Zero-knowledge credentials with deferred revocation checks

Melissa Chase, Esha Ghosh, and Srinath Setty
Microsoft Research

Daniel Buchner
Microsoft Identity

July 2, 2020

1 Introduction and overview

This document describes a new form of credentials with strong privacy guarantees, which we refer to as *zero-knowledge credentials*. Our target setting is the decentralized identity (DID) ecosystem where users are in charge of managing their own identities and credentials. To better understand the motivations for the choices and trade-offs made in the following credential scheme, we will examine three scenarios that feature unique issues and contextual considerations:

1. **Publishing a resume on a career networking app** is something hundreds of millions of people do every year. This represents an open disclosure of work history the user wants others in the wider business community to be aware of. The expectation of privacy for this information is inherently low, as the user intends for this information to be widely known and verified by all who see it. For this use case, we simply need a credential mechanism that is capable of *verifying signed statements by issuer* that are verifiable by observers.

   Once a user has shared their career-related credentials, Verifiers (i.e. employers, agencies, or other organizations) may need to periodically **check the current status of credentials** (e.g. an employee leaves one job for another). These sorts of status checks on credentials will generally be programmatically run at intervals, which may occur when the user is not actively engaged in an app, outside of normal waking hours, or when the user is not connected to the internet. This use case underscores the need for a feature that allows verifying parties to *check the status of a credential without requiring the user to connect to the site and perform an interactive task*.

   Similarly, **interviewing for a new job** can be stressful, and in many cases individuals do so while currently employed. While a user may not care if observers are able to validate the past work history they list on career networking sites, prospective employers often want to validate current employment, which likely includes checking the active status of the current employment credential an applicant presents. If the prospective employer is a competitor of the user's current employer, any check for active status that alerted the company to their employee's intentions could cause problems for the applicant. As such, we need a credential scheme that *verify active status of a credential without disclosing any information about the credential or its holder to the issuer*.

2. **Proving age or entitlement as a condition of access** to a product or activity is something almost everyone is familiar with. From proving your age for online purchases to showing your ID at a local bar, proving one's age is a frequent activity. In these scenarios, there are two primary user desires: 1) to prove only the age-related fact required to gain access (e.g. to avoid a bartender seeing your home

address on your driver's license), and 2) not being tracked everywhere you provide these proofs. To assuage these user requirements, we need a credential mechanism that is able to *prove facts without disclosing precise or related data* (e.g. age ≥ 21, not DOB), and *protects users from being tracked when they present credentials*.

Following on with the example of a driver's license, the status of such an entitlement may need to be verified in instances where there is no connectivity, such as checking the active status of a user's driver's license during a traffic stop in a remote location. In these cases, we need a credential scheme that allows verifying parties to *cache status proofs for offline use*, and use those proofs to *verify credentials locally, without connecting to a remote oracle*.

Some credentials indicating the holder is likely a real person, like a driver's license or university student card, are trusted by other Verifiers to gate access to goods and services. In some cases a provider may want to limit provision of a good or service to one per person, which requires awareness of whether or not a specific credential has already been used to redeem the good or service in question. To avoid duplicate provision of goods or services, *Verifiers should have the option to enforce replay protection* to determine whether a credential has already been used with them.

3. **Homeowner's insurance requirements** are imposed by banks to protect against potential losses. Banks require home loan holders to maintain an insurance policy, and a lapse in that policy can be subject to a range of penalties. Banks, and other entities with similar requirements, need to be able to check the status of credentials that prove users are abiding by their contractual terms. These checks are likely to be automated and recurring, thus we need a credential scheme that *does not require users to connect with relying parties to manually verify credentials that must be checked on a recurring basis*.

To address the above scenarios as well as many other similar scenarios, we treat a credential as a set of key-value tuples where keys hold typed attribute names with respect to some agreed upon schema (e.g. Key = DOB, Value = 01/01/2000) and values hold assigned attribute values. There are three types of entities: (1) *issuers* who issue credentials (e.g., enterprises, universities, department of licensing, insurance providers), (2) *users* who acquire credentials from one or more issuers; and (3) *relying parties* (RPs) who validate claims about credentials made by users. Our principal focus is to enable these interactions while still protecting privacy and security.

Desired Properties. In more detail, we desire the following features, which we state informally:

1. **Unforgeability.** A user cannot forge issuer-generated credentials.
2. **Unlinkability.** A coalition of RPs cannot link activities of users across RPs.
3. **Minimal disclosure.** A user only reveals what is necessary to establish a relationship with an RP
4. **Privacy against issuers.** An issuer does not learn whether a user (who acquired a credential from the issuer) has interacted with a given RP.
5. **Revocability.** An issuer can revoke a previously-issued credential without users' cooperation.
6. **Deferred and offline revocation checks.** An RP can verify the revocation status of a credential asynchronously without users' participation. An RP can also verify a new user's credential and revocation status without contacting the issuer, in which case revocation status is with respect to the last time the RP contacted the issuer.

7. **Protection against credential reuse.** An RP can ensure no credential is used by different DIDs. Note that we want to allow a user to use completely unrelated DIDs with the RP and the issuer, so we can't prevent a user from sharing their credential and secret key with a friend who will then present it to an RP with their own DID. However, we guarantee that only one user can use each credential (either the correct user or a friend with whom he has shared his secret key).

Prior literature on privacy-preserving credentials does not consider the deferred revocation property; in most schemes, the user is in charge of proving that their credential is not revoked. As a result, in applications such as the home owner's insurance scenario described above, a bank or a mortgage company cannot on their own check the revocation status of their customers' insurance policies. We believe this is a necessary feature if we want to transparently replace existing non-private credential schemes.

Assumptions To achieve the above property, our scheme makes the following assumptions. First, we assume that the issuers and relying parties do not collude to learn additional information about their users. This is a strong assumption and may not hold for all issuers, but we believe it may be reasonable to guarantee this via legal commitments, particularly when the issuer is a large entity with significant reputation risk in deviating from the protocol. Note that we make no assumption on what the issuer does internally with its own data, only that the issuer does not receive extra information from misbehaving RPs.

Second, users select an RP-specific DID to interact with each RP and present their credentials with respect to this DID. Furthermore, they use the same DID across multiple interactions with the same RP. As a result, for most of this document we focus on the case where unlinkability across multiple presentations to the *same* RP is not needed. See section 2.3 for brief discussion of how to allow unlinkability with respect to the same RP at the cost of giving up deferred revocation and reuse protection.

Overview of our solution. We employ standard cryptographic tools: (1) zero-knowledge proofs (which enables a prover to prove any mathematical statement to a verifier without disclosing anything besides the validity of the statement), and (2) digital signatures and hash functions.

Issuing credentials. An issuer digitally signs a *credential*—a list of attributes—with its private key. Any RP can download digests of each active credential from the issuer (the digest here is a cryptographic hash of a credential). To achieve unlinkability described above, these digests are RP-specific, and to prevent the RP from learning hidden attributes through a brute force attack we include a random credential specific nonce r known only to the user and the issuer. For example, the digest of a credential c downloaded by RP "Contoso Corp" is computed by the issuer as $H(\text{“Contoso Corp”}||c||r)$, where $||$ is a concatenation and H is a cryptographic hash function.

Presenting credentials and verifying their validity. For a user to prove claims about credentials to an RP, the user acts as a prover in a zero-knowledge proof system and the RP acts as the verifier. Specifically, to prove that its credential (issued by an issuer) is active and not revoked to an RP (e.g., "Contoso Corp"), the user produces a zero-knowledge proof of knowledge of c and nonce r such that $d == H(\text{“Contoso Corp”}||c||r)$ and sends d to the RP. The RP verifies the proof and checks if d is present in its active users list fetched from the issuer. Note that these checks can be deferred to a later point in time by the RP (e.g., an RP can collect these proofs and digests from the user, and then later download the list of active users from the issuer and perform the checks described earlier). Similarly, if the RP has already downloaded the list of digests, he can check d against this list without having to contact the issuer, and find out whether the credential was valid at the time of his download.

Presenting claims about credentials and verifying their validity. We have thus far ignored attributes issued to a user. By employing a general-purpose zero-knowledge proof system, referred to as a zkSNARK,

the user can produce proofs of arbitrary computation over their credentials. This includes equality checks over attributes (e.g., prove that an attribute equals a specific value or falls within a range), selective disclosure of certain attributes (e.g., reveal certain attributes while hiding everything else), or computation (e.g., prove that age ≥ 21 from using date of birth included in a credential). An RP can verify such zero-knowledge proofs. Note that, in practice, the user produces a single zero-knowledge proof proving several statements at once (e.g., proving validity of the credential as described above as well as statements over attributes).

Verifying credentials but not revocation status. In some applications, an RP may not be able to interact with an issuer (to download the list of digests of active credentials) or may not be concerned about revocation status of credentials. For such applications, the user can instead (or in addition) prove in zero-knowledge that it holds a credential c and a digital signature σ such that $\mathsf{Verify}(PK_I, c, \sigma) = 1)$, where PK_I is the public key of the issuer and Verify is a digital signature verification algorithm (e.g., ECDSA's verify).

Reuse protection. To guarantee that we can identify when one user's credential is used many times at the same RP, we require the user to also present an RP specific *pseudonym*, which is a hash of the RP DID and the nonce r chosen for the user's credential. The user will have to use a zero-knowledge proof to show that the nonce used here is the same as that used in the digest d. This guarantees that there is only one pseudonym that the user can compute for his credential at each RP. The fact that the nonce is unknown to the RPs means that two pseudonyms for the same user for different RPs can't be linked together.

2 Proposed construction

The high level protocol is as follows:

Issuing Credentials We assume the user has already authenticated to the issuer as the owner of DID $\mathsf{DID}_{(U,I)}$. The issuer holds a signing key pair $I_{\mathsf{pk}}, I_{\mathsf{sk}}$ which are tied to the Issuer's DID. To issue a credential to user with DID $\mathsf{DID}_{(U,I)}$, for attributes atts, the issuer does the following:

- Pick a random nonce r and random salt salt
- Retrieve the current public key for $\mathsf{DID}_{(U,I)}$ ($\mathsf{pk}_{U,I}$) and compute $\sigma = \mathsf{Sign}(I_{\mathsf{sk}}, (\mathsf{pk}_{(U,I)}, \mathsf{atts}, \mathsf{r}))$
- Send r, salt, and σ back to the user

The issuer maintains a table where there is a separate row for *each active credential per DID* with the following information:

1. The user's DID ($\mathsf{DID}_{(U,I)}$)
2. The current public key for that DID ($\mathsf{pk}_{(U,I)}$)
3. The random nonce r
4. The salted hash of the attributes atts that the issuer is certifying for this user (but not the salt*) ($H(\mathsf{salt}, \mathsf{atts})$)

Note that this requires the issuer keeping track of when users' public keys change.

The user stores $\sigma, \mathsf{r}, \mathsf{atts}, \mathsf{salt}$.

*The purpose of storing a salted hash of the atts, but not the salt, is to make offline brute-force attack for recovering atts from the hash hard, in case the issuer's database leaks.

Credential Revocation When a credential is revoked, the issuer removes the entry for that credential from its table.

RP retrieving latest active user list The RP authenticates to the issuer as the owner of $\mathsf{DID}_{\mathsf{RP}}$. The RP will then retrieve a set of values $H(\mathsf{DID}_{\mathsf{RP}}, \mathsf{pk}_{(U,I)}, \mathsf{r}, H(\mathsf{salt}, \mathsf{atts}))$ for all of the entries of the issuer's table. To make this more efficient, the RP can send the issuer the time of its most recent state retrieval and then the issuer will compute and send the hashes for members that have been added or removed since that time.

Credential Presentation We assume the user has already authenticated to the RP as the owner of some DID $\mathsf{DID}_{(U,\mathsf{RP})}$. To show the RP that he has a credential from issuer I with DID public key I_{pk} (received under a different DID, namely, $\mathsf{DID}_{(U,I)}$) demonstrating claim claim, the user will do one or both of the following:

1. Compute $\mathsf{pseudonym} = H(\mathsf{DID}_{\mathsf{RP}}, \mathsf{r})^*$ and send it to the RP.

   Prove knowledge of a signature σ, a key pair $\mathsf{pk}_{(U,I)}, \mathsf{sk}_{(U,I)}$, a nonce r, and a set of attributes atts such that

   - $\mathsf{Verify}(I_{\mathsf{pk}}, (\mathsf{pk}_{(U,I)}, \mathsf{atts}, \mathsf{r})) = \mathtt{accept}$,
   - $\mathsf{pseudonym} = H(\mathsf{DID}_{\mathsf{RP}}, \mathsf{r})$,
   - $\mathsf{pk}_{(U,I)}, \mathsf{sk}_{(U,I)}$ is a valid keypair,
   - and atts satisfy claim.

   *The pseudonym can be removed from this option to obtain stronger unlinkability (in which the RP cannot tell when the same user returns multiple times) at the cost of losing reuse protection.

2. Compute $h = H(\mathsf{DID}_{\mathsf{RP}}, \mathsf{pk}_{(U,I)}, \mathsf{r}, H(\mathsf{salt}, \mathsf{atts}))$ and $\mathsf{pseudonym} = H(\mathsf{DID}_{\mathsf{RP}}, \mathsf{r})$ and send them to the RP.

   Prove knowledge of a key pair $\mathsf{pk}_{(U,I)}, \mathsf{sk}_{(U,I)}$, a nonce r, a salt salt, and a set of attributes atts such that

   - $h = H(\mathsf{DID}_{\mathsf{RP}}, \mathsf{pk}_{(U,I)}, \mathsf{r}, H(\mathsf{salt}, \mathsf{atts}))$,
   - $\mathsf{pseudonym} = H(\mathsf{DID}_{\mathsf{RP}}, \mathsf{r})$,
   - $\mathsf{pk}_{(U,I)}, \mathsf{sk}_{(U,I)}$ is a valid keypair,
   - and atts satisfy claim.

   In this case the RP will check that h is in the list of hashes it obtained from the issuer, and that $\mathsf{pseudonym}$ has not been seen before with a different DID.

Credential Verification The RP has three different options of verifying the credential presentation from the user:

1. Do the first check only. The advantage is that this does not require any info from the issuer. The disadvantage is that it may give stale information, i.e. there is no way to tell if the credential or the key it was originally issued to has been revoked.
2. Do the second check only. If the RP has the most recent list from the issuer, this should still give the strongest guarantees.

3. Do both checks. If the RP has not pulled the most recent list, the user's hash h may not be in that list, in which case it won't know if the user's credential is still active. But it will be able to verify that the user had a credential at one point.

2.1 Security

This system achieves the properties discussed in the intro as follows:

Unlinkability During credential presentation, the user presents a claim, a pseudonym, a zero-knowledge proof, and in the case of Option 2 or 3, the hash h. Because the r is random and unknown to any of the RPs, we have the guarantee that

$$h_1 = H(\mathsf{DID}_{\mathsf{RP1}}, \mathsf{pk}_{(U,I)}, \mathsf{r}, H(\mathsf{salt}, \mathsf{atts})), \mathsf{pseudonym}_1 = H(\mathsf{DID}_{\mathsf{RP1}}, \mathsf{r})$$

and

$$h_2 = H(\mathsf{DID}_{\mathsf{RP2}}, \mathsf{pk}_{(U,I)}, \mathsf{r}, H(\mathsf{salt}, \mathsf{atts})), \mathsf{pseudonym}_2 = H(\mathsf{DID}_{\mathsf{RP2}}, \mathsf{r})$$

look like 4 random strings from the point of view of RPs 1 and 2, even if they pool all of their information. The zero-knowledge guarantee ensures that the proofs do not provide any additional information. Thus, the only thing the RPs will learn is the claims presented. In particular, they will not be able to tell that the same user presented a credential at both RPs unless that is inherently revealed by the claims.

Minimal disclosure As discussed in the previous argument, the hash and pseudonym look like random strings, and the zero-knowledge proof provides no additional information. Thus, the only thing the RP learns is that the claim holds, as desired.

Privacy against Issuers In the above protocol, the issuer only participates in credential issuance and in sending RPs the latest membership state, neither of which tells it anything about when/where a user's credential is presented.[1]

Revocability Credential revocation just involves the issuer removing an entry from its database.

Deferred revocation checks Credential presentation does not require the user or RP to contact the issuer or the ledger. Moreover, if the user authenticates with option 2 or 3, the RP can at any later point download an updated hash list and check whether the user's credential is still valid.

Reuse protection The pseudonym is computed from the r, which is fixed for a given credential. This means that if the user tries to for example share their credential with multiple friends, the RP will be able to detect that the same pseudonym has been used with multiple different DIDs. Note that this holds even if the user updates their DID key pair.

[1] As discussed below, this assumes that we can ensure, e.g. with legal agreements, that RPs and Issuers will not misbehave and combine information outside of this protocol. We believe this may be a reasonable assumption for large issuers with a reputation to maintain.

2.2 Design choices and known weaknesses

Here we discuss some of the known security weaknesses of our design. Some are inherent in any credential system, or implied by the features we want to achieve, and a few are the result of our specific protocol choices.

2.2.1 Inherent in any credential system

If issuer is corrupt A corrupt issuer can issue false credentials or (if the system allows issuer revocation) they can revoke a user's credential without justification.

Sharing credentials The user could give his credential to a friend. This would require sharing their secret key as well. Since we make no assumption about how the DIDs that the user uses with issuer and RPs are generated, there is no way to tell whether two DIDs actually belong to the same person.

Note that in our system the credential can only be used for a single DID at each RP because of reuse protection, so the cost is somewhat limited.

2.2.2 Inherent in any system which supports deferred revocation

If the issuer state leaks (in our case, the issuer database), an RP with the leaked information can identify (i.e. learn the DID they have established with the issuer) any user who has showed a credential. This is inherent because the adversarial RP can try "revoking" different DIDs and see which one causes the deferred revocation checks to fail.

Note that it is not necessary that the user attributes be revealed, and in fact in our scheme they are hidden because the issuer stores only the salted hashes.

If the RP and issuer collude they can identify all the users who authenticate to the RP using Option 2 or 3, for the same reason as above.

2.2.3 Inherent in any system which supports reuse protection and offline presentation

Credential compromise If the user's credential is compromised, an RP who gets to see the r can identify any past/future authentications by this user that use Option 2 or 3.

To see why this is inherent note that the RP can use the credential to generate a presentation for this user and compare it to the presentations he has recorded. Our reuse protection guarantees that the verifier will be able to tell if the same credential is being used a second time. This could theoretically be avoided by including the issuer in every credential presentation, but then we could no longer support offline credential presentation, and from a practical point of view it would make the system much more complex.

2.2.4 Other known weaknesses

If issuer state leaks Anyone with access to the leaked database can see who has a credential (i.e. learn the DID they have established with the issuer). Theoretically this could be prevented by an issuer who stores nothing about who credentials are issued to, but in reality it seems likely that any issuer would want to keeps some record of what credentials they have issued.

We could alternatively remove this by modifying our construction to not include the DID or the public key in the digest, in which case the issuer would not strictly have to store these values. This would however allow a user to share his credential with other users without sharing any long term secret key.

Note that it is not necessary that the user attributes be revealed, and in fact in our scheme they are hidden because the issuer stores only the salted hashes.

If the isser uses bad randomness If the randomness in the r is bad it may make it easier for an RP to brute force and identify the user behind a particular authentication, or identify the users in the current user list. If the randomness in the salt is bad, it may make it easier brute force and learn the attributes if the issuer database leaks or if the rs are also bad and one of the RPs is corrupt. This could be mitigated by having the user and issuer jointly generate the randomness.

RP can learn when the user last updated their public key, or when the credential was issued if the user has not updated their public key since credential issuance.[2] This combined with public DID info might leak info about what DID the user uses with the Issuer. In either case granularity that depends on how often the issuer pushes updates, and the RP doesn't learn how many times the public key has been updated.

If RPs collude with one another They can potentially correlate information about when different entries were added to the list of hashes. This means if only a few entries are added in each update, and one of those users authenticates to both RPs, they'll know that it's fairly likely they were talking to the same user.

This leakage is to some extent inherent in deferred revocation checks because the adversarial RPs can each perform multiple deferred revocation checks and see when a given credential becomes valid/invalid. However, theoretically interaction with the issuer could be required for each check, which could limit the number of such queries the RPs could make. In our scheme, once the RP has retrieved the most recent state, they can make unlimited revocation checks, which makes the potential for correlation somewhat larger. We see this as a reasonable compromise for the improved efficiency.

2.3 An option for stronger unlinkability

Note that, in our current design, multiple presentations of the same credential to the same RP is detectable by the RP. Here we sketch a variant of our scheme that can provide stronger unlinkability protection . This variant uses a traditional anonymous credential system, e,g. [16]. For a more detailed discussion of classical anonymous credential systems, please see Section 4.

Credential Presentation Now the first time that a user presents his credential to a given RP is treated differently from the subsequent presentations.

First Presentation When a user first uses his credential with an RP, he gets issued a signature on the presented attributes by the RP, where the RP uses a cryprographic blind signature mechanism to issue the signature. This credential can include $\mathsf{sk}_{(U,RP)}$ as an additional attribute. The RP also issues a revocation handle to the user. The RP stores the hash h from the presented credential along with the revocation handle for the anonymous credential system.

[2] This is because updating the public key will result in a new hash value in the set the RP retrieves from the issuer. By looking at when the hash the user presents first appeared in a set obtained from the issuer, the RP can tell roughly when the user's pk was changed.

Subsequent Presentations When the user wants to use his credential again with the RP he shows that he still owns the credential signature issued by the RP and that the corresponding revocation handle has not been revoked, using a ZK proof. He also proves knowledge of $\mathsf{sk}_{(U,RP)}$.

Credential Revocation When the RP downloads a new list of hashes from the issuer, if one of its stored hashes has been removed in the issuer's list, it uses the revocation handle stored against that hash, to revoke the (anonymous) credential issued to the corresponding user.

2.3.1 Security and Efficiency

Unlinkability. This gives unlinkability even when the credential is used multiple times with the same RP. Note that if we want this stronger unlinkability, reuse protection is inherently impossible.

The privacy guarantees are also stronger in the sense that even if the RP and issuer collude, they will only be able to tell when the user first accesses the RP; later visits will be anonymous even in face of this collusion.

Deferred revocation. This option won't support deferred revocation - the RP needs to ensure his list of digests is up to date before each anonymous credential is presented; if additional revocations are discovered after the anonymous credential is presented, there will be no way to tell whether the presented credential belonged to one of those users. On the other hand, it does allow offline revocation checking, in that as long as the RP has downloaded the latest list of digests, the user and RP do not need to contact issuer, ledger, or any other party during credential presentation.

Efficiency. There is somewhat more download cost for the user, who now needs to download something proportional to the number of users visiting the RP. But this should at least be smaller than the cost to download the issuer's active user list, which is what Sovrin etc do.

We note that here we are in the keyed verification setting[20], so we can use e.g. the credentials from [20] which rely only on elliptic curve operations.

3 Cryptographic tools

The credential scheme from the prior section requires producing zero-knowledge proofs of knowledge. For this, we rely on zero-knowledge succinct non-interactive arguments of knowledge (zkSNARKs) [24, 12], a type of zero-knowledge proof of knowledge with succinct proofs and fast verification times.[3] A bit more formally, a zkSNARK involves a *prover* and a *verifier* where the prover can prove to a verifier, by producing a proof π, the knowledge of a secret witness w such that $\Psi(x, w) = y$, where Ψ is a program (e.g., in a subset of the C programming language), and x and y and public inputs and outputs of Ψ respectively. More crucially, π does not reveal anything about w except that the prover knows a w such that $y = \Psi(w, x)$.

Choice of hash functions and digital signatures. By relying on a general-purpose proof system such as zkSNARKs, we can employ standardized hash functions (e.g., SHA-256) and digital signature schemes (e.g., ECDSA). For efficiency of proofs of digital signatures, we rely on SNARK-friendly elliptic curves, which are

[3] Another potential solution would be to use special signature schemes with efficient zero-knowledge proofs of knowledge. However, it does not directly allow proving predicates over attributes in a credential, which requires a machinery such as zkSNARKs. There are also no toolchains to automatically produce proofs using such signature schemes. In the future, we plan to investigate creating a hybrid between these two types of schemes to potentially improve efficiency.

standard elliptic curves but parameters are chosen so that operations on elliptic curves (e.g., point addition) have an efficient representation in the computational model of zkSNARKs.

Choice of a zkSNARK. Many modern zkSNARKs [23, 25] require a *trusted setup* where a special multi-party ceremony [11] must be held to create cryptographic parameters for producing and verifying proofs (which are called proving and verification keys). In general, this is undesirable from a cost and trust perspective. In our context, although issuers can create such trusted cryptographic material, it poses several issues: (1) Each issuer will have to create a separate proving key for each type of statement that a user might want to prove to an RP; this means that users with credentials from k different issuers will need to store at least k proving keys. Even with the state-of-the-art schemes [23, 25], the size of the proving key scales with the the size of the program; for our target statements this is hundreds of MBs per credential; (2) Since the proving key is tied to an issuer, an RP cannot verify statements made against credentials issued by multiple issuers. Beyond the trusted setup, these zkSNARKs also rely on non-standardized cryptographic hardness assumptions.

Fortunately, a new class of zkSNARKs, called *transparent zkSNARKs*, do not require a trusted setup nor do they require non-standardized cryptographic assumptions [30, 10, 22, 29]. Among these schemes, Spartan [29] offers the best performance in the literature, and its security is based on the hardness of the discrete logarithm problem. Specifically, it offers the fastest prover and verification times, up to several orders of magnitude lower costs than alternate schemes. The proof sizes are shorter than all other transparent schemes except for Bulletproofs [15], which incurs orders of magnitude higher proving and verification costs.

Encoding statements and cost estimates. Most zkSNARKs, including Spartan, support a general-purpose programming model for encoding statements to prove and verify. Specifically, they support rank-1 constraint satisfiability (R1CS), a popular and a standardization candidate for expressing statements for zero-knowledge proof systems. It is indeed tedious and error-prone to write programs in R1CS, but we will employ a compiler toolchain [1, 2, 4] to automatically produce statements in R1CS from high-level programs.

In zkSNARKs, the primary bottleneck is the cost of producing a proof. The cost to prove a statement scales linearly with the size of the R1CS program (i.e., the number of constraints), so we first depict the size of the R1CS program for various building blocks we rely on:

- Digital signature: knowledge of a 256-bit ECDSA secret key ($\approx 5,000$ constraints), ECDSA signature verification ($\approx 10,000$ constraints [26, 28])
- Hash functions: SHA-256 (≈ 400 constraints/byte [2]), Pedersen hash (≈ 21 constraints/byte), and MiMC (≈ 5 constraints/byte)

We now estimate the costs of the proof system under Spartan for proving simple predicates (equality, range checks, etc.) over a credential with 1 KB of (encoded attributes) signed using ECDSA with a SNARK-friendly Edwards curve. The R1CS program for this type of statement requires about 2^{15} constraints for credential presentation with option#1. For this program, Spartan, on a single CPU core of a Microsoft Surface Laptop 3 takes under 170 ms to produce a proof of size of 15 KB that can be verified in about 17 ms. (There is a configuration in Spartan to make proofs shorter by $3\times$, which approaches the proof sizes under Bulletproofs, but this incurs a small constant factor increase in proving and verification costs.) For option#2, the prover's costs increase by about $3\times$, and proof sizes and verification times by about $1.5\times$. Since costs primarily depend on the program size, one can improve costs by leveraging efficient encodings of statements.

4 Related work

There is a growing interest in building a decentralized identity (DID) ecosystem, with various startups and foundations working to unlock a wide range of business opportunities [27, 7, 6]. A core primitive in this ecosystem is a decentralized PKI, which replaces a traditional PKI, and a claims-based credential infrastructure, which aims to offer strong privacy guarantees. Here, we review existing proposals and discuss how our proposal compares with them. At a high level, all proposals employ a similar model and specify protocols among three types of entities: (1) an issuer, (2) a user, and (3) a relying party (RP).

Sovrin. The major difference between our proposal and that of Sovrin [5, 27] is in credential revocation. Sovrin handles it as follows: an issuer publishes a list of active users and a cryptographic accumulator containing the list of active users on a public blockchain, which it updates after each revocation. When a user proves its claims to an RP, it additionally needs to do the following: fetch the latest list of active users and the latest accumulator from the blockchain and prove to the RP that it is on that list in zero-knowledge, i.e., without revealing which user id corresponds to her in the list. This requires both the user and the RP to have online access to the latest accumulator and the list posted by the issuer. Users must also download the entire list, which may be large.

In our protocol, RPs can periodically download the latest list of active users from issuers and check the revocation status of all the claims presented in the last epoch, i.e., in a deferred manner. More fundamentally, the user does not need to download a list of active users from the ledger (or an issuer) in order to present a claim. Furthermore, they do not participate in the deferred revocation status check. This significantly reduces network costs for the users, which we expect to be mobile devices.

Another point of difference between Sorvrin's design and ours is in the unlinkability guarantee and the credential reuse protection. Sovrin's design provides full unlinkability across multiple claim presentations by the same user to the same RP, which our basic design does not. However, in order to get unlinkability they have to give up credential reuse protection: a user can reuse her claim to an RP multiple times without being detected. If an RP in Sovrin wants reuse protection, then they inherently must lose this stronger unlinkability guarantee, as discussed in Section 2.2.

Sovrin also provides privacy even when issuer and RP collude; as discussed in Section 2.2 this stronger property is incompatible with deferred revocation, so we do not satisfy it.

Finally, Sovrin proposes to use a different ZK proof machinery as compared to our proposal which uses Spartan [29].

Iden3. In the Iden3 proposal, an issuer computes a Merkle tree of all claims and posts the root of the tree on a tamper-proof ledger. When claims change, the issuer updates the root on the ledger. But, details of the Merkle tree construction and revocation protocols are unspecified. We found a preliminary presentation [3] from Concordium [6] where they propose an ID layer to cater to their payment system, but it does not specify protocols for credential revocation.

Anonymous credentials. Anonymous credentials, first proposed by Chaum [21], allow users to obtain credentials from issuers and then present them to RPs, revealing only whether their attributes satisfy the given claims. To allow for revocation the issuer, or in some cases the RP, post a list of revoked (or active) credentials and users prove in zero knowlege their credentials are not on (or on) this list. Accumulators [18] were proposed to allow these proofs to be more efficient. These schemes were originally proposed in a context without any global notion of identity; Sovrin essentially ports this approach to the DID context.

Some of these schemes do not provide unlinkability across multiple presentations of the same credential [14, 8] (and in that sense are weaker than our proposal), others provide full unlinkability across multiple presentations to the same or different RPs [17, 19, 9] (stonger than our proposal), and all of them consider the stronger setting where privacy must hold even when the issuer and RPs collude. In the schemes which gives unlinkability across multiple presentations the RP can decide whether or not to require reuse protection (at the cost of removing unlikability across multiple presentations to that RP). Verifier local signatures [13] allow for deferred revocation checks, but provide a weaker guarantee in that once a credential is revoked all previous and future uses of that credential are linkable.

References

[1] Pequin: An end-to-end toolchain for verifiable computation, SNARKs, and probabilistic proofs. `https://github.com/pepper-project/pequin`, 2016.

[2] A high-level framework for developing efficient zk-SNARK circuits. `https://github.com/akosba/xjsnark`, 2018.

[3] Concordium identity layer. `https://www.youtube.com/watch?v=PdehZjOjEOs&t=152s`, 2019.

[4] Zinc. `https://zinc.matterlabs.dev/index.html`, 2020.

[5] Anoncreds design. `https://github.com/hyperledger/indy-crypto/blob/master/libindy-crypto/docs/AnonCred.pdf`, Last Accessed: 6/3/2020.

[6] Concordium. `https://concordium.com/`, Last Accessed: 6/3/2020.

[7] iden3. `https://iden3.io/`, Last Accessed: 6/3/2020.

[8] F. Baldimtsi and A. Lysyanskaya. Anonymous credentials light. In A.-R. Sadeghi, V. D. Gligor, and M. Yung, editors, *ACM CCS 13*, pages 1087–1098. ACM Press, Nov. 2013.

[9] M. Belenkiy, M. Chase, M. Kohlweiss, and A. Lysyanskaya. P-signatures and noninteractive anonymous credentials. In R. Canetti, editor, *TCC 2008*, volume 4948 of *LNCS*, pages 356–374. Springer, Heidelberg, Mar. 2008.

[10] E. Ben-Sasson, I. Bentov, Y. Horesh, and M. Riabzev. Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Report 2018/046, 2018.

[11] E. Ben-Sasson, A. Chiesa, M. Green, E. Tromer, and M. Virza. Secure sampling of public parameters for succinct zero knowledge proofs. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2015.

[12] N. Bitansky, R. Canetti, A. Chiesa, and E. Tromer. From extractable collision resistance to succinct non-interactive arguments of knowledge, and back again. In *Proceedings of the Innovations in Theoretical Computer Science (ITCS) Conference*, 2012.

[13] D. Boneh and H. Shacham. Group signatures with verifier-local revocation. In V. Atluri, B. Pfitzmann, and P. McDaniel, editors, *ACM CCS 04*, pages 168–177. ACM Press, Oct. 2004.

[14] S. Brands. *Rethinking Public Key Infrastructure and Digital Certificates— Building in Privacy*. PhD thesis, Eindhoven Institute of Technology, Eindhoven, The Netherlands, 1999.

[15] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2018.

[16] J. Camenisch and A. Lysyanskaya. Efficient non-transferable anonymous multi-show credential system with optional anonymity revocation. In B. Pfitzmann, editor, *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, volume 2045, pages 93–118, 2001.

[17] J. Camenisch and A. Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In B. Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 93–118. Springer, Heidelberg, May 2001.

[18] J. Camenisch and A. Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In M. Yung, editor, *CRYPTO 2002*, volume 2442 of *LNCS*, pages 61–76. Springer, Heidelberg, Aug. 2002.

[19] J. Camenisch and A. Lysyanskaya. Signature schemes and anonymous credentials from bilinear maps. In M. Franklin, editor, *CRYPTO 2004*, volume 3152 of *LNCS*, pages 56–72. Springer, Heidelberg, Aug. 2004.

[20] M. Chase, S. Meiklejohn, and G. Zaverucha. Algebraic MACs and keyed-verification anonymous credentials. In G. Ahn, M. Yung, and N. Li, editors, *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, pages 1205–1216, 2014.

[21] D. Chaum. Showing credentials without identification: Signatures transferred between unconditionally unlinkable pseudonyms. In F. Pichler, editor, *EUROCRYPT'85*, volume 219 of *LNCS*, pages 241–244. Springer, Heidelberg, Apr. 1986.

[22] A. Chiesa, D. Ojha, and N. Spooner. Fractal: Post-quantum and transparent recursive proofs from holography. Cryptology ePrint Archive, Report 2019/1076, 2019.

[23] R. Gennaro, C. Gentry, B. Parno, and M. Raykova. Quadratic span programs and succinct NIZKs without PCPs. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2013.

[24] C. Gentry and D. Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 99–108, 2011.

[25] J. Groth. On the size of pairing-based non-interactive arguments. In *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2016.

[26] J. Lee, K. Nikitin, and S. Setty. Replicated state machines without replicated execution. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2020.

[27] M. Lodder and D. Khovratovich. Anonymous credentials 2.0. `https://s3-us-west-1.amazonaws.com/groupsioattachments/26296/30142553/103/0?AWSAccessKeyId=AKIAJECNKOVMCCU3ATNQ&Expires=1591239662&Signature=dE8QRTcjX9F3xnzbAR9%2FBEudXFA%3D&response-content-disposition=inline%3B+filename%3D%22AnonCreds2.pdf%22`, Last Accessed: 6/3/2020.

[28] A. Ozdemir, R. S. Wahby, and D. Boneh. Scaling verifiable computation using efficient set accumulators. Cryptology ePrint Archive, Report 2019/1494, 2019.

[29] S. Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In *Proceedings of the International Cryptology Conference (CRYPTO)*, 2020.

[30] R. S. Wahby, I. Tzialla, A. Shelat, J. Thaler, and M. Walfish. Doubly-efficient zkSNARKs without trusted setup. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, 2018.