OpenAC: Open Design for Transparent and Lightweight Anonymous Credentials

The zkID Team @ PSE
pse-zkid@ethereum.org

Ethereum Foundation

November 24, 2025

Abstract

Digital identity systems require mechanisms for verifiable, privacy-preserving presentations of user attestations. The trivial approach of utilizing selective disclosure by presenting individually signed attestationsintroduces persistent linkability that compromises user anonymity. Existing anonymous credential systems come with practical drawbacks. Some depend on trusted setups, others require substantial modifications to an issuer's established issuance flow.

We propose an open, transparent, and lightweight anonymous credential design that addresses these limitations with the use of zero-knowledge proofs. Our construction is modular, requires no trusted setup and integrates with existing workflows without the need for substantial changes to existing cryptographic mechanisms, procedure overhauls, or hardware devices. It delivers unlinkability while maintaining broad applicability across heterogeneous digital-identity ecosystems and current verifiable credential standards.

To demonstrate practicality, we provide a proof-of-concept implementation and benchmarks on mobile devices. Our results show best-in-class proving times, with a focus on efficient client-side proving, an essential requirement for usability in digital identity wallets.

OpenAC was purposely constructed to be compatible with the European Digital Identity Architecture and Reference Framework (EUDI ARF). In the appendix, we map EUDI ARF's functional, privacy, and interoperability requirements, illustrating how OpenAC satisfies regulatory constraints while preserving strong user privacy.

Contents

1 Introduction

Digital Identity. A digital identity system typically consists of three roles: an *issuer*, a *holder*, and a *verifier*. In the W3C Verifiable Credential Data Model, the issuer is a trusted authority responsible for asserting claims about one or more subjects, and creating a verifiable credential from these claims. The claims are digitally signed with the issuer's private key. The signed credential is delivered securely to the user and stored in the wallet instance.

The holder retains these signed verifiable credentials and, during a presentation request, provides consent based on the verifier's requirements. The holder sends the verifiable presentation (or proofs derived from them) to the verifier. The verifier, using the issuer's public key, checks the digital signature to confirm authenticity and integrity of the received presentation, and may additionally verify validity period, or revocation status. This issuer-holder- verifier pattern defines the standard operational model.

Such a system is illustrated in Fig. 1.

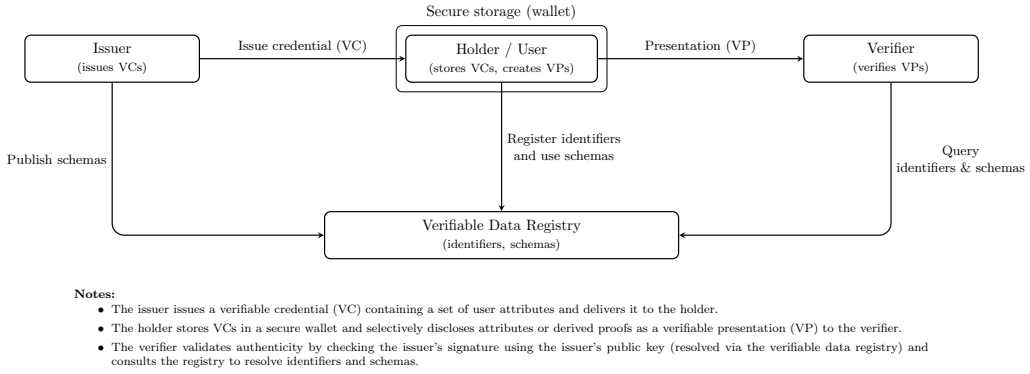

Figure 1: Digital identity system

Examples of such digital identity frameworks and credential formats that can be potentially compatible with OpenAC include:

- **eIDAS / EUDI Architecture and Reference Framework** -European digital identity systems in which qualified trust service providers or public authorities issue digitally signed identity attributes that citizens can present via the EUDI Wallet.
- **Mobile Driver's Licenses (mDL, ISO 18013-5)** - Government agencies issue a digitally signed set of driving-related attributes (e.g., name, age, license class), which users present to relying parties such as law enforcement or age-restricted service providers.
- **Verifiable Credentials (W3C VCDM)** -A general-purpose model where an issuer signs JSON-LD or SD-JWT-based claims about a subject; holders store these credentials and present them to verifiers who validate the issuer's signature.

Anonymous Credentials from Existing Digital Identity. We focus on constructing anonymous credentials for *existing* digital identity systems without requiring changes to the current issuance flow whilst enabling selectively disclosed presentations in an unlinkable manner.

A central target of this work is the *EUDI ARF*, which specifies how digital identities and credentials are issued, stored, and presented within the EUDI ARF. While our terminology in this section follows the generic issuer–holder–verifier model, Section 2.2 restates the same roles using the Wallet User, Relying Party, and EAA vocabulary of the EUDI ARF. Our construction is designed to integrate with this framework in a modular manner, allowing anonymous credentials to be derived from standard verifiable credentials, whilst complying with its security, interoperability, and regulatory requirements.[1]

Informally speaking, an *Anonymous Credential* AC scheme allows:

- An *Identity Provider* or *Issuer* IP to (possibly blindly[3]) sign a set of (eligible) attributes for a *User* U;
- The *User* U can show, only if they hold the signed attributes (a.k.a *Unforgeability*), usually through a *Presentation*, to a *Relying Party* RP such that:
  - The RP can verify that the set of attributes (signed by IP) that the User U holds satisfy some condition of their interest (a.k.a Correctness);
  - The RP cannot learn any *additional*[4] information beyond the fact that the condition is satisfied or information that can be inferred from the satisfaction of the condition (a.k.a Zero-Knowledge or Anonymity);
  - The immediate previous requirement also implies that the RP cannot link the various presentations by the same User U (a.k.a. Unlinkability);
- The IP can revoke all or a part of the signed attributes that it has issued to the User U, from upon which, the eligible attributes of the User U are updated, and subsequent presentations have to be based on the new and updated attributes (a.k.a *Revocation*);
- The User U cannot transfer its set of signed attributes to another User U' (a.k.a *Non-transferability*).

A Generic AC Framework. In a generic AC framework, the *Issuer* is an entity that uses a fixed public-key algorithm (e.g., RSA or ECDSA) and verifiable credential format (such as SD-JWT or mDL) to assert claims about one or more subjects. Since issuer elements are typically difficult to modify once deployed, the proposed change are to the holder's wallet and to the relying party (verifier).

The wallet operates in two phases. During an offline *Prepare* phase, run once per credential, the wallet:

1. Verifies the Issuer's signature using standard libraries
2. Parses and normalizes credential attributes (e.g., converting a date of birth to an integer age)
3. Commits to the attributes using a binding and hiding commitment scheme.

During the online *Show* phase, which is run for each presentation, the wallet:

1. Selects the attributes or predicates required by the *Relying Party*
2. Prove them in zero knowledge against the stored commitments

[1] The EU Digital Identity Wallet will be launched in all member states by the end of 2026, with strong requirements for unlinkability with respect to relying parties and identity providers. According to the Cryptographers' Feedback on the EU Digital Identity's ARF[2], an Anonymous Credential AC scheme is a suitable cryptographic primitive to instantiate the new EU Digital Identity Wallet (EUDIW, ARF version 1.4.0 [Eur24]), which is an important step towards developing interoperable digital identities in Europe for the public and private sectors.

3. Adds a fresh device signature over the session challenge to ensure device binding.

Noticeably, a key requirement of our framework is *modularity*: issuer-signature verification, attribute commitment, predicate proofs, and device binding are defined as separate modules with clear interfaces. This separation allows the underlying proof system to be replaced for quantum-proofing without modifying other components.

Related Work and Our Proposal. There exists several prominent approaches to constructing anonymous credentials from existing digital identity systems, including the BBS+ proposal [ASM06, CDL16], Microsoft's *Crescent* [PPZ24], and Google's *Longfellow* [Fas24]. Although each of these schemes enables some form of unlinkable selective disclosure, they exhibit limitations in practicality or generality that our design aims to overcome.

The BBS+ approach requires modifications to the issuer's existing credential-issuance flow, thereby reducing compatibility with established digital identity infrastructures. The Crescent system relies on a trusted setup, which introduces transparency concerns and operational overhead for ecosystem deployment. The Longfellow construction, while efficient, is tailored specifically to ECDSA-based identity systems and thus lacks general applicability across heterogeneous credential formats.

Our proposal offers an *open* and *fully transparent* design that imposes no changes on the issuer workflow, supports modular "plug-and-play" integration with diverse credential systems, and requires no trusted setup. Furthermore, our proof-of-concept implementation demonstrates best-in-class proving performance, including efficient execution on mobile devices, which is essential for real-world usability in frameworks such as the EUDI Wallet.

A summary of the existing solutions and our proposal with respect to the AC Framework is presented in Table 1. We additionally adapt a comparison for a 1920-byte credential from Vega [KS25], a concurrent work that follows the similar prepare-and-prove paradigm.[5] For OpenAC, the four latency columns follow the same convention as Vega: *Setup* covers the proof system setup for the Prepare and Show circuits; *Precomp.* is the one-time offline Prepare work; *Prove* is the online Show work for a single presentation; and *Verify* is the verifier's work across both phases. The proof size column reports the sum of the Prepare and Show proof sizes, and the PK/VK columns give the total proving and verifying key sizes for the two circuits.

We consider a client–server setting with a mobile prover and a server-side verifier. The verifying key, which is dominated by the Prepare circuit, is large but kept in memory on the verifier side and reused across many proof verifications, while the prover runs under tighter latency and resource constraints. For consistency with Vega, Table 2 reports measurements on commodity desktop hardware; mobile measurements and a more detailed benchmarking discussion, including a breakdown from the wallet's perspective, appear in Section 4.

2 Technical Overview and Applicability to EUDI ARF

2.1 Technical Overview

Notation For $n \in \mathbb{N}$, we write $[n] = \{1, \ldots, n\}$. Bold letters denote vectors, e.g., $\mathbf{m} = (m_1, \ldots, m_n)$. Concatenation is written $\|$. The security parameter is λ; $\mathsf{negl}(\lambda)$ denotes a negligible function. For a (possibly randomized) algorithm Alg, we write $y \leftarrow \mathsf{Alg}(x)$ for its output on input x.

[5] We have not yet attempted to compare our work with Vega in this version, we will conduct a more detailed analysis of Vega in the next version.

Table 1: Comparison of related approaches.

Feature	**BBS/BBS+**	**Longfellow**	**Crescent**	**OpenAC**
Issuer modification	Required	None	None	None
Offline phase	None	None	Lightweight, reusable Prepare	Lightweight, reusable Prepare
Setup	Pairing-based (no trusted setup)	Transparent	Large, per-circuit trusted setup	Transparent
Proof mechanism	Pairing-based signatures + ZK proofs	Sum-check + Ligero; custom ECDSA/SHA-256 circuits	Groth16 + Pedersen vector commitments; re-randomizable artifacts	Sum-check; Hyrax-style vector commitments
Device binding	Optional	Included	Optional	Integrated (in-circuit)
Reusability	No	No	Yes	Yes

Table 2: Performance comparison for a 1920-byte MSO, adapted from Vega [KS25].

Scheme	**Latency (ms)**				**Size (kB)**			**Trans. Setup**
	Setup	*Precomp.*	*Prove*	*Verify*	*Proof*	*pk*	*vk*	
Longfellow	7 235	—	680	324	325	202	202	✓
Crescent	172 437	14 725	237	118	16	710 565	1	×
Vega_{SC}	3 689	238	247	55	99	6 562	6 561	✓
Vega_{MC}	193	109	212	51	150	436	436	✓
OpenAC	4193	3442	102	83	149.7	433664	433664	✓

The Longfellow, Crescent, and Vega benchmarks are done in Azure Standard_F16as_v6 VM with 16 vCPUs and 64 GB RAM while we consider a commodity hardware (MacBook Pro, M4, 14-core GPU, 24GB RAM).

We follow the generic W3C VC roles:

- The *issuer* I signs credentials with a long-term key pair (SK_I, PK_I) (e.g., ECDSA P–256 or RSA).
- The *prover* P is the holder's wallet, which stores credentials and generates proofs.
- The *verifier* V is the relying party that checks proofs against a policy.

For device binding, the prover's secure element holds an additional signing key pair (SK_D, PK_D) used only to sign fresh per-session challenges.

Credentials A credential is a standardized signed object S (e.g., SD–JWT [FYC25] or mDL [fS21]). Parsing maps S into an ordered vector of attributes

$$\mathbf{m} = (m_1, \ldots, m_n).$$

Non-numeric fields (strings, dates) are encoded injectively into integers. The resulting integers are interpreted in a prime field $\mathbb{F} = \mathbb{F}_q$ chosen for the proof backend. For each attribute m_i we sample a salt $s_i \leftarrow \mathbb{F}$ and compute

$$h_i = \mathsf{H}(m_i \,\|\, s_i),$$

where H is instantiated as SHA–256. The issuer's signature is

$$\sigma_I = \mathsf{Sign}_{SK_I}(h_1, \ldots, h_n),$$

verified under PK_I.

zk-SNARKs. We summarise the zk-SNARK terminology used in this paper. A zk-SNARK is a non-interactive proof system between a prover and a verifier in which the prover, holding some private data (the *witness*) and public data (the *public input*), produces a short proof that the public input and witness satisfy a given relation, and the verifier decides whether to accept the proof using only the public input [Gro16]. In our setting, the zk-SNARK prover is always in the holder wallet P and the zk-SNARK verifier is always the relying party V.

Many zk-SNARK and succinct argument constructions are specified with a setup algorithm that outputs public parameters, typically split into a proving key (used by the prover) and a verifying key (used by the verifier) [GGPR13, BCTV14, Gro16, Set20]. In pairing-based systems such as Groth16 and related schemes [GGPR13, BCTV14, Gro16], this setup is usually described in terms of a structured reference string generated using secret randomness; if this randomness were ever revealed, an adversary could in principle construct convincing proofs for false statements. We refer to this as a *trusted setup*. In contrast, transparent argument systems such as Spartan derive their public parameters from public randomness or a fixed public description and do not rely on a trapdoor in the setup [Set20].

We use standard notions of *correctness*, *soundness*, and *zero-knowledge* [Gol01]. Correctness means that an honest prover, given a true statement and an appropriate witness, produces proofs that an honest verifier accepts. Soundness means that no efficient prover can make an honest verifier accept a false statement, except with negligible probability. Zero-knowledge means that the Verifier learns nothing beyond the validity of the statement; informally, there exists an efficient simulator that can produce indistinguishable proofs without using the witness. When we refer to a *lightweight* prover, we mean that, although zk-SNARK provers are typically the dominant cost compared to verification [Gro16], our prover fits within the latency and memory constraints of the target mobile devices, as shown in Section 4. In the remainder of the paper, the terms *witness*, *public input*, *proving key*, and *verifying key* are used in this sense.

Credential Circuit C We describe a high-level circuit C that captures the knowledge the Prover must demonstrate to the Verifier. We present the wrapper for SD-JWT credentials; the same approach applies to other credential formats as well. Following standard terminology, we use "message hashes" for digests and "messages" for disclosures. We denote the Issuer by I, the Prover by P, and the Verifier by V.

Let $w = S$ be the SD-JWT credential consisting of messages $\{m_i\}_{i=1}^{N}$, salts $\{s_i\}_{i=1}^{N}$, hashes $\{h_i\}_{i=1}^{N}$, and the Issuer signature $\sigma_I = \sigma(h_1, \ldots, h_N; SK_I)$. Without loss of generality, assume the Prover's public key PK_P is contained in m_1 and indexable as $m_1[1]$. The public instance is

$$x = \big(PK_I, \{f_i\}_{i=1}^{K}, \{p_i\}_{i=1}^{K}, \sigma_{\text{nonce}}\big),$$

which includes the Issuer's public key, the functions f_i over messages (whose outputs are compared against predicates p_i), and the nonce signature used for device binding. The f_i may encode either disclosures or predicates; for example, $f_i(m_1, \ldots, m_N) = m_1$ outputs the disclosure of m_1.

Underlying ZK Circuit C for Verifiable Credential

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

1. Assert $\text{parse}_{\text{SD-JWT}}(S) = (\{m_i\}, \{s_i\}, \{h_i\}, \sigma_I)$, i.e., correctly parse the SD-JWT into messages, salts, hashes, and the Issuer's signature.
2. Assert $h_i = \text{SHA256}(m_i, s_i)$ for all $i \in [N]$.
3. Assert $p_i = f_i(m_1, \ldots, m_N)$ for all $i \in [K]$.
4. Assert $\text{ECDSA.verify}(\sigma_I, PK_I) = 1$.
5. Assert $\text{ECDSA.verify}(\sigma_{\text{nonce}}, m_1[1]) = 1$, i.e., the live nonce signature matches the public key to which the credential was issued.

Commitments and Proof Interface To support selective disclosure without revealing raw attributes, the wallet commits to $\mathbf{m}$ using Pedersen vector commitments. Let $\mathbb{G}$ be a cyclic group of prime order q with public generators $(g_1, \ldots, g_n, h)$ derived from a domain-separated seed. For randomness $r \leftarrow \mathbb{F}$, the commitment is

$$C = \prod_{i=1}^{n} g_i^{m_i} \cdot h^r \ \in \ \mathbb{G}.$$

Under discrete-logarithm hardness in $\mathbb{G}$, these commitments are computationally binding; they are also perfectly hiding. To avoid linkability, the wallet re-randomizes r across sessions. If it precomputes several reusable commitments, we index them $(C^{(j)}, r^{(j)})$; both offline and online proofs in a session reference the same $C^{(j)}$, allowing the verifier to link the phases without learning $\mathbf{m}$.

Credential use is captured by two relations:

- *Prepare (offline).* Once per credential, the wallet verifies σ_I under PK_I, parses S into $\mathbf{m}$, computes digests $\{h_i\}$, derives a commitment $C^{(j)}$, and produces a reusable proof

$$\pi_{\text{prep}}^{(j)}: \quad \text{“}S \text{ parses to } \mathbf{m},\ \sigma_I \text{ verifies, and } C^{(j)} \text{ commits to } \mathbf{m}\text{”}.$$

- *Show (online).* For each presentation, the verifier sends a challenge ch. The device signs it as $\sigma_{ch} = \mathsf{Sign}_{SK_D}(ch)$. The wallet proves that all predicates in the verifier's policy hold with respect to $C^{(j)}$ and incorporates σ_{ch}:

$$\pi_{\text{show}}^{(j)}: \quad \text{“policy holds for } C^{(j)}\text{, and the session is bound via } \sigma_{ch}\text{”}.$$

  The verifier checks $\pi_{\text{prep}}^{(j)}$, $\pi_{\text{show}}^{(j)}$, their consistency on $C^{(j)}$, and verifies σ_{ch} under PK_D.

This split amortizes heavy work (signature verification, parsing, commitment) offline, leaving online interaction to short proofs plus one device signature.

Predicates and Policies A *predicate* is a Boolean function $f(\mathbf{m}[S]) \in \{0,1\}$ over a subvector indexed by $S \subseteq [n]$. Typical predicates include range checks ($m_i \geq 18$), equality or membership tests (e.g., m_i equals a country code), and cross-credential comparisons. A *policy* is a finite set of predicates chosen by the verifier. In each session, the wallet proves in zero knowledge that all predicates in the policy hold with respect to $C^{(j)}$, revealing only what the policy requires. Because predicates are modular, the proving backend can be swapped (e.g., from a SNARK to a post-quantum argument system) without changes to issuer infrastructure.

We assume the issuer is honest and operates standard PKI. Verifiers are semi-honest: they check proofs correctly but may collude to compare transcripts. Unlinkability relies on re-randomization of commitments, so the only stable value within a session is $C^{(j)}$, intentionally shared between *Prepare* and *Show*.

2.1.1 Overview of Security and Further Considerations

Adversarial Model We assume a malicious Prover and semi-honest Verifiers. If the Prover produces a valid proof that they own a credential with some property, the Verifier grants access to any service for which that property suffices. Malicious Verifiers are out of scope, and verifier authentication is not modeled. During issuance, the Issuer is trusted by both parties not to issue false credentials or to leak the personal information required for issuance.

Soundness and Zero-Knowledge Soundness requires that any probabilistic polynomial-time (PPT) Prover without a valid credential convinces the Verifier only with negligible probability in the security parameter λ. This guarantee follows directly from the soundness of the underlying proof system. Zero knowledge holds against semi-honest PPT Verifiers: there exists a simulator that, given only the public outputs and any explicitly disclosed attributes, generates a transcript computationally indistinguishable from a real execution. Hence, hidden attributes remain confidential. These guarantees apply independently to each presentation.

Collusion and Unlinkability Verifiers may collude. Specifically, $V_1, \ldots, V_N$ that receive transcripts $\pi_1, \ldots, \pi_N$ from a Prover P may compute joint functions $f(\pi_1, \ldots, \pi_N)$. The desired property is *unlinkability*: given $\pi_1, \ldots, \pi_N$, colluding Verifiers should not be able to determine whether two transcripts originate from the same P. Achieving this requires per-presentation re-randomization; otherwise, fixed transcripts would be linkable across sessions and presentations, and metadata (e.g., timing) could aid de-anonymization. Our construction re-randomizes proofs between presentations and admits simulation of each transcript without the witness. A joint view of colluding Verifiers can be simulated by independently simulating each transcript.

Issuer Visibility and Operational Considerations In the baseline presentation, Verifiers do not collude with Issuers even though the Issuer public key is visible. To mitigate Issuer tracking under potential collusion, a trusted Merkle tree of Issuer public keys can be maintained: the Prover proves knowledge of a valid Issuer signature under some key in the tree, and the public input is the Merkle root rather than a specific Issuer key. Presentation does not require Issuer interaction beyond initial issuance. Live network access is typically required to check credentials against the current state. When the Issuer public key is a public input, the Verifier checks an online registry of trusted Issuer keys; with Merkle inclusion, the Verifier checks that the public Merkle root matches the trusted online root. Deferred (offline) verification is possible but shifts risk to the Verifier;

acceptable delay and risk are application-specific (e.g., periodic registry snapshots and temporary validation against the last snapshot).

2.2 Alignment with the EUDI ARF

A central question within the EUDI Architecture and Reference Framework [Eur23], is how to introduce zero-knowledge capabilities without disrupting established roles, credential formats, or verification flows. This section explains how the OpenAC construction fits that setting and outlines the associated trade-offs. The subsections follow Topic G in the EUDI ARF discussion thread.

Throughout, we refer to the Wallet User as the *Prover* (Holder), the Relying Party as the *Verifier*, and the EUDI Attestation Authority (EAA), which issues PID and attestations as the *Issuer* in the W3C VC model.

Issuance. The construction is designed to wrap, rather than replace, existing credential encodings. It supports SD-JWT and ISO mDL as credential formats, allowing wallets and Verifiers to retain current disclosure grammars and parsing logic. Issuers remain unaware of the use of zkSNARKs; no changes to issuance pipelines or secure elements are required, and Issuers retain exclusive control over their private keys. The proof layer is circuit-defined and therefore highly programmable, enabling adaptation to future Issuer-side changes (e.g., a change of signature scheme) by updating the Prover circuit and public parameters, without introducing format-specific protocols. The approach interoperates with existing PKI based on ECDSA or RSA and does not mandate algorithm or hardware changes.

Efficiency. Proving is divided into two relations. A fixed relation handles Issuer-signature verification, credential parsing, and commitment preparation; it runs infrequently and is amortized per credential. A live, presentation-specific relation handles disclosures and predicates for a given session, and is run per presentation. This separation aims to keep Prover time and memory within common web and mobile budgets,[6] while minimizing latency during Prover–Verifier interaction. The proof system and commitment layer are modular, allowing upgrades to either without redesigning higher-level flows. In contrast to other approaches, verification of Issuer signatures and parsing are pre-computed offline to reduce online proving overhead.

Considerations for Post Quantum Resistance and Standardization. The current instantiation follows the Spartan family and relies on sumcheck and Hyrax-style Pedersen commitments under the discrete-logarithm assumption, avoiding pairing-based assumptions and any universal trusted setup. It thus avoids the operational burden of coordinating a trusted setup across many independent Provers, Issuers, and Verifiers. Although not yet post-quantum secure, the modular structure leaves a clear path toward replacing the commitment layer with lattice-based alternatives as they mature. Some components lack formal standardization; this is shared across competing designs and is noted explicitly. Concerning standardization: sumcheck is a classical, information-theoretic protocol, and Pedersen commitments—introduced in [Ped92]—rely only on the discrete-logarithm assumption, which already underpins standardized ECDSA signatures.

[6] Based on the a recent survey (`https://hackmd.io/@clientsideproving/AvgMobileHardware`), we model a baseline *iPhone* as having a 6-core CPU at 3.33 GHz and 5.7 GB of usable RAM. Similarly, from the available *Android* data, we derive a baseline Android device with a 7-core CPU at 2.16 GHz and 3.82 GB of usable RAM (computed by averaging the core counts and taking the minimum RAM values across the reported models).

2.2.1 Mapping of Requirements

The design is consistent with the format expectations of Annex 2, introduces no changes for Issuers, supports existing PKI deployments, and cleanly separates fixed from presentation-specific work to minimize live presentation costs. It avoids pairing-based assumptions and any universal setup, while preserving a clear path for future cryptographic upgrades without disrupting wallet or Issuer operations.
The EUDI Annex 2 spans more than fifty topics covering cryptographic guarantees, privacy, trust infrastructure, wallet lifecycle, and product-level usability. We group these requirements into four classes according to their relationship with the OpenAC protocol:

Group A: Directly satisfied by OpenAC.
Requirements already fulfilled by the OpenAC protocol as described in Sections §5–§5.3, without additional assumptions or infrastructure. These appear in Table 7.

Group B: Satisfied with minor extensions.
Requirements that can be met through small circuit adjustments, additional predicates, or simple interface configuration. No new trust anchor or cryptographic primitive is needed. These are listed in Table 8.

Group C: Rely on external systems or operational flows.
Requirements that depend on registries, PKI governance, revocation mechanisms, wallet attestation management, or other policy-layer components outside the proving system. OpenAC interoperates with these systems but does not replace them. These are summarized in Table 9.

Group D: Product, UX, and policy-facing requirements.
Requirements concerning wallet behaviour, user experience, accessibility, and legal presentation. These lie outside cryptographic protocol design but remain fully compatible with OpenAC. These appear in Table 10.

Each table records:

- the relevant Annex 2 topic and its HLRs;
- a brief informal description of the requirement;
- and the subsection(s) in this document where OpenAC's relation to the requirement is discussed.

The complete landscape tables for all groups are provided in Section 7.

3 Technical Details

In this section, we provide technical details of our OpenAC design. In Section 3.1, we discuss splitting the credential relation into two parts, *Prepare* and *Show*, and explain how these two relations are linked in the final construction. In Section 3.2, since our instantiation relies on *Spartan* for arithmetization and Spartan does not natively provide zero-knowledge, we describe how to add zero-knowledgeness to the Spartan arithmetization layer and analyze the security of this extension. Finally, in Section 3.3, we conclude with a high-level sketch of the security analysis for the full construction.

3.1 Pre-processing and linking proofs

We split C into two circuits that capture distinct relations about the credential: a `prepare` relation and a `show` relation (analogous to Microsoft's Crescent Credentials [PPZ24]). Proofs for `prepare` can be precomputed for any credential, since they are independent of the claims proved during presentation. Consequently, the online latency is dominated by the `show` relation.

A central requirement is the consistency of shared witnesses across these circuits ("linking"). Rather than the MAC-based technique of [Fas24], the Prover sends Hyrax commitments to the witness components reused across circuits—namely, the raw messages $\{m_i\}$. The Verifier then compares these commitments across proofs to enforce equality of the underlying messages.

We highlight that the split is by phase rather than by field arithmetic. The `prepare` circuit necessarily includes wrong-field arithmetic (e.g., SHA-256 message hashes over a binary extension field and an ECDSA verification over a prime field), but this cost is amortized by precomputation. We prioritize (i) an efficient `show` relation and (ii) simple linking between `prepare` and `show`. Because equality of Hyrax commitments is checked within a single group, both relations use the same curve. In particular, we target a curve whose scalar field matches the base field of the device-binding signature (e.g., P-256) so that in-circuit verification is efficient and commitment equality checks are defined over the same group. Accordingly, we use Tom256 (T256), whose scalar field equals the base field of P-256.

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

3.1.1 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 message hashes correspond to messages and salts
3. Assert $\text{ECDSA.verify}(\sigma_I, PK_I) = 1$, i.e., the credential signature verifies under the Issuer public key

The backend proving system we use for verifiably computing circuits is Spartan, coupled with a Hyrax-style Pedersen commitment scheme. We can express the circuit computation as some R1CS relation

$$(A \cdot Z) \circ (B \cdot Z) = (C \cdot Z),$$

where $\vec{Z} = (io, 1, \vec{w})$ and io are the public input/outputs. Spartan proves knowledge of a vector Z of length $n := |Z|$ that satisfies the R1CS instance.

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

1. `prepareCommit`: Separates out a column containing only message hashes $\{m_i\}_{i\in[N]}$ in Z and computes an initial Hyrax commitment $c^{(1)} = \{c_i^{(1)}\}_{i\in[\sqrt{n}]}$, which includes a Pedersen commitment to the messages column $c_1^{(1)} = com(m_1, \ldots, m_N; r_1^{(1)}) = g_1^{m_1} \ldots g_N^{m_N} g_{N+1}^{r_1^{(1)}}$ with initial randomness $r_1^{(1)}$.

2. `prepareBatch`:

   (a) Re-randomizes this initial Hyrax commitment to get a batch of commitments $c^{(j)} = \{c_i^{(j)}\}_{i\in[\sqrt{n}]}$, each of which contains a Pedersen commitment to the messages $c_1^{(j)} = com_1^{(1)} \cdot g_{N+1}^{r_1^{(j)} - r_1^{(1)}}$ for all $j \in [m]$, 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 $c^{(j)}$ to produce a batch of proofs $\{\pi^{(j)}_{\texttt{prepare}}\}$ for $j \in [m]$ of the `prepare` relation.

The prover will run `prepareBatch` periodically to both generate re-randomized commitments $c^{(j)}$ and store the randomness for the message column commitment $r_1^{(j)}$ for linking purposes, as well as generate and store batches of issuer-signature proofs $\pi^{(j)}_{\texttt{prepare}}$ that can be used for each presentation.

3.1.2 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. Let us denote this signature by $\sigma_P = \sigma(\text{nonce}; SK_P)$

Again, we 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 $\pi^{(j)}_{\texttt{show}}$ for presentation $j \in [m]$, the Prover will once again separate out the messages into a separate column to compute a Hyrax commitment over the Tom256 curve. In particular, the Prover uses *the same* randomness $r_1^{(j)}$ used during the `prepareBatch` process to compute the Pedersen commitment to the messages column. The verifier will then check that the Pedersen commitment to the messages column for $\pi^{(j)}_{\texttt{show}}$ equals that of proof $\pi^{(j)}_{\texttt{prepare}}$ for circuit C_1.

3.2 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 $\tau \in \mathbb{F}$

We blind the evaluations of z, a, b, c using random elements at the constraint system level. We insert into z at the end of r four random values to construct z'

$$z' = s \| p \| r \| (s_0, s_1, s_2, s_3) \| 1 \| x.$$

Then we insert three constraints into A, B, C to construct modified matrices A', B', C'. These constraints are

$$s_0 \times 0 = 0 \tag{1}$$
$$0 \times s_1 = 0 \tag{2}$$
$$s_2 \times 1 = s_2 \tag{3}$$

These force one random value into each of the MLEs for $a' = A'z'$ etc. In this way, the evaluations of $a'(\rho), b'(\rho), c'(\rho), z'(\rho)$ are independent of the witness w.

3.2.1 Blinding Sumchecks

To blind a sumcheck, we use the technique of Virgo. That is, for an n round, degree d sumcheck, we commit n random univariate masking polynomials of degree d, one for each round of the sumcheck. Then, given the evaluations we verify that the result of the sumcheck equals minus the evaluation is equal to an appropriate linear combination of the masking polynomial evaluations.

The prover first commits to a sequence of d values $\left(m_{i,x}\right)_{i \in [n], x \in \{0,2..d\}}$. Then in each round of the sumcheck, the prover will instead send

$$\mathcal{P} \to \mathcal{V} : \left(m_{i,x} + f_i(x)\right)_{x \in \{0,2..d\}}.$$

From which the verifier can interpolate the "blinded" $f'_i(X)$ and the blinded partial sum $\sigma'_i = f'_i(\alpha_i)$. Once again, we use $f'_i(1) = \sigma_{i-1} - f'_i(0)$ and $\sigma'_0 = \sigma_0$.

Then we replace the final consistency check with a zero knowledge proof π of $m = \big(m_{i,x}\big)_{i\in[n],x\in\{0,2..d\}}$ subject to the relation

$$\mathcal{R} = \left\{ \big(\sigma'_n, F(\alpha); m\big) : \sigma'_n - F(\alpha) = \big\langle m, \ell(\alpha)\big\rangle \right\}.$$

We do this using a zero-knowledge IPA protocol. Note this is not hiding of the evaluation of $F(\alpha)$ or the sumcheck value σ_0.

3.2.2 Security Analysis (Sketch) of zk-Spartan

To show this protocol is zero-knowledge, we argue in three steps. First, we assume that the PCS and IPA protocols are zero-knowledge. Thus, it is sufficient to show that given both (blinded) sumcheck transcripts and the evaluations of the MLEs $a'(\alpha), b'(\alpha), c'(\alpha), z'(\alpha')$ we learn nothing about w.

The Simulator $\mathcal{S}$ first chooses any witness z, a random m vector, and random commitment $cm(m)$. This is possible because the IPA is zero knowledge. Then, $\mathcal{S}$ will choose uniformly random values for all

$$\big(f_i'^{(\mathsf{inner})}(x), g_j'^{(\mathsf{outer})}(x)\big)_{i\in[n],j\in[m],x\in\{0,2..d\}}$$

and for the evaluations $a'(\alpha), b'(\alpha), c'(\alpha)$. $\mathcal{S}$ will calculate the final witness opening as

$$z'(\alpha') = \frac{\sigma'_n - \big\langle m, \ell(\alpha)\big\rangle}{A(\alpha,\alpha') + \chi B(\alpha,\alpha') + \chi^2 C(\alpha,\alpha')}.$$

Then, the verifier can solve a system of linear equations to find a unique (s_0, s_1, s_2, s_3) and construct a random polynomial commitment for the opening of z' to $z'(\alpha')$. This is because the polynomial commitment scheme is zero-knowledge.

Since the verifier performs one linear check, this yields a uniformly random valid transcript for the challenges $(\rho, \alpha, \chi, \alpha')$.

3.3 Security Analysis of OpenAC

Correctness. 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.

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

Zero-knowledge. 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
- $\{com(z_i)\}_{iin[\sqrt{n}]}$ Hyrax commitment to Z, which involves a Pedersen commitment to each of the columns of a $\sqrt{n} \times \sqrt{n}$ matrix representation of $\vec{Z}$
- Transcript from the IPA for opening $Z(r_1, \ldots, r_{\log n})$
- The claimed value of $Z(r_1, \ldots, r_{\log n})$

Since we appended random pads to $\vec{Z}$ in our ZK modification in Section 3.2, the distribution of $Z(r_1, \ldots, r_{\log n})$ 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 Cost Analysis and Benchmarks

4.1 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, $n = |\vec{w}| + io + 1$ 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 the 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 $O(m + n)$.
- **Proof length**: (1) $O(\log n) \cdot |\mathbb{F}|$ length of the sumcheck transcript, (2) $O(\sqrt{n}) \cdot |\mathbb{G}|$ length of the commitment to MLE of Z, (3) $O(\log n) \cdot |\mathbb{G}|$ length of argument opening the 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 multilinear extensions 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 zero-knowledge modifications to Spartan, we can see the asymptotic costs remain the same, as follows:

- **Proof length**: the sumcheck and openings are the same length but just padded, we additionally add $O(\log n)$ size $|\mathbb{G}|$ commitments and an $O(\log \log n)$ sumcheck-relation IPA proof, which still gives a proof length of $O(\sqrt{n})$ group or field elements.

- **Verifier time**: the verifier still needs $O(m)$ to evaluate MLEs of A, B, C, $O(\sqrt{n})$ to open the MLE of Z, and $O(\log n)$ for the sumcheck-relation IPA verification, which still gives $O(m + \sqrt{n})$ runtime.

4.2 Experimental Evaluation

Setup. Experiments were performed using our wallet proof-of-concept implementation (`wallet-unit-poc`, branch `mobile-benchmarks`)[7], with Spartan+Hyrax as the proving backend. For our target deployment setting, we benchmarked on two mobile devices: (i) an iPhone 17 with 8 GB RAM (iOS 18), and (ii) a Pixel 10 Pro. This matches the intended model of mobile provers and a server-side verifiers that preloads the verifying key and serve many concurrent users. In addition, desktop benchmarks across varying payload sizes (Tables 5 and 6) are obtained on a MacBook Pro (M4, 14-core GPU, 24 GB RAM).

The mobile benchmarks for the `show` and `prepare` circuits are summarised in Tables 3 and 4, respectively. The proof sizes reported in Tables 3 and 4 are determined by the circuit and payload size and match the desktop values for a 1920-byte credential in Table 6. We then report scaling behavior across payload sizes in Tables 5 and 6, obtained from the same reference implementation.

Benchmarked operations We evaluated two wallet operations:

- the `prepare` circuit: parses and verifies a signed SD-JWT within the circuit and executes the sum-check protocol over the induced constraint system, without outer commitments. This isolates the cost of in-circuit parsing and constraint satisfaction.
- the `show` circuit: proves possession of an ECDSA signature on a fresh verifier nonce under the device key. The circuit embeds one ECDSA verification and enforces key-ownership binding to the handset.

From the holder's perspective, these two circuits correspond to two wallet operations. When a new credential is added to the wallet, the Prepare circuit is run offline: the issuer's signature is checked, the verifiable credential is parsed, and a reusable precomputed state is produced and stored. This is what we refer to as *precompute* in Table 2. In the desktop benchmarks, this cost is given by the sum of the Prepare *Prove* and *Reblind* times in the 1920-byte row of Table 5; on mobile, it corresponds to the *Proving* and *Reblind* times reported in Table 4.

Each subsequent presentation corresponds to a *prove* step. The wallet takes a previously prepared credential, applies a *reblind* operation to re-randomise the precomputed state, and then runs the Show circuit to produce a fresh proof for the verifier's current policy and nonce. In Table 2, the Prove column aggregates the Show *Prove* and *Reblind* times from the same 1920-byte row of Table 5; on mobile, this quantity is reported as the *Prover time (incl. reblind)* in Table 3.

We report the performance characteristics of the credential circuits evaluated across varying payload sizes, ranging from 1 kB to 8 kB. For each payload size, we measure the execution times for setup, proving, reblinding, and verification. We additionally record the proving key (PK) size, verifying key (VK) size, proof size, and witness size. All results are reported using wall-clock timings obtained from the reference implementation.

Overall, the Prepare circuit shows a nonlinear increase in cost as the payload size grows. Setup time is the dominant contributor, rising from roughly 2.6s at 1 kB to about 16.6s at 8 kB. Prove

[7] `https://github.com/privacy-ethereum/zkID/tree/main/wallet-unit-poc#benchmarks`

Table 3: `show` circuit benchmarks

Device	Proving time	Reblind time	Verifier time	Key setup	Proof size (kB)
iPhone 17	99 ms	30 ms	13 ms	47 ms	40.41
Pixel 10 Pro	340 ms	125 ms	61 ms	122 ms	40.41

Table 4: `prepare` circuit benchmarks

Device	Proving time	Reblind time	Verifier time	Key setup	Proof size (kB)
iPhone 17	2987 ms	856 ms	151 ms	3499 ms	109.29
Pixel 10 Pro	7318 ms	1750 ms	318 ms	9233 ms	109.29

Table 5: Desktop execution times and key sizes for Prepare and Show circuits across payload sizes.

Payload	Prepare Circuit (ms)				Show Circuit (ms)				PK Size (MB)		VK Size (MB)	
(Bytes)	Setup	Prove	Reblind	Verify	Setup	Prove	Reblind	Verify	Prepare	Show	Prepare	Show
1024	2559	1683	382	35	36	77	25	9	252.76	3.45	252.76	3.45
1920	4157	2727	715	74	36	77	25	9	420.05	3.45	420.05	3.45
2048	4384	2934	753	83	36	77	25	9	433.76	3.45	433.76	3.45
3072	6466	4242	1357	119	36	77	25	9	636.35	3.45	636.35	3.45
4096	8529	5282	1374	131	36	77	25	9	836.79	3.45	836.79	3.45
5120	10979	6166	1460	140	36	77	25	9	964.70	3.45	964.70	3.45
6144	12993	8407	2821	280	36	77	25	9	1222.26	3.45	1222.26	3.45
7168	15151	8856	2732	230	36	77	25	9	1382.31	3.45	1382.31	3.45
8192	16559	9614	2683	246	36	77	25	9	1542.35	3.45	1542.35	3.45

Table 6: Desktop proof and witness size scaling across payloads.

Payload	Proof Size (kB)		Witness Size (MB)	
(Bytes)	Prepare	Show	Prepare	Show
1024	75.80	40.41	32.03	0.50
1920	109.29	40.41	64.06	0.50
2048	109.29	40.41	64.06	0.50
3072	175.77	40.41	128.13	0.50
4096	175.77	40.41	128.13	0.50
5120	175.77	40.41	128.13	0.50
6144	308.26	40.41	256.25	0.50
7168	308.26	40.41	256.25	0.50
8192	308.26	40.41	256.25	0.50

and Verify steps scale proportionally, while Reblind remains moderately cheaper but still increases with payload size.

The Show circuit exhibits near constant key sizes across all payload sizes. Setup takes $36ms$, proving $77ms$, verification $9ms$, and reblinding $25ms$. Key sizes are significantly smaller than those of the Prepare circuit.

Proof sizes for the Prepare circuit grow with payload size (from 75 kB to 308 kB), while the Show circuit maintains a constant proof size of 40.41 kB. Witness sizes for Prepare increase linearly with payload, whereas the Show circuit uses a fixed witness of 512.52 kB.

5 Related Work

BBS-based anonymous credentials [BBC+24] have been recommended in public feedback for the EUDI Wallet as a means to ensure that presentations cannot be tracked, linked, or correlated [BBC+24]. These systems treat a credential as a constant-size signature on an attribute vector in pairing-friendly groups, following Boneh–Boyen–Shacham and the BBS+ security proofs of Au–Susilo–Mu [BBS04, ASM06]. Holders generate zero-knowledge proofs that selectively disclose only required attributes or predicates, with each presentation freshly randomized to prevent linkability. This aligns with our reference model in which presentation-side privacy is enforced via per-session, non-repeating outputs.

However, issuance differs from our constraints. Using BBS/BBS+ requires issuers to adopt a pairing-based signing scheme instead of the RSA or ECDSA mechanisms prevalent in existing ID systems. To retain compatibility with standardized curves such as P-256, a pairing-free server-aided variant (BBS#) enables holders to obtain small auxiliary data through an oblivious interaction with an issuer-side helper and later produce non-interactive presentations; the helper data grows linearly with the number of planned presentations [CAHLT25]. Device binding and revocation can be encoded as attributes or verified inside the proof, ensuring transcripts and status queries do not introduce stable identifiers.

Microsoft's Crescent [PPZ24] targets environments in which issuers keep their existing credential formats (e.g., JWT, mDL) and continue using their current signing keys, requiring no issuer-side changes. Its workflow splits into a heavy one-time *Prepare* phase and a lightweight per-session *Show* phase. During Prepare, the wallet verifies the issuer's signature, parses attributes, and constructs two reusable artifacts: (i) a Groth16 proof attesting correct verification and parsing, and (ii) a Pedersen vector commitment enabling selective disclosure. Both artifacts can be re-randomized to ensure unlinkability.

In the Show phase, the wallet re-randomizes these artifacts and includes only the proofs required by the verifier's policy, such as age checks or linking two credentials to the same holder. Device binding can be added by having the secure element sign the verifier's challenge. In reference-system terms, Crescent achieves a two-phase model with reusable offline work and modular predicates while leaving issuers untouched.

The trade-offs are substantial: the Prepare phase is expensive (tens of seconds for JWTs and minutes for mDLs), the system relies on pairing-based Groth16 proofs with a large trusted setup (≈661 MB–1.1 GB [PPZ24, §4]), and the security guarantees are classical only. The Show phase is fast—typically 22–41 ms with ≈1 KB proofs, or around 315 ms when device binding is included [PPZ24, §4].

Google's longfellow. [Fas24] This approach targets ecosystems where issuers already sign credentials using ECDSA on standard curves (e.g., P–256) and hash data with SHA–256. The core difficulty is that proving ECDSA verification in zero knowledge is expensive for conventional proof systems: the arithmetic of P–256 and the bitwise structure of SHA–256 do not align with the fast polynomial techniques (such as NTT-based optimizations) used in many modern ZK libraries.

To address this, the authors design custom circuits for both ECDSA and SHA–256 and employ a layered protocol built on the sum-check technique with a lightweight Reed–Solomon encoding to keep proof sizes manageable. A consistency check ensures that the same hidden signing key is used in both the signature and hashing logic. During presentation, the wallet generates a fresh proof and the device signs the verifier's challenge, providing device binding.

In the reference-system perspective, issuer compatibility is fully preserved, selective disclosure is supported, and device binding is incorporated. The drawback is the absence of a reusable offline phase, meaning each presentation requires generating a full proof. Reported performance is approximately 60 ms to prove a single ECDSA signature and around 1.2 s for a complete mDL presentation on mobile devices [Fas24, §5.3,§6.2], with larger proofs and higher verification overhead than succinct SNARK-based systems with trusted setup.

6 Conclusion

The OpenAC construction provides an efficient instantiation of anonymous credentials that aligns with the high-level requirements of the EU Digital Identity Wallet. It has been realized using multiple proof systems, each with distinct security assumptions and performance trade-offs. This work presents a detailed mapping between these requirements and the generic zkID construction, together with a best-in-class implementation that minimizes online latency during credential presentation while avoiding any trusted setup. Because the design amortizes expensive computation and avoids re-running issuer-verification logic for each presentation, it is especially well-suited for scenarios requiring multi-credential linking.

The OpenAC construction forms a single component within a broader digital-identity ecosystem. Several related, but orthogonal, research directions remain open:

- credibly neutral infrastructure for operating issuer, verifier, and data registries;
- richer and more expressive semantics for verifier request policies;
- support for additional features such as revocation, plausible deniability, and guardianship;
- interoperable and compatible credential data models.

Contributors and Acknowledgement

Ying Tong took primary responsibility for the design and construction of the proposed scheme.

Liam Eagen authored the zero-knowledge modifications to Spartan, and contributed to the proof reblinding construction.

Vikas Rushi led the design, implementation, and benchmarking of the component.

Moven Tsai contributed to the mobile benchmarking and the witness generation optimisation efforts.

Hy Ngo & Janabel Xia contributed to the analysis of EUDI requirements, the preparation of the associated appendix, and the writing and revisions of the manuscript.

We thank Nam and Zoey for extensive revisions to the manuscript and for support that facilitated the completion of this work. We also thank Keewoo for their detailed comments across multiple rounds of internal review. We further thank Oskar, Winderica, Miha, and Vivian for feedback that strengthened the technical content of the paper.

AI tools are extensively used for the non-technical part of the document as well as for the correction of grammar and typos.

References

[ASM06] Man Ho Au, Willy Susilo, and Yi Mu. Constant-size dynamic k-TAA. In Roberto De Prisco and Moti Yung, editors, *SCN 06: 5th International Conference on Security in Communication Networks*, volume 4116 of *Lecture Notes in Computer Science*, pages 111–125, Maiori, Italy, September 6–8, 2006.

[BBC+24] Carsten Baum, Olivier Blazy, Jan Camenisch, Jaap-Henk Hoepman, Eysa Lee, Anja Lehmann, Anna Lysyanskaya, René Mayrhofer, Hart Montgomery, Ngoc Khanh Nguyen, et al. Cryptographers' feedback on the eu digital identity's arf. *Tech. Rep.*, 2024.

[BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. Short group signatures. In Matthew Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 41–55, Santa Barbara, CA, USA, August 15–19, 2004.

[BCTV14] Eli Ben-Sasson, Alessandro Chiesa, Eran Tromer, and Madars Virza. Succinct non-interactive zero knowledge for a von neumann architecture. In Kevin Fu and Jaeyeon Jung, editors, *USENIX Security 2014: 23rd USENIX Security Symposium*, pages 781–796, San Diego, CA, USA, August 20–22, 2014. USENIX Association.

[CAHLT25] Rutchathon Chairattana-Apirom, Franklin Harding, Anna Lysyanskaya, and Stefano Tessaro. Server-aided anonymous credentials. Cryptology ePrint Archive, Paper 2025/513, 2025.

[CDL16] Jan Camenisch, Manu Drijvers, and Anja Lehmann. Anonymous attestation using the strong Diffie Hellman assumption revisited. Cryptology ePrint Archive, Report 2016/663, 2016. `https://eprint.iacr.org/2016/663`.

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

[Eur24] European Commission. European digital identity wallet: Architecture and reference framework (arf). Website, 2024. Accessed 2025-11-06.

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

[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.

[GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 626–645, Athens, Greece, May 26–30, 2013.

[Gol01] Oded Goldreich. *Foundations of Cryptography: Basic Tools*, volume 1. Cambridge University Press, Cambridge, UK, 2001.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 305–326, Vienna, Austria, May 8–12, 2016.

[KS25] Darya Kaviani and Srinath Setty. Vega: Low-latency zero-knowledge proofs over existing credentials. *Cryptology ePrint Archive*, 2025.

[Ped92] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In Joan Feigenbaum, editor, *Advances in Cryptology – CRYPTO'91*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140, Santa Barbara, CA, USA, August 11–15, 1992.

[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.

7 Appendix: EUDI Annex 2 Requirements

Table 7: Requirements directly implemented by OpenAC

Annex 2 ID	Where in OpenAC
Topic 1: Online Identification and Authentication (OIA)	
OIA_01	see Overview in § 2 and details in § 3.1.
OIA_02	see component prepare batches in § 3.1.1 and component linking proof in § 3.1.
OIA_03a	see component predicate in § 3.1 and zkSNARK wrapper in § 2.1.
OIA_03b	see Prepare relation in § 3.1.1 and Show relation in § 3.1.2.
OIA_03c	see Prepare relation in § 3.1.1 and component predicate in § 3.1.
OIA_04	see component predicate in § 3.1 and Show relation in § 3.1.2.
OIA_05	see component predicate in § 3.1 and Overview of Security in § 2.2.1 and Security analysis in § 3.3.
OIA_06	see component predicate in § 3.1.
OIA_07	see component prepare batches in § 3.1.1, zkSNARK wrapper in § 2.1, and component predicate in § 3.1.
OIA_08	see Overview of Security in § 2.2.1, Security analysis in § 3.3, and zkSNARK wrapper in § 2.1.
OIA_09	see Overview of Security in § 2.2.1, Security analysis in § 3.3, and zkSNARK wrapper in § 2.1.
OIA_10	see component predicate in § 3.1 and Show relation in § 3.1.2.
OIA_11	see component predicate in § 3.1 and Show relation in § 3.1.2.
OIA_12	see Prepare relation in § 3.1.1 and Overview of Security in § 2.2.1 and Security analysis in § 3.3.
OIA_13	see Prepare relation in § 3.1.1 and Overview of Security in § 2.2.1 and Security analysis in § 3.3.
OIA_14	see Prepare relation in § 3.1.1 and Overview of Security in § 2.2.1 and Security analysis in § 3.3.
OIA_15	see Prepare relation in § 3.1.1 and Overview of Security in § 2.2.1 and Security analysis in § 3.3.
OIA_16	see Overview of Security in § 2.2.1, Security analysis in § 3.3, and component predicate in § 3.1.
Topic 10: Issuance and Credential Handling (ISSU)	
ISSU_02	see SD-JWT/mDL wrapper in § 2.1 and zkSNARK wrapper in § 2.1.
ISSU_07	see Prepare relation in § 3.1.1 and `prepareCommit` in § 3.1.1.
ISSU_08	see Prepare relation in § 3.1.1 and `prepareCommit` in § 3.1.1.
ISSU_09	see Discussion in § 2.1.1, Overview of Security in § 2.2.1, and Security analysis in § 3.3.

continued on next page

Annex 2 ID	Where in OpenAC
ISSU_10	see Prepare relation in § 3.1.1.
ISSU_12	see SD-JWT/mDL wrapper in § 2.1 and zkSNARK wrapper in § 2.1.
ISSU_12a	see SD-JWT/mDL wrapper in § 2.1 and Proof interface in § 2.1.
ISSU_16	see SD-JWT/mDL wrapper in § 2.1 and Overview in § 2 and details in § 3.1.
ISSU_27	see Show relation in § 3.1.2.
ISSU_33	see SD-JWT/mDL wrapper in § 2.1 and component commitment in § 2.1 and § 3.1.1.
ISSU_33a	see SD-JWT/mDL wrapper in § 2.1 and component commitment in § 2.1 and § 3.1.1.
ISSU_33b	see `prepareCommit` in § 3.1.1 and SD-JWT/mDL wrapper in § 2.1.
ISSU_35a	see `prepareCommit` in § 3.1.1 and Prepare relation in § 3.1.1.
ISSU_37	see component prepare batches in § 3.1.1 and `prepareBatch` in § 3.1.1.
ISSU_38	see Show relation in § 3.1.2.
ISSU_39	see `prepareBatch` in § 3.1.1 and Prepare relation in § 3.1.1.
ISSU_44	see component prepare batches in § 3.1.1 and Show relation in § 3.1.2.
ISSU_58	see Show relation in § 3.1.2.
ISSU_59	see Show relation in § 3.1.2.
ISSU_60	see component predicate in § 3.1 and Proof interface in § 2.1.
ISSU_61	see component predicate in § 3.1 and Proof interface in § 2.1.
ISSU_62	see SD-JWT/mDL wrapper in § 2.1 and zkSNARK wrapper in § 2.1.
ISSU_63	see `prepareCommit` in § 3.1.1 and SD-JWT/mDL wrapper in § 2.1.
ISSU_64	see Proof interface in § 2.1.
Topic 23: PID and (Q)EAA issuance	
Topic 23	covered by Topic 10 components in § 2, § 2.1, § 2.2.1, § 3.1, § 3.1.1, § 3.1.2, § 3.3.
Topic 47: Protocols and interfaces for PID and (Q)EAA issuance	
Topic 47	covered by Topic 10 / Topic 23 components in § 2, § 2.1, § 2.2.1, § 3.1, § 3.1.1, § 3.1.2, § 3.3.
Topic 53: Zero-Knowledge Proofs (ZKP)	
ZKP_01	see component predicate in § 3.1 and Overview of Security in § 2.2.1 and Security analysis in § 3.3.
ZKP_02	see Prepare relation in § 3.1.1 and component predicate in § 3.1.

continued on next page

Annex 2 ID	Where in OpenAC
ZKP_03	see component commitment in § 2.1 and § 3.1.1 and component linking proof in § 3.1.
ZKP_04	see component predicate in § 3.1 and Show relation in § 3.1.2.
ZKP_05	see Prepare relation in § 3.1.1 and Show relation in § 3.1.2.
ZKP_06	see zkSNARK wrapper in § 2.1 and SD-JWT/mDL wrapper in § 2.1.
ZKP_07	see Prepare relation in § 3.1.1, component commitment in § 2.1 and § 3.1.1, Overview of Security in § 2.2.1, and Security analysis in § 3.3.
ZKP_08	see backend modularity in § 2.1 and Overview of Security in § 2.2.1 and Security analysis in § 3.3.
ZKP_09	see Show relation in § 3.1.2 and Proof interface in § 2.1.

Table 8: Requirements implementable by minor extension or modification of OpenAC components

Annex 2 ID	Coverage
Topic 6: Relying Party authentication and User approval	
RPA_01	modify components Prepare relation in § 3.1.1 and Discussion in § 2.1.1.
RPA_01a	modify components Show relation in § 3.1.2 and Proof interface in § 2.1.
RPA_02	modify components Show relation in § 3.1.2 and Component predicate in § 3.1.
RPA_02a	modify components Proof interface in § 2.1 and Component linking proof in § 3.1.
RPA_03	modify components Prepare relation in § 3.1.1 and Show relation in § 3.1.2.
RPA_04	modify components Discussion in § 2.1.1 and Overview of Security in § 2.2.1 and Security analysis in § 3.3.
RPA_05	modify components Proof interface in § 2.1 and Component linking proof in § 3.1.
RPA_06	modify components Component predicate in § 3.1 and Show relation in § 3.1.2.
RPA_06a	modify components Show relation in § 3.1.2 and Proof interface in § 2.1.
RPA_07	modify components Show relation in § 3.1.2 and Component predicate in § 3.1.
RPA_07a	modify components Show relation in § 3.1.2 and Proof interface in § 2.1.
RPA_08	modify components Show relation in § 3.1.2 and Show relation in § 3.1.2.
RPA_09	modify components Component predicate in § 3.1 and Show relation in § 3.1.2.
RPA_10	modify components Proof interface in § 2.1 and Proof interface in § 2.1.
Topic 11: Pseudonyms	
PA_01	modify components Component predicate in § 3.1 and Prepare relation in § 3.1.1.
PA_02	modify components Show relation in § 3.1.2 and Show relation in § 3.1.2.
PA_03	modify components Proof interface in § 2.1 and Proof interface in § 2.1.
PA_04	modify components Prepare relation in § 3.1.1 and Component predicate in § 3.1.
PA_05	modify components Proof interface in § 2.1 and Proof interface in § 2.1.
PA_06	modify components Show relation in § 3.1.2 and Proof interface in § 2.1.
PA_07	modify components Proof interface in § 2.1 and Proof interface in § 2.1.
PA_08	modify components Proof interface in § 2.1 and Proof interface in § 2.1.
PA_08a	modify components Overview of Security in § 2.2.1 and Security analysis in § 3.3 and Proof interface in § 2.1.

continued on next page

Annex 2 ID	Coverage
PA_09	modify components Proof interface in § 2.1 and Proof interface in § 2.1.
PA_10	modify components Show relation in § 3.1.2 and Overview of Security in § 2.2.1 and Security analysis in § 3.3.
PA_11	modify components Show relation in § 3.1.2 and Overview of Security in § 2.2.1 and Security analysis in § 3.3.
PA_12	modify components Show relation in § 3.1.2 and Show relation in § 3.1.2.
PA_13	modify components Show relation in § 3.1.2 and Proof interface in § 2.1.
PA_14	modify components Show relation in § 3.1.2 and Overview of Security in § 2.2.1 and Security analysis in § 3.3.
PA_15	modify components Prepare relation in § 3.1.1 and Overview of Security in § 2.2.1 and Security analysis in § 3.3.
PA_16	modify components Prepare relation in § 3.1.1 and Component predicate in § 3.1.
PA_17	modify components Component prepare batches in § 3.2.1 and Prepare relation in § 3.1.1.
PA_18	modify components `prepareCommit` in § 3.1.1 and Overview of Security in § 2.2.1 and Security analysis in § 3.3.
PA_19	modify components Proof interface in § 2.1 and Overview of Security in § 2.2.1 and Security analysis in § 3.3.
Topic 17: Identity matching	
No HLRs	N/A.
Topic 18: Combined presentations of attributes	
ACP_01	modify components Component predicate in § 3.1 and Component linking proof in § 3.1.
ACP_02	modify components Component commitment in § 2.1 and § 3.1.1 and Show relation in § 3.1.2.
ACP_03	modify components Prepare relation in § 3.1.1 and Show relation in § 3.1.2.
ACP_04	modify components Proof interface in § 2.1 and Proof interface in § 2.1.
ACP_05	modify components Component commitment in § 3 and Component predicate in § 3.1.
ACP_06	modify components `prepareCommit` in § 3.1.1 and `prepareBatch` in § 3.1.1.
ACP_07	modify components Prepare relation in § 3.1.1 and Component prepare batches in § 3.2.1.
Topic 20: Strong User authentication for electronic payments	

continued on next page

Annex 2 ID	Coverage
SUA_01	modify components Show relation in § 3.1.2 and Show relation in § 3.1.2.
SUA_02	modify components Component predicate in § 3.1 and Show relation in § 3.1.2.
SUA_03	modify components Show relation in § 3.1.2 and Prepare relation in § 3.1.1.
SUA_04	modify components Show relation in § 3.1.2 and Component predicate in § 3.1.
SUA_05	modify components Overview of Security in § 2.2.1 and Security analysis in § 3.3 and Show relation in § 3.1.2.
SUA_06	modify components Component predicate in § 3.1 and Proof interface in § 2.1.
Topic 43: Embedded disclosure policies	
EDP_01	modify components Component commitment in § 2.1 and § 3.1.1 and `prepareCommit` in § 3.1.1.
EDP_02	modify components Component predicate in § 3.1 and Proof interface in § 2.1.
EDP_03	modify components Discussion in § 2.1.1 and Overview of Security in § 2.2.1 and Security analysis in § 3.3.
EDP_05	modify components Proof interface in § 2.1 and Proof interface in § 2.1.
EDP_06	modify components Component predicate in § 3.1 and Show relation in § 3.1.2.
EDP_07	modify components Component predicate in § 3.1 and Show relation in § 3.1.2.
EDP_09	modify components `prepareCommit` in § 3.1.1 and SD-JWT wrapper in § 2.1.
EDP_10	modify components `prepareCommit` in § 3.1.1 and Proof interface in § 2.1.
EDP_11	modify components Overview of Security in § 2.2.1 and Security analysis in § 3.3 and Prepare relation in § 3.1.1.
Topic 51: PID or attestation deletion	
PAD_01	modify components Proof interface in § 2.1 and Proof interface in § 2.1.
PAD_02	modify components `prepareCommit` in § 3.1.1 and SD-JWT wrapper in § 2.1.
PAD_03	modify components Component predicate in § 3.1 and Show relation in § 3.1.2.
PAD_04	modify components Show relation in § 3.1.2 and Show relation in § 3.1.2.
PAD_05	modify components Overview of Security in § 2.2.1 and Security analysis in § 3.3 and Component commitment in § 2.1 and § 3.1.1.
PAD_06	modify components Component prepare batches in § 3.1.1 and Show relation in § 3.1.2.
Topic 52: Relying Party intermediaries	

continued on next page

Annex 2 ID	**Coverage**
RPI_01	modify components Proof interface in § 2.1 and Overview in § 2 and details in § 3.1.
RPI_02	N/A.
RPI_03	modify components Proof interface in § 2.1 and Proof interface in § 2.1.
RPI_04	modify components Prepare relation in § 3.1.1 and Overview of Security in § 2.2.1 and Security analysis in § 3.3.
RPI_05	modify components Proof interface in § 2.1 and Proof interface in § 2.1.
RPI_06	modify components Proof interface in § 2.1 and Show relation in § 3.1.2.
RPI_06a	modify components Proof interface in § 2.1 and SD-JWT wrapper in § 2.1.
RPI_07	modify components Proof interface in § 2.1 and Show relation in § 3.1.2.
RPI_07a	modify components Prepare relation in § 3.1.1 and Show relation in § 3.1.2.
RPI_07b	modify components Overview in § 2 and details in § 3.1 and Show relation in § 3.1.2.
RPI_08	modify components Proof interface in § 2.1 and Overview of Security in § 2.2.1 and Security analysis in § 3.3.
RPI_09	modify components Prepare relation in § 3.1.1 and Component linking proof in § 3.1.
RPI_10	modify components Overview of Security in § 2.2.1 and Security analysis in § 3.3 and Proof interface in § 2.1.

Table 9: Requirements requiring integration with external systems or protocol adaptations

Annex 2 ID	Coverage
Topic 2: Mobile Driving Licence (mDL) within the EUDI Wallet ecosystem	
Topic 2	
Topic 3: PID Rulebook	
Topic 3	
Topic 4: mDL Rulebook	
Topic 4	
Topic 7: Attestation revocation and revocation checking	
Topic 7	
Topic 9: Wallet Instance Attestation / Wallet Unit Attestation	
Topic 9	
Topic 10: Issuance and Credential Handling (ISSU) (continued)	
ISSU_01	Protocol interoperability (OpenID4VCI profile); outside OpenAC components.
ISSU_01a	Provider-side protocol interoperability (OpenID4VCI); outside OpenAC.
ISSU_03	API-level interoperability (W3C Digital Credentials API); outside OpenAC.
ISSU_04	Protocol feature: batch issuance in OpenID4VCI; outside OpenAC.
ISSU_05	Wallet activation flow; wallet runtime responsibility, not OpenAC.
ISSU_06	Post-issuance content check by wallet; outside OpenAC.
ISSU_11	User-approval and display handled by wallet UX; outside OpenAC.
ISSU_11b	Failure handling (delete and notify) by wallet; outside OpenAC.
ISSU_12b	Per-credential keypair generation (device/HSM); key management, not OpenAC.
ISSU_12c	Expiry alignment (PID *leq* WUA expiry); lifecycle rule, not OpenAC.

continued on next page

Annex 2 ID	Coverage
ISSU_12d	Expiry alignment for attestations (if revocation chaining); lifecycle rule, not OpenAC.
ISSU_13-15	Rulebook compliance and OpenID4VCI support across Wallet/Provider; governance/protocol layer.
ISSU_17-20	Device binding, high-assurance identification, trusted lists, published support; provider operations, not OpenAC.
ISSU_21-23b	Trusted lists and signed issuer metadata; wallet verifies; governance/protocol.
ISSU_24-24a	Wallet validates PID Provider access/registration before requesting; refuse if untrusted; wallet policy.
ISSU_25-26	Attestation Providers: Rulebook compliance and OpenID4VCI support; provider-side.
ISSU_27a-27c	Attestation Provider subject/eligibility verification; provider-side.
ISSU_28-32	Registration/trusted-lists/metadata publication; governance/protocol (ISSU_31 empty).
ISSU_34-34a	Wallet checks provider authorisation/entitlement for requested type; block/warn if invalid.
ISSU_35-35b	Provider-side privacy (unique elements minimisation/discard); policy/operations.
ISSU_36	Wallet-side minimisation/scope enforcement; wallet policy.
ISSU_40-43	Anti-linkability methods wallet uses provider's preference if supported; user impact minimal; batch semantics where applicable; protocol/UX.
ISSU_45-50	Privacy-preserving handling during issuance flows; operational/protocol behaviours, not OpenAC components.
ISSU_51-57	General issuance lifecycle policies beyond OpenAC scope (retries, scheduling, operational controls); wallet/provider operations.
ISSU_65	Re-issuance binding continuity; protocol-layer control.

Topic 16: Signing documents with a Wallet Unit
Topic 16

Topic 24: User identification in proximity scenarios
Topic 24

Topic 25: Unified definition and controlled vocabularies for attributes
Topic 25

continued on next page

Annex 2 ID	Coverage
Topic 26: Catalogue of attestations	
Topic 26	
Topic 27: Registration of PID Providers, Providers of QEAAs, PuB-EAAs, and non-qualified	
Topic 27	
Topic 30: Interaction between Wallet Units	
Topic 30	
Topic 31: Notification and publication of PID Provider / Wallet Provider / Attestation Provider trust status	
Topic 31	
Topic 33: Wallet Unit backup and restore	
Topic 33	
Topic 35: PID issuance service blueprint	
Topic 35	
Topic 37: QES / Remote Signing - Technical Requirements	
Topic 37	
Topic 38: Wallet Unit revocation	
Topic 38	
Topic 39: Wallet-to-wallet technical topic	
Topic 39	

continued on next page

Annex 2 ID	Coverage
Topic 40: Wallet Instance installation / activation / management	
Topic 40	
Topic 44: Registration certificates for PID Providers, QEAAs, PuB-EAAs	
Topic 44	
Topic 48: Blueprint for requesting data deletion to Relying Parties	
Topic 48	

Table 10: Requirements dependent on product, user interface, or user-experience considerations

Annex 2 ID	**Coverage**
Topics 5, 8, 13, 14, 15, 21, 22, 36	There are no HLRs for this Topic.
Topics 12, 32, 41, 45, 46	Refers to the attestation rulebook
Topics 24, 30	Refers to wallet instance requirements
Topic 28	Refers to PID rulebook
Topic 29	Refers to natural person PID
Topic 33	Refers to functional requirements of back up and restore function of wallet instance
Topic 42	Refers to QTSP requirements
Topic 50	Refers to compliance requirements of the wallet provider and wallet instance