zkID technical report

Jane Doe[1,2] and John Doe[1]

[1] Institute A, City, Country, jane@institute
[2] Institute B, City, Country, john@institute

Abstract. Main deliveries: 1. Technical report on zk component for the digital id wallet 2. A comparison with current works 3. Applying to EUDI.

Keywords: Anonymous credential · programmable zkp

1 Introduction

It is widely accepted that digital identity systems need to ensure security, work reliably on common devices, and protect user privacy over repeated uses. These goals become much harder to achieve when systems are used in everyday settings, handling different types of platforms, various implementations, and unreliable networks. [KyPRK24]. Within the European Union's European Digital Identity (EUDI) Architecture & Reference Framework [Eur23], these aims are stated explicitly through selective disclosure and unlinkability. Achieving security, reliability, and privacy in digital identity systems requires not only robust cryptographic choices but also practical integration into real-world environments. We consider a typical deployment: wallets run on commodity smartphones, verifiers operate within standard web infrastructures, and network conditions vary widely [Eur23]. In such settings, stable identifiers or protocol patterns can enable unintended correlation of user interactions, compromising privacy. Effective designs must ensure predictable performance under load, minimal resource demands on devices, and low server-side verification costs, while avoiding non-standard cryptographic assumptions that hinder adoption [GGF17]. Our zkID system addresses these challenges by prioritizing unlinkability and deployability, as we explore in subsequent sections.

Security and privacy are distinct requirements in digital identity systems: while authenticity and integrity can be ensured, linkability often persists, compromising user privacy [KyPRK24, FFS88]. We work with the usual roles—issuer, holder, verifier—and with two flows: issuance and presentation. Correlation can arise from deterministic disclosure structures, from identifiers that remain stable (intended or incidental), and from protocol- or transport-level regularities across sessions. Unless policy authorizes correlation, two valid presentations should not be tied back to the same credential; this is the privacy target used here.

Standardized formats such as Selective Disclosure JWT (SD-JWT) [FYC25] and ISO mobile Driving Licence (mDL) [fS21] were designed to minimize disclosed attributes and do so effectively. Even so, disclosure encodings and patterns can be stable enough for cooperating verifiers to correlate presentations over time. Pairing-based credentials (e.g., BBS+[BCTV14]) address unlinkability at the credential layer; however, reliance on pairing-friendly curves complicates certification paths and leaves post-quantum migration unsettled in ecosystems that prioritize NIST-standardized primitives [TZ23].

A third direction wraps standardized credentials in general-purpose zero-knowledge proofs. This can suppress correlating structure, yet it reintroduces a familiar fork: heavier verification without a common reference string, or a universal setup whose governance is

difficult to reconcile with open, multi-party standardization. The question that follows is straightforward to state, although less simple to answer:

Can zero-knowledge be layered over existing, standardized credentials so that unlinkability holds against cooperating verifiers, while practical latency and verifier cost are preserved, and without a universal trusted setup or dependence on non-standard curves?

For a solution to be acceptable, it should satisfy four requirements. First, deployability: it must interoperate with mDL/JWT data structures and existing PKI (ECDSA or RSA) without changes to issuer processes or secure elements. Second, performance: proof generation and verification must remain within mobile and web latency budgets, and expensive steps (e.g., issuer-signature checks) should be cached or batched when possible. Third, unlinkability: distinct presentations of the same credential must be non-correlatable under the stated threat model, including colluding verifiers. Finally, transparency and post-quantum agility: no trusted setup and no pairing-friendly curves; rely on standard assumptions (e.g., discrete logarithm) and allow PQ replacements without redesigning the protocol.

1.1 Our zkID Construction

Motivated by these requirements, we propose *zkID*, a modular zk-SNARK wrapper aimed at real-world identity workflows. At a high level, the design follows a simple rule: pre-process once, reuse later. Proof work is split into two phases:

- **Prepare.** Run infrequently. The issuer's ECDSA signature is verified, credential attributes are parsed from SD-JWT (or related formats), and SHA-256 hashes are computed over intended disclosures. The results are committed with Hyrax vector commitments and can be cached per credential [WTas+17].
- **Show.** Run per presentation: selected attributes or predicates are disclosed, the device-bound nonce signature is validated, and equality is proved against cached commitments from the *Prepare* phase.

Our construction leverages Spartan interactive-oracle proofs over the Tom256 elliptic curve, aligning with ECDSA infrastructures (e.g., NIST P-256) and using Hyrax commitments for efficient linking [Set20, WTas+17]. Our evaluation considers a standard adversary model with malicious holders, colluding verifiers, and honest-but-curious issuers, deferring quantum adversaries and side-channel attacks to future work. To highlight differences in zkID's approach, we review related privacy-preserving credential systems, comparing their strategies for unlinkability and deployability with our solution [Fas24, PPZ24].

1.2 Related Work

Anonymous Credentials from ECDSA. Matteo Frigo and Abhi Shelat (Google) propose an anonymous credential scheme for legacy-deployed ECDSA that preserves deployability on existing infrastructures (e.g., P-256, SHA-256) without issuer or device changes and without trusted setup, addressing reluctance to upgrade infrastructures that only support RSA/ECDSA [Fas24]. Their design tackles the bottleneck of zero-knowledge proofs over non-NTT-friendly curves by building a proof stack around sum-check and the Ligero argument [AHIV17], introducing specialized ECDSA circuits and efficient Reed–Solomon encodings, and adding a witness-consistency mechanism to keep common witness values aligned across arithmetic over $\mathbb{F}_p$ (ECDSA) and GF(2^k) (SHA-256). As reported, the system generates a proof for a single ECDSA signature in 60 ms and a zero-knowledge proof for the ISO/IEC 18013-5 mDL presentation flow in 1.2 s on mobile devices [Fas24, §5.3,§6.2]. Device binding follows the mDL live-challenge pattern. The main trade-off

is comparatively larger proofs and higher verifier workload than succinct CRS-based SNARKs.

Crescent Credentials. Christian Paquin, Guru-Vamsi Policharla, and Greg Zaverucha introduce Crescent to upgrade privacy of existing credentials (JWTs, mDL) with selective disclosure and unlinkability without changing issuance processes [PPZ24]. Crescent achieves fast online presentations with compact proofs, while relying on a two-phase workflow:

- Prepare (offline, once per credential): verify issuer signatures, decode and parse the credential into attributes, and produce a Pedersen vector commitment via a Groth16 proof. Reported timings are approximately 27 s for a 2 KB JWT and 140 s for an mDL, with universal-setup parameters of size approximately 661 MB–1.1 GB [PPZ24, §4].
- Show (online, per presentation): re-randomize the Groth16 proof and attach a few proofs over committed attributes; typical latencies are approximately 22 ms for JWT and 41.2 ms for mDL, with proofs around 1 KB. Optional device binding increases these costs (e.g., about 315 ms Show, about 184 ms verify; proof size about 15 KB) [PPZ24, §4].

Crescent exposes a modular committed-attribute interface that admits sub-provers for range checks, credential linking, and binding to session context or device by proving knowledge of a signature under a committed public key. The principal trade-offs are reliance on pairing-based Groth16 with a trusted setup and large universal parameters, and a comparatively heavy (but amortized) offline Prepare phase; in its current form Crescent is not post-quantum secure [Gro16].

zkID. zkID is a modular, transparent (no-CRS) zero-knowledge wrapper for standardized credentials, prioritizing unlinkability and deployability. We adopt the same two-phase workflow as Crescent but avoid a universal setup, and we retain ECDSA compatibility as in Google's design while moving issuer verification to the offline phase to reduce online verifier work. Key distinctions from prior work include:

- Transparency and PQ agility: no trusted setup; standard discrete-log assumptions; an architecture designed to swap in PQ vector commitments in future instantiations.
- Deployment compatibility: interoperates with mDL/SD-JWT and existing PKI (ECDSA or RSA) without changes to issuer processes or secure elements.
- Reduced online burden: relative to prior ECDSA-based designs, issuer-signature verification and parsing move to the offline Prepare phase, reducing online compute.
- Avoidance of pairing-based cryptography: unlike pairing-based schemes, zkID avoids pairing-friendly curves.
- Verifier cost: reduces verifier work relative to prior ECDSA-based designs under our constraints while preserving unlinkability under the stated threat model.

In summary, zkID shows that a modular, standards-aligned zero-knowledge wrapper over existing credential formats can be built. Compared to pairing-based schemes, it avoids dependence on pairing-friendly curves; compared to universal-setup SNARK wrappers, it avoids a trusted setup; and relative to prior ECDSA-based designs, it reduces verifier cost while preserving unlinkability under our model. The construction is transparent and PQC-agile; the current instantiation is classical and not quantum-resistant.

In our evaluation, zkID achieves practical performance on commodity hardware. For a typical credential with ..., the one-time *Prepare* phase completes in ... seconds on a

standard laptop (or mobile) (..., ...GB RAM), while the per-presentation *Show* phase takes less than ... ms on a mid-range smartphone. Proof sizes are approximately ... KB for the full protocol, and verifier time remains below ... ms on a web server. These results demonstrate that zkID meets latency and resource targets for mobile and web deployments, while preserving unlinkability and avoiding trusted setup. Detailed benchmark results and further breakdowns will be provided in Section 4.

Several aspects are left out of scope. We do not provide comprehensive revocation, explicit defenses against hardware side channels, or end-to-end wallet-integration benchmarks, and a concrete instantiation of lattice-based vector commitments is deferred to future work.

2 Preliminaries - WIP

Notation. For $n \in \mathbb{N}$, let $[n] = \{1, \ldots, n\}$. Vectors are in bold, e.g., $\mathbf{a} = (a_1, \ldots, a_\ell)$. Concatenation is $\|$. For a (possibly randomized) algorithm Alg, we write $y \leftarrow \mathsf{Alg}(x)$ for its output. The security parameter is λ, and $\mathrm{negl}(\lambda)$ denotes a negligible function.

2.1 Algebraic setting and basic primitives

Let $\mathbb{F}$ be a prime field of order q and $\mathbb{G}$ a cyclic group of order q with generator g, where discrete logarithms are hard. We use SHA-256 as $\mathsf{H} : \{0,1\}^* \to \{0,1\}^{256}$. The issuer uses a digital signature scheme $\mathsf{Sig}_I = (\mathsf{KeyGen}_I, \mathsf{Sign}_I, \mathsf{Verify}_I)$ (e.g., ECDSA/RSA), and the device uses Sig_D for nonce-bound session signatures; both are assumed EUF-CMA.

Pedersen vector commitments. Fix generators $\mathbf{g} = (g_1, \ldots, g_\ell)$ and h in $\mathbb{G}$. For $\mathbf{a} \in \mathbb{F}^\ell$ and $r \in \mathbb{F}$ define

$$\mathsf{Com}(\mathbf{a}; r) = \prod_{i=1}^{\ell} g_i^{a_i} \cdot h^r \in \mathbb{G}.$$

Com is perfectly hiding and binding under discrete-log hardness. We will need efficient *openings* and *equality proofs* for Pedersen commitments; we realize these via inner-product arguments (IPAs) in the Hyrax framework [WTs+18].

Hyrax commitments. Hyrax commitments [WTs+18] allow us to commit to a multilinear extension $\widetilde{P}$ of a function $P : \{0,1\}^n \to \mathbb{F}$. We can express $\widetilde{P}(x_1, \ldots, x_n)$ as a vector-matrix-vector product $\vec{v_L} P \vec{v_R}$, where:

- P is a square matrix of evaluations of P on the $\{0,1\}^n$ hypercube, represented as $\langle \vec{p_i} \rangle_{i \in 2^{n/2}}$ where $\vec{p_i} \in \mathbb{F}^{2^{n/2}}$ are rows of P
- $\vec{v_L}(x_1, \ldots, x_{n/2}) = \langle \widetilde{\chi}_{(b_1,\ldots,b_{n/2})}(x_1, \ldots, x_{n/2}) \rangle_{(b_1,\ldots,b_{n/2}) \in \{0,1\}^{n/2}}$, and
- $\vec{v_R}(x_{n/2+1}, \ldots, x_n) = \langle \widetilde{\chi}_{(b_{n/2+1},\ldots,b_n)}(x_{n/2}, \ldots, x_n) \rangle_{(b_{n/2},\ldots,b_n) \in \{0,1\}^{n/2}}$,

and $\widetilde{\chi}_{b_1,\ldots,b_m}(x_1, \ldots, x_m) = \prod_{i \in [m]} \widetilde{\chi}_{b_i}(x_i)$, where $\widetilde{\chi_{b_i}}(x_i)$ is the multilinear extension of the function $\chi_{b_i}(x_i) : \{0,1\} \to \mathbb{F}$ given by $\chi_{b_i}(x_i) = x_i b_i + (1 - x_i)(1 - b_i)$ which equals 1 if $x_i = b_i$ and 0 otherwise.

Then our Hyrax commitment scheme is given by:

- $\mathbf{Setup}(1^\lambda, n) = (g_i)_{i \in 2^{n/2}}$ where $g_i \in \mathbb{G}$ are elements of the elliptic curve group $\mathbb{G}$ over which we compute our Pedersen commitments.
- $\mathbf{Commit}_h(pp, P) \to (C, S)$, where $C = \{C_i\}_{i \in 2^{n/2}}$ and $S = \{S_i\}_{i \in 2^{n/2}}$, and $(C_i, S_i) \leftarrow Commit_p(pp, \langle \vec{p_i} \rangle)$ where $Commit_p$ is the Pedersen vector commitment scheme.

- **Eval**$_h(pp, C, r, v, n; P, S)$, where r is the opening point and v is the claimed evaluation of $P(r)$, is given by:
  - Prover and Verifier compute $\vec{v_L} = \vec{v_L}(r_1, \ldots, r_{n/2})$, $\vec{v_R} = \vec{v_R}(r_{n/2+1}, \ldots, r_n)$, and $c = \sum_{i \in n/2} \vec{v_L}_i C_i$ (where addition is over $\mathbb{G}$)
  - Prover and Verifir engage in an interactive IPA protocol to prove that $P(r)$ is a dot product of v_R and c

2.2 Zero-knowledge proving interface

We use a *transparent* general-purpose SNARK in the Spartan family [Set20], instantiated over $\mathbb{F}$ for R1CS instances. Algorithms are

$$\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda), \quad \pi \leftarrow \mathsf{Prove}(\mathsf{pp}, x, w), \quad b \leftarrow \mathsf{Verify}(\mathsf{pp}, x, \pi) \in \{0, 1\},$$

where x is public input and w the witness. (We reference succinct CRS-based systems such as Groth16 [Gro16] later only for performance comparisons.)

R1CS instance. An *R1CS instance* is a tuple $(\mathbb{F}, A, B, C, io, n, m)$, where io denotes the public input and output of the instance, $A, B, C \in \mathbb{F}^{n \times n}$, where $m \geq |io| + 1$ and there are at most m non-zero entries in each matrix. An instance is said to be *satisfiable* if there exists a witness $\vec{w} \in F^{n-|io|-1}$ such that $(A \cdot Z) \circ (B \cdot Z) = (C \cdot Z)$, where $\vec{Z} = (io, 1, \vec{w})$, and $\circ$ is the Hadamard (entry-wise) product.

Throughout the paper, assume that we are dealing with sparse R1CS instances, where $m = O(n)$.

R1CS as a language We let the language

$$R_{\text{R1CS}} = \{\langle x = (\mathbb{F}, A, B, C, io, n, m) : x \text{ is satisfiable } \rangle\}.$$

The language R_{R1CS} is NP-complete.

A Spartan zero-knowledge proof is an argument of knowledge for the R1CS language. The Prover can prove not only that a given instance $x \in R_{\text{R1CS}}$, but knowledge of the corresponding witness $\vec{w}$. At a high level, this application considers R1CS instances that represent the computational structure of ownership of a valid credential, along with any other desired properties about the credential. Thus a valid argument of knowledge implies knowledge of the underlying credential that has the claimed properties, which is necessary for real-world authentication.

2.3 High-Level Credential Presentation Flow

The following outlines the high-level protocol for a credential presentation.

1. The Prover receives a signed credential from an Issuer to be stored securely in their wallet, issued to the Wallet Secure Cryptographic Device (WSCD) public key.
2. At presentation time, the Verifier sends over challenge $\texttt{nonce}_V$ for device-binding verification
3. The Prover signs the challenge `nonce` with the public key p_U controlled by their WSCD and specified in their credential

4. The Prover computes two separate but linked zero-knowledge proofs $\pi_{\texttt{prepare}}, \pi_{\texttt{show}}$ which together cover the following statements: SD-JWT parsing, verification of the SD-JWT Issuer signature, proper disclosures and/or arbitrary predicates on the disclosures, and device-binding (i.e. checks the `nonce` signature against their public key); then sends $\pi_{\texttt{prepare}}, \pi_{\texttt{show}}$ to the Verifier.

5. The Verifier verifies $\pi_{\texttt{prepare}}, \pi_{\texttt{show}}$ independently, and also verify that they are linked; grants Prover access to some service based on their credential disclosures.

2.4 Security model

In our security model, we assume that the Prover is malicious, and that each Verifier is semi-honest, meaning that if the Prover presents a valid proof that they own a credential with some property, the Verifier will grant access to any services for which the property suffices.

For security on the Verifier's side, our soundness analysis considers the probability that a Prover without real ownership of a valid credential can generate a false proof of ownership.

For security on the Prover's side, we guarantee that our proofs are zero-knowledge, so that a semi-honest computationally bounded Verifier cannot get any additional private information about the Prover's credential given the proof, beyond what is publically revealed in the proof. In particular, we do not consider the case where the Verifier is malicious, where e.g. a false Verifier pretends to be an authorized Verifier. The problem of Verifier identity lies outside the scope of this paper.

Futhermore, we assume that Verifiers can collude, i.e. that Verifiers $V_1, \ldots, V_N$ that have receieved proofs $\{\pi_1\}, \ldots, \{\pi_N\}$ from a given Prover P can compute functions $f(\pi_1, \ldots, \pi_N)$. Therefore, we desire the **unlinkability property**: given $pi_1, \ldots, \pi_N$, the Verifiers should not be able to determine whether or not any two of these proofs came from the same Prover P. Note that this requires the Prover to re-randomize each presentation's proof; a static zero-knowledge proof of the same statement, while not revealing private credential information, will still look the same. It is possible that Verifier's can effectively de-anonymize a Prover by linking their anonymous activity across presentations and analyzing corresponding metadata, e.g. time of presentation.

3 Our zkID

main deliveries: 1. describe zkID; 2. the detailed construction

At a high-level, we propose a generic zkSNARK wrapper over a credential, which will either in SD-JWT format or the mDL data format specified in standard ISO/IEC 18013-5). The backend proving system we use will be a combination of Spartan and Hyrax commitments, with modifications to be zero-knowledge.

There are two (2) key ideas to highlight within our proposed architecture:

- **Pre-processing batches of re-randomized proofs** of issuer-signature and credential-parsing, to re-use across each new presentation. We call this the `prepare` relation.
- **Committing to the credential disclosures with Hyrax commitments** [WTas+17], which allows us to re-use (or "link") witnesses across circuits "for free"

For the first item, we note that pre-processing proofs for the `prepare` relation is possible because the relation is independent of the presentation, including the choice of disclosures or predicate proofs. Further optimizations can likely be made to only parse the discosures

of certain attributes within the credential if it is known that the Wallet User will rarely or never present certain disclosures to external verifiers.

The second items differs from the linking circuits approach that Google uses [Fas24]. Note that Google verifiably computes a hiding and binding MAC of the shared witnesses as a public output of the circuit, which the verifier checks consistency of in plain. Although this is only a few linear relations, it requires the prover to also commit to their portion of the key to prevent forging. We instead simply manually separate out the disclosures $m_1, \ldots, m_n$ into a designated column when committing to the witness, which is already needed to prove the circuit relation. Then, the verifier simply checks consistency of these commitments when verifying each circuit's proof.

We note that by using Hyrax commitments, our PoC is not post-quantum secure. In particular, post-quantum computers break the discrete-log assumption, which breaks the binding property of the polynomial commitment scheme. Thus, a malicious Prover could potentially make false proofs about their identity. In future work, we hope to incorporate modified Ajtai lattice-based commitments to ensure post-quantum security [**?**].

Throughout the remainder of this section, we refer to the EUDI's Wallet User as the "Prover", the Relying Party as the "Verifier", and the EUDI Attestation Authority (EAA) as the "Issuer". Below, we briefly detail a high-level flow of the interaction between the Issuer, Prover, and Verifier.

3.1 Underlying ZK Circuit

In this section, we describe our high-level ZK circuit C underlying the knowledge the prover actually needs to present to the verifier. Throughout, we refer to the Issuer with variable I, Prover with variable P, and Verifier with variable V (e.g. in subscripts).

We will detail the ZK wrapper around the SD-JWT credential as an example, but the protocol is analogous for other credential formats. Throughout, we will refer to digests as "message hashes" or just "hashes", and disclosures as "messages".

We define a circuit C for proving ownership of an anonymous credential. We let our witness $w = S$ be the SD-JWT credential consisting of messages $\{m_i\}_{i=1}^N$, hash salts $\{s_i\}_{i=1}^N$, hashes $\{h_i\}_{i=1}^N$, and an Issuer signature $\sigma_I = \sigma(h_1, \ldots, h_N; SK_I)$. Without loss of generality, we assume that the Prover's public key PK_P is contained in message m_1 of the credential and indexable as $m_1[1]$. We let our instance $x = (PK_I, \{f_i\}_{i=1}^K, \{p_i\}_{i=1}^K)$ contain the Issuer's public key PK_I, functions f_i over the messages, output predicates p_i, and finally the nonce signature σ_{nonce} for proving device-binding. The f_i can be arbitrary statements we wish to prove about the messages. For example, one could define a function $f_i(m_1, \ldots, m_N) = m_1$ would output a predicate that is just the disclosure of message m_1.

Underlying ZK Circuit C for Verifiable Credential

We define circuit $C(x = (PK_I, \{f_i\}_{i=1}^K, \{p_i\}_{i=1}^K), w_C = (S))$ as follows:

1. Assert $\text{parse}_{\text{SD-JWT}}(S) = (\{m_i\}, \{s_i\}, \{h_i\}_i, \sigma_I)$ parsing of the SD-JWT into messages $\{m_i\}_{i=1}^N$, message salts $\{s_i\}_{i=1}^N$, hashes $\{h_i\}_{i=1}^N$ and Issuer signature σ_I.
2. Assert $h_i = \text{SHA256}(m_i, s_i) \quad \forall i \in [n]$, i.e. that messages hashes correspond to messages and salts
3. Assert $p_i = f_i(m_1, \ldots, m_n) \quad \forall i \in [n]$, i.e. correct evaluation of the predicates
4. Assert $\text{ECDSA.verify}(\sigma_I, PK_I) = 1$, i.e. the credential signature verifies under the Issuer public key

5. Assert ECDSA.verify($\sigma_{\text{nonce}}, m_1[1]) = 1$, i.e. that the live nonce signature corresponds to the public key the credential was issued to

3.2 Pre-processing and linking proofs

The main speedups from our proving system will come from splitting our high-level circuit C above into two (2) circuits for different relations regarding the digital credential, namely a `prepare` and a `show` relation, analogous to Microsoft's Crescent Credentials [PPZ24]. This is advantageous because proofs of the `prepare` relation can be computed a-priori for any credential, as they do not depend on the claim being proved at presentation time. Pre-computing these proofs will save significant time per presentation, and reduce the performance bottleneck to that of proving the `show` relation.

One issue that arises is the need to ensure consistency of witnesses across these separate circuits, or what we call "linking proofs". At a high level, as opposed to Google's MAC approach [Fas24], the prover sends Hyrax commitments to the parts of the witness reused across circuits, which ends up being just the raw messages $\{m_i\}_i$. The verifier can then check consistency of these witnesses across the circuits C_i by comparing the Hyrax commitments they receive as part of the proof. This approach gives us linking "for free", as the Prover alreadys needs to compute these Hyrax commitments as part of the proof.

We also want to highlight that we are no longer splitting up circuits by their field operations (e.g. SHA256 attestations over a binary extension field and an ECDSA verifications over a prime field), but rather we only split up our circuit by pre-processing and per-presentation relations. In particular, the circuit for `prepare` will necessarily involve wrong-field arithmetic by including both the SHA256 hashes and the Issuer ECDSA signature verification. However, because of our ability to pre-compute proofs of the `prepare` relation, the more important thing is to choose curves that allow for i) efficient show relations and ii) linking the prepare and show relation. Since the verifier can only check equality of Hyrax Pedersen commitments defined over the same curve, we must use the same curve for proving both the `prepare` and `show` relations. Thus we choose a curve with a scalar field equivalent to the base field of the nonce signature σ_{nonce} for efficient signature verification. Because most Hardware Security Modules (HSMs) sign over the P256 curve, we choose the Tom256 (T256) curve for our backend, which has scalar field equivalent to the base field of P256.

We now detail each of the two (2) relations/circuits below.

3.2.1 The `prepare` relation:

The `prepare` relation checks the validity of issuer signature, parses the SD-JWT, and verifies all the message hashes, none of which depend on the specific presentation. Thus, the prover will periodically pre-compute and store a batch of re-randomized proofs of the prepare relation. These proofs will utilize Hyrax Pedersen vector commitments as introduced above in order to link the proofs of `prepare` relation to the `show`.

Circuit C_1 for the `prepare` relation

We define circuit $C(x = (PK_I), w_i = S, w = (\{m_i\}_{i=1}^N))$ as follows:

1. Assert $\text{parse}_{\text{SD-JWT}}(S) = (\{m_i\}, \{s_i\}, \{h_i\}_i, \sigma_I)$ parsing of the SD-JWT into messages $\{m_i\}_{i=1}^N$, message salts $\{s_i\}_{i=1}^N$, hashes $\{h_i\}_{i=1}^N$ and Issuer signature σ_I.
2. Assert $h_i = \text{SHA256}(m_i, s_i) \quad \forall i \in [n]$, i.e. that messages hashes correspond to messages and salts

3. Assert ECDSA.verify$(\sigma_I, PK_I) = 1$, i.e. the credential signature verifies under the Issuer public key

To produce zkSNARK proofs for this circuit C_1, the prover will proceed in two phases:

1. `prepareCommit`: Computes a Hyrax commitment $com_1 = com(m_1, \ldots, m_N; r_1)$ to the witness column containing message hashes $\{m_i\}_{i\in[N]}$ using initial randomness r_1.
2. `prepareBatch`:
   (a) Re-randomizes this initial commitment to get a batch of commitments $com_i = com(m_1, \ldots, m_N; r_i) = com_1 \cdot g_{N+1}^{r_i - r_1}$ for all $i \in [m]$, where our batch size m depends on the frequency of proof generation and demand for the credential
   (b) Continues the Spartan sumcheck IOP on each com_i to produce a batch of proofs $\{\pi^{(i)}_{\texttt{prepare}}\}$ for $i \in [m]$ of the `prepare` relation.

The prover will run `prepareBatch` periodically to both generate re-randomized commitments com_i and store their randomness r_i (for linking purposes), as well as generate and store batches of issuer-signature proofs $\pi^{(i)}_{\texttt{prepare}}$ that can be used for each presentation.

3.2.2 The `show` relation:

At a high-level, our show relation will i) verifiably compute any functions f_i over the SD-JWT messages (such as disclosures, range checks, etc.), and ii) check that the credential belongs to the prover's device (also known as proof of "device-binding"). As part of device-binding, the prover will sign a verifier `nonce` outside of the circuit, as outlined in flow 2.3. Let us denote this signature by $\sigma_P = \sigma(\text{nonce}; SK_P)$

Again, we will use T256 curve for our backend proving system so that the holder in-circuit signature verification can proceed naturally in the right field.

Circuit C_2 for the `show` relation

We define circuit $C_2(x = (\{f_i\}_{i=1}^K, \{p_i\}_{i=1}^K), w = \{m_i\}_{i=1}^N)$ as follows:

1. Assert $p_i = f_i(m_1, \ldots, m_n) \quad \forall i \in [n]$, i.e. correct evaluation of the predicates
2. Assert ECDSA.verify$(\sigma_{\text{nonce}}, m_1[1]) = 1$, i.e. that the live nonce signature corresponds to the public key the credential was issued to

As part of computing proof $\pi^{(i)}_{\texttt{show}}$ for presentation $i \in [m]$, the prover computes the Hyrax commitment over the Tom256 curve $com(m_1, \ldots, m_N; r_i)$, notably with *the same* randomness r_i used during the `prepareBatch` process. The verifier will check that this equals the re-randomized commitment com_i from proof $\pi^{(i)}_{\texttt{prepare}}$ for circuit C_1.

3.3 Adding ZK to Spartan

Our construction uses Circom in the frontend to compile our computation into an R1CS (instance, witness) pair $(x = (\mathbb{F}, A, B, C, io, n, m), \vec{w})$, which we then feed into the Spartan IOP coupled with Hyrax-style Pedersen polynomial commitments.

Recall that our R1CS constraint looks like the following:

$$(A \cdot \vec{Z}) \circ (B \cdot \vec{Z}) - (C \cdot \vec{Z}) = 0$$

where our square matrices A, B, C have size n and $\vec{Z} = (\vec{w}, 1, io)$.

Recall that Spartan converts an R1CS constraint into the following zero-check:

$$\sum_{x\in\{0,1\}^{\log n}} \widetilde{eq}(x,\tau)\left[\left(\sum_{y\in\{0,1\}^{\log n}} \widetilde{A}(x,y)\,\widetilde{Z}(y)\right)\left(\sum_{y\in\{0,1\}^{\log n}} \widetilde{B}(x,y)\,\widetilde{Z}(y)\right) - \left(\sum_{y\in\{0,1\}^{\log n}} \widetilde{C}(x,y)\,\widetilde{Z}(y)\right)\right] = 0$$

for some random challenge $\tau \in \mathbb{F}$

There are two components in Spartan that we need to modify to be ZK. The first is making the sumchecks ZK. The second is to ensure that the opening $\widetilde{Z}$ using the commitment to $\vec{Z}$ does not leak information about our witness $\vec{w}$.

3.3.1 Adding ZK to sumcheck

The Spartan protocol consists of several sumchecks in parallel. There are various existing techniques to make sumcheck ZK. We employ one using similar methods as in Zhang et. al. [ZXZS19], which adds random pads to the sumcheck transcript.

In particular, suppose at each round i of the sumcheck protocol, the prover sends over $s_i(X) := \sum F(r_1, \ldots, r_{i-1}, X, x_{i+1}, \ldots, x_m)$ where r_i is the Verifier challenge sent for round i. Then instead of having the verifier check the sumcheck, the prover will prove in ZK that the unpadded transcript satisfies the verifier's (linear) checks. To do this, the prover will need to commit to the random pads ahead of time. Then, as long as the Fiat-shamir challenges is generated from the transcript including these random pad commitments, the prover cannot simply lie about the pads to satisfy the sumcheck relation.

Adding ZK to sumcheck

1. Prover commits to pads $r_i(X)$ for all $i \in [\log n]$. These are linear polynomials, and can thus be represented by its 2 coefficients $r_i[0]$ and $r_i[1]$.

2. Instead of sending partial sums

$$s_i(X) := \sum_{(x_{i+1},\ldots,x_n)\in\{0,1\}^{n-i}} F(r_1, \ldots, r_{i-1}, X, x_{i+1}, \ldots, x_n)$$

   for each round of sumcheck, the Prover sends polys $s'_i(X) = s_i(X) + r_i(X)$, essentially a one-time-padded transcript.

3. It suffices to show the following linear relation in zero-knowledge,

$$\left[\begin{array}{c|c|c} A & B & C \end{array}\right] \left[\begin{array}{c} \vec{S} \\ \hline \vec{R} \\ \hline C \\ F(r) \end{array}\right] = \vec{0}$$

   where

$$\vec{S} = \left[\, s'_1[0], s'_1[1], \ldots, s'_n[0], s'_n[1] \,\right]^{\mathsf{T}},$$

   is the column vector of sumcheck transcripts such that $s'_i = s'_i(X) = s'_i[0]X + s'_i[1]$, and

$$\vec{R} = \left[\, r_1[0], r_1[1], \ldots, r_n[0], r_n[1] \,\right]^{\mathsf{T}},$$

   is the column vector of random pads, and where matrices A, B, C are given by

$$A = \begin{bmatrix} 1 & 2 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ -r_1 & -1 & 1 & 2 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & 0 & -r_2 & -1 & 1 & 2 & 0 & \cdots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & -r_{n-1} & -1 & 1 & 2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & r_n & 1 \end{bmatrix}$$

$$B = \begin{bmatrix} -1 & -2 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ r_1 & 1 & -1 & -2 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 \\ 0 & 0 & r_2 & 1 & -1 & -2 & 0 & \cdots & 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & r_{n-1} & 1 & -1 & -2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & -r_n & -1 \end{bmatrix}$$

$$C = \begin{bmatrix} -1 & 0 \\ 0 & 0 \\ 0 & 0 \\ \vdots & \vdots \\ 0 & 0 \\ 0 & -1 \end{bmatrix}$$

4. The Prover computes a public random challenge α (e.g. hashing the transcript) and compresses the relation into a dot product

$$[\vec{u}] \begin{bmatrix} \vec{S} \\ \hline \vec{R} \\ \hline C \\ F(r) \end{bmatrix} = \vec{0} \tag{1}$$

where $\vec{u} = [1, \alpha, \alpha^2, \ldots, \alpha^n] \left[\begin{array}{c|c|c} A & B & C \end{array} \right]$ is a random linear combination of the rows.

5. The Prover and Verifier engage in a proof-of-dot product protocol to prove the relation above, such as an Inner Product Argument used in Bulletproofs [BBB+17]

3.3.2 Adding ZK to the opening of $\widetilde{Z}$

In order to add ZK to the opening of, we simply append Z with random pads, which is equivalent to assigning random evaluations on the remaining points in the hypercube $\{0,1\}^{\log n}$. so that $\widetilde{Z'}(x_1, \ldots, x_{\log n}) = \widetilde{Z}(x_1, \ldots, x_{\log n}) + \sum_{x \in P} eq(r, x) Z(x)$, where P is the set of points in $\{0,1\}^{\log n}$ that used to be assigned 0 by default, but were filled with random pads. Notice that if any of $Z(x)$ are random, then $\sum_{x \in P} eq(r, x) Z(x)$ is random. If $n = |Z|$ is not already a power of 2, then we can simply fill at least one of the remaining evaluations on $\{0,1\}^{\log n}$ with a single random pad. If $n = 2^m$, we can add another dimension to the hypercube of evaluations of Z.

3.4 Cost analysis

The following section computes the Prover and Verifier costs of Spartan instantiated with Hyrax Pedersen commitments on R1CS instances $(x = (\mathbb{F}, A, B, C, io, n, m), \vec{w})$, where io

denotes the vector of public inputs/outputs, $n = |\vec{w}|$ is the dimension of our matrices, and m is the number of nonzero entries in our matrices A, B, C. We let $Z = (\vec{w}, io, 1)$. It is often reasonable to assume that our R1CS matrices are sparse, i.e. $m = O(n)$. However, we present the costs below independent of this assumption.

- **Prover time**: (1) $O(m)$ to generate sumcheck transcript, (2) $O(m)$ to evaluate MLEs of A, B, C, (3) $O(n)$ to commit to the MLE of Z (computing $\sqrt{n}$ MSMs of size $\sqrt{n}$) and opening the MLE of Z, for a total cost of $O(m)$.
- **Proof length**: (1) $O(\log n) \cdot |\mathbb{F}|$ length of the sumcheck transcript, (2) $O(\sqrt{n}) \cdot |\mathbb{G}|$ length commitment to MLE of Z, (3) $O(\log n) \cdot |\mathbb{G}|$ length of argument opening MLE of Z, for a total length of $O(\sqrt{n})$ group or field elements.
- **Verifier time**: (1) $O(\log n)$ to verify sumcheck transcript, (2) $O(m)$ to evaluate the MLEs of A, B, C (with sparse commitment scheme and memory checking), (3) $O(\sqrt{n})$ to open the MLE of Z, for a total of $O(m + \sqrt{n})$.

With the ZK modifications to Spartan, we can see the asymptotic costs remain the same, as follows:

- **Prover time**: additionally computes $O(\log n)$ constant-size commitments to the sumcheck transcript pads r_i, and $O(\log n)$ engages in new sumcheck relation IPA (or some other ZK dot product argument) for vector of length $O(\log n)$, which still gives $O(m)$ prover work.
- **Proof length**: sumcheck and openings are the same length but just padded, but added on $O(\log n)$ size $|\mathbb{G}|$ commitments, and a length $O(\log \log n)$ sumcheck relation IPA proof, which still gives a proof length of $O(\sqrt{n})$ group or field elements.
- **Verifier time**: no longer needs to do $O(\log n)$ (sumcheck), but still needs $O(m)$ (evaluating MLEs of A, B, C) + $O(\sqrt{n})$ (opening MLE of Z) + $O(\log n)$ for sumcheck relation IPA verification, which still gives $O(m + \sqrt{n})$ runtime.

3.5 Security analysis

The correctness follows immediately from the correctness of the Spartan SNARK and the fact that the Prover uses the same randomness for the Hyrax commitments across the `show` and `prepare` circuits for each presentation i.

The soundness of our protocol follows from the soundness of Spartan. In particular, we can extract the full witness credential from the `prepare` relation.

Intuitively, zero-knowledge follows from the hiding property of the commitment scheme as well as the zero knowledge property of the Spartan zkSNARK proving system; For proof i, simulator can randomly sample the linked commitment com_i both distributions to reuse across both proofs, both in the commitment itself and also in the IPA used to open the Hyrax commitment to $Z(r_1, \ldots, r_{\log n})$. We can show that this commitment is independent of the rest of the view of the Verifier, which consists of the following:

- Sumcheck polynomials $\{s'_i(X)\}_{i \in [\log n]}$ for each of the sumchecks in Spartan
- $\{r_i\}$ Fiat-Shamir challenges during the sumcheck
- Transcript from the IPA on the sumcheck relation in ZK
- $\{com(z_i)\}_{iin[\log n]}$ Hyrax commitment to Z, which involves a Pedersen commitment to each of the rows of a $\log n \times \log n$ matrix representation of $\vec{Z}$
- Transcript from the IPA for opening $Z(r_1, \ldots, r_n)$

- The claimed value of $Z(r_1, \ldots, r_n)$

Since we appended random pads to $\vec{Z}$ in our ZK modifiction in Section 3.3.2, the distribution of $Z(r_1, \ldots, r_n)$ is random and independent of Z, and therefore independent of $\{m_i\}_i$. Furthermore, $s'_i(X)$ have totally random pads on them and their distribution is independent of Z, and therefore independent of $\{m_i\}_i$. Assuming the hiding property of the Pedersen commitment schemes for messages sent during an IPA, we can also use the simulators for the IPAs without changing their joint distribution with the rest of the transcript.

Then, we can simply run the piece-wise simulators for each zkSNARK proof for circuits C_1 and C_2 to simulate the remainder of the view.

4 Experiments

5 Application to EUDI

Within the EUDI Architecture and Reference Framework [Eur23], the practical question is how to introduce zero-knowledge capabilities without disrupting established roles, formats, and certification paths. This section states how the construction fits that setting and what trade-offs it entails.

Issuance. The construction is designed to wrap existing credential encodings rather than replace them. It accommodates SD-JWT and ISO/IEC 18013-5 mDL so that wallets and relying parties retain current disclosure grammars and parsing logic. Issuers (PID Providers and Attestation Providers) remain oblivious to the use of zero-knowledge proofs; no changes to issuance pipelines or device secure elements are required, and issuers keep exclusive control of their private keys. The proof layer is circuit-defined, which allows future issuer-side migrations (for example, a change of signature scheme) to be handled by updating the verification circuit rather than introducing format-specific protocols. The approach interoperates with current PKI based on ECDSA or RSA and does not prescribe a switch of algorithm or hardware.

Efficiency. Proving is split into two relations. A fixed relation captures issuer-signature verification, credential parsing, and commitment preparation; it runs infrequently and is amortized per credential. A live, presentation-specific relation captures the disclosures and predicates for a single session; it runs per presentation. This separation aims to keep holder and verifier costs within typical web and mobile budgets and to bound latency on the critical path. The proof system and commitment layer are modular, so improvements in either component can be adopted without redesigning the higher-level flow. In contrast to designs that keep issuer verification online, issuer-signature verification and parsing are moved to the offline step here to reduce presentation-time work at the verifier.

Discussion. The present instantiation follows the Spartan line and relies on sumcheck and Pedersen/Hyrax-style commitments rather than pairing-based assumptions; there is no universal trusted setup. The choice avoids pairing-friendly curves and the operational burden of a setup ceremony across many issuers and relying parties. The construction is not post-quantum secure in its current form, but the modular structure leaves a path to replacing the vector-commitment layer with lattice-based alternatives as they mature. Some components have not yet been standardized; this is a shared condition across competing approaches and is called out explicitly in our roadmap.

Summary for EUDI. The design aligns with Annex 2 format expectations, requires no changes to issuers, supports current PKI deployments, and separates fixed from presentation-specific work to keep online costs low. It avoids pairing-based assumptions and a universal setup and leaves a path to future cryptographic upgrades without disrupting wallet or issuer operations.

6 Security

7 Conclusion

References

[AHIV17] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 2087–2104, Dallas, TX, USA, October 31 – November 2, 2017. ACM Press.

[BBB+17] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. Bulletproofs: Short proofs for confidential transactions and more. Cryptology ePrint Archive, Paper 2017/1066, 2017.

[BCTV14] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Scalable zero knowledge via cycles of elliptic curves. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part II*, volume 8617 of *Lecture Notes in Computer Science*, pages 276–294, Santa Barbara, CA, USA, August 17–21, 2014.

[Eur23] European Commission. The european digital identity wallet architecture and reference framework. Technical report, European Commission, 2023.

[Fas24] Matteo Frigo and abhi shelat. Anonymous credentials from ECDSA. Cryptology ePrint Archive, Paper 2024/2010, 2024.

[FFS88] Uriel Feige, Amos Fiat, and Adi Shamir. Zero-knowledge proofs of identity. *Journal of Cryptology*, 1(2):77–94, June 1988.

[fS21] International Organization for Standardization. Iso/iec 18013-5:2021 personal identification — iso-compliant driving licence part 5: Mobile driving licence (mdl) application, 09 2021.

[FYC25] Daniel Fett, Kristina Yasuda, and Brian Campbell. Selective disclosure for JWTs (SD-JWT). Technical Report draft-ietf-oauth-selective-disclosure-jwt-22, IETF OAuth WG, May 2025. Internet-Draft.

[GGF17] Paul A Grassi, Michael E Garcia, and James L Fenton. Digital identity guidelines. *NIST Special Publication 800-63-3*, 06 2017.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In *EUROCRYPT 2016*, volume 9665 of *LNCS*, pages 305–326. Springer, 2016.

[KyPRK24] Evan Krul, Hye young Paik, Sushmita Ruj, and Salil S. Kanhere. Sok: Trusting self-sovereign identity. *Proceedings on Privacy Enhancing Technologies (PoPETs)*, 2024(4):60–85, 2024.

[PPZ24] Christian Paquin, Guru-Vamsi Policharla, and Greg Zaverucha. Crescent: Stronger privacy for existing credentials. Cryptology ePrint Archive, Paper 2024/2013, 2024.

[Set20] Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 704–737, Santa Barbara, CA, USA, August 17–21, 2020.

[TZ23] Stefano Tessaro and Chenzhi Zhu. Revisiting BBS signatures. In Carmit Hazay and Martijn Stam, editors, *Advances in Cryptology – EUROCRYPT 2023, Part V*, volume 14008 of *Lecture Notes in Computer Science*, pages 691–721, Lyon, France, April 23–27, 2023.

[WTas+17] Riad S. Wahby, Ioanna Tzialla, abhi shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zkSNARKs without trusted setup. Cryptology ePrint Archive, Paper 2017/1132, 2017.

[WTs+18] Riad S. Wahby, Ioanna Tzialla, abhi shelat, Justin Thaler, and Michael Walfish. Doubly-efficient zkSNARKs without trusted setup. In *2018 IEEE Symposium on Security and Privacy*, pages 926–943, San Francisco, CA, USA, May 21–23, 2018. IEEE Computer Society Press.

[ZXZS19] Jiaheng Zhang, Tiancheng Xie, Yupeng Zhang, and Dawn Song. Transparent polynomial delegation and its applications to zero knowledge proof. Cryptology ePrint Archive, Paper 2019/1482, 2019.

8 Appendix: EUDI Annex 2 Requirements

This secion is devoted to a review of the EUDI ARF's Annex 2, which covers high-level requirements for the EUDI Wallet. The full Annex can be found here.