Practical Privacy-Preserving Revocation of Verifiable Credentials

***Abstract*—Self-Sovereign Identity (SSI) has gained significant momentum across both private and public sectors, exemplified by initiatives like EBSI and eIDAS 2.0. At the core of SSI Systems are Verifiable Credentials (VCs), which substitute paper-based credentials, such as passports and diplomas, in a tamper-evident manner. Revocation is a necessary operation for VCs, driven by reasons such as the introduction of updated versions or the misuse of credentials. However, given that the VC status is accessible, mitigating the risk of sensitive information leakage while maintaining acceptable performance levels poses a significant challenge for revocation. Although existing revocation techniques satisfy some privacy requirements, no technique satisfies all and many have non-negligible computational and communication overheads that may render the technique impractical. In this article, we identify crucial privacy requirements for revocation of VCs and propose a revocation technique that satisfies all identified privacy requirements without imposing significant overheads. Our solution combines Bloom Filters and Merkle Tree Accumulators. The experiments validate the practicality of our revocation technique, demonstrating significantly lower computational and communication overheads compared to a widely adopted revocation technique.**

***Index Terms*—Verifiable Credential, Revocation, Privacy, SSI**

I. Introduction

Self-Sovereign Identity (SSI) [1]–[3] is gaining huge momentum in the industry and is increasingly getting adopted by different government organizations. For example, the European Blockchain Services Infrastructure (EBSI) [4], an initiative of the European Commission, is actively developing SSI-based systems to enhance citizen identification and facilitate the sharing of official records across European countries. A core element of SSI systems is the use of Verifiable Credentials [5]. These credentials represent traditional physical credentials (e.g., passports, diplomas, driving licenses) in a digital format that is both cryptographically secure and machine-verifiable. For example, in the context of EBSI, the European authorities (***Issuers***) may issue documents such as diplomas and driving licenses to citizens (***Holders***) in the form of Verifiable Credentials (VCs) [5]. Holders can then share VCs with banks and universities (***Verifiers***).

Revocation of VCs is one of the crucial operations in SSI Systems. For example, driving licenses of citizens issued in the form of VCs need to be revoked due to driving offenses. In such cases, verifiers should be able to verify the revocation status of VCs. The revocation status of VCs are commonly stored by issuers as cryptographic proofs in public ledgers, such as Distributed Ledger Technologies (DLTs), ensuring availability to verifiers. However, this raises privacy-related challenges, as sensitive information about holders and issuers may leak when comparing revocation status updates in the public registry.

Therefore, it is essential to understand the privacy requirements for VC revocation techniques. So far, existing revocation techniques [6]–[8] focused on the performance and functionalities of VCs on SSI systems, leaving privacy under-explored. In this work, we identify three privacy requirements that must be satisfied by revocation techniques. Although existing techniques satisfy some of these requirements, no technique satisfy all and many have non-negligible computational and communication overheads that may rend the technique impractical.

To close this gap, we propose a revocation technique that satisfy all three privacy requirements, while avoiding the significant overheads of previous proposals. The proposed technique combines a Bloom Filter and a Merkle Tree Accumulator (MTAcc), which are stored and managed in a Distributed Ledger Technology (DLT) through Smart Contracts. Storing revocation status in a Bloom Filter satisfies the privacy requirements; however, false positives may occur. To address this, we combine Bloom Filter and MTAcc in two steps. First, a verifier checks if the VC is in the Bloom Filter. (If not, the VC is not revoked. If yes, a second check is necessary to address potential false positives). In the second phase, the verifier checks if the VC is included in the MTAcc.

Due to the low rate of false positives in the Bloom Filter, and the structure of the MTAcc, the frequency of witness updates during revocation is significantly reduced when compared to popular solutions, such as cryptographic accumulators [9]. Furthermore, to reduce storage overheads and increase privacy guarantees, the MTAcc is only partially stored in the DLT – lower levels of the tree can be omitted.

To evaluate our proposals, we initially conduct a privacy analysis, examining how the identified privacy requirements are met. Then, we describe a series of experiments conducted on the Ethereum testnet. The goal of these experiments is to show evidence of the feasibility of the proposed technique, in terms of computational and communication overheads, in contrast to cryptographic accumulators, the revocation technique employed in Hyperledger Indy, a widely used SSI system. The results show that our technique performs better than Hyperledger Indy's technique with respect to witness updates. Our technique results in orders of magnitude fewer witness updates than Hyperledger Indy's technique because the number of witness updates in our technique is directly proportional to the number of false positives. The main contributions of this paper include:

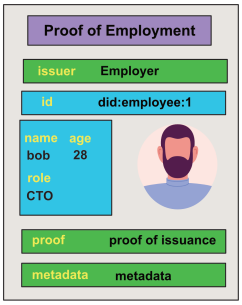


Fig. 1: Example of VC, an employee identification

1) Identification of three important privacy requirements that are not all satisfied by existing revocation techniques. To the best of our knowledge, this work is the first to discuss privacy in VC revocation.
2) Proposal of a practical revocation technique that satisfies all identified privacy requirements.
3) Experimental results validating the feasibility of the proposed technique. Additionally, we discuss some trade-offs between revocation technique and its associated overheads.

The remaining sections of this paper are organized as follows. Section II provides the necessary background. Section III discusses revocation privacy requirements. Our revocation technique is presented in Section IV and evaluated in Section V. Section VI covers related work, and Section VII concludes the paper.

II. Background

A. Distributed Ledger Technology (DLT) and Smart Contracts

A Distributed Ledger Technology (DLT) [10] is a peer-to-peer network that maintains a replicated copy of an append-only ledger (commonly known as blockchain). Prominent examples of DLTs include Bitcoin [11] and Ethereum [12]. DLTs provide robust availability because it may be composed by thousands of nodes. Illustratively, as of Oct 16, 2023, Ethereum boasted an impressive $7,054$ active nodes [13].

Smart Contracts [14], [15] are computer programs stored on DLTs. Smart Contracts encode state and a set of procedures to update the state. In Ethereum [12], to update Smart Contract states, users need to send transactions to miners. Sending transactions requires fees to be paid to the miners who execute the transactions. Reading Smart Contract states do not require transactions, therefore it is performed free of cost.

B. Verifiable Credential (VC)

Verifiable Credentials (VCs) [5] serve as digital representations of conventional credentials, including passports, diplomas, and driving licenses. They are designed to be both secure and machine-verifiable. An example of a VC is depicted in Fig. 1, showcasing details about an employee. It contains a digital signature of the issuer, which is used for the verification of authenticity and integrity.

C. Bloom Filters

A Bloom Filter [16], [17] is a space-efficient probabilistic data structure used to perform set membership tests. A Bloom Filter represents a set of r elements, $S = \{x_1, x_2, \cdots, x_r\}$, using an array of m bits. Each element in the set S is represented using k bits in the Bloom Filter, which are generated using k hash functions, H_i, $1 \leq i \leq k$. The hash functions map elements in the set S to random uniform numbers in the range $1, \cdots, m$. A Bloom Filter is initialized with 0 in all bits. An element $x \in S$ is inserted into a Bloom Filter by setting to 1 the bits in indexes $H_1(x), \cdots, H_k(x)$. A set membership test for x checks the same k indexes. If all the k bits are set to 1, then the x is present in the Bloom Filter, otherwise it is not. Bloom Filters take constant time, $\mathcal{O}(k)$, for insertion and querying elements and $\mathcal{O}(m)$ space to represent S.

Two properties of Bloom Filters are relevant in the context of this work. There are no false negatives: if the membership test indicates that $x \notin S$, then x was assuredly not inserted in the Bloom Filter. However, false positives may occur with known probability: the membership test might erroneously indicate that y is in the Bloom Filter, even if it has not been inserted. This error arises from the possibility of collisions among indexes.

D. Merkle Tree Accumulators (MTAccs)

Merkle Tree Accumulators (MTAccs) [18] use trees to represent a set of n elements $S = \{x_1, x_2, \cdots, x_n\}$. In MTAccs, the leaf nodes represent cryptographic hashes of the set elements. Each non-leaf node stores the hash digest of concatenated values of its children. The root hash is the value of the accumulator.

For each element added in the accumulator, the witness consists of the element's sibling's hash digest and hash digests in the co-path from the root to the element's leaf. The verification of the membership test reconstructs the path from the leaf to the root and compares the calculated root hash with the value of the accumulator. If both match, it means that the element is present in the accumulator; otherwise, it is not. MTAccs take $\mathcal{O}(\log n)$ for insertion and membership tests, and $\mathcal{O}(n)$ space to represent a set.

Two properties are relevant in the context of this work. First, the membership test is deterministic: assuming no collisions in the employed hash function, no false negatives or positives may occur. Second, whenever a MTAcc is updated, all the elements need to update their witnesses.

Fig. 2 shows a MTAcc representing $S = \{x_1, x_2, x_3, x_4\}$. The witness of x_1 is $(h_2, h_{3,4})$. To prove that x_1 is present in the MTAcc, x_1 presents its hash digest, h_1, and the witness. To verify the membership, the path from h_1 to root is recalculated: $root' = H(H(h_1||h_2)||h_{3,4})$. If $root'$ is the same as the root then it means that x_1 is present in the accumulator, otherwise it is not. If an element is updated, then all the elements need to update their witnesses.

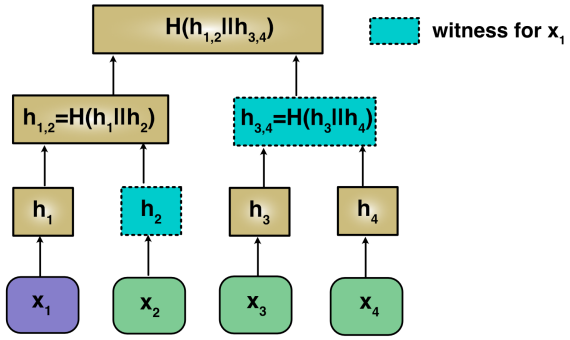


Fig. 2: Merkle Tree Accumulator example

E. SSI and Use Case

Self-Sovereign Identity (SSI) [2], [3] emphasizes that individuals should own and control their data without the involvement of any central authorities. IOTA Identity [19] and Hyperledger Indy [9] are examples of some of the systems that follow the SSI philosophy. In this paper, we refer to the systems that follow the SSI principles as Self-Sovereign Identity Systems (SSI Systems). SSI Systems utilize Verifiable Credentials (VCs). The are four different entity classes in SSI Systems: 1) issuers, 2) holders, 3) verifiers, and 4) registry. Issuers issue VCs to holders. Holders receive VCs from issuers and share them with verifiers. Verifiers verify the VCs shared by the holders. Registry assists with the issuance and verification of VCs by storing different information such as metadata and proofs.

A popular use case for SSI in the industry is the Workforce Identity [20]–[22]. We refer to, and extend, this use case throughout this work in order to motivate our proposals. The use case consists of Bob, who works at Employer 1, applying for a credit card from a Bank. Employer 1 issues an Identity card as a VC, which works as a proof of employment for Bob. Bob presents this VC to a Bank to apply for a credit card. The Bank verifies it to confirm whether Bob is indeed currently working at Employer 1. Once the Bank verifies the VC, the credit card can be provided to Bob. In this use case, Bob is a holder, Employer 1 is an issuer, and the Bank is a verifier. The registry can be a public DLT [23], such as Indy Ledger [9], or a centralized server managed by the Employer 1. This scenario is illustrated in Fig. 3.

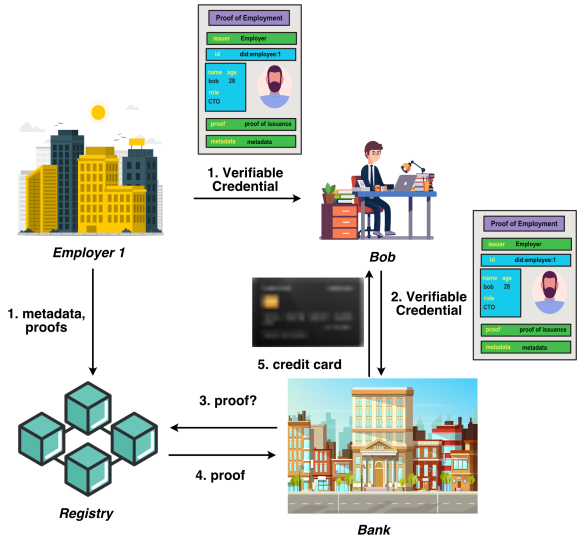


Fig. 3: SSI use case: workforce identity

III. SSI Systems: Revocation Privacy Issues

A Verifiable Credential (VC) can be revoked by various reasons, including: *i*) canceling, *ii*) updating, and *iii*) private key loss. The first scenario arises, for instance, when an employee leaves an organization, prompting the issuer to promptly invalidate the corresponding VC. The second reason prevents holders from using VCs with outdated attributes. For example, a malicious employee might exploit an old VC to potentially reduce tax payments. The third situation can occur for issuers or holders, where the loss of a private key fundamentally compromises system trust and may necessitate the revocation of VCs and identities. We consider only privacy-related requirements in this paper.

The issuing and the revocation process store information in a registry (DLT-based or not) to enable the verifiers to check the status of VCs. Examining this information can potentially compromise the privacy of either holders or issuers.

In this section, we define three Privacy Requirements (PR) and articulate their significance. Additionally, we delve into the impact of the requirement in the context of the Workforce Identity use case (Section II-E) using the scenario of Bob applying for a job at Employer 2. Fig. 4 visually represents the three identified privacy requirements, which are further described next.

PR1 - Concealing the association between Holders and Verifiers from Issuers

The association between holders and verifiers should not be learned by issuers. Whenever holders share VCs with verifiers, verifiers would verify the authenticity and integrity of the shared VCs. If verifiers contact issuers for verification, then issuers would learn the association. This problem of verifiers contacting issuers directly is also referred to as “Phone Home” problem [24]. By uncovering the association between holders and verifiers, issuers can model the behavior of holders and track the services accessed by them.

For example, in the Workforce Identity use case, if Employer 2 contacts Employer 1 to check the current employment status of Bob, then Employer 1 would learn that Bob is actively seeking other opportunities. This knowledge could potentially impact the working conditions of Bob. Hence, revocation techniques should not involve the interaction with issuers for verification of VCs.

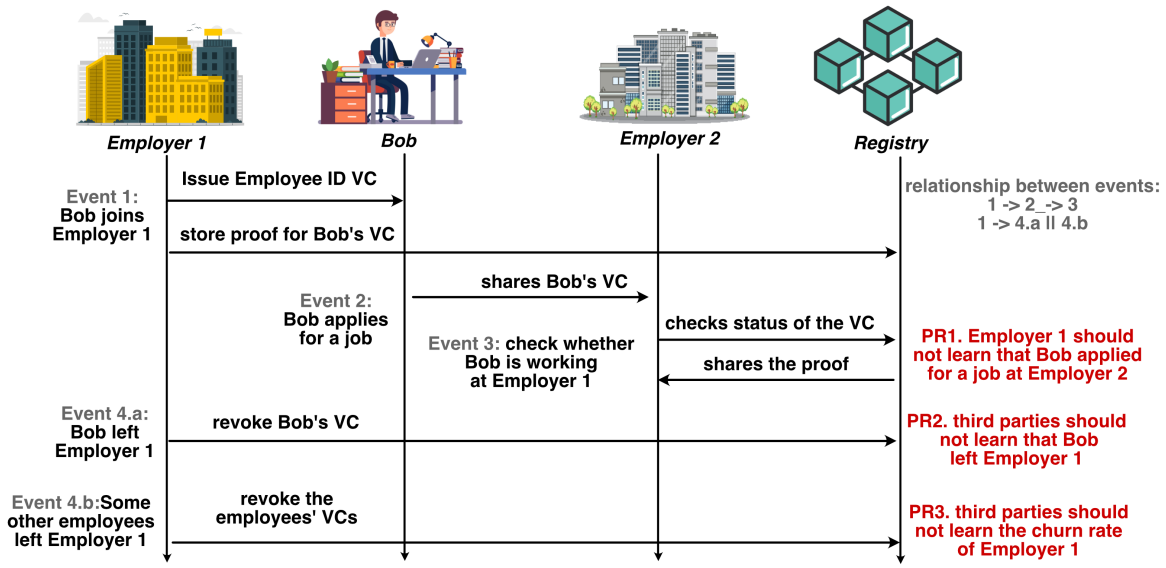


Fig. 4: Revocation privacy requirements

PR2 - Preventing leakage of Holders' Information

Revocation should not enable third parties to learn whose VC got revoked. Depending on how the revocation is performed, revocation could leak information such as unique identifiers. Leaking personal or sensitive information makes it possible to trace it back to the corresponding holders. We assume that the third parties do not have access to VCs, they only have read access to registry. The goal of this requirement is to ensure that the anonymity of holders is preserved.

For example, in the Workforce Identity use case, Employer 1 revokes Bob's VC once he resigns. If the private information of Bob such as a unique identifier is leaked by the revocation technique, then third parties can learn that Bob left Employer 1. If Bob is a crucial engineer in a very important project or an import executive, then revealing this information could affect the market share of Employer 1. Therefore, it is essential that revocation techniques should not reveal any private information about holders.

PR3 - Preventing leakage of revocation rate

The requirement is that revocation techniques should not leak the revocation rate. Revocation rate denotes the number of VCs revoked within a specific time period. Observing the revocation rate could lead to the disclosure of business-critical information. We assume that the third parties do not have access to VCs, they only have read access to registry.

For example, in the Workforce Identity use case, by observing the revocation rate, one could learn the employee churn rate. Leakage of the churn rate could have severe consequences for Employer 1, potentially indicating poor working conditions, a negative environment, and ineffective management.

IV. Proposal: Two-phase technique

As identified in Section III, revocation and verification of VCs may lead to privacy violations depending on how entities interact with issuers and what information is stored in proofs. We propose a Two-phase technique that satisfy the three identified privacy requirements. Furthermore, the proposed technique leverage data structures that limit the number of witness updates upon each VC revocation.

The Two-phase technique utilizes Bloom Filter, Merkle Tree Accumulators (MTAccs), and Smart Contract. The revoked VCs are inserted into these data structures and stored at a DLT using Smart Contract. During verification, Bloom Filter is checked first. If Bloom Filter results in false positives, MTAcc is checked. Holders retrieve the updated witnesses for MTAccs only for the false positive cases.

We assume only one issuer in our proposal, but it is possible to extend to multiple issuers by creating distinct instances of the Smart Contract for each issuer. Furthermore, we assume the use of a public DLT as a registry, which may incur fees, such as the Ethereum Network. However, the proposal could also be implemented using feeless DLTs (e.g. private/permissioned deployments). In this case, the considerations about the costs of operations may be disregarded.

Our proposal comprises five procedures: 1) setup, 2) issuance, 3) revocation, 4) witness update, and 5) verification. Setup is only performed once to bootstrap the required structures. Issuance issues new VCs. Both revocation and verification are performed for VCs that are already issued. The witness update procedure is called only during revocation and verification, to update the witnesses of VCs. The pseudo-code for the procedures are presented in Algorithms 1 to 5. Bloom Filter is represented as *bloomFilter* in the pseudo-code.

A. Procedures

1) Setup: This procedure initializes the necessary data structures at both issuer and Smart Contract. First, the issuer configures Bloom Filter and MTAcc. Bloom Filter is used to store only revoked VCs, whereas MTAcc stores both revoked and valid VCs. *n* represents the estimated total number of VCs that the issuer would issue. *r* represents the estimated number of VCs that would be revoked.

Algorithm 1 Two-phase technique: Setup

```
1:  procedure SETUP
2:      Issuer:
3:          choose n,m,k,z
4:          choose k hash functions H_i, 1 ≤ i ≤ k, 1 ≤ H_i ≤ m.
5:          h ← log_2 n
6:          leafIndex ← 0
7:          bfIndexStore ← hashtable[string][int × k]
8:          mtIndexStore ← hashtable[string][int]
9:          witStore ← hashtable[string][string × h]
10:         leafs ← hashtable[int][string]
11:         bloomFilter ← zeroed array of m bits
12:         MTAcc ← empty Merkle Tree Accumulator
13:     Smart Contract:
14:         store empty bloomFilter                ▷ size: m
15:         store empty MTAcc                      ▷ only z levels
```

The following parameters are specific to Bloom Filter: 1) p, 2) m, and 3) k. p denotes the false positive rate [17], which quantifies the probability of any membership test query incurs a false positive error. m represents the capacity of the Bloom Filter in bits. The optimal value for m can be configured depending on the values of r and p. As described in [16], $m = -\frac{r \ln p}{(\ln 2)^2}$ can be used. k denotes the number of hash functions where the optimal value of $k = (m/r) \ln 2$. The issuer then initializes an empty hash table [25] called *bfIndexStore*. The *bfIndexStore* stores the Bloom Filter indexes assigned to VCs. The *bfIndexStore* is used during the issuance and revocation of VCs to assist Bloom Filter.

The following parameter is specific to MTAcc: z. z denotes the number of levels of the MTAcc that the issuer would store in the Smart Contract (root node starts at level 0). The purpose of storing only z levels instead of the complete tree is to address PR3. We will explain it in detail when we analyze how this revocation technique addresses the privacy requirements. The issuer then calculates the height (h) of the MTAcc from n using $h \leftarrow log_2 n$. h is used in the WitnessUpdate procedure. The issuer initializes *leafIndex* to 0. The *leafIndex* identifies the position of VCs in MTAcc; this simplifies the processes of insertion, revocation, and computation of their witnesses. The issuer also initializes three hash tables: 1) *leafs*, 2) *mtIndexStore*, and 3) *witStore*. The *leafs* hash table stores the vcID values of VCs using *leafIndex* values as keys. The *mtIndexStore* stores the *leafIndex* values of VCs using vcID values as keys. The *witStore* stores the witnesses of VCs using vcID values as keys.

Once both Bloom Filter and MTAcc are configured, then issuer initializes an empty Bloom Filter and an empty MTAcc in Smart Contract. Storing in Smart Contract is crucial because verifiers query the Smart Contract to verify revocation status of VCs. We deploy the Smart Contract in Ethereum [12] DLT. Therefore, Smart Contract has public read access. However, it is possible to use other DLTs such as Solana [26] or Hyperledger Fabric [27]. The purpose of using Smart Contract is to address PR1, so that verifiers do not need interaction with the issuer for verification. The data structures stored at the issuer are used only for the purpose of issuance and revocation of VCs, not for the purpose of verification by verifiers.

Algorithm 2 Two-phase technique: Issuance

```
1:  procedure ISSUANCE
2:      Issuer:
3:          vcID ← randString()
4:          bfIndexes ← H_1(vcID), ···, H_k(vcID)
5:          bfIndexStore[vcID] ← bfIndexes
6:          leafs[leafIndex++] ← vcID
7:          mtIndexStore[vcID] ← leafIndex
8:          Insert H(vcID) at leafIndex in MTAcc
9:          WitnessUpdate(leafIndex)
10:     Smart Contract:
11:         store updated MTAcc
```

2) Issuance: This procedure is used by the issuer to issue new VCs to holders, issuing one VC at a time. We restrict the scope to revocation-related data. We do not discuss the details related to VCs themselves, e.g., employee name in the workforce identity use case. For a new VC, the issuer generates a unique identifier called "vcID" which distinguishes itself from other VCs.

The issuer generates Bloom Filter indexes whenever a new VC is issued. The issuer uses the vcID as a key for the VC in the *bfIndexStore* and stores the Bloom Filter indexes. It is important to note Bloom Filter do not need to be updated in the Smart Contract.

The issuer inserts the hash digest of vcID in the MTAcc at the position denoted by *leafIndex*. Before inserting in the MTAcc, *leafIndex* is incremented by 1. Once the MTAcc is modified, the issuer updates witnesses of the VCs that are affected by this insertion by calling the witnessUpdate(*leafIndex*) procedure with *leafIndex* as argument. Moreover, the procedure also calculates the witness for the newly inserted VC. Finally, the issuer publishes the first z levels of the updated MTAcc to the Smart Contract.

3) Revocation: This procedure is used by the issuer to revoke already issued VCs. It revokes one VC at a time. For a VC, the issuer first fetches its Bloom Filter indexes from the *bfIndexStore* using its vcID as a key. Then the issuer updates its Bloom Filter by setting the bits in those indexes to 1. The issuer then updates the MTAcc. The issuer retrieves *leafIndex* of the revoked VC from the *mtIndexStore*. Then, the issuer replaces the hash digest stored at the position *leafIndex* in

Algorithm 3 Two-phase technique: Revocation

```
1:  procedure REVOCATION(vcID)
2:      Issuer:
3:        bfIndexes ← bfIndexStore[vcID]
4:        for i ∈ bfIndexes do
5:            bloomFilter[i] ← 1
6:        leafIndex ← mtIndexStore[vcID]
7:        Insert H(randString()) at leafIndex in MTAcc
8:        WitnessUpdate(leafIndex)
9:      Smart Contract:
10:       store updated MTAcc and bloomFilter
```

the MTAcc with a hash digest of a random string. Then the issuer updates the witnesses of the affected VCs by calling witnessUpdate(*leafIndex*).

Finally, the issuer stores updated Bloom Filter and MTAcc in Smart Contract. It is important to note that the issuer only publishes the first z levels of the updated MTAcc to the Smart Contract. Once the updated MTAcc is published, the witness corresponding to the revoked VC becomes invalid.

Algorithm 4 Two-phase technique: Witness Update

```
1:  procedure WITNESSUPDATE(leafIndex)
2:      Issuer:
3:        for (i ← 1; i ≤ n − 2^{h−z}; i ← i + 2^{h−z}) do
4:            if leafIndex < i + 2^{h−z} then
5:                for (j ← i; j ≤ 2^{h−z}; j ← j + 1) do
6:                    witStore[leafs[j]] ← MTAcc.Witness(j)
7:                return
```

4) Witness Update: This procedure updates the witnesses of the VCs that are affected by MTAcc update. Whenever the MTAcc is modified due to issuance or revocation of a VC, VCs that are in co-path from that VC to level z are affected by that change. Therefore, the affected VCs need to update their witnesses. We identify the affected VCs using the *leafIndex* value of the VC, which is passed as an input to this procedure. The affected VCs and that VC have the same parent at level z and thus, they belong in the same sub-tree of non-leaf node at level z. There are 2^z non-leaf nodes at level z, where each non-leaf is a parent of 2^{h-z} leaf nodes. Therefore, we go over the index values of leaf nodes in blocks of size 2^{h-z}. The block that contains the *leafIndex* value is the one that consists of the affected VCs. Then, we recalculate the witness of all the leaf nodes in that block and update the *witStore*. The updated witness needs to be communicated to the corresponding holders.

For example, if issuer expects to issue 1 million VCs, then $h = 20$. For $z = 10$, each issuance and revocation only requires notification of $2^{20-10} = 1024$ holders.

5) Verification: This procedure is used by verifiers to verify the revocation status of already issued VCs. This procedure verifies one VC at a time. Figure 5 presents the interaction between different entities in the verification procedure.

Algorithm 5 Two-phase technique: Verification

```
1:  procedure VERIFICATION
2:      Holder: send bfIndexes to Verifier
3:      Verifier:
4:        status ← send bfIndexes to Smart Contract
5:      Smart Contract:
6:        status ← revoked
7:        for i ∈ bfIndexes do
8:            if bloomFilter[i] = 0 then
9:                status ← valid
10:               break
11:       return status
12:     Verifier:
13:       if status = valid then
14:           return
15:       if status = revoked then
16:           retrieve MTAcc from Smart Contract
17:           request witness from Holder
18:     Holder:
19:       wit ← Fetch witness from Smart Contract
20:       witness ← witness + wit
21:       send witness to Verifier
22:     Verifier:
23:       verify the witness in MTAcc
```

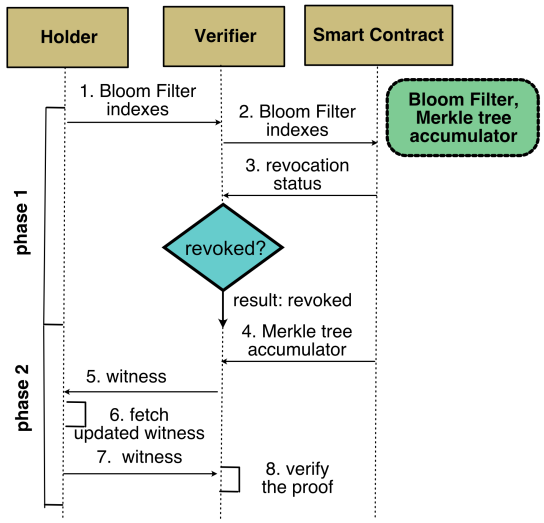


Fig. 5: Two-phase verification

First, a holder sends the Bloom Filter indexes to a verifier. Then, the verifier sends the Bloom Filter indexes to the Smart Contract. The Smart Contract checks it using the Bloom Filter and returns the revocation status to the verifier. If the revocation status is revoked, then the verifier performs phase 2. The verifier requests a witness from the holder.

The holder fetches from the witness from the Smart Contract, if there is one or more modifications to MTAcc that did not affect the holder in the corresponding witness updates. For example, in Figure 2, if only root and level 1 is stored in the

Smart Contract, a holder corresponding to x_1 has to fetch $h_{3,4}$ from the Smart Contract. The holder now possesses complete witness since h_2 must have been given by the issuer. The holder then sends the witness to the verifier. Then, the verifier retrieves the root of the MTAcc from the Smart Contract and verifies the witness by computing the root. If the computed root is the same as the one stored in the Smart Contract, the witness is valid indicating the VC is valid, otherwise, it is not.

B. Discussion

We discuss the trade-offs of the proposed revocation technique below.

Advantages: The first advantage is that Bloom Filters never result in false negatives. If a VC is valid (i.e. not included in the Bloom Filter), then it is not necessary to perform phase 2. The second phase is only required in case a VC is deemed invalid by the Bloom Filter (i.e. included). Since the false positives can be controlled, only very few valid VCs need to retrieve updated witnesses. This benefits use cases in which few VCs are revoked: most verifications will only require the first phase. In Section V-B, we discuss the experimental results and also compare them against the revocation technique used in Hyperledger Indy [9].

The second advantage is that the frequency of witness updates is reduced by a huge margin. Phase 1 ensures that only VCs that are included in the Bloom Filter (and thus may configure a false positive) need to retrieve witnesses. In addition, since we store z levels of MTAcc in Smart Contract, each update to the MTAcc only affects 2^{h-z} VCs. Therefore, VCs that are not affected by the MTAcc updates, can directly fetch their witnesses from Smart Contract instead of contacting the issuer. This minimizes the reliance on issuers.

The third advantage is the **cost-free verification** process since the verification procedure only performs read operations from Smart Contract. Reading the existing Smart Contract state does not require a fee in Ethereum. Therefore, verifiers do not need to pay any fee to read the Bloom Filters and MTAccs values.

The fourth advantage is the availability of revocation and verification due to the usage of Smart Contract. Deploying the Smart Contract to the public Ethereum network ensures very good availability since Ethereum $7,054$ active nodes [13]. Therefore, issuer does not need to be available all the time to service the verification requests.

Drawbacks: First, the issuance of VCs requires an update in the Smart Contract since the MTAcc needs to be updated. Therefore, issuance of VCs require fees. This solution is efficient because only holder whose VCs got a false positive result in phase 1 needs to retrieve witnesses from Smart Contract. However, the cost of both issuance and revocation is increased since both Bloom Filters and Merkle Tree Accumulators need to be updated. Second, the witness update frequency is reduced but not completely removed. Therefore, holders need to communicate with the issuer from time to time to update their witnesses. Third, storing more levels of MTAcc in Smart Contract may be expensive due to fees.

V. Evaluation

A. Privacy Analysis

In this section, we explain how the proposed solution address all three privacy requirements, outlined in Section III.

PR1 is satisfied because for the verification of VCs, verifiers only read existing state from Smart Contract. Reading state from the Smart Contract does not create a transaction, therefore, it is not recorded on-chain. It is sufficient to contact a single DLT node. Therefore, the issuer does not learn the association between holders and verifiers.

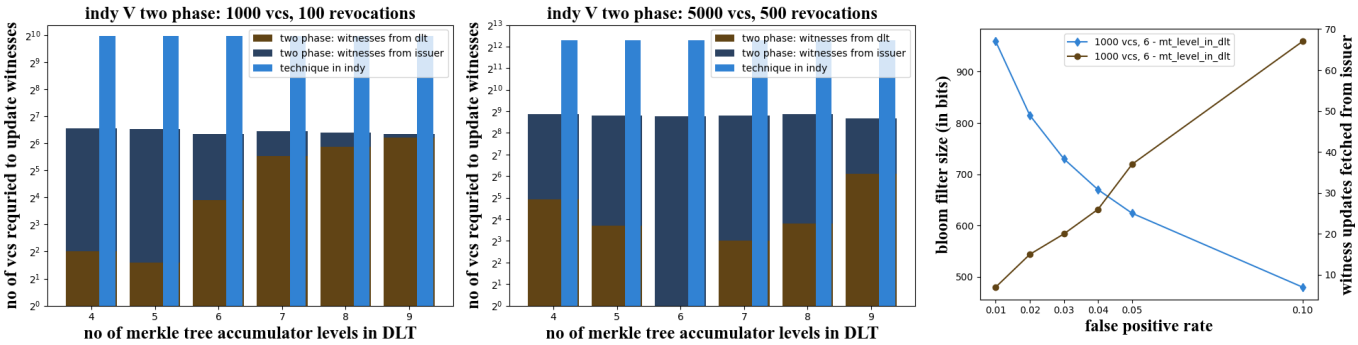
PR2 is satisfied because no personal or sensitive information of holders is stored in the Smart Contract. Both Bloom Filter and Merkle Tree Accumulator do not reveal holders because it is nontrivial and expensive to calculate vcIDs due to the one-way [28] property of hash functions. Bloom Filter indexes are hash digests of vcIDs, whereas MTAcc consists of a tree of hash digests of vcIDs.

PR3 is satisfied due to a) the probabilistic nature of Bloom Filters, and b) storing only z levels of MTAcc in Smart Contract. In Bloom Filters, the indexes are assigned to more than one VC. Therefore, Bloom Filters do not directly reveal how many VCs are revoked. To prevent adversaries who observe each update in the Smart Contract from compromising PR3, revocation of VCs should be performed in batches. In the MTAcc, each node at level z is a parent of 2^{h-z} VCs. Therefore, observing changes in the Smart Contract, adversaries can only estimate anywhere from one to 2^{h-z} VCs might be revoked. This can be further optimized by performing revocation in batches. If all nodes at level z are updated, then adversaries' predictions would even get worse since it might mean all n VCs might be revoked.

B. Experimental Results

Our experimental setup consists of: Smart Contract is deployed on private DLT using Ganache v7.9.1 [29]. We run a single issuer who issues and revokes VCs. We run a separate program responsible for synthetically generating the workload (issuance and revocations), and also for collecting results. In this section, we present the results and compare them against the revocation technique used in Hyperledger Indy [9]. Hyperledger Indy provides tools and libraries to create and deploy SSI Systems. Some of the popular deployments are Sovrin [30], IDUnion [31], and Indicio [32]. Hyperledger Indy's revocation technique is based on Bilinear accumulator [18]. The anonymized source code of our solution is available at: https://anonymous.4open.science/r/Revocation-Service-C557.

The metric we use for comparison is the number of witness updates performed by holders. This metric quantifies the objective of the two-phase technique, which is reducing the frequency of witness updates performed by holders. We present the results of the comparison in Fig. 6a and Fig. 6b. In Fig. 6a, we use the following configuration: $n = 1000, r = 100, p = 0.1$, where n denotes total number of issued VCs, r denotes total number of revoked VCs, and p denotes false positive



(a) Technique in Indy vs. two-phase technique (b) Technique in Indy vs. two-phase technique (c) Impact of false positives

Fig. 6: Experimental results

rate. Fig. 6b show results for $n = 5000, r = 500, p = 0.1$. Additionally, we performed experiments storing from 4 to 9 levels of MTAcc in the Smart Contract, as indicated in the horizontal axis of each figure. Each experiment was repeated three times, and we report just the averages (results were consistent, with low standard deviation). In each experiment, we first issue *n* VCs, then revoke *r* random VCs, and finally we check the revocation status of all VCs. Then we calculate the number of valid VCs that update witnesses. In both figures it is possible to see that our technique results in orders of magnitude fewer witness updates than Hyperledger Indy's technique as the number of witness updates in our technique is directly proportional to the number of false positives. It shows that many of the witnesses are obtained directly from Smart Contract instead of fetching it from issuer as we store more levels of Merkle Tree Accumulator in Smart Contract.

We also present the impact of p on witness updates in Fig. 6c. We use the following configuration: $n = 1000, r = 100$. We store 6 levels of MTAcc in Smart Contract. The results indicate that smaller values for p reduce the number of witnesses fetched from issuer. It is because smaller values for p result in fewer false positives and thus, the number of witness updates is minimal. The trade-off is the Bloom Filter size since it is configured based on r and p, as in Section IV.

The cost associated with the Smart Contract is not discussed since it is used only for storing and retrieving Bloom Filters and MTAccs, and does not perform any heavy computation.

VI. Related work

In this section, we compare our contributions against related work. In particular, 1) we check whether related work describe privacy requirements for revocation of VCs, and 2) we analyze revocation techniques in related work against our proposals.

EBSI [24] presents PR1 and PR2 in detail. Freitag [33] briefly discusses PR2. These two works also discuss other privacy requirements in revocation. However, PR3 is not discussed in any of these works. We are the first to introduce and discuss PR3. In addition, we are the first ones to propose a revocation technique to address privacy requirements. Revocation techniques discussed in related work are designed to address functional requirements, not privacy.

Schumm *et al.* [8] propose a technique based on Bloom Filters for revocation. The technique records revoked VCs in Bloom Filters, which is stored in a DLT. This solution addresses the three privacy requirements. However, it has a significant drawback. The drawback is that they did not address false positives and thus, would incorrectly classify a good number of valid VCs as revoked. Therefore, this solution can be only applicable to use cases that accept the probabilistic revocation status of VCs.

Hyperledger Indy uses bilinear cryptographic accumulators [7] to revoke VCs and store the accumulator in a DLT. This solution addresses PR1 and PR2, but it fails to address PR3 because it stores the list of witnesses of revoked Verifiable Credentials in DLT. Storing this list makes it easier for holders to update their witnesses, however it reveals the revocation rate. In addition, bilinear accumulators are much more computationally expensive than Bloom Filters and Merkle Tree Accumulators.

IOTA Identity [6] uses bitmaps to record revoked VCs and store them in DLT. This solution is very efficient in terms of storage and verification compared to related work and our techniques and satisfies PR1 and PR2. However, it fails to address PR3 because it is possible to count the number of revocations just by observing the changes in bitmaps.

VII. Conclusion and Future Work

In this work, we identified privacy requirements for revocation of VCs and proposed a two-phase technique based on Bloom Filters and Merkle Tree Accumulators that satisfies the identified requirements. Experimental results demonstrate the practical viability of our approach over widely used solutions.

For future work, we believe that finding an optimal number of levels of Merkle Tree Accumulators to store in a Smart Contract would be a compelling research direction. Furthermore, there may be more privacy issues not addressed by existing solutions. For instance, it is currently possible for a verifier to indefinitely monitor the revocation status of a VC it has received from a holder. Therefore, exploring additional privacy requirements and their respective solutions would be a captivating area for further research.

References

[1] M. S. Ferdous, F. Chowdhury, and M. O. Alassafi, "In search of self-sovereign identity leveraging blockchain technology," *IEEE Access*, vol. 7, pp. 103 059–103 079, 2019.

[2] M. R. Ahmed, A. K. M. M. Islam, S. Shatabda, and S. Islam, "Blockchain-based identity management system and self-sovereign identity ecosystem: A comprehensive survey," *IEEE Access*, vol. 10, pp. 113 436–113 481, 2022.

[3] Š. Čučko, Š. Bećirović, A. Kamišalić, S. Mrdović, and M. Turkanović, "Towards the classification of self-sovereign identity properties," *IEEE Access*, vol. 10, pp. 88 306–88 329, 2022.

[4] "European blockchain services infrastructure (ebsi)," https://ec.europa.eu/digital-building-blocks/wikis/display/EBSI/Home, accessed: May. 10, 2023.

[5] M. Sporny, D. Longley, and D. Chadwick, "Verifiable credentials data model v2.0," https://w3c.github.io/vc-data-model/, accessed: April. 24, 2023.

[6] "Iota idenity framework: Revoke a verifiable credential," https://wiki.iota.org/identity.rs/how-tos/verifiable-credentials/revocation/, 2023, accessed: Dec. 14, 2023.

[7] "Indy - how revocation works?" https://hyperledger-indy.readthedocs.io/projects/sdk/en/latest/docs/concepts/revocation/cred-revocation.html, 2023, accessed: Dec. 14, 2023.

[8] D. Schumm, R. Mukta, and H.-y. Paik, "Efficient credential revocation using cryptographic accumulators," in *2023 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPS)*, 2023, pp. 127–134.

[9] "Hyperledger indy," https://www.hyperledger.org/use/hyperledger-indy, accessed: April. 25, 2023.

[10] A. Sunyaev, *Distributed Ledger Technology*. Springer International Publishing, 2020, pp. 265–299.

[11] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," https://bitcoin.org/bitcoin.pdf, 2008.

[12] "Ethereum," https://ethereum.org/en/, 2023, accessed: Oct. 16, 2023.

[13] "Etherscan," https://etherscan.io/nodetracker, 2023, accessed: Oct. 16, 2023.

[14] S. Wang, L. Ouyang, Y. Yuan, X. Ni, X. Han, and F.-Y. Wang, "Blockchain-enabled smart contracts: Architecture, applications, and future trends," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 49, no. 11, pp. 2266–2277, 2019.

[15] M. H. Tabatabaei, R. Vitenberg, and N. R. Veeraragavan, "Understanding blockchain: Definitions, architecture, design, and system comparison," *Computer Science Review*, vol. 50, p. 100575, 2023.

[16] S. Tarkoma, C. E. Rothenberg, and E. Lagerspetz, "Theory and practice of bloom filters for distributed systems," *IEEE Communications Surveys & Tutorials*, vol. 14, no. 1, pp. 131–155, 2011.

[17] L. Luo, D. Guo, R. T. B. Ma, O. Rottenstreich, and X. Luo, "Optimizing bloom filter: Challenges, solutions, and comparisons," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1912–1949, 2019.

[18] M. Loporchio, A. Bernasconi, D. Di Francesco Maesa, and L. Ricci, "A survey of set accumulators for blockchain systems," *Computer Science Review*, vol. 49, p. 100570, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1574013723000370

[19] "Iota identity," https://www.iota.org/solutions/digital-identity, accessed: April. 25, 2023.

[20] "Credivera," https://www.credivera.com/, accessed: Dec. 4, 2023.

[21] "Microsoft entra verified id," https://learn.microsoft.com/en-us/entra/verified-id/, accessed: Dec. 4, 2023.

[22] "Indicio proven works: Verifiable credentials for frictionless, secure employee identification," https://indicio.tech/indicio-proven-works-verifiable-credentials-for-frictionless-secure-employee-identification/, accessed: Dec. 4, 2023.

[23] M. Belotti, N. Božić, G. Pujolle, and S. Secci, "A vademecum on blockchain technologies: When, which, and how," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3796–3838, 2019.

[24] "Revocation by ebsi - ebsi's credential status framework and how to choose a revocation method when using w3c verifiable credentials (and more)," https://ec.europa.eu/digital-building-blocks/sites/download/attachments/693209363/2023, accessed: Oct. 13, 2023.

[25] W. D. Maurer and T. G. Lewis, "Hash table methods," *ACM Computing Surveys (CSUR)*, vol. 7, no. 1, pp. 5–19, 1975.

[26] "Solana," https://solana.com/, 2023, accessed: Dec. 16, 2023.

[27] "Hyperledger fabric," https://www.hyperledger.org/projects/fabric, 2023, accessed: Dec. 16, 2023.

[28] M. Naor and M. Yung, "Universal one-way hash functions and their cryptographic applications," in *Proceedings of the twenty-first annual ACM symposium on Theory of computing*, 1989, pp. 33–43.

[29] "Ganache," https://trufflesuite.com/ganache/, 2023, accessed: Dec. 18, 2023.

[30] "Sovrin," https://sovrin.org/, 2023, accessed: Aug. 18, 2023.

[31] "Id union," https://idunion.org/?lang=en, 2023, accessed: Dec. 18, 2023.

[32] "Indicio networks," https://indicio.tech/indicio-networks/, 2023, accessed: Dec. 18, 2023.

[33] A. Freitag, "A new privacy preserving and scalable revocation method for self sovereign identity - the perfect revocation method does not exist yet," Cryptology ePrint Archive, Paper 2022/1658, 2022, https://eprint.iacr.org/2022/1658. [Online]. Available: https://eprint.iacr.org/2022/1658