Crescent: Stronger Privacy for Existing Credentials

Christian Paquin, Guru-Vamsi Policharla and Greg Zaverucha

DRAFT, December 13, 2024

Abstract

We describe Crescent, a construction and implementation of privacy-preserving credentials. The system works by upgrading the privacy features of existing credentials, such as JSON Web Tokens (JWTs) and Mobile Driver's License (mDL) and as such does not require a new party to issue credentials. By using zero-knowledge proofs of possession of these credentials, we can add privacy features such as selective disclosure and unlinkability, without help from credential issuers. The system has practical performance, offering fast proof generation and verification times (tens of milliseconds) after a once-per-credential setup phase. We give demos for two practical scenarios, proof of employment for benefits eligibility (based on an employer-issued JWT), and online age verification (based on an mDL). We provide an open-source implementation to enable further research and experimentation.

This paper is an early draft describing our work, aiming to include enough material to describe the functionality, and some details of the internals of our new library, available at https://github.com/microsoft/crescent-credentials.

1 Introduction

Digital Identity and Mobile Wallets Long-lived digital credentials for identity are becoming more common, with primary examples being the mobile Driver's License (mDL), Selective Disclosure JSON Web Tokens (SD-JWT) [FYC24], and W3C Verifiable Credentials (VCs). The mDL standard is already used by eleven US states to issue residents a digital version of their driver's license, with 14 other states working to implement it. Mobile wallets are also increasingly expanding beyond payments to include credentials relating to employment (e.g., Microsoft Entra Verified ID¹ and LinkedIn's Place of Work Verification²) and healthcare (e.g. Smart Health Cards ³ for proof of vaccination). The European digital identity (eID) regulation aims to provide identification and authentication by means of a personal digital wallet on a mobile phone [Eur]. This is exciting, since it will enable better authentication in new online scenarios, but we are also concerned with the privacy implications of these new technologies.

The mDL and SD-JWT did introduce some limited privacy protections. Basically the credential is a signed list of randomized commitments to the attributes, that the user can selectively open when presenting the credential. While selective disclosure (SD) is welcome and important, every use of the credential may be correlated (by using the signature as a handle). The problem is compounded when groups of verifiers collude and share information.

VCs define an abstract credential representation that can be realized using various underlying credential types that support SD. In addition to SD-JWT, they can also support schemes such as CL or BBS-based anonymous credentials, that provide both unlinkability and SD. Even once these schemes are standardized (e.g., BBS is going through CFRG standardization [LKWL24]), many extensions and integration profiles will need to be written, resulting in significant changes in today's issuance infrastructure, which will complicate and/or deter adoption of these technologies. It is therefore likely that we'll have entrenched deployments with limited privacy for the foreseeable future.

In the academic literature there is a folklore construction of anonymous credentials from traditional credentials; given a sufficiently general zero-knowledge proof system, the credential holder simply proves

¹https://learn.microsoft.com/en-us/entra/verified-id/

 $^{^{2}} https://learn.microsoft.com/en-us/entra/verified-id/linkedin-employment-verification$

³https://smarthealth.cards/en/

knowledge of a valid credential. More concretely, for a JWT, (which is a list of attributes encoded as JSON with an appended signature), the user would prove that 1) some private base64 data is signed by hashing it, 2) that the signature verification equation holds, 3) the signed data decodes to JSON, 4) the encoded expiry date is shown to be in the future, and 5) the attributes are revealed (or proven to satisfy some predicates).

Perhaps surprisingly, proving knowledge of such complex credentials has been shown to be approaching practicality. With recent advances on the engineering side of SNARKs (driven in part by academic research, but also by large investments in the blockchain space), these approaches are constantly getting more practical. We mention two prominent examples here and discuss related work in more detail in Section 5. There was the Cinderella paper [DFKP16] that created proofs of X.509 certificates (including chain building), and more recently the zk-creds [RWGM23] work builds a new credential system that proves knowledge of signed passport data as part of vetting users during issuance.

The most significant limitation of these works, common to many applications of zk-SNARKs is that the time required to generate a proof is high: tens of seconds. Flexibility is also limited since changing the statement to prove (e.g., selective disclosure with a different set of attributes) requires a change to the circuit. Furthermore, the performance of specific examples does not generalize all that well, since the cost of SNARKs is hard to predict, for instance a change in the issuer's signing algorithm could give wildly different performance. From the Cinderella or zk-creds effort, it therefore not obvious how expensive an mDL proof would be. Without benchmarks for commonly used credentials, proposing use of this idea is blocked since an informed consideration of costs is essential (for any technology).

Crescent: An anonymous credential system built from existing credentials and ZKPs We design and implement an anonymous credential system by creating zero-knowledge proofs (ZKPs) of existing credentials being issued today (as opposed to designing new credentials with privacy features built-in). The goal is to provide a ready-to-use library for multiple scenarios to showcase this approach and provide concrete benchmarks for all steps of the process. Our implementation includes two demo scenarios that illustrate the system end-to-end. Credentials are stored in a wallet, implemented as a browser extension. In the first scenario, an employment credential (in JWT format, that could be a Microsoft Entra or OpenID credential) is used to prove eligibility for employer provided mental health care benefits. Namely, the health care provider learns where the patient works, and nothing more, and the employer does not learn that the employee has accessed this benefit (which is highly personal). In the second scenario, using an mDL credential a user unlinkably proves they are at least 18 years old to create a new social media account, without revealing any other information from the mDL.

Technical Overview We observe that in each proof, almost all of the work is the same (the circuit cost is dominated by the signature verification proof). Thus, generating a single monolithic proof that repeats all work every time we want to present the credential is wasteful. Our design moves this common work to a pre-processing phase that the prover does once, after obtaining the credential. Since these credentials are long-lived, this happens infrequently. This offline work is relatively slow (tens of seconds), but subsequent proofs can be generated much faster (tens of ms), comparable to generating a BBS presentation proof. Proofs are about 1KB in size, often shorter than the input credential. (The size and time is dominated by a range proof to show the credential is not expired; that would also be required with BBS or other conceivable approaches).

Given this high-level design, we leverage mostly well-understood techniques to realize it. For the offline phase, we use the Groth16 [Gro16] proof system to create a proof of signed credential data that decodes and parses to a list of attributes that are then made outputs of the circuit. These outputs can be provided to the verifier in the form of a Pedersen vector commitment (as observed in LegoSNARK [CFQ19]), so they can be disclosed or kept hidden, as required by the scenario. The Groth16 proof and the commitment can be re-randomized, in a way that makes them unlinkable, so they can be computed once, and used repeatedly (as analyzed in [RWGM23]). Finally, we use a small Σ -proof to prove knowledge of the committed values, and a range proof [BFGW20] to show that the credential is not expired (or that age > 18). Since the final step is a proof about committed values, it is reminiscent of traditional anonymous credential systems [CV02, PZ33, LKWL24, CPY⁺, CPZ20], in particular it is very efficient. It's also much easier to change the statement to be proven given a committed input, and we can easily interface with other sub-provers (such as range proofs, or other advanced predicates) in a modular way.

Some additional advantages follow from the fact that the performance of the offline SNARK prover step is not critical. We do not need special techniques to prove knowledge of the credential and can use one SNARK for multiple credential types. We can also afford to express the circuit in a more readable, but less efficient way (we use the Circom [ide] high-level language). The circuit may be common to all issuers that use the same signing algorithm, since it must only parse out the attributes. One limitation is the need for a trusted set of parameters; it's important for verifiers to believe these were created honestly, requiring trust assumptions or distributed generation protocols. Fortunately, these have been well studied and deployed in the context of blockchains (see [NRBB24] for a recent survey).

Our implementation is a Rust library, together with Circom code expressing the circuits required for the offline phase. All code is available at https://github.com/microsoft/crescent-credentials. The main dependency is the arkworks library [ac22], it provides a Groth16 implementation and the primitives we need to implement the range proof and Σ -proof. In terms of performance, for a 2KB-long JWT signed with RSA-2048 and SHA-256, the offline phase (witness generation and proof generation done once per credential during their setup) takes 27s, showing a credential takes 22ms and verification takes 15ms. The proof is 987 bytes and the circuit has 1.5M constraints.⁴

To conclude, our work provides software demonstrating that strong privacy with existing credentials is practical. This will enable further experimentation, prototyping and research, a necessary step towards deployment of this technology.

2 Preliminaries

2.1 Overview of the Groth16 proof system

In this section we briefly review the [Gro16] proof system, in sufficient detail to explain how we use it to construct efficient credentials.

R1CS The circuits used in our construction will be expressed using a high-level language (Circom [ide]), which then compiles them to a rank-1 constraint system (R1CS) instance. The extended witness (often simply called the witness) is the input and output to the circuit, along with all intermediate wire values. An R1CS instance is a set of quadratic constraints on the witness values, such that a satisfying witness contains a valid (input, output) pair for the circuit. The circuit is an arithmetic circuit defined over a field \mathbb{F} , and all witness values, circuit inputs and outputs are elements of \mathbb{F} . Looking ahead, \mathbb{F} will be the scalar field of a pairing-friendly curve, such at BN254 or BLS12-381.

More concretely, let the R1CS system have n variables, out of which p are public and we specify m constraints on them. The witness is defined as a vector $w \in \mathbb{F}^n$. In most codebases, the convention is to set $w_0 = 1$, to rule out the *trivial* solution of all zeros, and the next p entries are the public input. The constraint system is specified by three matrices A, B and C of size $n \times m$. A witness w satisfies the constraint system if $(w \cdot A) \circ (w \cdot B) = w \cdot C$, where \cdot denotes standard matrix/vector multiplication and \circ denotes the Hadamard (element-wise) product.

QAPs A quadratic arithmetic program (QAP) is another way to represent an arithmetic circuit [GGPR13], and is the representation used by the Groth16 proof system. Therefore, the R1CS instance is first translated into a QAP. To interpret the R1CS constraints as a QAP:

- View the *i*-th row of each of the matrices A, B, C as the evaluations of polynomials $A_i(X), B_i(X), C_i(X)$ on a fixed domain $\Omega = \{x_1, \ldots, x_m\} \subset \mathbb{F}$ of size m, then compute A_i, B_i and C_i with polynomial interpolation. In practice Ω is chosen to be the roots of unity in order to use fast polynomial interpolation algorithms.
- Define polynomials $A(X) = \sum_{i=1}^{m} w_i A_i(X)$, and similarly for B(X) and C(X). Note that if $A(x_i)B(x_i) = C(x_i)$, then the *i*-th constraint of the R1CS instance is satisfied.

⁴Measured on a laptop with a 4-core Intel Core i7-10610U CPU @ 1.80GHz.

• From the above observation, it suffices to prove that the polynomial A(X)B(X) - C(X) evaluates to 0 at all points in Ω . Let $Z_{\Omega}(X)$ be the vanishing polynomial on Ω : $\mathbb{Z}_{\Omega}(X) = \prod_{i=1}^{m} (X - x_i)$. It is now sufficient to show that $A(X)B(X) - C(X) = Z_{\Omega}(X) \cdot H(X)$ for some "low-degree" polynomial H(X).

2.1.1 The Groth16 Proof System

The proof system uses an elliptic curve with a pairing which we denote $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. Point G_1 is a generator of group \mathbb{G}_1 and G_2 generates \mathbb{G}_2 . The groups have order r and the scalar field is \mathbb{F}_r . Below we use the notation $(a_1, \ldots, a_n) \cdot G$ as shorthand for (a_1G, \ldots, a_nG) .

We describe a variant of Groth's proof system that includes a simplification from [KMSV21] that preserves security, and allows proofs to be re-randomized [BKSV21]. This re-randomization technique (also used in zk-creds [RWGM23]) is very efficient, and allows us to quickly generate presentation proofs of a credential.

Setup This function must be run once, and can be re-used for circuits with R1CS instances having m constraints or fewer. Sample random $\tau, \alpha, \beta \leftarrow \mathbb{F}$ and output the public parameters:

$$(1, \tau, \tau^2, \dots, \tau^{2m-2}) \cdot G_1$$

$$(1, \tau, \tau^2, \dots, \tau^{m-1}) \cdot G_2$$

$$\alpha \cdot (1, \tau, \tau^2, \dots, \tau^{m-1}) \cdot G_1$$

$$\beta \cdot (1, \tau, \tau^2, \dots, \tau^{m-1}) \cdot G_1$$

$$\beta \cdot G_2$$

Circuit-Specific Setup. Once Setup has been run, a circuit-specific setup step is required for each circuit that will be used in proof generation and verification. Sample random $\delta \leftarrow \mathbb{F}$. Define polynomials L_i for $i \in [m]$ as:

$$L_i(X) = \beta \cdot A_i(X) + \alpha \cdot B_i(X) + C_i(X).$$

Next compute the prover key (required by proof generation), and the verifier key (required for proof verification). Note that the verifier key is much shorter than the prover key (recall that p is the number of public outputs).⁵

The prover key and verifier key are public outputs.

Proof Generation Let $w = (1, w_1, \ldots, w_m)$ be a witness that satisfies the R1CS instance specified by the matrices (A, B, C). Compute A(X), B(X), C(X), and H(X) as specified in Section 2.1 and $L(X) = \sum_{i=n}^{m-1} w_i \cdot L_i(X)$. Now, sample random $r, s \leftarrow \mathbb{F}$ and compute the proof:

$$\begin{split} A &= (\alpha + r \cdot \delta + A(\tau)) \cdot G_1 \\ B &= (\beta + s \cdot \delta + B(\tau)) \cdot G_2 \\ C &= (\delta^{-1}(L(\tau) + H(\tau) \cdot Z_{\omega}(\tau)) + s \cdot (\alpha + r \cdot \delta + A(\tau)) + r \cdot (\beta + s \cdot \delta + B(\tau)) - r \cdot s \cdot \delta) \cdot G_1 \end{split}$$

Verification Given a proof (A, B, C) and public input $(w_0 = 1, w_1, \ldots, w_p)$, the verifier checks that:

$$e(A,B) = e(\alpha \cdot G_1, \beta \cdot G_2) + e(\overline{L}, G_2) + e(C, \delta \cdot G_2), \tag{1}$$

where $\bar{L} = \sum_{i=0}^{p} w_i \cdot (L_i(\tau) \cdot G_1).$

⁵The terminology *prover key* and *verifier key* are common in the SNARK literature, but note that these values can be viewed as circuit-specific public parameters since they are public and shared by all provers and verifiers for a circuit.

3 Anonymous Credentials from Existing Credentials

The Crescent construction is a type of anonymous credential system. In this section we describe anonymous credentials and the Crescent construction.

3.1 Overview of the Crescent Credential System

Existing Credentials By *existing credentials*, we are broadly referring to the class of digital credentials that are a list of attributes, signed with a public key signature. Concrete examples are given in the introduction, the Crescent implementation currently supports JWT and mDL credentials but X.509 certificates and machine readable passports are also in this class (see [RWGM23]). In Appendix A we give an example JWT for reference.

Parties An anonymous credential system has three types of parties.

- **Issuer:** The **Issuer** (or *identity provider*) issues existing credentials to **Users** with a process that is out-of-scope for Crescent. The **Issuer** does not need to accommodate Crescent, nor even be aware of it.
- User: The User (or *credential holder*) has a credential from the issuer that they will show using Crescent to one or more verifiers. The user generates proofs to send to verifiers, that are referred to as *show proofs* or *presentation proofs*.
- Verifier: The Verifier (or *relying party*) receives proofs from one or more users, and if the proof is valid, provides access to a resource or service.

Protocols We informally describe the functionality as the six algorithms below along with some high-level comments describing Crescent. The existing credential system is assumed to have its own protocols, that are defined and run outside of Crescent; we denote them by (Setup, KeyGen, Issue, Verify). Below we focus on describing how these interact with Crescent. For simplicity we assume a single existing credential type, and a fixed set of predicates that will be used when presenting the credential.

Setup After Setup for the existing credential scheme has been run, the Setup algorithm for the anonymous credential system is run. Crescent requires that the Groth16 public parameters are generated, and the circuit-specific setup is run. The circuit depends on the schema of the credentials (when it is flexible), and the cryptographic algorithms used by the issuer.

KeyGen, Issue The key generation and issuance protocols are defined by the existing credential scheme. The Issuer runs KeyGen, makes their public key available to all parties, and can then issue credentials to users.

Prepare The User runs Prepare once per issued credential. Prepare can be run anytime after Issue and before the first call to Show. Prepare creates *user state*, preprocessed information that is later used to Show the credential. The input to Prepare is a credential, the Issuer's public key, the public and circuit-specific parameters, the prover and verifier keys. In Crescent Prepare creates a Groth16 proof, and parses out attributes from the credential. We note that the user state does not include the large prover key from Groth16 circuit-specific setup.

Show To create a show proof for a credential, Show needs the *user state* created by Prepare, and the predicate requested by the verifier. Example predicates are "age > 18" or "domain(email_address) = example.com". Certain validation checks from the existing credential may be proven during Show rather than in the circuit – a common example is the expiry check; it cannot be done in the circuit since it depends on the current time. The Crescent Show proof is a re-randomized instance of the Groth16 proof, and a small number of Σ -proofs about the committed attribute data. Show proofs can therefore be computed in a modular way, depending on the predicates requested by the verifier.

Verify The input to verification is the show proof, the issuer's public key and any additional data required for predicates. If the proof is valid the Verifier accepts the predicate(s) as being satisfied by the credential (and any revealed data as authentic).

3.2 Detailed Description of Crescent

Circuits The circuit required for the proof generation step of **Prepare** must basically run the Verify algorithm of the existing credential scheme, as a normal verifier would. For JWTs this means:

- 1. Splitting the token into the header, payload and signature parts,
- 2. Base64-decoding these three values
- 3. Verifying the signature:
 - (a) hashing the payload and header to compute the digest, and then
 - (b) running the signature verification function (RSA or ECDSA) with the digest and issuer public key as inputs.
- 4. Parsing the JSON payload data to locate the attributes of interest, optionally applying a function to each attribute, then outputting them. The function could apply a collision-resistant hash to long attributes, encoding strings as \mathbb{F} elements, or create derived attributes from existing ones (e.g., the email attribute could be divided into two parts: username and domain, so that the domain only can be revealed).

This circuit is non-trivial and the cost in R1CS constraints is high. For the example JWT given in Appendix A, which is about 2KB in length and signed with the RS256 algorithm (RSA PKCS#1v1.5 with SHA-256), the circuit has 1583 999 constraints. This breaks down roughly as 1M for SHA-256 hashing, 250k for RSA verification, and 250k for everything else.

For mDL credentials the circuit has a similar high-level structure, but the attribute data is encoded with CBOR rather than JSON, and randomized hashes of the attributes are in the payload, rather than the values directly. The example mDL credential in Crescent is signed with ES256 (ECDSA on curve P256 with SHA-256), the signed data is 1152 bytes and the circuit has 2635013 constraints. Here the dominant cost is ECDSA verification (\approx 2M constraints), which requires two non-native scalar multiplications (i.e., the field operations required for scalar multiplication on the P256 curve are emulated in an arithmetic circuit on the BN254 curve used by our Groth16 implementation).

3.3 Generating Show Proofs

Here we describe how Crescent uses one (expensive to generate) Groth16 proof for an unlimited number of efficient credential Show proofs, using an idea from [RWGM23].

The circuit outputs the parsed and formatted attribute values. The Show proof assigns to each attribute value one of the following types (chosen by the User).

Hidden Nothing is proven about the attribute during Show (though something may have been proven in the circuit). Let \mathcal{H} be the set of attribute indices having type Hidden.

Committed The attribute is provided to the verifier in a separate commitment (to only this attribute), so that a sub-prover may be run on this attribute. This is a Pedersen commitment, so it is friendly to efficient Σ -proofs. Let \mathcal{C} be the set of attribute indices having type Committed.

Revealed The attribute is revealed to the verifier. Let \mathcal{R} be the set of attribute indices having type Revealed.

Show proceeds as follows.

- 1. Let (A, B, C) be the Groth16 proof generated as described in Section 2.1 during Prepare and stored in the user state. Let the circuit public output values be (att_1, \ldots, att_p) .
- 2. Re-randomize the Groth16 proof. Sample $\rho, \omega, z \leftarrow \mathbb{F}$ and compute

$$\pi_A \leftarrow \rho^{-1} \cdot A$$
$$\pi_B \leftarrow \rho \cdot B + \rho \cdot \omega \cdot (\delta \cdot G_2)$$

$$\begin{split} \pi_C &\leftarrow C + \omega \cdot A - z \cdot G_1. \\ \pi_{\bar{L}} &\leftarrow \bar{L} + z \cdot (\delta \cdot G_1) \end{split}$$

It can be checked that $(\pi_A, \pi_B, \pi_C, \pi_{\bar{L}})$ satisfy the Groth16 verification equation (Eq. (1)).

- 3. For each attribute att_i , if $i \in \mathcal{C}$, sample a fresh random value $z_i \leftarrow \mathbb{F}$ and compute a Pedersen commitment $\operatorname{com}_i = \operatorname{att}_i \cdot (L_i(\tau)G_1) + z_i \cdot (\delta G_1)$. The prover then updates $\pi'_C \leftarrow \pi_C \sum_{i \in \mathcal{C}} z_i \cdot G_1$.
- 4. Generate a proof of knowledge of representation of $\pi_{\bar{L}}$ in the $(\{L_i(\tau) \cdot G_1\}_{i \in [p]}, \delta \cdot G_1)$ basis, where the scalars $\{\mathsf{att}_i\}_{i \in \mathcal{R}}$ are revealed in the clear. The same proof also proves knowledge of the commitment openings, and in particular that com_i is a commitment to att_i . Denote this proof π_{Σ} , a proof of the relation

$$\left\{ (\{\mathsf{att}_i\}_{i\in\mathcal{H}},\{(\mathsf{att}_i,z_i)\}_{i\in\mathcal{C}}): \pi_{\bar{L}} = z(\delta G_1) + \sum_{i=1}^p \mathsf{att}_i(L_i(\tau)G_1) \ \bigwedge \ \mathsf{com}_i = \mathsf{att}_i \cdot (L_i(\tau)G_1) + z_i \cdot (\delta G_1) \right\}$$

Instantiating (non-interactive) proofs of knowledge for this type of relation is well-known (see, e.g., [BS23, §20.2]).

5. Output $(\pi_A, \pi_B, \pi_C, \pi_{\bar{L}}, \{\mathsf{att}_i\}_{i \in \mathcal{R}}, \{\mathsf{com}_i\}_{i \in \mathcal{C}}, \pi_{\Sigma})$

Verify Aggregate the commitments into π_L , by computing $\pi'_{\bar{L}} \leftarrow \pi_{\bar{L}} + \sum_{i \in \mathcal{C}} \operatorname{com}_i$. Then verify $(\pi_A, \pi_B, \pi_C, \pi'_{\bar{L}})$ with Eq. (1). Finally, verify π_{Σ} . If all checks pass, accept the Show proof as valid.

Sub-provers for Show Using the hidden and revealed attribute types we can implement *selective disclosure* where the prover reveals a subset of the attributes to the verifier. With the committed type and an additional *sub-prover* we can implement more advanced predicates. The example implemented in Crescent is a module for range proofs, that we use to prove that JWTs and mDLs are not expired, and that the mDL birth date encodes an age greater than a given constant.

While there is some overhead to creating fresh commitments for sub-provers and proving they are consistent with the Groth16 vector commitment, we decided that the modularity this implementation provides is worth the cost. The alternative is to run all sub-provers in a monolithic proof. The consistency proof is an equality of discrete logarithms proof (DLEQ proof) that has a well-known and simple Σ -proof. Another natural extension enabled by committed attributes is the ability to link credentials together, for example, it is easy to prove that the name field of an employment JWT matches the name of the mDL. Simply create Show proofs for both credentials and make the name field committed. Then use a DLEQ proof to show the committed names are equal. This feature was also used in [RWGM23].

The DLEQ proof is also a natural place to bind context, session or nonce data to the proof, e.g., to ensure a proof is not being replayed, a verifier may send a nonce to a prover. Since we use the Fiat-Shamir [FS87] transform to make this proof non-interactive, we can efficiently bind data to the proof by hashing it when generating the challenge. Proofs that authenticate data in this way are sometimes called *signatures of knowledge* [CS97, CL06, GM17], and Crescent can also be used to create signatures, without the need for signers to manage a long-term signing key (which has proven difficult in practice). For example, a developer could (pseudonymously) sign code or commit messages by proving knowledge of an existing credential.⁶

We have also prototyped a method to efficiently form committed attributes with a ZK-friendly hash function to more efficiently use (zk-)SNARKs as sub-provers, however decided to hold this (rather complex) feature back, until a compelling scenario demands it.

 $^{^{6}}$ Care must be taken here not to use a credential that is frequently presented to many verifiers, since a malicious verifier could use the credential to generate a signature. We offer two potential mitigations: (i) use a set of credentials or (ii) during credential issuance, if the user is allowed to provide a nonce value that is included in the token, it can be made the hash of a secret value, and the circuit can check that the prover knows this value.

Circuits and multiple issuers Since the per-circuit parameters are large, and must be generated by a trusted party or with a distributed protocol, we would like to have as few parameter sets possible. With our current implementation, when issuers share the same schema and the same cryptographic signing algorithm, the same circuit may be used. More precisely, if the set of attributes that the circuit must output are present in all tokens, then same circuit will work (since it will ignore other attributes). We expect the mDL issuers to be consistent, at least at the country level (e.g., AAMVA has a profile [oMVA24] of the ISO mDL standard for the United States), but more investigation is required. We note that it is possible to create a more general circuit, that takes a list of attribute names as input, to allow sharing between issuers with different schemas but using the same signature algorithm.

3.4 Security

Intuitively the security properties we want from this system are the following.

Correctness ensures that for all valid credentials and sets of predicates, the User can generate a proof that will be accepted by the Verifier. This depends on the correctness of both the Groth16 and Σ -proofs, as well as the correctness of the circuit.

Unforgeability ensures that a malicious user cannot produce an accepting proof for a set of predicates, unless they have a valid credential satisfying those predicates. Again this reduces to soundness of the underlying proof systems that make up Crescent and correctness of the circuit. Additionally, it requires that the existing credentials be unforgeable.

Anonymity/Unlinkability ensures that proofs reveal only what is revealed by the predicates that are proven. This strong privacy ensures that users who do not reveal identifying information can remain anonymous. This implies unlinkability, since a verifier that receives two Show proofs cannot distinguish whether they were created by the same or different Users (provided the predicates and revealed data are the same). This property reduces to the zero-knowledge property of the underlying proof systems. As mentioned, the Groth16 system requires a circuit-specific setup to be generated with a ceremony, or by a trusted party. We can show that even with knowledge of the trapdoor (τ) , it is not possible to link a User across different Show proofs.

We add the disclaimer that we not yet done a formal analysis of Crescent as a whole, and the above properties are not proven, just what we expect will hold. The primitives we rely on are well analyzed. There are many existing analyses of Groth16 and variants of it [Gro16, BKSV21, CFQ19, GM17, RWGM23], the Σ -proofs are also straightforward to analyze [BS23] and the range proof we use is built on well-understood polynomial commitments [BFGW20, KZG10, MBKM19].

Quantum attacks NIST recently standardized new cryptographic algorithms that are expected to resist attacks using quantum computers. The initial goal of the transition to these post-quantum algorithms is to reduce the risk of a "record-now decrypt-later" attack, where an attacker records ciphertext today, protected with current algorithms, and potentially decrypts it in the future if they obtain a sufficiently capable quantum computer. Signature and authentication algorithms are also being migrated in some limited settings (e.g., for code signing and software update pipelines), but generally with a lower priority for online authentication scenarios, as there are no retroactive attacks like with confidentiality.

Therefore, many standards used in the identity space (such as JWT, and also new standards like mDL) are not currently using post-quantum signature algorithms. The credentials are relatively short-lived (2-5 years for mDL) and can be updated when these systems migrate to post-quantum cryptography.

The Groth16 proof system and the other proof techniques we use in Crescent have perfect zero-knowledge, which means that an attacker recording Show proofs today will not be able to learn additional user data given a quantum computer (or even a hypothetical machine much more capable than a quantum computer). Thus the privacy and unlinkability guarantees Crescent provides can be considered post-quantum secure.

What an attacker can do with a quantum computer is break soundness of the proof system, meaning they can create Show proofs, without having a valid credential, that a verifier will accept. This is because the soundness (or unforgeability) guarantees of Crescent depend on computation assumptions that a quantum computer can break. However, breaking these assumptions also means that the **Issuer**'s signature algorithms

are also broken, and we have to update the existing credential standards as well as the software of all parties (at which point we could also update the proof system).

Finally, currently known zero-knowledge proofs with post-quantum zero-knowledge and soundness that apply to the credential scenarios we are considering are much newer and less well studied than Groth16 and Σ -proofs. These new constructions often rely on new assumptions (or new parameter sets, e.g., with lattices), and they come with the main performance drawback of having larger proof sizes (hundreds of kilobytes or even megabytes).

4 Benchmarks

We give some preliminary benchmarks showing the CPU time required for Crescent operations, as well as the sizes of Crescent proofs and parameters. The code we benchmark is available at https://github.com/microsoft/crescent-credentials. In the JWT scenario, we create a proof that discloses the domain name of a email attribute. A test token is given in Appendix A, which is about 2KB in length and signed with the RS256 algorithm (RSA PKCS#1v1.5 with SHA-256), the circuit has 1583 999 constraints. In the mDL scenario, we prove that the holder is older than 18 years old. The credential is signed with ES256 (ECDSA on curve P256 with SHA-256), the signed data is 1152 bytes and the circuit has 2635013 constraints. Our implementation currently supports the BN254 elliptic curve (the curve used by Ethereum for smart contracts).

Our test machine is an Intel Xeon W-2133 CPU @ 3.60GHz (launched 2017), with 6 cores. The arkworks library we depend on for cryptographic computations uses multiple cores, primarily to accelerate multi-scalar multiplication operations.

The results are given in Table 1. Both scenarios have a similar performance profile and cost tradeoffs. The time for creating and verifying Show proofs is low, comparable to purpose-built anonymous credential systems like BBS. The size of proofs is also small – note that in the JWT case, the proof is about half the size of the token, so the required bandwidth is actually smaller when showing a token with a ZKP. In the mDL case, our proof is larger (but still comparable in size to using the mDL without a ZKP), since we require two range proofs (each is 636 bytes): one to prove the credential's valid until date is in the future, and one for the age predicate. We have not yet investigated batching these range proofs to reduce costs. The user state and parameters required for Show and Verify are also small.

The drawbacks are typical of applications using the Groth16 (or similar) proof system. The time for Prepare, where we generate the one-time Groth16 proof for a credential, is tens of seconds. We note that the Circom-generated WASM code for witness generation in the mDL case is highly inefficient, we suspect due to some large lookup tables present in the ECDSA circuit we are currently using. We expect that this will be in line with the JWT times (Prepare taking ≈ 35 s) after some optimization effort. The other drawback is the large size of the parameters required for the Prepare step (594 MB for JWT and 1.1 GB for mDL). Fortunately these are only required during Prepare, which is infrequent, and the parameters required during the more common operations are small.

Our range prover is an implementation of the proof described in [BFGW20]. For the 32-bit range we need, the size of the proof is 636 bytes, proofs take 9.8 ms to generate and 6.3 ms to verify. The size of the prover key is 17 KB and the verifier key is 640 bytes. As it is based on polynomial commitments, it also requires a trusted setup and a paring-friendly group. In the context of our credential system, these could be generated at the same time as the Groth16 parameters. Our implementation currently uses the same EC implementation from arkworks (the BN254 curve).

5 Additional discussion of related work

In zk-creds [RWGM23], the exiting credential issuer (e.g., the passport issuer) is oblivious to the zk-creds system and does not need to modify how they issue credentials. However, the zk-creds system adds a new functionality, a type of issuer, that maintains a list of credentials that have been issued, in the form of a membership list (encoded as a Merkle tree with a public root and idealized as a "bulletin board instantiated as a transparency log, Byzantine system, or even a blockchain"). A credential is added to the membership list, after a user presents zk-supporting documentation (which could take the form of a ZKP of a valid

	JWT	mDL
Prepare	19.2 s	149.7s
Witness gen.	$6.9 \mathrm{~s}$	$128.8s^*$
Groth16 prove	$12.3 \mathrm{~s}$	$20.9 \mathrm{~s}$
Show	28.5 ms	42.7 ms
Verify	11.7 ms	18.5 ms
User state	39 KB	39 KB
Show proof	$987 \mathrm{B}$	1727 B
Prepare params	594 MB	1.1 GB
Show params	$56~\mathrm{KB}$	$56~\mathrm{KB}$
Verifier params	40 KB	40 KB

Table 1: Preliminary benchmark results (December 2024). Timings are given in the top half of the table and sizes in the bottom half.

passport). A commitment to the zk-supporting documentation is also logged in the membership list. The system only uses the existing credential for vetting during the issuance phase, then "we need only trust the issuer to add credentials (and their supporting documentation) to the list". Once this new credential is issued, presenting it is much faster (150ms) since it requires only proving knowledge of inclusion in a Merkle tree. As partial justification for this new design, the paper argues that the folklore approach of Cinderella and Crescent is insufficient (Section 7.1), with the two sentences: "A proof over a certificate or indeed any existing credential is insufficient: it may not include the necessary data and that data cannot be hidden from the issuer. This is a challenge for, e.g., the random seed we use for cloning resistance. As a result, we must issue our own standalone credentials and can, at best, use existing certificates as supporting documentation to justify issuance.". We argue that working around this limitation (if needed), and avoiding new issuance infrastructure makes Crescent easier to understand, but more importantly easier to deploy. First, the need for extra data is only required for some features, and these can likely be obtained with another approach. (e.g., the random seed can be the hash of two existing credentials, issued by different issuers). Second, with some types of existing credentials (such as OpenID), the user can include a nonce which can encode arbitrary data hidden from the issuer, as in zkLogin, which they use to bind a signing key to an OpenID identity.

The $zkLogin[BCJ^+24]$ system also uses existing credentials, but to authenticate blockchain transactions. They focus on using JWTs issued for OpenID authentication. In zkLogin prover time was also considered to be prohibitively high for frequent authentication, and the paper addresses the issue by offloading proof generation to a trusted party with ample resources, and by binding a signing key to the OpenID token (by re-purposing the nonce field in the issuance protocol). The signing key is then used as a pseudonym. Crescent aims to work independent of the blockchain application, with multiple existing credential types, and does not need to reduce anonymity for performance.

6 Acknowledgments

We thank Larry Joy for help with the implementation of the demos included with the Crescent library. Thanks also to Tianyi Liu, who contributed to some early prototyping work. Finally, we thank Srinath Setty, Weidong Cui and Sebastian Angel for helpful discussions on this topic.

References

- [ac22] arkworks contributors. arkworks zkSNARK ecosystem, 2022. https://arkworks.rs. Presented at RWC 2022.
- [BCJ⁺24] Foteini Baldimtsi, Konstantinos Kryptos Chalkias, Yan Ji, Jonas Lindstrøm, Deepak Maram, Ben Riva, Arnab Roy, Mahdi Sedaghat, and Joy Wang. zklogin: Privacy-preserving blockchain authentication with existing credentials, 2024.

- [BFGW20] Dan Boneh, Ben Fisch, Ariel Gabizon, and Zac Williamson. A simple range proof from polynomial commitments, 2020.
- [BKSV21] Karim Baghery, Markulf Kohlweiss, Janno Siim, and Mikhail Volkhov. Another look at extraction and randomization of Groth's zk-SNARK. In Financial Cryptography and Data Security: 25th International Conference, FC 2021, Virtual Event, March 1–5, 2021, Revised Selected Papers, Part I 25, pages 457–475. Springer, 2021.
- [BS23] Dan Boneh and Victor Shoup. A graduate course in applied cryptography, 2023. Available online https://toc.cryptobook.us/.
- [CFQ19] Matteo Campanelli, Dario Fiore, and Anaïs Querol. LegoSNARK: Modular design and composition of succinct zero-knowledge proofs. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, ACM CCS 2019, pages 2075–2092. ACM Press, November 2019.
- [CL06] Melissa Chase and Anna Lysyanskaya. On signatures of knowledge. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 78–96. Springer, Berlin, Heidelberg, August 2006.
- [CPY⁺] Stephen Curran, Artur Philipp, Hakan Yildiz, Sam Curren and Victor Martinez Jurado, Aritra Bhaduri, and Artem Ivanov. AnonCreds specification. https://hyperledger.github.io/anoncredsspec/.
- [CPZ20] Melissa Chase, Trevor Perrin, and Greg Zaverucha. The Signal private group system and anonymous credentials supporting efficient verifiable encryption. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, ACM CCS 2020, pages 1445–1459. ACM Press, November 2020.
- [CS97] Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups (extended abstract). In Burton S. Kaliski, Jr., editor, CRYPTO'97, volume 1294 of LNCS, pages 410–424. Springer, Berlin, Heidelberg, August 1997.
- [CV02] Jan Camenisch and Els Van Herreweghen. Design and implementation of the idemix anonymous credential system. In Vijayalakshmi Atluri, editor, ACM CCS 2002, pages 21–30. ACM Press, November 2002.
- [DFKP16] Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, and Bryan Parno. Cinderella: Turning shabby X.509 certificates into elegant anonymous credentials with the magic of verifiable computation. In 2016 IEEE Symposium on Security and Privacy, pages 235–254. IEEE Computer Society Press, May 2016.
- [Eur] European Council. European digital identity (eID): Council makes headway towards EU digital wallet, a paradigm shift for digital identity in europe. https: //www.consilium.europa.eu/en/press/press-releases/2022/12/06/european-digital-identity-eidcouncil-adopts-its-position-on-a-new-regulation-for-a-digital-wallet-at-eu-level/.
- [FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, CRYPTO'86, volume 263 of LNCS, pages 186–194. Springer, Berlin, Heidelberg, August 1987.
- [FYC24] Daniel Fett, Kristina Yasuda, and Brian Campbell. Selective Disclosure for JWTs (SD-JWT). Internet-Draft draft-ietf-oauth-selective-disclosure-jwt-14, Internet Engineering Task Force, November 2024. Work in Progress.
- [GGPR13] Rosario Gennaro, Craig Gentry, Bryan Parno, and Mariana Raykova. Quadratic span programs and succinct NIZKs without PCPs. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 626–645. Springer, Berlin, Heidelberg, May 2013.

- [GM17] Jens Groth and Mary Maller. Snarky signatures: Minimal signatures of knowledge from simulation-extractable SNARKs. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 581–612. Springer, Cham, August 2017.
- [Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In Marc Fischlin and Jean-Sébastien Coron, editors, EUROCRYPT 2016, Part II, volume 9666 of LNCS, pages 305–326. Springer, Berlin, Heidelberg, May 2016.
- [ide] iden3. Circom: Efficient circuit framework for programmable zero-knowledge. Webpage: https://iden3.io/circom, Source: https://github.com/iden3/circom.
- [KMSV21] Markulf Kohlweiss, Mary Maller, Janno Siim, and Mikhail Volkhov. Snarky ceremonies. In Mehdi Tibouchi and Huaxiong Wang, editors, ASIACRYPT 2021, Part III, volume 13092 of LNCS, pages 98–127. Springer, Cham, December 2021.
- [KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In Masayuki Abe, editor, ASIACRYPT 2010, volume 6477 of LNCS, pages 177–194. Springer, Berlin, Heidelberg, December 2010.
- [LKWL24] Tobias Looker, Vasilis Kalos, Andrew Whitehead, and Mike Lodder. The BBS Signature Scheme. Internet-Draft draft-irtf-cfrg-bbs-signatures-07, Internet Engineering Task Force, September 2024. Work in Progress https://datatracker.ietf.org/doc/draft-irtf-cfrg-bbs-signatures/07/.
- [MBKM19] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-knowledge SNARKs from linear-size universal and updatable structured reference strings. In Lorenzo Cavallaro, Johannes Kinder, XiaoFeng Wang, and Jonathan Katz, editors, ACM CCS 2019, pages 2111–2128. ACM Press, November 2019.
- [NRBB24] Valeria Nikolaenko, Sam Ragsdale, Joseph Bonneau, and Dan Boneh. Powers-of-tau to the people: Decentralizing setup ceremonies. In Christina Pöpper and Lejla Batina, editors, ACNS 24International Conference on Applied Cryptography and Network Security, Part III, volume 14585 of LNCS, pages 105–134. Springer, Cham, March 2024.
- [oMVA24] American Association of Motor Vehicle Administrators. Mobile Driver's License Implementation Guidelines, r1.3, Sep 2024. https://www.aamva.org/getmedia/261ed16b-3f5c-4678-a2dbcc3016934234/MobileDLImplementationGuidelines-Version1-3.pdf.
- [PZ33] C. Paquin and G. Zaverucha. U-Prove Cryptographic Specification V1.1 (Revision 5), March 2033. Available online: www.microsoft.com/uprove.
- [RWGM23] Michael Rosenberg, Jacob D. White, Christina Garman, and Ian Miers. zk-creds: Flexible anonymous credentials from zkSNARKs and existing identity infrastructure. In 2023 IEEE Symposium on Security and Privacy, pages 790–808. IEEE Computer Society Press, May 2023.

A Example JSON Web Token

Here is an example JWT with a schema similar to the ones used by Microsoft Entra for corporate authentication. The base64url encoded token decodes to the JSON below.

 $eyJ0eXAi0iJKV1QiLCJhbGci0iJSUzI1NiIsImtpZCI6ImU2cTczbnRpLTQ3ZXhYbXdfeVNfLXJHbS1DRnZ5T3NvZ3RzRU1BcDBRUzgifQ. \\ eyJ1bWFpbCI6ImFsaWN1QGNvbnRvc28uY29tIiwiZmFtaWs5X25hbWU0iJFeGFtcGxlIiwiZZ12ZW5fbmFtZSI6IkFsaWN1IiwibG9naW5 \\ faGludCI6Ik8uYWFhYWFiYmJiYmJiYmJjY2)jZGRkZGRkZGV1ZWV1ZWVmZmZmZnZ2AZ2dZ2dn2dnaGhagWlpaWlqampqapqa \\ tra2tra2xsbGxbGxtbW1tbW1ubm5ubm5ubm5ub9vb29vb3BwcHBwcHExcKFxcnJycnJyc3Nzc3RkZGRkIiwibmFtZSI6IkFsaWN1IEV4Y \\ W1wbGUiLCJvaWQi0iIxMjMONTY30COxMjMOLWFiY2QtMTIzNC1hYmNkZWYxMjQ1NjciLCJvbnByZW1fc21kIjoiUy0xLTItMzQtNTY30Dkw \\ MTTZNC0xMjMONTY30DkwLTEyMzQ1Njc40TAtMTIzNDU2NyIsInByZW2IcnJ1ZF91c2VybmFtZSI6ImFsaWN1QGNvbnRvc28uY29tIiwicmg \\ i0iIwLmFhYWFhYmJiYmJjY2NjZGRkZGV1ZWZmZmYxMjMONWdnZ2cxMjMONV8xMjRfYWFhYS4LLCJzaWqi0iIxMjMONTY30COxMjMOLW \\ joiYwxpY2VAY29udG9zby5jb20iLCJ1dGki0iJBQUFCQkJCY2NjY2RkZGQxMjMONTY3IwidmVyaWZpZWRfcHJpbWFyeV91bWFpbCI6WyJh \\ bG1jZUBjb250b3NvLmNvbSJdLCJ2ZXJpZm12F9pzZWNvbRhcn1fZW1haWwi01siYWxpY2UyQGNvbnRvc28uY29t10sImFjY3qi0jjsImF \\ 12ClfInJ1bH1pbmdwYXJ0eS51eGFtcGx1LmNvbSIsImF1dGhfdG1tZSI6MfcZmZk2NTU20CwiZXhwIjoxNZM2NTY4LCJpYXQi0jESMz \\ kNgROFvVFEiLCJuYmYi0jESMzM5NjU1NjgsInR1W4F9jdHJ51joiVVCMLCJ0ZW5bNRfcm1aKfruNvSNjb3B11joiV1c1LCJ0aWqi0IIx \\ kNgROFvVFEiLCJuYmYi0jESMzM5NjU1NjgsInR1WFiYaJIJizNJYMYIVMILmRCV1RBZ \\ kNqROFvVFEiLCJuYmYi0jESMzM5NjU1NjgsInR1WFH9HJHJJjjijVNVFYUVM12NZUZM2NJ3Njb3B11joiV1c1LCJ0aWqi0IIx \\ kNqROFvVFEILCJuYmYi0jESMzM5NjU1NjgsInR1mFudF9jdHJ51joiVVMLCJ0ZW5bNRfcmVnaWuX3Njb3B11joiV1c1LCJ0aWqi0IIx \\ kNqROFvVFEILCJuYmYi0jESMzM5NjU1NjgsInR1mFudF9jdHJ51joiVVMILCJ0ZW5bNRfcmVnaWuX3Njb3B11joiV1c1LCJ0aWqi0IIx \\ kNqROFvVFEILCJuYmYi0jESMzM5NjU1NjgsInR1mFudF9jdHJ51jv1VYMIX$

 $\label{eq:mission} MjMONTY30C0xMjMOLWFiY2QtMTIzNC1hYmNkZWYxMjQ1NjciLCJ2ZXI10iIyLjAiLCJ4bXNfcGRsIjoiTkFNIiwieG1zX3RwbC16ImVuInO. Q6butUWCcM0hPmTcpj7jTv8a0j1G5cuLptA1xWpop0d12x_tyNdjG7udx59Af1Cz5Fd2FR3I95wjCkKFhCCC27GWLyjXP8PGAMCImOysXPp LC20PEyGBjDxemn9hJpUcjmTXhb204fJFQkFSNRHTJ1XgK0kCQYIZpQTRSzbgQD8Q7isPhy5no2TvmHQR-CW5DFg-xi9q5cAXJ20kU3J8ea fR_hPcNJcSsFAyM4s34iXIvPaFu5LD1xef2SArMe0uxlJwwkZIyop3n90o7IB8Ztn6f20MfY1Nq73fJS23FRpFH11Th-Z081WsmfAc2I1YK 0850pDToQLa10EtSveq0Q$

The token can be decoded and inspected with an online tool such as https://jwt.ms. The sample data is in the same format and length as a real token issued by Entra.

```
{
  "typ": "JWT",
  "alg": "RS256",
  "kid": "e6q73nti-47exXmw_yS_-rGm-CFvyOsogtsEMApOQS8"
}.{
  "email": "alice@contoso.com",
  "family_name": "Example",
  "given_name": "Alice",
  "]ogin_hint": "O.aaaaabbbbbbbbbbbbbbbbbbcccccccdddddddeeeeeeffffffgggggggghhhhhhiiiiiiijjjjjjjkkkkkkklllllllmmmmmm
  nnnnnnnnoooooopppppppqqqqrrrrrsssssdddd",
  "name": "Alice Example",
  "oid": "12345678-1234-abcd-1234-abcdef124567",
  "onprem_sid": "S-1-2-34-5678901234-1234567890-1234567890-1234567",
  "preferred_username": "alice@contoso.com",
  "rh": "0.aaaaabbbbbbccccddddeeeffff12345gggg12345_124_aaaaaaa.",
  "sid": "12345678-1234-abcd-1234-abcdef124567",
  "sub": "aaabbbbccccddddeeeeffffgggghhhh123456789012",
  "upn": "alice@contoso.com",
  "uti": "AAABBBBBccccdddd1234567",
  "verified_primary_email": [
    "alice@contoso.com"
  1.
  "verified_secondary_email": [
    "alice2@contoso.com"
 ],
  "acct": 0.
  "aud": "relyingparty.example.com",
  "auth_time": 1733965568,
  "exp": 1736557568,
"iat": 1733965568,
  "ipaddr": "203.0.113.0",
  "iss": "https://contoso.com",
"jti": "fGYCO1mK2dBWTAfCjGAoTQ",
  "nbf": 1733965568,
  "tenant_ctry": "US",
  "tenant_region_scope": "WW",
  "tid": "12345678-1234-abcd-1234-abcdef124567",
"ver": "2.0",
  "xms_pdl": "NAM",
  "xms_tpl": "en"
}.[Signature]
ŀ
```