Consensix Labs

Selective Disclosure for Blockchain-Anchored Professional Credentials

A Field-Level Selective Disclosure Extension to the W3C Verifiable Credentials Protocol of Decentralized Professional Credentials, Implemented on Ethereum and IOTA

Table of Contents

Executive Summary

The companion protocol paper, *Decentralized Professional Credentials*, established a chain-agnostic way to issue, anchor, and verify W3C Verifiable Credentials: issuers sign credentials as JWTs, a SHA-256 hash is anchored on-chain for existence and revocation, and verifiers check the signature and on-chain status independently. That protocol has one deliberate gap, named in its own Section 3 and deferred to its Section 9: presenting a credential means presenting all of it. A job applicant who wants to prove only where they worked must also reveal their role and dates; a certification holder who wants to prove only that they are certified must also reveal the level.

This paper closes that gap with **field-level selective disclosure**. The holder chooses which individual items to reveal at presentation time; everything else stays hidden, and the verifier learns only that some items were withheld, never their names or values. The scheme is structured after SD-JWT (Selective Disclosure for JWTs, standardized as RFC 9901) and reuses the base protocol's Ed25519 signing and `did:key` identifiers unchanged.

The central result is that selective disclosure is added **without touching the on-chain registry, the smart contracts, or the verification flow**, exactly as the base paper's Section 9 anticipated. This works because the on-chain anchor is computed over the signed JWS only, which is fixed at issuance and identical in every presentation derived from a credential. A plain credential, having no disclosures, hashes exactly as it did before, so the extension is fully backward compatible. An optional, per-presentation holder binding defends against replay. The proof of concept is validated end-to-end on both chains from the base PoC – Ethereum (Solidity) and IOTA Rebased (Move) – and adds no measurable on-chain cost, because the anchored value is the same 32-byte hash regardless of how many claims are disclosed.

This paper is a companion to *Decentralized Professional Credentials* and assumes familiarity with it. The base paper is required reading: the data model, signing, DID method, smart contracts, and verification flow are established there and are not repeated here.

1. Introduction

The Gap

The base protocol's privacy posture is strong at the storage layer and weak at the presentation layer. Credential content never touches the blockchain – only hashes are anchored – so nothing private is ever published on-chain. But when a holder presents a credential to a verifier, the unit of disclosure is the entire signed credential. The base paper states this plainly in its limitations (Section 3): "presenting a credential means presenting all of it. You cannot prove you worked at a specific company without also revealing your job title and dates."

This is the ordinary case in hiring, licensing, and background checks, where the verifier needs to confirm one fact and the holder would prefer not to hand over the rest. The base protocol's own Section 9 frames the desired end state precisely: prove employment at a company without revealing salary or exact dates; prove a valid certification without revealing its level. Those two examples map directly onto the demonstration scenarios in this paper.

What This Adds, and What It Does Not

This extension adds **field-level selective disclosure**: at presentation time the holder reveals a chosen subset of the credential's claims and withholds the rest. For a credential's array-valued claims (for example, a list of endorsed skills), individual elements can be revealed independently.

The scope is deliberately bounded. This work does **not** add predicate or range proofs – proving a property of a hidden value without revealing it. "Prove the certification is valid without revealing its level" is in scope (reveal the name and body, hide the level). "Prove at least two years of tenure without revealing the dates" is **not**: that is a predicate over hidden values and requires machinery (zero-knowledge range proofs) outside this design. Section 7 discusses why, and Section 9 treats it as future work. Keeping the scope at field-level reveal/hide is what allows the extension to reuse the existing signature scheme and leave the on-chain layer untouched.

Relationship to the Base Protocol

Everything the base paper establishes is reused unchanged: the W3C VC Data Model v2.0 payload, JWT signing with Ed25519, `did:key` resolution, the smart contracts on both chains, the open-issuer trust model, and the two-step verification (signature plus on-chain status). The base paper also made two forward-looking design choices that this extension depends on. First, its credential data model uses **flat `credentialSubject` fields** specifically so that "individual fields can be disclosed or withheld independently when selective disclosure is added" (base paper, Section 4). Second, its anchoring step hashes a fixed, signed artifact. This paper builds directly on both. Readers should treat the base paper as a prerequisite; the sections below assume its protocol, architecture, and terminology.

2. Design Space

Selective disclosure for verifiable credentials has several established approaches, with materially different trust, performance, and complexity profiles. The base paper's background section already contrasts the broad credential landscape (Blockcerts, AnonCreds, EAS, and centralized platforms); this section narrows to the specific question of *how to disclose selectively* and explains the choice made here.

Hash-commitment / SD-JWT. The issuer replaces each disclosable claim with a salted hash (a digest), signs the credential containing the digests, and gives the holder both the signed credential and the set of salts-and-values ("disclosures"). To present, the holder forwards the signature plus only the disclosures they choose; the verifier recomputes digests and matches them against the signed set. SD-JWT (RFC 9901) is the standardized realization of this idea for JWTs. It reuses ordinary signatures (here, Ed25519), so it is simple to implement on existing infrastructure. Its main privacy limitation is that the issuer's signature is constant across presentations, so multiple presentations of the same credential are linkable.

BBS+ signatures. A signature scheme that natively supports deriving, from one signed credential, a zero-knowledge proof that reveals only selected attributes – and, importantly, is unlinkable across presentations. This is the strongest privacy option, but it requires a different signature primitive (pairing-friendly curves such as BLS12-381), which would mean abandoning the base protocol's Ed25519/`did:key`/PyJWT foundation, and mature, well-audited tooling in Python is limited.

Zero-knowledge proofs (general). Systems such as AnonCreds use ZKPs to prove statements about hidden attributes, including predicates. This is the most expressive and most privacy-preserving option and the only one that supports range/predicate proofs, but it is also the most complex to implement and reason about, and it is typically tied to a specific ecosystem.

Plain Merkle commitments. Each field is hashed, the Merkle root is anchored, and a field is revealed with a Merkle proof. This is conceptually close to SD-JWT but anchors per-credential structure on-chain and does not integrate as cleanly with the JWT/JWS tooling the base protocol already uses.

The choice here is a **self-contained, SD-JWT-structured scheme**. It is worth being precise about the tradeoff this represents, because it is easy to overstate. SD-JWT is itself a hash-commitment scheme; the genuine fork in the road is hash-commitment versus BBS+ versus general ZKP, not "SD-JWT versus hash commitments." Among those, the hash-commitment route is the only one that preserves the base protocol's signing scheme and on-chain design intact. It is implemented in-house rather than via an external SD-JWT library so that every step is inspectable and the dependency surface stays at the base protocol's existing PyJWT/cryptography stack; the cost is that the implementation follows SD-JWT's structure without claiming interoperability with third-party SD-JWT verifiers. What this route gives up relative to BBS+ – multi-show unlinkability – and relative to ZKP – predicate proofs – is stated honestly in Section 7.

3. Design

Disclosable and Always-Visible Claims

A credential is split into two parts. The **always-visible** fields are those a verifier needs in order to make sense of any presentation at all: the `@context`, `type`, `issuer`, the validity window (`validFrom` / `validUntil`), and the subject identifier `credentialSubject.id` (the holder's `did:key`). These remain in the clear in every presentation. Everything else in `credentialSubject` – the actual claims – is **disclosable**: hidden by default and revealed only at the holder's choice.

Digests and Array Placeholders

Disclosable claims are encoded following SD-JWT's conventions. For each object claim, the issuer constructs a **disclosure** – an array of `[salt, claim_name, claim_value]`, serialized and base64url-encoded – and places its SHA-256 digest in a `_sd` array inside `credentialSubject`. A fresh 128-bit salt per disclosure makes a hidden value unguessable: without it, a verifier could confirm a suspected value by hashing a guess. Array-valued claims are handled element-by-element: each element becomes a `{"...": <digest>}` placeholder in the array, with its own disclosure, so a holder can reveal a subset of the elements.

Concretely, an employment credential's subject is transformed at issuance from this:

```
{
  "id": "did:key:z6Mk...",
  "employerName": "Acme Corp",
  "role": "Senior Engineer",
  "startDate": "2022-03-01",
  "endDate": "2024-06-30"
}
```

into this signed form (digests abbreviated), accompanied by four disclosures held by the holder:

```
{
  "id": "did:key:z6Mk...",
  "_sd": ["q1w2e3...", "r4t5y6...", "u7i8o9...", "p0a1s2..."],
  "_sd_alg": "sha-256"
}
```

The holder, possessing all four disclosures, can reconstruct the full credential for their own use. A verifier sees only the disclosures the holder forwards.

The Combined Format

A credential and its disclosures travel together in SD-JWT's combined serialization: the signed JWS, followed by each disclosure, joined by `~`, with an optional trailing key-binding token (Section 4):

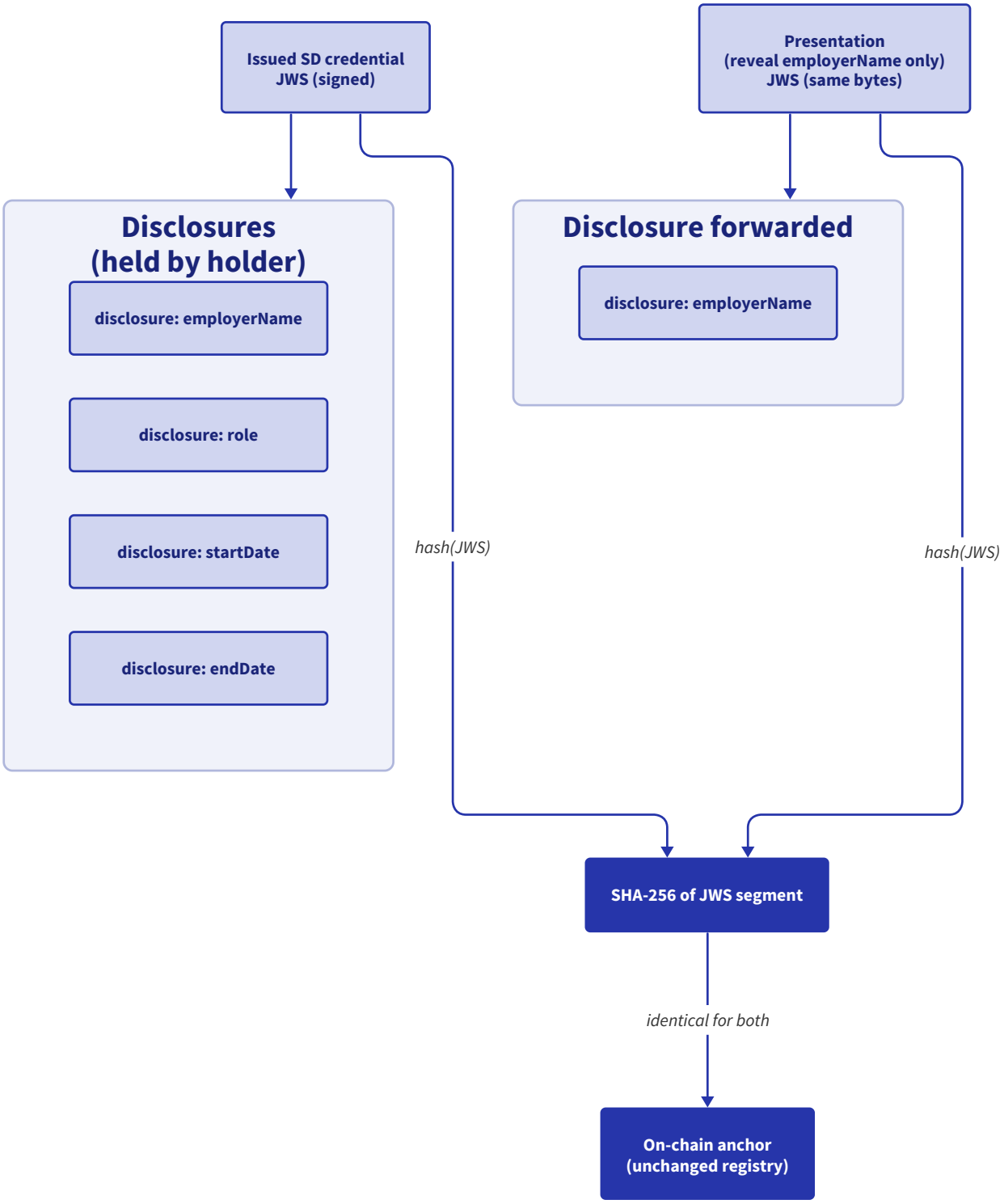
```
<JWS>~<disclosure_1>~ ... ~<disclosure_n>~[<KB-JWT>]
```

The `~` separator is safe because base64url – used for both the JWS and the disclosures – never contains it. To present a subset, the holder simply omits the disclosures for the claims they wish to withhold; the JWS is forwarded unchanged.

The Stable Anchor

This is the design's keystone. The base protocol anchors "the SHA-256 hash of the complete signed JWT string" (base paper, Section 4). This extension refines that definition to **the SHA-256 hash of the JWS segment only** – everything before the first `~`. The consequences follow directly:

- The JWS is fixed at issuance and is byte-for-byte identical in the issued credential and in every presentation derived from it, regardless of which claims are disclosed. So the anchor is the same in all of them.
- A plain (non-selective) credential contains no `~`, so the JWS segment *is* the whole token, and its hash is identical to the base protocol's. The change is therefore fully backward compatible: existing credentials and the contracts that hold them are unaffected.
- Because the anchor never depends on the disclosed set, the on-chain registry, both smart contracts, and the verification flow require **no changes whatsoever**. This realizes the base paper's Section 9 claim that selective disclosure "builds on the credential foundation established here without requiring changes to the on-chain contracts or the verification protocol."

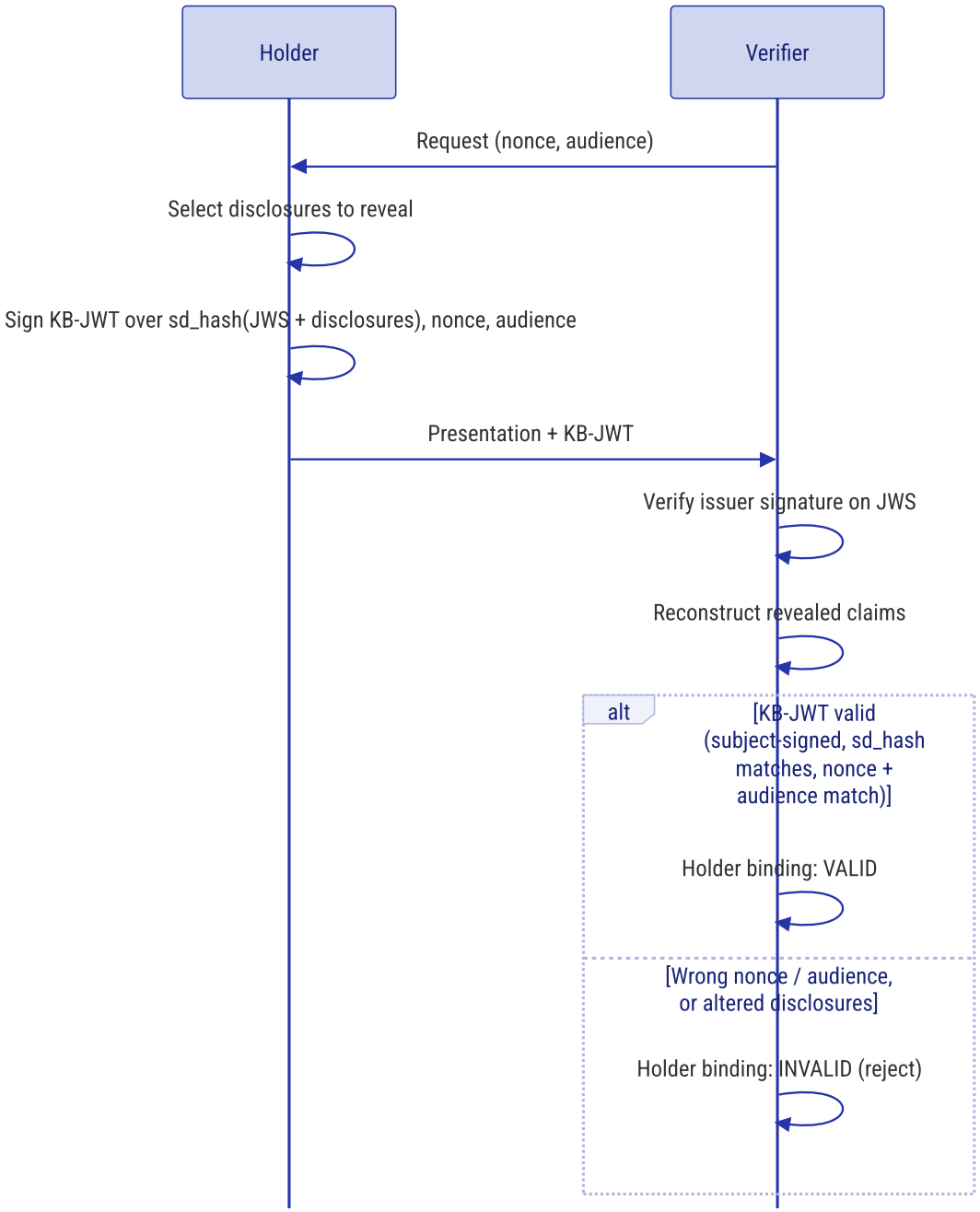

Issued SD credential
JWS (signed)
Presentation
(reveal employerName only)
JWS (same bytes)
Disclosures
(held by holder)
disclosure: employerName
disclosure: role
disclosure: startDate
disclosure: endDate
Disclosure forwarded
disclosure: employerName
hash(JWS)
hash(JWS)
SHA-256 of JWS segment
identical for both
On-chain anchor
(unchanged registry)

4. Holder Binding

A bare presentation is a bearer token: anyone who intercepts it can replay it to another verifier. The base protocol does not address this (its credentials are always presented whole, and binding was out of scope). This extension adds an **optional** holder binding, modeled on SD-JWT's Key Binding JWT (KB-JWT) and enabled per presentation.

When binding is requested, the holder appends a short JWT signed with their **own** key (the subject `did:key`), committing to a hash (`sd_hash`) of the exact presentation being made – the JWS plus the specific disclosures included – together with a verifier-supplied `nonce` and `audience`. A verifier that issued a nonce can then confirm three things: the binding was signed by the credential's subject, it commits to this exact set of disclosures, and it carries the expected nonce and audience. A presentation captured and replayed to a different verifier fails the nonce/audience check; a presentation whose disclosures are altered after binding fails the `sd_hash` check.

Binding is a toggle, not a requirement, because not every interaction needs it: a holder posting a one-time proof to an open audience may not want to bind to a single verifier, whereas a holder responding to a specific verifier's challenge should. Verification reports the binding state independently of the rest of the result, and a verifier that demanded binding treats a missing or failed binding as an overall failure (Section 6).


Holder
Verifier
Request (nonce, audience)
Select disclosures to reveal
Sign KB-JWT over sd_hash(JWS + disclosures), nonce, audience
Presentation + KB-JWT
Verify issuer signature on JWS
Reconstruct revealed claims
alt
[KB-JWT valid (subject-signed, sd_hash matches, nonce + audience match)]
Holder binding: VALID
[Wrong nonce / audience, or altered disclosures]
Holder binding: INVALID (reject)

5. Implementation

All selective-disclosure logic lives in a single self-contained module, kept dependency-light (it relies only on the base protocol's existing PyJWT and `did` module). It provides disclosure construction and digesting, the transformation of a flat claims dictionary into the `_sd` /placeholder form, reconstruction of revealed claims from a provided disclosure set, the combined-format parse/assemble, and the holder-binding sign/verify. The base protocol's `credential` module orchestrates: issuance gains a selective path that builds the SD subject and then signs exactly as before, and the anchoring helper is refined to hash the JWS segment (Section 3). The DID module, the signing scheme, the smart contracts, and the chain backends are untouched.

The user-facing surfaces mirror the base protocol's existing ones. The CLI gains a `--selective` flag on issuance, a `derive` command for the holder to produce a presentation (choosing fields, optionally binding to a nonce/audience), and a presentation-aware `verify`. The REST API and the web interface gain the corresponding capabilities: an issuance toggle, a presentation builder that lets the holder tick which claims to reveal, and a verifier view that reports the revealed claims, the count of withheld items, and the binding state alongside the unchanged on-chain status.

One implementation detail discovered during dual-chain validation is worth recording because it is a base-protocol correctness fix, not a selective-disclosure concern: IOTA Rebased derives an address as the Blake2b-256 hash of the raw Ed25519 public key, with no scheme-flag prefix – unlike Sui, which prepends the flag. The base IOTA backend used the Sui form, which produced a deployer address the network never recognized as the transaction signer. The fix belongs in the base repository's backend and is carried here as well.

6. Security Properties and What a Verifier Learns

The extension preserves the base protocol's guarantees (issuer-signed authenticity, on-chain existence and revocation) and adds disclosure-specific ones.

No claim injection. Reconstruction honors only digests that appear in the issuer-signed `_sd` array or array placeholders. A disclosure for a claim the issuer never signed has no matching digest, so it cannot smuggle a forged claim into a presentation. The scheme goes further and *rejects* any presentation carrying a disclosure that matches nothing in the signed credential, or a duplicate disclosure, per the SD-JWT verifier rules – a malformed disclosed set is reported distinctly from a bad signature.

Tamper and replay defense. Altering the JWS invalidates the issuer signature. Altering, adding, or removing disclosures in a bound presentation breaks the `sd_hash` the holder committed to. Replaying a bound presentation to a different verifier fails the nonce/audience check. A verifier that required binding but received none flags the presentation as failing.

Algorithm pinning. Both the issuer JWS and the holder's KB-JWT are verified with the algorithm fixed to EdDSA, closing algorithm-confusion and `alg:none` attacks. The advertised digest algorithm (`_sd_alg`) is checked against the one supported, so an unsupported value fails loudly rather than silently reconstructing as if everything were withheld.

What the verifier learns – and does not. From a presentation, a verifier learns the always-visible fields, the values of the revealed claims, and a count of how many items were withheld. It does **not** learn the names or values of withheld object claims. The honest exception, stated as a limitation in Section 7, concerns array structure.

7. Limitations

In keeping with the base protocol paper's practice of stating limitations alongside the affirmative case, the following are inherent to this design and are not defects.

Array structure is partially visible. Hidden object claims reveal neither name nor value (they are opaque digests in `_sd`). Array claims are weaker: the field name and the element *count* remain visible in the signed subject, because each element is a placeholder in a named array. Revealing one of several skills tells the verifier how many were withheld, and a fully hidden array still discloses that the field exists and how many elements it holds. This is inherent to per-element array disclosure; hiding the existence of an array entirely would require treating the whole field as a single object claim, at the cost of per-element selectivity.

Multi-show linkability. Because the issuer's signature on the JWS is constant, two presentations derived from the same credential carry the same JWS and are therefore linkable to one another (and to the on-chain anchor). This scheme provides selective disclosure, not unlinkability. Unlinkability across presentations requires BBS+ or a ZKP-based scheme (Section 2), which would mean changing the signature primitive.

No predicate or range proofs. As scoped in Section 1, the holder can reveal or hide whole claims but cannot prove a property of a hidden value (for example, a minimum tenure without dates). Such proofs require zero-knowledge range proofs and are out of scope.

Self-contained, not interoperable. The implementation follows SD-JWT's structure but is in-house and uses a single fixed disclosure serialization; it does not claim interoperability with third-party SD-JWT or SD-JWT VC verifiers. The SD-JWT VC profile (draft-ietf-oauth-sd-jwt-vc) remains an Internet-Draft at the time of writing; interoperating with it is future work.

Inherited base limitations. Everything the base protocol notes still applies: the blockchain proves what was recorded, not that it is accurate (the oracle problem); `did:key` has no key rotation; trust bootstrapping is the verifier's responsibility; and only hashes, never content, are anchored on-chain.

8. Evaluation

The proof of concept was validated end-to-end on both chains from the base PoC, using local test networks (the Ethereum and IOTA Local Testing Starter Packs), via automated scenario scripts. Each scenario issues a selectively-disclosable credential, derives a presentation revealing a chosen subset, and verifies it on-chain. Five scenarios exercise the protocol and its failure modes:

#	Scenario	Reveals	Withholds	Demonstrates
1	Employment	`employerName` only	role, department, start/end dates, employment type (5 items)	Single-field reveal + holder binding, with a wrong-nonce replay check
2	Certification	name + body	level, dateAwarded (2 items)	Prove a credential while hiding a sensitive field
3	Endorsement	endorser + one skill	relationship, statement, context, one skill (4 items)	Subset of an array claim
4	Revocation	role only	employer, start/end dates (3 items)	The same presentation verifies VALID, then REVOKED after the issuer revokes the underlying credential
5	Tamper	–	–	A presentation with an altered JWS is rejected at the signature check

Scenarios 1 and 2 are precisely the examples the base paper's Section 9 used to motivate selective disclosure: proving employment at a company without revealing the dates, and proving a certification without revealing its level. Scenario 4 is the sharpest demonstration of the stable-anchor design: because revocation operates on the JWS hash, the *same* presentation file flips from VALID to REVOKED with no change to the presentation itself.

On-chain cost is unchanged from the base protocol. Because the anchored value is the same 32-byte hash regardless of how many claims are disclosed, selective disclosure adds no on-chain overhead. Registration and revocation costs match the base protocol's measurements on both chains:

Operation	EVM (gas)	IOTA (NANOS)
Register	~140,242	3,994,400
Revoke	~31,976	1,000,000
Verify (incl. presentation)	0 (view call)	0 (direct read)

The verifier's selective-disclosure checks – reconstruction, withheld counting, and holder binding – are entirely off-chain and add no transaction cost. As in the base protocol, the cost structure is consistent across both chains, and the same caveat applies: these are local-network figures, and IOTA mainnet anchoring is effectively feeless.

The dual-chain runs are otherwise identical in behavior, differing only in the chain-specific surface the base paper already documented (EVM gas versus IOTA NANOS; `bytes32` versus the IOTA `PACKAGE_ID:REGISTRY_ID` address format; second-versus-millisecond timestamps). Full transcripts of both runs are in Appendix B.

9. Conclusion and Future Work

This extension delivers the capability the base protocol explicitly deferred: a holder can now prove individual claims and withhold the rest, with optional binding to a specific verifier, and – the point of the design – without any change to the on-chain registry, the smart contracts, or the verification flow. The stable-anchor refinement (hash the JWS, not the whole token) is what makes that possible while remaining backward compatible with existing credentials.

Three directions follow naturally. **Predicate and range proofs** would let a holder prove properties of hidden values (minimum tenure, certification not expired) without revealing them; this requires zero-knowledge techniques beyond hash commitments. **Unlinkability** – so that repeated presentations of a credential cannot be correlated – would require a BBS+ signature scheme in place of Ed25519, a larger change that trades the current simple foundation for stronger privacy. **Standards interoperability** with the emerging SD-JWT VC profile would let credentials issued here be consumed by third-party verifiers, at the cost of conforming to that draft's serialization and metadata rules. The base protocol's own future-work items – batch (Merkle) anchoring and status-list revocation for high-volume issuers – compose with this extension unchanged, since neither touches how a credential is disclosed.

Appendix A: Changes Relative to the Base Implementation

The extension is additive. New: a self-contained selective-disclosure module, a presentation-builder component in the web interface, and selective-disclosure scenario scripts for each chain. Modified: the credential module (selective issuance, presentation derivation and verification, and the JWS-segment anchoring refinement), and the CLI, REST API, and web views (issuance toggle, `derive`, presentation-aware verification). Unchanged: the DID module, the signing scheme, both smart contracts, the chain backends (apart from the IOTA address-derivation correctness fix noted in Section 5, which belongs to the base repository), and the entire on-chain registry and verification flow.

The implementation is published as a fork of the base project at https://github.com/Consensix-Labs/Decentralized-Professional-Credentials-with-Selective-Disclosure.

Appendix B: Full Scenario Output

The following are complete, unedited transcripts of the automated selective-disclosure scenario scripts run against local test networks. They are the reproducibility record for the results in Section 8.

B.1: EVM Scenarios

```
============================================================
  Setup: Generating Keys
============================================================

→ Cleaning up previous run artifacts...
→ Generating issuer keys (employer, certbody, endorser)...
Generated keypair: employer
  DID:  did:key:z6MkfBoFx3UdntT9ih21eCPJckjm4KMMGpTy5JpR3ZLo5ckE
  Saved to: /app/keys/employer.json
Generated keypair: certbody
  DID:  did:key:z6Mkh5R6KARHaCwjXaCbAeRyhKka3DRFABd9b85MgHczF8dY
  Saved to: /app/keys/certbody.json
Generated keypair: endorser-bob
  DID:  did:key:z6MkrCzBUnzKPdKm4bvY3S4JowNhuivfHLraX4tu4aeFV5tX
  Saved to: /app/keys/endorser-bob.json
→ Generating holder key (Alice)...
Generated keypair: alice
  DID:  did:key:z6MkhQJ2uoi43BTY9ZVpYR5ZZckvCKUxo3H5zekTp7C2RAZN
  Saved to: /app/keys/alice.json
  Alice's DID: did:key:z6MkhQJ2uoi43BTY9ZVpYR5ZZckvCKUxo3H5zekTp7C2RAZN

============================================================
  Setup: Deploying Contract
============================================================

→ Deploying CredentialRegistry on evm...
Deploying CredentialRegistry on evm...
Contract deployed at: 0xb439af398d01428f4eD2f10402E033A7a4E71DeB

Add to your .env file:
  EVM_CONTRACT_ADDRESS=0xb439af398d01428f4eD2f10402E033A7a4E71DeB
→ Contract address: 0xb439af398d01428f4eD2f10402E033A7a4E71DeB

============================================================
  Scenario 1: Employment -- reveal only the employer
============================================================

→ Issuing SD employment credential...
Issuer: did:key:z6MkfBoFx3UdntT9ih21eCPJckjm4KMMGpTy5JpR3ZLo5ckE
Credential: Employment: Senior Engineer at Acme Corp (2022-03-01 to 2024-06-30)  [selective disclosu
Hash: 0x68928ebae8cca34ad036c5e699c7a496ac16a9cf2280c44a1bfdc38c09555e43
Saved to: credentials/employment_68928eba.jwt
→ Registering the credential on-chain (issuer)...
Credential: Employment: Senior Engineer at Acme Corp (2022-03-01 to 2024-06-30)
Hash: 0x68928ebae8cca34ad036c5e699c7a496ac16a9cf2280c44a1bfdc38c09555e43
Issuer DID: did:key:z6MkfBoFx3UdntT9ih21eCPJckjm4KMMGpTy5JpR3ZLo5ckE
```

```
Expiration: none
Registering on evm...
Registered. Gas used: 140242
→ Holder derives a presentation revealing ONLY employerName, bound to a verifier...
Revealed: employerName
Presentation created (with holder binding, nonce=verifier-nonce-001, audience=hr@example.com).
Saved to: presentations/presentation_68928eba.jwt
→ Verifier checks the presentation with the matching nonce (expect VALID + binding VALID)...
Checking signature...
  Signature: VALID
  Credential: Employment: ? at Acme Corp (? to ?)
  Issuer: did:key:z6MkfBoFx3UdntT9ih21eCPJckjm4KMMGpTy5JpR3ZLo5ckE
  Holder: did:key:z6MkhQJ2uoi43BTY9ZVpYR5ZZckvCKUxo3H5zekTp7C2RAZN
  Disclosure: selective -- revealed employerName
  Withheld items: 5
  Holder binding: VALID

Checking on-chain status (evm)...
  Hash: 0x68928ebae8cca34ad036c5e699c7a496ac16a9cf2280c44a1bfdc38c09555e43
  Registered: yes (at block timestamp 1780743165)
  Issuer (on-chain): did:key:z6MkfBoFx3UdntT9ih21eCPJckjm4KMMGpTy5JpR3ZLo5ckE

  Result: VALID
→ Replay defense: a different verifier replays it with the WRONG nonce (expect binding INVALID)...
Checking signature...
  Signature: VALID
  Credential: Employment: ? at Acme Corp (? to ?)
  Issuer: did:key:z6MkfBoFx3UdntT9ih21eCPJckjm4KMMGpTy5JpR3ZLo5ckE
  Holder: did:key:z6MkhQJ2uoi43BTY9ZVpYR5ZZckvCKUxo3H5zekTp7C2RAZN
  Disclosure: selective -- revealed employerName
  Withheld items: 5
  Holder binding: INVALID (nonce mismatch (possible replay))

Checking on-chain status (evm)...
  Hash: 0x68928ebae8cca34ad036c5e699c7a496ac16a9cf2280c44a1bfdc38c09555e43
  Registered: yes (at block timestamp 1780743165)
  Issuer (on-chain): did:key:z6MkfBoFx3UdntT9ih21eCPJckjm4KMMGpTy5JpR3ZLo5ckE

  Result: BINDING_FAILED (holder binding did not verify)

============================================================
  Scenario 2: Certification -- prove the cert, hide the level
============================================================

→ Issuing SD certification credential...
Issuer: did:key:z6Mkh5R6KARHaCwjXaCbAeRyhKka3DRFABd9b85MgHczF8dY
Credential: Certification: AWS Solutions Architect - Professional from Amazon Web Services  [selecti
Hash: 0x0919a6228d4a677550b58c9cdce563864fabf781fe098c620ea6bc3b26788ee8
Saved to: credentials/certification_0919a622.jwt
→ Registering on-chain...
Credential: Certification: AWS Solutions Architect - Professional from Amazon Web Services
Hash: 0x0919a6228d4a677550b58c9cdce563864fabf781fe098c620ea6bc3b26788ee8
Issuer DID: did:key:z6Mkh5R6KARHaCwjXaCbAeRyhKka3DRFABd9b85MgHczF8dY
Expiration: none
Registering on evm...
Registered. Gas used: 140242
```

```
→ Holder reveals certificationName + certifyingBody, withholds 'level' and 'dateAwarded'...
Revealed: certificationName, certifyingBody
Presentation created (unbound).
Saved to: presentations/presentation_0919a622.jwt
→ Verifying (expect VALID, 2 claims withheld)...
Checking signature...
  Signature: VALID
  Credential: Certification: AWS Solutions Architect - Professional from Amazon Web Services
  Issuer: did:key:z6Mkh5R6KARHaCwjXaCbAeRyhKka3DRFABd9b85MgHczF8dY
  Holder: did:key:z6MkhQJ2uoi43BTY9ZVpYR5ZZckvCKUxo3H5zekTp7C2RAZN
  Disclosure: selective -- revealed certificationName, certifyingBody
  Withheld items: 2
  Holder binding: none (unbound presentation)

Checking on-chain status (evm)...
  Hash: 0x0919a6228d4a677550b58c9cdce563864fabf781fe098c620ea6bc3b26788ee8
  Registered: yes (at block timestamp 1780743177)
  Issuer (on-chain): did:key:z6Mkh5R6KARHaCwjXaCbAeRyhKka3DRFABd9b85MgHczF8dY

  Result: VALID

═══════════════════════════════════════════════════════════
  Scenario 3: Endorsement -- reveal one skill, hide the statement
═══════════════════════════════════════════════════════════

→ Issuing SD peer endorsement (Bob endorses Alice)...
Issuer: did:key:z6MkrCzBUnzKPdKm4bvY3S4JowNhuivfHLraX4tu4aeFV5tX
Credential: Endorsement: Bob Martinez endorses [distributed systems, technical leadership]  [selecti
Hash: 0x4c15a28f6777dbb09c1f3925a6226dfc4745266d6c5133717f3becb37c4f52a3
Saved to: credentials/endorsement_4c15a28f.jwt
→ Registering on-chain...
Credential: Endorsement: Bob Martinez endorses [distributed systems, technical leadership]
Hash: 0x4c15a28f6777dbb09c1f3925a6226dfc4745266d6c5133717f3becb37c4f52a3
Issuer DID: did:key:z6MkrCzBUnzKPdKm4bvY3S4JowNhuivfHLraX4tu4aeFV5tX
Expiration: none
Registering on evm...
Registered. Gas used: 140242
→ Holder reveals endorserName + ONE skill ('distributed systems'), hides the rest...
Revealed: endorserName, skills=distributed systems
Presentation created (unbound).
Saved to: presentations/presentation_4c15a28f.jwt
→ Verifying (expect VALID; relationship, statement, context, and 1 skill withheld)...
Checking signature...
  Signature: VALID
  Credential: Endorsement: Bob Martinez endorses [distributed systems]
  Issuer: did:key:z6MkrCzBUnzKPdKm4bvY3S4JowNhuivfHLraX4tu4aeFV5tX
  Holder: did:key:z6MkhQJ2uoi43BTY9ZVpYR5ZZckvCKUxo3H5zekTp7C2RAZN
  Disclosure: selective -- revealed endorserName, skills
  Withheld items: 4
  Holder binding: none (unbound presentation)

Checking on-chain status (evm)...
  Hash: 0x4c15a28f6777dbb09c1f3925a6226dfc4745266d6c5133717f3becb37c4f52a3
  Registered: yes (at block timestamp 1780743187)
  Issuer (on-chain): did:key:z6MkrCzBUnzKPdKm4bvY3S4JowNhuivfHLraX4tu4aeFV5tX
```

```
  Result: VALID

============================================================
  Scenario 4: Revocation -- the same presentation flips to REVOKED
============================================================

→ Issuing + registering a second SD employment credential...
Issuer: did:key:z6MkfBoFx3UdntT9ih21eCPJckjm4KMMGpTy5JpR3ZLo5ckE
Credential: Employment: Staff Engineer at Globex (2024-07-01 to 2025-12-31)  [selective disclosure]
Hash: 0x8b3f8592fecd0dc884fb9b37545001715ac4e689a155aca5e80178dbd19e418f
Saved to: credentials/employment_8b3f8592.jwt
Credential: Employment: Staff Engineer at Globex (2024-07-01 to 2025-12-31)
Hash: 0x8b3f8592fecd0dc884fb9b37545001715ac4e689a155aca5e80178dbd19e418f
Issuer DID: did:key:z6MkfBoFx3UdntT9ih21eCPJckjm4KMMGpTy5JpR3ZLo5ckE
Expiration: none
Registering on evm...
Registered. Gas used: 140242
→ Holder derives a presentation (reveal role only)...
Revealed: role
Presentation created (unbound).
Saved to: presentations/presentation_8b3f8592.jwt
→ Verifying before revocation (expect VALID)...
Checking signature...
  Signature: VALID
  Credential: Employment: Staff Engineer at ? (? to ?)
  Issuer: did:key:z6MkfBoFx3UdntT9ih21eCPJckjm4KMMGpTy5JpR3ZLo5ckE
  Holder: did:key:z6MkhQJ2uoi43BTY9ZVpYR5ZZckvCKUxo3H5zekTp7C2RAZN
  Disclosure: selective -- revealed role
  Withheld items: 3
  Holder binding: none (unbound presentation)

Checking on-chain status (evm)...
  Hash: 0x8b3f8592fecd0dc884fb9b37545001715ac4e689a155aca5e80178dbd19e418f
  Registered: yes (at block timestamp 1780743195)
  Issuer (on-chain): did:key:z6MkfBoFx3UdntT9ih21eCPJckjm4KMMGpTy5JpR3ZLo5ckE

  Result: VALID
→ Issuer revokes the underlying credential...
  Credential hash (JWS): 0x8b3f8592fecd0dc884fb9b37545001715ac4e689a155aca5e80178dbd19e418f
Revoking credential 0x8b3f8592fecd0dc884fb9b37545001715ac4e689a155aca5e80178dbd19e418f on evm...
Revoked. Gas used: 31976
→ Verifying the SAME presentation again (expect REVOKED)...
Checking signature...
  Signature: VALID
  Credential: Employment: Staff Engineer at ? (? to ?)
  Issuer: did:key:z6MkfBoFx3UdntT9ih21eCPJckjm4KMMGpTy5JpR3ZLo5ckE
  Holder: did:key:z6MkhQJ2uoi43BTY9ZVpYR5ZZckvCKUxo3H5zekTp7C2RAZN
  Disclosure: selective -- revealed role
  Withheld items: 3
  Holder binding: none (unbound presentation)

Checking on-chain status (evm)...
  Hash: 0x8b3f8592fecd0dc884fb9b37545001715ac4e689a155aca5e80178dbd19e418f
  Registered: yes (at block timestamp 1780743195)
  Issuer (on-chain): did:key:z6MkfBoFx3UdntT9ih21eCPJckjm4KMMGpTy5JpR3ZLo5ckE
  Revoked: yes (at block timestamp 1780743202)
```

```
  Result: REVOKED

═══════════════════════════════════════════════════════════
  Scenario 5: Tamper -- a forged presentation is rejected
═══════════════════════════════════════════════════════════

→ Creating a tampered copy of the employment presentation (one byte flipped in the JWS)...
wrote /app/presentations/tampered.jwt
→ Verifying the tampered presentation (expect INVALID signature)...
Checking signature...
  Signature: INVALID (Signature verification failed)

═══════════════════════════════════════════════════════════
  Summary
═══════════════════════════════════════════════════════════

Selective-disclosure scenarios completed on evm.

Credentials issued:    4
Presentations derived: 4

Note: each credential and its presentations share one on-chain hash,
because the anchor is the SHA-256 of the JWS only.
```

B.2: IOTA Scenarios

```
═══════════════════════════════════════════════════════════
  Step 0: Compiling Move Package
═══════════════════════════════════════════════════════════

→ Building credential_registry Move package via starter pack iota-tools...

FETCHING GIT DEPENDENCY https://github.com/iotaledger/iota.git
INCLUDING DEPENDENCY Iota
INCLUDING DEPENDENCY MoveStdlib
BUILDING credential_registry
  Build output:
total 16
drwxr-xr-x 3 root root 4096 Jun  6 12:54 .
drwxr-xr-x 5 root root 4096 Jun  6 12:54 ..
-rw-r--r-- 1 root root 1637 Jun  6 12:54 credential_registry.mv
drwxr-xr-x 4 root root 4096 Jun  6 12:54 dependencies

═══════════════════════════════════════════════════════════
  Setup: Generating Keys
═══════════════════════════════════════════════════════════

→ Cleaning up previous run artifacts...
→ Generating issuer keys (employer, certbody, endorser)...
Generated keypair: employer
  DID:  did:key:z6MkiYsQcAb6FJW4jbTXxPmw1C3xg4z6evkLL8J8cYWsufE4
  Saved to: /app/keys/employer.json
Generated keypair: certbody
```

```
  DID:  did:key:z6MksgcAc1t9ZTsYjAhMpMHgLeMRpDYkgPHGRaerdSRCzR7P
  Saved to: /app/keys/certbody.json
Generated keypair: endorser-bob
  DID:  did:key:z6MkjBcQQ7rufKQdcQ73nitH6EuexGb8TaMjEKVtsYnnrpuw
  Saved to: /app/keys/endorser-bob.json
→ Generating holder key (Alice)...
Generated keypair: alice
  DID:  did:key:z6MkjYH2okfBU1GbBuaSLqiqEnqt5maZmFzZVnW6HxuaoaWt
  Saved to: /app/keys/alice.json
  Alice's DID: did:key:z6MkjYH2okfBU1GbBuaSLqiqEnqt5maZmFzZVnW6HxuaoaWt

============================================================
  Setup: Deploying Contract
============================================================

→ Deploying CredentialRegistry on iota...
Deploying CredentialRegistry on iota...
Contract deployed at: 0xb3431ba7a9397c2089dcbe7d72a242c7449d6b98ea6fe65d8c1c26f1ee5abab5:0x6e9748ef6

Add to your .env file:
  IOTA_CONTRACT_ADDRESS=0xb3431ba7a9397c2089dcbe7d72a242c7449d6b98ea6fe65d8c1c26f1ee5abab5:0x6e9748e
→ Contract address: 0xb3431ba7a9397c2089dcbe7d72a242c7449d6b98ea6fe65d8c1c26f1ee5abab5:0x6e9748ef6f6

============================================================
  Scenario 1: Employment -- reveal only the employer
============================================================

→ Issuing SD employment credential...
Issuer: did:key:z6MkiYsQcAb6FJW4jbTXxPmw1C3xg4z6evkLL8J8cYWsufE4
Credential: Employment: Senior Engineer at Acme Corp (2022-03-01 to 2024-06-30)  [selective disclosu
Hash: 0x0dc2b5cefd940e58c5deb409223f02e50f95a6627636f7721cb9f9102fda9a21
Saved to: credentials/employment_0dc2b5ce.jwt
→ Registering the credential on-chain (issuer)...
Credential: Employment: Senior Engineer at Acme Corp (2022-03-01 to 2024-06-30)
Hash: 0x0dc2b5cefd940e58c5deb409223f02e50f95a6627636f7721cb9f9102fda9a21
Issuer DID: did:key:z6MkiYsQcAb6FJW4jbTXxPmw1C3xg4z6evkLL8J8cYWsufE4
Expiration: none
Registering on iota...
Registered. Cost (NANOS): 3994400
→ Holder derives a presentation revealing ONLY employerName, bound to a verifier...
Revealed: employerName
Presentation created (with holder binding, nonce=verifier-nonce-001, audience=hr@example.com).
Saved to: presentations/presentation_0dc2b5ce.jwt
→ Verifier checks the presentation with the matching nonce (expect VALID + binding VALID)...
Checking signature...
  Signature: VALID
  Credential: Employment: ? at Acme Corp (? to ?)
  Issuer: did:key:z6MkiYsQcAb6FJW4jbTXxPmw1C3xg4z6evkLL8J8cYWsufE4
  Holder: did:key:z6MkjYH2okfBU1GbBuaSLqiqEnqt5maZmFzZVnW6HxuaoaWt
  Disclosure: selective -- revealed employerName
  Withheld items: 5
  Holder binding: VALID

Checking on-chain status (iota)...
  Hash: 0x0dc2b5cefd940e58c5deb409223f02e50f95a6627636f7721cb9f9102fda9a21
  Registered: yes (at block timestamp 1780743269)
```

```
  Issuer (on-chain): did:key:z6MkiYsQcAb6FJW4jbTXxPmw1C3xg4z6evkLL8J8cYWsufE4

  Result: VALID
→ Replay defense: a different verifier replays it with the WRONG nonce (expect binding INVALID)...
Checking signature...
  Signature: VALID
  Credential: Employment: ? at Acme Corp (? to ?)
  Issuer: did:key:z6MkiYsQcAb6FJW4jbTXxPmw1C3xg4z6evkLL8J8cYWsufE4
  Holder: did:key:z6MkjYH2okfBU1GbBuaSLqiqEnqt5maZmFzZVnW6HxuaoaWt
  Disclosure: selective -- revealed employerName
  Withheld items: 5
  Holder binding: INVALID (nonce mismatch (possible replay))

Checking on-chain status (iota)...
  Hash: 0x0dc2b5cefd940e58c5deb409223f02e50f95a6627636f7721cb9f9102fda9a21
  Registered: yes (at block timestamp 1780743269)
  Issuer (on-chain): did:key:z6MkiYsQcAb6FJW4jbTXxPmw1C3xg4z6evkLL8J8cYWsufE4

  Result: BINDING_FAILED (holder binding did not verify)

═══════════════════════════════════════════════════════
  Scenario 2: Certification -- prove the cert, hide the level
═══════════════════════════════════════════════════════

→ Issuing SD certification credential...
Issuer: did:key:z6MksgcAc1t9ZTsYjAhMpMHgLeMRpDYkgPHGRaerdSRCzR7P
Credential: Certification: AWS Solutions Architect - Professional from Amazon Web Services  [selecti
Hash: 0x04ab9972660ea4420a146be8f3d66600f9127cbc1e032cc74625429b81955efe
Saved to: credentials/certification_04ab9972.jwt
→ Registering on-chain...
Credential: Certification: AWS Solutions Architect - Professional from Amazon Web Services
Hash: 0x04ab9972660ea4420a146be8f3d66600f9127cbc1e032cc74625429b81955efe
Issuer DID: did:key:z6MksgcAc1t9ZTsYjAhMpMHgLeMRpDYkgPHGRaerdSRCzR7P
Expiration: none
Registering on iota...
Registered. Cost (NANOS): 3994400
→ Holder reveals certificationName + certifyingBody, withholds 'level' and 'dateAwarded'...
Revealed: certificationName, certifyingBody
Presentation created (unbound).
Saved to: presentations/presentation_04ab9972.jwt
→ Verifying (expect VALID, 2 claims withheld)...
Checking signature...
  Signature: VALID
  Credential: Certification: AWS Solutions Architect - Professional from Amazon Web Services
  Issuer: did:key:z6MksgcAc1t9ZTsYjAhMpMHgLeMRpDYkgPHGRaerdSRCzR7P
  Holder: did:key:z6MkjYH2okfBU1GbBuaSLqiqEnqt5maZmFzZVnW6HxuaoaWt
  Disclosure: selective -- revealed certificationName, certifyingBody
  Withheld items: 2
  Holder binding: none (unbound presentation)

Checking on-chain status (iota)...
  Hash: 0x04ab9972660ea4420a146be8f3d66600f9127cbc1e032cc74625429b81955efe
  Registered: yes (at block timestamp 1780743276)
  Issuer (on-chain): did:key:z6MksgcAc1t9ZTsYjAhMpMHgLeMRpDYkgPHGRaerdSRCzR7P

  Result: VALID
```

```
============================================================
  Scenario 3: Endorsement -- reveal one skill, hide the statement
============================================================

→ Issuing SD peer endorsement (Bob endorses Alice)...
Issuer: did:key:z6MkjBcQQ7rufKQdcQ73nitH6EuexGb8TaMjEKVtsYnnrpuw
Credential: Endorsement: Bob Martinez endorses [distributed systems, technical leadership]  [selecti
Hash: 0x01cc32d6b5494651b7ac99fdcaec23d0bfe31b6c8927085689a3761d5a69f735
Saved to: credentials/endorsement_01cc32d6.jwt
→ Registering on-chain...
Credential: Endorsement: Bob Martinez endorses [distributed systems, technical leadership]
Hash: 0x01cc32d6b5494651b7ac99fdcaec23d0bfe31b6c8927085689a3761d5a69f735
Issuer DID: did:key:z6MkjBcQQ7rufKQdcQ73nitH6EuexGb8TaMjEKVtsYnnrpuw
Expiration: none
Registering on iota...
Registered. Cost (NANOS): 3994400
→ Holder reveals endorserName + ONE skill ('distributed systems'), hides the rest...
Revealed: endorserName, skills=distributed systems
Presentation created (unbound).
Saved to: presentations/presentation_01cc32d6.jwt
→ Verifying (expect VALID; relationship, statement, context, and 1 skill withheld)...
Checking signature...
  Signature: VALID
  Credential: Endorsement: Bob Martinez endorses [distributed systems]
  Issuer: did:key:z6MkjBcQQ7rufKQdcQ73nitH6EuexGb8TaMjEKVtsYnnrpuw
  Holder: did:key:z6MkjYH2okfBU1GbBuaSLqiqEnqt5maZmFzZVnW6HxuaoaWt
  Disclosure: selective -- revealed endorserName, skills
  Withheld items: 4
  Holder binding: none (unbound presentation)

Checking on-chain status (iota)...
  Hash: 0x01cc32d6b5494651b7ac99fdcaec23d0bfe31b6c8927085689a3761d5a69f735
  Registered: yes (at block timestamp 1780743282)
  Issuer (on-chain): did:key:z6MkjBcQQ7rufKQdcQ73nitH6EuexGb8TaMjEKVtsYnnrpuw

  Result: VALID

============================================================
  Scenario 4: Revocation -- the same presentation flips to REVOKED
============================================================

→ Issuing + registering a second SD employment credential...
Issuer: did:key:z6MkiYsQcAb6FJW4jbTXxPmw1C3xg4z6evkLL8J8cYWsufE4
Credential: Employment: Staff Engineer at Globex (2024-07-01 to 2025-12-31)  [selective disclosure]
Hash: 0x0aecc613d04740c0d3dbcb32663d9f825b0e5d2640f87034cf4416d3e45a3991
Saved to: credentials/employment_0aecc613.jwt
Credential: Employment: Staff Engineer at Globex (2024-07-01 to 2025-12-31)
Hash: 0x0aecc613d04740c0d3dbcb32663d9f825b0e5d2640f87034cf4416d3e45a3991
Issuer DID: did:key:z6MkiYsQcAb6FJW4jbTXxPmw1C3xg4z6evkLL8J8cYWsufE4
Expiration: none
Registering on iota...
Registered. Cost (NANOS): 3994400
→ Holder derives a presentation (reveal role only)...
Revealed: role
Presentation created (unbound).
```

```
Saved to: presentations/presentation_0aecc613.jwt
→ Verifying before revocation (expect VALID)...
Checking signature...
  Signature: VALID
  Credential: Employment: Staff Engineer at ? (? to ?)
  Issuer: did:key:z6MkiYsQcAb6FJW4jbTXxPmw1C3xg4z6evkLL8J8cYWsufE4
  Holder: did:key:z6MkjYH2okfBU1GbBuaSLqiqEnqt5maZmFzZVnW6HxuaoaWt
  Disclosure: selective -- revealed role
  Withheld items: 3
  Holder binding: none (unbound presentation)

Checking on-chain status (iota)...
  Hash: 0x0aecc613d04740c0d3dbcb32663d9f825b0e5d2640f87034cf4416d3e45a3991
  Registered: yes (at block timestamp 1780743287)
  Issuer (on-chain): did:key:z6MkiYsQcAb6FJW4jbTXxPmw1C3xg4z6evkLL8J8cYWsufE4

  Result: VALID
→ Issuer revokes the underlying credential...
  Credential hash (JWS): 0x0aecc613d04740c0d3dbcb32663d9f825b0e5d2640f87034cf4416d3e45a3991
Revoking credential 0x0aecc613d04740c0d3dbcb32663d9f825b0e5d2640f87034cf4416d3e45a3991 on iota...
Revoked. Cost (NANOS): 1000000
→ Verifying the SAME presentation again (expect REVOKED)...
Checking signature...
  Signature: VALID
  Credential: Employment: Staff Engineer at ? (? to ?)
  Issuer: did:key:z6MkiYsQcAb6FJW4jbTXxPmw1C3xg4z6evkLL8J8cYWsufE4
  Holder: did:key:z6MkjYH2okfBU1GbBuaSLqiqEnqt5maZmFzZVnW6HxuaoaWt
  Disclosure: selective -- revealed role
  Withheld items: 3
  Holder binding: none (unbound presentation)

Checking on-chain status (iota)...
  Hash: 0x0aecc613d04740c0d3dbcb32663d9f825b0e5d2640f87034cf4416d3e45a3991
  Registered: yes (at block timestamp 1780743287)
  Issuer (on-chain): did:key:z6MkiYsQcAb6FJW4jbTXxPmw1C3xg4z6evkLL8J8cYWsufE4
  Revoked: yes (at block timestamp 1780743292)

  Result: REVOKED

============================================================
  Scenario 5: Tamper -- a forged presentation is rejected
============================================================

→ Creating a tampered copy of the employment presentation (one byte flipped in the JWS)...
wrote /app/presentations/tampered.jwt
→ Verifying the tampered presentation (expect INVALID signature)...
Checking signature...
  Signature: INVALID (Signature verification failed)

============================================================
  Summary
============================================================

Selective-disclosure scenarios completed on iota.

Credentials issued:    4
```

```
Presentations derived: 4

Note: each credential and its presentations share one on-chain hash,
because the anchor is the SHA-256 of the JWS only.
```