zkRevoke: Configurable Untraceability for Verifiable Credentials using ZKPs

Praveensankar Manimaran
University of Oslo
Norway

Mayank Raikwar
University of Oslo
Norway

Thiago Garrett
University of Oslo
Norway

Arlindo F. da Conceição
Federal University of São Paulo
Brazil

Leander Jehl
University of Stavanger
Norway

Roman Vitenberg
University of Oslo
Norway

Abstract

Systems managing Verifiable Credentials are becoming increasingly popular. Unfortunately, their support for revoking previously issued credentials allows verifiers to effectively monitor the validity of the credentials, which is sensitive information. While the issue started to gain recognition, no adequate solution has been proposed so far.

In this work, we propose a novel framework for time-limited continuous verification. The holder is able to individually configure the verification period when sharing information with the verifier, and the system guarantees proven untraceability of the revocation status after the verification period expires. Differently from existing systems, the implementation adopts a more scalable blacklist approach where tokens corresponding to revoked credentials are stored in the registry. The approach employs Zero-Knowledge Proofs that allow holders to prove non-membership in the blacklist. In addition to theoretically proving security, we evaluate the approach analytically and experimentally and show that it significantly improves bandwidth consumption on the holder while being on par with state-of-the-art solutions with respect to the other performance metrics.

Keywords

Verifiable Credential, Revocation, Privacy

1 Introduction

Verifiable Credentials (VCs) [1–6] represent conventional credentials, such as passports and driving licenses, in a digital format. Similar to how individuals store and share conventional credentials, digital identity holders store VCs and share them with verifiers. VC management systems [7, 8] are increasingly adopted in the industry, including both public [9, 10] and private [11–14] sector organizations. This adoption is further fuelled by commonly accepted standards such as the W3C Verifiable Credentials Specification [1].

A VC can be revoked for reasons such as misuse or invalidation. For example, an employee ID issued in the form of a VC will be revoked when the corresponding employee leaves the organization. Therefore, it is critical for verifiers to ascertain that the VC has not been revoked prior to granting services. Support for revocation is provided by many systems, including IRMA [7, 15], Anoncreds [8], and IOTA Identity [16, 17].

Unfortunately, revocation introduces a new issue: the verifier may now be able to repeatedly probe the revocation status and thus, effectively monitor the validity of the VC. A VC may pertain to sensitive information, such as employee's position with the current employer, credit score, or healthcare-related situation. A holder may want the verifier to perform the verification within a limited time after sharing the VC rather than indefinitely. This issue is widely recognized by the state-of-the-art [17–20]; it is considered one of the key privacy challenges by the IETF Standard of [21] and by EBSI [9, 22], an initiative of the European Commission and the European Blockchain Partnership. IRMA, Anoncreds, and the academic works of [18] and [23] allow the verification period to be shorter than the VC expiration period, see Table 1. Unfortunately, they do not let the holder configure the verification period, thereby lending themselves only to specific use cases. Adding this configurability to existing approaches is challenging for two reasons: (a) the main responsibility of cryptographic computations in existing approaches falls on the issuer creating the VC rather than on the holder sharing it, and (b) existing approaches rely on state compaction techniques such as accumulators [24] to achieve good performance. Alas, these techniques prevent implementation of configurable verification period.

The main contribution of our work is that we propose a novel framework for time-limited continuous verification. The holder is able to individually configure the verification period when sharing information with the verifier, and the system guarantees proven **untraceability** of the revocation status after the verification period expires. The framework includes two additional requirements: (a) one-time sharing for a continuous verification: the verifier does not need to contact the holder after the initial sharing is completed, and (b) unobservable verification: the issuer should not be able to learn anything about the verification. We provide a scalable solution for this framework, and we demonstrate that all cryptographic operations take practically acceptable times.

Our solution is a radical departure from existing schemes: all previously known solutions for VC revocation that provide untraceability are based on a whitelist of tokens corresponding to issued unrevoked VCs. In contrast, we employ a blacklist of previously revoked VCs combined with ZK proofs that allow holders to prove non-membership in the blacklist. Since in practice, the number of revoked VCs during a period of time is several times smaller than the number of issued VCs during the same period, the blacklist approach is inherently more efficient in terms of the bandwidth consumption and storage. For example, only 1% of certificates are revoked in the domain of Web's PKI [30, 31]. To compensate, many systems based on the whitelist approach employ state compaction

Proceedings on Privacy Enhancing Technologies 2026(3), 360–377

https://doi.org/10.56553/popets-2026-0086

Table 1: Representative solutions proposed in the state-of-the-art (✓- covered; ×- not covered;)

Related Work	Separation between expiration and verification	Configurable Period	One-time VP sharing	Unobservable verification
IRMA [7, 15], Anoncreds [8], Sitouah *et al.* [18], zk-creds [23], Xu *et al.* [25]	✓	×	×	✓
EVOKE [26]	×	×	×	✓
Prevoke [27]	×	×	×	✓
Schumm *et al.* [28], IOTA Identity: Revocation Bitmap 2022 [16]	×	×	✓	✓
IOTA Identity: Revocation Timeframe 2024 [17]	×	×	×	✓
Papathanasiou *et al.* [29]	×	×	✓	×
This work	✓	✓	✓	✓

techniques, such as accumulators. Accumulators are even better than blacklist in terms of required storage space: they achieve constant complexity compared to the linear complexity of blacklist or whitelist without accumulators. However, state compaction techniques do not mitigate bandwidth consumption and besides, they prevent implementation of configurable verification period.

In summary, our contributions are as follows:

- We introduce a novel framework for time-limited continuous verification, with formally defined requirements. For example, this is the first work to define a formal security game for the concept of untraceable revocation status.
- We propose a blacklist-based solution as a departure from commonly used whitelist-based schemes.
- We prove the correctness of our protocol under standard cryptographic assumptions.
- We evaluate the proposed solution both analytically and experimentally. We conclude that it significantly improves bandwidth consumption on the holder while providing a nuanced comparison with state-of-the-art solutions wrt. the other performance metrics.

2 Verifiable Credentials and the System Model

Verifiable Credentials (VCs) are digital representations of conventional credentials, which are designed to be privacy-preserving, secure, and machine-verifiable. VCs and the conceptual system model were originally introduced in the W3C Verifiable Credentials Specification [1]. The system model is consistent across other state-of-the-art systems [7, 8, 16, 17, 32] and literature [23, 26, 27] based on VCs. It consists of the following entities: a set of issuers, a set of holders, a set of verifiers, and a registry:

Issuer - An issuer issues VCs to holders. It also revokes previously issued VCs.

Holder - A holder receives a VC from an issuer. Once the VC is received, the holder would store it. The holder also constructs one or more Verifiable Presentations (VPs) from the VC.

Verifier - A verifier receives VPs from holders. It verifies authenticity, integrity, and revocation status of the VPs.

Registry - The registry stores metadata such as issuers' public keys and proofs to assist the verification of VPs. Some of the possible implementations for a registry are DLTs, centralized servers, etc.

Similar to conventional credentials, a typical VC consists of information about the issuer (e.g., issuer's name), claims (such as hoder's name and age), and proofs (such as digital signatures). While VCs are produced by issuers, Verifiable Presentations (VPs) are constructed by holders and passed to the verifiers for verification. VPs are constructs atop VCs and provide privacy advantages such as selective disclosure of a subset of claims from a VC, as well as aggregation of data from multiple VCs.

3 Related Work

In this section, we show how *zkRevoke* differs from other systems that support revocation of verifiable credentials. Our description focuses on the coverage of requirements as summarized in Table 1 and the reasons explaining why it is not trivial for the systems to satisfy the missing requirements. First, we present the systems that separate expiration period of VCs and verification period of VPs. Subsequently, we discuss other implementations supporting VC revocation. We do not cover protocols designed specifically for X.509 certificates such as Certificate Revocation List (CRL) [33] and Online Certificate Status Protocol (OCSP) [34]. These protocols are not suitable for systems based on verifiable credentials due to differences in design goals and functionalities.

All works except Papathanasiou *et al.* [29] satisfy unobservable verification since they implement registry using DLTs with public read-access. Essentially, proofs needed for verification are stored in DLTs and accessed by verifiers. Therefore, issuers do not observe anything about the verification process.

The following systems in the related work have separate verification period for VPs: (a) IRMA [7, 15, 35], (b) Anoncreds [8, 36], (c) Sitouah *et al.* [18], (d) zk-creds [23], and (e) Xu *et al.* [25]. These systems utilize accumulators [24] and ZKPs [37–39] for the purpose of revocation check. At a high level, all these systems follow a whitelist approach, where identifiers corresponding to valid VCs are inserted in the accumulator during issuance and removed during revocation. To prove non-revocation, a holder generates a witness and computes a ZK proof. The ZK proof proves the inclusion of identifiers in the accumulator. In these systems, the verification period for VPs corresponds to the duration between accumulator updates. Therefore, the verification period is derived rather than controlled.

Due to the witness update problem [36] in accumulators, a holder needs to update the witness to account for the removed identifiers during presentation. Thus, it is not possible for the holder (a) to configure the verification period since a holder needs to update the witness every time the accumulator changes, and (b) to share witnesses that are valid in the future since new witnesses need to be recomputed every time the accumulator changes. Hence, these systems do not satisfy configurable period or one-time VP sharing.

Dynamic Status List [32] is a protocol by EBSI [9] designed to perform untraceable revocation status check. It offers separation between expiration and verification periods. However, it has a confirmed security issue where verifiers can accept fake tokens. It is nontrivial to fix without significant changes to the protocol [40]. Due to this, we do not include the protocol in Table 1.

The work-in-progress paper of [19] introduces a number of requirements overlapping with ours, albeit in a more informal way. Their solution is currently under development.

Next, we discuss the works that do not offer separation between expiration period and verification period. EVOKE [26] proposes a revocation protocol that is based on accumulators. EVOKE does not use ZKPs unlike the other-based accumulator-based works. Holders directly share accumulator witnesses in plain-text. Once a verifier gets access to a VP consisting of a witness, the verifier itself can update the witness and continously monitor the revocation status. Due to the witness update issue [36] in accumulators, this protocol does not satisfy the one-time VP sharing requirement.

Schumm *et al.* [28] and Revocation Bitmap 2022 [16] uses bloom filters to store revoked VCs. A holder shares bloom filter index encoded in its VC with a verifier. The verifier checks index inclusion in the bloom filter to verify non-revocation. The protocol addresess one-time VP sharing since holders are not required to provide any proofs after bloom filter indexes have been shared.

Prevoke [27] proposes a protocol that combines a bloom filter and a merkle tree accumulator [24] to store and verify the revocation status of VCs. A holder shares the bloom filter index and merkle witness with the verifier. This protocol does not address one-time VP sharing since holders need to present updated Merkle witnesses.

To address the problem of untraceability, Revocation Timeframe 2024 [17] extends Revocation Bitmap 2022 [16]. The extension consists of an issuer periodically re-issuing a VC. The issuer is responsible for setting the expiry period, which impacts how long a verifier can track the revocation and how often the issuer needs to re-issue VCs. This technique neither separates the expiration period and verification period nor addresses the configurable period and one-time VP sharing requirements.

Papathanasiou *et al.* [29] uses a hashtable to store revoked VCs. For each VC, a unique identifier based on VRF proofs are inserted in the hashtable. Holders share the identifiers with verifiers. The protocol addresses one-time VP sharing since no other proofs are needed once a VP is shared. It does not address unobservable verification since hashtables are managed by issuers rather than DLTs.

Use of blacklists for revocation has been explored in areas other than verifiable credentials, e.g., for revocable group signatures [41]. It is suggested in [42] that this scheme might be extended to support verifiable credentials. However, neither the original scheme of [41] nor its extension proposal in [42] aim to provide untraceability.

4 Time-limited continuous verification

In existing systems, verifiers use VPs received from the holders to ascertain the status of underlying VCs. For example, in the Workforce Identity [12, 43, 44] use case, VCs prove employment status (refer to Appendix C). When Bob applies for a job, he shares a VP with a potential employer. The latter verifies the revocation status and expiration period to ascertain the validity of Bob's VC.

The main issue in this scenario is that the verfication can be performed at any point in time. This is especially problematic because the verification does not invalidate the VP; it can be repeated indefinitely until the underlying VC expires. Since many VCs are issued for months or even years, while certain cerfiticates such as diploma may not expire at all, this allows the verifier to effectively monitor the revocation status of a VC. For example, in the Workforce Identity use case, the potential employer can continuously monitor whether Bob remains employed in his current position.

To address the problem, we introduce the notion of a verification period for VPs. While a verifier can still perform the verification repeatedly, it can only do it within the verification period, which is typically much shorter than the expiration period of the underlying VC. The core requirement is that after a VP expires, the status of the underlying VC has to be *untraceable* for the verifier.

In order to make time-limited continuous verification practical, the proposed framework includes three additional requirements besides the basic requirement of untraceability:

Configurable verification period: The holder must be able to configure the verification period individually for each VP, even when different VPs are based on the same underlying VC. For example, when Bob applies for a job at two different companies, the verification period might be tailored for each company.

One-time VP sharing: Once shared, the VP contains sufficient information for the verifier to continuously monitor the VC status during the verification period without any repeated interaction with the holder. The initial sharing may take multiple communication rounds so that the requirement is different from non-interactive proof. A realistic deployment scale would include thousands of holders and verifiers, making lack of repeated interaction very desirable from the standpoint of bandwidth consumption, especially because a holder may use a mobile device. Besides, one-time sharing eliminates the need for the holder's device to be always online.

Unobservable verification: An issuer should not be able to observe anything about the verification process. This includes the identity of the holder, the identity of the verifier, or even the fact that verification related to some VC produced by the issuer is taking place. At a basic level, it means that the verifier does not need to contact the issuer for the verification. However, the issuer may potentially be able to learn sensitive information by colluding with the registry, depending on the registry implementation. This is further discussed in Sections 5.3 and 5.5.

The requirement of unobservable verification is important for the privacy of any identity management scheme, not just for continuous verification. Note, however, that it is not entirely orthogonal to the proper requirements of time-limited continuous verification. For example, the requirement of one-time VP sharing may lead to more frequent interactions between verifiers and the registry, thereby imposing further constraints on the registry implementation.

Table 2: Notations

Symbol	Description
vc	a VC
vp	a VP
m	verification period of a VP
$\mathsf{RevList}_{\mathsf{iss}}$	a list of seed values corresponding to revoked VCs managed by an issuer *iss*
$\mathsf{blacklist}_{\mathsf{iss}}$	a list of tokens corr. to revoked VCs
ts_0	intial timestamp
e	current epoch's counter
e_i	the i'th epoch's counter
token_i	a token corresponding to epoch i
dur	the duration of a single epoch
w	private inputs in the ZK circuit
x	public inputs in the ZK circuit
π	a ZK proof
crs	a common reference string of a ZK circuit

It is important to note that the typical challenges of VC management are further compounded by the above requirements. For instance, the requirements imply that the solution would use a variety of cryptographic primitives, yet their overhead must remain practical, even at a large scale of holders and verifiers.

5 The *zkRevoke* system

The goal of *zkRevoke* is to address the requirements presented in Section 4. We follow the blacklist approach, storing tokens corresponding to revoked VCs in a public ledger. The main motivation is providing configurable verification period. However, the blacklist approach also achieves lower bandwidth consumption because the revocation rate tends to be smaller than the total number of issued VCs. In most domains, the revocation rate is very low. For example, in the Web's PKI, only 1% of certificates are revoked [30, 31]. Section 8 discusses the performance aspects in detail.

In our construction, time is divided into a sequence of epochs. All time windows such as the expiration and verification periods are expressed in the number of epochs. The verification period is set by the holder for a VP of an underlying VC.

Each entity performs periodic maintenance at the start of every epoch. The duration of epoch is configured as a global system parameter. In practice, it should be in the order of minutes or hours: significantly longer than communication delays and sufficiently long so as to amortize the cost of periodic maintenance, yet not too long in order to allow for shorter verification periods.

5.1 Scheme

Definition 5.1 (*zkRevoke* Scheme)**:** *The zkRevoke scheme is a tuple of algorithms:* Π *= (Setup, Issuance, Revocation, Refresh, Presentation, Verification):*

- $\mathsf{Setup}\,(1^\lambda) \to (\mathsf{sk}, \mathsf{pk}, \mathsf{crs}, \mathsf{ts}_0, \mathsf{dur}, \mathsf{RevList}_{\mathsf{iss}}, \mathsf{blacklist}_{\mathsf{iss}})$: Given a security parameter, generate and output (a) a secret parameter sk, (b) global public parameters pk and crs, (c) the initial timestamp ts_0, (d) epoch duration dur (e) an empty list $\mathsf{RevList}_{\mathsf{iss}}$ of revoked VCs, and (f) global state $\mathsf{blacklist}_{\mathsf{iss}}$.
- $\mathsf{Issuance}\,(\mathsf{sk}, \mathsf{claims}) \to \mathsf{vc}$: Given (a) a secret parameter sk, and (b) a set claims of claims, output vc, a new VC.
- $\mathsf{Revocation}\,(\mathsf{vc}, \mathsf{RevList}_{\mathsf{iss}}) \to \mathsf{RevList}_{\mathsf{iss}}'$: Given vc as well as $\mathsf{RevList}_{\mathsf{iss}}$, output $\mathsf{RevList}_{\mathsf{iss}}'$ that appends the seed value encoded in vc to the $\mathsf{RevList}_{\mathsf{iss}}$.
- $\mathsf{Refresh}\,(\mathsf{e}, \mathsf{RevList}_{\mathsf{iss}}) \to \mathsf{blacklist}_{\mathsf{iss}}$: Given (a) the current epoch counter value e, and (b) $\mathsf{RevList}_{\mathsf{iss}}$, output $\mathsf{blacklist}_{\mathsf{iss}}$, a list of tokens corresponding to seed values in $\mathsf{RevList}_{\mathsf{iss}}$.
- $\mathsf{Presentation}\,(\mathsf{pk}, \mathsf{crs}, \mathsf{e}, \mathsf{vc}, \mathsf{m}, \mathsf{challenge}) \to \mathsf{vp}$: Given (a) pk and crs, (b) e, (c) vc, (d) verification period m in number of epochs, and (e) challenge, which is a random number $\in \{0,1\}^*$, output vp.
- $\mathsf{Verification}\,(\mathsf{pk}, \mathsf{crs}, \mathsf{e}, \mathsf{blacklist}_{\mathsf{iss}}, \mathsf{m}, \mathsf{vp}, \mathsf{challenge}) \to \{0,1\}$: Given these parameters, output 1 if vp is valid in e and the corresponding VC is neither revoked nor expired. The vp is valid in e if it is correctly formed and e is within the verification period. Otherwise output 0.

zkRevoke extends the standard operations on VCs in the following ways: First, *zkRevoke* includes parameters that affect the validity period of VPs. Second, *zkRevoke* introduces the Refresh operation, which is used to support time-limited verification. Note also that, in *zkRevoke*, the Verification operation may be called in a different epoch e than the epoch e′ in which the VP was generated. The verifier may even decide to perform Verification repeatedly in different epochs within the verification period of the VP.

The implementation presented in Section 5.6 does not include details of advanced functionality such as selective disclosure, as the primary focus is on untraceability-related features. Instead, selective disclosure is presented as an extension in Section 5.7.

Table 2 lists down notation used throughout the paper.

5.2 Formal requirements

A VC scheme should satisfy the three properties of Completness, Soundness, and Untraceability. We now provide a formal definition of these properties. To the best of our knowledge, this is the first such definition in the literature.

Definition 5.2 (**Completeness**)**:** *Given a VP valid in epoch* e *generated from a non-revoked VC that is not expired in epoch* e*, the verification of VP in epoch* e *should result in success.*

Let $(\mathsf{sk}, \mathsf{pk}, \mathsf{crs}, \mathsf{ts}_0, \mathsf{dur}, \mathsf{RevList}_{\mathsf{iss}}, \mathsf{blacklist}_{\mathsf{iss}}) \leftarrow \mathsf{Setup}(1^\lambda)$. Let claims be any valid set of claims, e be any valid epoch, and challenge be any valid challenge. Let e′ and m be such that $\mathsf{e}' \le \mathsf{e} < (\mathsf{e}' + \mathsf{m})$. Let $\mathsf{blacklist}'_{\mathsf{iss}}$ be the set of revoked tokens at e. Let C_e be the set of non-expired and non-revoked VCs at e. A VC scheme satisfies Completeness if the following equality holds for any $\mathsf{vc} \in C_e$, and a VP generated from vc as $\mathsf{vp} \leftarrow \mathsf{Presentation}(\mathsf{pk}, \mathsf{crs}, \mathsf{e}', \mathsf{vc}, \mathsf{m}, \mathsf{challenge})$:

$$\Pr[\mathsf{Verification}(\mathsf{pk}, \mathsf{crs}, \mathsf{e}, \mathsf{blacklist}'_{\mathsf{iss}}, \mathsf{m}, \mathsf{vp}, \mathsf{challenge}) = 1] \ge 1 - \mathsf{negl}(\lambda)$$

Definition 5.3 (**Soundness**)**:** *Given a VP which is either not valid in epoch* e *or created using a revoked or expired VC in epoch* e*, the verification of VP in epoch* e *should returns* 0*.*

Let $(\mathsf{sk}, \mathsf{pk}, \mathsf{crs}, \mathsf{ts}_0, \mathsf{dur}, \mathsf{RevList}_{\mathsf{iss}}, \mathsf{blacklist}_{\mathsf{iss}}) \leftarrow \mathsf{Setup}(1^\lambda)$. Let claims be any valid set of claims, e be any valid epoch, and challenge be any valid challenge. Let e' and m be such that $\mathsf{e}' \leq \mathsf{e} < (\mathsf{e}' + \mathsf{m})$. Let C_e be the set of non-expired and non-revoked VCs at e. Let $\mathsf{blacklist}'_{\mathsf{iss}}$ be the set of revoked tokens at e. Let V_e be the set of VPs valid in epoch e generated from VCs in C_e with the use of pk, crs, m, challenge. A VC scheme satisfies Soundness if for any $\mathsf{vp} \notin V_e$ or $\mathsf{vp} \leftarrow \mathsf{Presentation}(\mathsf{pk}, \mathsf{crs}, \mathsf{e}', \mathsf{vc}, \mathsf{m}, \mathsf{challenge})$ where $\mathsf{vc} \notin C_e$, the following equality holds:

$$\Pr[\mathsf{Verification}(\mathsf{pk}, \mathsf{crs}, \mathsf{e}, \mathsf{blacklist}'_{\mathsf{iss}}, \mathsf{m}, \mathsf{vp}, \mathsf{challenge}) = 0] \geq 1 - \mathsf{negl}(\lambda)$$

Definition 5.4 (Untraceability): *Given a VP with verification period m, the verifier should not be able to learn the revocation status of VC corresponding to the VP once m is over.*

We formalize untraceability as a game where an adversary $\mathcal{A}$ tries to guess the revocation status of a VC. The advantage of $\mathcal{A}$ in the game ($\mathbf{Game}_{\mathcal{A}}^{untrace}$) (refer to Appendix D) should be negligible:

$$|\Pr[\mathbf{Game}_{\mathcal{A}}^{untrace}(1^\lambda) = 1] - 1/2| \leq \mathsf{negl}(\lambda)$$

5.3 Adversarial model

We assume that issuers generate VCs according to the protocol, but may attempt to glean information about the verification process as an attack on *unobservable verification*. In other words, issuers are honest but curious. They do not leak information about holders to verifiers.

Similarly, the registry follows the protocol. For the sake of *unobservable verification*, we assume that the information stored in the registry is public and read accesses to the registry are not authenticated. Furthermore, the registry should not be able to discern which VC is being verified. We briefly describe the implementation of such registry in Section 5.5.

We assume that holders follow the protocol with the exception that they may generate fake VPs and proofs during presentation in order to pass the verification by verifiers. For example, the fake VPs may be based on expired, revoked or non-existent VCs.

Regarding verifiers, we assume that they are trusted to perform the verification operation correctly, but not to preserve the privacy of holders. Not only can they attempt to passively extract information from the VPs and the registry, but they can also actively perform a replay attack.

To summarize, we consider the entities follow the protocol with the following exceptions: (a) an issuer may try to break *unobservable verification*, (b) a holder might share fake proofs, (c) a verifier may violate the privacy of holders, and (d) a verifier might also impersonate holders. This model is similar to, but less benign than the one in W3C Verifiable Credentials Specification [1], which considers registry and issuers to be fully trusted entities.

5.4 Additional assumptions

Expiration periods for VCs are set by issuers, whereas verification periods for VPs are set by holders. Since verification is performed by verifiers, this requires clock syncronization. Every entity in the system maintains an epoch counter, which is incremented when the entity transitions to the next epoch. In practice, every entity synchronizes its clock with synchronization servers such as NTP. This way, deviations are within seconds, and are thus much smaller than the epoch duration. We assume that permanent clock failures are detected and repaired. However, transient lack of synchronization or communication delays may result in a discrepancy of one epoch between the counters maintained by different entities, leading to a failed verification for a valid VP. The protocols presented in our solution do not explicitly show how to handle such a situation for the sake of more focused presentation. Note that it is easy for holders to overcome such scenarios by creating a new VP and sharing it with the verifier again, because the clocks and epoch counters are likely to get synchronized in the meantime.

For the sake of simplified presentation, we assume an issuer uses only one pair (pk, sk) of public/secret keys for signing VCs, in line with how it is done in the state-of-the-art [26, 27, 45]. It is trivial to extend the presented protocol to support multiple key pairs. For example, a public key can be encoded in the VC and shared in the VP. The set of public keys would be published in the registry.

Table 3: ZK circuit inputs

Private inputs (w)	sig, seed, nonce
Public inputs (x)	pk, H(claims), h, challenge, e, exp, token

5.5 Building Blocks

Before presenting our *zkRevoke* construction, we describe **tokens** and **ZK circuit**, the integral building blocks of the system.

Tokens: The system produces tokens to identify a VC within a given epoch. The tokens are refreshed at the beginning of each epoch. A token for an epoch with counter e is produced by computing a hash digest $\mathsf{token} \leftarrow \mathsf{H}(\mathsf{seed}, \mathsf{e})$ of two inputs, where seed is a secret value assigned to each VC by the issuer. This way, with knowledge of the secret seed, it is easy to compute future tokens. If a verifier knows a VC token for a specific epoch, it will not be able to calculate the next epoch token corresponding to the same VC, which lays the foundation for *untraceability*. Collision resistance of the hash function ensures uniqueness of tokens within one epoch. A holder will be able to prove validity and ownership of a token by employing a ZK circuit that proves knowledge of the seed without disclosing it.

ZK circuit: The specific ZK circuit we use in our construction allows each holder to prove the correctness of his/her tokens without revealing the seed. The private and public inputs to our ZK circuit are presented in Table 3.

The ZK circuit asserts the following three conditions:

(1) $\mathsf{Verify}(\mathsf{pk}, \mathsf{sig}, \mathsf{H}(\mathsf{seed}, \mathsf{H}(\mathsf{claims}), \mathsf{exp})) = 1$
(2) $\mathsf{H}(\mathsf{seed}, \mathsf{e}) = \mathsf{token}$
(3) $\mathsf{H}(\mathsf{challenge}, \mathsf{nonce}) = \mathsf{h}$

The first condition verifies the signature (sig) on (a) seed, (b) H (claims), and (c) exp. These fields are included in a VC. Since sig and seed are private inputs, they are never shared with verifiers. Therefore, this condition enables holders to prove correctness of signature without revealing the signature or any of the underlying fields. Thus, this condition binds the tokens with claims.

Algorithm 1 *zkRevoke*: Setup

```
1:  procedure SETUP(1^λ)
2:      ts_0 ← time()                                  ▷ initial timestamp
3:      choose epoch duration (dur)
4:      (sk, pk) ← KeyGen(1^λ)
5:      Let circuit be a ZK circuit asserting:
6:          Verify(pk, sig, H(seed, H(claims), exp)) = 1 ∧
7:          H(seed, e) = token ∧
8:          H(challenge, nonce) = h
9:      crs ← G16.Setup(1^λ, circuit)
10:     RevList_iss ← φ                                ▷ stores revoked VCs
11:     blacklist_iss ← φ
12:     output (sk, pk, crs, ts_0, dur, RevList_iss, blacklist_iss)
```

Algorithm 2 *zkRevoke*: Issuance

```
1:  procedure ISSUANCE(sk, claims)
2:      seed ← {0,1}^λ                  ▷ cryptographically secure value
3:      choose exp
4:      sig ← Sign(sk, H(seed, H(claims), exp))
5:      vc ← (seed, claims, exp, sig)
6:      output vc
```

Algorithm 3 *zkRevoke*: Revocation

```
1:  procedure REVOCATION(vc, RevList_iss)
2:      RevList_iss' ← RevList_iss ∪ vc.seed
3:      output RevList_iss'
```

The second condition verifies that the token is computed correctly. Since seed is a private input, holders prove correctness of the token without revealing the seed.

The third condition is used to bind each VP to both the holder and the verifier. The value for challenge is provided by the verifier, whereas the holder generates a random nonce and computes a hash digest. If this condition is not included, then a malicious verifier could reuse a VP to impersonate the holder elsewhere, e.g., by presenting it to another service (*replay attack*). To prevent this, the VP must be bound to both the holder and the verifier.

Our *zkRevoke* construction uses ZK circuit to create a ZK proof for tokens during the creation of a VP by a holder. Later, the verifier uses the ZK circuit to verify the ZK proof.

Registry: The registry is implemented via a DLT, as in [8] or [46]. Since this is a standard off-the-shelf component, we only describe what information is stored in the registry at what granularity and we elaborate on the interactions between the registry and the other entities, i.e., read and write accesses. In particular, it is used to store a record for every issuer that can be retrieved using an issuer's index, e.g., the public key. To this end, the registry allows for authenticated writes. In addition, the registry provides public unauthenticated read access, as mentioned in Section 5.3. Furthermore, the DLT nodes and other entities should not be able to monitor what data a verifier reads from the registry. This can be achieved, for example, when the verifier runs its own DLT node.

5.6 Algorithms

There are six algorithms in *zkRevoke*: 1) setup, 2) issuance, 3) revocation, 4) refresh, 5) presentation, and 6) verification. The pseudo-code is presented in Algorithms 1 to 6.

Setup. This procedure is used by each issuer to bootstrap the required structures. The input λ is the security parameter in our protocol.

First, in L.2-3, the issuer records the initial timestemp ts_0 and initiates epoch duration dur. The issuer also generates a cryptographic key pair (sk, pk) for the digital signature scheme to be used in the ZK circuit (L.4).

Subsequently, the issuer defines a ZK circuit to verify the ownership of tokens and computes a common reference string crs for the circuit. Our construction employs the Groth16 [38] ZKP scheme (L.5-9), which requires a trusted setup phase to generate the common reference string (crs). Trusting the issuer to perform the setup aligns with the trust model of the W3C Verifiable Credentials Specification and related work on VCs. While transparent ZKP schemes without trusted setup exist, they impose significant performance penalties in proof size and verification time [47]. Given that our primary design goal is practical deployability and that the trusted setup is a one-time operation per issuer, we find this trade-off justified for current deployments.

Then, the issuer initializes two empty lists, $\mathsf{RevList}_{\mathsf{iss}}$ to locally store revoked VCs, and $\mathsf{blacklist}_{\mathsf{iss}}$, the global blacklist (L.10-11).

All chosen parameters, keys, and data structures are stored by the issuer. $\mathsf{blacklist}_{\mathsf{iss}}$, crs, ts_0, dur, and pk are published to the registry. These values are used by verifiers for verification of VPs.

Issuance. This procedure is used by an issuer to generate a new VC for a holder. It takes the issuer's secret key sk and a set claims of claims as inputs. Depending on the use case, the set of claims can be provided either by the issuer or by the holder or collectively by both of them.

To issue a new VC, first, the issuer samples a λ-bit seed uniformly. Then, the issuer determines the expiration period denoted as exp for the new VC. The issuer encodes these values in the VC. In addition, the issuer can include a unique identifier *id* for the VC. We do not mandate including *id* in a VC since this field is optional in the W3C Verifiable Credentials Specification.

Signing raw data would increase the size of the ZK circuit leading to reduced performance. Therefore, the signature is performed on a hash digest to keep the size of the messages fixed. The issuer computes a hash digest of the following data fields and generates a digital signature (sig): (a) seed, (b) H(claims), and (c) exp. Finally, a new VC vc is created; it consists of the signed data fields along with the signature. Finally, the procedure outputs the vc.

Once the new VC vc is generated, the issuer sends vc to the corresponding holder along with crs, pk, ts_0, and dur. The holder will use these values to generate proofs while creating a VP (Algorithm 5).

Revocation. This procedure is used by an issuer to revoke a previously issued VC. When revoking a given VC, the issuer appends the corresponding seed to $\mathsf{RevList}_{\mathsf{iss}}$, a list that stores the seeds of revoked VCs. The procedure then outputs the updated list $\mathsf{RevList}_{\mathsf{iss}}'$. The issuer stores the updated list locally.

Algorithm 4 *zkRevoke*: Refresh

procedure REFRESH (e, $\text{RevList}_{\text{iss}}$)
  $\text{blacklist}_{\text{iss}} \leftarrow \phi$
  for seed $\in \text{RevList}_{\text{iss}}$ **do**
    token $\leftarrow$ H(seed, e) ▷ updates global state
    $\text{blacklist}_{\text{iss}} \leftarrow \text{blacklist}_{\text{iss}} \cup$ token
  output $\text{blacklist}_{\text{iss}}$

While revocation is most commonly used when the holder's status changes, there could be additional technical reasons. In practice, it is essential to revoke a VC if the underlying seed is leaked because adversaries may impersonate the holder by generating ZK proofs using the seed.

Refresh. This procedure is used by an issuer to periodically refresh tokens corresponding to revoked VCs. It is executed every time a new epoch starts. At the start of a new epoch, assuming the local time of current_time, the issuer first computes the current epoch value (e) as follows:

$$\text{e} \leftarrow (\text{current\_time} - \text{ts}_0)/\text{dur} \tag{1}$$

Then, the issuer uses the Refresh procedure to compute tokens corresponding to revoked VCs. After that, the issuer sends the tokens to the registry, updating the global blacklist $\text{blacklist}_{\text{iss}}$.

Before starting the procedure, the issuer is expected to clean up seeds corresponding to expired VCs from $\text{RevList}_{\text{iss}}$.

We note that revocation of VCs take effect only after a Refresh operation is performed, since during Refresh, tokens corresponding to seeds added to $\text{RevList}_{\text{iss}}$ are published in the $\text{blacklist}_{\text{iss}}$. The issuer must perform Refresh at the start of each epoch.

Presentation. A holder uses this procedure to generate a VP that will be sent to a verifier. The procedure takes the following inputs: (a) pk, (b) crs, (c) e, (d) vc, (e) m, and (f) challenge. Parameter vc denotes the holder's VC issued by the issuer. The holder receives pk and crs from the issuer alongside the vc.

Parameter m specifies the verification period of the new VP vp in terms of the number of epochs starting from the current e. It translates into the number of tokens that the holder wants to include in vp.

Before generating vp, the holder receives challenge from the verifier. The parameter challenge is used to prevent replay attacks on the proofs in vp.

The Presentation procedure is shown in Algorithm 5. First, the holder computes the current epoch value using Equation 1. Subsequently, the holder generates a random nonce. Then, it computes a hash digest h of two inputs: challenge provided by the verifier and nonce.

The holder generates m tokens, which act as pseudonyms representing the holder for m consecutive epochs starting from the current one. Since e acts as a counter, the holder can simply increment e by 1 to compute the next epoch index. In addition to tokens, the holder generates ZK proofs π_{zkp} to prove the correctness of tokens. For each token_i, the holder generates a Zero-Knowledge Proof π_i using private and public inputs. The private (w) and public (x) inputs are described in Table 3. π_i proves that "The VC encodes a signed seed using which the provided token_i is generated." We

Algorithm 5 *zkRevoke*: Presentation

procedure PRESENTATION (pk, crs, e, vc, m, challenge)
  nonce $\leftarrow \{0,1\}^{\lambda}$
  h $\leftarrow$ H(challenge, nonce)
  epochs $\leftarrow \phi$
  tokens $\leftarrow \phi$
  $\pi_{\text{zkp}} \leftarrow \phi$
  for $i \in 1, \cdots,$ m **do**
    $\text{e}_i \leftarrow$ e
    $\text{token}_i \leftarrow$ H(vc.seed, e_i)
    w $\leftarrow$ (vc.seed, vc.sig, nonce)
    $x \leftarrow$ (pk, token_i, e_i, challenge, h, vc.exp, vc.claims)
    $\pi_i \leftarrow$ G16.Prove (x, crs, w)
    $\pi_{\text{zkp}} \leftarrow \pi_{\text{zkp}} \cup \pi_i$
    epochs $\leftarrow$ epochs $\cup$ e_i
    tokens $\leftarrow$ tokens $\cup$ token_i
    e $\leftarrow$ e +1
  vp $\leftarrow$ (tokens, epochs, m, h, vc.claims, vc.exp, π_{zkp})
  output vp

Algorithm 6 *zkRevoke*: Verification

procedure VERIFICATION (pk, crs, e, $\text{blacklist}_{\text{iss}}$, m, vp, challenge)
  if VERIFYPROOFS(pk, crs, m, vp, challenge) $\neq 0$ **then**
    output VERIFYREVOCATIONSTATUS(e, vp, $\text{blacklist}_{\text{iss}}$)
  output 0

procedure VERIFYPROOFS (pk, crs, m, vp, challenge)
  for $i \in 1, \cdots,$ m **do**
    $\text{e}_i \leftarrow$ extract the counter from vp.epochs corr. to i
    $\text{token}_i \leftarrow$ extract the token from vp.tokens corr. to i
    $\pi_i \leftarrow$ extract the π from vp.π_{zkp} corr. to i
    $x \leftarrow$ (pk, token_i, e_i, challenge, vp.h, vp.vc.exp, H(vp.vc.claims))
    if G16.Verify (x, crs, π_i) = 0 **then**
      output 0
  output 1

procedure VERIFYREVOCATIONSTATUS (e, vp, $\text{blacklist}_{\text{iss}}$)
  if e $\notin$ vp.epochs $\wedge$ e > vp.vc.exp **then**
    output 0
  token $\leftarrow$ extract the token from vp.tokens corr. to e
  if token $\in \text{blacklist}_{\text{iss}}$ **then**
    output 0
  output 1

use π_i to bind vp to both the holder and the verifier, see the third condition for the ZK circuit in Section 5.5. The proofs are accumulated into π_{zkp}. Using ZK proofs allows the holder to prove the ownership of tokens without revealing the underlying seed.

Finally, the holder constructs vp consisting of: (a) m tokens, (b) m epochs, (c) m, (d) h, (e) vc.claims, (f) vc.exp, and (g) π_{zkp}. After vp is generated, the holder may share it with a verifier. The holder also shares identifying information of the issuer that allows the verifier to extract the issuer's record from the registry.

Verification. The input parameter challenge to the verification procedure should have the same value that was previously sent by the verifier to the holder. First, the verifier retrieves the issuer's record from the registry including the public key pk, the common

reference string crs, and the global blacklist $\text{blacklist}_{\text{iss}}$. Then, the verifier computes the current epoch value using Equation 1.

Afterwards, the verifier performs the steps shown in Algorithm 6. *Verification* includes two procedures: *VerifyProofs* and *VerifyRevocationStatus*. *VerifyProofs* checks the correctness of proofs included in vp. The proofs are verified for all epochs included in vp. For each proof, the verification checks that the claims and the secret seed have been signed by the issuer, and that the token is computed correctly. The procedure returns 1 if all proofs are valid, and 0 otherwise. *VerifyProofs* checks all proofs included in vp, not just the proof belonging to the current epoch, to ensure they are not malformed.

In contrast, *VerifyRevocationStatus* checks the revocation status of vp only in the current epoch using the current blacklist $\text{blacklist}_{\text{iss}}$ retrieved from the registry. The procedure also checks that vc underlying shared vp is not expired.

To explain the assymetry between *VerifyProofs* and *VerifyRevocationStatus*, recall the verification semantics for VPs: the verification fails, if vp contains a malformed or incorrect proof for any epoch, or if vp is invalid in the current epoch. As an optimization, if the verifier decides to call *Verification* on same vp repeatedly for different epochs, it is sufficient to invoke *VerifyProofs* only the first time, whereas *VerifyRevocationStatus* needs to be called each time using the current values of e and $\text{blacklist}_{\text{iss}}$.

5.7 Extensions

So far, we have been presented the core functionality of *zkRevoke* that focuses on implementing untraceability. In this section, we provide five extensions to the core functionality: (a) provision for selective trust in issuers, (b) support for selective disclosure of claims, (c) support for unlinkability, (d) a single ZK proof covering multiple tokens, and (e) support for list commitment.

5.7.1 Selective Trust in Issuers. In practice, verifiers have the flexibility to choose which issuers to trust based on their individual credibility [1, 48]. The *Verification* function may reject a VP based on a VC from an untrusted issuer. This trust is determined at a policy-level. For example, a potential employer (verifier) might decide to accept workforce identity VCs issued by organizations (issuers) known only to the employer. This is orthogonal to the core functionality of time-limited verification, though it requires specific straightforward adjustments to the scheme interface and definitions in Section 5.1, as well as to the implementation of *Verification*.

5.7.2 Selective Disclosure. Selective disclosure means that only a subset of claims included into a VC is transferred into a VP. While not directly related to our objectives, it is one of the key requirements in many VC management systems, as observed in [49]. Our approach employs a mechanism similar to the one used in IOTA Identity: Revocation Bitmap 2022 [16]. Each holder can choose which claims to reveal and which to hide. It hides a claim by only revealing the hash digest of that claim. For example, if a VC includes a name and a date-of-birth, a VP can include the date-of-birth and the hash digest of the name.

Thus, VPs include hash digest of claims instead of claims themselves. Similarly, the signature is computed on the hash digest of individual claims rather than on all claims collectively. When sharing the digest with a verifier, the holder additionally sends the claims selected for disclosure and the index of those claims in the hash digest. The verifier can verify the correctness of a claim by computing its hash digest and matching the computed digest against the provided one.

Since digital signatures are verified on the ZK circuit, the verification function is updated to verify the signature of the hash digest of claims instead of the claims themselves. The public input of $\mathsf{H}(\text{claims})$ should be replaced with the hash digest of individual claims.

5.7.3 Unlinkability. A number of systems [7, 8] and papers [23] have considered an auxiliary design goal of unlinkability. Unlinkability is defined in W3C Verifiable Credentials Specification [50] as the property ensuring that colluding verifiers cannot correlate the VPs they receive.

While unlikability is not a design goal of our work, *zkRevoke* can be extended to provide a limited form of unlinkability. In particular, we extend the threat model described in Section 5.3 and allow verifiers receiving VPs from holders to collude with each other in order to link VPs. We consider two scenarios: (a) VPs are generated from the same VC without any overlapping verification period, and (b) VPs are generated from the same VC with overlapping verification period.

To address the scenario of non-overlapping verification period, the selective disclosure extension described above should be updated so that no identifying information is shared with verifiers. In the current extension, hash digests of all claims are shared with verifiers, which could be used to correlate multiple VPs. This could be addressed by randomizing the hash digests shared with the verifiers by a nonce, and proving the correctness of these digests in the ZKP. Once no correlatable information is shared, *zkRevoke* would provide unlinkability when multiple VPs from the same VC do not contain overlapping verification period.

This approach can be further extended to address the scenario of overlapping verification period. To this end, multiple seed values should be assigned to a VC in order to create different time-bound tokens for that VC within an epoch. A holder would share a distinct token with each verifier such that the corresponding VPs will be unlinkable. The cost of this approach is that the blacklist size would increase proportionally to the number of seeds per VC.

5.7.4 Proving Multiple Tokens using a Single ZK Proof. In the algorithmic description above, a separate ZK proof is used for each token. Below, we provide an optimization that allows a holder to prove multiple tokens using a single ZK proof. During the setup procedure, before generating the circuit, an issuer has to choose k that specifies the number of tokens that can be verified in the ZK circuit. The issuer needs to store k in the registry so that verifiers can fetch it during verification.

The ZK circuit should be updated as follows: the condition of $\mathsf{H}(\text{seed}, \mathsf{e}) = \text{token}$ in the ZK circuit defined in Section 5.5 is replaced with the following condition: $\mathsf{H}(\text{seed}, \mathsf{e}_i) = \text{token}_i \mid \forall i \in 1 \cdots k$. The public input e in the ZK circuit is replaced with $\mathsf{e}_i \mid \forall i \in 1 \cdots k$. The public input token is replaced with $\text{token}_i \mid \forall i \in 1 \cdots k$.

The holder generates tokens in blocks of k because the ZK circuit requires exactly k tokens. If m is not a multiple of k, then the last token (token_m) is duplicated to fill the remaining spaces in the

last block. For each block of k tokens, the holder generates a Zero-Knowledge Proof π_i using private and public inputs. The latter proves the following statement: The VC encodes a signed seed using which the provided tokens are generated.

However, this optimization has three challenges. First, before generating a ZK circuit, the value of k needs to be fixed. Therefore, an issuer needs to select a single value of k that would be used for all VCs generated by that issuer. Second, increasing the value of k directly results in increased ZK circuit size, which affects the bandwidth of issuers, holders, and verifiers. Third, the circuit generation time would also be affected by increasing the value of k. Section 8.5 experimentally evaluates the impact of k and discusses the results.

5.7.5 List Commitment. DLT technologies such as Ethereum are known to have high storage cost. If used for implementing the registry, it will affect the scalability of the system, since the size of the blacklist grows linearly with the number of revoked VCs. To avoid this limitation, an issuer can use a hash-based commitment scheme [51] to create a commitment for the blacklist and store the produced commitment in the registry, while storing the complete list using a cheaper off-chain solution, such as a database. During verification, verifiers need to fetch the list and ascertain its integrity using the commitment for each epoch in addition to verifying non-inclusion of tokens for VPs. We experimentally evaluate the performance benefits of using a list commitment in Section 8.4.

6 Security Analysis

We prove that *zkRevoke* satisfies completeness, soundness and untraceability properties even in the presence of a computationally-bounded adversary $\mathcal{A}$. The proofs are given in Appendix E.

Theorem 6.1: *If the ZKP system is Complete and the hash function is collision-resistant, then* Π *= (Setup, Issuance, Revocation, Refresh, Presentation, Verification) satisfies the Completeness property.*

Theorem 6.2: *If the ZKP system is sound and Zero-Knowledge, the digital signature scheme is Unforgeable, and the hash function is pre-image resistant, then* Π *= (Setup, Issuance, Revocation, Refresh, Presentation, Verification) satisfies the Soundness property.*

Theorem 6.3: *If the ZKP system is Zero-Knowledge and the hash function is pre-image resistant, then* Π *= (Setup, Issuance, Revocation, Refresh, Presentation, Verification) satisfies the Untraceability property.*

7 Discussion

In this section, we discuss how *zkRevoke* satisfies additional requirements other than the three analyzed in Section 6.

Configurable Verification Period. Before sharing a VP with a verifier, a holder and the verifier determine m, the verification period of the VP expressed in the number of epochs. The holder generates a VP consisting of m tokens corresponding to m epochs starting from the current one. Therefore, configurability of verification period is provided by the construction.

It is also possible for holders to include tokens for specific epochs instead of including a sequence of m tokens starting from the current epoch e. To do this, the Presentation and Verification algorithms need to be modified. First, in the Presentation, the input parameter m should be replaced with specific e values. Then, the corresponding tokens should be generated instead of generating a sequence of m tokens. Finally, the VP should include e values instead of m. In the Verification procedure, a verifier needs to verify the correctness of the included e values.

One-time VP sharing. A holder includes one or more ZK proofs (π_{zkp}) in a VP. π_{zkp} is sufficient for a verifier to ascertain the validity of tokens included in the VP. Therefore, further interactions are not required for continuous verification in our protocol.

Unobservable Verification. We provide an informal intuition for how *zkRevoke* achieves unobservable verification. An issuer publishes the revocation list and other public parameters in the registry, as explained in Section 5.5. The verifier retrieves this information from the registry prior to VP verification. The registry provides public read access, and no permissions are required to access the information. In addition, no VC identifiers are stored in the registry. Therefore, the issuer does not learn about the verification process.

8 Evaluation

In this section, we evaluate *zkRevoke* against related work. We implement issuer, holder, and verifier using Golang for the purpose of experimental evaluation. The source code is available at https://github.com/praveensankar/zkRevoke/. Additional implementation details are provided in Appendix A.

Our main evaluation objectives are listed below; they reflect the key differences between *zkRevoke* and related work.

- The impact of providing configurable period and one-time VP sharing on performance is considered in Section 8.3.
- The performance tradeoffs of following a blacklist approach are assessed in Section 8.4.
- The benefits and costs of using general-purpose ZKPs compared to custom-built ones are evaluated in Section 8.5.
- The performance breakdown for our custom ZK curcuit is presented in Section 8.6.

We measure bandwidth and computation requirements on different entities for sharing and verifying VPs, and for refreshing the blacklist every epoch. These operations are most relevant to revocation protocols. We also evaluate the storage requirements at the registry and the impact of proving multiple tokens using a single ZK proof, as presented in Section 5.7. For each objective, we focus on the metrics and entities that are most impacted.

Hardware. The hardware choice only impacts computation times in the experiments, without affecting storage and bandwidth consumption. Regarding computation times, our main focus is on the holder. Since holders are likely to use commodity machines, we evaluate *zkRevoke* on a desktop computer running a 12-Core Apple M4 Pro CPU with 24GB RAM. Using a more powerful server would affect the absolute values, but would be unlikely to significantly influence comparative evaluation relative to the baselines.

8.1 Workload

We consider the use case of Workforce Identity (refer to Appendix C). We are not aware of any workload analysis for this use case. Therefore, we adopt the scale and parameters from published works on similar use cases, and complement it with sensitivity analysis.

We consider the scale n of issued VCs to be one million in total, similar to EVOKE [26]. The expiration period of VCs is set to 365 days: We assume an employer would update the identities of employees after each financial year for reasons such as salary increase, promotion, etc. The epoch duration is set to one day. Such refresh duration is common in existing systems, such as Anoncreds. In IRMA [7], holders fetch updated witnesses from issuer every day.

Similarly to the evaluation in EVOKE [26] and Sitouah et al [18], we only consider revocation of VCs in the workload, without expiration. We evaluate revocation rate $\mathcal{R}$ from 1% to 15%. This range is very close to the one considered by EVOKE [26], which is based on studies that analyze PKI certificate revocation. In experiments, we assume a fixed number r of VCs are revoked per epoch. For example, with 1 million VCs, expiration period of 356 epochs (days), and $\mathcal{R} = 1\%$, results in revoking $r \approx 27$ VCs each epoch.

For credentials shared during a hiring process, we assume that verification periods between one day and two months are reasonable, depending on the hiring process. Thus, we vary the verification period m from 1 to 60 epochs.

When evaluating the extension of using a single ZK proof for k tokens, we consider the values of k from 1 to 2^5, since values of k larger than m create overhead without benefits.

8.2 Baseline Solution

We opt for IRMA [7, 15] as a baseline due to the following reasons:

- The guarantees listed in Table 1 and Section 4 and provided by *zkRevoke* come at a performance cost due to the use of complex techniques. To allow for a fair comparison, a system with similar guarantees needs to be selected as a baseline. Since no state-of-the-art system provides all listed guarantees, we choose IRMA that implements separation separation between expiration and verification periods alongside a form of untraceability.
- IRMA is a popular system deployed in real-world use cases. Furthermore, it provides well-documented open-source libraries.
- IRMA is a good match for our evaluation objectives. In contrast to *zkRevoke*, it does not have one-time sharing (Objective 1), it is based on a whitelist (Objective 2), and it uses custom-built ZKPs (Objective 3).

To achieve similar performance configuration in IRMA compared to *zkRevoke*, we let the issuer update the accumulator once every epoch, removing factors from revoked VCs. In IRMA, when the accumulator is updated, the issuer broadcasts all revoked factors to the holders, including both newly and older revoked factors. This allows holders to receive the factors also after offline periods. To account for different deployments, we also investigate two additional settings, *no repetition* and *registry*. In IRMA-*no repetition*, every revoked factor is broadcast once to all holders. In IRMA-*registry*, instead of broadcasting, revoked factors are published in the registry, making them available to holders. This variant significantly differs from IRMA, which does not use DLTs, but is similar to approaches in other accumulator-based protocols, such as Anoncreds [8], and Sitouah et al [18].

8.3 One-time Sharing

Our first objective is to measure the benefits of the one-time sharing feature. We compare the bandwidth and computation requirements of holders, since the one-time sharing feature requires holders to compute and share tokens and proofs corresponding to m epochs provided in the verification period. To understand the results, it is important to note that we compare the overhead of sharing a single VP with verification period m in *zkRevoke* with the overhead of repeatedly sharing proofs in IRMA, during m consecutive epochs. In addition to repeatedly computing and sharing the VP in IRMA, the holder also needs to receive a witness update from an issuer, and update the proof accordingly.

Metric 1: Holder's Bandwidth. In *zkRevoke*, we measure the total bytes for ZK proofs sent by the holder. In IRMA, we measure the total bytes of witness update messages a holder receives from an issuer during the validity period, in addition to the bytes for proofs sent to the verifier.

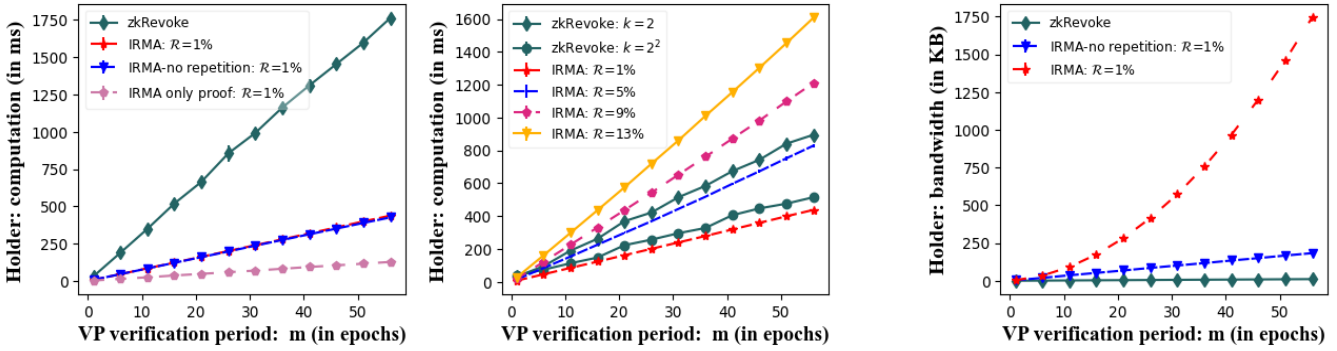
Figure 1c illustrates the bandwidth requirements for holders. In *zkRevoke* and IRMA-no repetition, the bandwidth requirement is linear. *zkRevoke* outperforms IRMA by a huge margin. For example, when m is configured to 26 epochs, the required bandwidth in IRMA-*no repetition* is around 85 KB, whereas it is 4 KB in *zkRevoke*. In IRMA with broadcast everything, the bandwidth is 415 KB. The bandwidth is further increased when increasing the revocation rate in IRMA. IRMA-*registry* variant is not shown, but values will be the same as the ones used in IRMA-*no repetition*, since a holder can fetch the same amount of proofs directly from the registry instead of receiving it from an issuer.

Metric 2: Holder's Computation. In *zkRevoke*, we evaluate the total time a holder requires to generate ZK proofs. For IRMA, where the holder performs witness updates and recomputes proofs at each epoch, we report the total time required for m epochs.

Figure 1a shows that IRMA outperforms *zkRevoke* for small revocation rate $\mathcal{R}$. For example, when $\mathsf{m} = 1$, a holder takes 8 milliseconds in IRMA, whereas in *zkRevoke*, a holder takes 32 milliseconds: the computation time in IRMA is four times shorter. The results for computing only proofs show that approximately 70% of the computation done in IRMA is due to witness updates. The figure also illustrates that the computation cost for holders in IRMA-*no repetition* is the same as in IRMA, since the difference does not impact the computation, only bandwidth.

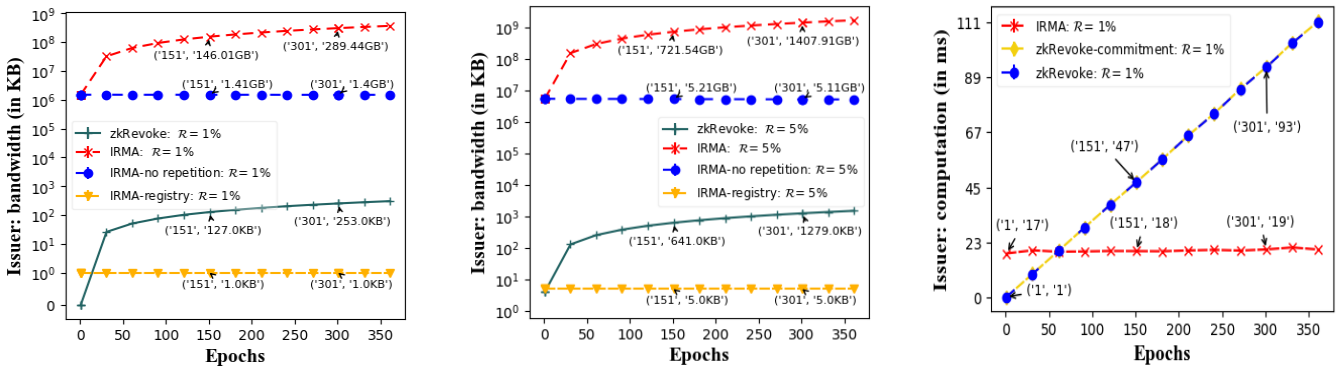
Higher revocation rates increase computation time in IRMA since the holder needs to incorporate more witness updates. The revocation rate does not impact the holder computation in *zkRevoke*. In addition, using the extension of proving k tokens using a single ZK proof reduces the computation time in *zkRevoke*. Figure 1b shows the impact of increasing k and $\mathcal{R}$. For example, when $k = 4$ and $\mathcal{R} > 5$, IRMA requires more computation from the holder.

Summary. Holders in *zkRevoke* require significantly less bandwidth compared to IRMA since constructing VPs in *zkRevoke* does not depend on the revocations of other VCs, unlike in IRMA. Similarly, the computation for holders in *zkRevoke* does not depend on the revocation of other VCs, in contrast to IRMA. When the extension of proving multiple tokens is used, *zkRevoke* incurs computation overhead for holders that is comparable to or better than the one in IRMA.



(a) Impact of small $\mathcal{R}$ on computation **(b) Impact of increasing $\mathcal{R}$, k on computation** **(c) The bandwidth requirements for holders**

Figure 1: The computation and bandwidth overheads of proof sharing for a single VP



(a) The bandwidth requirements for issuers **(b) Impact of increasing $\mathcal{R}$ on bandwidth** **(c) The computations requirements for issuers**

Figure 2: The performance impact of blacklist vs. accumulators on revocation

8.4 Blacklist approach

Our second objective is to quantify the benefits and costs of following a blacklist approach compared to a whitelist. We compare computation and bandwidth requirements of issuers, since issuers are the ones who update the list and share the list periodically irrespective of whether a blacklist or whitelist is used. In addition, since storage in DLTs typically incurs high cost, we also evaluate the storage requirements at the registry. The overhead for holders was already covered in Section 8.3, while the bandwidth overhead for verifiers to download lists can be inferred from the storage overhead at the registry. In addition, we benchmark the computation cost of the list commitment extension for both issuers and verifiers.

In Figure 2, we show how the performance impact on the issuer evolves over 365 epochs as more VCs are revoked.

Metric 3: Issuer's Bandwidth. Figure 2a shows the bandwidth requirements for issuers. In the experiment, every day $r = 27$ VCs are revoked, eventually reaching 10000 revoked VCs ($\mathcal{R} = 1\%$).

In *zkRevoke*, the issuer publishes the blacklist every epoch, which grows in size as more VCs are revoked. The blacklist eventually reaches $\mathcal{R} \times n \times 32$ bytes = $\Theta(\mathcal{R} \cdot n)$.

In IRMA, the issuer publishes the accumulator every epoch, and sends all revoked factors to the holders of valid VCs. Even without repetition (IRMA-*no repetition*) this requires significant bandwidth $\Theta(r \cdot n)$ from the issuer.

In IRMA-*registry*, the issuer only publishes the new accumulator and the new revoked factors to the registry. Thus, the issuer bandwidth requirement does not depend on the total number of revoked VCs but only on the number of VCs revoked per epoch (r).

Figure 2b shows that increased revocation rate leads to higher bandwidth consumption in *zkRevoke*, but the difference from the variants of IRMA stays the same.

Metric 4: Issuer's Computation. In IRMA, we measure the time to update accumulator in each epoch. In *zkRevoke*, we measure the time to compute tokens in each epoch. Figure 2c plots the results. In *zkRevoke*, the time to compute tokens increases linearly over the period of time since revoked VCs are accumulating in the blacklist. For example, *zkRevoke* requires more compuation than IRMA by a factor of 3 in epoch 201 and by a factor of 5 in epoch 301. The figure also shows that usage of the list commitment extension (described in Section 5.7.5) in *zkRevoke* adds insignificant overhead to compute a commitment of the list of tokens for issuers compared to the time to compute the list of tokens each epoch.

Metric 5: Storage. We evaluate storage requirements at the registry. We provide only theoretical analysis since it mostly depends

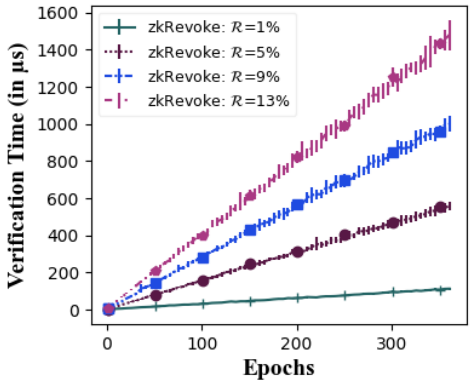


Figure 3: Time to verify commitment of list of tokens

on the number of hash values stored, not on the implementation. In IRMA, accumulators require a constant amount $\Theta(1)$ of storage. In *zkRevoke*, the blacklist requires linear amount $\Theta(\mathcal{R} \cdot n)$ of storage. The extension of list commitment requires a constant amount of storage since only a hash digest is stored each epoch. The variant IRMA-*registry* requires the same amount of storage as *zkRevoke*, since the revoked factors are stored in the registry.

Metric 6: Verification Cost for Commitment. When using the list commitment extension described in Section 5.7, verifiers need to check the blacklist against a commitment stored at the registry. Figure 3 illustrates the cost of this verification. The results show that the time to verify the commitment is on the order of microseconds even for higher revocation rates, which is insignificant compared to the computation requirements for verification of proofs in the VP (Figure 4c).

Summary. Our evaluation shows that issuers need to perform more computation in *zkRevoke* compared to IRMA, since the issuer has to recompute tokens for all revoked VCs after each epoch. *zkRevoke* avoids significant bandwidth consumption required in IRMA to broadcast factors, but requires more bandwidth than IRMA-*registry* since the blacklist is replaced every epoch, not just appended to.

8.5 General-purpose vs. custom-built ZKPs

In our construction, we use general-purpose ZKPs based on the Groth16 ZKP scheme, whereas in IRMA, custom-built ZKPs are used. The objective is to analyze the trade-offs between these approaches and identify the benefits and costs.

The theoretical analysis of proof size, proof generation time, and proof verification time is discussed, along with the definition of the Groth16 scheme in [38]. We directly use the scheme as it is, without any modification. Therefore, the analysis also applies to our protocol. The theoretical analysis of these metrics along with the definition of the ZKP scheme used in IRMA is defined in [35].

We benchmark the performance of the IRMA and Groth16 schemes here and compare the results. The protocols are benchmarked on the same machine to ensure fair comparison.

Metric 7: ZK Proof Size. The size of a ZK proof in *zkRevoke* is 164 bytes, whereas the size of a non-revocation proof in IRMA is 1831 bytes, as shown in Figure 4a. *zkRevoke* requires 11x less bandwidth for proofs, compared to IRMA.

Metric 8: ZK Proof Generation Time. Figure 4b illustrates the differences in the time required to compute proofs in IRMA and *zkRevoke*. The results show that the proof generation time in IRMA is 10x faster than the proof generation time in *zkRevoke*.

Metric 9: ZK Proof Verification Time. Figure 4c illustrates the differences in the time required to verify all proofs for m epochs in IRMA and *zkRevoke*. The results show that the proof verification time in *zkRevoke* is 6x faster than the proof generation time in IRMA.

Metric 10: ZK Circuit. To generate and verify proofs, ZK circuits are one of the parameters in *zkRevoke*, whereas public keys are sufficient in IRMA. In IRMA, the ZKP scheme is designed as a digital signature scheme and enables holders and verifiers to use public keys to generate and verify proofs. The size of the circuit is impacted only by the extension of proving multiple tokens using a single ZK proof. Figure 5a and Figure 5b compare the size of the circuits and time to generate them during setup in *zkRevoke* with key generation time and key size in IRMA. The results show that IRMA outperforms *zkRevoke* since public keys are shorter in size and take a significantly smaller amount of time to generate compared to ZK circuits.

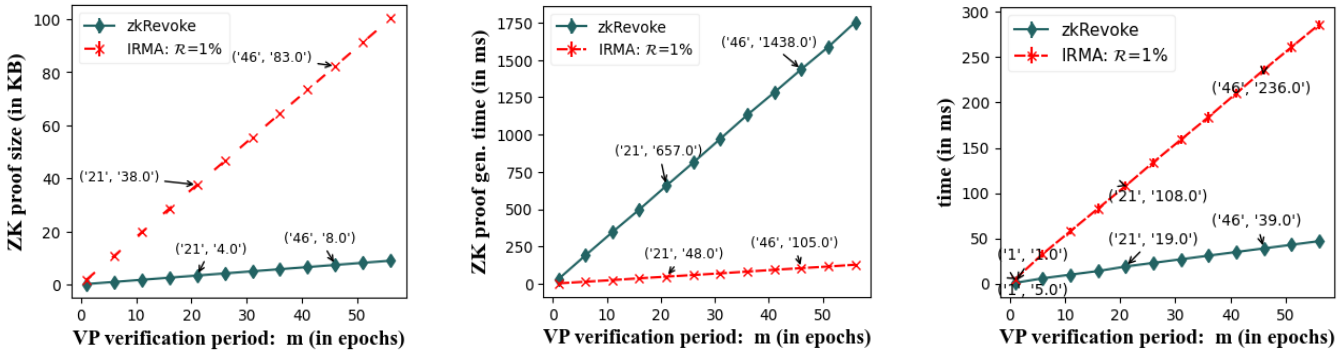
Summary. *zkRevoke* provides shorter proofs (11× smaller) and shorter verification times (6× faster) compared to IRMA, reducing bandwidth requirements for holders and computation requirements for verifiers. However, IRMA provides shorter proof generation times (10× faster) affecting holders' computation.

Another relevant difference is that circuits generated in *zkRevoke* have a significant size ($\geq$ 88 KB). In IRMA, public keys (2.5 KB) serve a similar role as circuits in *zkRevoke*. In *zkRevoke*, verifiers need to retrieve circuits from registry, and holders receive them from issuers and store them. Even though the circuit size is larger than public keys used in IRMA, the circuit size is still practical for many use cases since it is in the order of few KB.

8.6 ZK Circuit Breakdown

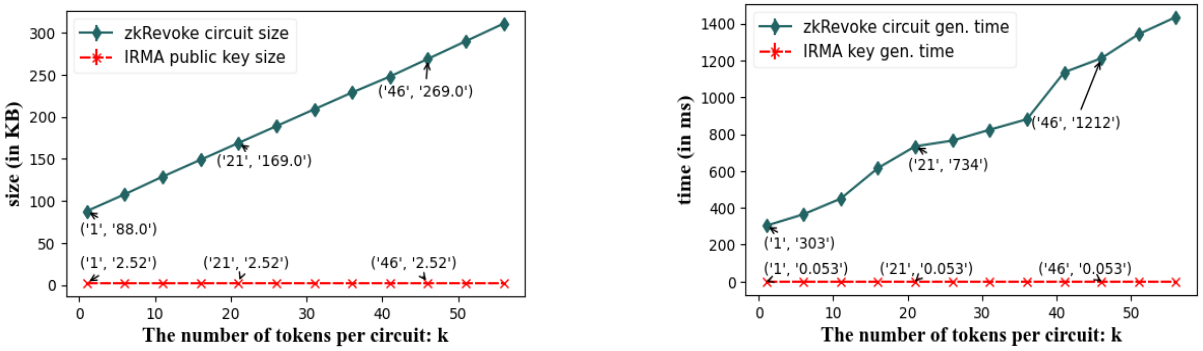
In this section, we evaluate the performance of our ZK circuit, which is defined in Section 5.5. First, Table 4 presents the performance breakdown of the circuit components by benchmarking the number of constraints induced by the individual conditions in the ZK circuit. The results show that hash-related conditions require 661 constraints, whereas signature verification requires 7993 constraints. This indicates that signature verification is a dominant part (85.8%) of the circuit.

Secondly, we evaluate the overhead of the Groth16 scheme by considering the performance of the "empty" circuit instantiated without any conditions. Table 5 compares the performance of the complete and empty circuits. The results show that witness and proof generation for the empty circuit add 5% and 1.5% overhead respectively, whereas proof verification for the empty circuit requires almost the same amount of time as for the complete circuit.



(a) The total amount of proofs for a VP (b) The time to generate proofs for a VP (c) The time to verify proofs for a VP

Figure 4: The ZK proof size and the time to generate and verify proofs for a single VP having verification period m



(a) Size: ZK circuits in *zkRevoke* vs. public keys in IRMA (b) Generation time: ZK circuits in *zkRevoke* vs. public keys in IRMA

Figure 5: Comparing the size and generation time of ZK circuits in *zkRevoke* with public keys in IRMA

Table 4: Circuit Constraints Breakdown

Constraint	Signature Ver.	Token Ver.	Challenge Ver.	Complete Circuit
No. of Constraints	7993	661	661	9315

Table 5: Groth16 Overhead

	Witness Gen. Time	Proof Gen. Time	Proof Ver. Time
Empty Circuit	5 µs	461 µs	710 µs
Complete Circuit	97 µs	30428 µs	734 µs

9 Conclusion

In this work, we have proposed *zkRevoke*, a revocation technique for Verifiable Credentials that prevents verifiers from monitoring the revocation status indefinitely. *zkRevoke* follows a blacklist approach utilizing time-bound tokens and Zero-Knowledge Proofs (ZKPs). An issuer computes tokens corresponding to revoked VCs after each epoch and inserts those tokens in a list. A holder generates (a) tokens corresponding to current and future epochs and (b) corresponding ZK proofs. Afterwards, the holder shares those proofs with a verifier to prove the correctness of the tokens.

zkRevoke solves the problem of *untraceability* since verifiers can check the revocation status of each token only during the corresponding epoch. In addition, *zkRevoke* satisfies our auxiliary goals: (a) each holder configures how long a verifier can verify revocation status corresponding to the holder's VP, (b) a holder shares a ZK proof only once rather than periodically, and (c) an issuer is not able to learn anything about the verification of VPs.

Transitioning to post-quantum secure primitives is essential for long-term deployability of zkRevoke. Current post-quantum ZKP schemes and signature algorithms (e.g., lattice-based or hash-based constructions) would require replacing our Groth16 proofs and EdDSA signatures, introducing larger proof sizes and computational overhead. However, the modular design of our system facilitates such transitions: the ZK circuit definition and protocol logic remain largely unchanged, requiring only the substitution of underlying cryptographic primitives. As post-quantum ZKP schemes mature, future implementations can adopt these primitives while maintaining the core privacy properties of configurable untraceability and one-time VP sharing.

Acknowledgments

This work was supported by the Research Council of Norway through IKTPLUSS Program under Grant 288106. Arlindo F. da Conceição is supported by FAPESP, grant 2023/00783-7. Leander Jehl was supported by the "Data For All" project funded by Interreg North Sea. We would like to thank the anonymous reviewers and the editor for their helpful suggestions.

References

[1] Manu Sporny, Dave Longley, and David Chadwick. Verifiable Credentials Data Model v1.1. https://w3c.github.io/vc-data-model/.
[2] Openid for Verifiable Credentials - Overview. https://openid.net/sg/openid4vc/.
[3] Carlo Mazzocca, Abbas Acar, Selcuk Uluagac, Rebecca Montanari, Paolo Bellavista, and Mauro Conti. A Survey on Decentralized Identifiers and Verifiable Credentials. *IEEE Communications Surveys & Tutorials*, pages 1–1, 2025.
[4] Kheng Leong Tan, Chi-Hung Chi, and Kwok-Yan Lam. Survey on Digital Sovereignty and Identity: From Digitization to Digitalization. *ACM Comput. Surv.*, 56(3), October 2023. https://doi.org/10.1145/3616400.
[5] Julia Hesse, Nitin Singh, and Alessandro Sorniotti. How to bind anonymous credentials to humans. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 3047–3064, Anaheim, CA, August 2023. USENIX Association.
[6] Olivier Blazy, Céline Chevalier, Guillaume Renaut, Thomas Ricosset, Eric Sageloli, and Hugo Senet. Efficient implementation of a post-quantum anonymous credential protocol. In *Proceedings of the 18th International Conference on Availability, Reliability and Security*, ARES '23, New York, NY, USA, 2023. Association for Computing Machinery.
[7] IRMA - Revocation. https://docs.yivi.app/revocation/.
[8] Anoncreds Specification v1.0. https://hyperledger.github.io/anoncreds-spec/.
[9] EBSI Hub: VC Framework, 2025. https://hub.ebsi.eu/vc-framework.
[10] Orgbook. https://orgbook.gov.bc.ca/.
[11] Dock. https://www.dock.io/.
[12] Microsoft Entra Verified ID. https://www.microsoft.com/en-in/security/business/identity-access/microsoft-entra-verified-id.
[13] Indicio. https://indicio.tech/.
[14] Digital Bazaar. https://www.digitalbazaar.com/.
[15] Gergely Alpár, Fabian Van Den Broek, Brinda Hampiholi, Bart Jacobs, Wouter Lueks, and Sietse Ringers. Irma: practical, decentralized and privacy-friendly identity management using smartphones. In *10th Workshop on Hot Topics in Privacy Enhancing Technologies (HotPETs 2017)*, pages 1–2, 2017.
[16] IOTA Identity - Revocation Bitmap 2022, 2024. https://docs.iota.org/developer/iota-identity/references/revocation-bitmap-2022.
[17] IOTA Identity - Revocation Timeframe 2024, 2024. https://docs.iota.org/developer/iota-identity/references/revocation-timeframe-2024.
[18] Nacereddine Sitouah, Francesco Bruschi, Francesco Lorenzo Pallotta, Riccardo Mencucci, and Donatella Sciuto. An untraceable credential revocation approach based on a novel merkle tree accumulator. In *2024 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 210–214, May 2024.
[19] Francesco Buccafurri and Carmen Licciardi. Towards privacy-preserving revocation of verifiable credentials with time-flexibility. In *2025 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 516–521, 2025.
[20] Andreas Freitag. A new privacy preserving and scalable revocation method for self sovereign identity – the perfect revocation method does not exist yet, 2022.
[21] Tobias Looker, Paul Bastian, and Christian Bormann. Token Status List (TSL). Internet-Draft draft-ietf-oauth-status-list-12, Internet Engineering Task Force, July 2025. Work in Progress.
[22] Revocation by EBSI - EBSI's credential status framework and how to choose a revocation method when using W3C verifiable credentials (and more), 2023. https://www.linkedin.com/posts/eublockchainserviceinfrastructure_revocation-whitepaper-activity-7118152526264201216-l4I4/.
[23] Michael Rosenberg, Jacob 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 (SP)*, pages 790–808, 2023.
[24] Matteo Loporchio, Anna Bernasconi, Damiano Di Francesco Maesa, and Laura Ricci. A survey of set accumulators for blockchain systems. *Computer Science Review*, 49:100570, 2023.
[25] Li Xu, Tianyu Li, and Zekeriya Erkin. Verifiable credentials with privacy-preserving tamper-evident revocation mechanism. In *2023 Fifth International Conference on Blockchain Computing and Applications (BCCA)*, pages 266–273, 2023.
[26] Carlo Mazzocca, Abbas Acar, Selcuk Uluagac, and Rebecca Montanari. EVOKE: Efficient revocation of verifiable credentials in IoT networks. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 1279–1295, Philadelphia, PA, August 2024. USENIX Association.
[27] Praveensankar Manimaran, Arlindo F. Da Conceição, Thiago Garrett, Mayank Raikwar, and Roman Vitenberg. Prevoke: Privacy-preserving configurable method for revoking verifiable credentials. In *2024 IEEE International Conference on Blockchain (Blockchain)*, pages 354–361, 2024.
[28] Daria Schumm, Rahma Mukta, and Hye-young Paik. Efficient credential revocation using cryptographic accumulators. In *2023 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*, pages 127–134, 2023.
[29] Athanasia Maria Papathanasiou and George C. Polyzos. Privacy-preserving revocation of verifiable credentials with verifiable random functions. In *2024 International Conference on Information Networking (ICOIN)*, pages 391–394, 2024.
[30] John M Schanck. Clubcards for the WebPKI: smaller certificate revocation tests in theory and practice. *Cryptology ePrint Archive*, 2025.
[31] Yabing Liu, Will Tome, Liang Zhang, David Choffnes, Dave Levin, Bruce Maggs, Alan Mislove, Aaron Schulman, and Christo Wilson. An end-to-end measurement of certificate revocation in the web's PKI. In *Proceedings of the 2015 Internet Measurement Conference*, IMC '15, page 183–196, New York, NY, USA, 2015. Association for Computing Machinery. https://doi.org/10.1145/2815675.2815685.
[32] Dynamic Status List, 2024. https://hub.ebsi.eu/vc-framework/credential-status-framework/vcs#dynamic-status-list.
[33] Sharon Boeyen, Stefan Santesson, Tim Polk, Russ Housley, Stephen Farrell, and David Cooper. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280, May 2008.
[34] Stefan Santesson, Michael Myers, Rich Ankney, Ambarish Malpani, Slava Galperin, and Dr. Carlisle Adams. X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. RFC 6960, June 2013.
[35] Foteini Baldimtsi, Jan Camenisch, Maria Dubovitskaya, Anna Lysyanskaya, Leonid Reyzin, Kai Samelin, and Sophia Yakoubov. Accumulators with applications to anonymity-preserving revocation. In *2017 IEEE European Symposium on Security and Privacy*, pages 301–315, 2017.
[36] Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente. An accumulator based on bilinear maps and efficient revocation for anonymous credentials. In Stanisław Jarecki and Gene Tsudik, editors, *Public Key Cryptography – PKC 2009*, pages 481–500, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.
[37] Alfredo De Santis and Giuseppe Persiano. Zero-knowledge proofs of knowledge without interaction. In *IEEE FOCS*, pages 427–436, 1992.
[38] Jens Groth. On the Size of Pairing-Based Non-interactive Arguments. In *Advances in Cryptology – EUROCRYPT 2016*, pages 305–326, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.
[39] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas Ward. Marlin: Preprocessing zkSNARKs with Universal and Updatable SRS. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020*, pages 738–768, Cham, 2020. Springer International Publishing.
[40] obfuscated for the purpose of double-blinded submission.
[41] Toru Nakanishi, Hiroki Fujii, Yuta Hira, and Nobuo Funabiki. Revocable group signature schemes with constant costs for signing and verifying. In *Public Key Cryptography – PKC 2009*, pages 463–480, 2009.
[42] Jorn Lapon, Markulf Kohlweiss, Bart De Decker, and Vincent Naessens. Analysis of revocation strategies for anonymous idemix credentials. In *12th Communications and Multimedia Security*, pages 3–17, 2011.
[43] Credivera. https://www.credivera.com/.
[44] Velocity network foundation. https://www.velocitynetwork.foundation/.
[45] Deepak Maram, Harjasleen Malvai, Fan Zhang, Nerla Jean-Louis, Alexander Frolov, Tyler Kell, Tyrone Lobban, Christine Moy, Ari Juels, and Andrew Miller. Candid: Can-do decentralized identity with legacy compatibility, sybil-resistance, and accountability. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1348–1366, 2021.
[46] IOTA Identity, 2025. https://docs.iota.org/developer/iota-identity/.
[47] Jens Ernstberger, Stefanos Chaliasos, George Kadianakis, Sebastian Steinhorst, Philipp Jovanovic, Arthur Gervais, Benjamin Livshits, and Michele Orrù. zk-bench: A toolset for comparative evaluation and performance benchmarking of snarks. In Clemente Galdi and Duong Hieu Phan, editors, *Security and Cryptography for Networks*, pages 46–72, Cham, 2024. Springer Nature Switzerland.
[48] Praveensankar Manimaran, Thiago Garrett, Leander Jehl, and Roman Vitenberg. Decentralization trends in identity management: From federated to self-sovereign identity management systems. *Computer Science Review*, 58:100776, 2025.
[49] Šeila Bećirović Ramić, Ehlimana Cogo, Irfan Prazina, Emir Cogo, Muhamed Turkanović, Razija Turčinhodžić Mulahasanović, and Saša Mrdović. Selective disclosure in digital credentials: A review. *ICT Express*, 10(4):916–934, 2024.
[50] W3C. Verifiable Credentials Data Model v2.0, 2025. https://www.w3.org/TR/vc-data-model-2.0/.
[51] Ivan Damgård. *Commitment Schemes and Zero-Knowledge Protocols*, pages 63–86. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999.
[52] Ganache. https://archive.trufflesuite.com/ganache/.
[53] gnark-crypto. https://github.com/Consensys/gnark-crypto.
[54] Package rand, 2025. https://pkg.go.dev/crypto/rand.

[55] Package sha256, 2025. https://pkg.go.dev/crypto/sha256.
[56] Gabi: Implementation of the idemix attribute based credential scheme used in irma, 2025. https://github.com/privacybydesign/gabi.
[57] Paulo S. L. M. Barreto and Michael Naehrig. Pairing-friendly elliptic curves of prime order. In Bart Preneel and Stafford Tavares, editors, *Selected Areas in Cryptography*, pages 319–331, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.
[58] Martin Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. MiMC: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016*, pages 191–219, Berlin, Heidelberg, 2016. Springer Berlin Heidelberg.
[59] Daniel J. Bernstein, Niels Duif, Tanja Lange, Peter Schwabe, and Bo-Yin Yang. High-speed high-security signatures. *Journal of Cryptographic Engineering*, 2(2):77–89, 2012.
[60] Barry WhiteHat, Jordi Baylina, and Marta Bellés. Baby jubjub elliptic curve. *Ethereum Improvement Proposal, EIP-2494*, 29, 2020.
[61] Ethereum improvement proposal(eip) - 7825, 2025. https://github.com/ethereum/EIPs/blob/master/EIPS/eip-7825.md.

A Additonal Setup Details

Code. We use Ethereum for implementing the registry, though this choice is mostly orthogonal to the rest of the design. In order to interact with the registry, each issuer deploys a dedicated smart contract, which allows the issuer to publish and update parameters and a blacklist. Our choice for deploying a separate contract for each issuer is in line with related work [8], [27]. The contract is implemented in the Solidity programming language. It is deployed on a private Ethereum DLT using Ganache v7.9 [52].

Zero-Knowledge Proofs, random numbers, and the SHA-256 hash are produced using the gnark-crypto [53], rand [54], and sha256 [55] libraries, respectively. IRMA implementation is based on the Gabi [56] library.

Instantiating Cryptographic Primitives. We instantiate the Groth16 ZKP scheme with the BN254 curve [57].

The MIMC [58] hash function is used in the ZK circuit. For digital signatures, we use the Edwards-curve Digital Signature Algorithm [59] with Twisted Edwards Curve BN254 [60]. We employ MIMC hash functions and Twisted Edwards Curve BN254 because they are compatible with ZK circuits. The seed and hash parameters are chosen by the libraries so as to match the target preimage resistance and security level.

B Cryptographic Preliminaries

B.1 Zero-Knowlege Proofs

Zero-Knowledge Proof (ZKP) schemes [37–39] allow a prover to convince a verifier of a statement's validity without revealing any additional information.

Formally, a ZKP system consists of a prover P and a verifier V. Given an efficiently computable binary relation $R(x, w)$, where x is a statement and w is a witness, the corresponding language is $\mathcal{L} = \{x \mid \exists w \text{ s.t. } R(x, w) = 1\}$. A ZKP scheme ensures that P can prove $x \in \mathcal{L}$ without disclosing w.

Properties. A Zero-Knowledge Proof system satisfies the following three properties:

- *Completeness:* If $x \in \mathcal{L}$, Prover P can convince the verifier V that P has the correct input.
- *Soundness:* If $x \notin \mathcal{L}$, the statement x cannot be proven.
- *Zero-Knowledge:* Verifier V learns nothing beyond the truth of statement x.

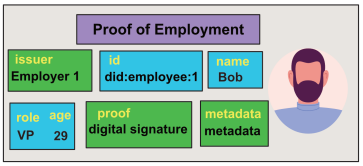


Figure 6: Example of VC: an employee identification.

We use Groth16 [38] ZKP scheme in our solution. Groth16 ZKP scheme consists of the following algorithms:

- G16.Setup $(1^\lambda, C) \rightarrow crs$: Given a security parameter λ and a circuit C, output a common reference string (CRS) crs.
- G16.Prove $(x, crs, w) \rightarrow \pi$: Given a statement x, crs, and witness w, output a ZK proof π .
- G16.Verify $(x, crs, \pi) \rightarrow \{0, 1\}$: Given a statement x, crs, and a ZK proof π, output accept or reject.

The notations used in the algorithms are described in zk-creds [23].

Zero-Knowledge Circuit. A ZK circuit is a mathematical abstraction that encodes a computer program as a set of constraints suitable for Zero-Knowledge Proofs (ZKPs). The execution of the program is represented as a constraint system comprising two types of constraints:

(1) *Private constraints*, which capture relationships involving private inputs (known only to the prover) and internal variables of the circuit.
(2) *Public constraints*, which involve public inputs (known to both the prover and the verifier) and internal variables.

A valid witness set of input values satisfying all constraints serves as evidence of correct program execution. The prover uses this witness to construct a proof asserting the correctness of the computation without revealing any private input.

In the Groth16 ZKP scheme, once the ZK circuit is defined, a common reference string, crs, is generated based on a given security parameter. This crs is subsequently used in both the proof generation and verification phases, ensuring soundness and succinctness of the proof system.

C Additional Details on Verifiable Credentials

An example of a VC is depicted in Fig. 6, showcasing details about an employee covering the name of an employer who issued the VC and claims about the employee, covering the employee's identifier, name, age, role, and picture.

Use Case: Workforce Identity. Workforce Identity [12, 43, 44] is one of the popular use cases of VCs. In Workforce Identity, VCs are used for the purpose of proving employment status. Figure 7 illustrates a scenario where Employer 1 issues a VC signifying employee ID to Bob who is employed by Employer 1. After some time, when Bob applies for a job at Employer 2, Bob generates a VP from the VC and shares the VP with Employer 2 to prove that he is currently working at Employer 1. When Bob leaves Employer 1, Employer 1 revokes Bob's Employee ID VC and stores a proof in the registry. We refer to and extend this use case throughout this work to motivate the problem and other functional requirements.

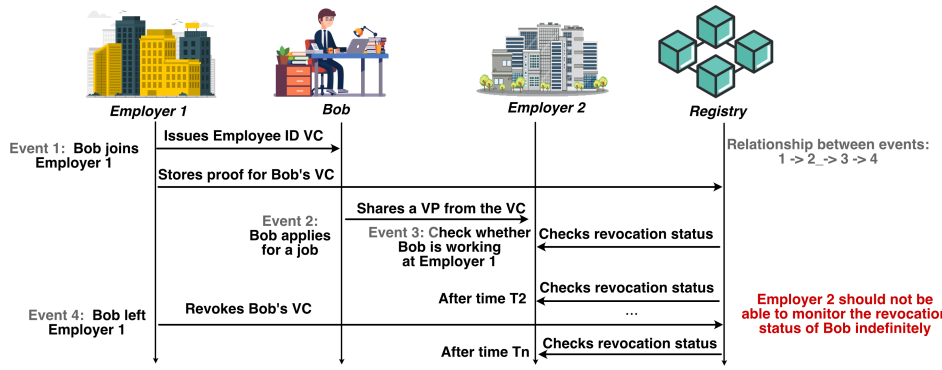


Figure 7: The untraceability requirement

Operations on VCs. A system for managing VCs consists of the following main operations:

- **Issuance** An issuer performs this operation to issue VCs to a holder based on the claims (attributes) provided by the holder.
- **Revocation** An issuer performs this operation to revoke a VC.
- **Presentation** A holder executes this operation to generate a VP for the underlying VC, in order to share the generated VP with a verifier.
- **Verification** A verifier performs this operation to check the validity of the VP received from the holder.

VCs may include an expiration period, as specified in the W3C Verifiable Credentials Specification [1]. However, expiration is optional and depends on the use case. For example, passports usually expire after a certain period, while diplomas often remain valid indefinitely.

D Untraceability: Security Game

The game ($\mathbf{Game}_{\mathcal{A}}^{untrace}$) is played between a challenger and an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ and consists of four phases: (a) Setup Phase, (b) Query Phase, (c) Challenge Phase, and (d) Guess Phase. The phases are defined below:

Setup Phase

(1) Challenger runs Setup $(1^\lambda) \rightarrow$ (sk, pk, crs, ts_0, dur, $\text{RevList}_{\text{iss}}$, $\text{blacklist}_{\text{iss}}$)
(2) Challenger keeps a set Q_h of honest holder's VCs
(3) Challenger initializes query sets $Q_c \leftarrow \phi, Q_r \leftarrow \phi$, and $Q_p \leftarrow \phi$
(4) Challenger provides (pk, crs, $\text{RevList}_{\text{iss}}$) and the following oracles to Adversary $\mathcal{A}_1$: a) $O^{\text{GenVC}} \rightarrow$ vc; b) $O^{\text{Revoke}}; \rightarrow \text{RevList}_{\text{iss}}'$ or $\perp$; c) $O^{\text{Refresh}} \rightarrow \text{blacklist}_{\text{iss}}$; d) $O^{\text{GenVP}} \rightarrow$ vp

Oracle O^{GenVC}
- claims $\leftarrow \{0, 1\}$
- vc $\leftarrow$ Issuance(sk, claims)
- $Q_c \leftarrow Q_c \cup \{\text{vc}\}$
- **return** vc

Oracle $O^{\text{Revoke}}(\text{vc}, \text{RevList}_{\text{iss}})$
- **if** vc $\in Q_c$ **then**
  - $\text{RevList}_{\text{iss}}' \leftarrow \text{Revocation}(\text{vc}, \text{RevList}_{\text{iss}})$
  - $Q_r \leftarrow Q_r \cup \{\text{vc}\}$
  - **return** $\text{RevList}_{\text{iss}}'$
- **else return** $\perp$

Oracle $O^{\text{Refresh}}(\text{e}, \text{RevList}_{\text{iss}}')$
- $\text{blacklist}_{\text{iss}} \leftarrow \text{Refresh}(\text{e}, \text{RevList}_{\text{iss}}')$
- **return** $\text{blacklist}_{\text{iss}}$

Oracle $O^{\text{GenVP}}(\text{pk}, \text{crs}, \text{e}, \text{m}, \text{challenge})$
- Fetch a valid vc, s.t., vc $\in Q_h \cup Q_c$
- $\text{vp}_0 \leftarrow \text{Presentation}(\text{pk}, \text{crs}, \text{e}, \text{vc}, \text{m}, \text{challenge})$
- $Q_p \leftarrow Q_p \cup \{\text{vp}\}$
- **return** vp

Query Phase

Adversary $\mathcal{A}_1$ adaptively queries the oracles $O^{\text{GenVC}}, O^{\text{Revoke}}, O^{\text{Refresh}}$, and O^{GenVP} polynomial many times.

Challenge Phase

(1) Adversary $\mathcal{A}_1$ outputs (challenge_0, challenge_1, $state_{\mathcal{A}_1}$)
(2) Challenger selects two VCs vc_0, vc_1 where $\text{vc}_0, \text{vc}_1 \in Q_h$; vc_0, $\text{vc}_1 \notin Q_c \cup Q_r$, and $\text{vc}_0 \neq \text{vc}_1$, and computes:
   - Verification periods m_0, m_1
   - The current epoch value: $\text{e} \leftarrow (time() - \text{ts}_0)/\text{dur}$.
   - $\text{vp}_0 \leftarrow \text{Presentation}(\text{pk}, \text{crs}, \text{e}, \text{vc}_0, \text{m}_0, \text{challenge}_0)$
   - $\text{vp}_1 \leftarrow \text{Presentation}(\text{pk}, \text{crs}, \text{e}, \text{vc}_1, \text{m}_1, \text{challenge}_1)$
(3) After verification period $\max(\text{m}_0, \text{m}_1)$ expires, Challenger samples $b \in \{0, 1\}$
(4) Challenger computes the current epoch value: $\text{e} \leftarrow (time() - \text{ts}_0)/\text{dur}$ and runs the following:
   - $\text{RevList}_{\text{iss}}' \leftarrow \text{Revocation}(\text{vc}_b, \text{RevList}_{\text{iss}})$
   - $\text{blacklist}_{\text{iss}}' \leftarrow \text{Refresh}(\text{e}, \text{RevList}_{\text{iss}}')$
(5) Challenger gives (vp_0, vp_1, pk, crs, $\text{blacklist}_{\text{iss}}'$, $state_{\mathcal{A}_1}$) to adversary $\mathcal{A}_2$

Guess Phase

(1) Adversary $\mathcal{A}_2$ outputs guess $b' \in \{0, 1\}$
(2) Game returns 1 if $b = b'$, otherwise returns 0

We consider a hybrid adversary model in which the adversary can act both as a credential holder and as a verifier during the query phase. As a holder, the adversary can request VCs using the O^{GenVC} oracle, which returns the full credential, including metadata and signature. As a verifier, it can interact with the O^{GenVP} oracle to obtain VPs from honest holders based on arbitrary queries. However, in the challenge phase, the VPs are derived from credentials never revealed to the adversary, reflecting a setting in which honest holders generate proofs using unseen VCs. This models a malicious verifier who has significant observational knowledge, but cannot correlate challenge VPs with previously seen credentials.

E Security Analysis

We prove that *zkRevoke* satisfies completeness, soundness and untraceability properties.

E.1 Completeness

Our construction satisfies completeness if, for a VP valid in epoch e generated from a non-revoked VC that is not expired in epoch e, the verification of VP in epoch e always succeeds.

Proof Sketch. We prove completeness of *zkRevoke* as follows: Let vp be a valid VP generated using the *Presentation* procedure in epoch e′ from a non-revoked vc that is not expired in epoch e such that $e' \leq e < (e' + m)$. The verification of vp in epoch e consists of two main components:

ZK Proof Verification. For m tokens, the holder generates m ZK proofs π_{zkp}. Pr[All ZK proofs Succeed] = $(1 - \epsilon_1)^m$, where $\epsilon_1 = \text{negl}(\lambda)$ due to Groth16 ZKP completeness.

Revocation Status Verification. For any valid epoch e in the verification period m, the *VerifyRevocationStatus* algorithm checks if token $\in \text{blacklist}'_{\text{iss}}$. Due to the collision resistance of H, for epoch e and unique seed values, $H(\text{seed}_1, e) \neq H(\text{seed}_2, e)$, meaning it is not possible for any revoked vc to result in the same token value. Therefore, for epoch e: token = H(vc.seed, e) $\notin \text{blacklist}'_{\text{iss}}$: Pr[token $\notin \text{blacklist}'_{\text{iss}}$] = $1 - \epsilon_2$, where $\epsilon_2 = \text{negl}(\lambda)$ due to the collision-resistance property of hash function. Hence, for all m epochs: Pr[All revocation checks pass] = $(1 - \epsilon_2)^m$

Hence, combined probability of verification of vp: Pr[Verification(pk, crs, e, $\text{blacklist}'_{\text{iss}}$, m, vp, challenge) = 1] = Pr[All ZK proofs Succeed] · Pr[All revocation checks pass]

$$\begin{aligned}\Pr[\text{Verification}(\text{vp}, \cdots) = 1] &= (1 - \epsilon_1)^m (1 - \epsilon_2)^m \\ &\geq 1 - \text{negl}(\lambda)\end{aligned}$$

□

E.2 Soundness

Our scheme satisfies soundness if for a VP which is either invalid or created using a revoked or expired VC in epoch e, the verification of VP in epoch e results in failure.

Proof Sketch. We prove soundness of *zkRevoke* as follows:

Regarding the invalid VPs, a VP can be formed in many ways. Nevertheless, if the VP does not include correct information, if the ZKP system is sound and the digital signature scheme is unforgeable, then the verification of the ZK proof included in the VP would result in failure.

Regarding a VP generated from revoked or expired VC using the presentation procedure, the *VerifyRevocationStatus* algorithm deterministically captures the revocation status of tokens corresponding to the VP in epoch e and checks for expiry.

We justify the soundness property of Π by analyzing two attack scenarios. Let $\mathcal{A}$ be a malicious holder of a VC vc, and let vp be a VP generated from vc by presentation. The goal for $\mathcal{A}$ is to create a VP vp′ that differs vp by one of the following: (a) vp′ includes a different token than vp, avoiding revocation checks, (b) vp′ includes different claims than vp, or (c) vp′ includes a different expiration period than vp, avoiding expiration checks.

Scenario 1: Generating a proof. The ZK circuit verifies the generation of token from seed and e. $\mathcal{A}$ may attempt to generate vp′ by selecting a different seed, changing the claims or the expiration period. However, the ZK circuit verifies that these values are signed by the issuer. The adversary $\mathcal{A}$ can try to forge the signature and proof. However,

Signature Forgery: Pr[$\mathcal{A}$ forges valid signature on s'] $\leq \text{negl}(\lambda)$ due to signature unforgeability.

ZK Proof Forgery: Pr[$\mathcal{A}$ generates valid π_{zkp} without proper signature] $\leq \text{negl}(\lambda)$ due to ZKP Soundness.

Combined: Pr[Scenario 1 succeeds] $\leq \text{negl}(\lambda)$

Scenario 2: Reusing a proof. Suppose $\mathcal{A}$ obtains a valid VP vp′, that includes the token, and the ZK proof π_{zkp}. $\mathcal{A}$ may try to either reuse the proofs instead of generating new ones or recover private inputs used in the proofs such as digital signatures and seed values. However, the reused proof will not verify with different claims, expiration period, or most importantly, a different challenge. In addition, recovery of private inputs is not computationally feasible.

ZK Proof Soundness: Pr[π_{zkp} verifies with different public values] $\leq \text{negl}(\lambda)$ due to ZKP Soundness.

Recovery from ZK Proofs: Pr[$\mathcal{A}$ learns signature and other private inputs from $\pi_{\text{zkp}} \leq \text{negl}(\lambda)$ due to ZKP zero-knowledge.

Therefore, $\mathcal{A}$'s best strategy has success probability: Pr[$\mathcal{A}$ constructs passing vp] $\leq \max(\text{negl}(\lambda), \text{negl}(\lambda)) = \text{negl}(\lambda)$. Hence;

$$\begin{aligned}&\Pr[\text{Verification}(\text{pk}, \text{crs}, \text{e}, \text{blacklist}'_{\text{iss}}, \text{m}, \text{vp}, \text{challenge}) = 0] \\ &= 1 - \Pr[\text{Verification}(\text{vp}, \cdots) = 1] \\ &\geq 1 - \Pr[\mathcal{A} \text{ constructs passing vp}] \\ &\geq 1 - \text{negl}(\lambda)\end{aligned}$$

□

E.3 Untraceability

Proof Sketch. We prove untraceability by showing that an adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ has at most negligible advantage in the untraceability game defined in Definition 5.4. In the challenge phase of the game, the adversary $\mathcal{A}_2$ receives (vp_0, vp_1, pk, crs, $\text{blacklist}'_{\text{iss}}$, $state_{\mathcal{A}_1}$). Tokens for revoked vc_b appear in $\text{blacklist}'_{\text{iss}}$. The goal of $\mathcal{A}_2$ is to distinguish whether vc_0 or vc_1 was revoked. In other words, $\mathcal{A}_2$ needs to track the revocation status of the vc_b corresponding

to vp_b after its verification period is over. Below we show that this is not feasible, since $\mathcal{A}_2$ cannot extract the seed from the tokens or ZK proofs in the VPs, and therefore cannot link tokens from different epochs.

Scenario 1 : Recovering the seed from tokens. $\mathcal{A}_2$ might attempt to recover the seed from tokens in the VPs, then compute the current epoch token to check against $\text{blacklist}'_{\text{iss}}$. For any $\text{token}_i \in vp_b.\text{tokens}$, we have $\text{token}_i = \text{H}(vc_b.\text{seed}, e_i)$ for epoch e_i. By the pre-image resistance of H: $\Pr[\mathcal{A}_2 \text{ recovers } vc_b.\text{seed from token}_i] \leq \text{negl}(\lambda)$

Scenario 2 : Extracting the seed from ZK proofs. $\mathcal{A}_2$ might attempt to extract the seed from the ZK proofs π_{zkp} in the VPs. Each $\pi \in \pi_{zkp}$ proves knowledge of $vc_b.\text{seed}$ without revealing it. By the zero-knowledge property of the ZKP system: $\Pr[\mathcal{A}_2 \text{ learns } vc_b.\text{seed from } \pi_{\text{zkp}}] \leq \text{negl}(\lambda)$

Scenario 3 : Linking tokens across epochs. $\mathcal{A}_2$ might try to link tokens from the VPs to tokens in $\text{blacklist}'_{\text{iss}}$ through correlation attacks. However, tokens for different epochs are computationally independent: (a) tokens in VPs: $\text{H}(vc_b.\text{seed}, e_i)$ for epochs $e_i \in \{e, e+1, \ldots, e + m_b - 1\}$, and (b) tokens in $\text{blacklist}'_{\text{iss}}$: $\text{H}(vc_b.\text{seed}, e')$ for current epoch $e' > e + \max(m_0, m_1) - 1$.

Since the verification periods have expired ($e' > e + \max(m_0, m_1) - 1$), there is a "gap" between the epochs covered by VP tokens and the current epoch. By pre-image resistance and the pseudorandom properties of H, these tokens appear independent. Therefore, $\Pr[\mathcal{A}_2 \text{ links tokens across epochs}] \leq \text{negl}(\lambda)$.

Combining all attack vectors, $\Pr[\mathcal{A}_2 \text{ succeeds}] \leq \Pr[\mathcal{A}_2 \text{ recovers } vc_b.\text{seed from token}_i] + \Pr[\mathcal{A}_2 \text{ learns } vc_b.\text{seed from } \pi_{\text{zkp}}] + \Pr[\mathcal{A}_2 \text{ links tokens across epochs}] \leq \text{negl}(\lambda) + \text{negl}(\lambda) + \text{negl}(\lambda) \leq \text{negl}(\lambda)$.

Therefore;

$$|\Pr[\mathbf{Game}_{\mathcal{A}}^{untrace}(1^{\lambda}) = 1] - 1/2| \leq \text{negl}(\lambda)$$

□

F *zkRevoke*: Microbenchmarks

Table 6: The Cost of Storing Tokens in Smart Contract

Number of Tokens	Size	Gas
1	32 B	21993
10	320 B	26591
100	3200 B	72443
1000	32 KB	553240
10000	320 KB	5339501
100000	3200 KB	53201326

Table 6 presents the cost of storing tokens in the Ethereum Blockchain. Ethereum blockchain network limits the maximum gas used by a transaction to 30 million [61] units. Due to this limitation, it is possible to store only 1200 tokens per block in the smart contract since it requires close to 30 million units of gas, the allowed gas limit per block. Thus, storing more than 1200 tokens would require sending multiple transactions, each requiring a new block. Storing a single token (32 bytes) consumes 21993 units of gas.

Our ZK circuit consists of 9315 constraints when $k = 1$. The size of our circuit is 89667 bytes. Similar to the limit on the number of tokens stored, storing the complete ZK circuit would require multiple transactions.