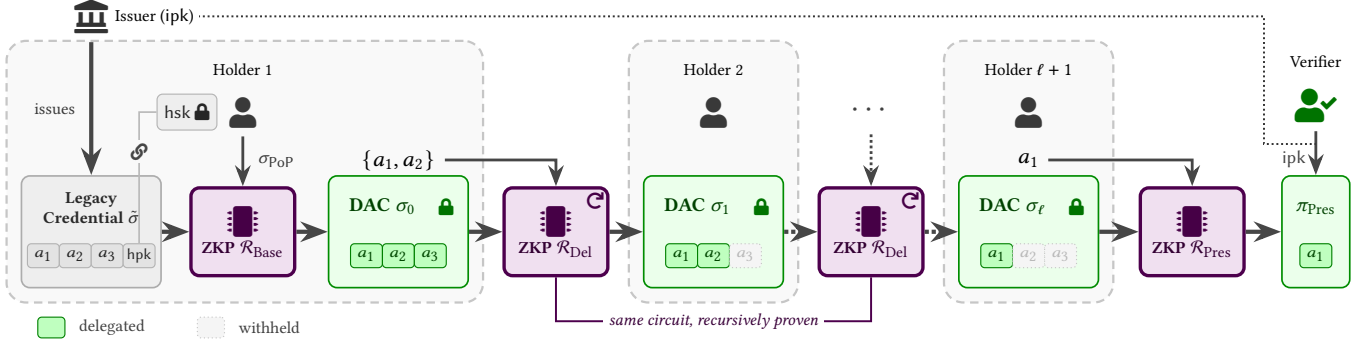


Figure 1: Overview of our construction. A zk-SNARK $\mathcal{R}_{\text{Base}}$ transforms a legacy credential $\tilde{\sigma}$ into a delegatable credential σ_0 ; a recursive delegation relation $\mathcal{R}_{\text{Del}}$ produces each subsequent σ_ℓ while restricting attributes, and $\mathcal{R}_{\text{Pres}}$ enables selective disclosure.

proofs across IoT swarms. We compare recursive proving systems and motivate our choice in Section 5.1.

1.2 Technical Overview

We give a high-level outline of our DAC construction and provide a visual overview in Figure 1. Our construction builds upon a legacy credential scheme Cred that lacks strong privacy guarantees (i.e., multi-show unlinkability) and delegation capabilities, such as JWTs signed with ECDSA. We augment Cred with a recursive ZKP system NIZK and a vector commitment scheme VC to obtain a DAC scheme with full unlinkability and selective-attribute delegation at constant credential size across delegation levels $\ell \leq D$, for a constant D .

Setting. Consider a user holding a legacy credential $\tilde{\sigma}$ issued by an issuer with public issuing key ipk , attesting an attribute vector $\mathbf{a}$ and bound to a user public key hpk . To cover device-bound credentials, the user generates a proof of possession (PoP) $\sigma_{\text{PoP}} \xleftarrow{\$} \text{GenPoP}(\text{hsk}, \text{aux})$ for the secret key hsk corresponding to hpk using a domain separator aux that identifies the conversion into a delegatable credential.

Init delegation ($\ell = 0$). To convert $\tilde{\sigma}$ into the first delegatable credential, the user commits to the attribute vector: $\text{com}_0 \leftarrow \text{VC.Commit}(\mathbf{a})$. It then generates a ZKP π_0 that simultaneously proves (i) knowledge of a valid legacy credential $\tilde{\sigma}$ for $(\text{ipk}, \text{hpk}, \mathbf{a})$ together with a valid PoP σ_{PoP} , and (ii) that com_0 correctly commits to $\mathbf{a}$. The resulting delegatable credential is $\sigma_0 := (\pi_0, \text{com}_0, 0)$.

Delegation ($\ell \rightarrow \ell + 1$). A holder of a level- ℓ credential $\sigma_\ell = (\pi_\ell, \text{com}_\ell, \ell, \ell_{\text{Max}})$ can delegate to another user while restricting attributes to a subvector $\mathbf{a}_{\text{Del}} \subseteq \mathbf{a}$. The delegator computes $\text{com}_{\ell+1} \leftarrow \text{VC.Commit}(\mathbf{a}_{\text{Del}})$ and generates a proof $\pi_{\ell+1}$ that proves: (i) knowledge of a valid level- ℓ proof π_ℓ with respect to ipk and com_ℓ ; (ii) that com_ℓ commits to the attribute vector $\mathbf{a}$; (iii) that $\mathbf{a}_{\text{Del}} \subseteq \mathbf{a}$ and $\text{com}_{\ell+1}$ commits to $\mathbf{a}_{\text{Del}}$; (iv) that the new delegation level is $\ell + 1$; and (v) that its cap satisfies $\ell + 1 \leq \ell'_{\text{Max}} \leq \ell_{\text{Max}}$ and $\ell'_{\text{Max}} \leq D$. The resulting delegated credential is $\sigma_{\ell+1} := (\pi_{\ell+1}, \text{com}_{\ell+1}, \ell + 1, \ell'_{\text{Max}})$.

The delegation relation $\mathcal{R}_{\text{Del}}$ is *recursive*: $\pi_{\ell+1}$ verifies π_ℓ as part of its own proof, which in turn verifies $\pi_{\ell-1}$, and so on. This recursive proof composition ensures that every delegatable credential is a single succinct proof regardless of delegation depth.

Presentation and verification. To present a delegatable credential $\sigma_\ell = (\pi_\ell, \text{com}_\ell, \ell, \ell_{\text{Max}})$ while selectively revealing only the subvector $\mathbf{a}_{\text{Rev}} \subseteq \mathbf{a}$, the user generates a proof π_{Pres} that proves (i) knowledge of a valid level- ℓ proof π_ℓ with respect to ipk and com_ℓ ; (ii) that com_ℓ commits to the attribute vector $\mathbf{a}$; and (iii) that $\mathbf{a}_{\text{Rev}} \subseteq \mathbf{a}$. The verifier accepts if and only if π_{Pres} is valid with respect to ipk , $\mathbf{a}_{\text{Rev}}$, and a fresh verifier-chosen nonce that prevents replaying the presentation. Crucially, the presentation reveals nothing about the delegation chain, the delegation depth, or any undisclosed attributes.

Security. Assuming the simulation extractability of NIZK, unforgeability reduces to the unforgeability and holder binding of Cred and the binding of VC: an adversary producing a valid presentation for attributes not included in any honestly issued or delegated credential breaks one of these three properties. Unlinkability follows from the zero-knowledge property of NIZK: a presentation reveals nothing beyond the disclosed attributes and their validity under ipk , so any two presentations disclosing the same attributes are indistinguishable.

Technical challenges. The straightforward approach of realizing our construction would be to instantiate a set of (recursive) zk-SNARK circuits for each legacy credential type, including per-delegation-level circuits. This is impractical given circuit building costs and distribution effort. Instead, we aim for a single cyclic recursive delegation circuit that serves all credential types and levels. Since each delegation proof represents the DAC handed to the delegatee in zero-knowledge, we enable Plonky2 [69] to support cyclic recursion with blinding. We decouple the credential-specific proving circuit (i.e., the base circuit) from the delegation circuit via a thin wrapper. The wrapper re-proves the base proof under the proof shape expected by the delegation circuit, so a single conditional cyclic recursive verifier inside the delegation (and presentation) circuit uniformly handles the initial step at $\ell = 0$ and every subsequent delegation.

2 Preliminaries

We introduce notation and necessary definitions of the building blocks used in our construction including a legacy credential scheme with its instantiation.

2.1 Notation

We denote vectors by bold-face lowercase letters such as $\mathbf{v}$ and use $\mathbf{v}[i]$ to refer to the i -th element of $\mathbf{v}$. For two vectors $\mathbf{v}_{\text{Sub}}$ and $\mathbf{v}$, we say $\mathbf{v}_{\text{Sub}} \subseteq \mathbf{v}$ (read: $\mathbf{v}_{\text{Sub}}$ is a *subvector* of $\mathbf{v}$) if for all $i \in [n]$, either $\mathbf{v}_{\text{Sub}}[i] = \mathbf{v}[i]$ or $\mathbf{v}_{\text{Sub}}[i] = \perp$, where $\perp$ denotes an empty marker. We write $y \leftarrow A(x)$ to denote that y is assigned the output of algorithm A running on input x and write $y \stackrel{\$}{\leftarrow} A(x)$ if the algorithm is probabilistic. We assume that the security parameter λ and public parameters pp are implicitly given as input to all algorithms. For a polynomial-time verifiable binary relation $\mathcal{R}$, we write $(\phi, w) \in \mathcal{R}$ if the statement ϕ and the witness w satisfy $\mathcal{R}$. We summarize variables used in this work in Table 2 in Appendix A.

2.2 Recursive Zero-Knowledge Proof System

We define a non-interactive ZKP system NIZK for a polynomial-time decidable relation $\mathcal{R}$.

DEFINITION 2.1 (NON-INTERACTIVE ZERO-KNOWLEDGE PROOF SYSTEM). A non-interactive ZKP system NIZK for a relation $\mathcal{R}$ is a tuple of the following algorithms:

- $\text{crs} \stackrel{\$}{\leftarrow} \text{Setup}(1^\lambda, \mathcal{R})$: The setup algorithm generates a common reference string crs for a relation $\mathcal{R}$.
- $\pi \stackrel{\$}{\leftarrow} \text{Prove}(\text{crs}, \phi, w)$: The proving algorithm generates a proof π for a statement ϕ and witness w such that $(\phi, w) \in \mathcal{R}$ using the common reference string crs .
- $0/1 \leftarrow \text{Verify}(\text{crs}, \phi, \pi)$: The verification algorithm outputs a bit indicating the validity of the proof π for the statement ϕ under relation $\mathcal{R}$ and common reference string crs .

Since NIZK.Verify runs in polynomial time, it can be evaluated inside another polynomial-time relation. This allows a proof to establish the validity of an inner proof. We call such a proof system a *recursive* NIZK; when the inner proof uses the same relation $\mathcal{R}$, we call it a *cyclic recursive* NIZK.

We require that a NIZK proof system fulfills *completeness*, *knowledge soundness*, and *zero-knowledge*. Completeness guarantees that honestly generated proofs for $(\phi, w) \in \mathcal{R}$ should verify. Knowledge soundness ensures that no adversary can generate a verifying proof for a statement without knowledge of a corresponding witness satisfying the relation. Formally, a knowledge extractor $\mathcal{E}$ can compute such a witness from any adversary producing a verifying proof. Because the security proof replaces oracle responses with simulated proofs, we require *simulation extractability* [37]; after seeing simulated proofs, an accepting proof for a statement not submitted to the corresponding simulator allows extracting a witness. Our proofs recursively contain inner proofs. We further assume *straight-line extraction* in the ideal global random oracle model (GROM) [30], which does not rewind the adversary and can be applied successively to these inner proofs. Lastly, zero-knowledge ensures that proofs do not leak any information about the witness beyond the validity of the statement. Formally, there exists a simulator Sim that can generate proofs computationally indistinguishable from real proofs without access to the witness.

2.3 Vector Commitment

A vector commitment scheme VC [24] allows committing to a vector of n values in a compact representation. As we open VC

commitments inside our ZKPs by recomputing the deterministic commitment without revealing it, we provide a minimal definition with only a deterministic commitment algorithm.

DEFINITION 2.2 (VECTOR COMMITMENT SCHEME). A vector commitment scheme VC for vectors of length n is defined by the following algorithms:

- $\text{pp} \stackrel{\$}{\leftarrow} \text{Setup}(1^\lambda)$: Generates the public parameters pp .
- $\text{com} \leftarrow \text{Commit}(\text{pp}, \mathbf{v})$: The deterministic commit algorithm outputs a commitment com to the vector $\mathbf{v} = (v_1, \dots, v_n)$.

We require that a vector commitment scheme VC fulfills *binding*, which guarantees that it is computationally infeasible to find two different vectors $\mathbf{v} \neq \mathbf{v}'$ that produce the same commitment $\text{VC.Commit}(\text{pp}, \mathbf{v}) = \text{com} = \text{VC.Commit}(\text{pp}, \mathbf{v}')$.

3 Legacy Credential Scheme

In the following, we define a legacy credential scheme Cred (i.e., a credential scheme without multi-show unlinkability or delegation capabilities) and then show a concrete instantiation using JWTs with ECDSA, which is a widely used credential format in practice.

3.1 System Model and Syntax

In a legacy credential system, we consider three parties: an *issuer*, a *holder*, and a *verifier*. We provide a definition below. The issuer issues legacy credentials that are bound to a holder public key and attest a vector of at most n attributes. Holders obtain credentials from the issuer and can present credentials to a verifier. The verifier receives credential presentations and verifies their validity (e.g., before providing access to a resource).

DEFINITION 3.1 (LEGACY CREDENTIAL SCHEME). We define a legacy credential scheme Cred over attribute vectors of length n as a tuple of the following algorithms:

- $\text{pp} \stackrel{\$}{\leftarrow} \text{Setup}(1^\lambda)$: The setup algorithm generates the public parameters pp .
- $(\text{isk}, \text{ipk}) \stackrel{\$}{\leftarrow} \text{IKGen}(\text{pp})$: Generates an issuer secret key isk and public key ipk .
- $(\text{hsk}, \text{hpk}) \stackrel{\$}{\leftarrow} \text{HKGen}(\text{pp})$: Generates a holder secret key hsk and public key hpk .
- $\tilde{\sigma} \stackrel{\$}{\leftarrow} \text{Issue}(\text{isk}, \text{hpk}, \mathbf{a})$: Generates a credential $\tilde{\sigma}$ on the holder public key hpk and attributes $\mathbf{a} = (a_1, \dots, a_n)$.
- $\sigma_{\text{PoP}} \stackrel{\$}{\leftarrow} \text{GenPoP}(\text{hsk}, \text{aux})$: Generates a proof of possession σ_{PoP} for the holder secret key hsk with auxiliary data aux .
- $0/1 \leftarrow \text{Verify}(\text{ipk}, \tilde{\sigma}, \text{hpk}, \mathbf{a}, \sigma_{\text{PoP}}, \text{aux})$: Verifies the credential $\tilde{\sigma}$ is for the holder public key hpk and attributes $\mathbf{a} = (a_1, \dots, a_n)$ under the issuer public key ipk , and if the proof of possession σ_{PoP} for hpk is valid for auxiliary data aux .

We expect a legacy credential scheme to fulfill *correctness*, *unforgeability*, and *holder binding*. Correctness states that an honestly issued credential $\tilde{\sigma}$ on a holder public key hpk and some attribute vector $\mathbf{a}$ together with an honestly generated PoP σ_{PoP} successfully verifies using the respective holder public key hpk and issuer public key ipk . Unforgeability and holder binding consider a malicious holder and an honest issuer and verifier. Even after querying for credentials, the adversary cannot generate a valid credential $\tilde{\sigma}^*$ for

a pair (hpk, a^*) that was not previously queried. Lastly, holder binding is unforgeability of the PoP under an honestly generated holder key. The challenger generates issuer and holder key pairs and issues a credential $\tilde{\sigma}$ for an adversary-chosen attribute vector a under hpk . The adversary receives $(\text{ipk}, \text{hpk}, \tilde{\sigma}, \text{a})$ and may obtain PoPs for auxiliary inputs of its choice. It wins by outputting $(\sigma_{\text{PoP}}^*, \text{aux}^*)$ such that $\text{Cred.Verify}(\text{ipk}, \tilde{\sigma}, \text{hpk}, \text{a}, \sigma_{\text{PoP}}^*, \text{aux}^*) = 1$ but it never obtained a PoP for aux^* . We require every efficient adversary to win with only negligible probability.

REMARK 3.1. *Some schemes we consider as legacy credentials also support selective disclosure of attributes (e.g., SD-JWT [39]), which Definition 3.1 does not capture. We keep the simpler definition because a presentation still reveals the credential itself, rather than a randomized version or a proof of knowledge of it, and hence remains linkable across presentations.*

3.2 JWT-based ECDSA Credential

We instantiate the legacy credential scheme using JWTs [55], a standardized credential format in which the credential's attributes are encoded in a JSON structure, and consider ECDSA as the signature algorithm.

JWT structure. A JWT consists of three parts. The *header* is a JSON object that specifies the signing algorithm identifier and the token type (e.g., JWT). The *payload* is a JSON object whose key-value pairs are the *claims*, each consisting of a *claim key* and a *claim value*. The credential attributes are encoded as these claims along with *registered* claims, e.g., *iss* specifying the issuer and *iat* the issuance time. The *signature* is computed by the issuer over the encoded header and payload using the algorithm specified in the header. In the compact serialization as defined by RFC 7519 [55], each part is individually Base64url-encoded and the three encoded strings are concatenated with dots:

$\text{B64url}(\text{header}) \parallel . \parallel \text{B64url}(\text{payload}) \parallel . \parallel \text{B64url}(\text{signature}).$

The holder public key is carried in the *confirmation* claim *cnf*, and Listing 1 in Appendix C shows a decoded example.

Signature scheme. In this work, we consider the ECDSA signing algorithm with the P-256 elliptic curve (also known as *secp256r1*) and the SHA-256 hash function corresponding to the JWT algorithm identifier ES256. The P-256 curve is standardized by NIST [29, 65], which mandates it for use across U.S. federal information systems, and its broad hardware support in secure elements makes ES256 the required algorithm in both the ISO/IEC 18013-5 mDL standard [54] and the OpenID4VC High Assurance Interoperability Profile [81], which governs high-security credential ecosystems including the EUDI Wallet.

The signed message is the SHA-256 hash of the concatenation $\text{B64url}(\text{header}) \parallel . \parallel \text{B64url}(\text{payload}).$

Instantiation. We instantiate Cred as JWTcred as follows. Setup outputs the P-256 curve parameters, and both IKeyGen and HKeyGen generate ES256 key pairs. Issue constructs a JWT whose payload carries a as attribute claims and hpk as the confirmation claim, signs the encoded header-payload concatenation with isk , and outputs the compact serialization $\tilde{\sigma}$. GenPoP signs a key-binding JWT whose payload carries aux and the current timestamp with hsk under

ES256. Verify Base64url-decodes both tokens and checks the ECDSA signature of $\tilde{\sigma}$ under ipk and of σ_{PoP} under hpk , that the payload of $\tilde{\sigma}$ matches a with a confirmation claim encoding hpk , and that the payload of σ_{PoP} matches aux .

4 Delegatable Anonymous Credentials from Legacy Credentials

In this section, we first describe the system and threat model we consider for DACs. We then provide our construction of DACs from legacy credentials Π_{DAC} using recursive ZKP systems, and lastly argue why it is secure in our threat model.

4.1 System Model and Syntax

The model of DACs extends the model of legacy credentials (see Section 3.1) by allowing holders to delegate their credentials to others while selectively restricting the delegated attribute vector. Each credential along the *delegation chain* is associated with a delegation level ℓ , where the initial delegatable credential derived from the legacy credential is at level $\ell = 0$ and each delegation increments ℓ by one. Each credential further carries a commitment to its attribute vector; the commitment is handed to the delegatee together with the credential but enters every presentation only as a private witness, so verifiers never learn it. We note that only the legacy credential is bound to a holder public key, thus requiring proving possession of the corresponding secret key for the initial delegation; further delegations carry no such binding. This design choice is justified by the fact that a delegated credential is never presented directly, but only through a proof of knowledge of it, which inherently provides replay protection and reduces the need for explicit holder binding. Nevertheless, in Appendix B we show how to support key-bound delegated credentials.

DEFINITION 4.1 (DELEGATABLE ANONYMOUS CREDENTIALS). *We define a DAC scheme DAC extending a legacy credential scheme Cred (Definition 3.1) with the following algorithms:*

- $\text{pp} \xleftarrow{\$} \text{Setup}(1^\lambda)$: The setup algorithm generates the public parameters pp for the DAC scheme with a fixed constant maximum delegation level $D \geq 2$.
- $\sigma_0 \xleftarrow{\$} \text{InitDel}(\text{ipk}, \tilde{\sigma}, \text{hpk}, \sigma_{\text{PoP}}, \text{aux}, \text{a})$: Initializes a delegatable credential σ_0 under issuer public key ipk from a legacy credential $\tilde{\sigma}$ with attributes a using a proof of possession σ_{PoP} for the secret key corresponding to the holder public key hpk and a domain separator aux .
- $\sigma_{\ell+1} \xleftarrow{\$} \text{Delegate}(\text{ipk}, \sigma_\ell, \text{a}, \text{a}_{\text{Del}}, \ell'_{\text{Max}})$: Delegates credential σ_ℓ under issuer public key ipk with attribute vector a while restricting to the subvector $\text{a}_{\text{Del}} \subseteq \text{a}$ and capping the delegation depth at ℓ'_{Max} , where $\ell + 1 \leq \ell'_{\text{Max}} \leq \ell_{\text{Max}}$; for σ_0 , we set $\ell_{\text{Max}} = D$, and otherwise ℓ_{Max} is the cap carried by σ_ℓ .
- $\pi_{\text{Pres}} \xleftarrow{\$} \text{Present}(\text{ipk}, \sigma_\ell, \text{a}, \text{a}_{\text{Rev}}, N)$: Generates a presentation for credential σ_ℓ under issuer public key ipk with attribute vector a while selectively revealing the subvector $\text{a}_{\text{Rev}} \subseteq \text{a}$, bound to a verifier-chosen nonce N ; it returns $\perp$ if inputs are malformed or do not describe a valid credential.
- $0/1 \leftarrow \text{Verify}(\text{ipk}, \pi_{\text{Pres}}, \text{aux}, N, \text{a}_{\text{Rev}})$: Verifies the presentation π_{Pres} for domain separator aux , the nonce N , and a set of revealed attributes a_{Rev} under the issuer public key ipk .

Analogously to legacy credential schemes (Definition 3.1), we require a DAC scheme to satisfy *correctness* and *unforgeability*, and additionally *unlinkability*. A DAC scheme is correct if all honestly issued credentials are accepted at every delegation level. Unforgeability ensures that no adversary can present attributes that were neither issued for an adversary-chosen holder public key nor explicitly delegated to the adversary, and that the delegation of a holder-bound legacy credential cannot be initialized without a valid PoP. Unlinkability ensures that it is infeasible to link a presentation to the underlying credential, even if the adversary participated in its delegation.

We consider a malicious holder for unforgeability. For unlinkability, we consider a malicious verifier that can also collude with the issuer as well as with holders that participated in the delegation. In both settings, an honest holder's PoP σ_{PoP} remains private and no delegation or presentation reveals it. The holder-binding experiment may give the adversary the legacy credential $\tilde{\sigma}$ itself; a holder that leaks the PoP before the initial delegation step is outside our model. We formalize both properties in Section 4.3.

4.2 Construction

We now describe our DAC construction Π_{DAC} based on a legacy credential scheme Cred using a recursive ZKP system NIZK and a vector commitment scheme VC.

Our scheme allows for selective attribute delegation and presentation; we present extensions to this construction with additional features such as revocation in Appendix B.

4.2.1 Setup. The setup expects the legacy credential scheme Cred to be already set up with public parameters pp_{Cred} including issuer and holder key generation. For the vector commitment scheme, it runs $\text{pp}_{\text{VC}} \xleftarrow{\$} \text{VC.Setup}(1^\lambda)$, and for the ZKP system, it runs $\text{NIZK.Setup}(1^\lambda, \cdot)$ on relations $\mathcal{R}_{\text{Base}}$, $\mathcal{R}_{\text{Del}}$, and $\mathcal{R}_{\text{Pres}}$ to obtain the respective common reference strings crs_{Base} , crs_{Del} , and crs_{Pres} . The maximum delegation level D is a system-wide constant independent of the security parameter and is included in pp .

4.2.2 Init delegation. To transform a legacy credential $\tilde{\sigma}$, bound to a holder public key hpk , to a delegatable credential, the delegator proceeds as follows. It first generates its PoP for the secret key of hpk by computing $\sigma_{\text{PoP}} \xleftarrow{\$} \text{GenPoP}(\text{hsk}, \text{aux})^1$. It then commits to the attributes of its credential $\text{com} \leftarrow \text{VC.Commit}(\text{pp}_{\text{VC}}, \mathbf{a})$ and creates the proof for the base credential relation $\mathcal{R}_{\text{Base}}$ as defined in Relation 4.1 with the statement $\phi := (\text{ipk}, \text{aux}, \text{com}, \ell = 0)$ and witness $w := (\tilde{\sigma}, \text{hpk}, \sigma_{\text{PoP}}, \mathbf{a})$ as $\pi_0 \xleftarrow{\$} \text{NIZK.Prove}(\text{crs}_{\text{Base}}, \phi, w)$. The proof π_0 proves knowledge of a valid legacy credential $\tilde{\sigma}$ under issuer public key ipk for attributes $\mathbf{a}$ and holder public key hpk , together with a valid PoP σ_{PoP} . It further proves that com correctly commits to the attribute vector $\mathbf{a}$ and that the delegation level ℓ is equal to 0. The DAC is then set to $\sigma_0 := (\pi_0, \text{aux}, \text{com}, \ell)$. While com is handed to the delegatee as part of the credential, it acts as a private witness in all subsequent delegations and presentations and is never revealed to a verifier.

¹The domain separator aux is a public constant specific to the DAC scheme; binding the PoP to aux ensures that it attests the holder's intent to delegate this legacy credential and cannot be replayed across protocols.

4.2.3 Delegation. To delegate $\sigma_\ell := (\pi_\ell, \text{aux}, \text{com}_\ell, \ell, \ell_{\text{Max}})$ while restricting the attribute vector to the subvector $\mathbf{a}_{\text{Del}} \subseteq \mathbf{a}$, the delegator proceeds as follows; a level-0 credential σ_0 carries no depth cap yet, so we set $\ell_{\text{Max}} := D$ for it. The delegator first computes the commitment $\text{com}_{\ell'} \leftarrow \text{VC.Commit}(\text{pp}_{\text{VC}}, \mathbf{a}_{\text{Del}})$. It then sets $\ell' \leftarrow \ell + 1$ and chooses the depth cap ℓ'_{Max} of the delegated credential with $\ell' \leq \ell'_{\text{Max}} \leq \ell_{\text{Max}}$ and $\ell'_{\text{Max}} \leq D$: the cap can only be tightened along the chain, and $\ell'_{\text{Max}} = \ell'$ prevents any further re-delegation outright. It computes the proof for the delegation relation $\mathcal{R}_{\text{Del}}$ as defined in Relation 4.2 with the statement $\phi := (\text{ipk}, \text{aux}, \text{com}_{\ell'}, \ell', \ell'_{\text{Max}})$ and witness $w := (\pi_\ell, \text{com}_\ell, \ell, \ell_{\text{Max}}, \mathbf{a}, \mathbf{a}_{\text{Del}})$ as $\pi_{\ell'} \xleftarrow{\$} \text{NIZK.Prove}(\text{crs}_{\text{Del}}, \phi, w)$. The proof $\pi_{\ell'}$ proves knowledge of a valid NIZK proof π_ℓ for the issuer public key ipk , delegation level ℓ , depth cap ℓ_{Max} , and commitment com_ℓ . It further proves that com_ℓ correctly commits to the attribute vector $\mathbf{a}$, that $\text{com}_{\ell'}$ correctly commits to the delegated subvector $\mathbf{a}_{\text{Del}} \subseteq \mathbf{a}$, that the new level is incremented by one, and that the new depth cap satisfies $\ell' \leq \ell'_{\text{Max}} \leq \ell_{\text{Max}}$. It outputs the new delegatable credential $\sigma_{\ell'} := (\pi_{\ell'}, \text{aux}, \text{com}_{\ell'}, \ell', \ell'_{\text{Max}})$ to the delegatee. As VC.Commit is deterministic, the delegatee checks $\text{com}_{\ell'} = \text{VC.Commit}(\text{pp}_{\text{VC}}, \mathbf{a}_{\text{Del}})$ for the subvector it receives together with the credential.

RELATION 4.1 (BASE CREDENTIAL). Relation $\mathcal{R}_{\text{Base}}$ is defined over statements $\phi = (\text{ipk}, \text{aux}, \text{com}, \ell)$ and witnesses $w = (\tilde{\sigma}, \text{hpk}, \sigma_{\text{PoP}}, \mathbf{a})$ as follows:

$$\mathcal{R}_{\text{Base}} = \left\{ \begin{array}{l} ((\text{ipk}, \text{aux}, \text{com}, \ell), (\tilde{\sigma}, \text{hpk}, \sigma_{\text{PoP}}, \mathbf{a})) : \\ \text{Cred.Verify}(\text{ipk}, \tilde{\sigma}, \text{hpk}, \mathbf{a}, \sigma_{\text{PoP}}, \text{aux}) \\ \wedge \text{com} = \text{VC.Commit}(\text{pp}_{\text{VC}}, \mathbf{a}) \\ \wedge \ell = 0 \end{array} \right\}$$

RELATION 4.2 (DELEGATION). Relation $\mathcal{R}_{\text{Del}}$ is defined over statements $\phi = (\text{ipk}, \text{aux}, \text{com}_{\ell'}, \ell', \ell'_{\text{Max}})$ and witnesses $w = (\pi_\ell, \text{com}_\ell, \ell, \ell_{\text{Max}}, \mathbf{a}, \mathbf{a}_{\text{Del}})$ as follows:

$$\mathcal{R}_{\text{Del}} = \left\{ \begin{array}{l} ((\text{ipk}, \text{aux}, \text{com}_{\ell'}, \ell', \ell'_{\text{Max}}), (\pi_\ell, \text{com}_\ell, \ell, \ell_{\text{Max}}, \mathbf{a}, \mathbf{a}_{\text{Del}})) : \\ \left(\begin{array}{l} (\ell = 0 \\ \wedge \text{NIZK.Verify}(\text{crs}_{\text{Base}}, (\text{ipk}, \text{aux}, \text{com}_\ell, \ell), \pi_\ell)) \\ \vee (\ell \neq 0 \\ \wedge \text{NIZK.Verify}(\text{crs}_{\text{Del}}, (\text{ipk}, \text{aux}, \text{com}_\ell, \ell, \ell_{\text{Max}}), \pi_\ell)) \end{array} \right) \\ \wedge \text{com}_\ell = \text{VC.Commit}(\text{pp}_{\text{VC}}, \mathbf{a}) \\ \wedge \text{com}_{\ell'} = \text{VC.Commit}(\text{pp}_{\text{VC}}, \mathbf{a}_{\text{Del}}) \wedge \mathbf{a}_{\text{Del}} \subseteq \mathbf{a} \\ \wedge \ell' = \ell + 1 \wedge \ell' \leq \ell'_{\text{Max}} \leq \ell_{\text{Max}} \wedge \ell'_{\text{Max}} \leq D \end{array} \right\}$$

4.2.4 Presentation and verification. To create a presentation of a credential $\sigma_\ell := (\pi_\ell, \text{aux}, \text{com}_\ell, \ell, \ell_{\text{Max}})$, while selectively revealing the subvector $\mathbf{a}_{\text{Rev}} \subseteq \mathbf{a}$, the holder proceeds as follows. The holder first obtains a fresh nonce N from the verifier, which prevents the presentation from being replayed in another session. As part of the public statement, the nonce is absorbed into the Fiat–Shamir transcript and thereby binds the proof to that session. It then generates a proof for the presentation relation $\mathcal{R}_{\text{Pres}}$ as defined in Relation 4.3 with the statement $\phi := (\text{ipk}, \text{aux}, N, \mathbf{a}_{\text{Rev}})$ and witness $w := (\pi_\ell, \text{com}_\ell, \ell, \ell_{\text{Max}}, \mathbf{a})$ as $\pi_{\text{Pres}} \xleftarrow{\$} \text{NIZK.Prove}(\text{crs}_{\text{Pres}}, \phi, w)$. The presentation proves knowledge of a valid proof π_ℓ under $\mathcal{R}_{\text{Del}}$ for the issuer public key ipk , commitment com_ℓ , level ℓ , and depth cap ℓ_{Max} . It further proves that com_ℓ correctly commits to the attribute vector $\mathbf{a}$, and that the revealed attributes $\mathbf{a}_{\text{Rev}} \subseteq \mathbf{a}$ (subvector). The

values com_ℓ , ℓ , and ℓ_{Max} remain private witnesses and are not revealed to the verifier. As $\mathcal{R}_{\text{Pres}}$ verifies only proofs under $\mathcal{R}_{\text{Del}}$, which has no special case for $\ell = 0$ (omitted for brevity), $\text{Present}(\text{ipk}, \cdot)$ on a level-0 credential first runs $\text{Delegate}(\text{ipk}, \sigma_0, \mathbf{a}, D)$, delegating the full attribute vector to its own holder, and then presents the resulting level-1 credential.

Having received the presentation proof π_{Pres} , the verifier verifies the presentation for the revealed attributes $\mathbf{a}_{\text{Rev}}$ and its chosen nonce N by checking $\text{NIZK.Verify}(\text{crs}_{\text{Pres}}, (\text{ipk}, \text{aux}, N, \mathbf{a}_{\text{Rev}}, \pi_{\text{Pres}})) = 1$; the proof verifies only against the nonce it was produced for.

RELATION 4.3 (PRESENTATION). *Relation $\mathcal{R}_{\text{Pres}}$ is defined over statements $\phi = (\text{ipk}, \text{aux}, N, \mathbf{a}_{\text{Rev}})$ and witnesses $w = (\pi_\ell, \text{com}_\ell, \ell, \ell_{\text{Max}}, \mathbf{a})$ as follows:*

$$\mathcal{R}_{\text{Pres}} = \left\{ \begin{array}{l} ((\text{ipk}, \text{aux}, N, \mathbf{a}_{\text{Rev}}), (\pi_\ell, \text{com}_\ell, \ell, \ell_{\text{Max}}, \mathbf{a})) : \\ \text{NIZK.Verify}(\text{crs}_{\text{Del}}, (\text{ipk}, \text{aux}, \text{com}_\ell, \ell, \ell_{\text{Max}}, \pi_\ell)) = 1 \\ \wedge \text{com}_\ell = \text{VC.Commit}(\text{pp}_{\text{VC}}, \mathbf{a}) \wedge \mathbf{a}_{\text{Rev}} \subseteq \mathbf{a} \end{array} \right\}$$

4.3 Security Analysis

We now formalize unforgeability and unlinkability of Definition 4.1 as game-based definitions and sketch why our construction Π_{DAC} satisfies them under the considered threat model.

Correctness. Correctness of Π_{DAC} follows from the correctness of the underlying legacy credential scheme Cred and the completeness of the ZKP system NIZK . By the correctness of Cred , an honestly generated legacy credential $\tilde{\sigma}$ passes the credential verification, which is part of the relation $\mathcal{R}_{\text{Base}}$ for the initial delegation. By the completeness of NIZK , honestly generated proofs for the relations $\mathcal{R}_{\text{Base}}$, $\mathcal{R}_{\text{Del}}$, and $\mathcal{R}_{\text{Pres}}$ are accepted by the verifier, thus ensuring that presentations of honestly generated delegatable credentials are accepted by the verifier.

DEFINITION 4.2 (UNFORGEABILITY). *Consider the following experiment between a challenger and a stateful adversary $\mathcal{A}$:*

- (1) *The challenger runs $\text{pp} \xleftarrow{\$} \text{Setup}(1^\lambda)$ and $(\text{isk}, \text{ipk}) \xleftarrow{\$} \text{IKGen}(\text{pp})$, fixes the domain separator aux , and gives $(\text{pp}, \text{ipk}, \text{aux})$ to $\mathcal{A}$.*
- (2) *$\mathcal{A}$ may repeatedly query the following oracles:*
 - $\mathcal{O}_{\text{Issue}}(\text{hpk}, \mathbf{a})$: *Issues and outputs a legacy credential on hpk and $\mathbf{a}$ using the issuer key isk and records $\mathbf{a}$.*
 - $\mathcal{O}_{\text{Issue}'}(\mathbf{a})$: *Generates a fresh holder key pair (hsk, hpk) and issues a legacy credential on hpk and $\mathbf{a}$ using the issuer key isk ; it outputs $(\text{hpk}, \tilde{\sigma})$ and records $(\text{hpk}, \mathbf{a}, \tilde{\sigma})$.*
 - $\mathcal{O}_{\text{Del}}(\mathbf{a}, \mathbf{a}_{\text{Del}}, \ell'_{\text{Max}})$: *Issues a legacy credential on a freshly generated holder public key hpk and $\mathbf{a}$, initializes it via InitDel , and outputs $\text{Delegate}(\text{ipk}, \sigma_0, \mathbf{a}, \mathbf{a}_{\text{Del}}, \ell'_{\text{Max}})$; it records $\mathbf{a}_{\text{Del}}$ together with the statement of the returned delegation proof.*
 - $\mathcal{O}_{\text{Pres}}(\mathbf{a}, \mathbf{a}_{\text{Rev}}, N)$: *Issues a legacy credential on a freshly generated holder public key hpk and $\mathbf{a}$, and outputs a presentation revealing $\mathbf{a}_{\text{Rev}}$ for N ; it records $(N, \mathbf{a}_{\text{Rev}})$.*
- (3) *$\mathcal{A}$ outputs a N^* , attributes $\mathbf{a}_{\text{Rev}}^*$, and a presentation π_{Pres}^* .*

$\mathcal{A}$ succeeds if (1) $\text{Verify}(\text{ipk}, \pi_{\text{Pres}}^*, \text{aux}, N^*, \mathbf{a}_{\text{Rev}}^*) = 1$, (2) $\mathbf{a}_{\text{Rev}}^*$ is not a subvector of any $\mathbf{a}$ recorded by $\mathcal{O}_{\text{Issue}}$ or any $\mathbf{a}_{\text{Del}}$ recorded by $\mathcal{O}_{\text{Del}}$, and (3) $(N^*, \mathbf{a}_{\text{Rev}}^*)$ was never recorded by $\mathcal{O}_{\text{Pres}}$.

A DAC scheme is unforgeable if $\Pr[\mathcal{A} \text{ succeeds}]$ is negligible in λ for every efficient adversary $\mathcal{A}$.

Unforgeability of Π_{DAC} . Assuming the simulation extractability of NIZK , we argue that any adversary succeeding in the experiment of Definition 4.2 breaks the unforgeability or holder binding of Cred , or the binding of VC . We first replace the delegation proofs returned by $\mathcal{O}_{\text{Del}}$ and the presentation proofs returned by $\mathcal{O}_{\text{Pres}}$ with simulated proofs, without constructing their hidden inner proofs. Zero-knowledge makes this change indistinguishable from the unforgeability experiment. Simulator queries are tagged by their relation. Thus, condition (3) ensures that the adversary's final $\mathcal{R}_{\text{Pres}}$ statement was not submitted to the simulator, and the straight-line extractor from Section 2.2 extracts its witness from the proof and the global random-oracle transcript. The reduction then follows the extracted chain, as in the PCD literature [12]: if an inner $\mathcal{R}_{\text{Del}}$ statement was submitted to the simulator by $\mathcal{O}_{\text{Del}}$, it stops and uses the corresponding oracle record. Otherwise, it extracts that proof's witness, continuing through the delegation proofs and finally the base proof. Because every extracted $\mathcal{R}_{\text{Del}}$ proof verifies, the constraints $\ell' = \ell + 1$ and $\ell' \leq \ell'_{\text{Max}} \leq \ell_{\text{Max}} \leq D$ hold at each step, so the level strictly decreases from at most D . The chain thus requires one presentation, at most D delegations, and one base extraction, hence at most $D + 2$ calls. This process either encounters a delegation statement recorded by $\mathcal{O}_{\text{Del}}$ or yields a legacy credential $\tilde{\sigma}$ with a PoP σ_{PoP} from $\mathcal{R}_{\text{Base}}$. Writing $\mathbf{a}_i$ for the vector committed by the level- i credential, where $\mathbf{a}_0$ is authenticated by the extracted legacy credential, the binding of VC and the subvector constraints in $\mathcal{R}_{\text{Del}}$ imply $\mathbf{a}_{\text{Rev}}^* \subseteq \mathbf{a}_\ell \subseteq \dots \subseteq \mathbf{a}_1 \subseteq \mathbf{a}_0$. If the extracted chain passes through the commitment of a credential that $\mathcal{O}_{\text{Del}}$ delegated to the adversary, this subvector chain forces $\mathbf{a}_{\text{Rev}}^* \subseteq \mathbf{a}_{\text{Del}}$, contradicting condition (2). Otherwise, consider the holder public key and attribute vector authenticated by the extracted $\tilde{\sigma}$. If this tuple was queried through $\mathcal{O}_{\text{Issue}}$, the same subvector chain contradicts condition (2). If the tuple was never issued, $\tilde{\sigma}$ is a forgery against Cred . The remaining case is that the tuple was returned by $\mathcal{O}_{\text{Issue}'}$. Since the adversary received neither the corresponding holder secret key nor a PoP for aux , the extracted σ_{PoP} breaks the holder binding of Cred . Finally, strong unforgeability of the legacy signature is not needed: mauling a signature the adversary holds leaves the signed holder-key and attribute tuple unchanged, so condition (2) is unaffected. Our model further excludes an honest holder leaking a PoP for the DAC-specific domain separator aux (Section 4.1).

DEFINITION 4.3 (UNLINKABILITY). *Consider the following experiment between a challenger and a stateful adversary $\mathcal{A}$:*

- (1) *The challenger runs $\text{pp} \xleftarrow{\$} \text{Setup}(1^\lambda)$ and gives pp to $\mathcal{A}$.*
- (2) *$\mathcal{A}$ outputs an issuer public key ipk and domain separator aux , after which the challenger chooses $b \xleftarrow{\$} \{0, 1\}$.*
- (3) *$\mathcal{A}$ may repeatedly query the left-or-right presentation oracle $\mathcal{O}_{\text{LR}}^b(N, \mathbf{a}_{\text{Rev}}, (\sigma^0, \mathbf{a}^0), (\sigma^1, \mathbf{a}^1))$. For each $c \in \{0, 1\}$, the oracle independently computes $\pi_{\text{Pres}}^c \xleftarrow{\$} \text{Present}(\text{ipk}, \sigma^c, \mathbf{a}^c, \mathbf{a}_{\text{Rev}}, N)$. It returns $\perp$ if either presentation does not verify under ipk for aux , N , and $\mathbf{a}_{\text{Rev}}$, and otherwise returns π_{Pres}^b .*
- (4) *$\mathcal{A}$ outputs a bit b' and succeeds if $b' = b$.*

A DAC scheme is unlinkable if $\text{Adv}_{\mathcal{A}, \text{DAC}}^{\text{unlink}} := |\Pr[\mathcal{A} \text{ succeeds}] - \frac{1}{2}|$ is negligible in λ for every efficient adversary $\mathcal{A}$.

Unlinkability of Π_{DAC} . We proceed by a hybrid over the left-or-right oracle. In the hybrid, the oracle performs the same validity

check but, whenever it does not return $\perp$, replaces π_{Pres}^b with a simulated NIZK proof for the statement $(\text{ipk}, \text{aux}, \mathbf{N}, \mathbf{a}_{\text{Rev}})$.

Two observations make the hybrid independent of b . First, the oracle returns $\perp$ exactly when π_{Pres}^0 or π_{Pres}^1 fails to verify, which does not depend on b or on the returned proof. Second, when it does not return $\perp$, both presentations verify, so the statement $(\text{ipk}, \text{aux}, \mathbf{N}, \mathbf{a}_{\text{Rev}})$ is true and the simulator applies. Moreover, the statement is determined by the oracle inputs rather than the selected credential and is therefore the same for $b = 0$ and $b = 1$. By zero-knowledge, replacing the polynomially many oracle responses makes the hybrid computationally indistinguishable from the unlinkability experiment. The adversary’s view in the hybrid is independent of b , so it succeeds with probability exactly $\frac{1}{2}$ and $\text{Adv}_{\mathcal{A}, \text{DAC}}^{\text{unlink}}$ is negligible in λ .

5 Implementation

The implementation is itself a primary contribution of this work: zk-SNARKs remain costly enough that naive, purpose-built circuit instantiations sharply limit practicability, yet academic zk-SNARK work underdocuments the constraint-system and compiler-level decisions that determine it [62]. We therefore expose our engineering decisions alongside the per-circuit description of our publicly available implementation² of Π_{DAC} in the following.

Our principal engineering contribution is a windowed JWT parser for secure in-circuit parsing of variable-length claims (Section 5.3, elaborated in Appendix C). Beyond this, we document three contributions. First, a wrapper circuit that enables the use of a *single* cyclic recursive verifier with $\mathcal{R}_{\text{Base}}$ as the recursion anchor and $\mathcal{R}_{\text{Del}}$ at every subsequent level (Section 5.4). Second, an addition to Plonky2 for cyclic recursion *with zero-knowledge* (Section 5.5), a combination not natively supported by the proving system but required by our construction. Third, a chain-binding mechanism that ties each delegation chain to its credential type via a verifier-key digest (Section 5.5).

5.1 Choosing a Proving System

Our Π_{DAC} construction requires a NIZK proving system with two properties. First, *cyclic recursive proof composition*: a single circuit that efficiently verifies proofs of itself, so that one delegation circuit, compiled once, serves all delegation levels and credential types. Under non-cyclic recursion, where proofs can only verify proofs of a different circuit, each level would instead require its own circuit (see Section 5.5.1). Second, *per-step zero-knowledge*: each delegation proof is itself a DAC, produced by the delegator from their own DAC as a private witness and handed to the delegatee, so every proof must individually hide its witness, including the credential attributes of previous levels. We compare recursive proving systems below and motivate our choice of Plonky2 [69].

5.1.1 Recursion in other proving systems. Folding schemes based on Nova [59–61] accumulate a relaxed R1CS instance whose individual steps provide neither zero-knowledge nor correctness; re-randomizing the accumulator [60] yields a ZKP but terminates the chain, as continuing requires the underlying witness in the clear. Halo [16] forwards a non-zero-knowledge accumulator for the same

reason. Halo2 [34] does achieve per-step zero-knowledge through individually blinded proofs, but its recursion over the Pallas/Vesta curve cycle [51] defers verification work along the chain, so accepting a single proof in full costs additional linear-time work, which is impractical where delegation proofs are themselves DACs that must be uniformly verifiable when recursively consumed. Pairing-based schemes such as Groth16 [46] and PLONK variants [3, 42] with KZG commitments [57] require a trusted setup, which we cannot assume when transforming legacy credentials, and verifying an inner proof requires in-circuit pairings; specialized curve cycles render this arithmetic native [11], yet the pairing remains the dominant proving cost [53]. Plonky3 [70] is a toolkit for assembling proof systems rather than one itself and is mainly used for STARK-based zkVMs like SP1 [76].

5.1.2 Plonky2. Plonky2 [69] is the closest fit among existing proving systems. It combines PLONK-style arithmetization [42] with FRI-based polynomial commitments [9], requires no trusted setup, and reduces pairing-free FRI verification to native operations over the Goldilocks field, making in-circuit proof verification comparatively inexpensive. It natively supports cyclic recursion and, independently, zero-knowledge augmentation of individual proofs, but not their combination. The restriction prevents any protocol that continues a recursive chain and releases its intermediate proofs; we close this gap with an addition to Plonky2 in Section 5.5.

Plonky2 circuits are static arithmetic circuits without dynamic branching, with the gate set, circuit size, and public-input count fixed at build time. Compiling a circuit C therefore gives two artifacts. The first is its *shape*, written $\langle C \rangle$ and called `CommonCircuitData` in Plonky2, which fixes the procedure by which proofs of C are verified. The second is a verifier key vk committing to its preprocessed polynomials, which identifies C among all circuits of that shape. Recursion relies on this split: a verification gadget is built against a shape rather than a circuit, so it verifies proofs of that one shape, but accepts them from *any* circuit carrying it, taking the verifier key as a prove-time input instead of a build-time constant. Which circuit produced the inner proof is thus left open by the gadget and has to be constrained separately, an obligation each of our circuits discharges differently (Sections 5.4, 5.5.3 and 5.6). Recursion is *cyclic* when the gadget is built against the shape of its own circuit.

5.2 Overview

We implement our Π_{DAC} construction from Section 4.2 through four Plonky2 circuits.

- C_{Base} implements the base relation $\mathcal{R}_{\text{Base}}$ for JWTs.
- C_{Wrap} re-proves a C_{Base} proof so C_{Del} can verify it as the chain’s initial step.
- C_{Del} implements cyclic recursive delegation for $\mathcal{R}_{\text{Del}}$, generating DACs from a C_{Wrap} or a previous C_{Del} proof.
- C_{Pres} implements the presentation relation $\mathcal{R}_{\text{Pres}}$.

C_{Base} parses the JWT in-circuit and is compiled for one *credential type* $\mathcal{T}$, as is C_{Wrap} ; we write $C_{\text{Base}}^{\mathcal{T}}$ and $C_{\text{Wrap}}^{\mathcal{T}}$ where the type matters. C_{Wrap} confines the dependence on $\mathcal{T}$, so C_{Del} can operate independently. C_{Del} and C_{Pres} are thus built once and distributed, while a new JWT type is added through a fresh pair $(C_{\text{Base}}^{\mathcal{T}}, C_{\text{Wrap}}^{\mathcal{T}})$.

²<https://github.com/AppliedCryptoGroup/DAC-from-legacy-credential>

Throughout this section we adopt JWT terminology and use *claims* for the key-value pairs of the credential payload, covering both holder-specific attributes (e.g., employer, age) and registered claims such as the issuer identifier (iss); accordingly, n denotes the *claim capacity*, the maximum number of claims a circuit is built to process.

Instantiation of building blocks. The recursive proof system NIZK is Plonky2 [69] over the Goldilocks field $\mathbb{F}_p$ with $p = 2^{64} - 2^{32} + 1$, using FRI-based polynomial commitments [9] and Poseidon [44] as the algebraic hash function. The vector commitment scheme VC is a binary Poseidon Merkle tree over $\mathbb{F}_p$ of depth $d = \log_2 n$, where n is the maximum attribute count shared across all supported credential types, and each attribute is a byte string of length at most $L_i^{\max}$ declared in $\mathcal{T}$. The legacy credential scheme Cred is the ES256-signed JWT of Section 3.2, i.e., ECDSA over the P-256 curve with SHA-256.

Security qualification. The generic unforgeability argument in Section 4 relies on the straight-line simulation extractor specified in Section 2.2. Such an extractor is known to exist for an idealized proof system based on BCS [10] and FRI [9]: Under honest-verifier zero-knowledge and knowledge-soundness assumptions of the underlying interactive oracle proof (IOP), the BCS compilation gives the simulation-secure straight-line extraction, we require in the GROM [30]. Corresponding knowledge-soundness results are known for FRI-based IOPs [13]. These assumptions, together with modeling the Poseidon hash function as a (global) random oracle, have not been concretely established for the deployed Plonky2 backend, so we treat it as a heuristic instantiation of the idealized proof system.

5.3 Base Circuit

The base circuit C_{Base} implements Relation 4.1 ($\mathcal{R}_{\text{Base}}$), parameterized at build time by a *credential type* $\mathcal{T}$ that instantiates the JWT parser: it fixes the JWT header, the expected claim keys k_i with their maximum claim value lengths $L_i^{\max}$ for all $i \in [n]$, and the structure *cnf* of the confirmation claim (Listing 1 shows an example employee credential). We use $\mathcal{T}$ informally here and defer a formal definition to Appendix C.

At a high level, C_{Base} verifies the issuer’s signature over the signed JWT bytes, extracts the claims from the decoded payload to build the initial Merkle commitment, and binds the holder’s secret key to the public key recovered from the confirmation claim to realize a PoP. We depict the circuit in Figure 2 and describe each step in the following.

5.3.1 Signature verification and payload decoding. The Base64url-encoded JWT header *hdr* is a build-time constant from $\mathcal{T}$; at prove time the prover supplies the Base64url-encoded payload and the issuer’s signature as witnesses. The circuit concatenates the two, hashes the result with SHA-256, and feeds the digest into an in-circuit ECDSA verifier that constrains the signature against the issuer’s public key, which C_{Base} exposes as a public input. An in-circuit Base64url decoder then derives the decoded payload, the single source of bytes consumed by the subsequent parsing and PoP. Because that payload is derived deterministically from the signed

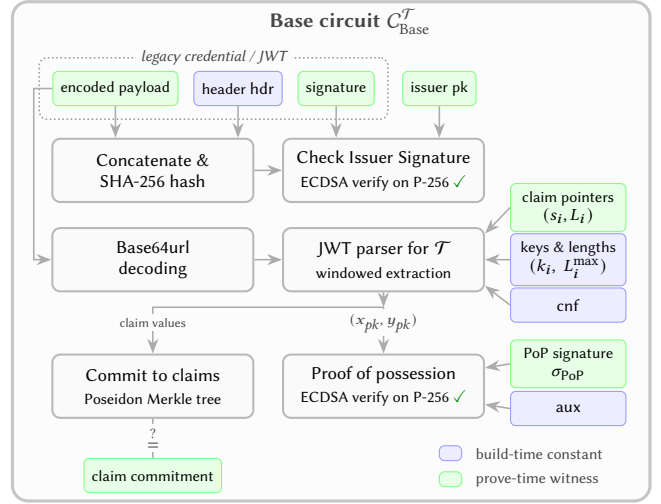


Figure 2: High-level overview of base circuit $C_{\text{Base}}^{\mathcal{T}}$.

bytes, every constraint over its content is implicitly authenticated by the issuer’s signature.

5.3.2 JWT parsing. The JWT parser extracts the credential’s claims and commits them into the Merkle tree of the initial DAC. Sound parsing is critical, as a flawed parser could admit arbitrary attributes, and prior in-circuit JWT parsers have been shown vulnerable to attacks [6, 25]. The difficulty is that claim values are variable-length, so two credentials of the same $\mathcal{T}$ place their claims at different byte offsets, yet arithmetic circuits cannot branch on runtime data: one fixed constraint system must accommodate every valid offset and length. Padding would resolve this but force issuers to re-issue all credentials in a non-standard format, breaking compatibility with deployed JWTs. We instead adopt a *windowed* approach, handling each claim key with a dedicated sub-circuit over a bounded slice of the decoded payload. We sketch the construction and its security intuition here, deferring the formal description to Appendix C.

Build-time vs. prove-time split. At build time, $\mathcal{T}$ fixes the ordered list of expected claim keys k_i , the maximum claim value length $L_i^{\max}$, and the fully concretized JSON template *cnf* of the confirmation claim, and instantiates one sub-circuit per claim key. At prove time, the prover supplies, for each i -th claim, a start offset s_i and a *length witness* encoding the value length L_i as a vector of size $L_i^{\max}$ with a single 1 at position L_i . These pointers align the variable-length payload to the fixed sub-circuits: each (s_i, L_i) selects the byte window that the i -th sub-circuit consumes, which in turn verifies that the window truly delimits the corresponding claim.

Pinning the claim window. A malicious prover could point a window at the wrong region of the payload, for example at key bytes occurring inside another claim’s value. Each sub-circuit therefore enforces, at fixed offsets relative to s_i , a structural anchor byte ({ or ,), an exact match of the embedded key k_i surrounded by quotes, a colon, and a value-opening quote. Because keys are unique within a well-formed JSON object and the anchor restricts

matches to top-level positions, exactly one offset satisfies this pattern; embedding the keys as literal byte constants further rules out the non-canonical key encodings flagged by Celi et al. [25]. The value length is pinned analogously: from the length witness the circuit derives an *active mask* over the content positions and enforces that no active byte equals " " while the byte at position L_i does, placing L_i at the first quote after the value opening, which is the true JSON string boundary assuming the issuer never signs values containing unescaped quotes. The confirmation claim follows the same mechanism with fixed L_i . Appendix C.1 states these constraints in full. Prior in-circuit parsers omit this per-byte check and validate only the first and last byte of the value window [25], so a prover supplying a too-long length can extend the value past its boundary. Each extracted attribute then becomes a leaf of a Poseidon [44] Merkle tree committing to the credential's attributes.

5.3.3 Proof of possession. We instantiate GenPoP as an ECDSA signature on P-256, the same curve as the issuer signature, which lets the holder generate σ_{PoP} through the standard signing interface of a secure element holding hsk ; C_{Base} verifies it in-circuit against the public key (x_{pk}, y_{pk}) parsed from the confirmation claim. The domain separator aux is fixed at build time to a constant identifying the conversion of a legacy credential into a DAC. This ties σ_{PoP} to the conversion context, preventing its replay elsewhere, and lets C_{Base} embed the hash-to-scalar of aux as a non-native constant, sparing a second in-circuit SHA-256.

5.4 Wrapping the Base Proof

A proof of $C_{\text{Base}}^{\mathcal{T}}$ serves as the recursion anchor of the delegation chain. Hence, at $\ell = 0$, C_{Del} must verify a $C_{\text{Base}}^{\mathcal{T}}$ proof and at $\ell > 0$ it must verify one of its own (Relation 4.2). The two carry different shapes $(\langle C_{\text{Base}}^{\mathcal{T}} \rangle, \langle C_{\text{Del}} \rangle)$ that would force C_{Del} to instantiate two expensive verifiers carried throughout the chain (Section 5.1). Critically, C_{Del} would depend on $\mathcal{T}$, preventing reusability (Section 5.5.1) and therefore the universal one-time setup of Π_{DAC} (Section 6.2).

We solve this with a wrapper circuit $C_{\text{Wrap}}^{\mathcal{T}}$ (Figure 3). It verifies a base proof once and aligns its public inputs under $\langle C_{\text{Del}} \rangle$. Anchor and delegation proof now share one shape, so a *single conditional cyclic verifier* suffices: C_{Del} expects a wrapper proof at $\ell = 0$ and one of its own at $\ell > 0$. It thereby becomes independent of the credential type. Since $\text{vk}_{\text{Base}}^{\mathcal{T}}$ is a build-time constant of the wrapper, $\text{vk}_{\text{Wrap}}^{\mathcal{T}}$ commits to it. This binding attests to $\mathcal{T}$ and is carried along the delegation chain (Section 5.5.3).

Shape alignment via co-build. A single cyclic recursive verifier accepts both proofs only if their shapes align, i.e., $\langle C_{\text{Wrap}} \rangle = \langle C_{\text{Del}} \rangle$. Since shapes are fixed at build time, the wrapper and delegation circuit are co-built once, adapting a technique used in zkEVM³, which we apply in three steps: (1) compile C_{Del} to learn $\langle C_{\text{Del}} \rangle$; (2) compile C_{Wrap} with $\langle C_{\text{Del}} \rangle$ as its build target; (3) identify missing gate types in C_{Del} and register them to rebuild C_{Del} .

Registering a gate type adds no constraints and therefore does not affect soundness. Since the gate set only grows, one rebuild

suffices and both circuits end up with the same shape. The co-build is independent of $\mathcal{T}$, which changes only how many gates C_{Base} instantiates, not which types; every further wrapper compiles against the fixed $\langle C_{\text{Del}} \rangle$.

5.5 Cyclic Recursive Delegation

The delegation circuit C_{Del} realizes Relation 4.2 ($\mathcal{R}_{\text{Del}}$) through a single *conditional cyclic recursive verifier*, which verifies a $C_{\text{Wrap}}^{\mathcal{T}}$ proof in the initial step and one of its own in every succeeding one (Figure 3). It further propagates the identifier of that wrapper proof, fixing the credential type of the delegation, and realizes the selective attribute delegation.

5.5.1 Per-level circuits vs. a single cyclic verifier. Implementing $\mathcal{R}_{\text{Del}}$ does not strictly require cyclic recursion: a non-cyclic approach would serve each delegation level ℓ by its own circuit $C_{\text{Del}}^{(\ell)}$, whose recursive verifier is built against the shape of $C_{\text{Del}}^{(\ell-1)}$, and would require one presentation circuit per level, as its verifier must be built against that proof's now level-dependent shape. This takes less engineering effort, but costs one base circuit plus two circuits per level, up to a maximum delegation depth fixed before deployment. Each verifier then holds one presentation verifier key per level and credential type, and each delegator and holder carries prover data for the level it is at. We deem this impractical at the scale of an EUDI deployment, which may involve millions of parties. We opt for cyclic recursion instead: a single C_{Del} , built once during setup, serves every level and, via the wrapper of Section 5.4, every credential type, emitting proofs of one shape under one key vk_{Del} , against which C_{Pres} is likewise built once.

5.5.2 Dummy proofs under zero-knowledge. The conditional verifier carries a fixed proof input for each of its two cases, the wrapper proof and the preceding delegation proof. As the circuit is static, both must be supplied at every step even though only one applies to the current level; the inapplicable one receives a structurally valid *dummy proof* of $\langle C_{\text{Del}} \rangle$. Plonky2 supplies such a placeholder through `dummy_circuit`, which pads a trivial circuit with no-op gates until its degree matches the target, deriving the padding count from a closed-form expression in that degree. The expression ignores blinding gates, so `dummy_circuit` aborts whenever blinding is enabled, which our per-step zero-knowledge requirement (Section 5.1) makes mandatory. No closed form exists: the blinding-gate count depends on the FRI parameters, which depend on the post-blinding degree, which in turn depends on the gate count. We resolve it by extending Plonky2⁴: we re-derive its blinding-gate accounting and use it as an oracle in a binary search over the padding count, selecting the largest candidate whose simulated post-blinding degree matches the target degree. The search terminates in $O(\log N)$ steps and adds negligible cost to circuit setup. Soundness is unaffected, as the conditional verifier nullifies every constraint on the input that does not apply, so a structurally well-formed proof suffices. The fix is independent of our construction: it restores dummy-proof generation for any Plonky2 circuit combining recursion with blinding, a prerequisite for every protocol whose intermediate recursive proofs are themselves released and must hide their witnesses.

³The mechanism follows `add_common_recursion_gates` in `0xPolygonZero/zk_evm/.recursive_verifier.rs#L340`, which fixes the extra gates in advance; our co-build instead discovers them from a build mismatch.

⁴[github.com/.../dummy_circuit.rs#L115-L204](https://github.com/0xPolygonZero/zk_evm/blob/main/dummy_circuit.rs#L115-L204)

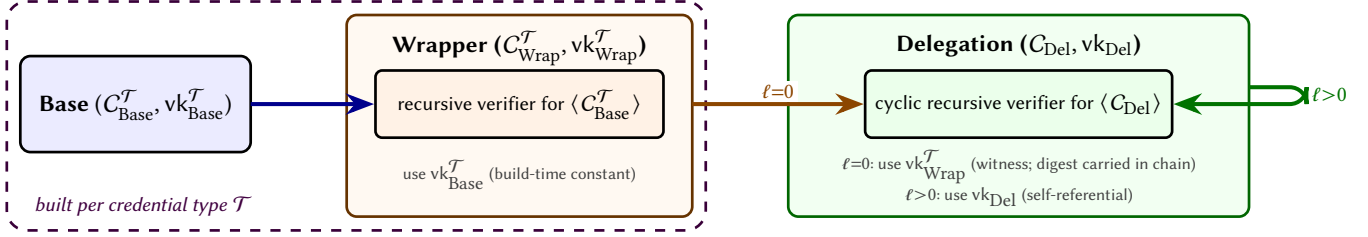


Figure 3: The wrapper $C_{\text{Wrap}}^{\mathcal{T}}$ re-proves a $C_{\text{Base}}^{\mathcal{T}}$ proof for credential type $\mathcal{T}$ under $\langle C_{\text{Del}} \rangle$.

5.5.3 Binding the chain to a base circuit. Because C_{Del} and C_{Pres} are shared across credential types, C_{Del} must expose to which credential type and issuer the anchoring base proof attests. We realize both through a PCD approach [12], carrying two public inputs unchanged through the chain. The first is the wrapper digest $H(\text{vk}_{\text{Wrap}}^{\mathcal{T}})$, pinned at $\ell = 0$ and propagated thereafter (Section 5.4). The second is the issuer public key, exposed by $C_{\text{Base}}^{\mathcal{T}}$ as a public input (Section 5.3.1), which pins the chain to the issuer the legacy credential was verified against. During presentation verification, the verifier supplies the wrapper digest and issuer public key it expects, and verification succeeds only if both match the values carried through the chain. No trusted mapping is needed, as any party can recompute $\text{vk}_{\text{Wrap}}^{\mathcal{T}}$ and its digest.

5.5.4 Selective attribute delegation. The delegator supplies the previous credential’s attributes and a boolean mask with one bit per attribute as private witnesses. Each attribute enters C_{Del} as a Poseidon digest of four field elements, so its witness size does not depend on its byte length. The circuit builds a Merkle tree over these digests and constrains its root to the root carried by the previous proof. This binds the delegator to the previous attribute set. A multiplexer then replaces each masked attribute with the empty leaf, and the new digests form a tree whose root becomes the new commitment value. The delegator can thus only restrict attributes. Uniform leaf width also fixes C_{Del} ’s claim capacity to its slot count, and unused slots hold the empty leaf.

5.6 Presentation

The presentation circuit C_{Pres} realizes the presentation relation $\mathcal{R}_{\text{Pres}}$ (Relation 4.3): it verifies a delegation proof recursively and lets the user disclose an arbitrary subset of the attributes that proof carries. Both the delegation proof and the attribute leaf digests (Section 5.5.4) enter as private witnesses, and C_{Pres} recomputes the Merkle tree in-circuit from the digests and constrains it to equal the root carried by that proof. A private per-slot reveal bit then sets each slot’s public input, constraining a revealed slot to the attribute’s Poseidon leaf digest and an unrevealed one to the empty leaf. In deviation from $\mathcal{R}_{\text{Pres}}$, a disclosed attribute thus appears as its fixed-length leaf digest rather than its raw value, which keeps the disclosed public inputs at a fixed layout regardless of the credential type and lets C_{Pres} , like C_{Del} , be built once for any credential within its claim capacity. In practice the user claims to hold particular attributes, and the verifier confirms this by hashing the presented attributes itself and verifying the proof against the resulting digests. The verifier-chosen nonce of $\mathcal{R}_{\text{Pres}}$ enters C_{Pres} as a public

input that the circuit re-exposes unchanged. Plonky2 hashes all public inputs into the Fiat–Shamir transcript before deriving proof challenges [69], so changing the nonce changes the verification transcript. Our implementation further supports *predicates* as an extension of $\mathcal{R}_{\text{Pres}}$ (Appendix B.1).

Soundness. C_{Pres} verifies the delegation proof against vk_{Del} , embedded as a build-time constant, which pins it to C_{Del} itself rather than to an arbitrary circuit of the same shape. It then exposes the two chain-binding values that proof carries (Section 5.5.3), which the verifier checks against the expected credential type and issuer. Verifying a presentation thus requires only vk_{Pres} , one wrapper digest per accepted credential type, and the issuer public key. A matching C_{Pres} proof therefore attests that the presented DAC derives through $C_{\text{Base}}^{\mathcal{T}}$ from a legacy credential of type $\mathcal{T}$ signed by that issuer.

6 Evaluation

In this section, we evaluate the four circuits from Section 5.2. All benchmarks run on an Apple M4 Pro (12 cores, 48 GB RAM) using Plonky2’s default multithreading, built with Rust 1.92.0. For each configuration, we assume a fixed claim key length of 8 bytes and cap every claim value at $L_i^{\text{max}} = 32$ bytes, varying the claim capacity over $n \in \{8, 16, 32, 64, 128\}$. All reported timings are means over 10 runs. We first report on the circuit sizes, and then present setup cost and runtime benchmarks.

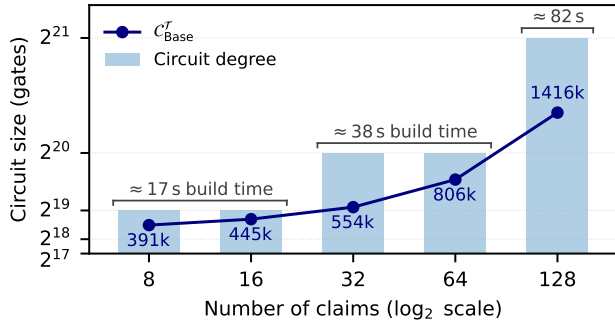
6.1 Circuit sizes

In Plonky2, the cost of building and proving a circuit is dominated by its degree, the smallest power of two that fits its circuit size; crossing a degree boundary roughly doubles the cost for building and proving a circuit. Figure 4 reports the sizes and respective degrees of our circuits for n .

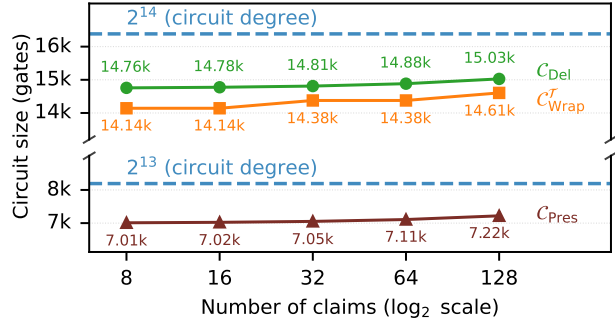
6.1.1 Base circuit. The circuit size of C_{Base} grows linearly in n , since the JWT parser instantiates one sub-circuit per claim while the ECDSA verifier and Base64url decoder contribute a constant base. C_{Base} therefore falls into degree 2^{19} for $n \in \{8, 16\}$, 2^{20} for $n \in \{32, 64\}$, and 2^{21} for $n = 128$ (see Figure 4a).

6.1.2 Wrapper. C_{Wrap} recursively verifies C_{Base} and therefore depends only on the degree of C_{Base} , not on n directly. It thus takes one form per degree class of C_{Base} , i.e., for $n \in \{8, 16\}$, $\{32, 64\}$, and $\{128\}$, and lands within degree 2^{14} in all three cases (Figure 4b).

6.1.3 Delegation and presentation. C_{Del} fits into degree 2^{14} and C_{Pres} into degree 2^{13} for every n we consider (Figure 4b). Their



(a) C_{Base}^T . Bars give the circuit degree; brackets report the build time.



(b) C_{Del} and C_{Wrap}^T fall into circuit degree 2^{14} , C_{Pres} into 2^{13} .

Figure 4: Circuit sizes across the claim capacity.

circuit sizes rise linearly in n , but with a small slope. The dominant cost is the recursive proof verifier, which is independent of n . The linear increase stems from the larger Merkle tree used to commit to the claims, which roughly doubles the number of Poseidon hash computations whenever n doubles. A linear fit to the measured sizes shows that both circuits remain within their degree up to $n = 512$.

6.2 One-time setup

As described in Sections 5.4 and 5.5, C_{Del} and C_{Pres} are built once and serve all credential types and delegation levels, while C_{Base}^T and C_{Wrap}^T are built once per credential type; any well-resourced, non-trusted party can produce and publish these artifacts for reuse. Table 1 reports build times for $n = 64$. As discussed in Section 6.1, the build times of C_{Wrap}^T , C_{Del} , and C_{Pres} remain constant across our evaluated range of n and only C_{Base}^T scales with n , roughly doubling at each degree boundary (Figure 4). Notably, C_{Del} is more expensive to build than C_{Wrap}^T at the same degree because its conditional cyclic recursive verifier requires several internal build iterations to converge to a fixed circuit shape.

6.3 Running cost

The running cost of our scheme splits into three components: (i) the initial delegation, computed once by the legacy credential holder for C_{Base} and C_{Wrap} ; (ii) the per-step cost paid by each delegator for C_{Del} and by the DAC holder for C_{Pres} ; and (iii) the verification of a presentation. Table 1 reports timings for $n = 64$. As established in

Table 1: Build and running costs at claim capacity $n = 64$; $\pm$ reports the relative standard deviation over 10 runs. Shaded cells mark the per-use costs for delegation and presentation.

Circuit	C_{Base}	C_{Wrap}	C_{Del}	C_{Pres}
Degree	2^{20}	2^{14}	2^{14}	2^{13}
Build	$38.0 \text{ s} \pm 0.7\%$	$361 \text{ ms} \pm 0.3\%$	$2.6 \text{ s} \pm 0.3\%$	$183 \text{ ms} \pm 0.4\%$
Prove	$39.2 \text{ s} \pm 1.5\%$	$1.2 \text{ s} \pm 1.0\%$	$1.2 \text{ s} \pm 0.9\%$	$517 \text{ ms} \pm 1.1\%$
Proof size	190.89 KB	146.75 KB	146.75 KB	183.29 KB
Verify	—	—	—	$3.30 \text{ ms} \pm 0.2\%$

Section 6.1, timings track the circuit degree, so these numbers are representative across our n range, with the exception of C_{Base} .

6.3.1 Initial delegation. Transforming a JWT into a DAC requires one C_{Base} proof followed by one C_{Wrap} proof and one C_{Del} proof, totaling 41.6 s at $n = 64$ (Table 1). C_{Base} dominates and scales linearly with its degree. The holder pays this cost only once, e.g., overnight. Subsequent delegations and presentations of the resulting DAC are then independent of the underlying legacy credential.

6.3.2 Delegation and presentation. Delegation (i.e., proving from C_{Del}) takes 1.2 s and presentation (i.e., proving from C_{Pres}) 517 ms at $n = 64$ (Table 1). Following Section 6.1.3, both timings stay constant for much larger choices of n .

6.3.3 Verification. A verifier checks a presentation proof in 3.30 ms (Table 1), independent of n and of the delegation level ℓ . The proofs of C_{Base} , C_{Wrap} , and C_{Del} are never verified individually and only consumed recursively by downstream circuits.

6.4 Discussion

We do not benchmark directly against prior work because no existing DAC operates on legacy credentials (Section 1.1). Native DACs delegate and present in tens of milliseconds [64], well below our per-step cost of about a second, but their costs grow at least linearly with the delegation depth and require issuance over pairing-friendly curves. Among schemes that retrofit legacy credentials, Crescent [66] reports a preprocessing cost of the same order as ours, whereas Frigo and Shelat [40] and Vega [58] report sub-second timings. None supports delegation; adding it through recursion would require predecessor-proof verification, which dominates our per-step cost and may be impractical in their proof systems.

Our construction trades proving time for delegation of legacy credentials and unlinkable presentations while retaining constant per-step cost, constant credential size, and fast verification. Because issuance remains unchanged, it applies to the EUDI Wallet or JWT-based deployments of OAuth and OIDC as currently specified [35, 49, 74]. This prover-verifier asymmetry suits server-hosted AI agents presenting delegated access rights to latency-sensitive web services: subvector delegation limits each agent to least privilege, while unlinkability prevents colluding services from linking its presentations.

Acknowledgments

This work has been funded in part by the German Federal Ministry of Education and Research and the Hessen State Ministry for Higher Education, Research and the Arts within their joint support of the National Research Center for Applied Cybersecurity ATHENE, and by the ERC Consolidator Grant 101044770 (CRYPTOLAYER). Views and opinions expressed are those of the authors only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

Ethical Considerations

This work raises no ethical concerns. We address the four principles of the Menlo Report [4] in turn.

Respect for Persons. This work involves no human subjects, no user studies, and no personal data of any kind. Our implementation uses only synthetically generated credentials with artificial attribute values; no data from real individuals was collected, used, or processed at any stage. The work was therefore not submitted to an external ethics board or institutional review board.

Beneficence. Our work is constructive. We propose a credential scheme that strengthens the privacy guarantees of existing digital identity systems, preventing unauthorized tracking and profiling of their users. We do not attack, exploit, or weaken any deployed system or protocol, and we identify no plausible avenue for misuse.

Justice. Our results are publicly available and the privacy benefits of this work are accessible to all users of digital identity systems, with no selective burden placed on any group.

Respect for Law and Public Interest. This work complies with applicable law and serves the public interest by advancing privacy in widely deployed identity systems. Our implementation builds entirely on open-source software.

Open Science

To facilitate reproducibility and to allow others to build on our work, we make our implementation publicly available⁵. The repository also includes the raw per-run measurements underlying the benchmarks reported in Section 6.

We note that our implementation should be considered a research prototype and has not undergone the security engineering required for production deployment.

References

- [1] Tolga Acar and Lan Nguyen. 2011. Revocation for Delegatable Anonymous Credentials. In *Public Key Cryptography (Lecture Notes in Computer Science)*. Springer, 423–440.
- [2] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkatasubramanian. 2017. Liger: Lightweight Sublinear Arguments Without a Trusted Setup. In *CCS. ACM*, 2087–2104.
- [3] Aztec Labs. 2022. Noir: A Domain-Specific Language for Zero-Knowledge Proofs. <https://github.com/noir-lang/noir>
- [4] Michael D. Bailey, David Dittrich, Erin Kenneally, and Douglas Maughan. 2012. The Menlo Report. *IEEE Secur. Priv.* 10, 2 (2012), 71–75.
- [5] Foteini Baldimtsi, Jan Camenisch, Maria Dubovitskaya, Anna Lysyanskaya, Leonid Reyzin, Kai Samelin, and Sophia Yakubov. 2017. Accumulators with Applications to Anonymity-Preserving Revocation. In *EuroS&P. IEEE*, 301–315.
- [6] Foteini Baldimtsi, Konstantinos Kryptos Chalkias, Yan Ji, Jonas Lindström, Deepak Maram, Ben Riva, Arnab Roy, Mahdi Sedaghat, and Joy Wang. 2024. zkLogin: Privacy-Preserving Blockchain Authentication with Existing Credentials. In *CCS. ACM*, 3182–3196.
- [7] Balthazar Bauer and Georg Fuchsbauer. 2020. Efficient Signatures on Randomizable Ciphertexts. In *SCN (Lecture Notes in Computer Science)*. Springer, 359–381.
- [8] Mira Belenkiy, Jan Camenisch, Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Hovav Shacham. 2009. Randomizable Proofs and Delegatable Anonymous Credentials. In *CRYPTO (Lecture Notes in Computer Science)*. Springer, 108–125.
- [9] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. 2018. Fast Reed-Solomon Interactive Oracle Proofs of Proximity. In *ICALP (LIPIcs)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 14:1–14:17.
- [10] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. 2016. Interactive Oracle Proofs. In *TCC (B2) (Lecture Notes in Computer Science)*. Springer, 31–60.
- [11] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. 2014. Scalable Zero Knowledge via Cycles of Elliptic Curves. In *CRYPTO (2) (Lecture Notes in Computer Science)*. Springer, 276–294.
- [12] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. 2013. Recursive composition and bootstrapping for SNARKS and proof-carrying data. In *STOC. ACM*, 111–120.
- [13] Alexander R. Block, Albert Garreta, Jonathan Katz, Justin Thaler, Pratyush Ranjan Tiwari, and Michał Zając. 2023. Fiat-Shamir Security of FRI and Related SNARKs. *IACR Cryptol. ePrint Arch.* (2023), 1071.
- [14] Johannes Blömer and Jan Bobolz. 2018. Delegatable Attribute-Based Anonymous Credentials from Dynamically Malleable Signatures. In *ACNS (Lecture Notes in Computer Science)*. Springer, 221–239.
- [15] Christian Bormann and Anja Lehmann. 2025. SoK: Anonymous Credentials for Digital Identity Wallets. In *SSR (Lecture Notes in Computer Science, Vol. 16466)*. Springer, 3–25.
- [16] Sean Bowe, Jack Grigg, and Daira Hopwood. 2019. Halo: Recursive Proof Composition without a Trusted Setup. *IACR Cryptol. ePrint Arch.* (2019), 1021.
- [17] Tim Bray. 2017. *The JavaScript Object Notation (JSON) Data Interchange Format*. RFC 8259. Internet Engineering Task Force. <https://doi.org/10.17487/RFC8259>
- [18] Jan Camenisch, Manu Drijvers, and Maria Dubovitskaya. 2017. Practical UC-Secure Delegatable Credentials with Attributes and Their Application to Blockchain. In *CCS. ACM*, 683–699.
- [19] Jan Camenisch, Manu Drijvers, and Jan Hajny. 2016. Scalable Revocation Scheme for Anonymous Credentials Based on n-times Unlinkable Proofs. In *WPES@CCS. ACM*, 123–133.
- [20] Jan Camenisch and Anna Lysyanskaya. 2001. An Efficient System for Non-transferable Anonymous Credentials with Optional Anonymity Revocation. In *EUROCRYPT (Lecture Notes in Computer Science)*. Springer, 93–118.
- [21] Jan Camenisch and Anna Lysyanskaya. 2002. Dynamic Accumulators and Application to Efficient Revocation of Anonymous Credentials. In *CRYPTO (Lecture Notes in Computer Science)*. Springer, 61–76.
- [22] Jan Camenisch and Anna Lysyanskaya. 2002. A Signature Scheme with Efficient Protocols. In *SCN (Lecture Notes in Computer Science)*. Springer, 268–289.
- [23] Jan Camenisch and Anna Lysyanskaya. 2004. Signature Schemes and Anonymous Credentials from Bilinear Maps. In *CRYPTO (Lecture Notes in Computer Science)*. Springer, 56–72.
- [24] Dario Catalano and Dario Fiore. 2013. Vector Commitments and Their Applications. In *PKC (Lecture Notes in Computer Science)*. Springer, 55–72.
- [25] Sofia Celi, Hamed Haddadi, and Kyle Den Hartog. 2026. Analysis and Vulnerabilities in zkLogin. *IACR Cryptol. ePrint Arch.* (2026), 227.
- [26] Melissa Chase, Markulf Kohlweiss, Anna Lysyanskaya, and Sarah Meiklejohn. 2014. Malleable Signatures: New Definitions and Delegatable Anonymous Credentials. In *CSF. IEEE Computer Society*, 199–213.
- [27] Melissa Chase and Anna Lysyanskaya. 2006. On Signatures of Knowledge. In *CRYPTO (Lecture Notes in Computer Science)*. Springer, 78–96.
- [28] David Chaum. 1985. Security Without Identification: Transaction Systems to Make Big Brother Obsolete. *Commun. ACM* 28, 10 (1985), 1030–1044.
- [29] Lily Chen, Dustin Moody, Andrew Regenscheid, Angela Robinson, and Karen Randall. 2023. *Recommendations for Discrete Logarithm-based Cryptography: Elliptic Curve Domain Parameters*. NIST Special Publication 800-186. National Institute of Standards and Technology. <https://doi.org/10.6028/NIST.SP.800-186>
- [30] Alessandro Chiesa and Giacomo Fenzi. 2024. zkSNARKs in the ROM with Unconditional UC-Security. *IACR Cryptol. ePrint Arch.* (2024), 724.
- [31] Alessandro Chiesa and Eran Tromer. 2010. Proof-Carrying Data and Hearsay Arguments from Signature Cards. In *ICS. Tsinghua University Press*, 310–331.
- [32] Elizabeth C. Crites and Anna Lysyanskaya. 2019. Delegatable Anonymous Credentials from Mercurial Signatures. In *CT-RSA (Lecture Notes in Computer Science)*. Springer, 535–555.
- [33] Elizabeth C. Crites and Anna Lysyanskaya. 2021. Mercurial Signatures for Variable-Length Messages. *Proc. Priv. Enhancing Technol.* 2021, 4 (2021), 441–463.
- [34] Electric Coin Company. 2020. The halo2 Proof System. <https://github.com/zcash/halo2>

⁵<https://github.com/AppliedCryptoGroup/DAC-from-legacy-credential>

- [35] European Commission. 2024. The European Digital Identity Wallet Architecture and Reference Framework. <https://github.com/eu-digital-identity-wallet/eudi-doc-architecture-and-reference-framework>
- [36] European Parliament and Council. 2024. Regulation (EU) 2024/1183 amending Regulation (EU) No 910/2014 as regards establishing the European Digital Identity Framework. Official Journal of the European Union L 2024/1183. <https://eur-lex.europa.eu/eli/reg/2024/1183/oj>
- [37] Sebastian Faust, Markulf Kohlweiss, Giorgia Azzurra Marson, and Daniele Venturi. 2012. On the Non-malleability of the Fiat-Shamir Transform. In *INDOCRYPT (Lecture Notes in Computer Science, Vol. 7668)*. Springer, 60–79.
- [38] Armando Faz-Hernández, Watson Ladd, and Deepak Maram. 2021. ZKAttest: Ring and Group Signatures for Existing ECDSA Keys. In *SAC (Lecture Notes in Computer Science)*. Springer, 68–83.
- [39] Daniel Fett, Kristina Yasuda, and Brian Campbell. 2025. *Selective Disclosure for JSON Web Tokens*. RFC 9901. Internet Engineering Task Force. <https://www.rfc-editor.org/rfc/rfc9901>
- [40] Matteo Frigo and Abhi Shelat. 2026. Anonymous Credentials from ECDSA. *IACR Commun. Cryptol.* 3, 1 (2026), 7.
- [41] Georg Fuchsbaauer. 2011. Commuting Signatures and Verifiable Encryption. In *EUROCRYPT (Lecture Notes in Computer Science)*. Springer, 224–245.
- [42] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. 2019. PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge. *IACR Cryptol. ePrint Arch.* (2019), 953.
- [43] Taher El Gamal. 1984. A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms. In *CRYPTO (Lecture Notes in Computer Science)*. Springer, 10–18.
- [44] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. 2021. Poseidon: A New Hash Function for Zero-Knowledge Proof Systems. In *USENIX Security Symposium*. USENIX Association, 519–535.
- [45] Scott Griffy, Anna Lysyanskaya, Omid Mir, Octavio Perez-Kempner, and Daniel Slamanig. 2024. Delegatable Anonymous Credentials from Mercurial Signatures with Stronger Privacy. In *ASIACRYPT (2) (Lecture Notes in Computer Science)*. Springer, 296–325.
- [46] Jens Groth. 2016. On the Size of Pairing-Based Non-interactive Arguments. In *EUROCRYPT (2) (Lecture Notes in Computer Science)*. Springer, 305–326.
- [47] Jens Groth and Amit Sahai. 2008. Efficient Non-interactive Proof Systems for Bilinear Groups. In *EUROCRYPT (Lecture Notes in Computer Science)*. Springer, 415–432.
- [48] Christian Hanser and Daniel Slamanig. 2014. Structure-Preserving Signatures on Equivalence Classes and Their Application to Anonymous Credentials. In *ASIACRYPT (1) (Lecture Notes in Computer Science)*. Springer, 491–511.
- [49] Dick Hardt. 2012. *The OAuth 2.0 Authorization Framework*. RFC 6749. Internet Engineering Task Force. <https://doi.org/10.17487/RFC6749>
- [50] Jonas Hofmann, Philipp-Florens Lehwald, Shahriar Ebrahimi, Parisa Hasanizadeh, and Sebastian Faust. 2026. PIRANHAS: Privacy-Preserving Remote Attestation in Non-Hierarchical Asynchronous Swarms. In *NDSS*. The Internet Society.
- [51] Daira-Emma Hopwood. 2020. The Pasta Curves for Halo 2 and Beyond. <https://electriccoin.co/blog/the-pasta-curves-for-halo-2-and-beyond/>
- [52] Daira-Emma Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. 2025. Zcash Protocol Specification, Version 2025.6.3 [NU6.1]. <https://zips.z.cash/protocol/protocol.pdf>
- [53] Youssef El Housni and Aurore Guillevic. 2022. Families of SNARK-Friendly 2-Chains of Elliptic Curves. In *EUROCRYPT (2) (Lecture Notes in Computer Science)*. Springer, 367–396.
- [54] ISO/IEC JTC 1/SC 17. 2021. *Personal Identification — ISO-Compliant Driving Licence — Part 5: Mobile Driving Licence (mDL) Application*. ISO/IEC Standard 18013-5:2021. International Organization for Standardization. <https://www.iso.org/standard/69084.html>
- [55] Michael B. Jones, John Bradley, and Nat Sakimura. 2015. *JSON Web Token (JWT)*. RFC 7519. Internet Engineering Task Force. <https://doi.org/10.17487/RFC7519>
- [56] Michael B. Jones, Anthony Nadalin, Brian Campbell, John Bradley, and Chuck Mortimore. 2020. *OAuth 2.0 Token Exchange*. RFC 8693. Internet Engineering Task Force. <https://doi.org/10.17487/RFC8693>
- [57] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. 2010. Constant-Size Commitments to Polynomials and Their Applications. In *ASIACRYPT (Lecture Notes in Computer Science)*. Springer, 177–194.
- [58] Darya Kaviani and Srinath Setty. 2026. Vega: Low-Latency Zero-Knowledge Proofs over Existing Credentials. In *S&P*. IEEE, 1951–1969.
- [59] Abhiram Kothapalli and Srinath T. V. Setty. 2022. SuperNova: Proving universal machine executions without universal circuits. *IACR Cryptol. ePrint Arch.* (2022), 1758.
- [60] Abhiram Kothapalli and Srinath T. V. Setty. 2024. HyperNova: Recursive Arguments for Customizable Constraint Systems. In *CRYPTO (10) (Lecture Notes in Computer Science)*. Springer, 345–379.
- [61] Abhiram Kothapalli, Srinath T. V. Setty, and Ioanna Tzialla. 2022. Nova: Recursive Zero-Knowledge Arguments from Folding Schemes. In *CRYPTO (4) (Lecture Notes in Computer Science)*. Springer, 359–388.
- [62] Junkai Liang, Daqi Hu, Pengfei Wu, Yunbo Yang, Qingni Shen, and Zhonghai Wu. 2025. SoK: Understanding zk-SNARKs: The Gap Between Research and Practice. In *USENIX Security Symposium*. USENIX Association, 2085–2104.
- [63] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. 1990. Algebraic Methods for Interactive Proof Systems. In *FOCS*. IEEE Computer Society, 2–10.
- [64] Omid Mir, Daniel Slamanig, Balthazar Bauer, and René Mayrhofer. 2023. Practical Delegatable Anonymous Credentials From Equivalence Class Signatures. *Proc. Priv. Enhancing Technol.* 2023, 3 (2023), 488–513.
- [65] NIST. 2023. Digital Signature Standard (DSS). FIPS PUB 186-5. <https://doi.org/10.6028/NIST.FIPS.186-5>
- [66] Christian Paquin, Guru-Vamsi Policharla, and Greg Zaverucha. 2024. Crescent: Stronger Privacy for Existing Credentials. *IACR Cryptol. ePrint Arch.* (2024), 2013.
- [67] Parliament of Australia. 2024. Digital ID Act 2024. No. 25, 2024 (Cth), Royal Assent 30 May 2024, commenced 30 November 2024. <https://www.legislation.gov.au/C2024A00025/asmade>
- [68] Parliament of the United Kingdom. 2025. Data (Use and Access) Act 2025. 2025 c. 18, Royal Assent 19 June 2025, mostly in force 20 August 2025. <https://www.legislation.gov.uk/ukpga/2025/18>
- [69] Polygon Zero Team. 2022. Plonky2: Fast Recursive Arguments with PLONK and FRI. <https://github.com/0xPolygonZero/plonky2>
- [70] Polygon Zero Team. 2024. Plonky3: A Toolkit for Polynomial IOPs (PIOPs). <https://github.com/PolygonZero/plonky3>
- [71] Colin Putman and Keith M. Martin. 2023. Selective Delegation of Attributes in Mercurial Signature Credentials. In *IMACC (Lecture Notes in Computer Science)*. Springer, 181–196.
- [72] Leandro Rometsch, Philipp-Florens Lehwald, Anh-Tu Hoang, Dominik Kaaser, and Stefan Schulte. 2025. UPPR: Universal Privacy-Preserving Revocation. In *Blockchain*. IEEE, 161–170.
- [73] Michael Rosenberg, Jacob D. White, Christina Garman, and Ian Miers. 2023. zk-creds: Flexible Anonymous Credentials from zkSNARKs and Existing Identity Infrastructure. In *S&P*. IEEE, 790–808.
- [74] Nat Sakimura, John Bradley, Michael B. Jones, Breno de Medeiros, and Chuck Mortimore. 2023. OpenID Connect Core 1.0 incorporating errata set 2. OpenID Foundation specification. https://openid.net/specs/openid-connect-core-1_0.html
- [75] Srinath T. V. Setty, Victor Vu, Nikhil Panpalia, Benjamin Braun, Andrew J. Blumberg, and Michael Walfish. 2012. Taking proof-based verified computation a few steps closer to practicality (extended version). *IACR Cryptol. ePrint Arch.* (2012), 598.
- [76] Succinct Labs. 2025. SP1 zkVM. <https://github.com/succinctlabs/sp1>
- [77] Swiss Confederation. 2026. Swiss Profile for Verifiable Credentials 1.0. swiyu Trust Infrastructure Specification. <https://swiyu-admin-ch.github.io/specifications/swiss-profile-vc/>
- [78] Swiss Federal Assembly. 2024. Federal Act on Electronic Identity Credentials and Other Electronic Credentials (e-ID Act, BGEID). Bundesblatt BBl 2025 20, adopted 20 December 2024, approved by referendum on 28 September 2025. <https://www.fedlex.admin.ch/eli/fga/2025/20/de>
- [79] Justin Thaler. 2022. Proofs, Arguments, and Zero-Knowledge. *Found. Trends Priv. Secur.* 4, 2-4 (2022), 117–660.
- [80] Paul Valiant. 2008. Incrementally Verifiable Computation or Proofs of Knowledge Imply Time/Space Efficiency. In *TCC (Lecture Notes in Computer Science)*. Springer, 1–18.
- [81] Kristina Yasuda, Torsten Lodderstedt, Christian Bormann, and Joseph Heenan. 2025. OpenID4VC High Assurance Interoperability Profile 1.0. OpenID Foundation Specification. https://openid.net/specs/openid4vc-high-assurance-interoperability-profile-1_0.html

A Notation

We summarize the notation used in our work in Table 2.

B Extensions

We keep the DAC construction Π_{DAC} of Section 4 minimal, covering selective attribute delegation and a delegation depth cap, and treat further functionality as an extension. Because Π_{DAC} states delegation and presentation as the relations $\mathcal{R}_{\text{Del}}$ and $\mathcal{R}_{\text{Pres}}$ over a general-purpose recursive proof system, an extension is an additional constraint on what a delegator or a presenter must prove, affecting neither the primitives nor the issuer. DACs that derive their functionality from the algebraic structure of a signature scheme and its randomization [18, 32, 64] instead have to express every

Table 2: A summary of the notation used in this work.

λ	Security parameter	$\mathcal{R}$	Relation
pp	Public parameters	ϕ	Statement
n	Number of attributes	w	Witness
$\mathcal{T}$	Credential type	crs	Common reference string
isk	Issuer secret key	π	Zero-knowledge proof
ipk	Issuer public key	π_ℓ	Credential proof
hsk	Holder secret key	π_{Pres}	Presentation proof
hpk	Holder public key	N	Presentation nonce
$\tilde{\sigma}$	Legacy credential	aux	Domain separator
σ_{PoP}	Proof of possession	$\mathcal{A}$	Adversary
σ_ℓ	Delegatable credential	C	Circuit
ℓ	Delegation level	$\langle C \rangle$	Circuit shape
ℓ_{Max}	Delegation depth cap	vk	Circuit verifier key
$\mathbf{a}$	Attribute vector	C_{Base}	Base circuit
$\mathbf{a}_{\text{Del}}$	Delegated attributes	C_{Wrap}	Wrapper circuit
$\mathbf{a}_{\text{Rev}}$	Revealed attributes	C_{Del}	Delegation circuit
com	Commitment	C_{Pres}	Presentation circuit

added check within that structure. The cost of an extension in our construction is thus the constraint cost of its check inside a proof that already verifies a recursive proof. We sketch the extensions we consider most relevant for deployment in the following.

B.1 Presentation

B.1.1 Predicates. Presentations naturally support predicates over the attributes or delegation levels besides selective disclosure. For instance, holders could prove that their age is above a certain threshold (e.g., 18) without disclosing the exact value, by having the threshold as a public input of $\mathcal{R}_{\text{Pres}}$ and constraining in-circuit that the chosen attribute or level is within the range. Plonky2 supports such range checks natively via bit decomposition, reducing a threshold predicate $v \geq t$ to verifying that $v - t$ fits within the attribute’s bit width at a gate cost linear in the number of bits, which is negligible for attributes such as ages or the level.

B.1.2 Device binding. The construction requires a PoP σ_{PoP} only for the base proof, where the legacy holder’s secret key hsk may reside in a secure element and is only accessible through the GenPoP interface. From the initial delegation onward, the delegated credential carries no binding to a holder public key anymore as possessing a σ_ℓ credential is sufficient to delegate or present. Trust models that demand hardware-bound credentials at every level can extend the construction by introducing a delegatee key pair $(\text{hsk}_\ell, \text{hpk}_\ell)$ at each step, with hsk_ℓ resident in the delegatee’s secure element, and requiring a fresh σ_{PoP} over hsk_ℓ at every presentation. The relation $\mathcal{R}_{\text{Del}}$ would bind σ_ℓ to a given hpk_ℓ , and $\mathcal{R}_{\text{Pres}}$ would verify a corresponding σ_{PoP} , adding a cost of one in-circuit PoP verification per presentation generation.

B.2 Delegation

B.2.1 Expiration. A delegator may want to delegate their credential only for some time, e.g., to grant a contractor access for a single workday. We can add an expiration timestamp t_{Exp} as a public input to $\mathcal{R}_{\text{Del}}$ such that it constrains each subsequent delegation to carry a value no greater than the parent’s, so delegatees cannot extend the lifetime by re-delegating. At presentation, the verifier supplies the current time t_{Now} and $\mathcal{R}_{\text{Pres}}$ enforces $t_{\text{Now}} \leq t_{\text{Exp}}$. Like the depth cap ℓ_{Max} (Section 4.2), t_{Exp} propagates through $\mathcal{R}_{\text{Del}}$ ’s public inputs

but is not re-exposed by $\mathcal{R}_{\text{Pres}}$, preserving the same unlinkability towards the verifier.

B.2.2 Extending attributes. In our construction, delegators can only selectively restrict the attributes to be delegated. Other DAC schemes [8, 18, 27, 64] treat every delegator as an issuer along the delegation chain, enabling them to attest new attributes to the credential. This setting is useful when the root issuer aims to delegate issuance-power to intermediate issuers, e.g., when a department head attests subordinate-specific project assignments that the root issuer never enumerated.

Our construction can be extended to this setting by expressing dedicated policies for changes to the vector commitment com in $\mathcal{R}_{\text{Del}}$ that allow adding new attributes instead of only restricting them to subvectors. For instance, $\mathcal{R}_{\text{Del}}$ could restrict every delegator to extend the set of attributes in com only by a single new attribute per delegation, which would allow delegating one new attribute per level.

B.3 Revocation

Revocation is a desirable property that allows credentials to be invalidated after issuance, for example in the case of credential theft or misuse. While there exists a variety of work on revocation for ACs [5, 19, 21, 72] as well as for DACs [1, 45], we sketch two approaches for how revocation could be integrated into our DAC scheme. Note that the revocation system for the legacy credential may not offer any privacy guarantees; our DAC scheme can nonetheless be extended to support anonymous revocation by introducing a dedicated revocation authority, which is responsible for revocation tasks.

In a first approach with user-specific revocation, following Griffy et al. [45], users register at the revocation authority, which holds a signing key pair $(\text{pk}_{\text{Rev}}, \text{sk}_{\text{Rev}})$, to obtain a re-randomizable revocation token consisting of a signed revocation public key $\text{hp}_{\text{Rev},i}$, while the revocation authority holds the corresponding secret key $\text{hsk}_{\text{Rev},i}$. To revoke a user, the revocation authority adds $\text{hsk}_{\text{Rev},i}$ to a public revocation list. For delegating a credential, the delegator proves in $\mathcal{R}_{\text{Del}}$ that they know a valid signature w.r.t. pk_{Rev} over a revocation public key $\text{hp}_{\text{Rev},i}$ and that the randomized $\text{hp}_{\text{Rev},i}$ is contained in an additional revocation vector commitment com_{Rev} , part of the delegatable credential. During presentation, a user holding a delegated credential of level ℓ additionally reveals all re-randomized revocation public keys $\text{hp}_{\text{Rev},i}$, while proving that these (unrandomized) keys are included in com_{Rev} . The verifier then checks for each $\text{hp}_{\text{Rev},i}$ if there exists a corresponding $\text{hsk}_{\text{Rev},i}$ in the revocation list (requiring an efficient linking between a randomized key and its secret counterpart), which would indicate that one of the users along the delegation chain has been revoked. Likewise, the verifier can request revocation by providing the presented $\text{hp}_{\text{Rev},i}$ to the revocation authority, which can retrieve the corresponding $\text{hsk}_{\text{Rev},i}$ and add it to the revocation list.

A similar revocation approach, without considering users, is that the revocation authority generates a key pair $(\text{hsk}_{\text{Rev}}, \text{hp}_{\text{Rev}})$ of a randomizable encryption scheme such as ElGamal [43] only once for a delegatable credential. To revoke a user, the revocation authority adds hsk_{Rev} to a public revocation list. It then encrypts a public value under hp_{Rev} to obtain a ciphertext ct_{Rev} and provides ct_{Rev}

Listing 1: A decoded JWT signed using ECDSA on P-256.

```
{ "header":{"alg":"ES256","typ":"JWT"},
  "payload":{"
    "iss":"https://server.example.com",
    "iat":"1231006505",
    "employer":"University X",
    "position":"Research Assistant",
    "department":"CS",
    "cnf":{"jwk":{"kty":"EC","crv":"secp256r1",
      "x":"e7Sv...wqFk","y":"78wy...wXhQ"}} },
  "signature":"MEUCIQDr...AIEA..." }
```

along with a signature on it to the initial delegator. For the base proof, the delegator proves in $\mathcal{R}_{\text{Del}}$ that they know a valid signature over ct_{Rev} , which is added as public input to $\mathcal{R}_{\text{Del}}$. For each delegation, the delegator re-randomizes ct_{Rev} to ct'_{Rev} and proves in $\mathcal{R}_{\text{Del}}$ that it honestly re-randomized the previous ciphertext belonging to the previous credential. During presentation, the user again reveals a randomized ciphertext ct'_{Rev} and proves correctness over the re-randomization. The verifier then checks if there exists a hsk_{Rev} in the revocation list that can decrypt ct'_{Rev} to the public value, which would indicate that the credential is revoked. Likewise, the verifier can request revocation by providing the presented ct'_{Rev} to the revocation authority, which retrieves the corresponding hsk_{Rev} and adds it to the revocation list.

C In-Circuit JWT Claim Parsing

A JWT [55] consists of three parts: a *header* specifying the signing algorithm, a *payload* containing the claims, and a *signature*. For ECDSA-based JWTs, the signature is computed over the SHA-256 hash of the Base64url-encoded concatenation of the header and payload. The payload is a JSON object whose key-value pairs are called *claims*; the specification defines registered claims such as *iss* (issuer) and *iat* (issued-at timestamp), while application-specific claims carry the credential's attributes. Listing 1 shows an example JWT used throughout this section.

We treat the decoded payload as bound to the issuer's signature: the base circuit C_{Base} verifies the ECDSA signature over the encoded payload, and an in-circuit Base64url decoder derives the decoded payload from those signed bytes. This appendix focuses on how the circuit extracts individual claims from this payload. Correctly extracting claim values is critical, as all subsequent delegation and presentation logic depends on them. Previous works that parse JWT credentials in-circuit [6] have been shown to be vulnerable to parsing attacks [25], making careful treatment of this step essential.

Extracting claims from a decoded JWT inside an arithmetic circuit is non-trivial for two reasons. First, claim values are typically of variable length: in Listing 1, the claim value of *department* is "CS" (2 bytes), but a different credential of the same type might contain the claim value "PolSci" (6 bytes). The actual length of each claim value is determined at prove time, not when the circuit is compiled. Second, arithmetic circuits have no dynamic branches; every constraint is evaluated regardless of the actual input. A simple workaround would have the issuer pad all claim values to a fixed length so each claim appears at a predetermined offset, but this is

incompatible with already-deployed credentials and the JWT-based ecosystems we target (e.g., EUDI), which we cannot ask issuers to retrofit. Instead, we build a single circuit that securely extracts variable-length claim values without padding for a specific credential type.

Notation. A credential type

$$\mathcal{T} = (\text{hdr}, \{k_0, \dots, k_{n-1}\}, \{L_0^{\max}, \dots, L_{n-1}^{\max}\}, \text{cnf})$$

is defined by a static JWT header *hdr*, an ordered list of n unique claim keys $\{k_0, \dots, k_{n-1}\}$ with corresponding maximum value-region sizes $\{L_0^{\max}, \dots, L_{n-1}^{\max}\}$, and the structure of the confirmation claim *cnf*. The i -th claim key starts at position s_i in the payload, and the closing delimiter of its claim value v_i sits at position L_i within the value region following the opening quote, with $2 \leq L_i \leq L_i^{\max}$. The header *hdr* specifies the credential's signing algorithm, and the confirmation structure *cnf* defines the fixed JSON scaffolding into which the holder's public key coordinates (x_{pk}, y_{pk}) are placed at position s_{cnf} in the payload.

Assumptions. We make minimal trust assumptions about the credential issuer. The issuer only signs credentials adhering to $\mathcal{T}$. Furthermore, it ensures that the JWT payload is well-formed JSON in compact serialization (no whitespace outside claims), that claim values contain no " " characters, and that claim keys are unique across the entire payload, i.e., no claim key k_i recurs as a key inside the confirmation structure *cnf*.

C.1 Per-Claim Extraction

The base circuit is compiled once for a given $\mathcal{T}$: all components of $\mathcal{T}$ become circuit constants, while the payload, claim positions, value lengths, and owner key are supplied as prove-time witnesses. For each claim key k_i , a dedicated *claim extractor* sub-circuit is instantiated at build time (Figure 5). All extractors share the same decoded payload (produced by the in-circuit Base64 decoder) as input, which is in turn constrained to be valid under the credential's ECDSA signature. The following paragraphs describe how each sub-circuit locates the claim, verifies its key, and extracts its variable-length value. For reference, Figure 6 illustrates the concrete example of claim $i = 4$ ("department":"CS") from Listing 1, with $L_4^{\max} = 6$ and $L_4 = 3$ marking the closing quote that delimits the value "CS".

Locating the claim. At build time, the circuit knows each claim key k_i , its byte length $|k_i|$, and the maximum claim value length $L_i^{\max}$ to expect. This fully determines the extraction window for each claim: it spans from s_i to $s_i + |k_i| + 4 + L_i^{\max}$, accounting for the key bytes, four structural characters (opening and closing key quotes, colon, opening value quote), and up to $L_i^{\max}$ bytes for the claim value including its closing delimiter. At prove time, the prover supplies two witnesses per claim: the start offset s_i in the decoded payload, and a length witness encoding L_i . The circuit selects a window of the predetermined size beginning at s_i and applies the constraints described below.

Structural and key verification. Since the claim keys are embedded in the circuit in a fixed order and assumed to be unique, each extractor can verify that the window selected by s_i contains exactly the expected key k_i at the correct structural position. The extractor enforces the following constraints at fixed offsets within the window:

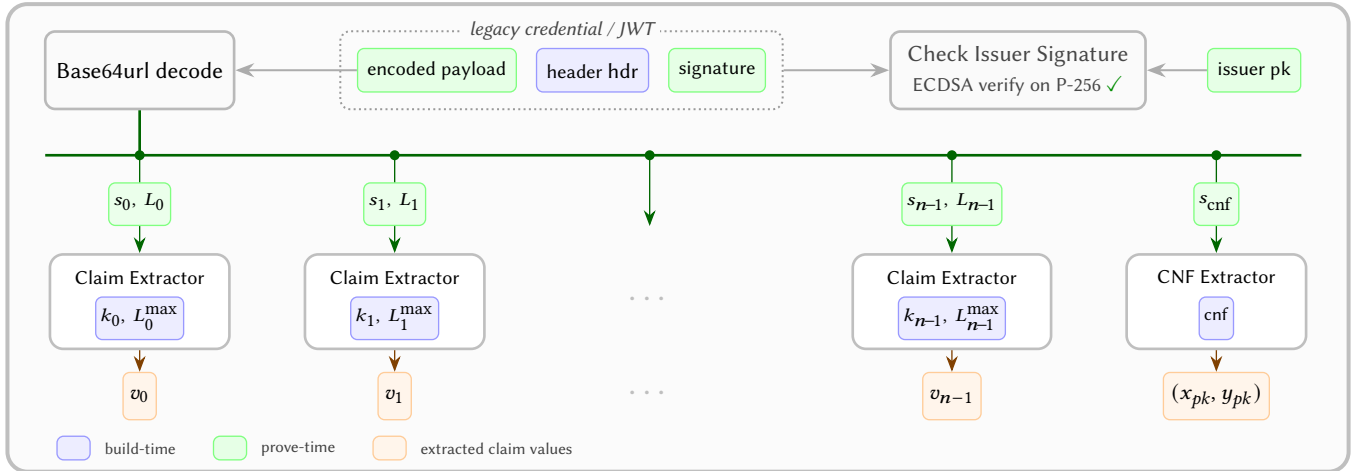


Figure 5: Overview of JWT claim extraction in $C_{\text{Base}}^{\mathcal{T}}$. Blue boxes denote build-time constants embedded in the circuit for a given $\mathcal{T}$; green boxes denote prove-time witnesses supplied by the prover; orange boxes are the extracted outputs. All claim extractors and the CNF extractor share the same decoded payload, which is verified against the ECDSA signature.

- (1) **Structural anchor (position s_i).** The first byte of the window must be `{` (first claim in the JSON object) or `,` (subsequent claim). This prevents a match against key bytes that happen to occur inside another claim value. The anchor alone does not exclude positions inside the nested confirmation structure, whose scaffolding contains the same delimiters; such matches are ruled out by the assumption that no claim key recurs inside cnf.
- (2) **Claim key match (positions s_i+1 through $s_i+|k_i|+3$).** Position s_i+1 must be `"` (opening quote), positions s_i+2 through $s_i+|k_i|+1$ must match the embedded key bytes k_i exactly (so any deviating byte, including a quote, fails this match), position $s_i+|k_i|+2$ must be `"` (closing quote), and position $s_i+|k_i|+3$ must be `:`.
- (3) **Value opening (position $s_i+|k_i|+4$).** Must be `"`, confirming the starting delimiter of the claim value.

Together, these constraints admit exactly one valid start position per claim in any well-formed JSON object whose claim keys are unique and do not recur inside cnf: the prover is forced to set s_i to the unique position where k_i appears as a claim key, preceded by a valid delimiter.

Extracting the variable-length value. The claim value bytes begin at position $s_i+|k_i|+5$ and extend for at most $L_i^{\max}$ positions. Due to the nature of arithmetic circuits, all constraints are evaluated in parallel over a finite field $\mathbb{F}$; there is no conditional branching, forcing us to handle all $L_i^{\max}$ positions regardless of the actual value length L_i . Each position requires a different constraint depending on where it falls relative to L_i : positions before L_i hold content bytes that must not be quote characters ($\neq \text{quote}$), position L_i holds the closing quote ($= \text{quote}$), and positions beyond L_i are unconstrained. Since L_i varies across credentials, every position must be capable of carrying any of these three roles.

Arithmetic-circuit constraints natively express polynomial equalities, whereas a disequality such as $a \neq 0$ must be encoded indirectly [79]. To enforce $a \neq 0$, we apply the standard multiplicative-inverse technique from Setty et al. [75], widely used in practice (e.g., in Zcash [52]): the prover supplies an auxiliary witness b and the circuit enforces $a \cdot b = 1$ (equivalently, $a \cdot b - 1 = 0$). Because every non-zero field element has a multiplicative inverse, the prover can set $b = a^{-1}$ whenever $a \neq 0$. Conversely, if $a = 0$, no choice of b satisfies $0 \cdot b = 1$, so the constraint is unsatisfiable.

Given these two building blocks, the circuit proceeds as follows. The prover supplies a *length witness*: a vector of size $L_i^{\max}$ with a single 1 at position L_i and zeros elsewhere. The circuit verifies that the witness has this form (all entries boolean, exactly one set), and from it derives an *active mask* marking positions 1 through $L_i - 1$ as *active* and the remaining positions as *inactive*. It then enforces two constraints:

- **Non-quote on active bytes.** For each active position j , the circuit requires

$$(v_i[j] - \text{quote}) \cdot \text{inv}_j = 1,$$

where the prover supplies inv_j as the field inverse of the difference. If $v_i[j] = \text{quote}$, the left-hand side is zero regardless of inv_j , making the constraint unsatisfiable. Since the byte values are bound to the signed payload, the prover cannot alter them and must provide the correct inverses to satisfy the constraints. For clarity, we omit these auxiliary inverse witnesses from our figures and the security argument, as they are fully determined by the payload bytes and introduce no prover freedom.

- **Closing quote at boundary.** The byte at position L_i must be `"`.

Active bytes are copied into the output; inactive positions are zeroed. The result is a $L_i^{\max}$ -byte buffer with the $L_i - 1$ content bytes of v_i followed by zeros.

Confirmation claim. The confirmation claim cnf embeds the holder’s public key in a fixed JSON structure defined by cnf . Since the entire structure is known at build time, this amounts to a simplified variant of the extraction circuit above with $L_i = L_i^{\max}$: all structural bytes, including key names and delimiters, are matched as constants, and the two coordinates are extracted at fixed offsets based on s_{cnf} .

C.2 Security Argument

Under the issuer assumptions stated at the beginning of Appendix C, namely that claim keys are unique within the JSON object and that claim values contain no quote characters, we argue that the extracted claim values are exactly those in the issuer-signed JWT payload, leaving the prover no degrees of freedom in the start position or value length of any claim. We first show how the issuer’s signature binds the decoded payload, the header, and the owner’s public key, then argue that the per-claim constraints uniquely determine each claim’s start position and value length within this bound payload. Throughout, we contrast our design with the vulnerabilities Celi et al. [25] identified in zkLogin [6].

Adversary model. The prover of the base circuit C_{Base} supplies its own credential as a witness. Because the circuit must not assume that the prover is honest, every witness value that influences claim extraction must be constrained. The prover controls two quantities per claim: the start position s_i in the decoded payload and the length witness that determines L_i . A malicious prover could point to a wrong position, causing the extractor to read bytes from an unrelated part of the payload, or supply a length witness that selects a shorter or longer value than the true one. The decoded payload bytes, the header, and the owner’s public key are not under the prover’s control either, as we establish next. Similarly, the claim keys are build-time constants embedded in the circuit for a given $\mathcal{T}$.

Header and payload binding. Before arguing about individual claim extraction, the circuit must ensure that the claims originate from the issuer-signed message. Three properties establish this binding. First, the JWT header is constrained to exact constant bytes matching the Base64url encoding of hdr , not merely a size bound. This prevents algorithm-confusion attacks (e.g., $\text{alg}:\text{none}$) and any header manipulation. Second, the decoded payload used for claim extraction is produced by an in-circuit Base64url decoder. The decoder’s input is constrained to match the SHA-256 message bytes that are ECDSA-verified against the issuer’s public key, a public input of the proof. The decoder gadget itself constrains each decoded byte to be the unique decoding of its encoded input via per-character alphabet lookups and per-group bit-packing checks. The decoded payload that all claim extractors and the confirmation-claim extractor share is therefore deterministically fixed by the issuer-signed message. Third, the credential owner’s public key (x_{pk}, y_{pk}) is not provided as a separate witness but extracted in-circuit from the cnf claim, whose surrounding JSON structure is matched byte-for-byte against the constant cnf . The circuit additionally proves knowledge of the secret key corresponding to the extracted public key.

Celi et al. [25] identified cross-impersonation attacks in zkLogin arising from the absence of cryptographic binding between the JWT subject, the credential owner’s key, and the issuer. In our design,

the chain from ECDSA signature to issuer public key, from signed payload to decoded claims, and from the cnf claim to the owner’s key is fully constrained in-circuit. A prover cannot substitute a different owner key without either breaking the cnf structure match or failing the PoP check.

Start position uniqueness. For each claim key k_i , the structural anchor requires the byte at position s_i to be `{` or `,`, and the key match requires positions s_i+1 through $s_i+|k_i|+3$ to equal the quoted key followed by a colon. In a well-formed JSON object with unique keys, this pattern occurs at exactly one position outside the confirmation structure. A match inside a claim value is impossible, as the quoted-key pattern requires `"` bytes that values, by assumption, do not contain; a match inside cnf , the only nested structure, would require a claim key to recur as one of its keys, which the payload-wide key-uniqueness assumption excludes; and any other position fails the delimiter check or the byte comparison. The prover is therefore forced to select the unique correct start position.

Celi et al. [25] showed that zkLogin’s parsing accepts duplicate keys (enabling claim shadowing, where different JSON parsers disagree on which value to use) and non-canonical key encodings such as Unicode escapes (e.g., `\u0069ss` for `iss`). Our design avoids both issues. Because the claim keys are embedded as literal byte constants at build time, any non-canonical encoding simply fails to match. Because claim values contain no quote characters and no claim key recurs inside cnf , key bytes appearing inside a string value or the confirmation structure cannot produce a false match. These are static constraints that hold regardless of the payload content, requiring no assumptions about the prover.

Value length truncation. Suppose a prover claims $L' < L_i$, attempting to extract only a prefix of the actual claim value. The closing-quote check inspects the byte at position L' within the value region, which falls inside the actual value string. Since the issuer guarantees that claim values do not contain `"`, this byte cannot be a quote character, and the closing-quote check fails. The prover therefore cannot truncate the extracted value.

Value length extension. Suppose a prover claims $L' > L_i$, attempting to absorb bytes beyond the true claim value. The active range now includes the real closing quote at position L_i . The non-quote constraint requires every active byte to satisfy $(v_i[j] - \text{"}) \cdot \text{inv}_j = 1$, which is unsatisfiable when $v_i[j] = \text{"}$. Consequently, the real closing quote inside the active range violates this constraint.

This is the constraint that zkLogin lacks. As Celi et al. [25] note, zkLogin’s parsing verifies only that the first and last bytes of each extracted value are quotes, leaving embedded bytes unchecked. Without a per-byte non-quote check, a prover can therefore extend the claimed value length past the true closing quote, absorbing the subsequent delimiter and adjacent claims into a single extracted value. Our multiplicative-inverse-based constraint closes this gap: the closing-quote check and the per-byte non-quote constraint together pin the claimed length to the unique position where a quote character first appears after the value opening, which is exactly the true JSON string boundary.

Discussion and limitations. Our parser handles string-valued top-level claims and a single level of nesting for the cnf structure.

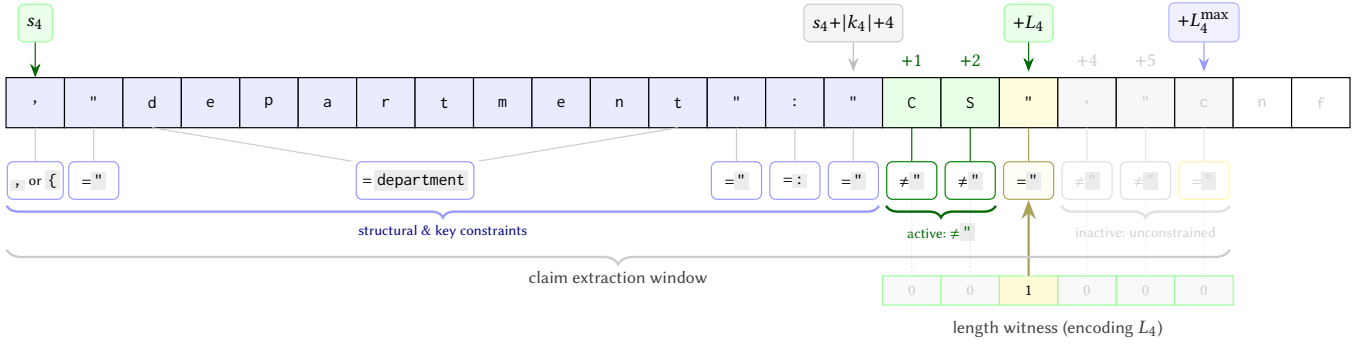


Figure 6: Claim extractor for key `department` ($i = 4$), with $L_4^{\max} = 6$ and $L_4 = 3$ marking the closing quote, starting in the payload at position s_4 . Blue denotes constraints fixed at build time. Green denotes constraints conditionally activated at prove time.

Extending it to JSON numbers, booleans, or more deeply nested objects requires additional per-type extractor variants but follows the same principles.

Within this scope, the security argument rests on two properties of the issuer’s output, stated as assumptions in Appendix C: claim keys are unique across the payload, including the nested confirmation structure, and claim values contain no `"` character. Both are realistic. RFC 8259 [17] recommends key uniqueness and notes that non-unique names yield unpredictable parser behavior. Issuers produce JWTs through standard JSON serializers, which output objects with unique keys by default. Uniqueness across nesting levels goes slightly beyond RFC 8259, which scopes names per object, but holds for the credential types we target, as the keys inside the confirmation structure are fixed JOSE member names (e.g., `jwt`, `kt`, `x`) that do not double as attribute claim keys. The no-quote restriction holds for the claim types our system targets (identifiers, names, timestamps, and Base64url-encoded keys), which do not contain unescaped quotes.

The circuit relies on the issuer to maintain these properties rather than re-enforcing them in-circuit. This matches the trust model of conventional JWT verification, which similarly accepts the issuer’s signature as authority over the payload’s contents, and is reasonable for credential issuers in deployed identity ecosystems. Any tampering with the encoded payload, the header, or the signature is caught by the in-circuit ECDSA check, since every circuit constraint must hold simultaneously. Temporal validation of `iat` or `exp` is not performed in-circuit and is deferred to verifier-side policy, in line with standard JWT verifier practice.

Under these assumptions, the constraints leave the prover with a unique valid assignment: the start position is uniquely determined by the embedded claim key at a top-level JSON position, and the value length is uniquely determined by the first closing quote. The extracted claim values are exactly those in the signed JWT payload.