Vega: Low-Latency Zero-Knowledge Proofs over Existing Credentials

Darya Kaviani
UC Berkeley

Srinath Setty
Microsoft Research

***Abstract*—As digital identity verification becomes increasingly pervasive, existing privacy-preserving approaches are still limited by complex circuit designs, large proof sizes, trusted setups, or high latency. We present Vega, a practical zero-knowledge proof system that proves statements about existing credentials without revealing anything else. Vega is simple, does not require a trusted setup, and is more efficient than the prior state-of-the-art: for a 1920-byte credential, Vega achieves 92 ms proving time, 23 ms verification time, 108 kB proofs, and a 464 kB proving key. For smaller credentials (896 bytes), these drop to 62 ms proving, 17 ms verification, and 83 kB proofs. At the heart of Vega are two principles that together enable a lightweight proof system that pays only for what it needs. First, *fold-and-reuse proving* exploits repetition and folding opportunities (i) across presentations, by pushing repeated work to a rerandomizable precomputation; (ii) across uniform hashing steps, by folding many steps into a single step; and (iii) for zero-knowledge, by folding the public-coin transcript with a random one. Second, *lookup-centric arithmetization* extracts relevant values from credential bytes, both for extracting relevant fields without full in-circuit parsing, and to enable length-hiding hashing.**

1. Introduction

Many modern platforms demand a full scan of a government credential, turning themselves into prime targets for attackers. The consequences are tangible: in October 2025, Discord announced that 70,000 users may have had their government IDs leaked in a data breach [1]; in July 2025, the "number one on the App Store" Tea app leaked 13,000 driver's license images [2].[1] Two major applications drive these demands and their attendant breach risks:

Age verification. User identity requirements such as age verification have forced application providers into a double bind: they must either use unreliable approaches like AI-based age prediction or compromise user privacy by requiring ID uploads. OpenAI's ChatGPT and YouTube recently announced teen-safety measures requiring AI-based age prediction and potential ID uploads to ensure users accessing the adult version are over 18 [3], [4], [5]. However, AI-based approaches do not have strong security guarantees and can be easily derailed. Indeed, the EU's new age-verification blueprint [6] and the UK's Online Safety Act [7] mandate government ID and other sensitive identity-linked methods for age verification.

Proof-of-humanity. With AI agents now bypassing CAPTCHAs [8], new "proof-of-humanity" services have emerged that ask users to upload an ID to certify that they are real people, further enlarging the attack surface [9]. Already, services such as CLEAR routinely funnel IDs into third-party onboarding flows [10] to verify user identity (e.g., during LinkedIn sign-up).

There is a more principled alternative: keep the credential under the holder's control and reveal nothing beyond whether a requested policy predicate holds. For example, a digital driver's license can be used to prove that the holder is over 18 without disclosing the exact birthdate. In cryptography, this capability is formalized as a *zero-knowledge proof* (ZKP) [11], [12], [13], [14], [15]: a protocol in which an untrusted *prover* convinces a *verifier* that a statement about the prover's secret data is true without leaking any information beyond the statement's validity. Figure 1 depicts this approach with a three-party workflow, and it requires no modifications to today's credential issuance infrastructure, yet provides powerful privacy guarantees.

1) **Credential issuance (unmodified).** A trusted issuer encodes the holder's attributes (the fields of a standards-compliant digital credential) as a canonical byte string and attaches a digital signature using their private key. The issuer's public key is available to the world via existing certificate authority (CA) infrastructure.

2) **Zero-knowledge presentation.** When challenged, the holder produces a ZKP that the credential is valid and that the signed attributes satisfy the verifier's predicate without exposing them verbatim.

3) **Zero-knowledge verification.** The relying party checks the ZKP, learning only the predicate's output.

Because the proof simply replaces and does not alter the signed blob, existing issuers do not need to change their software, and verifiers can continue to express policies over the same attributes they validate today. All that is required is a credential format with a stable, byte-level encoding and an accompanying digital signature.

1. These major breaches demonstrate that deleting uploaded IDs after checks is not sufficient: exposure persists during upload, third-party processing, and in logs or backups beyond providers' control.

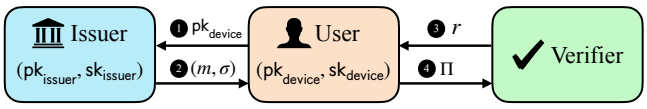

Figure 1. **System model.** During credential issuance (which happens infrequently), (1) the user's client sends the public key of their secure element $\mathsf{pk}_{\mathsf{device}}$ to the issuer; (2) the issuer produces and signs the credential m with its issuer secret key $\mathsf{sk}_{\mathsf{issuer}}$ to produce signature σ and sends (m, σ) to the user's client (the signed message includes the device public key). During zero-knowledge presentation (which happens many times) (1) the verifier sends a randomly sampled nonce r to the user; (2) the user's secure element signs the nonce r with its device secret key $\mathsf{sk}_{\mathsf{device}}$ and the user generates a zero-knowledge proof Π, where the proof establishes the knowledge of a signature on the nonce such that the signature verifies with the public key embedded in m (providing device binding, §3.1) and that the credential m is a valid credential (i.e., (m, σ) verifies using the public key of the issuer) and that attributes in m satisfy the verifier's predicate.

1.1. Recent advances and system goals

There has been substantial progress on realizing the aforementioned three-party workflow [16], [17], [18], [19], [20], [21], [22], by building on a long line of advances in efficient zero-knowledge proof systems [13], [14], [15], [23], [24], [25], [26], [27]. Two recent state-of-the-art works that prove statements over existing credentials (e.g., mDL, JWT) include Crescent [19] and Longfellow [20].

Crescent combines several proof systems including Groth16 [14] for bulk of the statement, Spartan [15] for device binding checks, and KZG polynomial commitments [28] for range proofs. Crescent's proofs are compact and online proving latency is moderate. But, because it relies on Groth16, it requires a per-circuit trusted setup (which hinders agility).[2] Another issue with Crescent is large prover keys. Longfellow builds on Hyrax [24] and instantiates with a polynomial commitment scheme based on Ligero [25]; it also uses Ligero to achieve zero-knowledge. Longfellow does not require a trusted setup. However, for efficiency, it relies on layered circuits over two different fields (a binary field and a prime field) and requires bespoke (designated verifier) cross-circuit linking mechanisms. It also exhibits higher online proving and verification latency than Crescent.

Our goal is to realize the aforementioned three-party workflow without issuer modification, while achieving transparency instead of a trusted setup, low proving latency, and fast verification. In particular, the prover must run in about a couple hundred milliseconds on commodity hardware, while the verifier should efficiently check proofs in tens of milliseconds. In settings where clients may use the system to prove humanity or present multiple credentials while browsing the web, minimizing proving latency is crucial. Equally important is lowering verification latency, as services verifying proofs from millions of users must keep both throughput and dollar cost manageable.

Additionally, the system must preserve core security properties of prior work, including full zero-knowledge and device binding (§3.1). For example, for proof of age, the verifier only needs to learn that the prover is at least a required age (e.g., $\geq$18 or $\geq$21), nothing more. Revealing the full birthdate not only exposes unnecessary personal information but also provides a stable cross-session identifier.[3] Support for device binding is critical: without it, a leaked credential (e.g., an mDL posted online) would enable anyone to generate valid zero-knowledge proofs such as "over 18," making it impossible for verifiers to distinguish legitimate users from impostors.

1.2. Our work: Vega

We introduce Vega, a new transparent (trusted-setup-free) zero-knowledge proof system for proving statements over existing credentials. On a machine with 16 vCPUs, for a standard 1920-byte mDL, Vega produces a ZKP of age in 92 ms of latency with a proof size of 108 kB, and the proof can be verified in 23 ms; for smaller credentials (896 bytes), these drop to 62 ms, 17 ms, and 83 kB proofs, respectively.

Vega's efficiency is inherent: even single-threaded, it performs $2\times$ less prover work and $4.3\times$ less verifier work than the prior transparent state-of-the-art [20]. For 1920-byte credentials, with only 4 threads, Vega already achieves 105 ms proving and 25 ms verification; with 16 threads, this translates to $5.4\times$ lower proving latency and $10.6\times$ lower verifier latency over the prior state-of-the-art with a transparent setup [20]. Even when compared to a prior work with a trusted setup and a heavy precomputation phase [19], Vega's proving latency is $2\times$ lower (92 ms vs. 182 ms) and verification is $3.9\times$ lower (23 ms vs. 89 ms).

To achieve these efficiency improvements, Vega exploits the inherent repetition, folding, and lookup opportunities introduced by the credential-proving pipeline:

- **Fold-and-reuse proving.** Vega pushes repeated work into a rerandomizable precomputation that is reused across presentations. Rerandomization enables safe reuse of witness commitments across presentations, avoiding the cost of rebuilding them each time. Additionally, Vega employs folding schemes to achieve zero-knowledge, and to combine many identical steps into a single step to speed up proving and reduce key sizes.
- **Lookup-centric arithmetization.** Vega treats credential bytes as a lookup table and uses lookup operations to extract needed values; this approach eschews full in-circuit parsing. Additionally, Vega uses lookups to enable

2. Trusted setup ceremonies sample parts of the prover's key via a multi-party computation and require everyone else using the parameters to trust at least one party involved in the ceremony. Any change to the circuit bounds (longer credential, new predicate, circuit bug fix) would require rerunning the entire ceremony. Systems with per-circuit trusted setup are even worse as each circuit shape requires a distinct proving and verifier key, so total storage on client devices scales with the number of supported predicates.

3. In existing credential standards like mDLs, attributes such as the birthdate are hashed with a per-field nonce and stored in the signed credential; because both values remain constant across presentations, the resulting hash becomes a linkable identifier across services.

length-hiding hashing, which is necessary as different users' credentials vary in length.

1.2.1. Fold-and-reuse proving. Vega's starting points are the transparent proof systems Spartan [15] and NeutronNova [29]. We modify these proof systems to enable a safe reuse of some of the prover's work across presentations by splitting the prover's work into a (rerandomizable) precomputation phase and an online phase. We refer to these modified variants as SplitSpartan and SplitNeutronNova. In addition, we add: (i) commit-and-prove capabilities; and (ii) randomized checks inside circuits (which enables functionality such as lookups). Below, we describe how we leverage these capabilities in our circuit design and to improve the proving latency in our specific setting of proving statements over existing credentials.

Folding for efficient zero-knowledge. We introduce a new method to make our proof systems zero-knowledge via Nova's folding scheme [26]. Specifically, our method makes any public-coin polynomial interactive oracle proof (PIOP) zero-knowledge. When combined with a zero-knowledge polynomial commitment scheme, it leads to a zero-knowledge argument. Since we apply folding to the verifier's checks in a PIOP rather than to the full instance-witness pair for the original credential statement, our approach provides an exponential improvement in the prover's work over prior uses of Nova for achieving zero-knowledge [27, §D].

Reusable proving for lower latency. In Vega, the prover's precomputation phase is used to commit to a partial witness that does not depend on the verifier's inputs (e.g., credential bytes, witness elements arising from the verification of issuer's signature on the credential), and the online phase is used to commit to remaining witness that depends on the verifier's inputs (e.g., checks on the device signature for device binding, today's date). The prover's precomputation work is reused across proofs for multiple verifiers. To ensure that the reuse of commitments does not violate zero-knowledge, the prover rerandomizes commitments created in the precomputation phase, so they are indistinguishable from fresh commitments. This is asymptotically and concretely less expensive than creating new commitments.

Folding for leveraging uniformity. We observe that circuits that check statements over digital credentials naturally split into a "uniform" part (iterated SHA-256 over credential bytes) and a non-uniform part (the remaining checks including signature checks). We design a folding-friendly arithmetization that separates the computation into identical SHA-256 "step" circuits and a "core" circuit for the other checks (e.g., signatures). Different circuit instances share state via the commit-and-prove capabilities. The prover folds many witnesses of a step circuit into one using a folding scheme [29], and then proves that the folded witness is valid together with the core circuit's witness using SplitSpartan [15]. A key benefit is that it reduces proving-key sizes and lowers the setup time as the proving key includes only one step circuit and one core instead of an unrolled circuit.

1.2.2. Lookup-centric arithmetization. We devise efficient circuits that leverage lookups into a table. These lead to a more compact arithmetization, which directly leads to lower proving times. We employ lookups in two distinct places.

Lookups for efficient credential parsing. We devise an arithmetization that treats credential bytes as a byte-addressable table. In our context, credential bytes are signed by a trusted issuer and circuits verify the validity of the signature (so we can assume that the bytes are well-formed). This allows accessing fields in the credential without full in-circuit credential parsing. Parsing and validating fields via lookups is far more efficient than doing it without lookups as otherwise the circuit would have to emulate a RAM program that parses the credential and extracts relevant fields.

Lookups for length-hiding SHA-256. A standard R1CS gadget for hashing messages with SHA-256 does not hide the true length of the message hashed. Note that this is a crucial feature as different credentials have different sizes, and we must use the same circuit to hash those.[4] Since SHA-256 hashes messages block-by-block and there is a digest at the end of each block, we have the circuit formulate a table containing all the intermediate digests. We then use lookups with a prover-provided index to pick the right digest from the table.[5] Of course, the prover can be malicious and provide the wrong index. We address this by verifying the issuer's signature on the digest returned by the lookup operation. We do signature verification with an efficient arithmetization of the ECDSA signature verification algorithm over the P-256 elliptic curve.

Beyond digital IDs, Vega's arithmetization techniques apply to any byte-encoded document digitally signed by a trusted issuer. More broadly, fold-and-reuse proving and lookup-centric arithmetization advance proof-system design for resource-constrained clients and offer a general approach for computations that combine uniform and non-uniform components in client-side proving.

2. A technical overview of Vega

We provide two instantiations of Vega (Figure 2): Vega_{SC} and Vega_{MC}. The primary difference is in how they arithmetize the statement to be proven and what proving backend they use. Vega_{SC} employs a simple, single circuit that encodes all checks over an existing credential and proves that circuit with SplitSpartan. Vega_{MC} instead employs a multi-circuit design that contains a collection of repeated execution of a step circuit, as well as a core circuit that ties all the step circuits together via shared commitments.

Both Vega_{SC} and Vega_{MC} share a set of core checks, which comprises issuer signature verification on the mobile security object (MSO) bytes, device nonce-signature verification against an embedded device key, and lookup-based field

4. Naively padding all credentials to the same length leads to a digest that is different from a legal SHA-256 digest of the original message.

5. Such lookup operations are referred to as indexed lookups [30].

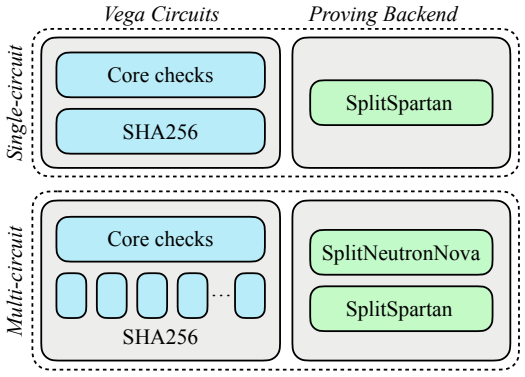

Figure 2. **Vega proving pipeline.** We depict two variants: (1) Vega_{SC}, which uses a single circuit and proves it with SplitSpartan; and (2) Vega_{MC}, which uses a collection of step circuits to represent steps of SHA-256 computation (these step circuits are folded into a single step circuit using SplitNeutronNova's folding scheme) and a single core circuit that includes remaining checks (the folded step circuit and the core circuit are then proven with SplitSpartan). Both Vega_{SC} and Vega_{MC} share a set of core circuit checks, including lookups and device/issuer signature verification. The two blue core boxes depict the same shared checks in the two variants.

authenticity checks. The variants differ in their methodology for arithmetizing SHA-256, and therefore their proving backends. Vega_{MC} uses SplitNeutronNova to fold per-block SHA-256 compression steps into a single step. The folded step and the core circuit instance are then proven with SplitSpartan. Appendix B provides a concrete mDL example and illustrates the signed credential structure underlying the proof statement.

End-to-end protocol. We give a walkthrough of the proving protocol to ground the technical components that follow; we focus on Vega_{MC} and note where Vega_{SC} differs.

Arithmetization. The statement is encoded as a collection of split R1CS instances (§2.1.1): k step instances and one core instance. Each step circuit performs one SHA-256 block transform (§4.2); the core circuit performs two ECDSA signature verifications, field lookups, and semantic checks (§2.2). Every instance's witness is split into three vectors—*shared*, *precommitted*, and *online*—with the shared vector identical across all instances, linking the step and core circuits. Vega_{SC} uses a single circuit that encodes all of these checks and proves it directly with SplitSpartan. Figure 2 depicts the circuit structure.

Precomputation (once per credential). The prover computes witnesses for the reusable (shared and precommitted) splits of all step circuits and the core circuit, then commits to each. For commitments, we use Hyrax PCS [24]. In Vega_{SC}, the same applies to its single circuit.

Proving phase (once per presentation).

1) *Re-randomize and commit.* The prover re-randomizes the pre-computed commitments and commits to the remaining presentation-specific (online) witness splits, yielding k step instances $U_1, \ldots, U_k$ and one core instance U_{core}. In Vega_{SC}, there is a single instance.
2) *Folding uniform computation with SplitNeutronNova.* A SplitNeutronNova folding sum-check ($\log k$ rounds) folds all step instances into a single folded instance, combining commitments via random linear combination. Vega_{SC} skips this step.
3) *Batched SplitSpartan.* A batched outer+inner sum-check ($2 \log n$ rounds, where n is step/core circuit size) jointly reduces the folded step and core R1CS claims to point-evaluation claims on the witness polynomials. In Vega_{SC}, this operates on the single split R1CS instance.
4) *Succinct NovaBlindFold (§2.1.2).* The prover commits to round messages in SplitNeutronNova and SplitSpartan with Pedersen commitments, encodes the verifier's checks in a small split R1CS instance, and folds it with a randomly sampled relaxed split R1CS instance. Fiat-Shamir challenge derivation and witness polynomial evaluation checks remain outside the split R1CS instance—no recursive proof composition is needed.
5) *Relaxed SplitSpartan.* A non-ZK SplitSpartan proves satisfiability of the small folded verifier split R1CS instance.
6) *PCS evaluation argument.* The witness evaluation claims are verified via a single Hyrax ZK evaluation argument.

2.1. Efficient zero-knowledge proving backend

2.1.1. Split-committed witnesses for reusable and shared state. In our setting, proofs must perform the same checks (e.g., issuer signature verification and predicate evaluation) across different presentations, and share state across different circuits. At the same time, some checks like the device nonce-signature verification must be proven per presentation. Split-committed witnesses realize precisely this: slices are committed separately. Concretely, we adapt Spartan and NeutronNova to a split-committed variant of R1CS [13], [31], [32]. This extension is folklore [33] and has appeared in recent work [21], [22], [34], [35], [36]. Our main contribution here is to implement split-committed R1CS in a developer-friendly fashion around an existing R1CS circuit framework [37]. A nice benefit of our implementation is that a developer specifies their R1CS circuits as before (with Rust), but they can now partition their functionality (which automatically splits witness variables). Furthermore, all existing R1CS gadgets remain usable and require no modifications. This provides two capabilities.

(1) Share state across multiple circuit satisfiability instances. Since witness vectors are split and committed separately, two circuit satisfiability instances could share one or more splits. Furthermore, the verifier can ensure that they in fact share data by checking the equality of commitments.

(2) Support for randomized checks in circuits. With split committed R1CS, one can efficiently encode randomized checks (e.g., permutations, lookups) within R1CS [33]. More generally, circuits can encode verifier's checks in a public-coin interactive protocol that is made non-interactive via Fiat-Shamir [38]. In more detail, given an ℓ-round

public-coin interactive protocol, it is first turned non-interactive via Fiat-Shamir [38]. One then writes a set of constraints over witnesses $(w_1, \ldots, w_\ell)$ and public inputs $(c_1, \ldots, c_\ell, x)$, where each w_i is prover's message in the ith round of the interactive protocol and c_i is a challenge; x is auxiliary public input. The constraints are satisfying if and only if the verifier in the non-interactive version of the public coin interactive protocol accepts. In addition to encoding lookup checks with this, we leverage this idea to introduce a novel method for adding zero-knowledge to our base proof systems, which we describe further in §2.1.2.

Moving prover's work to a precomputation phase. In Vega, proofs over the same credential are presented to different verifiers, so we can modify our base provers to push work to a precomputation phase. In particular, using our split witness introduced above, we devise a circuit such that witness elements that do not depend on a particular presentation to a verifier (e.g., credential bytes, signature checks on the credential using the issuer's public key) are separated from witness elements that depend on the verifier (e.g., nonce, signature checks on the device, today's date). This allows the prover to commit to verifier-independent witness once and reuse it across multiple presentations. Naively, this violates zero-knowledge as two different proofs about the same credential share some of the commitments.

To obtain zero-knowledge while reusing commitments, the prover rerandomizes the commitments prior to reusing them. This ensures that rerandomized commitments are indistinguishable from commitments generated from scratch. Vega uses Hyrax PCS [24], where rerandomization of commitments is less expensive than generating fresh commitments, both asymptotically and concretely.

2.1.2. Adding ZK via folding verifier's checks. We describe a new method to add ZK to Spartan and NeutronNova. We first describe a prior technique that we adapt. Consider the notion of a folding scheme (§3, Def. 3.2).

NovaBlindFold: Folding-based approach to ZK. Suppose that we have a committed R1CS instance-witness pair (u, w) for a constraint system encoded by sparse matrices (A, B, C), where the instance $u = (\overline{w}, x)$, where $\overline{w}$ is a hiding commitment to w and x is public inputs. The witness w is satisfying if $\overline{w}$ is a commitment to w and $Az \circ Bz = Cz$, where $z = (w, 1, x)$. (Cf. committed relaxed R1CS in §3, Def. 3.1.) Suppose that the prover holds (u, w) and the verifier holds u. Note that the verifier does not learn anything about the witness w because u does not reveal anything about w. Without additional protocols, the verifier cannot check if the prover's witness satisfies the instance without additional knowledge about the witness.

To address this, Kothapalli and Setty [27] provide NovaBlindFold, a protocol based on Nova's folding scheme [26]. In that protocol:

1) The prover samples a random committed relaxed R1CS instance-witness pair (u', w'), and sends u' to the verifier (u' contains no information about w).
2) The prover and the verifier execute Nova's folding scheme, where at the end of the protocol, the verifier computes a folded instance u'' and the prover computes a folded witness w''. Nova's folding scheme involves the prover sending a single hiding commitment, so it does not reveal anything about w to the verifier.
3) The prover sends the folded witness w'' to the verifier.
4) The verifier checks that folded witness satisfies u''. If so, the verifier accepts; otherwise the verifier rejects.

In the above, instead of the prover sending w'' to the verifier, the prover can use a non-ZK variant of Spartan to prove that it knows a valid w''. This makes Spartan ZK. However, this carries a significant cost: in our context, w only contains small values (e.g., most values are binary or at most 8 bits). So, their approach to make Spartan zero-knowledge makes the prover incur orders of magnitude higher costs to commit to the random witness w' over the cost to commit to w.[6]

Our approach. We devise an alternate approach to make SplitSpartan and SplitNeutronNova zero-knowledge. Our key insight is that instead of folding the original committed R1CS instance-witness pair with a randomly sampled instance-witness pair (which is expensive), we run the prover of SplitSpartan (and SplitNeutronNova) on the original instance-witness pair. We then write a split-committed R1CS instance-witness pair that encodes the checks of the verifier of non-ZK version of SplitSpartan (and SplitNeutronNova). We apply the approach of Kothapalli and Setty [27] to that instance-witness pair. This instance-witness pair is satisfying if and only if the non-ZK version of SplitSpartan verifier accepts the prover's messages. Crucially, the verifier of SplitSpartan (and SplitNeutronNova) performs a logarithmic number of checks, so the size of the circuit to which Nova's folding scheme is applied is exponentially smaller than if we applied it to the original instance-witness pair. For Spartan, this comes from two $\log m$-round sum-checks plus constant-sized checks; the polynomial commitment scheme proves evaluations in ZK outside this circuit, so the verifier-checks circuit stays $O(\log m)$.

Our approach to make SplitSpartan (and SplitNeutronNova) zero-knowledge is reminiscent of the seminal approach to achieve ZK [39], [40]: the prover commits to its messages using a hiding commitment and then uses a collection of Schnorr proofs to prove that the verifier's checks pass with plaintext messages underneath the commitments. In contrast, our approach is simpler, easy to engineer, and makes use of a single tool (Nova's folding scheme) in place of many types of Schnorr proofs.

6. The NovaBlindFold approach is acceptable in the recursive folding setting as it is applied after many steps, so the cost of ZK is negligible.

2.2. Compact circuits for existing credentials

To produce proofs of statements (e.g., age over 18) over existing credentials (e.g., mDL), we must encode the statement to be proven in the language of our zero-knowledge proof systems. In our context, this is split-committed R1CS (§2.1). Because the prover's costs scale with the size of the circuit, it is crucial to minimize circuit size for practical performance.

In our target application, we observe that the statement to be proven can be encoded with a compact circuit if we had efficient sub-circuits for: (1) computing SHA-256 of a message provided as a byte vector; (2) verifying ECDSA signatures over the elliptic curve P-256; and (3) parsing a byte vector containing attributes of a credential in CBOR [41] (the format used in mDL), extracting relevant fields from CBOR (e.g., device public key, expiration date, date of birth), and enforcing predicates on extracted fields.

Since credentials of different users can be of different lengths, we need a circuit that works for all credentials up to a certain bound on the credential length. This is for both logistical reasons and for zero-knowledge: if different users use different circuits, then that trivially leaks information about the user to a verifier. This additional requirement needs care in sub-circuits (1) and (3).

2.2.1. Length-hiding SHA-256 (§4). Recall that the SHA-256 computation is a repeated invocation of a SHA-256 compression function applied on a padded vector encoding the length of the message. Specifically, SHA-256 computation appends the length of the message hashed as an 8-byte value along with a 1-byte prefix and then iteratively applies the compression function on 64-byte blocks.

Lookups for single-circuit length-hiding SHA-256 (§4.1). To make this length-hiding, we require the setup procedure to specify an upper bound on the length of the messages hashed (e.g., 2048 bytes). The circuit will always hash vectors of this length, so the length of the circuit does not reveal anything about the message hashed. However, this does not reflect the usual SHA-256 computation. Our core observation is that because SHA-256 is an incremental computation, there is a digest at every 64-byte increments. For instance, for an upper bound of 2048, there are 32 digests computed. Of these, only one of the digests is the correct digest depending on the length of the message hashed. We let the prover provide as non-deterministic witness the index corresponding to the real digest in the vector of intermediate digests. The SHA-256 sub-circuit then returns the digest at that location along with the index provided by the prover. Viewed through the lookup lens, the vector of intermediate digests forms a table, and a one-hot/lookup selector (private index) picks the canonical digest.

Of course, the prover could be malicious and return the wrong index and the wrong digest at that index. However, in our context, the digest picked from the vector of digests is used to verify an ECDSA signature using the issuer's public key. So, if the prover maliciously picks the wrong digest or hashes an incorrect vector of bytes, the signature verification fails. Additionally, we use the index returned by the SHA-256 sub-circuit to ensure that the circuit only reads fields from the sub-vector bytes whose signature is checked. Our single-circuit variant Vega_{SC} uses this approach.

Folding for multi-circuit length-hiding SHA-256 (§4.2). Since SHA-256 is an incremental computation, we observe that it is perfectly suited for the application of folding schemes. The main benefit is that the prover only needs a circuit for a single iteration rather than an unrolled version of the SHA-256 computation. Additionally, we can apply SplitNeutronNova's folding scheme to fold multiple steps into a single step and only apply SplitSpartan on the single step, resulting in our multi-circuit variant Vega_{MC}. To ensure that different circuits encode the incremental computation without leaking information about the length of the message hashed, we use split witnesses where one of the segments is shared across all steps. Furthermore, each step performs certain checks on the shared state. In particular, the shared segment is used to store a digest vector and each step merely attests to the correctness of one of them.

2.2.2. ECDSA signature verification over P-256 (§4.3). We must verify both the issuer's signature on the MSO bytes and the device binding signature, which we describe in §3.1. Verifying an ECDSA signature over P-256 elliptic curve requires arithmetic over the base field of P-256. If the constraint system supported by the proof system is defined over a field that is not the same as the base field of P-256, the constraint system must use field emulation, which is up to 3 orders of magnitude more expensive than native field arithmetic. To avoid field emulation, a common technique is to make the field supported by the proof system match the base field of P-256 [21], [42], [43]. Fortunately, a prior work [44] identifies a class of elliptic curves whose scalar field matches the base field of P-256: T-256. This curve is used in recent works to prove signature verification inside Spartan efficiently [21]. We adapt this approach and make it even more efficient. Specifically, while the bulk of signature verification involves operations over the base field of P-256, there are some operations over the scalar field of P-256. Recent work [21] uses field emulation to handle those operations. We observe that the modulus of the scalar field of P-256 is smaller than the modulus of the base field of P-256. Using this observation, we eliminate all field emulation.

2.2.3. Extracting CBOR fields via lookups (§5). During credential issuance, bytes of the credential (e.g., MSO bytes) are hashed and signed by the issuer. Given that the issuer is trusted in our system model, we observe that we can safely assume that the MSO bytes are well-formed i.e., the issuer derives the bytes of MSO by following the serialization rules of CBOR. With this observation, we eschew full in-circuit CBOR parsing, which is expensive and complex. Instead, we treat the vector of credential bytes as a byte-addressable lookup table. In our context, this vector is returned by the

SHA-256 sub-circuit along with an index denoting the prefix of the vector containing the real message hashed.

The prover first pinpoints where each relevant field occurs in the lookup table and supplies these indices as addresses and the corresponding values. The circuit first checks that the provided addresses lie in the range corresponding to the message that was hashed by SHA-256 computation. The circuit then checks that the values are correctly looked up using the provided addresses. The latter uses the logup lookup argument [45],[7], which we encode in the circuit by leveraging split-committed R1CS. The circuit then enforces (i) field-specific CBOR prefixes at the hinted offsets (Figure 8), (ii) address contiguity to prevent splicing, and (iii) semantic checks such as whether the expiry date extracted from the document is in the future.

3. Background

We let $\mathbb{F}$ denote a finite field (e.g., the prime field $\mathbb{F}_p$ for a prime p) and let $\mathbb{F}^n$ denote vectors of length n with elements in $\mathbb{F}$. We refer to prior work [26], [29] for a formal definition of an additively homomorphic commitment scheme. We write $\mathbb{F}[X_1, \ldots, X_n]$ to denote multilinear polynomials over field $\mathbb{F}$ in the variables $(X_1, \ldots, X_n)$.

Definition 3.1 (Committed relaxed R1CS). Consider a finite field $\mathbb{F}$ and a commitment scheme Com over $\mathbb{F}$. Let the public parameters consist of size bounds $m, n, \ell \in \mathbb{N}$ where $m > \ell$, and commitment parameters pp for vectors of size up to $\max\{m, m - \ell - 1\}$. The committed relaxed R1CS structure consists of sparse matrices $A, B, C \in \mathbb{F}^{m \times m}$ with at most $n = \Omega(m)$ non-zero entries in each matrix. A committed relaxed R1CS instance is a tuple $(\overline{E}, u, \overline{W}, \mathsf{x})$, where $\overline{E}$ is a commitment to a vector in $\mathbb{F}^m$ and $\overline{W}$ is a commitment to a vector in $\mathbb{F}^{m-\ell-1}$, $u \in \mathbb{F}$, and $\mathsf{x} \in \mathbb{F}^\ell$ are public inputs and outputs. An instance $(\overline{E}, u, \overline{W}, \mathsf{x})$ is satisfied by a witness $(E, r_E, W, r_W) \in (\mathbb{F}^m, \mathbb{F}, \mathbb{F}^{m-\ell-1}, \mathbb{F})$ if $\overline{E} = \mathsf{Com}(\mathsf{pp}, E, r_E)$, $\overline{W} = \mathsf{Com}(\mathsf{pp}, W, r_W)$, and $(A \cdot Z) \circ (B \cdot Z) = u \cdot (C \cdot Z) + E$, where $\circ$ denotes component-wise multiplication and $Z = (W, \mathsf{x}, u)$.

A folding scheme [26] is a protocol in which a prover and a verifier reduce the task of checking two instances of a relation $\mathcal{R}$ with some structure s to the task of checking a single instance in $\mathcal{R}$ with structure s. Formally, it can be expressed as a particular reduction of knowledge (RoK) [47].

Definition 3.2 (Folding scheme). A folding scheme for $\mathcal{R}_1^m$ and $\mathcal{R}_2^n$ is a RoK of type $\mathcal{R}_1^m \times \mathcal{R}_2^n \to \mathcal{R}_1$ where the encoder outputs its input structure. We call a reduction of type $\mathcal{R}^n \to \mathcal{R}$ simply as a folding scheme for $\mathcal{R}$.

A lookup argument [30], [45], [46], [48], [49] allows proving that all elements in one vector are contained in another vector. Vega relies on an indexed lookup argument, which generalizes lookup arguments to allow specifying positions where lookups are made.

Definition 3.3 (Statement proven in an indexed lookup argument). For $m, n \in \mathbb{N}$, given commitments $\overline{a}$, $\overline{b}$, $\overline{t}$, the prover knows an opening $a = (a_0, \ldots, a_{m-1}) \in \mathbb{F}^m$ of $\overline{a}$, $b = (b_0, \ldots, b_{m-1}) \in \mathbb{F}^m$ of $\overline{b}$, and $t = (t_0, \ldots, t_{n-1}) \in \mathbb{F}^n$ of $\overline{t}$ such that for each $i = 0, \ldots, m-1$, $a_i = t[b_j]$, where $t[b_j]$ is shorthand for the b_j'th entry of t.

Definition 3.4 (Polynomial evaluation relation). Let $(\mathsf{Gen}, \mathsf{Com})$ denote an additively homomorphic commitment scheme. Consider a size bound $\ell \in \mathbb{N}$ and let pp denote public parameters of the commitment scheme. We define the polynomial evaluation relation, PE, as follows.

$$\mathsf{PE} = \left\{ (\mathsf{pp}, (\overline{Q}, x, \overline{y}), (Q, y)) \;\middle|\; \begin{array}{l} Q \in \mathbb{F}[X_1, \ldots, X_\ell], \\ \overline{Q} = \mathsf{Com}(\mathsf{pp}, Q), \\ \overline{y} = \mathsf{Com}(\mathsf{pp}, y), \\ y = Q(x) \end{array} \right\}.$$

3.1. Mobile driving license (mDL) standard

The ISO/IEC 18013-5 standard [50] defines the Mobile Driving License (mDL) credential format that underpins many state-issued digital driver licenses in the United States and an increasing number of national identity schemes worldwide. An issuing authority encodes the citizen's attributes into a signed Mobile Security Object (MSO). The MSO is a CBOR structure that bundles (i) issuer metadata such as validity period, document type, and the device's public key, and (ii) a digest table that lists a salted hash for each data element. During presentation, the holder reveals only the requested plaintext elements together with their respective salts; the verifier recomputes the digests and checks them against the signed table, thereby achieving selective disclosure.

Device binding. A security feature of ISO/IEC 18013-5 is its mandatory *device binding*. When the credential is issued, the mobile device creates a key pair $(\mathsf{sk}_{\mathsf{device}}, \mathsf{pk}_{\mathsf{device}})$. The public key $\mathsf{pk}_{\mathsf{device}}$ is embedded in the signed MSO, and the holder proves possession of $\mathsf{sk}_{\mathsf{device}}$ during every presentation by signing a verifier-supplied challenge. Consequently, a stolen copy of the credential alone is insufficient for impersonation; an adversary must also compromise the client's secure enclave. As such, a zero-knowledge version of the mDL identity system must also support device binding. Appendix B provides an example mDL credential.

4. Compact circuits for SHA-256 and ECDSA

Digital identity documents are issued by an issuer who digitally signs credential data. In the case of mDL, the mobile security object (MSO) is signed with ECDSA over SHA-256 following ISO/IEC 18013-5 [50]. We therefore focus on verifying ECDSA and SHA-256 inside circuits using the fold-and-reuse framework and lookup-centric arithmetization. Although other identity protocols may employ different signature or hash algorithms, several of our techniques

7. More efficient lookup arguments exist [30], [46]. In Vega, the table is small (e.g., 2048 entries) and the number of lookups is small (e.g., a few hundred), so logup [45] introduces a tiny fraction of the overall constraints and a faster lookup argument does not lead to better performance.

generalize to other algorithms. To enable zero-knowledge proofs about these credentials, we must efficiently arithmetize hashing of the MSO data and other issuer-signed items.

Different users will have credentials with different MSO lengths. For a ZKP, we need a circuit that can work with MSO of any user in the system—without leaking information about the true length of the MSO of a user. The latter is important otherwise the length information can be used to potentially link presentations of the same user.

We present an approach to arithmetize SHA-256 computations that hides the length of the message hashed beyond an upper bound using lookups. We also describe an extension that allows leveraging folding schemes: the key idea is to decompose a single SHA-256 circuit into many small, independent per-block circuits that can be efficiently aggregated via folding. In §4.3, we show how to efficiently arithmetize ECDSA verification itself.

As previewed in §1, this length-hiding design is based on lookups: the circuit uses one-hot/lookup selection to pick the canonical digest while hiding the true message length.[8]

4.1. Single-circuit length-hiding SHA-256

We begin with the single-circuit length-hiding SHA-256 circuit used in Vega_{SC} (Figure 2). We recall the classic hash computation. The input to the algorithm is a message M padded into n 512-bit blocks $M_0, \ldots, M_{d-1}$, and the output is a 256-bit digest H_n. Starting from the standard IV, iteratively compute $H_{i+1} \leftarrow \text{SHA256Compress}(H_i, M_i)$ for $i = 0, \ldots, n-1$ over the padded 512-bit blocks M_i and return the final digest H_n. It is easy to build a circuit that hashes the entire padded message in one shot, but the circuit size trivially leaks information about the number of blocks hashed n. To hide the true message length while fixing a circuit size, we have the setup publish a public upper bound ℓ (in 512-bit blocks) and every prover supplies exactly ℓ blocks. Padding proceeds in two stages:

1) Canonical SHA-256 padding: append `0x80`, then zeros up to the penultimate 64 bits of the last block, and finally the big-endian bit length of the original message. Let the canonical padded message have $d+1$ blocks for some $d \in \{0, \ldots, \ell-1\}$.
2) If $d+1 < \ell$, append full zero blocks after the 64-bit length field until the total length equals ℓ blocks.

The circuit enforces this canonical padding at the private boundary d via lookups on the padding bytes: it checks that the looked-up bytes are contiguous, that the first padding byte is `0x80`, that the final 8-byte field encodes a legitimate bit-length for the original message, and that all intervening padding bytes are zero. This binds d to the unique valid padding boundary.

8. Our construction relies on SHA-256's Merkle–Damgård structure; sponge hashes such as SHA-3 would require a different design.

Private inputs: Blocks $M_0, \ldots, M_{\ell-1}$; one-hot vector $b = (b_0, \ldots, b_{\ell-1})$; index $d \in \{0, \ldots, \ell-1\}$.

1) $M \leftarrow (M_0, M_1, \ldots, M_{\ell-1})$.
2) For $i = 0, \ldots, \ell-1$:
   - If $i = 0$, set $H_i \leftarrow \mathsf{IV}$.
   - Enforce $H_{i+1} = \text{SHA256Compress}(H_i, M_i)$.
3) Enforce canonical padding at d.
4) Enforce one-hot binding:
   - Global: $\sum_{j=0}^{\ell-1} b_j = 1, \ \sum_{j=0}^{\ell-1} j\, b_j = d$.
   - Per-bit: $b_j \in \{0,1\}$ for all j.
5) Set $H_{d+1} \leftarrow \sum_{j=0}^{\ell-1} b_j\, H_{j+1}$.
6) Return (M, d, H_{d+1}).

Figure 3. $\text{SHA256}_{\text{sc}}$: Single-circuit length-hiding SHA-256

The circuit size grows linearly with the public upper bound ℓ because the circuit contains one 512-bit compression subcircuit per block. However, we can no longer arithmetize the computation as before because the digest computed at the end of ℓ blocks no longer matches the real SHA-256 digest. We address this as follows.

4.1.1. Digest selection via lookups with one-hot vectors. The circuit computes the entire digest vector $(H_1, \ldots, H_\ell)$ and internally selects the canonical digest H_{d+1} using a one-hot selector determined by a private index d.

We encode the canonical position using a private index $d \in \{0, \ldots, \ell-1\}$ and a one-hot vector $b = (b_0, \ldots, b_{\ell-1})$. We enforce the following constraints:

$$\sum_{j=0}^{\ell-1} b_j = 1, \qquad d = \sum_{j=0}^{\ell-1} j\, b_j, \qquad b_j \in \{0,1\}\ \forall j.$$

We compute the canonical digest with: $H_{d+1} \leftarrow \sum_{j=0}^{\ell-1} b_j\, H_{j+1}$. Practically, this entails two R1CS constraints, plus the standard per-bit Booleanity checks for the b_j's; no auxiliary variables are required beyond b and d. So the overhead of length-hiding is minor, especially relative to the per-block compression logic.

Note that this selection logic via one-hot vectors can be viewed as implementing Shout lookup argument [46] using R1CS. In our context, ℓ is small (e.g., 32), so the cost of digest selection via lookups with one-hot vectors is minimal.

Gadget interface. The single-circuit length-hiding SHA-256 gadget is invoked once for a fixed upper bound ℓ. Only ℓ is public; the blocks $(M_0, \ldots, M_{\ell-1})$, the vector b, the index d, and intermediate H_i remain secret values.

The pseudocode for this gadget is shown in Figure 3.

4.1.2. Correctness. Let M be any message of bit length at most $512\ell - 1$. The padding expands M to exactly ℓ blocks, and the circuit evaluates the classical SHA-256

compression on every block in sequence. The one-hot selection binds d and internally selects the canonical digest H_{d+1}, ignoring digests induced by padding blocks beyond the standard SHA-256 padding. Thus, the enforced digest equals SHA-256(M) even though the circuit processes all ℓ blocks. The construction is equivalent to a naive variable-length circuit that hashes the canonical padded message of length $d+1$ and enforces the same digest H_{d+1}. Any other d either violates the padding constraints or fails the issuer-signature check.

4.2. Multi-circuit length-hiding SHA-256

We now describe the multi-circuit length-hiding circuit used in Vega$_{\text{MC}}$ (Figure 2). Rather than arithmetizing the entire computation in one circuit, we devise a collection of step circuits (where each only checks a single SHA-256 compression function) and a core circuit (that picks the right digest). All circuits use split-committed R1CS with the first split/segment dedicated to values shared across all the step circuits and the core circuit (this is enforced by the ZKP verifier with commitments). In particular, we store all the intermediate digests in the shared segment, allowing different step circuits to assert the correctness of one of them and the core circuit to select an appropriate digest.

Gadget interface. Conceptually, we split the circuit into:

- a *binding gadget* that fixes the global witness $(M_0, \ldots, M_{\ell-1})$ and $(H_1, \ldots, H_\ell)$ and selects the canonical digest H_{d+1}; and
- a *step gadget* that enforces the single transition at a public step index i.

Figure 4 depicts the per-step gadget, and Figure 5 depicts the binding gadget. All ℓ per-step circuits are independent (they reference only the shared witness and the public index i). The one-hot mechanism of §4.1.1 is also used here to select the canonical digest H_{d+1}. When proving, the prover folds per-step circuit instances into a single instance and then proves the folded instance alongside the core instance.

4.2.1. Multiple independent messages. The same machinery extends to multiple unrelated payloads. Suppose the prover wishes to hash messages $M^{(1)}, \ldots, M^{(m)}$ padded to $\ell_1, \ldots, \ell_m$ blocks, respectively. Concatenate all $\ell := \sum_{t=1}^{m} \ell_t$ blocks and their intermediate digests into a single witness, and reuse a one-hot selector over $\{0, \ldots, \ell-1\}$. Whenever the per-step index crosses a message boundary, reset the chaining value to the standard SHA-256 IV so that each segment is verified independently. No change to the circuit shape is necessary; the public setup fixes the per-message block budgets $(\ell_1, \ldots, \ell_m)$.

4.2.2. Correctness. Each per-step circuit takes as input an index i and the shared witness values (H, M), which are reused across all steps. The circuit recomputes $H_{i+1} = \text{SHA256Compress}(\mathsf{IV}, M_0)$ when $i = 0$ and $H_{i+1} = \text{SHA256}\textsc{Compress}(H_i, M_i)$ otherwise. The one-hot vector of §4.1.1 selects the canonical digest H_{d+1} via the private index d. A collection of ℓ step gadgets (one for each index value) and a binding gadget therefore certifies that $(H_1, \ldots, H_\ell)$ is the correct hash evaluation of $(M_0, \ldots, M_{\ell-1})$.

Public input: step index $i \in \{0, \ldots, \ell-1\}$.
Private inputs: $(M_0, \ldots, M_{\ell-1})$, $(H_1, \ldots, H_\ell)$.

1) One-hot selection with $\mathbf{b} = (b_0, \ldots, b_{\ell-1})$ for i:
   - Global: $\sum_{j=0}^{\ell-1} b_j = 1$, $\sum_{j=0}^{\ell-1} j\, b_j = i$.
   - Per-bit: $b_j \in \{0, 1\}$ for all j.
   - Readout: $M_i = \sum_{j=0}^{\ell-1} b_j\, M_j$, $H_{i+1} = \sum_{j=0}^{\ell-1} b_j\, H_{j+1}$, and if $i > 0$: $H_i = \sum_{j=0}^{\ell-1} b_j\, H_j$.
2) If $i = 0$, set $H_i \leftarrow \mathsf{IV}$.
3) Enforce $H_{i+1} = \text{SHA256Compress}(H_i, M_i)$.

Figure 4. SHA256PERSTEP$_{\text{MC}}$: Per-step SHA-256 checks. Each step only performs a single invocation of the SHA-256 compression function.

Private inputs: Blocks $M_0, \ldots, M_{\ell-1}$; one-hot vector $\mathbf{b} = (b_0, \ldots, b_{\ell-1})$; private index $d \in \{0, \ldots, \ell-1\}$; digest vector $(H_1, \ldots, H_\ell)$.

1) $M \leftarrow M_0 \| \cdots \| M_{\ell-1}$.
2) Enforce canonical padding at d.
3) Enforce one-hot binding:
   - Global: $\sum_{j=0}^{\ell-1} b_j = 1$, $\sum_{j=0}^{\ell-1} j\, b_j = d$.
   - Per-bit: $b_j \in \{0, 1\}$ for all j.
4) Set $H_{d+1} \leftarrow \sum_{j=0}^{\ell-1} b_j\, H_{j+1}$.
5) Return (M, d, H_{d+1}).

Figure 5. SHA256BIND$_{\text{MC}}$: Length-hiding SHA-256 binding circuit. This does not compute any compression function.

4.3. Compact circuit for ECDSA

Verifying an ECDSA signature inside a circuit is typically a heavy operation due to potential field mismatch: the base field of the elliptic curve over which the signature scheme is defined might not match the field over which the proof system is defined. It is well known that matching the two fields leads to fewer constraints for elliptic curve arithmetic [26], [42], [43], [51], [52]. In the context of mDL (see §3.1), ECDSA is defined over the P-256 curve. Intuitively, because the P-256 scalar-field modulus is smaller than the circuit field, these values fit natively and do not require field emulation. The same applies to other popular curves such as secp256k1.

It is also well known that Spartan [15] (with Hyrax

PCS [24]) can be instantiated with any elliptic curve and that one can construct an elliptic curve with a specified group order [53]. In the past, spartan-ecdsa [54] showed how to efficiently prove ECDSA over secp256k1 by using Spartan over secq256k1 (which forms a cycle with secp256k1). Recently, Woo et al. [21] instantiate Spartan [15] with T-256 curve [44], which forms a chain with P-256, to efficiently prove circuits that encode the verification of P-256 ECDSA.

We extend this approach to SplitSpartan and SplitNeutronNova. More notably, we improve it further by eliminating all non-native arithmetic. In particular, while the bulk of signature verification involves arithmetic over the base field of P-256 (which can be expressed efficiently if the proof system is instantiated with T-256), there is work done in the scalar field. This is handled with field emulation in prior work. We address this with a circuit-friendly verification equation and by leveraging the fact that the scalar field of P-256 is smaller than the base field of P-256 (i.e., $n < p$).

Circuit-friendly ECDSA. Let (r, s) be an ECDSA signature on the message digest $z := \text{SHA-256}(M)$, where M denotes the padded message, let G be the standard generator of P-256, and let Q_A denote a public key. Also, let n denote the order of P-256 curve, and p denote the modulus of the base field of P-256 (which equals the scalar field of T-256).

Typically, signature verification requires computing only two group scalar multiplications and some modular arithmetic: check that $r, s \in \{1, \ldots, n-1\}$, $u_1 \leftarrow zs^{-1} \bmod n$ and $u_2 \leftarrow rs^{-1} \bmod n$, then computing the curve point (x_1, y_1) as $u_1 \cdot G + u_2 \cdot Q_A$, and then checking if $r = x_1 \bmod n$. Unfortunately, this requires non-native arithmetic to compute u_1 and u_2 as they are modular multiplications performed over the scalar field of P-256, which is not equal to the scalar field of T-256 (the native field of the circuit). Additionally, s^{-1} must be computed non-natively and one must check that r and s are non-zero and smaller than n.

We instead leverage the fact that $n < p$, so any number in the set $\{0, 1, \ldots, n-1\}$ can be represented with a single value in the circuit's witness (rather than with multiple field elements each holding a limb). Instead of the signature being (r, s), we have the prover provide the signature as $(r, sinv)$ where an honest prover supplies $sinv \leftarrow s^{-1} \pmod n$. The circuit checks that r and $sinv$ are non-zero with a single constraint by witnessing their inverse [55]. Additionally, we ensure that the provided r and $sinv$ are smaller than n by using simple bit decomposition (Appendix A). Additionally, instead of two group scalar multiplications and field emulation, we compute group elements $P_1 \leftarrow z \cdot G$, $P_2 \leftarrow r \cdot Q_A$, and then compute (x_1, y_1) as $sinv \cdot (P_1 + P_2)$. We implement these with Nova's elliptic curve gadgets [56]. Each group scalar multiplication operation uses standard double-and-add scalar multiplication with incomplete addition, so no coordinate conversion or special reduction is required. The only remaining arithmetic concerns the final check $r \equiv x_1 \pmod n$. Because $n < p$, we enforce it with a simple quotient check by witnessing a value q: $r - x_1 = q \cdot n$ and $q < n$. The latter is enforced with a simple gadget based on bit decomposition. Finally, there are some auxiliary checks: we constrain the public key Q_A and the fixed generator G to lie on P-256, require Q_A to be non-identity, and enforce that the point $sinv \cdot (z \cdot G + r \cdot Q_A)$ is not the point at infinity. Figure 6 depicts the ECDSA verification gadget.

Public input: generator G.
Private inputs: signature component r; $sinv$; message digest z; public key $Q_A = (x_{Q_A}, y_{Q_A})$.

1) Domain checks:
   - Enforce G lies on the P-256 curve.
   - Enforce Q_A lies on the P-256 curve.
   - Enforce $G \neq \mathcal{O}$.
   - Enforce $Q_A \neq \mathcal{O}$.
2) Range and non-zero checks:
   - Enforce $0 < r < n$.
   - Enforce $0 < sinv < n$.
   - Enforce $r \cdot r^{-1} = 1$.
   - Enforce $sinv \cdot sinv^{-1} = 1$.
3) Elliptic curve operations:
   - Compute $P_1 \leftarrow z \cdot G$.
   - Compute $P_2 \leftarrow r \cdot Q_A$.
   - Compute $P_3 \leftarrow P_1 + P_2$.
   - Compute $(x_1, y_1) \leftarrow sinv \cdot P_3$.
   - Enforce $(x_1, y_1) \neq \mathcal{O}$.
4) Mod-n equality:
   - Witness $q \in [0, n-1]$.
   - Enforce $r - x_1 = q \cdot n$.

Figure 6. ECDSAVERIFY: ECDSA verification gadget

In our implementation, the circuit has ≈10,000 constraints for a pre-hashed message. In addition to the efficiency benefit, removing non-native arithmetic eliminates an entire class of potential soundness bugs that stem from incorrect limb handling or faulty modular reduction.

5. Lookups for efficient document parsing

Proving statements over digital identity documents with a ZKP requires accessing semantic fields within the document (e.g., date of birth or zip code) and then performing some computation (e.g., whether one is over 18). ISO 18013-5 [50] defines the mDL format in CBOR (§3.1). A naive solution to address this is to build a CBOR parser as a circuit (e.g., as proposed in prior work [20]). However, encoding CBOR rules in-circuit entails significant complexity and costs and is a major source of bugs. Importantly, signature verification only authenticates the raw MSO bytes; the expensive part is recovering semantic fields from those bytes.

Our key insight is that we can assume that the documents issued by an issuer are well-formed so it is unnecessary to perform a full validation of the document (§1). Additionally, we observe that verifiers typically require validation

of only a *subset* of document fields. Thus, we design a circuit that extracts specific fields from byte-addressable data without full document parsing. Although we focus on CBOR because ISO 18013-5 mandates it for mDLs, the same lookup-based pattern applies to any stable byte encoding with unique prefixes and delimiters.

In particular, we require that documents are encoded as key-value maps where each key has a unique prefix and clear delimiters indicating field boundaries. Figure 8 depicts representative mDL field structures that satisfy these requirements. Based on this, our approach leverages the fact that circuits can take non-deterministic inputs: the prover can supply field location *hints*, reducing the problem to verifying the correctness of the extracted values. For the latter, we leverage *lookup arguments* (see §3), which we use to efficiently prove membership of values within the MSO. Once the authenticity of each field is established via lookups, we constrain only the extracted field values according to their semantic meaning. The field-extraction pseudocode is shown in Figure 7, and the end-to-end mDL checks are summarized in Figure 9.

5.1. Parsing and field extraction

Our circuit first verifies that the claimed variables correspond to the values associated with the specified fields in the MSO. The prover non-deterministically constructs two vectors: addr, which specifies document byte positions, and vals, which contains the corresponding bytes obtained by treating the MSO as a lookup table. The field extraction algorithm is described as a series of checks in Figure 7.

Lookup (line 1) checks that the bytes in vals were extracted from the MSO at positions specified in addr, preventing the prover from providing arbitrary byte values.

Prefix (line 3) checks that each field begins with its unique prefix (e.g., `0x69+"deviceKey"` for `deviceKey`), ensuring that extracted field values indeed correspond to the intended fields.

Contiguity (line 4) confirms that field bytes are extracted from consecutive positions in the MSO, preventing substitution attacks where bytes from different document locations could be spliced together.

If any of the checks fail, the circuit is unsatisfiable, and the prover will not be able to generate a valid proof.

5.1.1. Hash-based fields. For fields like `birthDate`, the MSO `valueDigests` structure stores SHA-256 hashes of these field values rather than the plaintext values themselves. To prove properties about such fields, we additionally require the prover to provide the original field value as a private witness (preimage). These preimages are stored outside of the MSO and in a data structure called `issuerSignedItems`. After extracting the digest using the same lookup protocol, we compute SHA-256(preimage) in-circuit using the approach from §4 and constrain it to equal the extracted digest. This enables proving predicates over preimage without revealing it to the verifier.

5.2. Semantic checks

After field extraction and verification, we apply semantic constraints to the validated field values to prove properties without disclosure. At a bare minimum, we constrain the `validUntil` field to exceed the current date, proving the document is unexpired without revealing the exact expiry and use the device public key to verify the device signature for device binding (§3.1). This can be extended to support any semantic check required by the verifier such as age verification (e.g., over 21), residency (e.g., state of issuance), document type, issuer authenticity, or revocation status.

Example: validity period. We now provide a concrete example for how the checks come together for the validity period. Let t_{now} be the current date (supplied as public input). The circuit performs the following checks:

1) **Lookup:** Verify $\textsc{Lookup}(\mathsf{addr}, \mathsf{vals}, \text{MSO}) = 1$ binds the hinted positions and bytes to the authenticated MSO.
2) **Prefix:** Check the unique prefix for `validUntil` at the hinted offset (Figure 8).
3) **Contiguity:** Enforce adjacency for the bytes comprising the prefix and the ISO 8601 datetime.
4) **Semantic:** Map the datetime bytes to an integer timestamp $t_{\text{validUntil}}$ using a fixed, constant-branch arithmetic parser. Enforce $t_{\text{validUntil}} > t_{\text{now}}$.

The circuit reveals only the outcome of the comparison; everything else remains hidden.

6. Adding ZK with Succinct NovaBlindFold

We now describe a general recipe that makes SplitSpartan and SplitNeutronNova zero-knowledge. Our approach is general and extends easily to any other public-coin polynomial interactive oracle proof (PIOP) [57], such as those of Lasso [30], Jolt [49], and Twist and Shout [46]. Our approach leverages representing the verifier's checks in these protocols with a split-committed R1CS instance and then folds that instance with a randomly sampled split-committed relaxed R1CS instance, in-line with our fold-and-reuse paradigm. In this section, "verifier" refers to the proof-system verifier, not the relying party in Figure 1. We start with a slight generalization of the committed relaxed R1CS [26] (§3, Def. 3.1): the single witness commitment $\overline{W}$ is replaced by a vector of commitments $(\overline{W}_1, \ldots, \overline{W}_\ell)$.

Definition 6.1 (Split Committed RR1CS (SCRR1CS)). Consider a finite field $\mathbb{F}$ and a commitment scheme com defined over $\mathbb{F}$. Let the public parameters consist of size bounds $m, n, \ell, t, s \in \mathbb{N}$ with $m > t \geq \ell$ and $s = m/\ell$, and commitment parameters pp for vectors

Private inputs: MSO bytes; $(\mathsf{addr}, \mathsf{vals})$; per-field descriptors $(\mathsf{pos}_f, \mathsf{prefix}_f, |\mathsf{data}_f|)$.

1) **Lookup:** Enforce $\textsc{Lookup}(\mathsf{addr}, \mathsf{vals}; \mathrm{MSO}) = 1$.
2) **For each field** $f \in \{\texttt{deviceKey}, \texttt{validUntil}\}$:
   - **Prefix:** Enforce $\mathsf{vals}[\mathsf{pos}_f : \mathsf{pos}_f + |\mathsf{prefix}_f|] = \mathsf{prefix}_f$.
   - **Contiguity:** Enforce $\mathsf{addr}[t] = \mathsf{addr}[t-1] + 1$ for $t \in [\mathsf{pos}_f, \mathsf{pos}_f + |\mathsf{data}_f|]$.
   - **Extract:** $\mathsf{data}_f \leftarrow \mathsf{vals}[\mathsf{pos}_f + |\mathsf{prefix}_f| : \mathsf{pos}_f + |\mathsf{prefix}_f| + |\mathsf{data}_f|]$.
3) **Return:** extracted field data $\{\mathsf{data}_f\}_f$.

Figure 7. EXTRACTFIELDS: lookup-based field extraction

Field	Size	Structure
`birth_date`	Variable	Tag 24 (0xd8 0x18) + length prefix + IssuerSignedItem data
`device_public_key`	85 bytes	0x69+"deviceKey" (10B prefix) + map header + P-256 coordinates
`valid_until`	33 bytes	0x6a+"validUntil" (11B prefix) + 0xc0 + 0x74 + ISO 8601 datetime

Figure 8. CBOR field structures with unique prefixes.

of length up to m. The SCRR1CS structure consists of sparse matrices $A, B, C \in \mathbb{F}^{m \times m}$ with at most $n = \Omega(m)$ non-zero entries in each matrix. An *instance* is a tuple $(\overline{E}, u, \overline{W}_1, \ldots, \overline{W}_\ell, x)$, where $\overline{E}$ and each $\overline{W}_i$ are commitments, $u \in \mathbb{F}$, and $x \in \mathbb{F}^t$ are public inputs and outputs.

An instance $(\overline{E}, u, \overline{W}_1, \ldots, \overline{W}_\ell, x)$ is *satisfied* by a witness $(E, r_E, W_1, \ldots, W_\ell, r_{W_1}, \ldots, r_{W_\ell})$ if

$$\begin{aligned} \overline{E} &= \mathsf{com}(\mathsf{pp}, E, r_E), \\ \overline{W}_i &= \mathsf{com}(\mathsf{pp}, W_i, r_{W_i}) && \text{for all } i \in [\ell], \\ |W_i| &= s && \text{for all } i \in [\ell], \\ (AZ) \circ (BZ) &= u\,(CZ) + E, \end{aligned}$$

where $Z = (W_1, \ldots, W_\ell,\, 1,\, x)$.

Split-committed R1CS (SCR1CS) is SCRR1CS with $u = 1$, $E = 0$, $r_E = 0$, and $\overline{E} = \mathsf{com}(\mathsf{pp}, 0, 0)$.

6.1. An overview of Succinct NovaBlindFold

6.1.1. Replacing the verifier's checks with SCR1CS. Consider a public-coin PIOP that proceeds in ℓ rounds, where in each round the prover sends a vector encoding a polynomial, and the verifier sends a challenge. At the end, the verifier performs a set of checks using the values it receives and its own challenges. Given a PIOP between a prover $\mathcal{P}$ and $\mathcal{V}$, we consider a transformed protocol between $\mathcal{P}^\star$ and $V^\star$:

1) When $\mathcal{P}$ sends a message m to $\mathcal{V}$ in the PIOP, $\mathcal{P}^\star$ sends $\mathsf{com}(\mathsf{pp}, m)$ to $\mathcal{V}^\star$.
2) When $\mathcal{V}$ sends a challenge c to $\mathcal{P}$ in the PIOP, $\mathcal{V}^\star$ sends c to $\mathcal{P}^\star$.

Unfortunately, $\mathcal{V}^\star$ can no longer perform checks that $\mathcal{V}$ would have done. However, we can write a split-committed R1CS structure that encodes the checks of the verifier $\mathcal{V}$ on the prover's plain-text messages and the public-coin challenges.[9] Note that $\mathcal{V}^\star$ can turn the commitments sent by $\mathcal{P}^\star$ and its challenges into a split-committed R1CS instance on its own. Similarly, the prover aggregates all of its vectors into $(W_1, \ldots, W_\ell)$. Except for a small soundness error (arising from the use of commitments), it is easy to see that the satisfiability of the instance-witness pair is equivalent to the PIOP verifier accepting.

6.1.2. Making the constraint system succinct. We now apply the above recipe to Spartan's PIOP for R1CS (the same applies to our variant SplitSpartan and SplitNeutronNova).

Suppose that the R1CS instance u on which Spartan is run contains m constraints, $m - |x|$ witness variables, and n non-zero entries in R1CS matrices. In non-preprocessing Spartan, the prover sends its purported witness polynomial in the first message. Then, the prover and the verifier run two sequential invocations of the sum-check protocol [58] (for a total of $2 \log m$ rounds). At the end, the verifier queries the witness polynomial and the polynomials encoding the R1CS structure at a single point to perform the final check. If we encode the verifier's checks here as a split-committed R1CS instance $\mathsf{u}^\star$ as discussed earlier, the size of the split-committed R1CS is $\Omega(n)$ constraints. If we were to apply NovaBlindFold (§2.1.2) to the instance $\mathsf{u}^\star$, it is no better than applying NovaBlindFold to the original instance u.

We now describe how to make the number of constraints and witness variables in $\mathsf{u}^\star$ to be $O(\log m)$.

First, we observe that the challenges and R1CS matrices are public (i.e., known to the verifier). So, the split-committed R1CS can simply take as public input purported evaluations of polynomials encoding R1CS matrices and the verifier can check these claimed public values. This brings down the size of the constraint system from $O(n)$ to $O(m)$.

9. For most PIOPs of interest (including Vega's and others mentioned earlier), the verifier's checks can be efficiently represented with R1CS.

Public inputs: issuer key Q_I; device digest H_{dev}; date (Y, M, D); age threshold τ.
Private inputs: MSO bytes; $(\mathsf{addr}, \mathsf{vals})$; issuer sig $(r_I, sinv_I)$; device sig $(r_D, sinv_D)$; DOB preimage.

1) Field extraction:
   - $(\mathsf{data}_{\texttt{deviceKey}}, \mathsf{data}_{\texttt{validUntil}}) \leftarrow$ EXTRACTFIELDS(MSO, addr, vals).
   - Parse (p_x, p_y) from $\mathsf{data}_{\texttt{deviceKey}}$.
2) Valid-until freshness: derive t_{end} from $\mathsf{data}_{\texttt{validUntil}}$ and enforce $(Y, M, D) < t_{\text{end}}$.
3) DOB hashing:
   - Read $h_{\text{birth}} = \text{MSO}[\texttt{valueDigests}][\texttt{org.iso.18013.5.1}][\texttt{birth\_date}]$.
   - Enforce $\text{SHA256}_{\text{SC}}(\text{DOB\_preimage}) = h_{\text{birth}}$.
4) Age threshold: derive age from DOB_preimage and the public date (Y, M, D); enforce $\mathsf{age} \geq \tau$.
5) MSO hashing: compute $H_{\text{MSO}} \leftarrow \text{SHA256}_{\text{SC}}(\text{MSO})$.
6) Signature verifications:
   - Enforce $\text{ECDSAVERIFY}((r_I, sinv_I), H_{\text{MSO}}, Q_I) = 1$.
   - Enforce $\text{ECDSAVERIFY}((r_D, sinv_D), H_{\text{dev}}, (p_x, p_y)) = 1$.

(a) The circuit used in Vega_{SC}.

Public inputs: issuer key Q_I; device digest H_{dev}; date (Y, M, D); age threshold τ.
Private inputs: MSO bytes; $(\mathsf{addr}, \mathsf{vals})$; issuer sig $(r_I, sinv_I)$; device sig $(r_D, sinv_D)$; DOB preimage.

1) Field extraction:
   - $(\mathsf{data}_{\texttt{deviceKey}}, \mathsf{data}_{\texttt{validUntil}}) \leftarrow$ EXTRACTFIELDS(MSO, addr, vals).
   - Parse (p_x, p_y) from $\mathsf{data}_{\texttt{deviceKey}}$.
2) Valid-until freshness: derive t_{end} from $\mathsf{data}_{\texttt{validUntil}}$ and enforce $(Y, M, D) < t_{\text{end}}$.
3) DOB hashing:
   - Read $h_{\text{birth}} = \text{MSO}[\texttt{valueDigests}][\texttt{org.iso.18013.5.1}][\texttt{birth\_date}]$.
   - $H_{\text{birth}} \leftarrow \text{SHA256BIND}_{\text{MC}}(\text{DOB\_preimage})$; enforce $H_{\text{birth}} = h_{\text{birth}}$.
4) Age threshold: derive age from DOB_preimage and (Y, M, D); enforce $\mathsf{age} \geq \tau$.
5) MSO hashing: compute $H_{\text{MSO}} \leftarrow \text{SHA256BIND}_{\text{MC}}(\text{MSO})$.
6) Signature verifications:
   - Enforce $\text{ECDSAVERIFY}((r_I, sinv_I), H_{\text{MSO}}, Q_I) = 1$.
   - Enforce $\text{ECDSAVERIFY}((r_D, sinv_D), H_{\text{dev}}, (p_x, p_y)) = 1$.

(b) Core circuit used in Vega_{MC}.

Figure 9. End-to-end mDL checks

Second, the $O(m)$ costs arise because the first message sent by the prover is $O(m)$-sized: the prover sends a witness polynomial $\widetilde{w}$ to instance u. The rest of the messages sent by the prover are $O(\log m)$-sized in total. Fortunately, the circuit only needs an evaluation of the $O(m)$-sized polynomial at a random point r determined by the verifier's challenges, i.e., $y \leftarrow \widetilde{w}(r)$ (Def. 3.4). The circuit also needs to ensure that y is indeed the correct evaluation of the polynomial sent earlier $\widetilde{w}$ at r. Naively, encoding this in the circuit requires $O(m)$ constraints. Instead, we do the following: (1) the witness polynomial is not taken as witness to the split-committed R1CS; the prover still sends a commitment to $\widetilde{w}$ (denote it as $C_{\widetilde{w}}$) as its first message, so the verifier's first challenge depends on it; and (2) the prover sends a commitment to the claimed evaluation (denote it as C_y) in the last round. The split-committed R1CS consumes the value y and performs the appropriate check. Crucially, the size of the split-committed R1CS is now only $O(\log m)$ constraints. We are now left with ensuring that C_y is a commitment to the evaluation of the polynomial underneath $C_{\widetilde{w}}$ at r. For this, we invoke the zero-knowledge evaluation argument of the polynomial commitment scheme. Many schemes [24], [59], [60] support such a functionality.

Finally, we incorporate a number of now-standard optimizations to reduce the size of the constraint system (e.g., make heavy use of "free" additions in R1CS). The number of constraints in the split-committed R1CS is only a few hundred constraints even when the original constraint system has a million or more constraints.

6.1.3. Run NovaBlindFold on the succinct instance. We now run the NovaBlindFold protocol [27], which we covered in depth in §2.1.2, on the succinct split-committed R1CS instance-witness pair, releasing only the split-committed instance $\mathsf{u}^\star$, a random relaxed instance, and the folded witness to the verifier. By the guarantees of NovaBlindFold and

the zero-knowledge evaluation argument of the polynomial commitment scheme, the verifier simply learns that the split-committed instance is satisfying. This in turn implies that the original PIOP verifier is accepting—without the verifier learning the transcript of the PIOP.

6.2. Details of Succinct NovaBlindFold

We now provide a formal description of Succinct NovaBlindFold. Suppose that we have a PIOP for an instance-witness pair (u, w). Let the public IO of u be x. Let s denote the R1CS structure of the split-committed R1CS encoding the PIOP verifier's checks. Suppose that we have applied optimizations to move linear-sized polynomial evaluations outside the constraint system.

1) $\mathcal{P}^\star$: Run the public-coin PIOP where during round i, send $\mathsf{com}(\mathsf{pp}, W_i, r_{W_i})$ and receive a challenge c_i. Record the full transcript $(W_1, r_1, c_1, \ldots, W_\ell, r_{W_\ell}, c_\ell)$. Form the SCR1CS witness $\mathsf{w}^\star = (W_1, \ldots, W_\ell, r_{W_1}, \ldots, r_{W_\ell})$.
2) $\mathcal{V}^\star$: Form the SCR1CS instance $\mathsf{u}^\star = (\overline{W}_1, \ldots, \overline{W}_\ell, x^\star)$, where $x^\star = (c_1, \ldots, c_\ell, x)$.
3) $\mathcal{P}^\star$: Sample a random split-committed relaxed R1CS instance-witness pair $(\mathsf{u}_r, \mathsf{w}_r)$ for structure s (Kothapalli and Setty [27] provide details of this procedure for committed relaxed R1CS and it generalizes to split-committed relaxed R1CS). Send u_r to the verifier.
4) $\mathcal{P}^\star \leftrightarrow \mathcal{V}^\star$: Run Nova's folding scheme to fold u_r with $\mathsf{u}^\star$. $\mathcal{V}^\star$ computes u_{out} and $\mathcal{P}^\star$ computes w_{out}.
5) $\mathcal{P}^\star$: Send w_{out} to the verifier.
6) $\mathcal{V}^\star$: Check that w_{out} is a satisfying witness to u_{out}. If all the checks pass, accept, otherwise reject.

Optimization. To optimize proof length, instead of sending the folded witness to the verifier in step 5, we can invoke Spartan to prove the knowledge of a satisfying folded witness to u_{out}. Crucially, this Spartan invocation does not need to provide zero-knowledge.

Intuition for zero-knowledge. Kothapalli and Setty [27, Lemma 9] formally prove that NovaBlindFold (i.e., the use of Nova's folding scheme to fold a committed R1CS instance-witness pair with a randomly sampled committed relaxed instance-witness pair and then releasing the folded witness to the verifier) provides honest-verifier zero-knowledge. As a result and due to the zero-knowledge property of the polynomial evaluation argument, the verifier in Succinct NovaBlindFold learns nothing about the PIOP transcript beyond validity.

7. Implementation

We provide high-performance implementations of our base proving schemes SplitSpartan and SplitNeutronNova in ≈8.5K lines of Rust atop the Spartan implementation in `spartan2` [61], which also includes general-purpose performance optimizations. We use Hyrax-PCS [24] for polynomial commitments and extend `bellpepper` [37] to support the construction of R1CS circuits with split witness vectors as well as pause/resume witness synthesis (and commitment generation). These modifications allow the prover to split its work into a precomputation phase and an online phase. Next, we implement Vega's circuits in ≈4.9K lines of Rust using RustCrypto [62] for cryptographic primitives including SHA-256 hashing and ECDSA over P-256.

8. Experimental evaluation

We evaluate Vega's implementation (§7) for prover and verifier latency, proof size, key sizes, and peak memory.

Experimental setup. As baselines, we compare with two recent schemes with similar functionality: Longfellow [20] and Crescent [19]. We evaluate them using their public reference implementations [63], [64]. We run our experiments on a Microsoft Azure Standard_F16as_v7 VM (16 vCPUs, 64 GB RAM) in West US 3. A key distinction is that Crescent requires a per-circuit trusted setup, while both Vega and Longfellow employ a transparent setup (§1.1). We focus on the concrete scenario of proving that a credential holder is at least a certain age (in addition to the required checks to ensure the credential is valid). The proof also attests to the possession of a secure element (i.e., device binding) to which the credential was issued. In our experiments, we use a mock device that signs a nonce provided by the verifier.

8.1. Latency results

Figure 10 reports end-to-end latency for proving a 1920 byte mDL MSO across four phases:

- **Setup** takes in global parameters (e.g., MSO length bound) and outputs keys for the prover and the verifier. This cost is incurred once per circuit configuration.
- **Precomputation** precomputes reusable state from static credential components (MSO bytes, issuer signature, and lookup positions), enabling faster proofs during presentation. This cost is incurred once per credential.
- **Proving** combines the precomputed state with dynamic inputs (e.g., current date, device signature) to generate a proof upon each credential presentation. This step is latency-critical for users, who may present credentials frequently (e.g., when browsing the web).
- **Verification** verifies the proof against the verifier key. This step is latency-critical for application providers who may need to handle many concurrent presentations, so latency directly governs throughput and service cost.

Setup. Vega_{SC} requires 606 ms and Vega_{MC} only 51 ms, compared to 5,590 ms for Longfellow and 106,158 ms for Crescent (16 threads). Crescent's setup cost stems from Groth16's expensive trusted setup. Vega_{MC} is 110× faster

Scheme	Thread	Latency (ms)				Size (kB)			Transparent
		Setup	*Precompute*	*Prove*	*Verify*	*Proof*	pk	vk	*Setup*
Crescent	16	106 158	25 653	182	89	16	710 565	1	✗
Longfellow	1	5590	—	501	244	325	202	202	✓
Vega_{SC}	1	726	351	325	127	91	6588	6587	✓
	2	660	250	231	79				
	4	630	209	187	54				
	8	618	165	161	43				
	16	606	158	155	40				
Vega_{MC}	1	210	390	237	57	108	464	464	✓
	2	122	209	159	36				
	4	80	118	105	25				
	8	61	84	92	23				
	16	51	65	92	23				

Figure 10. Performance comparison for a 1920 byte MSO (latency in ms, sizes in kB). Gains diminish beyond 4 threads due to serial work (e.g., witness generation) and the computation being too small to benefit from additional cores.

than Longfellow and 2,082× faster than Crescent; Vega_{SC} is 9.2× faster than Longfellow. Indeed, Vega_{MC} reduces setup latency by 92% because only one SHA-256 sub-circuit must be set up, as opposed to setting up the constraints for each iteration of the SHA-256 computation (§4.2).

Precomputation. Crescent reduces proving latency by precomputing 26 s of work per credential (16 threads), whereas Vega requires only 158 ms in Vega_{SC} and 65 ms in Vega_{MC}. Longfellow does not use precomputation, performing all proving work online. Overall, Vega_{MC} is 395× faster than Crescent for precomputation (162× for Vega_{SC}).

Proving. Vega_{MC} requires 237 ms of online prover work single-threaded, 2× less than Longfellow's 501 ms. Vega_{SC} also requires less work at 325 ms. Note that much of this improvement comes as a result of one-time precomputation, which takes 390 ms under Vega_{MC} and 351 ms under Vega_{SC}. The lower online prover latency of Vega_{MC} over Vega_{SC} shows the benefit of folding uniform computation (§2, §4.2).

With 16 threads, latency drops to 92 ms for Vega_{MC} and 155 ms for Vega_{SC}, both surpassing Crescent's 182 ms while retaining transparency. Most of this speedup is already realized at 4 threads (105 ms for Vega_{MC}), with diminishing returns beyond that (Figure 10). Crescent already uses precomputation, so Vega's advantage there also comes from its circuit and proof-system design. Finally, it is unclear how Longfellow's latency scales with additional threads.

Verification. Verification is efficient in both Vega_{SC} and Vega_{MC}: even single-threaded, Vega_{MC} verifies in 57 ms and Vega_{SC} in 127 ms, which is 4.3× and 1.9× faster than Longfellow's 244 ms, respectively. With 16 threads, verification drops to 23 ms for Vega_{MC} and 40 ms for Vega_{SC}, both outperforming Crescent (89 ms) as well. For application providers, verifier latency is critical: services often validate many users concurrently, so shaving tens to hundreds of milliseconds per check reduces queueing, lowers server cost, and improves user experience at peak load. Because the proof replaces the raw ID transfer (§1.2), a faster verifier helps avoid the privacy and security pitfalls of storing IDs while meeting real-time SLAs in production.

Peak memory. Peak memory was about 100 MB during precomputation and 562 MB during online proving for Vega_{MC}. These measurements suggest that the presentation-time phase is suitable for memory-constrained devices such as mobile devices.

Scaling with MSO size. Figure 11 shows performance as the MSO size increases. Setup cost grows with input length in Vega_{SC}, while remaining nearly constant in Vega_{MC} because the constraints for only one SHA-256 per-step circuit must be set up (§4.2), as opposed to the unrolled SHA-256 computation. Precomputation, proving, and verification scale similarly across Vega_{SC} and Vega_{MC}.

8.2. Proof sizes and key sizes

At an MSO of 1920 bytes, Vega_{SC} produces 91 kB proofs and Vega_{MC} 108 kB proofs. Both are substantially smaller than Longfellow's 325 kB, though larger than Crescent's 16 kB. Crescent achieves compactness by using Groth16, but at the cost of a per-circuit trusted setup. As Figure 11 shows, proof sizes in both Vega_{SC} and Vega_{MC} scale sublinearly with MSO length. In practice, Vega strikes a balance: proofs are 3.6× smaller than Longfellow for Vega_{SC} and 3× smaller for Vega_{MC}, while retaining transparency.

Figure 10 reports proving and verifier key sizes. Vega_{SC} requires 6,588 kB for both pk and vk, while Vega_{MC} reduces to 464 kB for both (pk and vk). Longfellow's keys are smaller (202 kB), but Crescent's design produces a large prover key (710,565 kB) and a minimal 1 kB verifier key. Importantly, Crescent's compact verifier comes only with the cost of a per-circuit trusted setup. The use of folding dramatically improves Vega key sizes relative to Vega_{SC} and stabilizes them as input size increases, since the set of circuits is

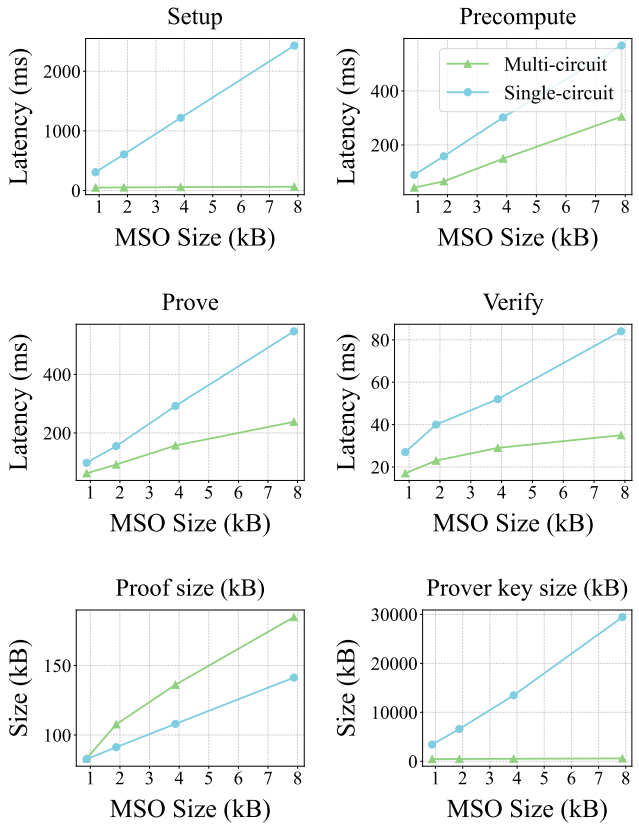

Figure 11. Performance across MSO sizes for all Vega operations.

fixed. Concretely, Vega_{MC}'s prover key is $2.3\times$ larger than Longfellow's, but $1{,}531\times$ smaller than Crescent's.

While our baselines each only satisfy a subset of the goals in §1, Vega achieves them all: it requires no issuer changes and uses a transparent setup, while delivering hundred-millisecond client proving and tens-of-milliseconds verification. Vega_{MC} further improves on Vega_{SC} by reducing setup cost and prover key size substantially and keeps both largely insensitive to input size. Proofs are smaller than Longfellow and competitive overall, while Crescent's smaller proofs come at the cost of a trusted setup. These results indicate that Vega provides client-friendly proving and server-efficient verification without trusted setup, making it well-suited for real deployments.

9. Related work

Anonymous credentials. Chaum [65] introduced anonymous credentials and Lysyanskaya et al. [66] formalized them with constructions. Since then, a rich line of work [67], [68], [69], [70], [71], [72], [73], [74], [75] has been developed. Our work instead creates anonymous credentials from *existing* credentials, enabling backwards compatibility without new issuance infrastructure.

Proofs about existing credentials. Prior work falls into two categories. *One-shot* systems directly prove statements about the issuer-signed legacy credential at presentation time; Cinderella [16], Crescent [19], Longfellow [20], and Vega follow this approach. *Convert-then-prove* systems first prove possession of a legacy credential and then issue a new privacy-preserving credential for later use; zk-creds [17] follows this approach, using either issuer-signed derived credentials or Merkle-tree-backed ones. In that model, the expensive proof over the legacy document is amortized across later presentations. Additionally, zk-creds leverages rerandomization capabilities of Groth16 zkSNARK [14] to minimize prover computation.

Crescent also shifts credential-specific work out of presentation via reusable precomputation. Vega's contribution is to make that reusable precomputation compatible with a transparent setup. zkLogin [76] authenticates to blockchains using existing OpenID credentials. Vega's proof systems can be used to improve the performance of these works.

Godden et al. [77] design an R1CS protocol to parse and prove fields from national eID cards, but it is significantly slower than Vega. BBS# [78] targets eIDAS 2.0 by making ACs pairing-free over ECDSA/ECSDSA and suitable for secure elements, but requires issuer infrastructure changes.

Proofs about committed documents. Beyond credential presentations, orthogonal work studies proofs about committed documents using Nova [26] as the proof system. Reef [18] gives fast succinct non-interactive regex proofs over committed strings, while Coral [22] provides fast non-interactive CFG proofs. Their focus is on parsing the documents and asserting predicates over them. They assume the verifier holds a commitment signed by a trusted authority and verifies the signature locally; thus, for ZKPs over existing credentials, they provide no unlinkability guarantees.

Proofs of SHA-256 and ECDSA. Many works target standard primitives such as SHA-256 and P-256 ECDSA, including ring/group signatures over existing keys [44] and efficient proofs of possession [21]. Vega improves on these works by eliminating all non-native arithmetic for ECDSA verification (§4.3) and hiding the length of the message hashed beyond a global upper bound (§4.1, §4.2). Additionally, Vega's setup work and the size of the circuit do not depend on the size of the message hashed.

References

[1] J. Peters, "Discord says 70,000 users may have had their government ids leaked in breach," https://www.theverge.com/news/797051/discord-government-ids-leaked-data-breach, 2025.

[2] B. Ortutay, "Tea, an app for women to safely talk about men they date, has been breached, user IDs exposed," https://apnews.com/article/tea-app-women-breach-ids-selfies-dating-5433d5929bdfeb73f495d4775580a55f, 2025.

[3] K. Collier, "New ChatGPT teen-safety measures will include age prediction and verification, OpenAI says," https://www.nbcnews.com/news/amp/rcna231637, 2025.

[4] B. D. Griffiths, "OpenAI says it's working to tell if a user is under 18 and will send them to an 'age-appropriate' ChatGPT," https://www.yahoo.com/news/articles/openai-says-working-tell-user-152212475.html, 2025.

[5] J. Beser, "Extending our built-in protections to more teens on YouTube," https://blog.youtube/news-and-events/extending-our-built-in-protections-to-more-teens-on-youtube/, 2025.

[6] European Commission, "Commission makes available an age verification blueprint," https://digital-strategy.ec.europa.eu/en/news/commission-makes-available-age-verification-blueprint, 2025.

[7] D. Milmo and R. Booth, "What are the new UK online safety rules and how will age checks on adult content be enforced?" https://www.theguardian.com/technology/2025/jul/24/what-are-the-new-uk-online-safety-rules-and-how-will-they-be-enforced, 2025.

[8] N. Al-Sibai, "OpenAI's ChatGPT agent clicks 'I Am Not a Robot' button without a wink of irony," https://futurism.com/chatgpt-agent-captcha, 2025.

[9] L. Bonos, "Visit this store for a free iris scan to 'prove' you're human and not AI," https://www.washingtonpost.com/technology/2025/05/02/world-id-iris-biometric-altman, 2025.

[10] "Get verified with CLEAR on LinkedIn," https://www.clearme.com/partner-detail-pages/linkedin, 2025.

[11] S. Goldwasser, S. Micali, and C. Rackoff, "The knowledge complexity of interactive proof-systems," in *STOC*, 1985.

[12] J. Kilian, "A note on efficient zero-knowledge proofs and arguments (extended abstract)," in *STOC*, 1992.

[13] R. Gennaro, C. Gentry, B. Parno, and M. Raykova, "Quadratic span programs and succinct NIZKs without PCPs," in *EUROCRYPT*, 2013.

[14] J. Groth, "On the size of pairing-based non-interactive arguments," in *EUROCRYPT*, 2016.

[15] S. Setty, "Spartan: Efficient and general-purpose zkSNARKs without trusted setup," in *CRYPTO*, 2020.

[16] A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, and B. Parno, "Cinderella: Turning shabby X.509 certificates into elegant anonymous credentials with the magic of verifiable computation," in *S&P*, 2016.

[17] M. Rosenberg, J. White, C. Garman, and I. Miers, "zk-creds: Flexible anonymous credentials from zkSNARKs and existing identity infrastructure," in *S&P*, 2023.

[18] S. Angel, E. Ioannidis, E. Margolin, S. Setty, and J. Woods, "Reef: Fast succinct non-interactive zero-knowledge regex proofs," in *USENIX Security*, 2024.

[19] C. Paquin, G.-V. Policharla, and G. Zaverucha, "Crescent: Stronger privacy for existing credentials," in *Cryptology ePrint Archive*, 2024.

[20] M. Frigo and A. Shelat, "Anonymous credentials from ECDSA," in *Cryptology ePrint Archive*, 2024.

[21] A. P. Y. Woo, A. Ozdemir, C. Sharp, T. Pornin, and P. Grubbs, "Efficient proofs of possession for legacy signatures," in *S&P*, 2025.

[22] S. Angel, S. Celi, E. Margolin, P. Mishra, M. Sander, and J. Woods, "Coral: Fast succinct non-interactive zero-knowledge CFG proofs," in *S&P*, 2026.

[23] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: Nearly practical verifiable computation," in *S&P*, 2013.

[24] R. S. Wahby, I. Tzialla, A. Shelat, J. Thaler, and M. Walfish, "Doubly-efficient zkSNARKs without trusted setup," in *S&P*, 2018.

[25] S. Ames, C. Hazay, Y. Ishai, and M. Venkitasubramaniam, "Ligero: Lightweight sublinear arguments without a trusted setup," in *CCS*, 2017.

[26] A. Kothapalli, S. Setty, and I. Tzialla, "Nova: Recursive Zero-Knowledge Arguments from Folding Schemes," in *CRYPTO*, 2022.

[27] A. Kothapalli and S. Setty, "HyperNova: Recursive arguments for customizable constraint systems," in *CRYPTO*, 2024.

[28] A. Kate, G. M. Zaverucha, and I. Goldberg, "Constant-size commitments to polynomials and their applications," in *ASIACRYPT*, 2010.

[29] A. Kothapalli and S. Setty, "NeutronNova: Folding everything that reduces to zero-check," in *Cryptology ePrint Archive*, 2024.

[30] S. Setty, J. Thaler, and R. Wahby, "Unlocking the lookup singularity with Lasso," in *EUROCRYPT*, 2024.

[31] S. Setty, B. Braun, V. Vu, A. J. Blumberg, B. Parno, and M. Walfish, "Resolving the conflict between generality and plausibility in verified computation," in *EuroSys*, 2013.

[32] E. Ben-Sasson, A. Chiesa, D. Genkin, E. Tromer, and M. Virza, "SNARKs for C: Verifying program executions succinctly and in zero knowledge," in *CRYPTO*, 2013.

[33] S. Setty, "HyperNova: Recursive arguments for customizable constraint systems [video]," Science of Blockchain Conference. https://www.youtube.com/watch?v=0C3HhWNCpls, 2023.

[34] D. Boneh and B. Chen, "Latticefold+: Faster, simpler, shorter lattice-based folding for succinct proof systems," in *CRYPTO*, 2025.

[35] A. Arun and S. Setty, "Nebula: Proving machine executions via folding schemes," in *S&P*, 2026.

[36] A. Ozdemir, E. Laufer, and D. Boneh, "Volatile and persistent memory for zkSNARKs via algebraic interactive proofs," in *S&P*, 2025.

[37] "bellpepper," https://crates.io/crates/bellpepper.

[38] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *CRYPTO*, 1986.

[39] M. Ben-Or, O. Goldreich, S. Goldwasser, J. Håstad, J. Kilian, S. Micali, and P. Rogaway, "Everything provable is provable in zero-knowledge," in *CRYPTO*, 1988.

[40] R. Cramer and I. Damgård, "Zero-knowledge proofs for finite field arithmetic, or: Can zero-knowledge be for free?" in *CRYPTO*, 1998.

[41] C. Bormann and P. Hoffman, "Concise binary object representation (CBOR)," https://www.rfc-editor.org/rfc/rfc8949, 2020.

[42] C. Costello, C. Fournet, J. Howell, M. Kohlweiss, B. Kreuter, M. Naehrig, B. Parno, and S. Zahur, "Geppetto: Versatile verifiable computation," in *S&P*, 2015.

[43] J. Lee, K. Nikitin, and S. Setty, "Replicated state machines without replicated execution," in *S&P*, 2020.

[44] A. Faz-Hernández, W. Ladd, and D. Maram, "ZKAttest: Ring and group signatures for existing ECDSA keys," in *SAC*, 2021.

[45] U. Haböck, "Multivariate lookups based on logarithmic derivatives," in *Cryptology ePrint Archive*, 2022.

[46] S. Setty and J. Thaler, "Twist and Shout: Faster memory checking via one-hot addressing and increments," in *Cryptology ePrint Archive*, 2025.

[47] A. Kothapalli and B. Parno, "Algebraic reductions of knowledge," in *CRYPTO*, 2023.

[48] A. Gabizon and Z. J. Williamson, "plookup: A simplified polynomial protocol for lookup tables," *Cryptology ePrint Archive*, 2020.

[49] A. Arun, S. Setty, and J. Thaler, "Jolt: SNARKs for virtual machines via lookups," in *EUROCRYPT*, 2024.

[50] P. I.-C. D. Licence, "Part 5: Mobile driving licence (mDL) application," *International Organization for Standardization, Standard ISO/IEC FDIS*, 2021.

[51] E. Ben-Sasson, A. Chiesa, E. Tromer, and M. Virza, "Scalable zero knowledge via cycles of elliptic curves," in *CRYPTO*, 2014.

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

[53] R. Bröker and P. Stevenhagen, "Efficient CM-constructions of elliptic curves over finite fields," *Math. Comp.*, vol. 76, 2007.

[54] "Spartan-ecdsa," https://github.com/personaelabs/spartan-ecdsa.

[55] S. Setty, V. Vu, N. Panpalia, B. Braun, A. J. Blumberg, and M. Walfish, "Taking proof-based verified computation a few steps closer to practicality," in *USENIX Security*, 2012.

[56] "Nova," https://github.com/Microsoft/Nova.

[57] B. Bünz, B. Fisch, and A. Szepieniec, "Transparent SNARKs from DARK compilers," in *EUROCRYPT*, 2020.

[58] C. Lund, L. Fortnow, H. Karloff, and N. Nisan, "Algebraic methods for interactive proof systems," in *FOCS*, 1990.

[59] J. Lee, "Dory: Efficient, transparent arguments for generalised inner products and polynomial commitments," in *TCC*, 2021.

[60] S. Setty and J. Lee, "Quarks: Quadruple-efficient transparent zkSNARKs," in *Cryptology ePrint Archive*, 2020.

[61] "Spartan2," https://github.com/microsoft/spartan2.

[62] "RustCrypto," https://github.com/rustcrypto.

[63] "Longfellow," https://github.com/google/longfellow-zk.

[64] "Crescent," https://github.com/microsoft/crescent-credentials.

[65] D. Chaum, "Security without identification: Transaction systems to make big brother obsolete," in *CACM*, 1985.

[66] A. Lysyanskaya, R. L. Rivest, A. Sahai, and S. Wolf, "Pseudonym systems," in *SAC*, 1999.

[67] S. Brands, *Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy*. MIT Press, 2000.

[68] C. Paquin and G. Zaverucha, "U-Prove cryptographic specification v1.1," Microsoft Research Technical Specification, 2023.

[69] J. Camenisch and A. Lysyanskaya, "An efficient system for non-transferable anonymous credentials, with optional anonymity revocation," in *EUROCRYPT*, 2001.

[70] J. Camenisch and E. V. Herreweghen, "Design and implementation of the *idemix* anonymous credential system," in *CCS*, 2002.

[71] J. Camenisch, M. Drijvers, and A. Lehmann, "Anonymous attestation using the strong Diffie Hellman assumption revisited," in *TRUST*, 2016.

[72] M. Chase, T. Perrin, and G. Zaverucha, "The signal private group system and anonymous credentials supporting efficient verifiable encryption," in *CCS*, 2020.

[73] T. Looker, V. Kalos, A. Whitehead, and M. Lodder, "The BBS signature scheme," https://datatracker.ietf.org/doc/draft-irtf-cfrg-bbs-signatures/, 2025.

[74] "Hyperledger AnonCreds specification," https://hyperledger.github.io/anoncreds-spec/, 2023.

[75] M. Chase, C. Ganesh, and P. Mohassel, "Efficient zero-knowledge proof of algebraic and non-algebraic statements with applications to privacy preserving credentials," in *CRYPTO*, 2016.

[76] F. Baldimtsi, K. K. Chalkias, Y. Ji, J. Lindstrøm, D. Maram, B. Riva, A. Roy, M. Sedaghat, and J. Wang, "zklogin: Privacy-preserving blockchain authentication with existing credentials," in *CCS*, 2024.

[77] T. Godden, R. D. Smet, K. Steenhaut, and A. Braeken, "Efficiently parsing existing eID documents for zero-knowledge proofs," in *Cryptology ePrint Archive*, 2025.

[78] N. Desmoulins, A. Dumanois, S. Kane, and J. Traoré, "Making BBS anonymous credentials eIDAS 2.0 compliant," in *Cryptology ePrint Archive*, 2025.

Appendix A. Less than operation

Suppose we have two primes p and n with $n < p$. Given $v \in \mathbb{F}_p$, we want to check that $v < N$, i.e., v is in the range 0 to $n-1$. Let k denote the minimum number of bits required to represent n as an unsigned integer, i.e., k is the smallest integer such that $n < 2^k$. Since we are checking that $v < n$, we decompose v into k bits $(v_0, v_1, \ldots, v_{k-1})$ such that $v = \sum_{i=0}^{k-1} v_i \cdot 2^i$. Let n_i denote the ith bit of n (these do not need to be allocated as variables, but are used as constants in the constraint system). We want a constraint system that is satisfiable if and only if $0 \leq v < n$. We do this as follows.

- Initialize $eq_k = 1$ and iteratively compute $eq_i = eq_{i+1} \wedge (v_i = n_i)$ for $i = k-1, \ldots, 0$.
- Initialize $lt_k = 0$ and then iteratively compute $lt_i = lt_{i+1} \vee (eq_{i+1} \wedge n_i \wedge \not{v}_i)$ for $i = k-1, \ldots, 0$.

Correctness intuition. Observe that while the "high" bits match, eq is set to 1. And, it becomes 0 when the bits stop matching. Additionally, lt starts with 0 and becomes 1 at the first bit where n has a 1 and v has a 0 (denoting v is strictly smaller than n). Once lt is set, it continues to be set because of the or operation. Finally, lt_0 holds the output of

```
{ "doc_type": "org.iso.18013.5.1.mDL",
  "issuer_signed": {
    "namespaces": {
      "org.iso.18013.5.1": [
        { "digest_id": 1, "random": "...",
          "element_identifier": "birth_date",
          "element_value": "1980-06-15" }
      ]
    },
    "issuer_auth": {
      "protected": { "alg": "ES256" },
      "payload_mso": {
        "version": "1.0",
        "digest_algorithm": "SHA256",
        "value_digests": {
          "org.iso.18013.5.1": {
            "1": "<digest>" }
        },
        "device_key_info": {
          "device_key": "<P-256 public key>"
        },
        "validity_info": {
          "valid_from": "2025-08-17T12:34:56Z",
          "valid_until": "2035-08-17T12:34:56Z"
        }
      },
      "signature": "<issuer signature>"
    }
  },
  "device_signed": {
    "device_auth": {
      "protected": { "alg": "ES256" },
      "payload_device_namespaces": {},
      "signature": "<device signature>"
    }
  }
}
```


Figure 12. Minimal JSON mDL credential (mDoc).

the less than operation. It is set to 1 if the less than condition holds, and 0 otherwise.

Optimizations. In our setting, n is a constant, so we specialize the eq and lt updates per bit of n: when $n_i = 0$, only eq needs updating (1 constraint); when $n_i = 1$, both eq and lt are updated (3 constraints). In expectation, the gadget costs about 2 constraints per bit.

Appendix B. Example mDL credential

Figure 12 illustrates a minimal JSON serialization of an mDoc. The top-level object separates two signed blocks. The `issuer_signed` block carries (i) a `namespaces` array containing salted digests of each attribute, and (ii) the `issuer_auth` object, whose field `payload_mso` contains the MSO itself along with metadata and the client's public key. Each issuer-signed item directly embeds the plaintext attribute value (the pre-image) and its salt whose hash appears in the MSO's `value_digests` table. The complementary `device_signed` block holds the holder's signature proving possession of that key at presentation time.

Appendix C. Meta-Review

The following meta-review was prepared by the program committee for the 2026 IEEE Symposium on Security and Privacy (S&P) as part of the review process as detailed in the call for papers.

C.1. Summary

Vega is a zero-knowledge proof system designed to prove statements about existing credentials, such as mobile driver's licenses, without requiring modifications to issuer infrastructure. The system achieves high performance on resource-constrained devices by splitting proving into precomputation and online phases.

C.2. Scientific Contributions

- Provides a Valuable Step Forward in an Established Field.

C.3. Reasons for Acceptance

1) The paper shows how to split the proving and still ensure soundness and zero-knowledge using (randomizable) commitments.
2) Vega demonstrates substantial speedups in verification and reductions in proof size compared to recent state-of-the-art systems like Longfellow.