EPFL

Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

armasuisse
Science and technology

Scalable and Privacy-Preserving Revocation of Verifiable Credentials

Alessandro Colombo

School of Computer and Communication Sciences

Master's Thesis

September 2024

Responsible	**Supervisor**
Prof. Serge Vaudenay	Dr. Martin Burkhart
EPFL / LASEC	Cyber Defence Campus

CYD
CYBER
DEFENCE
CAMPUS

Abstract

Electronic Identification (e-ID) is being increasingly adopted worldwide, as it offers a fast and reliable method for remote identity verification. When building national e-ID systems, governments often adhere to the principle of Self-Sovereign Identity (SSI), which requires citizens to have *complete* control over their identity data. This goal is typically achieved with verifiable credentials, which are digital tokens signed by the issuer (e.g., the government) and containing the identity information of the respective credential holder. Verifiable credentials allow holders to selectively disclose the information they wish to share, and ensure that subsequent disclosures remain unlinkable.

In certain circumstances, governments may need to *revoke* some e-ID credentials, such as when the credential's hosting device is lost or stolen, in cases of criminal prosecution, or if the security of the issuer has been compromised. Popular list-based revocations approaches are not privacy-preserving, as they require the disclosure of *unique identifiers*, while unlinkable approaches are not practical enough for adoption in e-ID systems.

In this thesis, we address the challenge of revoking verifiable credentials by proposing a privacy-preserving revocation scheme based on cryptographic accumulators, designed to be scalable for national e-ID systems. The scalability of the proposed scheme is not only limited to the Swiss e-ID instance but could also be extended to multi-national e-ID systems, such as those in the European Union.

Contents

Chapter 1

Introduction

1.1 Motivation

In today's internet, the most widespread identification paradigm is Single Sign-On (SSO). This system allows users to access multiple services with a single set of credentials. A group of Identity Providers (idPs), often operated by *private* entities, provides authentication services to the entire world. Single Sign-On has reduced users "password fatigue", and shifted the responsibility of managing online identification from single service providers to idPs, which typically have larger budgets for securing their systems. However, centralizing the identification process places digital identities in the hands of idPs, who have the potential to monitor users online activities for *commercial* or *political* purposes.

Electronic Identification (e-ID) is a technology that enables *secure* and *reliable* verification of user identities. When developing national e-ID systems, governments often adhere[1] to SSI principles[2]. To align with these principles, e-ID credentials should grant users *complete control* over their own identity data, similar to how they control their physical identification documents, and do so without involving any *third-party* intermediaries.

In an increasingly digitalized world, e-ID can simplify access to a wide range of services, without requiring *in-person* identification. This includes government portals, electronic voting, digital driving license, and healthcare services. Reducing the need to present physical documents can have a broad impact on our society. In emergency situations, such as during pandemic crisis, a national e-ID offers a fast and reliable identification method, ensuring prompt access to essential services. Furthermore, it ensures that communities living in rural areas have access to the same digital services available to the urban population. The e-ID can also be used to establish trustworthy commercial interactions, providing online businesses with a foundational

[1]e.g., `https://www.eid.admin.ch/en/technology`

[2]`https://www.lifewithalacrity.com/article/the-path-to-self-soverereign-identity/`

identification infrastructure.

The European Union has been a pioneer in implementing digital identities, launching several pilot projects[3], and establishing a legal framework that regulates digital authentication across all its member states [Par14]. However, the EU's most recent technical proposal for digital credentials known as ARF 1.4.0 [Eur23], has been highly criticized in an open letter signed by some of the most prominent cryptographers and privacy experts working in the field [Bau+24]. Their primary concern is about the proposed credential schema, which is *linkable* both towards the *issuing entity* (e.g, the government), and the *relying entities* (e.g., online service providers). When unlinkability is not ensured, online businesses can act as "virtual stalkers", tracking users' activities and linking different pieces of disclosed information over time, progressively de-anonimyzing users identity. As a solution, the open letter advocate for the adoption of *anonymous credentials*, a type of verifiable credentials that is *unlinkable* by design.

The Swiss Federal Council plans to introduce a national e-ID by 2026, which should be available to all people living in Switzerland, as well as Swiss citizens living abroad. As of today, the Swiss e-ID community is likely to pursue[4] a stack with two credential formats: one that complies with the European ARF and is based on standard and well-known technologies (Scenario A in their proposal), and an alternative format which achieves higher levels of privacy and responds to the linkablity concerns of the EU's prorposal implementing anonymous credentials (Scenario B).

However, *revoking* anonymous credentials remains an open problem: current revocation methods do not ensure unlinkability, while privacy preserving solutions seem to lack the scalability required for national e-ID systems. This thesis aims to address the revocation problem in the Swiss e-ID system, by developing a *practical* and *unlinkable* revocation scheme.

We devised an accumulator-based revocation scheme that provides efficient updates, improving the state of the art performances. Considering 10M credentials and a 2% rate of yearly revocations, we reduced the overhead of updating witnesses after a *month* of inactivity (i.e., 16.5K revocations) to less than 0.4s on a laptop, compared to the 1.4s required by previous approaches. Furthermore, we proposed two methods for limiting the maximum number of updates that holders need to compute after long offline periods. The first method limits the cost of updating to $0.1s$, at the cost of downloading weekly updates of 48B. Under a different threat model, the second method removes the need of downloading periodic updates, while achieving similar update complexity.

The technical scalability of the identified accumulator-based revocation

[3]ec.europa.eu

[4]`https://github.com/e-id-admin/open-source-community/blob/main/tech-roadmap/tech-roadmap.md`

solutions is not limited to small or medium-sized nations but could also be extended to larger settings, such as those in the European Union. We hope that, by demonstrating the feasibility of privacy-preserving revocation methods, we can contribute in rendering anonymous credentials more appealing to stakeholders and decision-makers.

1.2 Contributions

In this thesis we primarily focus on studying accumulator-based revocation, assessing and improving its scalability potential. In our e-ID scenario, an accumulator represents a set of non-revoked credential holders. Each holder can show non-revocation of his credential by proving membership in the set of non-revoked credentials. These membership proofs are zero knowledge (ZK), and therefore unlinkable. However, when credentials are revoked the accumulator value must change, requiring every credential holder to update. This update process has been identified as the main scalability obstacle, as it demands computationally expensive operations from the holders.

In the following, we list the primary contributions of this thesis:

- In Section 4.2, we provide an open-source implementation of a popular pairing-based accumulator scheme [KB21], highlighting its shortcomings;

- In Section 4.3, we substitute a popular Σ-protocol typically used for proving membership in pairing-based accumulators with a more efficient protocol, that was recently proposed in the context of BBS disclosure proofs [TZ23]. In our implementation, we show that this translates into a $10\times$ improvement;

- In Section 4.4, we optimize the batch update technique introduced in [VB22], adopting Pippenger's approach for efficient polynomial evaluation. In our implementation, we registered between 3 and $4\times$ improvement compared to previous approaches.

  Furthermore, we propose a *novel* approach for efficiently aggregating multiple updates in a single batch. Consequently, we eliminate the Issuer's need of generating expensive update polynomials that required quadratic complexity in the update size;

- In Section 4.5, we propose a method for limiting the penalty imposed to offline users, using periodic update distribution. Our final accumulator construction was presented at the Cyber-Alp Retreat event organized by the Cyber Defence Campus. Furthermore, we discussed our approach with the Swiss e-ID team, that decided to include the efficient membership proofs in the specification draft for accumulator-based revocation[5];

- In Section 5.1, we integrate our accumulator-based revocation scheme to BBS signatures, proposing a Σ-protocol that binds accumulator's membership proofs to BBS presentations. The idea for the protocol was suggested by Jonas Niestroj from the e-ID team, while in this thesis we formalize it and proved security of the resulting Σ-protocol.

  Furthermore, we show that this binding inherently increases the security properties of our accumulator, without requiring the additional static signature adopted by previous approaches;

- In Section 6.3, we apply double-server Private Information Retrieval (PIR) techniques as an alternative update solution for our e-ID holders (Section 6.3). In a software implementation, we show that for 2^{23} credentials, this only requires ≈ 1 ms of user computation and less than 1 kilobyte of communication.

[5]`https://github.com/e-id-admin/open-source-community/tree/main/tech-roadmap/rfcs`

Chapter 2

Background

In this chapter, we give some notions on different revocation approaches. First we explain how revocation is typically enforced in today's Public Key Infrastructure (PKI) (Section 2.1). Then, we give some background on cryptographic accumulators (Section 2.2).

2.1 Certificate Revocation in the PKI

Before starting to evaluate alternative ways for revoking e-ID credentials, we illustrate how certificate revocation is enforced in today's internet.

Due to the massive amount of digital certificates existing in the PKI, revocation already have to scale to hundreds of millions of certificates. These numbers are (at least) an order of magnitude larger than what we expect for a medium-small country as Switzerland. However, as we discuss in Section 3.2, PKI methods lack some of the privacy guarantees required for our e-ID setting.

2.1.1 Pull-Based Approaches

In pull-based approaches, clients query revocation information *on-demand*, just before visiting a specific domain. Therefore, these methods typically require an extra round-trip for verifying certificate validity. The standard ways for checking revocation status of X.509 certificates are Certificate Revocation Lists (CRLs), and the Online Certificate Status Protocol (OCSP).

Certificate Revocation List

A CRL is a static file which includes the list of all digital certificates revoked by a specific Certificate Authority (CA). CAs regularly updates their CRL to include newly revoked credentials. Each list is signed by the respective CA and is downloadable from well-known distribution points, which are indicated in every certificate issued by that CA. When the client's browser encounters a new certificate, it downloads the CRL from the respective distribution point and checks that the certificate is not included in the list. Each CRL specifies a validity time and can be cached by the browser; in this way the user avoids re-downloading the CRL each time he encounters certificates issued by a given CA.

The main downside of CRLs are client's bandwidth and storage requirements, for instance in 2015 Apple's CRL had a size of over 76MB [Liu+15]. Obviously, CAs could split the revocation list into multiple CRLs to reduce sizes, however, when lists are too small, a client may need to download a new list every time he encounters a new certificate, even if he has already cached a valid CRL for that CA. Consequently, revocation lists for large CAs often weight several megabytes. At the time of writing, an Apple's CRL, and a DigiCert's CRL weight around 2MB [1] and 7MB [2] respectively.

Online Certificate Status Protocol

OCSP is a web service protocol that, with respet to CRLs, provides a more scalable way for checking non-revocation of single certificates. The CA includes in each issued X.509 certificate the URL of a designated OCSP server. To check for non-revocation of a certificate, clients can query the linked OCSP server, specifying the unique certificate ID. OCSP responses are signed by the CA and contain a binary answer. Responses also indicate a validity period; web browsers can cash them to avoid repeating the same queries for a certain period of time.

From the client side perspective, OCSP has two important drawbacks: (1) CAs can learn the client's browsing habits by looking at the domain name of the certificate for which non-revocation is being checked, and (2) the protocol is vulnerable to downgrading attacks in which an attacker suppress the OCSP response, the subsequent time-out results in "soft failure" (i.e., client accesses the website without checking for non-revocation). OCSP-stapling and the must-staple header was introduced to solve the last issue, however it requires to be actively adopted by every webserver.

2.1.2 Push-Based Approaches

Because of the severe limitations of pull-based approaches, web browser recently started exploring *push-based* alternatives. In push-based approaches, the web browser proactively downloads and caches revocation information that the client will use to check for non-revocation without online overhead. In the following we describe three important examples.

[1] `http://crl.apple.com/wwdrca.crl`

[2] `http://crl3.digicert.com/DigiCertGlobalG2TLSRSASHA2562020CA1-1.crl`

Browser Updates

This technique involves the browser vendor, which centrally gathers revocation information and distribute them by periodically pushing browser updates to the clients. Two instances of this method are OneCRL and CRLSets, which are provided by Firefox and Chrome respectively. As storing complete revocation information would require from hundreds of megabytes up to several gigabytes [Lar+17], the updates only include sets of *critical* revocations. For example, OneCRL updates only include intermediate CA revocation and a few manually selected others [3]. Despite being efficient, those methods only map a small percentage of the total number of revoked credentials. At the time of writing, both methods include only a few thousands revocations and the updated lists occupies roughly 20KB/600KB for CRLSets and OneCRL respectively.

CRLite

CRLite adopts probabilistic data structures (i.e., Bloom filters [Blo70]) to concisely store revocation information for the *entire* PKI credential space. A central entity (e.g., the browser vendor) obtains the revocation status for all non-expired certificates contained in the Certificate Transparency (CT), using public CRLs or OCSP queries. Then, he uses Bloom filters to represent the set of revoked certificates. As a single Bloom filter would produce some false positives due to collisions, a *cascade* of additional layers is attached until no more false positives are left (see [Lar+17] for details). Some version of Firefox already implement CRLite, furthermore Mozilla has announced that he plans to adopt CRLite as primary revocation mechanism in the future [4]. At the time of writing, Mozilla's full CRLite list occupies roughly 19.5MB, while update files (generated every 4 hours) are around 200KB [5]

2.2 Cryptographic Accumulators

A cryptographic accumulator is a data structure that offers a *concise* representation of a set of elements, and provides secure *membership testing.* The set of elements represented by an accumulator is called *accumulated set.* A *witness* is an additional piece of information that allows to prove (non-)membership of a specific element in the accumulated set. Some accumulators are ZK-friendly, meaning that they support proofs of (non-)membership that *hide* the specific element for which the proof was generated.

[3]`https://wiki.mozilla.org/CA/Revocation_Checking_in_Firefox#OneCRL`
[4]`https://wiki.mozilla.org/CA/Revocation_Checking_in_Firefox#CRLite`
[5]Measured with `https://github.com/mozilla/moz_crlite_query/tree/main`

2.2.1 Accumulator-Based revocation

Accumulators have been frequently adopted for handling revocation in anonymous credentials schemes (e.g., [CV02; CL02; CKS09; KC21]).

The general idea is the following: an accumulator maps the set of *non-revoked* credentials. When new credentials are issued, a unique credential identifier is added to the accumulator, and a *membership witness* is provided to the new credential holder. Similarly, when credentials need to be revoked, the associated element is removed from the accumulator. Only holders that are associated to currently accumulated credentials, can use their witnesses to produce a *valid* membership proof. Therefore, proving membership in the accumulator is *equivalent* to proving non-revocation.

A similar approach can be used if we let the accumulator map a set of revoked users. Credential holders would then receive non-membership witnesses, and proving non-revocation would be equivalent to showing non-membership in the accumulator.

Accumulators can be classified based on the type of proofs they support: *positive* accumulators only support membership proofs, *negative* accumulators only support non-membership proofs, while *universal* accumulators support both.

Some accumulators also supports efficient *modifications* to the accumulated set. Accumulators are called *additive* when they only support addition of new elements, *negative* when they only support removals, and *dynamic* when they support both additions and removals.

Depending on the trust settings, we can have *trapdoor-based* and *trapdoor-less* accumulators. Trapdoor-based accumulators are managed by a single entity, often called *revocation manager*, which controls the accumulator's trapdoor. Efficient additions/deletions of elements and witness issuance require knowledge of the accumulator's trapdoor, hence the accumulator manager is the only entity that can enforce them. On the contrary, trapdoor-less accumulators support public additions and deletions, and autonomous issuance of witnesses. However, operations are typically more expensive than with trapdoor-based accumulators.

In this thesis we focus on *positive dynamic* accumulators in the *trapdoor-based* setting. Our choice is due to the following reasons:

1. in a national e-ID, numerous users will presumably be added/revoked overtime. Consequently, we need the accumulator to be *dynamic*;

2. *membership* proofs/witnesses are generally much more efficient to compute/update then their *non-membership* counterparts. As revocation can be achieved in both ways, we prefer using *positive* accumulators;

3. in anonymous credential systems, a *single* entity (i.e., the issuer) is typically entitled of adding/revoking credentials. In the Swiss e-ID, the

only entity entitled of issuing e-ID credential is the Swiss Government (for details, see Section 3.1). Hence, trapdoor-less solutions do not fit our requirements;

In Section 3.2, we will analyize more in detail some specific accumulator instances.

2.2.2 Accumulator Definition

In this section, we give the definition of a positive, dynamic accumulator. Our definitions are mainly based on those provided in [KL24]. In addition, we label each function according to the entity executing it:

- the credential *issuer* is denoted as $\mathcal{I}$;
- a credential *holder* is denoted as $\mathcal{H}$;
- a credential *verifier* is denoted as $\mathcal{V}$.

Definition 2.2.1 (**Positive Dynamic Accumulator**). *A positive dynamic accumulator is a set of algorithms* ($\mathbf{Gen}_{\mathcal{I}}, \mathbf{Add}_{\mathcal{I}}, \mathbf{Del}_{\mathcal{I}}, \mathbf{WitUpAdd}_{\mathcal{H}}$, $\mathbf{WitUpDel}_{\mathcal{H}}, \mathbf{VerifyWit}_{\mathcal{V}}$) *defined as follows:*

- $\mathbf{Gen}_{\mathcal{I}}(1^\lambda) \rightarrow (acc_0, sk, pp, st_0)$*: this algorithm initializes the system. It takes as input the security parameter* λ*, and outputs an empty accumulator* acc_0*, the accumulator's secret key* sk*, the public parameters* pp *(e.g., describing the accumulator's domain), and the initial issuer's state* st_0*;*

- $\mathbf{Add}_{\mathcal{I}}(sk, acc_t, x, st_t) \rightarrow (V_{t+1}, w_{x,t+1}, upmsg_{t+1}, st_{t+1})$*: this algorithm adds a new element* x *to the accumulated set. It takes as input accumulator's secret key* sk*, and the current accumulator value* acc_t*. It returns the new accumulator value* acc_{t+1}*, a membership witness for the* $w_{x,t+1}$ *for the added element* x*, an update message* $upmsg_{t+1}$*, and the new issuer's state* st_{t+1}*.*

- $\mathbf{Del}_{\mathcal{I}}(sk, acc_t, x, st_t) \rightarrow (acc_{t+1}, upmsg_{t+1}, st_{t+1})$*: this algorithm removes an element from the accumulated set. It takes as input the secret key* sk*, the current accumulator value* acc_t*, an element* x *that was previously added to the accumulator, and the issuer's state* st_t*. The function returns the new accumulator value* acc_{t+1}*, an update message* $upmsg_{t+1}$*, and the new issuer's state* st_{t+1}*;*

- $\mathbf{WitUpAdd}_{\mathcal{H}}(acc_{t+1}, acc_t, x, w_{x,t}, upmsg_{t+1}) \rightarrow w_{x,t+1}$*: this algorithm updates the witness of an accumulated element after a single addition event. It takes as input the current and the old accumulator values* $acc_t + 1$ *and* acc_t*, the element* x *and its respective witness* $w_{x,t}$*, and the update message* $upmsg_{t+1}$*. It returns the updated witnesses* $w_{x,t+1}$*;*

- $\mathbf{WitUpDel}_{\mathcal{H}}(acc_{t+1}, acc_t, x, w_{x,t}, upmsg_{t+1}) \rightarrow w_{x,t+1}$: *this algorithm updates the witness of an accumulated element after a single deletion event. It takes as input the current and the old accumulator values* acc_{t+1} *and* acc_t*, the element* x *and its respective witness* $w_{x,t}$*, and the update message* $upmsg_{t+1}$*. It returns the updated witnesses* $w_{x,t+1}$*;*

- $\mathbf{VerifyWit}_{\mathcal{V}}(acc_t, x, w_{x,t}) \rightarrow 0/1$: *this algorithm tests membership of element* x *in the accumulator* acc_t*. It takes as input the current accumulator value* acc_t*, the element* x *and its respective witness* $w_{x,t}$*. It returns* 1 *if* $w_{x,t}$ *is a valid membership witness for* x *and* 0 *otherwise.*

2.2.3 Security Properties

The cryptographic accumulator defined in Definition 2.2.1 must satisfy *correctness* and *soundness* to be considered secure. Correctness implies that the honest holder of an up-to-date witness can *succesfully* proof membership of the respective element in the accumulator. Soundness implies that an adversary with oracle access to the accumulator's addition/deletion functions cannot produce a valid witness for a non-accumulated element. We consider two levels of soundness: *non-adaptive* soundness (Definition 2.2.2) requires the attacker to commit the list of elements that will be added to/removed from the accumulator in advance. *Adaptive* soundness (Definition 2.2.3), is a stronger notion where the adversary can adaptively pick elements to add/remove. As the formulation of correctness is straightforward, in the following we only write definitions for (non-)adaptive soundness, which are based on those provided in [KB21].

Definition 2.2.2 (Non-Adaptive Soundness). *For a security parameter* λ*, and all probabilistic polynomial time (PPT) adversaries* $\mathcal{A}$ *with black-box access to the addition/deletion oracles* $\mathcal{O}_{\mathbf{Add}}, \mathcal{O}_{\mathbf{Del}}$ *on an accumulator for a dynamic set* S *with changing value* V*, let* L_A, L_D *respectively be the list of additions and deletions that* $\mathcal{A}$ *submits ahead of time. There exists a negligible function* $negl(\cdot)$ *such that:*

$$\Pr\left[\begin{array}{r}\{(x_1,\ldots,x_{|A|}) \in L_A, (x_1,\ldots,x_{|D|}) \in L_D,\} \leftarrow \mathcal{A}(\lambda);\\ (acc_0, sk, pp, st_0) \leftarrow \mathbf{Gen}_{\mathcal{I}}(1^\lambda);\\ (x, w_{x,t}) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathbf{Add}},\mathcal{O}_{\mathbf{Del}}}(acc_0);\\ x \notin S' : \mathbf{VerifyWit}_{\mathcal{V}}(acc_t, x, w_{x,t}) = 1\end{array}\right] \leq negl(\lambda)$$

Where S' *denotes the updated accumulated set after addition/deletion queries. The addition oracle* $\mathcal{O}_{\mathbf{Add}}$ *returns nothing when asked to add an element* $x \in S'$ *or* $x \notin L_A$*, while the deletion oracle* $\mathcal{O}_{\mathbf{Del}}$ *returns nothing when asked to delete an element* $x \notin S'$ *or* $x \notin L_D$*.*

Definition 2.2.3 (**Adaptive Soundness**). *For a security parameter λ, and all PPT adversaries $\mathcal{A}$ with black-box access to the addition/deletion oracles $\mathcal{O}_{\mathbf{Add}}, \mathcal{O}_{\mathbf{Del}}$ on an accumulator for a dynamic set S with changing value V. There exists a negligible function $negl(\cdot)$ such that:*

$$\Pr\left[\begin{array}{r} (acc_0, sk, pp, st_0) \leftarrow \mathbf{Gen}_{\mathcal{I}}(1^\lambda); \\ (x, w_{x,t}) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathbf{Add}}, \mathcal{O}_{\mathbf{Del}}}(acc_0); \\ x \notin S' : \mathbf{VerifyWit}_{\mathcal{V}}(acc_t, x, w_{x,t}) = 1 \end{array}\right] \leq negl(\lambda)$$

Where S' denotes the updated accumulated set after addition/deletion queries. The oracles $\mathcal{O}_{\mathbf{Add}}, \mathcal{O}_{\mathbf{Del}}$ return nothing when asked to add an element $x \in S'$, or to delete an element $x \notin S'$.

Additionally, we report the definition of q-Strong Diffie-Hellman (q-SDH) assumption. The q-SDH assumption is relevant for us, as it is used to prove correctness of the pairing-based accumulators we consider in Section 3.2.

Definition 2.2.4 (**q-Strong Diffie Hellman Assumption**). *Let $\mathbb{G}_1, \mathbb{G}_2$ be two cyclic groups of prime order p, respectively generated by G_1, G_2, and $x \leftarrow\!\!\$\, \mathbb{Z}_p$ be a randomly chosen element. Any PPT adversary $\mathcal{A}$ on input $(G_1, xG_1, \ldots, x^q G_1)$ has negligible probability of computing a pair $\left(\frac{1}{(x+c)}G_1, c\right)$, for some freely chosen $c \in \mathbb{Z}_p$.*

Chapter 3

Revocation in the Swiss e-ID Setting

3.1 Setting

In this section, we first present the entities involved in the Swiss e-ID (Section 3.1.1). Then, we describe the two credential formats proposed by the e-ID team (Section 3.1.2), and discuss which one we consider in this thesis (Section 3.1.4). Finally, we list the requirements that a revocation scheme needs to satisfy to be adopted in the Swiss e-ID (Section 3.1.3).

3.1.1 Entities

The e-ID setting involves the participation of three distinct entities:

- **issuer**: the only entity entitled to issue e-ID credentials. In the context of e-ID, the issuer is identified with the Swiss government, and it is assumed to have high (e.g., enterprise-level) resources;
- **holder**: a Swiss citizen or any other subject entitled to possess a Swiss e-ID. The holder obtains his e-ID from the issuer and he is assumed to be resource-constrained (e.g., bound to a few computation seconds on a smartphone);
- **verifier**: a service provider enrolled in the e-ID ecosystem. The verifier decides whether to provide his services to a specific holder depending on whether the holder's credential satisfies his policy (e.g., holder is > 18 y.o.).

3.1.2 Swiss e-ID Proposals

The Swiss e-ID community[1] has selected two alternatives for e-ID credentials, their design is depicted in Scenario A (Figure 3.1) and Scenario B (Figure 3.2). Scenario A uses a subset of technologies included in the ARF [Eur23], and prioritize compatibility with the Electronic IDentification, Authentication and trust Services (eIDAS) implementation. On the other hand, Scenario B adopts a set of privacy-preserving technologies to address the linkability concerns that were pointed out for eIDAS 2.0 [Bau+24]. In the following, we briefly analyze both scenarios, focussing on the proposed revocation strategy(s).

Scenario A

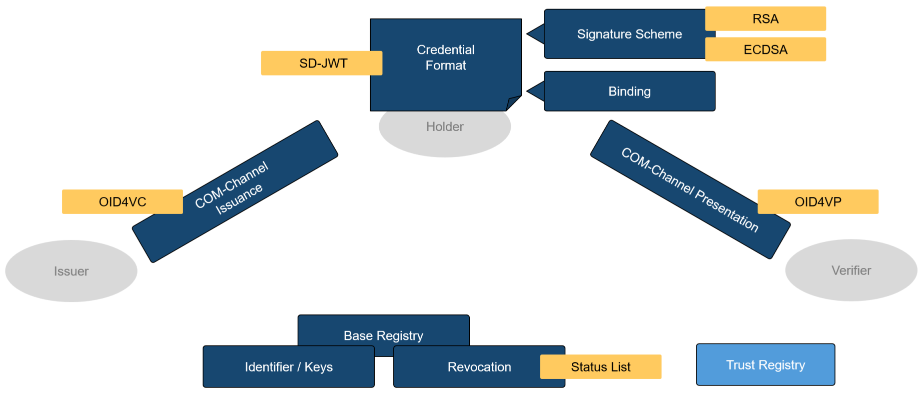

Figure 3.1: Scenario A

Scenario A (Figure 3.1) is based on classic RSA/ECDSA signatures, and adopts the SD-JWT credential format [FYC24]. In brief, SD-JWT provides selective disclosure by attaching to the credential the (salted) hash of all the holder's attributes/claims. On credential presentations, the holder only discloses with the verifier the openings for the attributes he intends to proof.

The proposed revocation method is a simple status list, in which each entry represents the revocation status of a given credential. On presentations, the holder discloses the position of his credential in the status list. Then, the verifier checks that the entry respective to the holder's credential is marked as non revoked.

Privacy considerations Both the presentation and revocation methods proposed in this scenario suffer from linkability issues:

[1] https://github.com/e-id-admin/open-source-community/blob/main/discussion-paper-tech-proposal/discussion-paper-tech-proposal.md

- on *credential presentation*, the holder discloses the signature and hashed digests of attributes contained in his credential. This data constitute a unique fingerprint of the holder's credential, and allow a verifier to *link* subsequent presentations from the same holder. Collusion between multiple verifiers could simplify de-anonymization, and enable tracking of holders online behaviors;

- on *credential revocation*, the entry of the credential in the status list changes from "not revoked" to "revoked". As the status list is public, each verifier can memorize the positions of credentials that were presented to him, and *track* any status change. Furthermore, the position of a credential in the status list constitutes an additional *unique identifier*.

Scenario B

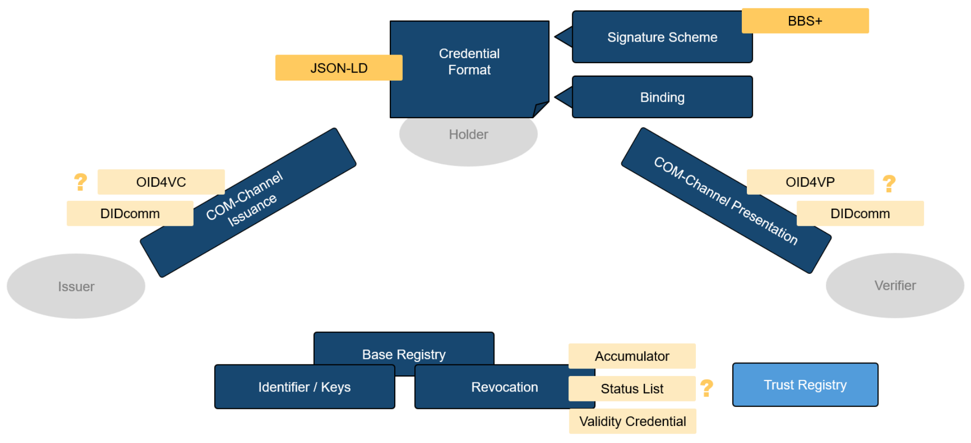

Figure 3.2: Scenario B

Scenario B (Figure 3.2) is based on BBS signatures [Loo+23], and adopts the JSON-LD format [Spo+20] for credential representation. Contrarily to the RSA/ECDSA signatures adopted by Scenario A, BBS signatures intrinsically provide selectively disclosure of the credential's attributes in a zero-knowledge way. Holder's proofs are randomized in such a way that even colluding verifiers cannot link them. The choice of the revocation method is still open, they propose three option: cryptographic accumulators, status lists, and validity credentials.

3.1.3 Requirements

In this section, we list the three main requirements of our e-ID scenario. A new revocation mechanism needs to preserve all these requirements to be considered adoptable:

R1. the issuer has *no knowledge* on if or how the verifiable credentials it issued are used;

R2. a verifier should not be able to *link* multiple presentations of the same credential, even by colluding with other verifiers;

R3. it must be possible to revoke a credential *within* 24h.

Requirement R1 preserves holder's unlinkability towards the *issuer*. Solutions that require the issuer's participation in the credential presentation process leak sensitive metadata and are discouraged by the Swiss e-ID draft Act [2].

Requirement R2 preserves holder's unlinkability towards the *verifier*. As Scenario B adopts BBS+ signatures and Zero Knowledge Proofs (ZKPs) to achieve unlinkable presentations, we do not consider revocation schemes that prove non-revocation by disclosing unique identifiers (e.g., classic CRLs). This requirement does not include unique identifiers that are possibly disclosed as part of the presentation (e.g., name and date of birth).

Regarding R3, both European e-ID regulations[3] and PKI baseline requirements[4] concur on requiring that revocations must be enforced within 24 hours upon receiving the revocation request. Given the absence of specific indications, it seems reasonable to extend the same requirement to the Swiss e-ID scenario.

3.1.4 Our Work

Since the linkability concerns in Scenario A are inherent to the credential format, we see little advantages in implementing a privacy-preserving revocation method. Therefore, the rest of this thesis will focus on identifying a scalable and privacy-preserving revocation method for credentials of the type described in Scenario B.

3.2 Related Work on Privacy-Preserving Revocation

In this section, we study the related works on unlinkable revocation, aiming to identify some methods that are applicable to the e-ID setting we have just introduced in Section 3.1.

[2]Article 7.8, **www.fedlex.admin.ch**
[3]Article 24, **eur-lex.europa.eu**
[4]Section 4.1.1, **cabforum.org**

3.2.1 Initial Remarks

For applicability in our e-ID scenario, a revocation method must preserve the unlinkability of BBS presentations. All the PKI methods discussed in Section 2.1 require the holder to disclose a *unique identifier* with the verifier. Non-revocation is verified either by checking the holder's identifier against a set of revoked IDs (CRLs, CRLSets, OneCRL, and CRLite), or by querying the issuer directly (OCSP). With both approaches, subsequent presentations of the same credential can be *linked* through the disclosed identifier. This renders the above methods incompatible with our privacy requirements (R2).

Some works have obtained unlinkable revocation with ephemeral credentials, or periodic update (e.g., [CKS10; CDH16]). Our requirements stipulate that revocations must be executed *within* 24 hours (R3). Since the latter methods do not support direct revocation, enforcing timely revocation would necessitate issuing short-lived credentials. Consequently, holders would need to frequently contact the issuer for updates or re-issuance, resulting in significant computational and communication overhead for the system. From a privacy perspective, this 'call-home' behavior leaks *timing metadata*, as holders update their credentials just before presenting them.

On the other hand, cryptographic accumulators provide *instant* revocation and allow valid holders to produce *unlinkable* proofs of non-revocation. Credential holders can *autonomously* update their witness,without further interaction with the issuer (i.e., avoiding to call-home). The key scalabilty factors in an accumulator-based revocation scheme are the communication and computational overhead that such updates impose on holders. In the following, we evaluate some practical implementations of the accumulators introduced in Section 2.2, specifically *Merkle hash-trees*, *pairings-based*, and *RSA-based* accumulators.

3.2.2 Notation

Henceforth, we let $|a|$ denote the number of elements *added* to the accumulator, $|d|$ the number of deleted elements, m the total number of modifications to the accumulator value, and N the number of non-revoked users in the system.

3.2.3 Merkle Hash-Trees

Recently, numerous credential schemes combining Merkle hash-trees with with Zero-Knowledge Succinct Non-Interactive Argument of Knowledge (zk-SNARK) protocols have been proposed (e.g., [BS23; Ros+23]). These schemes make use of general-purpose ZKPs to support arbitrary predicate proofs (e.g., proof geo-location [BS23]), at the expense of imposing additional computational burden on provers (i.e., credential holders).

When it comes to membership proofs, RSA and pairing-based accumulators provide efficient $O(1)$ proofs, while Merkle trees incur a cost of $\Omega(\log N)$. Furthermore, to anonymously update his witness, a holder would need to download the *entire* Merkle tree that contains the set of non-revoked credentials. As a practical reference, in [BS23] holders download several MB per *single* update. As noted by the authors, holders can limit the update size by selectively retrieving one of the sub-trees containing their credential. however this proportionally reduces the anonimity set of the update. For instance, fetching just the left (or right) sub-tree *halves* the anonimity set, while fetching a specific roof-to-leaf path reduces the anonimity set to the number of credentials contained in the leaf (e.g., in [BS23] each leaf stores 253 credentials).

3.2.4 RSA accumulators

The earliest accumulator construction introduced by Benaloh and de Mare [BD93] was based on RSA accumulators. The authors picked an RSA modulus n, and a random accumulator $acc_\emptyset \leftarrow\!\!\$\, \mathbb{Z}_n^*$. To accumulate a set of elements $S = \{e_1, \ldots, e_m\}$, one would compute $acc_S \leftarrow acc_\emptyset{}^{\prod_{e_i \in S} e_i}$, while the membership witness for an element $e \in S$ was defined as $wit_e \leftarrow acc^{\prod_{e_i \in S \setminus \{e\}} e_i}$, such that $(wit_e)^e = acc_S$. The initial construction supported efficient additions only, and soundness was guaranteed only when accumulating prime elements. Subsequently, Camenisch and Lysyanskaya [CL02] improved the initial construction, adding support for efficient deletions and unlinkable membership proofs. Baldimtsi et al. [Bal+17], eliminated the need of updating on additions of new elements, proposing Braavos, an RSA accumulator with *communication-optimal* $O(|d|)$ updates. Among their contributions, the authors presented a modular construction for building dynamic accumulators with strong security, integrating a static accumulators with strong security, and a dynamic accumulator with weaker security. All these works were limited in accumulating prime elements only. Consequently, adding a non-prime element to the accumulator involved expensive hash-to-prime operations. In a recent work, Kemmoe and Lysyanskaya [KL24] developed an RSA-based accumulator that does not require accumulated elements to be represented as prime numbers. According to their benchmarks, this translates in around 2× performance improvements when computing modular exponentiations.

3.2.5 Paring-Based accumulators

In the pairing setting, the most popular accumulator is due to L. Nguyen [Ngu05]. Nguyen's accumulator is positive, dynamic, and adaptively sound under the q-SDH assumption (Definition 2.2.4). Unlike RSA accumulators, Nguyen's accumulator is defined on a group $\mathbb{G}$ of *well-known* prime order p, with a symmetric bilinear pairing map $\tilde{e} : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_m$. Given an accumulator $acc_S \in \mathbb{G}$, additions of an element e requires knowledge of the accumulator's trapdoor $sk \leftarrow_{\$} \mathbb{Z}_p$, which is summed to the new element and used to produce a new accumulator $acc_{S \cup \{e\}} \leftarrow (sk + e) \cdot acc_S$. Similarly, deletion of an element e requires dividing the accumulator by $(sk + e)$, i.e., $acc_{S \setminus \{e\}} \leftarrow (sk + e)^{-1} \cdot acc_S$. The membership witness for an accumulated element $e \in S$ is defined as $wit_e \leftarrow (sk + e)^{-1} acc_S$. Direct membership verification is possible by checking $(sk+e) \cdot wit_e = acc_S$, however it requires knowledge of the *accumulator's trapdoor* sk. Instead, thanks to the bilinear properties of the pairing, membership in the accumulator can be verified by checking $\tilde{e}(wit_e, e \cdot g_1 + pk) = \tilde{e}(acc_S, g_2)$, where $pk \leftarrow sk \cdot g_1$, and $g_1, g_2 \in \mathbb{G}$ are fixed generators. The author also proposes a Σ-protocol for verification of the latter pairing equation[5], hiding the value of the holder's element e and associated witness wit_e.

Karantaidou et al. follow the work in [Bal+17] to propose a communication-efficient construction of [Ngu05], with $O(|d|)$ update complexity [KB21]. On trapdoor-based setting, [KB21] enables more efficient operations than RSA accumulators, as it adopts groups of known order and has smaller parameters size. Furthermore, the soundness of [KB21] is based on the same security assumption of BBS signatures (i.e., the q-SDH assumption), and can be implemented on the same curves.

Vitto and Biryukov, propose a universal Nguyen-based accumulator [VB22]. The accumulator has communication complexity $O(|a|+|d|)$, as it is updated both on additions and deletions. However, they provide a method for batching multiple updates with two polynomials, called *update polynomials*. The *evaluation* of these polynomials on input an accumulated element yields some coefficient that can be used to *update* the associated witness. However, the adaptive-soundness of their universal accumulator has been broken. One of the two polynomials used to batch additions can be efficiently factored (as it is defined over a prime-field) yielding the list of added elements. By knowing the list of additions, an attacker who follows the strategy presented in [JLM22, Section 3.1] can produce a valid membership witness for a non-accumulated element.

In [JLM22], Jacques et al. instantiate the communication efficient construction of Nguyen's accumulator over type-3 pairings. To update witnesses, the authors outsource the evaluation of the update polynomial technique introduced in [VB22] using Multi-Party Computation (MPC) techniques, achieving $O(\sqrt{m})$ holder computation. However, for updates to be anonymous, they assume that a fraction of the involved parties do not collude, which requires a modification in our threat model. Furthermore, the proposed MPC protocol would introduce considerable complexity in the system. First of all, each holder would need to wait for the complete message

[5]To ensure adaptive soundness, the prover needs to also demonstrate possession of a static BB signature on the element e.

exchange with the MPC servers, paying for all the additional communication time. Furthermore, to satisfy holders queries, trusted parities would need to create expensive update polynomials and evaluate them *on-the-fly*. If additional countermeasures are not applied, this could facilitate Denial of Service (DoS) attacks with large amplification factors. For instance, an attacker can flood some of the MPC parties with update queries. Note that mitigating these kind of attacks is non-trivial: if each update query is anonymous, it is hard to identify the source of the attack. For all these reasons, we believe that such an MPC solution is unlikely to scale up to millions of users.

Our work In the following chapter, we build an accumulator starting from the non adaptively sound communication-optimal construction of [KB21]. Then, we apply the Σ-protcols for BBS disclosure that were recently introduced by Tessaro and Zhu [TZ23], to provide faster ZK membership proofs. Using the update polynomials introduced in [VB22] to batch *deletions*, together with efficient polynomial evaluation techniques, we reduce the computational cost for holder update by a $\log m$ factor, while proving that the new accumulator construction preserves security. Then, we propose an efficient method for evaluating multiple batch/individual updates as a single batch. Such method eliminates the need for computing batch polynomials and, to the best of our knowledge, represents a novel contribution of this thesis. Finally, we limit the maximum number of holder updates introducing the concept of revocation epochs.

Note that in [KB21], Baldimtsi et al. propose an accumulator with stronger security, however their construction requires binding accumulator proofs to a long-term BB signature. For this reason we chose to adopt the version with non-adaptive security. In Chapter 5, we explain how to achieve adaptive soundness using the BBS signatures that are already provided to *every* e-ID holder (i.e., without introducing any additional signature).

Chapter 4

Building a Dynamic Accumulator with Efficient Update

In this chapter, we present a positive dynamic non-adaptively sound accumulator scheme that supports *efficient updates*. We start by introducing the notation (Section 4.1), and giving a preliminary construction of the scheme (Section 4.2). Then, we proceed addressing privacy (Section 4.3) issues of the initial construction, adding support for ZK membership proofs. In the final part of the chapter, we focus on improving the scheme's scalability by tackling the update problem (Sections 4.4 and 4.5).

In the following Chapter 5, we will explain how this scheme can be used for revoking BBS credentials.

4.1 Notation

In Table 4.1 we describe the symbols that we will use throughout this thesis.

We will assume to have fixed a prime field $\mathbb{F}_q$, and elliptic curves[1] E, E' with a non-degenerate bilinear pairing map:

$$\tilde{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_t,$$

such that $\mathbb{G}_1 \subset E(\mathbb{F}_q)$, and $\mathbb{G}_2 \subset E'(\mathbb{F}_{q^k})$ for some integer k. The three subgroups have same prime order p (i.e., $p = |\mathbb{G}_1| = |\mathbb{G}_2| = |\mathbb{G}_t|$). A specific pairing instance will be indicated by a tuple $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, \tilde{e}, G_1, G_2)$, where G_1, G_2, are generators of $\mathbb{G}_1$, and $\mathbb{G}_2$ respectively.

Note that each element of $\mathbb{G}_1$ can be expressed in $\log q$ bits, each element of $\mathbb{G}_2$ in $k \log q$ bits, and each scalar in $\log p$ bits. Scalars and group elements are respectively indicated by lower-case and upper-case letters.

[1]In our software implementation, we use the pairing-friendly BLS12-381 curve.

Symbol	Description
$\leftarrow$	Assignment
$\rightarrow$	Returned values
$\leftarrow_{\$}$	Chosen uniformly at random
$=$	Equality
$\perp$	Failure
$\tilde{e}$	Bilinear pairing map
q	Base field modulus
p	Order of the inner subgroups
$0_{\mathbb{G}}$	Identity element of group $\mathbb{G}$
log	$\lceil \log_2 \rceil$ (i.e., number of bits)

Table 4.1: Symbols table

To simplify notation, we denote the witness associated with an element e at time t as A_t instead of $wit_{e,t}$ (as was done in our accumulator definition).

Finally, we use *additive notation* to express operations between group elements.

4.2 Preliminary Construction of our Accumulator Scheme

In this section, we first give a preliminary construction of our accumulator scheme for revoking BBS signatures (Section 4.2.1). Then, we analyze the performances of the accumulator through a theoretical analysis (Section 4.2.2), and a software implementation (Section 4.2.3). Finally, we outline the shortcomings of the scheme (Section 4.2.4), that motivate the work of the following sections.

4.2.1 Our Preliminary Accumulator Scheme

In Figure 4.1, we define our preliminary accumulator scheme, which serves as the foundation for our final design. This accumulator is borrowed from the communication-optimal, positive dynamic non-adaptively sound construction presented in [KB21, Section IV], and implemented over type-3 pairings. Soundness is proven under the q-SDH assumption.

The advantage of [KB21] construction with respect to the original Nguyen's accumulator [Ngu05], is that the accumulator value remains constant when new elements are *added*. When considering our e-ID setting, this translates into requiring no holder update on *issuance* of new credentials.

Note that, since we only update on deletions, in Figure 4.1 we have slightly modified our accumulator definition to include a *single* update function $\mathbf{WitUp}_{\mathcal{H}}$ instead of the pair $\mathbf{WitUpAdd}_{\mathcal{H}}, \mathbf{WitUpDel}_{\mathcal{H}}$. Further-

```
$\underline{\mathbf{Gen}_{\mathcal{I}}(1^{\lambda})}$
1. choose $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, \tilde{e}, G_1, G_2)$ with security $\lambda$;
2. $x \leftarrow\$ \mathbb{F}_p$;
3. $X \leftarrow xG_2$;
4. $V_0 \leftarrow\$ \mathbb{G}_1$;
5. $S_0 \leftarrow \emptyset$;
6. $pp \leftarrow ((p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, \tilde{e}, G_1, G_2), X)$;
7. return $(V_0, x, pp, S_0)$

$\underline{\mathbf{Add}_{\mathcal{I}}(x, V_t, e, S_k)}$
1. if $e \in S_k$ : return $\perp$;
2. $A_t \leftarrow \left(\frac{1}{x+e}\right) V_t$;
3. $S_{k+1} \leftarrow S_k \cup \{e\}$;
4. return $(A_t, S_{k+1})$

$\underline{\mathbf{Del}_{\mathcal{I}}(x, V_t, e, S_k)}$
1. if $e \notin S_k$ : return $\perp$
2. $V_{t+1} \leftarrow \left(\frac{1}{x+e}\right) V_t$;
3. $S_{k+1} \leftarrow S_k \setminus \{e\}$;
4. $upmsg_{t+1} \leftarrow (V_{t+1}, e)$;
5. return $(V_{t+1}, upmsg_{t+1}, S_{k+1})$

$\underline{\mathbf{WitUp}_{\mathcal{H}}(V_t, e, A_t, upmsg_{t+1})}$
1. $(V_{t+1}, e_t) \leftarrow upmsg_{t+1}$;
2. if $e = e_t$ : return $\perp$
3. $A_{t+1} \leftarrow \left(\frac{1}{e_t - e}\right) (A_t - V_{t+1})$
4. return $A_{t+1}$

$\underline{\mathbf{VerifyWit}_{\mathcal{V}}(V_t, e, A_t)}$
1. return $\tilde{e}(A_t, eG_2 + X) = \tilde{e}(V_t, G_2)$
```

Figure 4.1: An Initial Accumulator Construction

more, to simplify notation, we denote the witness associated to an element e at time t as A_t instead of $A_{e,t}$ (as done in Definition 2.2.1).

Initialization the issuer runs $\mathbf{Gen}_{\mathcal{I}}$ to initialize the system. He chooses the underlying type-3 bilinear pairing instance $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_t, \tilde{e}, G_1, G_2)$ using the security parameter λ. Then, he randomly picks the initial accumulator value V_0, the accumulator's *secret key* x, and derives the public key $X = xG_2$. Finally, the algorithm initializes the issuer's state with the (initially) empty accumulated set S_0.

The public parameters pp are set as first *implicit* parameter in each of the following algorithms.

Additions the issuer runs $\mathbf{Add}_{\mathcal{I}}$ to add an element e in the accumulated set. Initially, the issuer ensures that e is not accumulated already. Then, given the current accumulator V_t, he computes the witness A_t for the new element e. Finally, he updates the state by including e in the accumulated set. Note that, as the accumulator value is *not* modified, no update message needs to be returned.

Deletions the issuer runs $\mathbf{Del}_{\mathcal{I}}$ to delete an element e from the accumulator. Initially, the issuer ensures that e is currently in the accumulated set. Then, he updates the accumulator value to incorporate the deletion of e. Finally, he updates the issuer's state by removing e from the accumulated set, and creates the update $upmsg_t$ from the new accumulator value V_{t+1}, and the deleted element e_t.

Update the holder runs $\mathbf{WitUp}_{\mathcal{H}}$ to update his witness after a revocation event. He uses the new accumulator value V_{t+1}, and the deleted element e contained in message $upmsg_t$ to compute an individual witness update.

Verification any party can run $\mathbf{VerifyWit}_{\mathcal{V}}$ to check whether the A_t is a valid witness for the element e with respect to the input accumulator V_t. Verification of the pairing equation on input the issuer's public key X implies that the witness is correctly formed.

4.2.2 Theoretical Analysis

In this section, we analyze the performances of the preliminary construction defined in Figure 4.1. In Table 4.2 we measure the computational cost in terms of group operations, while in Table 4.3 we count the number of exchanged bits. The costs are relative to the entity executing the specified primitive, and are always referred to *single* executions.

We let A, M, I, P respectively denote *additions*, *multiplications*, *inversions*, and *pairing* operations.

$\mathbf{Gen}_{\mathcal{I}}$	$\mathbf{Add}_{\mathcal{I}}/\mathbf{Del}_{\mathcal{I}}$		$\mathbf{WitUp}_{\mathcal{H}}$		$\mathbf{VerifyWit}_{\mathcal{V}}$	
$\mathbb{G}_2$	$\mathbb{G}_1$	$\mathbb{F}_p$	$\mathbb{G}_1$	$\mathbb{F}_p$	$\mathbb{G}_1$	$\mathbb{F}_p$
$1M$	$1M$	$1A + 1I$	$1A + 1M$	$1A + 1I$	$2P + 1M + 1A$	$1M + 1A$

Table 4.2: Required group operations for computing the primitives in Figure 4.1

$\mathbf{Gen}_{\mathcal{I}}$	$\mathbf{Add}_{\mathcal{I}}/\mathbf{Del}_{\mathcal{I}}$	$\mathbf{WitUp}_{\mathcal{H}}$	$\mathbf{VerifyWit}_{\mathcal{V}}$
$2\log q + 2k \log q$	$\log q + \log p$	$\log q + \log p$	$\log q + \log p$

Table 4.3: Communication cost (in bits) of the primitives in Figure 4.1

Initialization Line 3 of $\mathbf{Gen}_{\mathcal{I}}$ requires one multiplication in $\mathbb{G}_2$. At the end, the issuer needs to publish the new accumulator V_0, which has size $\log q$, and the public parameters, comprising of 2 generators $(G_1, G_2) \in \mathbb{G}_1 \times \mathbb{G}_2$ of size $\log q + k \log q$, and the public key $X \in \mathbb{G}_2$ of size $k \log q$. Hence, the total communication cost is of $2 \log q + 2k \log q$.

Additions/Deletions Lines 2 of both $\mathbf{Add}_{\mathcal{I}}$, and $\mathbf{Del}_{\mathcal{I}}$ require $1M$ in $\mathbb{G}_1$, and $1A + 1I$ operations in $\mathbb{F}_p$. The communication cost for both primitives consist in sending a pair $(A_t, e) \in \mathbb{G}_1 \times \mathbb{F}_p$, which has size $(\log q + \log p)$ bits.

Witness Update Line 3 of $\mathbf{WitUp}_{\mathcal{H}}$ requires $1A{+}1M$ in $\mathbb{G}_1$, and $1A{+}1I$ operations in $\mathbb{F}_p$. The communication cost consist in downloading the update *upmsg*, which has size $(\log q + \log p)$. We highlight that, updating after m revocations requires m sequential applications of the $\mathbf{WitUp}_{\mathcal{H}}$ algorithm. Hence, the total update cost would be of $m(1A + 1M)$ operations in $\mathbb{G}_1$, $m(1A + 1I)$ operations in $\mathbb{F}_p$, and $m(\log q + \log p)$ downloaded bits.

Verification Line 1 of $\mathbf{VerifyWit}_{\mathcal{V}}$ requires $2P$, and $1M + 1A$ in $\mathbb{G}_1$. Downloading the user's witness/element pair costs $(\log p + \log q)$ bits.

Note that by rewriting the equation in Line 1 as $\tilde{e}(A_t, eG_2 + X) - \tilde{e}(V_t, G_2) = 0_{\mathbb{G}_t}$, we can use efficient techniques for multi-pairings evaluation (e.g., see [Sco19]), which require *less* than two individual pairing operations.

4.2.3 Software Implementation

We developed an open-source Rust implementation[2] of our accumulator scheme. In this section, we only report the performance relative to our preliminary construction. All the experiments are executed as a single thread on a 13-inch 2020 MacBook Pro, equipped with an Apple M1 chip, and 16 GB of RAM. The results are generated after 30 independent runs, and error bars are 95% confidence intervals.

Our accumulator adopts the pairing-friendly BLS12-381[3] elliptic curve implementation contained in the `blsful` crate[4]. The BLS12-381 curve been selected by the BBS standard [5], and provides around 128 bits of security. The field modulus q is of size $\log q = 381$ bits, while the subgroups of the pairing instance have prime order p of size $\log p = 255$ bits. We also have $\mathbb{G}_1 \subset E(\mathbb{F}_q)$, and $\mathbb{G}_2 \subset E'(\mathbb{F}_{q^2})$.

$\mathbf{Gen}_{\mathcal{I}}$	$\mathbf{Add}_{\mathcal{I}}$	$\mathbf{Del}_{\mathcal{I}}$	$\mathbf{WitUp}_{\mathcal{H}}$	$\mathbf{VerifyWit}_{\mathcal{V}}$
224.2 μs 288B	75.9 μs 80B	75.9 μs 80B	76.4 μs 80B	863.12 μs 80B

Table 4.4: Average computational costs, and communication costs for the primitives in Figure 4.1

Issuer Operations We observe that $\mathbf{Add}_{\mathcal{I}}$, and $\mathbf{Del}_{\mathcal{I}}$ have the same run-time, this is expected as they have the same complexity in terms of group operations. The run-time of $\mathbf{Gen}_{\mathcal{I}}$ is higher, as multiplications in $\mathbb{G}_2$ are almost $3\times$ more expensive than in $\mathbb{G}_1$ in our library.

[2]eid-revocation-rs

[3]draft-irtf-cfrg-pairing-friendly-curves-11

[4]blsful

[5]draft-irtf-cfrg-bbs-signatures

Verification As expected, $\mathbf{VerifyWit}_{\mathcal{V}}$ is the most expensive function since it involves computing pairing operations.

Single Witness Update The computational cost of $\mathbf{WitUp}_{\mathcal{H}}$ is dominated by a single multiplication in $\mathbb{G}_1$, which makes the runtime comparable with those of $\mathbf{Add}_{\mathcal{I}}$, and $\mathbf{Del}_{\mathcal{I}}$.

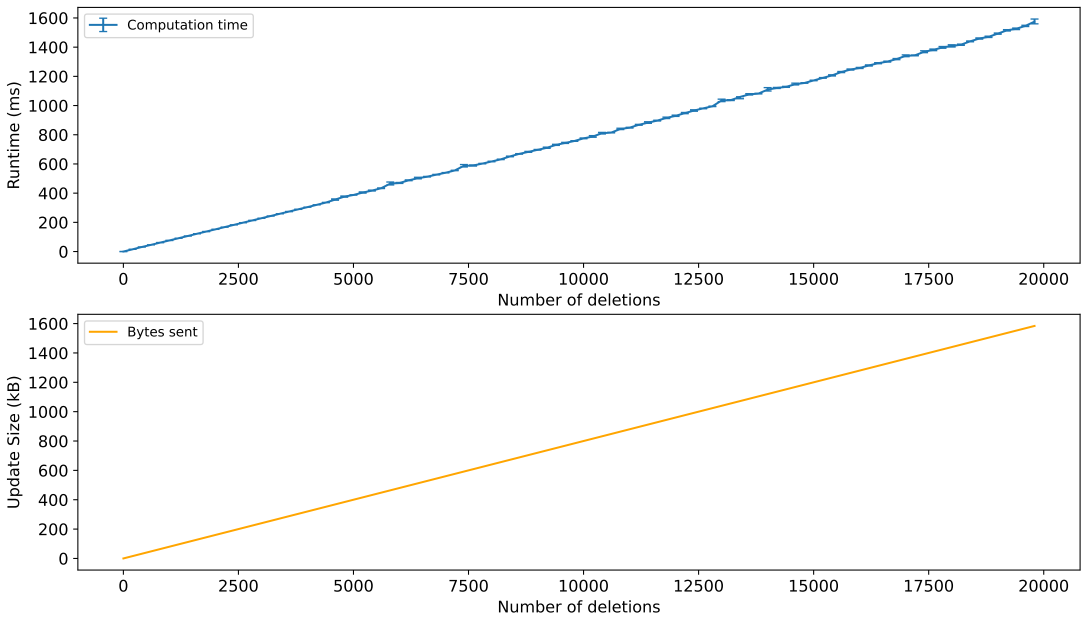

Figure 4.2: $\mathbf{WitUp}_{\mathcal{H}}$ on an increasing number of deletions

Sequential Witness Update In Figure 4.2, we evaluate the performances of updating a witness after an increasing number of deletion, using a sequential application of $\mathbf{WitUp}_{\mathcal{H}}$. Consistently with our theoretical analysis, we notice that both computational and communication complexity grow *linearly* in the number of deletions.

4.2.4 Limitations of the Scheme

In this section, we highlight three significant limitations of applying our initial accumulator scheme to the e-ID-setting described in Section 3.1:

L1. **Proofs are linkable**: in the current construction, the only way a verifier can check for inclusion of an element in the accumulator is by executing the $\textbf{VerifyWit}_{\mathcal{V}}$ function, on input the prover's element e, and witness A_t.

In our e-ID setting, each holder will be associated with a *static* and *unique* element e. To show non-revocation, the holder needs to prove that e is accumulated in V_t. Using $\textbf{VerifyWit}_{\mathcal{V}}$ is clearly not privacy-preserving: a verifier can *link* subsequent presentations of the same credential by simply comparing inputs to the verification function;

L2. **Update Scalabity**: before performing a membership proof, the prover needs to ensure that his witness is up-to-date with the current accumulator value (otherwise verification fails). As we noted in Section 4.2.2, updating a witness requires an execution of $\textbf{WitUp}_{\mathcal{H}}$ for each revocation that happened from the last update. This both implies computing a linear number of multiplications in $\mathbb{G}_1$, and downloading a linear number of group elements in the number of revocations.

In Section 4.2.3, we observe that for $m = 10\,000$ the update already requires 800KB of communication, and around $1s$ of computation on a high-end laptop. Assuming a 2% revocation rate and 10M credentials, we would get this number of revocations in around 2 weeks of inactivity. Considering that our holders devices (i.e., smartphones) have limited resources, updating would likely be $\approx 4-5\times$ slower than on my laptop. Hence, our e-ID holders would be highly penalized for being offline (i.e., missing updates). Overall, the overhead of updating witnesses poses a sever limitation on the scalability of our scheme, and may prevent the adoption in a real-world e-ID setting.

L3. **Witnesses not bound to BBS signatures**: there is no *binding* between the unique element e, and the BBS signature associated to a specific holder. Anyone knowing a valid (A_t, e) pair for the current accumulator V_t can proof non-revocation of his BBS signature. A non-revoked holder could even sell his (A_t, e) pair to anyone who wants to prove non-revocation. Note that the verification function $\textbf{VerifyWit}_{\mathcal{V}}$ already gives the verifier access to many valid (A_t, e) pairs.

In the following sections, we expand the scheme to address all the above limitations. We start by introducing support for ZK membership proofs (Section 4.3). Then, and we improve the scalability of the scheme by adding support for batch operations (Section 4.4), and limiting the maximum number of required holder updates (Section 4.5). In the next chapter, we design a Σ-protocol to ensure that the element e used in the membership proof is also included in the BBS signature (Section 5.1).

4.3 Adding Unlikability via ZK Proofs of Membership

In this section, we extend the preliminary accumulator construction introduced in Section 4.2, adding support for ZK proofs of membership (Section 4.3.1). Then, we analyze the performances of the new primitives (Section 4.3.3, and Section 4.3.4).

4.3.1 Unlikable Proofs of Memebrship

In Section 4.2.4, we highlighted the privacy implications of directly applying $\mathbf{VerifyWit}_{\mathcal{V}}$ for membership verification. In $\mathbf{VerifyWit}_{\mathcal{V}}$, the verifier checks membership by directly verifying the following pairing equation:

$$\tilde{e}(A_t, eG_2 + X) = \tilde{e}(V_t, G_2), \tag{4.1}$$

which requires knowledge of the holder's *unique* element e. Our goal, is to define a Σ-protocol between holder and verifier that enforces validity of eq. (4.1), without leaking *any* information about the holder.

To this end, we start by considering the Σ-protocol designed for BBS verification that was recently proposed in [TZ23, Section 5.2]. In the special case where the signed message is empty (i.e., $\mathbf{m} = \emptyset$), the verification equation on the BBS signature $\hat{\sigma} = (\hat{A}, \hat{e})$ looks as follows:

$$\tilde{e}(\hat{A}, \hat{e}G_2 + \hat{X}) = \tilde{e}(C, G_2), \tag{4.2}$$

where $C \in \mathbb{G}_1$, $\hat{x} \in \mathbb{Z}_p$, and $\hat{X} \leftarrow \hat{x}G_2$ are the issuer's secret and public key respectively, $\hat{e} \in \mathbb{Z}_p$ is a random element, and $\hat{A} \leftarrow \left(\frac{1}{\hat{x}+\hat{e}}\right) C$. After performing the following assignments:

$$C \leftarrow V_t, \quad \hat{A} \leftarrow A_t, \quad \hat{e} \leftarrow e, \quad \hat{X} \leftarrow X,$$

we notice that eq. (4.2) is equivalent to eq. (4.1). Hence, we adopt the Σ-protocol designed for BBS signatures to proof membership in our accumulator scheme.

4.3.2 Adding ZK Proofs of Membership

In Figure 4.3, we introduce new primitives defining a Σ protocol for membership proofs. Both primitives are borrowed from the full-disclosure protocol in [TZ23], and assume that the public accumulator V is given by the issuer. The security of the Σ-protocol is proved in the original paper ([TZ23, Section 5.2]).

```
MemProof_H(V_t, e, A_t)
1. MemProof.Comm_H(V_t, e, A_t)
     → (Ā, B̄, U, α, β, r);
2. send (Ā, B̄, U) to verifier;
3. receive c from verifier;
4. MemProof.Resp_H(e, α, β, r, c)
     → (s, t);
5. send (s, t) to verifier.

MemVer_V(V_t, X)
1. receive (Ā, B̄, U) from prover;
2. c ←$ Z_p;
3. send c to prover;
4. receive (s, t) from prover;
5. b_1 ← ẽ(Ā, X) = ẽ(B̄, G_2);
6. b_2 ← U + cB̄ = sV_t + tĀ;
7. return b_1 ∧ b_2.

MemProof.Comm_H(V_t, e, A_t)
1. r ←$ Z_p^*
2. α, β ←$ Z_p;
3. Ā ← rA_t;
4. B̄ ← r(V_t − eĀ);
5. U ← αV_t + βĀ;
6. return (Ā, B̄, U, α, β, r).

MemProof.Resp_H(e, α, β, r, c)
1. s ← α + r · c;
2. t ← β − e · c;
3. return (s, t).
```

Figure 4.3: Primitives adding support for ZKPs of membership

MemProof$_{\mathcal{H}}$		**MemVer**$_{\mathcal{V}}$	
$\mathbb{G}_1$	$\mathbb{F}_p$	$\mathbb{G}_1 \times \mathbb{G}_2$	$\mathbb{G}_1$
$5M + 2A$	$2M + 2A$	$2P$	$3M + 2A$

Table 4.5: Required operations for computing primitives in Figure 4.3

4.3.3 Theoretical Analysis

Prover Computing Line 3 of **MemProof**.**Comm**$_{\mathcal{H}}$ requires $1M$ in $\mathbb{G}_1$, while computing Lines $4, 5$ requires $2M+1A$ each. In total we have $5M+2A$ operation in $\mathbb{G}_1$. Computing Lines $1, 2$ of **MemProof**.**Resp**$_{\mathcal{H}}$, requires $2M + 2A$ operations in $\mathbb{F}_p$.

The prover sends 3 commitments in $\mathbb{G}_1$ (Line 2 of **MemProof**$_{\mathcal{H}}$), 2 responses in $\mathbb{F}_p$ (Line 5), and downloads one challenge in $\mathbb{F}_p$ (Line 3). In total, the exchanged communication is of $(3 \log q + 3 \log p)$ bits.

Verifier Verifying the first equation (Line 5 of **MemVer**$_{\mathcal{V}}$) requires $2P$, while verifying the second equation requires $3M + 2A$ in $\mathbb{G}_1$ (Line 6). We have a total of $2P$, and $3M + 2A$ in $\mathbb{G}_1$.

The verifier sends 1 challenge in $\mathbb{G}_1$ (Line 3), and downloads three commitments in $\mathbb{G}_1$ (Line 1), and two responses in $\mathbb{F}_p$ (Line 4). In total, the communication cost is of $(3 \log q + 3 \log p)$ bits (as expected since the communication is symmetric).

MemProof$_{\mathcal{H}}$	**MemVer**$_{\mathcal{V}}$
$3\log q + 3\log p$	$3\log q + 3\log p$

Table 4.6: Communication cost (bits) of primitives in Figure 4.3

4.3.4 Software Implementation

In Table 4.7, we can compare the average run-times, and communication costs of the new primitives for membership verification, with respect to linkable witness verification introduced in the previous section.

MemProof$_{\mathcal{H}}$	**MemVer**$_{\mathcal{V}}$	**VerifyWit**$_{\mathcal{V}}$
406.14 μs 240B	971.50 μs 240B	863.12 μs 80B

Table 4.7: Average computational costs, and communication costs of the primitives for ZK verification defined in Figure 4.3, compared to their linkable counterpart defined in Figure 4.1

The run-times of **MemVer**$_{\mathcal{V}}$ and **VerifyWit**$_{\mathcal{V}}$ are comparable, as both functions require the same amount of pairing operations. Furthermore, even though **MemVer**$_{\mathcal{V}}$ requires 3 more multiplication in $\mathbb{G}_1$, the single multiplication in $\mathbb{G}_2$ required by **VerifyWit**$_{\mathcal{V}}$ has higher individual cost.

The introduced overhead is mostly on the prover, who needs to run the **MemProof**$_{\mathcal{H}}$ algorithm instead of just presenting his witness-element pair. However, **MemProof**$_{\mathcal{H}}$ is the function with smallest run-time, as it does not involve any pairing operation.

Comparison with Original Membership Proof In Table 4.8, we benchmarked the performances of the original Σ-protocol for unlinkable proof verification introduced in [Ngu05].

Ngu.Proof	**Ngu.Ver**
5.8152 ms 720 B	3.503 ms 720 B

Table 4.8: Average computational costs, and communication costs of the original membership proof

The protocol was slightly improved in [JLM22], but still requires 4 pairing operations and 2 additions in $\mathbb{G}_t$ on the holder side. As a result, we observe that our [TZ23]-based disclosure protocol performs $\approx 10\times$ better in proof computation, and $\approx 4\times$ better in proof verification. The code used to compute the benchmarks was taken from hyperledger's open-source implementation[6], which also adopts the `blsful` crate as underlying BLS library.

4.4 Adding Support for Batch Operations

In this section, we improve the scalability of our scheme, by adding support for batch updates. We start highlighting how the current update strategy limits the scalability of our scheme (Section 4.4.1), and describe how multiple deletions can be batched (Section 4.4.2). Then, we introduce techniques to efficiently perform batch operations (Section 4.4.3), and design an efficient way to aggregate multiple batch deletions into a single batch (Section 4.4.4). Afterwards, we extend our accumulator scheme with new primitives for batch operations (Section 4.4.5), and analyze the security of our new construction (Section 4.4.6). We conclude this section evaluating the performances (Sections 4.4.7 and 4.4.8) of our new primitives.

4.4.1 The update problem

The main client overhead of our preliminary scheme (Figure 4.1) comes from keeping witnesses *up-to-date*. Let $|a|$, $|d|$ respectively denote the number of credentials added to/removed from the accumulator after a witness was last updated. Updating a witness in our [KB21]-based scheme requires $m = |d|$ multiplications in $\mathbb{G}_1$ (as we only update on deletions). This improves the original update protocol proposal by [Ngu05], which required $m = |a| + |d|$ multiplications in $\mathbb{G}_1$. However, as noted in Section 4.2.4 (L2), the induced client overhead remains unpractical for national e-ID settings. This problem frequently arises when considering potential application of cryptographic accumulators in anonymous credential systems, and it has limited a real-world adoption of such solutions(e.g., [Dun22] shows scalability limitations of Hyperledger Indy).

The update strategy of both [Ngu05; KB21], consists in sequentially applying the single update algorithm m times. One may wonder whether it is possible to design an algorithm for updating witnesses *in batches* that runs in *constant time*. Unfortunately, for a positive dynamic accumulator, the answer is negative: in [CH10] the authors proof that updating a witness after m changes to the accumulator value requires $\Omega(m)$ operations. The lower bound comes from *reading* the input only (the update message itself has size $\Omega(m)$).

Nonetheless, such lower bound still gives us room for improvement: the main overhead of sequentially applying the single update algorithm is computing m multiplications in $\mathbb{G}_1$. Our goal is to come up with a *batch* update algorithm that, while being inevitably subjected to $\Omega(m)$ complexity, reduces the number of required *multiplications* in $\mathbb{G}_1$.

[6]See agora-allosaur-rs

4.4.2 Polynomials for Batch Updates

The authors of [VB22] designed a batch update protocol for a universal dynamic variant of [Ngu05] accumulator. Among their contributions is the ideation of *update polynomials*. Such polynomials are computed by the issuer to batch multiple deletion events[7], and distributed to all clients. Evaluating the polynomials on some accumulated element e yields the update for the associated witness.

Given a list of elements to delete $D = (e_1, \dots, e_m)$, and the accumulator's secret key x, the authors define the following update polynomials:

$$d(X) = \prod_{i=1}^{m}(e_i - X), \tag{4.3}$$

$$v(X) = \sum_{s=1}^{m}\left(\prod_{i=1}^{s}(e_i + x)^{-1}\prod_{j=1}^{s-1}(e_j - X)\right). \tag{4.4}$$

Calling V_t the initial accumulator, the accumulator's value after a batch deletion of all the elements in D can be computed as:

$$V_{t+1} = \frac{1}{\prod_{i=1}^{m}(e_i + x)}V_t = \frac{1}{d(-x)}V_t. \tag{4.5}$$

Furthermore, a holder can update his witness A_t by evaluating $d(X)$ and $v(X)$ on input his element e as follows:

$$A_{t+1} = \frac{1}{d(e)}(A_t - v(e)V_t). \tag{4.6}$$

Equation (4.6) can be proven by induction. The proof is quite straightforward, and can be found in the original paper [VB22, Appendix D].

Updating witnesses using eq. (4.6) would be *extremely* efficient, as both $d(X)$, and $v(X)$ are defined over $\mathbb{F}_p$. However, the authors observe that $v(X)$ leaks information about the accumulator's key x, hence it cannot be published directly. Instead, the issuer publishes the polynomial $\Omega(X) \leftarrow v(X)V_t$. After re-writing $v(X) = \sum_{i=0}^{m-1} c_i X^i$ for some coefficients $c_0, \dots, c_{m-1} \in \mathbb{F}_p$, the polynomial $\Omega(X)$ is defined as follows:

$$\Omega(X) = V_t \sum_{i=0}^{m-1} c_i X^i. \tag{4.7}$$

Then, by substituting $\Omega(e)$ in eq. (4.6), we get the equation for computing witness batch updates:

$$A_{t+1} = \frac{1}{d(e)}(A_t - \Omega(e)) \tag{4.8}$$

[7]As we only update on deletions, we do not consider polynomials batching additions.

Note that the coefficients $\Omega = (c_0 V_t, \dots, c_m V_t)$ are now defined in $\mathbb{G}_1$, where the Discrete Logarithm problem is assumed to be hard. In Section 4.4.6, we proof that the update polynomials do not leak any more information than the single update messages generated by a sequential application of $\mathbf{Del}_{\mathcal{I}}$.

4.4.3 On Efficient Polynomial Evaluation Techniques

In this section, we investigate efficient ways for evaluating the batch update polynomials defined in the previous section.

Applying the batch update equation (4.8), requires evaluating polynomials $d(X) \in \mathbb{F}_p[X]$, and $\Omega(X) \in \mathbb{G}_1[X]$, on input the holder's element e. As both polynomials have m coefficients, a direct evaluation of $\Omega(e)$ requires m multiplications in $\mathbb{G}_1$.

In [VB22], users are both given a membership *and* a non-membership witness. The result of the polynomial evaluation can be used to update both witnesses and only requires $m + 2$ multiplications, instead of the $2m$ multiplications required by sequentially updating both witnesses[8].

However, our e-ID holders are provided with a membership witness *only*, as it suffices for proving non-revocation. As we said, a sequential application of the single update algorithm also requires m point multiplications. Hence, the batch update strategy of [VB22] does not seem to after any advantage compared to the trivial approach.

At this point, one may wonder if there are more efficient approaches for evaluating the polynomial $\Omega(X)$. The underlying problem takes the name of Multi Scalar Multiplication (MSM), and its importance has grown recently due to its applications in zk-SNARKs (e.g., Zcash [Hop+16], and Turbo-PLONK [Spo+20]).

The MSM problem can be defined as follows:

Def (MSM problem)**.** *Let $\mathbb{G}$ be a group of prime order p. Given a vector of scalars $\vec{c} = (c_0, \cdots, c_{n-1}) \in \mathbb{F}_p^n$, and a vector of coefficients $\vec{P} = (P_0, \dots, P_{n-1}) \in \mathbb{G}^n$, evaluate the following sum:*

$$S = \sum_{i=0}^{n-1} c_i \cdot P_i.$$

In our setting, $\vec{P}$ is the coefficient vector $\vec{\Omega} \in \mathbb{G}_1^m$ of the polynomial $\Omega(X)$, while each c_i is the i-th power of the holder's element e evaluated in $\mathbb{F}_p$.

The trivial algorithm adopted by [VB22] computes the result by definition: using the double-add algorithm, each scalar-point multiplication would

[8]Membership and non-membership witnesses have different update algorithms

```
1. $res \leftarrow \sum_{P_i \; s.t. \; c_{i,j}=b-1} P_i$;

2. $temp \leftarrow res$;

3. for $k \in \{b-2, \dots, 2\}$:
    1. $temp \leftarrow temp + \sum_{P_i \; s.t. \; c_{i,j}=k} P_i$;
    2. $res \leftarrow res + temp$;

4. $res \leftarrow res + \sum_{P_i \; s.t. \; c_{i,j}=1} P_i$;
```

Figure 4.4: Inner Sum Computation in Pippenger Approach

require $2 \log p$ additions [9] in $\mathbb{G}_1$ (in the worst case). Hence, it has a total complexity of $2m \log p$ additions.

State-of-the-art approaches are optimized variants of Pippenger's multi-exponentiation algorithm, applied to MSM as in [Ber+12, Section 4]. The general idea is to split the $\log p$ bits of each coefficient into $h = \log p/w$ windows of size w. Defining $b = 2^w$, each coefficient c_i can be expressed in base-b as $c_i = \sum_{j=0}^{h-1} c_{i,j} b^j$, where $0 \leq c_{i,j} < b$. Using our new base-b representation, we re-write S as:

$$
\begin{aligned}
S &= \sum_{i=0}^{n-1} c_i \cdot P_i \\
&= \sum_{i=0}^{n-1} \sum_{j=0}^{h-1} c_{i,j} \cdot b^j \cdot P_i \\
&= \sum_{j=0}^{h-1} b^j \sum_{i=0}^{n-1} c_{i,j} \cdot P_i \\
&= \sum_{j=0}^{h-1} b^j \left(\sum_{k=0}^{b-1} k \sum_{P_i \; s.t. \; c_{i,j}=k} P_i \right)
\end{aligned}
$$

The *inner* sum (within the parenthesis) can be computed as in Figure 4.4. In the figure, computing Line 1, Line 3.1, and Line 4 require summing over all $P_i \in \vec{P}$, which costs n additions in $\mathbb{G}$. Line 3.2 costs $b-3$ additions in $\mathbb{G}$. Altogether, the computation of all inner sums requires $h(n+b-3) \simeq h(n+b)$ additions in $\mathbb{G}$.

[9] For simplicity, we do not consider the difference between point doubling and point

The *outer* sum can be computed by iterating in reverse order, with one addition and w doubling per iteration (remember that $b = 2^w$). This consists in $h(w + 1)$ additions in $\mathbb{G}$.

Overall, the total complexity of evaluating S is of $h(n + b) + h(w + 1) \approx h(n + b)$ additions in $\mathbb{G}$.

In our setting, Pippenger's approach with $w \leftarrow \log m$ (i.e., $h = \log p / \log m$), evaluates $\Omega(e)$ with roughly $2m \log p / \log m$ additions in $\mathbb{G}_1$. Note that this *improves* the naive solution by a $\log m$ factor. In Section 4.4.8, we show how this reduction in complexity translate to actual performance improvements.

4.4.4 Aggregating Multiple Batches

In the previous section, we described an efficient method for computing the update after a *single* batch deletion event. However, at the time a holder updates, *multiple* batch deletion events may have occurred. In this section, we design an efficient method for aggregating multiple batch deletions into a single batch update.

Let us assume that n batch deletion happened since the client last updated. The i-th batch deletion, is associated with a list of deleted elements $D_i = (e_{i,1}, \ldots, e_{i,m_i})$, and respective update polynomials $\Delta_i := (\Omega_i(X), d_i(X))$. The holder would then need to download the list update polynomials $\Delta_i^n := (\Delta_i)_{i=1}^n$, and sequentially apply the update as per eq. (4.8).

However, due to the $(\log m)^{-1}$ factor in Pippenger's complexity, the efficiency of the update evaluation *depends* on the sizes of the single batch deletions. For instance, sequentially updating after three batch deletions of size m costs $\approx 6m\frac{\log p}{\log m}$ additions, while updating after a batch deletion of size $3m$ costs $\approx 6m\frac{\log p}{\log 3m}$ additions. Note that, in the worst case were each $m_i = 1$ (i.e., the issuer only performs single updates), the update is performed with the single update algorithm, which costs $6m \log p$ additions.

In [VB22, Section 4.2], the authors propose a way to aggregate a list of batch updates Δ_{i+1}^j, into a single update $\Delta_{i\to j} := (\Omega_{i\to j}(X), d_{i\to j}(X))$ such that:

$$A_{i+j} = \frac{1}{d_{i\to j}(e)}(A_i - \Omega_{i\to j}(e)). \tag{4.9}$$

By definition (4.3), it follows:

$$d_{i\to j}(X) = \prod_{t=i+1}^{j} d_t(X). \tag{4.10}$$

additions.

Furthermore, the authors proof that:

$$\Omega_{i\to j}(X) = \sum_{t=i+1}^{j} d_{i\to t-1}(X) \cdot \Omega_t(X), \tag{4.11}$$

with $d_{i\to i}(X) \coloneqq 1$.

Correctness of eq. (4.11) is proven in [VB22, Appendix D.2]. Vitto et al. compute $\Omega_{i\to j}(e)$ by directly evaluating eq. (4.11), which requires a linear number of multiplications in $\mathbb{G}_1$.

Instead, we can achieve better performances as follows: for a list of update polynomials $(\Omega_t(X), d_t(X))_{t\in\{i+1,\ldots,j\}}$, let $\Omega_t(e) = \sum_{k=1}^{m_t} e^{k-1}\Omega_{t,k}$. Furthermore, let $d_{i\to t-1}(e) = a_t$ for $t \in \{i+1,\ldots,j\}$.

We can rewrite eq. (4.11) as:

$$\begin{aligned}\Omega_{i\to j}(e) &= \sum_{t=i+1}^{j} d_{i\to t-1}(e)\ \Omega_t(e)\\ &= \sum_{t=i+1}^{j} a_t \sum_{k=1}^{m_t} e^{k-1}\Omega_{t,k}\\ &= \sum_{t=i+1}^{j}\sum_{k=1}^{m_t}(a_t \cdot e^{k-1})\ \Omega_{t,k} \end{aligned}\tag{4.12}$$

Computing all coefficients $(a_t \cdot e^{k-1})$, has total cost $m = \sum_{t=i+1}^{j} m_t$ *scalar* multiplications. Note that unlike the method in [VB22], eq. (4.12) does not require *any* multiplication in $\mathbb{G}_1$.

At this point, evaluating $\Omega_{i\to j}(e)$ reduces to solving an instance of the MSM problem with coefficients of length m. Hence, exactly as in Section 4.4.3, we can use Pippenger's approach to speed up the evaluation of $\Omega_{i\to j}(e)$, with a cost of $\approx 2m\log p/\log m$ multiplications in $\mathbb{G}_1$.

4.4.5 Adding Support for Batch Updates

In fig. 4.5, we define primitives for batch operations. As done in the previous section, we let $D = (e_1,\ldots,e_m)$ denote a list of elements that the issuer wants to revoke, and Δ the respective update polynomials. Furthermore, we let $\Delta_{i+1}^{j} \coloneqq (\Delta_t)_{t=i+1}^{j}$ denote the list of update polynomials that, after being applied to a witness A_i, yield the updated witness A_j.

Creating Update Polynomials the issuer runs **GenUpPoly**$_{\mathcal{I}}$ to create the batch update polynomials from the deletion list D_{t+1}. He computes the polynomials $d_{t+1}(X)$, $\Omega_{t+1}(X)$ according to the equations introduced in Section 4.4.2, and returns the update Δ_{t+1}.

```
$\mathbf{GenUpPoly}_{\mathcal{I}}(x, V_t, D_{t+1})$
1. $(e_1, \ldots, e_m) \leftarrow D_{t+1}$;
2. compute $d_{t+1}(X)$ as per eq. (4.3)
3. compute $\Omega_{t+1}(X)$ as per eq. (4.7);
4. $\Delta_{t+1} \leftarrow (d(X), \Omega(X))$;
5. return $\Delta_{t+1}$;
```

```
$\mathbf{DelBatch}_{\mathcal{I}}(x, V_t, D_{t+1}, S_k)$
1. if $D_{t+1} \not\subseteq S_k$ : return $\perp$;
2. $\mathbf{GenUpPoly}_{\mathcal{I}}(x, V_t, D_{t+1}) \rightarrow \Delta_{t+1}$;
3. $(d_{t+1}(X), \Omega_{t+1}(X)) \leftarrow \Delta_{t+1}$
4. $V_{t+1} \leftarrow \frac{1}{d_{t+1}(-x)} V_t$
5. $S_{k+1} \leftarrow S_k \setminus D$;
6. return $(V_{t+1}, \Delta_{t+1}, S_{k+1})$
```

```
$\mathbf{AggUpPoly}_{\mathcal{H}}(e, \Delta_{t+1}^{t+n})$
1. $\{d_i(X),\ \Omega_i(X)\}_{i=t+1}^{t+n} \leftarrow \Delta_{t+1}^{t+n}$;
2. $(\Omega_{i,1}, \ldots, \Omega_{i,m_i}) \leftarrow \vec{\Omega}_i$;
3. if $\exists\ d_i(X)\ s.t.\ d_i(e) = 0$ : return $\perp$;
4. $a_{t+1} \leftarrow 1$;
5. for $i \in \{t+2, \ldots, t+n\}$:
6.     $a_i \leftarrow a_{i-1} \cdot d_{i-1}(e)$;
7. for $i \in \{t+1, \ldots, t+n\}$, $k \in \{1, \ldots, m_i\}$:
8.     $c_{i,k} \leftarrow a_i \cdot e^{k-1}$
9. $\Omega_{t \to t+n}(e) \leftarrow \sum_{i=t+1}^{t+n} \sum_{k=1}^{m_i} c_{i,k} \Omega_{i,k}$;
10. $d_{t \to t+n}(e) \leftarrow a_{t+n} \cdot d_{t+1}(e)$;
11. return $(d_{t \to t+n}(e), \Omega_{t \to t+n}(e))$
```

```
$\mathbf{WitUpBatch}_{\mathcal{H}}(V_t, e, A_t, \Delta_{t+1}^{t+n})$
1. $\mathbf{AggUpPoly}_{\mathcal{H}}(e, \Delta_{t+1}^{t+n}) \rightarrow (d_{t \to t+n}(e),$
   $\Omega_{t \to t+n}(e))$;
2. $A_{t+n} \leftarrow \frac{1}{d_{t \to t+n}(e)} (A_t - \Omega_{t \to t+n}(e))$;
3. return $A_{t+n}$
```

Figure 4.5: Primitives for batch update operations

Batch Deletions the issuer runs $\mathbf{DelBatch}_{\mathcal{I}}$ to perform a batch deletion of all elements in the deletion list D_{t+1}. First the issuer checks that all the element to delete are in the accumulated set, and aborts otherwise. Then, he generates the polynomials for D_{t+1} using $\mathbf{GenUpPoly}_{\mathcal{I}}$. Finally, he derives the new accumulator value V_{t+1} (eq. (4.5)), and updates the issuer's state.

Aggregating updates the holder runs $\mathbf{AggUpPoly}_{\mathcal{H}}$ to compute the aggregated evaluation of a list of batch deletion events. First, the holder checks that he has not been revoked (i.e., $\nexists\ d_i\ s.t.\ d_i(e) = 0$), and fails otherwise. Then, he computes the evaluation of the aggregated update polynomials $\Omega_{t \to t+n}(e)$ (Lines 4-9) as by eq. (4.10), and $d_{t \to t+n}(e)$ (Line 10) as by eq. (4.11).

Witness update the holder runs $\mathbf{WitUpBatch}_{\mathcal{H}}$ to update his witness after a sequence of batch deletion events. First, the holder calls $\mathbf{AggUpPoly}_{\mathcal{H}}$ to get the aggregated evaluations $\Omega_{t \to t+n}(e), d_{t \to t+n}(e)$. Then, he applies the update as per eq. (4.9).

4.4.6 Soundness of the Extended Construction

In the previous section, we expanded our non-adaptively sound accumulator, introducing support for batch updates. In this section, we show that batch update polynomials do not leak *any* additional information compared to the single update messages generated by a sequential execution of the deletion algorithm.

Consider an initial accumulator value V_0, and pair of update polynomials $(\Omega(X), d(X))$ defined as in Section 4.4, and batching deletions of all elements in $D = (e_1, \dots, e_m)$. For $i \in [m]$, each sequential execution of $\mathbf{Del}_{\mathcal{I}}$ on input e_i, returns the update message $upmsg_i = (e_i, V_i)$, where $V_i = \frac{1}{x+e_i} V_{i-1}$ is the new accumulator value.

Clearly, anyone having access to $\{upmsg_i\}_{i\in[m]}$ can compute:

$$d(X) = \prod_{i=1}^{m}(e_i - X).$$

Let us now consider the polynomial $\Omega(X)$. By rewriting eq. (4.7), we have:

$$\begin{aligned}\Omega(X) &= V_0 \sum_{s=1}^{m}\prod_{i=1}^{s}(e_i + x)^{-1}\prod_{i=1}^{s-1}(e_i - X)\\ &= \sum_{s=1}^{m} V_0 \prod_{i=1}^{s}(e_i + x)^{-1}\prod_{i=1}^{s-1}(e_i - X)\\ &= \sum_{s=1}^{m} V_s \prod_{i=1}^{s-1}(e_i - X).\end{aligned}$$

The update messages contain each accumulator value V_s for $s \in [m]$. Therefore, anyone having access to $\{upmsg_i\}_{i\in[m]}$ can compute $\Omega(X)$ in polynomial time. As both update polynomials do not leak any additional information, our new construction preserves the non-adaptive soundness property of the initial construction.

4.4.7 Theoretical Analysis Evaluation

In this Tables 4.9 and 4.10, we analyze the performances of the new primitives defined in the previous section. We only indicate the costs for $\mathbf{WitUpBatch}_{\mathcal{H}}$, and $\mathbf{DelBatch}_{\mathcal{I}}$, as they are mainly determined by the underlying calls to $\mathbf{AggUpPoly}_{\mathcal{H}}$, and $\mathbf{GenUpPoly}_{\mathcal{I}}$ respectively.

When evaluating $\mathbf{WitUpBatch}_{\mathcal{H}}$, we let n denote the number of batch updates *aggregated* by the holder, m_{max} the maximum number of elements deleted in a *single* batch update (i.e., $m_{max} = max_{i\in[1..n]}\ m_i$), and m the total number of deleted elements (i.e., $m = m_1 + ... + m_n$). As in the previous sections, Table 4.9 measures the computational cost in terms of group operations, while Table 4.10 counts the number of exchanged bits.

Batch Deletion Line 3 of $\mathbf{DelBatch}_{\mathcal{I}}$ requires computing the update polynomials. Computing $v(X)$ according to eq. (4.4) requires $\approx m^2$ multiplications. Computing $\Omega(x)$ requires multiplying each coefficient of $v(x)$

$\mathbf{WitUpBatch}_{\mathcal{H}}$		$\mathbf{DelBatch}_{\mathcal{I}}$	
$\mathbb{G}_1$	$\mathbb{F}_p$	$\mathbb{G}_1$	$\mathbb{F}_p$
$2m\frac{\log p}{\log m}A$	$3mM$	mM	m^2M

Table 4.9: Required operations for computing primitives in Figure 4.5

$\mathbf{WitUpBatch}_{\mathcal{H}}$	$\mathbf{DelBatch}_{\mathcal{I}}$
$m(\log q + \log p)$	$m(\log q + \log p)$

Table 4.10: Communication cost (bits) of primitives in Figure 4.5

by the accumulator V_t. As $v(x)$ has degree $m - 1$, this can be done in m multiplications in $\mathbb{G}_1$ (even less using window techniques). The polynomial $d(X)$ can be computed from $v(X)$ by maintaining an intermediate result, with only $2(m - 1)$ additional multiplications.

The communication cost of distributing the update remains the same of the non-batched approach, as $\vec{\Omega} \in \mathbb{G}_1^m$, and $\vec{d} \in \mathbb{F}_p^m$.

Batch Witness Update Line 5 of $\mathbf{AggUpPoly}_{\mathcal{H}}$ requires $m - m_1$ multiplications in $\mathbb{F}_p$ to evaluate each $d_i(e)$, and other $n - 1$ multiplications to compute each a_i. Line 8 requires m_{max} multiplications in $\mathbb{F}_p$ to precompute each e^{k-1}, and other m multiplications in $\mathbb{F}_p$ to compute each coefficient $c_{i,k}$. Line 9 requires $\approx (2m\frac{\log p}{\log m})$ additions in $\mathbb{G}_1$ to compute the MSM with Pippenger's approach, while line 10 requires other $m_1 + 1$ multiplications in $\mathbb{F}_p$. At the end, we have a total complexity of around $3m$ multiplications[10] in $\mathbb{F}_p$, and $2m\frac{\log p}{\log m}$ additions in $\mathbb{G}_1$.

Line 2 of $\mathbf{WitUpBatch}_{\mathcal{H}}$ only adds 1 inversion in $\mathbb{F}_p$, and 1 addition and 1 multiplication in $\mathbb{G}_1$ to the overhead of calling $\mathbf{AggUpPoly}_{\mathcal{H}}$. Hence, we can consider the computational cost of $\mathbf{WitUpBatch}_{\mathcal{H}}$ to be the same as the one of $\mathbf{AggUpPoly}_{\mathcal{H}}$.

4.4.8 Software Implementation

In this section, we evaluate the performances of the two primitives described in *Figure* 4.5.

Batch Deletion In table 4.11, we observe that, for small values of m, the cost of $\mathbf{DelBatch}_{\mathcal{I}}$ is comparable to a sequential application of $\mathbf{Del}_{\mathcal{I}}$. However, the performance rapidly degrade due to the quadratic complexity of creating update polynomials. For $m = 2^{14}$, batch deletion becomes $\approx 6.5\times$ slower than sequential deletion.

[10] Note that as each $m_i \geq 1$, we have $(n - 1) + m_{max} \leq m$. Therefore $2m + (n - 1) + m_{max} \leq 3m$.

	Batch Operations		**Sequential Operations**	
m	$\mathbf{DelBatch}_{\mathcal{I}}$	$\mathbf{WitUpBatch}_{\mathcal{H}}$	$\mathbf{Del}_{\mathcal{I}}$	$\mathbf{WitUp}_{\mathcal{H}}$
1	0.207 ms	0.076 ms	0.076 ms	0.076 ms
2^2	0.586 ms	0.302 ms	0.306 ms	0.308 ms
2^4	2.137 ms	1.233 ms	1.235 ms	1.2147 ms
2^6	6.690 ms	4.855 ms	4.977 ms	5.061 ms
2^8	26.920 ms	12.685 ms	19.885 ms	19.871 ms
2^{10}	123.98 ms	34.094 ms	78.493 ms	80.218 ms
2^{12}	787.76 ms	111.13 ms	314.84 ms	321.31 ms
2^{14}	8325.2 ms	355.80 ms	1262.7 ms	1279.5 ms

Table 4.11: Average computational costs for the primitives in Figure 4.1

m	**Uploaded/Downloaded Bytes**
1	0.08 KB
2^2	0.32 KB
2^4	1.28 KB
2^6	5.12 KB
2^8	20.48 KB
2^{10}	81.92 KB
2^{12}	327.68 KB
2^{14}	1310.72 KB

Table 4.12: Communication cost for increasing m values. The costs are equal for all primitives in Table 4.11

The issuer is assumed to have high computational power and does not need to *immediately* publish the update, so the he overhead added by polynomial computation could be considered acceptable. However, thanks to the efficient method for aggregated evaluation we designed in Section 4.4.4, the issuer can completely avoid computing update polynomials and release *individual updates only*. For example, instead of computing one polynomial of degree 2^{14} in 8.3s, the issuer could compute 2^{14} individual updates in around 2.8s. Then, each holder can efficiently aggregate the 2^{14} updates in a *single* update polynomial, and evaluate the result with Pippenger's approach. In Figure 4.7, we show that aggregating single deletions and evaluating the resulting polynomial does not introduce any significant overhead compared to directly evaluating a single polynomial representing the entire batch.

Batch Updates In Table 4.9, we see that for large enough m values (e.g., $m > 64$), $\mathbf{WitUpBatch}_{\mathcal{H}}$ sensibly improves the run-time of $\mathbf{WitUp}_{\mathcal{H}}$. In Figure 4.5, we compared the two algorithms up to $m = 20\,000$. As expected, when the batch size increases, $\mathbf{WitUpBatch}_{\mathcal{H}}$ becomes increasingly more

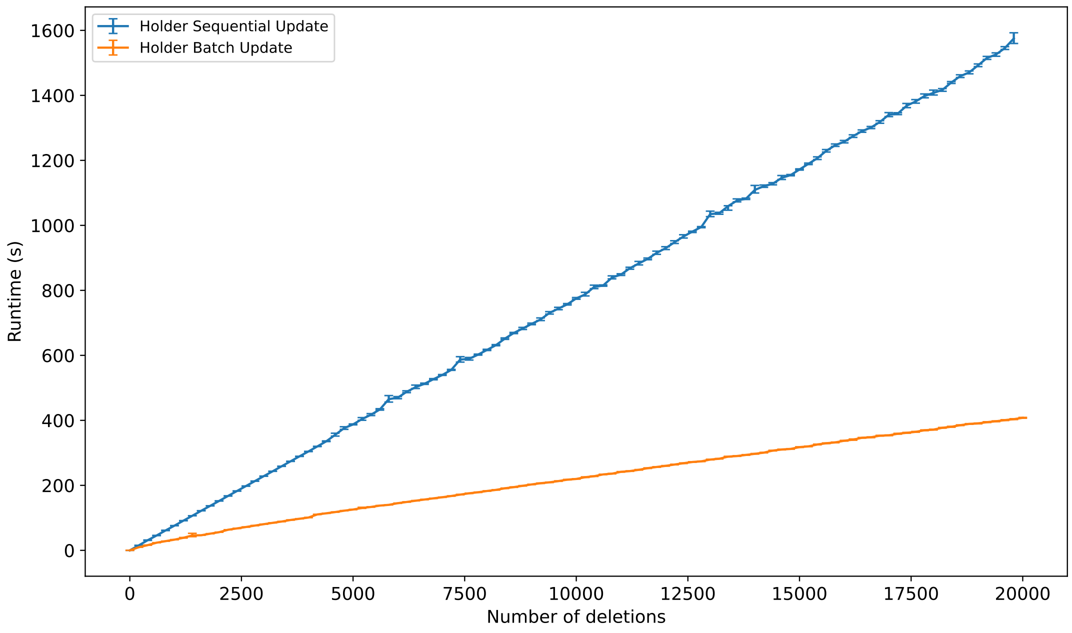

Figure 4.6: $\mathbf{WitUpBatch}_{\mathcal{H}}$ vs $\mathbf{WitUp}_{\mathcal{H}}$ on an increasing number of deletions

efficient than $\mathbf{WitUp}_{\mathcal{H}}$. For instance, when $m = 5000$ updating via sequential application of $\mathbf{WitUp}_{\mathcal{H}}$ requires around 0.383, while $\mathbf{WitUpBatch}_{\mathcal{H}}$ takes $0.125s$, yielding a $3\times$ improvement. Instead, when $m = 20\,000$, updating via $\mathbf{WitUp}_{\mathcal{H}}$ requires around $1.58s$, while $\mathbf{WitUpBatch}_{\mathcal{H}}$ only takes $0.4s$, improving by $4\times$.

Aggregating Multiple Batches In Figure 4.7, we can appreciate the advantages of aggregating multiple update polynomials. In the experiment, we fix a total of 5000 deletions, which we split into multiple batches, all of equal size[11]. As expected, by varying the size of the individual batches (i.e., each m_i), we observe differences in the performances of the batch update algorithm. When the batch sizes are very small (e.g., $m_i = 1$), the run-time of the batch update algorithm *without* aggregation is close to the single update algorithm. As the batch sizes become larger, the performances become similar to those in *Figure* 4.6. On the opposite, the cost of the batch update algorithm with aggregation remains *constant*, independently on the size of the individual batches.

[11]Except for the last batch which could be smaller than the others

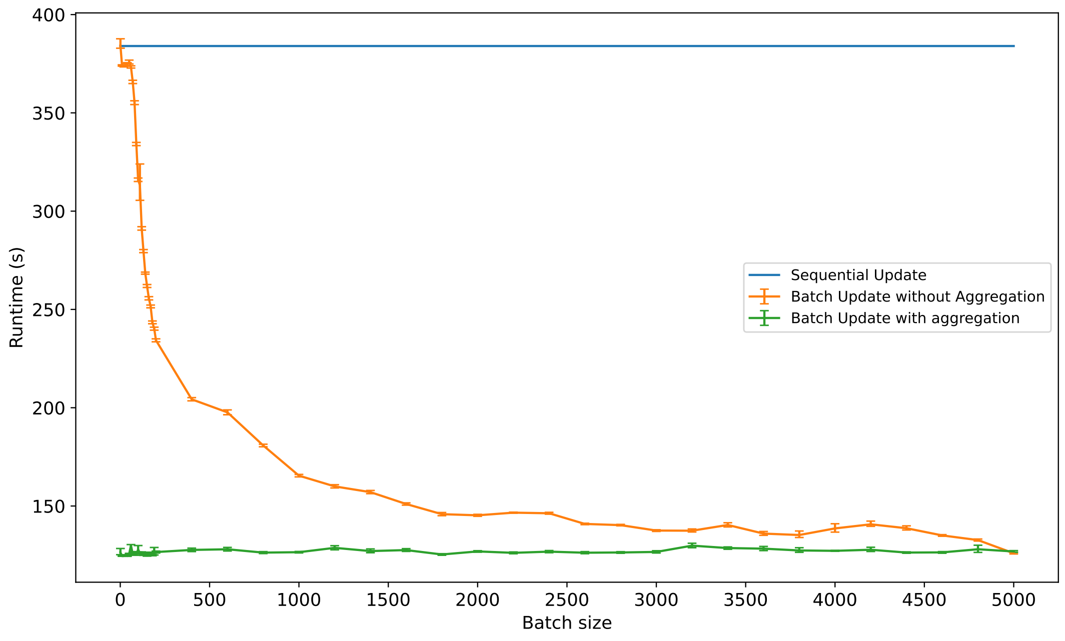

Figure 4.7: Comparing performances of **WitUpBatch**$_{\mathcal{H}}$ with/without aggregation on $m = 5000$ deletions split in sub-batches of equal size m_i, varying from $m_i = 1$ to $m_i = 5000$.

Final remarks Figure 4.7 shows that, for $m = 5000$ total deletions, computing an aggregated evaluation of the individual deletions does not introduce any considerable overhead compared to evaluating all deletions as a single batch. As we anticipated in the **Batch Deletion** paragraph, this cancels the need of computing update polynomials.

Since computing the coefficients for the aggregated evaluation introduces an overhead of m scalar multiplications, one could think that for larger update sizes the performances would be sensibly affected. However, in the `blsful` implementation, one multiplications in $\mathbb{F}_p$ only takes around 18 ns. Therefore, for reasonable batch sizes, the overhead of the additional m scalar multiplications can be considered negligible.

Finally, we note that there optimized versions of Pippenger's approach, and some of them could further reduce the cost of polynomial evaluation. For instance, rewriting the polynomial in Non-Adjacent Form (NAF) would restrict each coefficient to the range $[-b/2, b/2]$, where b is the based used in the MSM. Exploiting the fact that $(-b/2)P = (b/2)(-P)$ for every $P \in \mathbb{G}_1$, we could reduce the complexity of the evaluation from $2m\frac{\log p}{\log q}$ to $\frac{3}{2}m\frac{\log p}{\log q}$

additions in $\mathbb{G}_1$[12]. A more in-depth study on efficient polynomial evaluation techniques is left as interesting future work.

4.5 Improving Update Scalability with Revocation Epochs

In this section, we further improve the scalability of our scheme by introducing the concept of *revocation epochs.* We start by noticing that witness updates still require high communication costs (Section 4.5.1), and we propose a method for limiting such costs (Section 4.5.2). Then, we modify our scheme to support the new proposal (Section 4.5.3), and theoretically analyze the performances (Section 4.4.7). Finally, we simulate a realistic e-ID scenario to show the practical advantages of our proposal (Section 4.5.5), and conclude this chapter by exposing the remaining limitations (Section 4.5.6).

4.5.1 On the Communication Overhead of Witness Updates

In Section 4.4, we reduced the computational cost of updating witnesses by supporting batch operations. However, we noted that batching does not decrease communication costs: after m revocations, the update still involves downloading $m(\log p + \log q)$ bits. The update size remains the same whether we download the batch update polynomials generated by the batch deletion algorithm or the m update messages produced by the single-update algorithm. As shown in Table 4.12, for large m values, this requires holders to download several *hundreds* of kilobytes. Unfortunately, the communication lower-bound in [CH10] indicates that our scheme is already communication-optimal.

A remaining way for reducing the communication complexity would be by reducing the number of revocations. Obviously, we cannot set an upper bound on the maximum revocation rate, as it would make our scheme impractical. Nonetheless, we can set a limit on the number of revocations for which a user *needs to compute* his update. A trivial solution would be setting a threshold on the maximum update size: when the threshold is reached, the holder directly asks for his *specific* up-to-date witness. However, as the holder asks for his own witness, this solution is not privacy-preserving[13].

4.5.2 Limiting Maximum Updates with Revocation Epochs

The issue with the previous approach is that a holder only updates when his witness is not fresh enough to be accepted by a verifier. This implies that holder updates are *tied* to subsequent credential presentations: linking

[12]see e.g., `https://hackmd.io/jNXoVmBaSSmE1Z9zgvRW_w` for details

[13]If the user asks for more witnesses, either he achieves a constant-sized anonymity

```
**WitUpAll**$_{\mathcal{I}}(x, S_k)$
1. $V_0 \leftarrow\!\$\, \mathbb{G}_1$;
2. $W \leftarrow \emptyset$;
3. **foreach** $e \in S_k$:
4.         $A_0 = \frac{1}{x+e} V_0$;
5.         $W \leftarrow W \cup (e, A_0)$;
6. **return** $(V_0, W)$
```

Figure 4.8: Primitive for updating all witnesses

a holder to a specific update event exposes sensitive *timing information.*

To break the timing relation, we propose to introduce fixed time intervals in which *everyone* fetches his updated witness. We call this fixed intervals *revocation epochs*[14]. At the end of each epoch, new witnesses can be distributed as software updates, similarly as in CRLite [Lar+17]. Note that in our case, the update size is constant (e.g., 48 bytes for BLS12-381).

After adding revocation epochs, we can always assume that (in the worst case) the holder's witness was last updated at the beginning of the current epoch. A holder would then need to update only for the number of revocation that happened *within* the current epoch. Consequently, by bounding the *time length* of each epoch, we indirectly bound the update size.

4.5.3 Adding Revocation Epochs

In this Figure 4.8, we define the primitive **WitUpAll**$_{\mathcal{I}}$, which is used by the issuer at the end of each epoch. In Line 1, the Issuer initializes an accumulator V_0 for the new epoch, invalidating all existing witnesses. In Line 3, he derives witnesses for each non-revoked user in the system, using the accumulator's private key. Note that no single update message is returned, as every holder will fetch his specific update.

At the end of each epoch the Issuer can also rotate the accumulator secret key: instead of feeding **WitUpAll**$_{\mathcal{I}}$ with the old secret key x, the issuer can use a *new* secret key x'. This *automatically* distributes to every user an up-to-date witness that is valid under the new accumulator's public key $X' \leftarrow x'G_2$.

4.5.4 Theoretical Analysis

In Table 4.13, we analyze the computational performances of **WitUpAll**$_{\mathcal{I}}$. We let n_h indicate the number of non-revoked holders in the system (i.e., $n_h = |S|$). Line 4 can be done in roughly $n_h \log p$ additions in $\mathbb{G}_1$ (i.e., around

set, or he downloads a non-constant number of witnesses. Hence, having large-enough anonymity sets would introduce huge communication costs (e.g., 80 MB for a set size of 1M).

[14]Note that revocation epochs (or update epochs) have already been described by other

$n_h/2$ multiplications), and n_h additions and inversions in $\mathbb{F}_p$. Note that this a $2\times$ improvement compared to the naive algorithm that requires n_h multiplications in $\mathbb{G}_1$ (i.e., $2n_h \log p$ additions). In the following paragraph we briefly explain how we get such improvement.

WitUpAll$_\mathcal{I}$	
$\mathbb{G}_1$	$\mathbb{F}_p$
$n_h \log p$ A	$n_h(1\text{I} + 1\text{A})$

Table 4.13: Required operations for **WitUpAll$_\mathcal{I}$**

Speeding up witness generation with window methods The algorithm **WitUpAll$_\mathcal{I}$** requires computing the list $a_1V_0, \dots, a_{n_h}V_0$, where $a_i = \frac{1}{e_i+x}$ for each $e_i \in S_k$. Each multiplication with the double-add algorithm requires $2\log p$ additions in $\mathbb{G}_1$, hence the naive algorithm has a total complexity of $2n_h \log p$ additions in $\mathbb{G}_1$. Instead, for some $c \neq 0$ and $b = 2^c$, we can rewrite each coefficient in base-b as:

$$a_i = \sum_{j=0}^{\lceil \log p/c \rceil} b^i a_{i,j},$$

where each $a_{i,j} < b$. Then, each multiplication can be re-written as:

$$a_i V = \sum_{j=0}^{\lceil \log p/c \rceil} b^i a_{i,j} V,$$

The issuer can pre-compute each $a_{i,j}V$ at the cost of b additions in $\mathbb{G}_1$. Using the pre-computed coefficients, calculating each single witness a_iV only requires $(c+1)\log p/c$ additions. Hence, computing the all list costs $b+n_h(c+1)\lceil \log p/c \rceil \simeq b + n_h \log p$ additions in $\mathbb{G}_1$. By setting $c = \log n_h$, we get a total of $n_h(\log p + 1)$, which is roughly $n_h \log p$ additions in $\mathbb{G}_1$.

4.5.5 Practical Advantages of Revocation Epochs

In Figure 4.9, we illustrate the practical impact of introducing revocation epochs in a simulation of our e-ID setting.

Assuming to have weekly revocation epochs, we compute the estimated number of updates that a holder would need to compute after an increasing number of days of inactivity. To estimate the number of revocation occurring within a given time period we assume to have 10M users, and a 2% revocation rate, uniformly distributed over the year. Clearly, in a real world scenario, the revocation rate would not have such uniform distribution. However, note that a 2% rate is likely to be much higher than what we would experience in the real world. As a comparison, the revocation rate in the PKI says well below 1%[15], according to the measurements in [KC21].

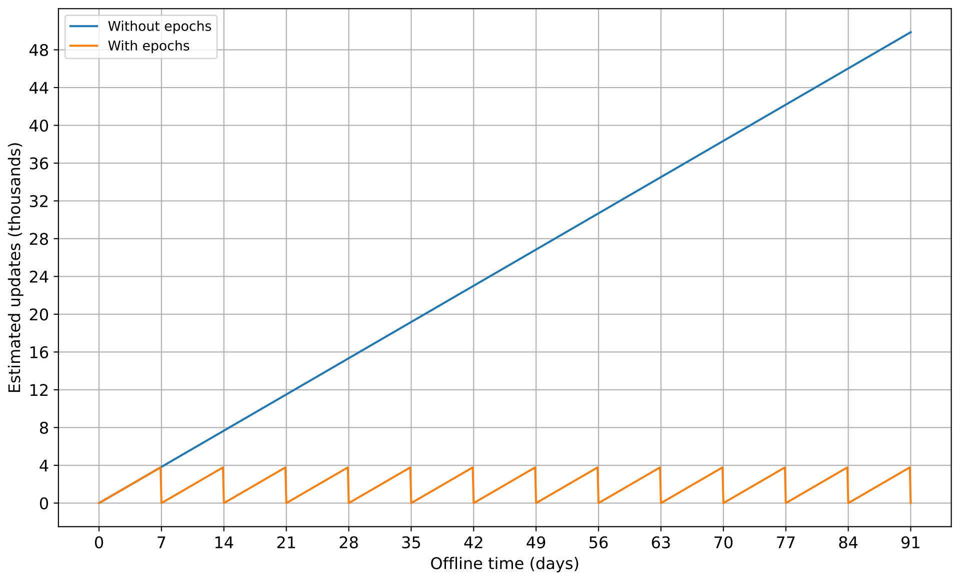

Figure 4.9: Number of estimated holder updates after an increasing number of offline days, comparing approaches with vs without revocation epochs.

We notice that after only a month of inactivity, a user would need to catch up with around 16.5K revocations. In BLS12-381, this would require downloading around 1.3**MB**, and $\approx$ 0.36s of computation on a high-end laptop (according to the benchmarks in Table 4.11). On the other hand, when using epochs the user only needs to catch up with the number of weekly revocations, which are at most 4K independently on when he last presented his credential. Using the same benchmarks as before, this would only require downloading around 320KB of data and $\approx$ 0.11s of computation.

Finally, we note that our choice of having *weekly* revocation epochs was based on the number of users, and on the revocation rate that we were considering for the simulation. After deploying the system and gathering data, the length of the epochs should be adjusted, also considering the maximum client overhead we deem acceptable. For instance, in the Swiss e-ID setting

papers. However, to the best of our knowledge, they were never used for periodically distributing up-to-date witnesses.

[15]Except during massive revocation events.

we would likely have less users and a smaller revocation rate, hence it could be possible to have longer epochs (e.g., one month).

4.5.6 Remaining Limitations

In the previous sections, we highlighted the advantages of introducing revocation epochs in our accumulator construction. However, there are also some shortcomings:

- **Flexibility**: setting revocation epochs means that every user need to fetch his updated witness at fixed time intervals, even if he will never present his credential during the entire epoch. If epochs are too short, this could generate an excessive communication overhead on the system. Furthermore, the update should be done regularly (e.g., in the background). If a holder asks for his up-to-date witness just before presentation, he become linkable, and the advantage of using epochs is lost.
- **Communication Costs**: even though revocation epochs can considerably reduce communication costs, high revocation rates or long epochs might still necessitate downloading hundreds of kilobytes, as in the case of our simulation. Although this overhead might be considered manageable with today's internet, it represent a significant increase in communication compared to the 48 bytes required to download a single witness.

In Chapter 6, we will explore alternative techniques for updating witnesses based on PIR. These methods have the potential to provide anonymous updates with constant computational and communication overhead, though they require a slight adjustment to the threat model.

Chapter 5

Credential Scheme with Privacy-Preserving Revocation

In this chapter, we propose a BBS credential scheme compatible with the Swiss e-ID setting that supports scalable privacy-preserving revocation with cryptographic accumulators. We open the chapter, by describing a protocol to bind membership proofs for the accumulator defined in Chapter 4 with BBS disclosure proofs (Section 5.1). Then, we make use of the binding protocol to present a BBS credential scheme with efficient accumulator-based revocation (Section 5.2). Finally, we analyze the security of the protocol that we introduced at the beginning of the chapter (Section 5.3).

5.1 A Protocol for Binding Membership Proofs and BBS Proofs

In this section, we present a protocol for binding membership proofs of the positive dynamic accumulator defined in Section 4.3 to BBS presentation proofs. We begin with a brief introduction to the setting (Section 5.1.1), followed by the description of the protocol (Section 5.1.2). The security analysis of the protocol is deferred to the end of the chapter (Section 5.3).

5.1.1 Setting

We consider the e-ID setting defined in Section 3.1 (Scenario B). The issuer's public parameters are composed of the accumulator parameters defined in our initial accumulator construction (Section 4.2), and some additional parameters for BBS verification.

The issuer generates the BBS parameters as follows:

- picks the BBS secret key $\hat{x} \leftarrow\!\!\$\, \mathbb{Z}_p$;
- publishes the public key $\hat{X} = \hat{x}G_2 \in \mathbb{G}_2$, together with a list of generators $(\mathrm{H}_1, \ldots, \mathrm{H}_l) \in \mathbb{G}_1^l$;

On issuance, a credential holder receives a BBS signature and a membership witness from the issuer:

- the signature certifies that the holder is associated to list of attributes $\mathbf{m} \leftarrow (\mathbf{m}_1, \ldots, \mathbf{m}_l) \in \mathbb{Z}_p^l$. A BBS signature is a pair $\hat{\sigma} = (\hat{A}, \hat{e}) \in \mathbb{G}_1 \times \mathbb{Z}_p$, where $\hat{A} = \frac{1}{\hat{x}+\hat{e}}C(\mathbf{m})$, and $C(\mathbf{m}) = G_1 \prod_{i=1}^{l} \mathbf{m}[i]\mathrm{H}_i$;
- the membership witness certifies that the holder's signature has not been revoked. We remind that a witness pair is defined as $(A, e) \in \mathbb{G}_1 \times \mathbb{Z}_p$, where $A = \frac{1}{(x+e)}V$, and V is the public accumulator value. The unique element e associated to a holder is *randomly* picked from the accumulator domain (i.e., $e \leftarrow\!\!\$\, \mathbb{F}_p \setminus \{-x\}$).

Finally, we assume that the issuer sets the k-th attribute in the BBS signature to the e element of the holder's witness, i.e., $\mathbf{m}[k] = e$.

Goal The holder can demonstrate validity of his BBS signature with respect to a partially disclosed message $\mathbf{m}' \subset \mathbf{m}$ by proving:

$$\tilde{e}(\hat{A}, \hat{X}) = \tilde{e}(C_J(\mathbf{m}) + (\sum_{i \in I} \mathbf{m}[i]\mathrm{H}_i) - \hat{e}\hat{A}, G_2)$$

where $I \subseteq [l]$ is the set of indexes of *non-disclosed* attributes, $J = [l] \setminus I$, and $C_J(\mathbf{m}) = G_1 \sum_{i \in J} \mathbf{m}[i]\mathrm{H}_i$.[1]

As we showed in Section 4.3, a holder can show non-revocation of his witness against the public accumulator by proving:

$$\tilde{e}(A, X) = \tilde{e}(V - eA, G_2).$$

However, *in addition* to prove the individual validity of the two signatures, the holder should also prove that the element e contained in k-th position of $\mathbf{m}$ is the *same* element used to produce the non-revocation proof. If the latter check is not enforced, a holder can use *any* valid witness to prove non-revocation of his credential, as we highlighted in Section 4.2.4 (L3).

5.1.2 Protocol Description

We move now to the description of the protocol. The algorithms $\mathbf{MemProof}.\mathbf{Comm}_{\mathcal{H}}$, and $\mathbf{MemProof}.\mathbf{Resp}_{\mathcal{H}}$, are defined as in Figure 4.3.

- *Proof initialization*:

[1]Note that $C_J(\mathbf{m}) + (\sum_{i \in I} \mathbf{m}[i]\mathrm{H}_i) = C(\mathbf{m})$

- the prover starts initializing the proof by computing the commitments for the membership proof:

$$\mathbf{MemProof}.\mathbf{Comm}_{\mathcal{H}}(V, A, e) \to (r, \alpha, \beta, \bar{A}, \bar{B}, U);$$

- then, the prover computes the commitments for the BBS proof: he picks random $r_1, r_2 \leftarrow\!\!\$\, \mathbb{Z}_p^*$, $\alpha', \beta', \gamma \leftarrow\!\!\$\, \mathbb{Z}_p$, and $\delta_i \leftarrow\!\!\$\, \mathbb{Z}_p$ for every $i \in I$. Then, he computes:

$$\bar{A}' \leftarrow r_1 {r_2}^{-1} \hat{A}, \qquad D \leftarrow {r_2}^{-1} C(\mathbf{m}), \qquad \bar{B}' \leftarrow r_1 D - \hat{e}\hat{A},$$

  and

$$U_1 = \alpha' D + \beta' \bar{A}', \qquad U_2 \leftarrow \gamma D + \sum_{i \in I} \delta_i \mathrm{H}_i.$$

- finally, the prover sends $(\bar{A}, \bar{A}', \bar{B}, \bar{B}', U, U_1, U_2, D)$ to the verifier;

- *Challenge*: the verifier picks a random challenge $c \leftarrow\!\!\$\, \mathbb{Z}_p$ and sends it back to the prover;

- *Response*: The prover computes responses for the memebership proof:

$$\mathbf{MemProof}.\mathbf{Resp}_{\mathcal{H}}(e, r, \alpha, \beta, c) \to (s, t),$$

  and for the BBS proof:

$$s' \leftarrow \alpha' + r_1 \cdot c, \quad t' \leftarrow \beta' - \hat{e} \cdot c,$$
$$z \leftarrow \gamma + r_2 \cdot c, \quad u_i \leftarrow \delta_i - \mathbf{m}[i] \cdot c \quad \forall i \in I.$$

  Then, he sends $(s, s', t, t', z, (u_i)_{i \in I})$ back to the verifier;

- *Verification*: the verifier only accepts if the BBS proof is verified:

$$\tilde{e}(\bar{A}', \hat{X}) = \tilde{e}(\bar{B}', G_2) \tag{1.1}$$
$$U_1 + c\bar{B}' = s'D + t'\bar{A}' \tag{1.2}$$
$$U_2 + cC_J(\mathbf{m}) = zD + \sum_{i \in I} u_i \mathrm{H}_i, \tag{1.3}$$

  the membership proof is verified:

$$\tilde{e}(\bar{A}, X) = \tilde{e}(\bar{B}, G_2), \tag{2.1}$$
$$U + c\bar{B} = sV + t\bar{A}, \tag{2.2}$$

  and the blinded forms of e, and $\mathbf{m}[k]$ have the same value:

$$t = u_k \tag{3.1}$$

The above Σ-protocol is almost an AND composition of the Σ protcol for BBS presentations defined in [TZ23, Appendix B] (which is adopted in the BBS standard draft [2]), and the one for membership proofs defined in Section 4.3. However, as the prover uses the same randomness for computing t, and u_k, and we claim that (3.1) ensures $\mathbf{m}[k] = e$, we need to prove that our protocol is secure. At the end of this chapter (Section 5.3), we prove that it is both *sound and ZK*.

5.2 Digital Credential Scheme with Accumulator-Based Revocation

In this section, we present our complete construction of a BBS credential scheme with accumulator-based revocation, which is well-suited to the Swiss e-ID setting. We start by presenting the full construction of our scheme (Section 5.2.1). Then, we make some remarks on adaptive soundness (Section 5.2.2), and we prove that our scheme defines a positive dynamic *adaptively sound* accumulator (Section 5.2.3). We conclude the section with some considerations on unlinkability (Section 5.2.4).

5.2.1 A BBS Credential Scheme with Accumulator Based Revocation

In Figure 5.1, we outline our BBS scheme with accumulator-based revocation. To keep the scheme concise, we avoided re-defining the accumulator's primitives, which were detailed in Chapter 4. Furthermore, to avoid additional notation overhead, we assume that the issuer keeps an offline mapping between each signed message $\mathbf{m}$, and the respective random element e that is stored in k-th position.

Issuance On issuance of a new BBS credential, the issuer has a message $\mathbf{m}$, which is the list of attributes describing the holder. We assume that the k-th position of $\mathbf{m}$ is empty.

First, the issuer picks a random element e and sets it in k-th position of $\mathbf{m}$ (Lines 1,2). Then, the issuer generates a BBS signature on $\mathbf{m}$ (Lines 3,4). Finally, the issuer runs $\mathbf{Add}_{\mathcal{I}}$ which issues a membership witness for e, and includes the element in the accumulated set (Line 5).

Revoke To revoke a list of BBS credentials associated to messages $\hat{D}_{t+1} \leftarrow (\mathbf{m}_1, \ldots, \mathbf{m}_m)$, the issuer creates a deletion list D_{t+1} containing the associated elements $(e_1, \ldots, e_m)$ (Line 2). Then, he executes $\mathbf{DelBatch}_{\mathcal{I}}$ to remove D_{t+1} from the accumulator, and publishes the resulting update polynomials Δ_{t+1}, and new accumulator value V_{t+1} on a public repository.

[2] https://identity.foundation/bbs-signature/draft-irtf-cfrg-bbs-signatures.html

```
$\mathbf{Gen}_{\mathcal{I}}(1^\lambda)$
1. $\mathbf{Setup}_{\mathcal{I}}(1^\lambda) \to (V_0, x, pp', S_0)$;
2. $\hat{x} \leftarrow \mathbb{F}_p$;
3. $\hat{X} \leftarrow \hat{x}G_2$;
4. $(\mathrm{H}_1, \ldots, \mathrm{H}_l) \leftarrow \$ \mathbb{G}_1^l$
5. $pp \leftarrow (pp', \hat{X}, \mathrm{H}_1, \ldots, \mathrm{H}_l)$;
6. publish $pp$;
7. return $(\hat{x}, x, V_0, S_0)$.
```

```
$\mathbf{Issue}_{\mathcal{I}}(x, \hat{x}, V_t, \mathbf{m})$
1. $e \leftarrow \$ \mathbb{F}_p$;
2. $\mathbf{m}[k] \leftarrow e$;
3. $C(\mathbf{m}) \leftarrow G_1 \prod_{i=1}^{l} \mathbf{m}[i]\mathrm{H}_i$;
4. $\hat{A} \leftarrow \frac{1}{\hat{x}+\hat{e}} C(\mathbf{m})$;
5. $\mathbf{Add}_{\mathcal{I}}(x, V_t, e, S_k) \to (A_t, S_{k+1})$;
6. $\sigma_t \leftarrow (A_t, e)$;
7. $\hat{\sigma} \leftarrow (\hat{A}, \hat{e})$;
8. return $(\sigma_t, \hat{\sigma}, S_{k+1})$.
```

```
$\mathbf{Revoke}_{\mathcal{I}}(x, V_t, \hat{D}_{t+1})$
1. $(\mathbf{m}_1, \ldots, \mathbf{m}_m) \leftarrow \hat{D}_{t+1}$;
2. $D_{t+1} \leftarrow (\mathbf{m}_i[k])_{i \in [m]}$
3. $\mathbf{DelBatch}_{\mathcal{I}}(x, V_t, D_{t+1}, S_k) \to ret$;
4. $(V_{t+1}, \Delta_{t+1}, S_{k+1}) \leftarrow ret$;
5. publish $\Delta_{t+1}, V_{t+1}$;
6. return $S_{k+1}$.
```

```
$\mathbf{Update}_{\mathcal{H}}(V_t, \sigma_t, \Delta_{t+1}^{t+n})$
1. $(A_t, e) \leftarrow \sigma_t$;
2. $\mathbf{WitUpBatch}_{\mathcal{H}}(V_t, e, A_t, \Delta_{t+1}^{t+n}) \to A_{t+n}$
3. $\sigma_{t+n} \leftarrow (A_{t+n}, e)$;
4. return $\sigma_{t+n}$.
```

```
$\mathbf{Proof.Final}_{\mathcal{H}}(e, \hat{e}, c, open, \mathbf{m}, I)$
1. $(r, r_1, r_2, \alpha, \alpha', \beta, \beta', (\sigma_i)_{i \in I}) \leftarrow open$;
2. $\mathbf{MemProof.Resp}_{\mathcal{H}}(e, r, \alpha, \beta, c) \to (s, t)$
3. $s' \leftarrow \alpha' + r_1 \cdot c$;
4. $t' \leftarrow \beta' - \hat{e} \cdot c$;
5. $z \leftarrow \gamma + r_2 \cdot c$;
6. for $i \in I$:
7.     $u_i \leftarrow \delta_i - \mathbf{m}[i]c$;
8. $resp \leftarrow (s, s', t, t', z, (u_i)_{i \in I})$;
9. return $resp$.
```

```
$\mathbf{Proof}_{\mathcal{H}}(V_t, \sigma_t, \hat{\sigma}, \mathbf{m}, I)$
1. $(A_t, e) \leftarrow \sigma_t$;
2. $\mathbf{Proof.Init}_{\mathcal{H}}(V_t, \sigma_t, \hat{\sigma}, m, I) \to ret$;
3. $(commits, open) \leftarrow ret$;
4. send $commits$ to verifier;
5. receive $c$ from verifier;
6. $\mathbf{Proof.Final}_{\mathcal{H}}(e, \hat{e}, c, open, \mathbf{m}, I) \to resp$
7. send $resp$ to verifier;
```

```
$\mathbf{Verify}_{\mathcal{V}}(V_t, \mathbf{m}')$
1. receive $commits$ from holder;
2. $c \leftarrow \$ \mathbb{Z}_p$;
3. send $c$ to prover;
4. receive $resp$ from prover;
5. $commits \leftarrow (\bar{A}, \bar{A}', \bar{B}, \bar{B}', U, U_1, U_2, D)$;
6. $resp \leftarrow (s, s', t, t', z, (u_i)_{i \in I})$
7. $b_1 \leftarrow \tilde{e}(\bar{A}, X) \overset{?}{=} \tilde{e}(\bar{B}, G_2)$;
8. $b_2 \leftarrow U + c\bar{B} \overset{?}{=} sC + t\bar{A}$;
9. $b_3 \leftarrow \tilde{e}(\bar{A}', \hat{X}) \overset{?}{=} \tilde{e}(\bar{B}', G_2)$;
10. $b_4 \leftarrow U_1 + c\bar{B}' \overset{?}{=} s'D + t'\bar{A}'$;
11. $b_5 \leftarrow U_2 + c\, C_J(\mathbf{m}) \overset{?}{=} zD + \sum_{i \in I} u_i \mathrm{H}_i$;
12. $b_6 \leftarrow t \overset{?}{=} u_k$
13. $b \leftarrow \wedge_{i \in [6]} b_i$
14. return $b$;
```

```
$\mathbf{Proof.Init}_{\mathcal{H}}(V_t, \sigma_t, \hat{\sigma}, \mathbf{m}, I)$
1. $(A_t, e) \leftarrow \sigma_t$;
2. $(\hat{A}, \hat{e}) \leftarrow \hat{\sigma}$;
3. $\mathbf{MemProof.Comm}_{\mathcal{H}}(V_t, A_t, e) \to ret$;
4. $(r, \alpha, \beta, \bar{A}, \bar{B}, U) \leftarrow ret$;
5. $r_1, r_2 \leftarrow \$ \mathbb{Z}_p^*$;
6. $\alpha', \beta', \gamma \leftarrow \$ \mathbb{Z}_p$;
7. for $i \in I$:
8.     $\delta_i \leftarrow \$ \mathbb{Z}_p$;
9. $\bar{A}' \leftarrow r_1 {r_2}^{-1} \hat{A}$;
10. $D \leftarrow {r_2}^{-1} C(\mathbf{m})$;
11. $\bar{B}' \leftarrow r_1 D - \hat{e}$;
12. $U_1 \leftarrow \alpha' D + \beta' \bar{A}'$;
13. $U_2 \leftarrow \gamma D + \sum_{i \in I} \delta_i \mathrm{H}_i$;
14. $commits \leftarrow (\bar{A}, \bar{A}', \bar{B}, \bar{B}', U, U_1, U_2, D)$;
15. $open \leftarrow (r, r_1, r_2, \alpha, \alpha', \beta, \beta', (\sigma_i)_{i \in I})$;
16. return $(commits, open)$.
```

Figure 5.1: Complete BBS Credential Scheme with Accumulator-Based Revocation

Update We assume that before update his membership witness, the has fetched the list of update polynomials Δ_{t+1}^{t+n} from the public repository. Then, the holder executes $\mathbf{WitUpBatch}_{\mathcal{H}}$, which directly returns the updated witness A_{t+n} (Line 2).

Proofs The proofs (i.e., $\mathbf{Proof}_{\mathcal{H}}$), and verification (i.e., $\mathbf{Verify}_{\mathcal{V}}$) algorithms follow the binding protocol defined in Section 5.1.2. We have split $\mathbf{Proof}_{\mathcal{H}}$ into $\mathbf{Proof}.\mathbf{Init}_{\mathcal{H}}$, and $\mathbf{Proof}.\mathbf{Final}_{\mathcal{H}}$ for better readability.

Note that the additional overhead of the binding protocol, compared to executing a single BBS proof, is limited to the overhead introduced by the Σ-protocol for membership verification, which we analyzed in Section 4.3.3.

5.2.2 Modular Construction for Adaptively-Sound Accumulators

Soundness is a crucial security property for cryptographic accumulators, as it makes unfeasible for attackers to *forge* membership proofs. As membership proofs are used to prove non-revocation, a sound accumulator ensures that revoked holders cannot cheat on their status. We remind the reader that soundness is defined in two distinct forms, offering different levels of security. *Non-adaptive* soundness (Definition 2.2.2), requries the attacker to commit to a list of elements to be added/removed in advance. A non-adaptively sound accumulator remains secure against attackers who cannot modify their strategy after observing the accumulator's value and update messages. On the other hand, *adaptively sound* accumulators (Definition 2.2.3) are secure against adversaries who can modify their strategy based on what they *observe*.

As noted in [Bal+17], non-adaptively sound accumulators can be safely used to accumulate *randomly* selected elements, as each element's choice is independent of the others. Building on this observation, the authors define a modular construction for an *adaptively sound dynamic* accumulator ACC. Their construction integrates an *adaptively* sound positive accumulator ACC_A, and a *non-adaptively* sound positive *dynamic* accumulator ACC_{NA}, as follows:

- to *add* an element x into ACC, a random element r is added into ACC_{NA}, and (r, x) is added into ACC_A;
- to *delete* the element x, the random element r is removed from ACC_{NA};
- to *prove* $x \in ACC$, the prover needs to show:

$$r \in ACC_{NA} \wedge (x, r) \in ACC_A$$

.

This modular construction has been used in [Bal+17; KB21]. In the next section, we apply the same approach to prove that our BBS scheme defines a positive dynamic adaptively sound accumulators.

5.2.3 On the Adaptive Soundness of our Scheme

In the e-ID setting, holders are already associated with a positive *adaptively sound* accumulator ACC_A, which is the BBS signature. More precisely, the issuer's *public key* corresponds to the *accumulator* value, and the BBS *signature* is the membership *witness*. In fact, a BBS signature certifies that a list of attributes $\mathbf{m}$ (i.e., a message) describing a holder is *contained* in the accumulator. Clearly, the issuer can add holders to the accumulator, by signing other messages with his private key. However, as the issuer's public key is static, accumulated messages cannot be deleted, i.e., holders cannot be *revoked*.

To support deletions, in Chapter 4 we designed a additional positive *dynamic* accumulator. Our initial construction (Section 4.2), adopts the *non-adaptively sound* accumulator presented in [KB21, Section IV]. The extended version which supports batch operations *preserves* non-adaptive soundness, as we proved in Section 4.4.6.

In the previous section, we presented a construction that integrates the two accumulators (Section 5.2.1). After executing $\mathbf{Issue}_{\mathcal{I}}$, a new holder is provided with a membership witness-element pair $\sigma \leftarrow (A, e)$, where e is chosen at random. The element e is also added in the attribute list $\mathbf{m}'$ describing the holder, producing the message $\mathbf{m} \leftarrow \mathbf{m}' \cup \{e\}$ which is signed with BBS. The resulting signature $\hat{\sigma}$ is also given to the holder. Finally, we have:

- an efficient accumulator ACC_{NA}, containing the random element e;
- a BBS public key ACC_A containing $(e, \mathbf{m}')$;
- a Σ-protocol for presentations which succeeds only if both $e \in ACC_{NA} \wedge (e, \mathbf{m}') \in ACC_A$ (as we prove in Section 5.3);

Hence, our final construction defines a positive *dynamic adaptively sound* accumulator based on BBS signatures.

5.2.4 On Holders Unlinkability

The scheme we presented in Figure 5.1, is designed to support unlinkable presentations while also providing an anonymous and efficient revocation method. As we show in Section 5.2.3, the protocol that binds BBS disclosure proofs and non-revocation proofs is zero-knowledge, therefore a verifier cannot link subsequent presentation from the same holder.

Now, let us consider the *issuer*'s perspective. When a holder updates his credential, he requests updates starting from the last accumulator version for which he had a valid witness. Let V_t be the last accumulator for which the holder has a valid witness, and V_{t+n} be the current accumulator, the holder's update request *leaks* the update interval $[t, t+n]$. By keeping the history of all update intervals, a issuer can correlate subsequent update requests, similarly as in [JLM22, Appendix A.3].

Note that, all intervals with the same ending point, have the same probability of being the starting point of the next update request (i.e., they yield an anonymity set). Hence, as the number of holders increases, the attack becomes more complicated, since many updates will share the same ending points. We underline that, with revocation epochs, the time window for the attack can be greatly reduced: at the end of each epoch, *every* holder will get an up-to-date witness, and the previous update history becomes useless. Furthermore, as we provide an efficient method for aggregating multiple batch update polynomials (Section 4.4.4), fuzzing techniques can also be effective. For example, a holder who needs updates for the interval $[t, t+n]$ can split requests into multiple sub-intervals: he can request update polys $(\Delta_{t-2} ... \Delta_{t+k})$, $(\Delta_{t+k-4} ... \Delta_{t+n})$, and merge them into $\Delta_{t+1 \to t+n}$ (discarding the useless ones).

As a final remark, throughout this thesis, we assume that holders communicate using an *anonymous communication* system (e.g., TOR) or a trusted VPN provider. Without such an additional obfuscation layer, holders could be trivially linked, for example, by tracking their IP addresses.

5.3 Security Analysis of the Binding Protocol

In this section, we prove that the protocol described in Section 5.1.2 is both sound and $(\mathcal{L})$HVZK. Our work is based on the on the security proofs of the individual BBS Σ-protocols presented in [TZ23].

5.3.1 Soundness

We start by proving 2-*special soundness*, by analyzing a 2-tree of accepting transcripts.

Assume a prover executes *Proof initialization*, which returns the BBS commitments $\bar{A}', \bar{B}', D, U_1, U_2$, and the membership proof commitments $\bar{A}, \bar{B}, U$.

Given a 2-tree of accepting transcripts with challenges $c_1, c_2 \in \mathbb{Z}_p$ (with $c_1 \neq c_2$), we have:

- BBS proof parameters $(s'_1, s'_2, t'_1, t'_2, z_1, z_2, (u_{i,1})_{i \in I}, (u_{i,2})_{i \in I})$, s.t. for

$j \in \{1, 2\}$:

$$\tilde{e}(\bar{A}', \hat{X}) = \tilde{e}(\bar{B}', G_2), \tag{4.1}$$

$$U_1 + c_j \bar{B}' = s'_j D + t'_j \bar{A}', \tag{4.2}$$

$$U_2 + c_j C_J(\mathbf{m}) = z_j D + \sum_{i \in I} u_{i,j} \mathrm{H}_i. \tag{4.3}$$

- membership proof parameters (s_1, s_2, t_1, t_2) such that for $j \in \{1, 2\}$:

$$\tilde{e}(\bar{A}, X) = \tilde{e}(\bar{B}, G_2) \tag{5.1}$$

$$U + c_j \bar{B} = s_j V + t_j \bar{A} \tag{5.2}$$

$$t_j = u_{k,j} \tag{5.3}$$

We start by showing that we can extract a valid BBS proof on a message $\mathbf{m}$, such that $\mathbf{m'} \subset \mathbf{m}$, and $\mathbf{m}[k] = \mu_k$.

By (4.2), (4.3) it holds:

$$(c_1 - c_2)\ \bar{B}' = (s'_1 - s'_2)\ D + (t'_1 - t'_2)\ \bar{A}', \tag{6.1}$$

$$(c_1 - c_2)\ C_J(\mathbf{m}) = (z_1 - z_2)\ D + \sum_{i \in I} (u_{i,1} - u_{i,2})\ \mathrm{H}_i. \tag{6.2}$$

Then, we extract $r_1 = \frac{(s'_1 - s'_2)}{(c_1 - c_2)}$, $r_2 = \frac{(z_1 - z_2)}{(c_1 - c_2)}$, $\hat{e} = \frac{(t'_2 - t'_1)}{(c_1 - c_2)}$, and $\mu_i = \frac{(u_{i,2} - u_{i,1})}{(c_1 - c_2)}$ for $i \in I$.

Now, we can rewrite (6.1), (6.2) using the extracted values:

$$\bar{B}' = r_1 D - \hat{e} \bar{A}', \tag{7.1}$$

$$C_J(\mathbf{m}) = r_2 D + \sum_{i \in I} -\mu_i \mathrm{H}_i. \tag{7.2}$$

Depending on the values of r_1, r_2, we have three different cases:

- $\mathbf{r_1 = 0}$: from (7.1), we have $\bar{B}' = -\hat{e}\bar{A}'$. Furthermore, by (4.1) it holds that $\tilde{e}(\bar{A}', \hat{X}) = \tilde{e}(-\hat{e}\bar{A}', G_2)$. By bilinearity, $\hat{x} = -\hat{e}$ is the secret key associated with $\hat{X}$. Our extractor can use $\hat{x}$ to output a valid BBS signature on $\mathbf{m}$ such that $\mathbf{m'} \subset \mathbf{m}$, and $\mathbf{m}[i] = \mu_i$ for all $i \in I$;
- $\mathbf{r_1 \neq 0 \wedge r_2 = 0}$: by (7.2) it holds:

$$\begin{aligned} 0_{\mathbb{G}_1} &= C_J(\mathbf{m}) + \sum_{i \in I} \mu_i \mathrm{H}_i \\ &= G_1 + \sum_{i \in J} \mathbf{m}[i] \mathrm{H}_i + \sum_{i \in I} \mu_i \mathrm{H}_i. \end{aligned}$$

Setting $\mathbf{m}[i] = \mu_i$ for all $i \in I$, we can rewrite the equation above as:

$$0_{\mathbb{G}_1} = C(\mathbf{m})$$

For $\hat{A} = 0_{\mathbb{G}_1}$, we have $\tilde{e}(\hat{A}, \hat{X}) = \tilde{e}(C(\mathbf{m}) - \hat{e}\hat{A}, G_2)$. Hence, our extractor can produce a valid BBS signature $(0_{\mathbb{G}}, \hat{e})$ for $\mathbf{m}$ such that $\mathbf{m'} \subseteq \mathbf{m}$ and $\mathbf{m}[i] = \mu_i$ for all $i \in I$.

- $\mathbf{r_1 \neq 0 \wedge r_2 \neq 0}$: set $\hat{A} = r_2 {r_1}^{-1} \bar{A}'$. If we multiply both sides of (4.1) by $r_2 {r_1}^{-1}$, we obtain:

$$\tilde{e}(\hat{A}, \hat{X}) = \tilde{e}(r_2 {r_1}^{-1} \bar{B}', G_2).$$

From (7.1), (7.2) we have:

$$\begin{aligned} r_2 {r_1}^{-1} \bar{B}' &= r_2 D - \hat{e}\hat{A} \\ &= C_J(\mathbf{m}) + \sum_{i \in I} \mu_i \mathrm{H}_i - \hat{e}\hat{A}. \end{aligned}$$

Setting $\mathbf{m}[i] = \mu_i$ for all $i \in I$, we can rewrite the equation above as:

$$r_2 {r_1}^{-1} \bar{B}' = C(\mathbf{m}) - \hat{e}\hat{A}.$$

As $\tilde{e}(\hat{A}, \hat{X}) = \tilde{e}(C(\mathbf{m}) - \hat{e}\hat{A}, G_2)$, our extractor can produce a valid BBS signature $(\hat{A}, \hat{e})$ on a message $\mathbf{m}$ such that $\mathbf{m'} \subseteq \mathbf{m}$ and $\mathbf{m}[i] = \mu_i$ for all $i \in I$.

Now, let us show that our extractor can also compute a valid *membership witness*. Similarly as before, from (5.2) we obtain:

$$(c_1 - c_2)\bar{B} = (s_1 - s_2)V + (t_1 - t_2)\bar{A}.$$

Setting $r = (s_1 - s_2)/(c_1 - c_2)$, and $e = (t_2 - t_1)/(c_1 - c_2)$, we rewrite the above equation as:

$$\bar{B} = rV - e\bar{A}. \tag{8.1}$$

Depending on the value of r we have two cases:

- $\mathbf{r \neq 0}$: call $A = r^{-1}\bar{A}$. By (8.1) we have:

$$r^{-1}\bar{B} = V - eA,$$

and by (5.1):

$$\tilde{e}(A, X) = \tilde{e}(V - eA, G_2).$$

Therefore, (A, e) is a valid membership witness-element pair for the public accumulator V.

- $\mathbf{r} = \mathbf{0}$: by (8.1), $\bar{B} = -e\bar{A}$, therefore $\tilde{e}(A, X) = \tilde{e}(-e\bar{A}, G_2)$. This implies that $x = -e$ is the secret key associated with X and our extractor can use it to output a valid membership witness for V.

Finally, we demonstrate the binding between the two extracted proofs (i.e., $e = \mathbf{m}[k]$). By (5.3), we can rewrite e substituting t_j with $u_{k,j}$, which gives: $e = \frac{u_{k,2} - u_{k,1}}{c_1 - c_2} = \mu_k$. As we showed before, our extractor can always compute a valid BBS signature $\sigma = (\hat{A}, \hat{e})$ on a message $\mathbf{m}$ s.t. $\mathbf{m}[k] = \mu_k$. Therefore, the BBS signature σ, and the membership witness $\sigma = (A, e)$ are linked.

5.3.2 Zero-Knowledge

We prove that the protocol is $\mathcal{L}$-HVZK, for $\mathcal{L}$ which on inputs $(G_1, x, \hat{x})$ outputs a tuple $(P, xP, \hat{x}P)$. As noted in [TZ23], the oracle $\mathcal{L}$ does not leak anything more than any valid message-signature pair. In fact, given (A, e), $(\hat{A}, \hat{e})$ and the respective $\mathbf{m}$, and one can compute:

$$xA = V - eA.$$

and:

$$\hat{x}\hat{A} = C(\mathbf{m}) - \hat{e}\hat{A},$$

Our simulator S is given an instance $\mathbb{X} = (\mathbf{m}', V)$ and a tuple pair $(P, \hat{x}P, xP)$. The simulator starts by generating parameters for the *membership proof*:

- picks $r \leftarrow_\$ \mathbb{Z}_p$ and computes $\bar{A} = rP$ and $\bar{B} = r(xP)$;
- picks random $s \leftarrow_\$ \mathbb{Z}_p$, and sets $t = u_k$;
- finally, it sets $U = sV + t\bar{A} - c\bar{B}$.

The points $\bar{A}, \bar{B}$, and the scalars s, c are distributed uniformly at random as by an honest execution of our protocol. Furthermore, t is uniquely determined by u_k (3.1), and U is uniquely determined by $(\bar{A}, \bar{B}, V, s, t)$ (2.1). By construction, it is easy to check that the simulated values satisfy the verification equations (2.1), (2.2), (3.1).

Then, our simulator generates parameters for the *BBS proof* on $\mathbf{m}'$:

- picks $\hat{r} \leftarrow_\$ \mathbb{Z}_p^*$, and computes $\bar{A}' = \hat{r}U$ and $\bar{B}' = \hat{r}(\hat{x}U)$;
- picks a random challenge $c \leftarrow_\$ \mathbb{Z}_p$;
- picks random $s', t', z, (u_i)_{i \in I} \leftarrow_\$ \mathbb{Z}_p$, and $D \leftarrow_\$ \mathbb{G}_1$;
- defines $\mathbf{m} \in \mathbb{Z}_p^l$ s.t.: $\mathbf{m}[i] = u_i$ for all $i \in I$, $\mathbf{m}[i] = \mathbf{m}'[j]$ for $j \in [|J|]$, and $i \in J$ indicating the index associated to $\mathbf{m}'[j]$ within the set of disclosed indexes;

- computes $U_1 = s'D + t'\bar{A}' + -c\bar{B}'$, and $U_2 = zD + \sum_{i\in I} u_i \mathrm{H}_i - c\, C_J(\mathbf{m})$.

As by an honest execution, the points $\bar{A}', \bar{B}', D$ and the scalars $s', t', z, (u_i)_{i\in I}$, are uniformly distributed. Furthermore, as by (1.2) and (1.3), U_1 and U_2 are uniquely determined by the above values and the disclosed message $\mathbf{m}'$. Since we also have $\bar{B}' = \hat{x}\bar{A}'$, it is easy to see that all the BBS verification equations (i.e., (4.1), (4.2), (4.3)) hold for the simulated values.

We have proven that the verifier's view generated by our simulator is indistinguishable from the view obtained after an honest execution of the protocol.

Chapter 6

Private Information Retrieval for Updating Witnesses

In this chapter, we explore the PIR problem and assess whether PIR schemes are well-suited for providing alternative witness update methods. We begin by introducing the PIR problem and explaining its relevance to *anonymous witness updates* (Section 6.1). Next, we examine some state-of-the-art single-server PIR schemes and estimate their performance in our setting (Section 6.2). Finally, we consider two-server PIR schemes and evaluate their applicability through a proof-of-concept implementation (Section 6.3).

6.1 Introduction

The PIR problem involves two entities: a *client* and a *server*. The server maintains a set of *public* elements in a database $D = \{d_1, \dots, d_n\}$. The client is interested in a specific element d_i but wishes to keep his interest private from the server. A PIR protocol allows the client to retrieve the desired element d_i without revealing any information about the queried element to the server. It is important to note that PIR schemes do not necessarily protect the privacy of the server's entire database; a client could potentially learn more than just the queried entry.

The witness update problem (Section 4.4.1), is a specific instance of PIR: the issuer maintains a server with a list of *up-to-date* witnesses $D = \{A_1, \dots, A_N\}$. A credential holder needs to fetch his witness, say A_i, without revealing it to the issuer. In our threat model, the list of witnesses is not considered private: even if someone fetches a holder's witness A_i, he cannot use it to produce a valid membership proof, as he misses the holder's associated element e_i.

The accumulator update strategy presented in Section 4.4, achieved efficient and privacy-preserving updates but it suffered from linear communication complexity. In this chapter, we explore alternative update methods based on PIR, with the goal of improving the communication complexity. We remark that secure PIR schemes inherently eliminate *any* possible leakage regarding the holder's query.

6.2 Single-Server PIR Schemes

Let us first consider the setting where the database D is maintained by a *single* server. A naive way to implement PIR would be downloading the entire issuer's database. Obviously, this preserves the user's privacy, but requires $\Omega(N)$ communication, where N is the size of the database. To be considered efficient, a PIR scheme must improve the communication complexity of the naive solution.

PIR from homomorphic encryption Many single-server PIR schemes are based on homomorphic encryption. The general idea is to represent a database D of size N as a $\sqrt{N}$-by-$\sqrt{N}$ matrix. A client interested in the element in position (i, j), defines a query vector $\vec{q}$ of $\sqrt{N}$ elements. The vector $\vec{q} = (0, \dots 0, 1, 0, \dots, 0)$ is filled with 0s, except for the entry in position j, which is set to 1. Next, the client computes the encrypted query $\vec{q}_{enc} = Enc(\vec{q})$, where Enc is an homomorphic encryption scheme, and sends $\vec{q}_{enc}$ to the server. The server computes the answer $\vec{a}_{enc} = D \cdot \vec{q}_{enc}$ and returns it to the client. Due to the homomorphic properties of the encryption scheme we have: $\vec{a}_{enc} = D \cdot \vec{q}_{enc} = Enc(D \cdot \vec{q})$. Hence, decrypting the answer $\vec{a} \leftarrow Enc^{-1}(\vec{a}_{enc})$ yields the j-th column of the database. Although this approach reduces the communication costs of the naive solution by only exchanging $2\sqrt{N}$ ciphertext elements, it requires the server to compute N ciphertext additions and multiplications *for each query*.

In this thesis, we considered two recent state of the art single PIR approaches [Hen+23; DPC23], based on the Learning with Errors (LWE) assumption, and the additively-homomorphic Regev encryption scheme [Reg09]. Both approaches reduce the server computation by requiring each client to download a 'hint' about the database content. All subsequent client queries utilize this same hint, enabling the server to respond more efficiently, after some initial pre-computation.

Simple PIR Simple PIR ([Hen+23]) achieves high server throughput, reducing by 99% the number of required ciphertext operations for answering queries. However, each client must download a hint of size $n\sqrt{dN}\log q$, where N is the dimension of the database, $n = 1024$ is the lattice security parameter, $q = 2^{32}$ is the ciphertext modulus for the Regev encryption, and d is the number of plaintext database entries required to represent one element. According to their simulator[1], using the suggested security parameters and considering a database of $N = 2^{20}$ witnesses, each 48B in size, Simple PIR would require downloading a hint of approximately 25MB. For comparison, downloading the *entire* database would require around 50MB. For *static* databases, clients can amortize the cost of the one-time download by reusing their hint across multiple queries. Unfortunately, in our setting, new revocation events invalidate all previous witnesses, and every entry in the database must be updated. As a result, holders would need to download a new hint for *every* different update, making the approach infeasible.

FrodoPIR With SimplePIR, representing a single witnesses requires many database entries, which implies having high d values (i.e., in their simulation $d = 39$). As the hint size is $n\sqrt{dN}\log q$ bits, d values are a multiplicative factor of the database size N, and have considerable impact in the communication complexity.

The approach of FrodoPIR [DPC23] is very similar to SimplePIR, but it uses smaller hints of size $\approx nd \log q$ (i.e., independent of N). Smaller hints comes at the cost of having higher per-query communication of $\approx N \log q$. In our use case, this translates to hints of $\approx$ 221.7KB and online communication of $\approx$ 4.2MB. With additional optimizations (i.e., the sharding techniques discussed in Section 6.1), we could achieve a total communication cost of 1.8 MB. Note that this is a 14$\times$ improvement with respect to Simple PIR. However, our original accumulator's update strategy, required downloading 80B for each revocation that happened since the lat update. Therefore, in terms of holder communication, FrodoPIR only starts to outperform the default strategy after $\approx$ 22.000 revocations.

In conclusion, single server PIR schemes do not appear to offer sufficient advantages to justify the added complexity of integrating them into our accumulator scheme. In the next section, we explore two-server PIR schemes that have extremely low communication complexity, though they operate under different threat assumptions.

6.3 2-Server PIR for Efficient Witness Update

6.3.1 Distributed Point Functions

A *point function* is a function that evaluates to 0 for every element in its domain, except for a specific element where it returns a designated value. More formally, given some $x, y \in \{0,1\}^*$, a point function $F_{x,y}$ is defined as:

$$\begin{cases} F_{x,y}(x') = 0^{|y|} & \forall\ x' \in \{0,1\}^*\ s.t.\ x' \neq x, \\ F_{x,y}(x) = y. \end{cases}$$

[1]https://github.com/ahenzinger/simplepir/tree/438b4590aceedf76c7588b03125dfc0db39e361f

The notion of *Distributed Point Functions (DPFs)* was introduced in [GI14]. A DPF can be seen as splitting a point function $F_{x,y}$, into a pair[2] of *additive shares* k_0, k_1, such that each share k_i individually hides x, y. The shares k_0, k_1 are also called *keys*.

More precisely, a DPF for a point function $F_{x,y}$, is defined as a pair of PPT algorithms (*Gen*, *Eval*):

- the *generation* algorithm *Gen* is such that:

$$Gen(x, y) \to (k_1, k_2) \qquad \forall\, x, y \in \{0,1\}^*$$

- the *evaluation* algorithm *Eval*, on input two keys k_1, k_2, is such that:

$$\begin{cases} Eval(k_0, x') \oplus Eval(k_1, x') = 0^{|y|} & \forall\, x' \in \{0,1\}^* \ s.t.\ x' \neq x, \\ Eval(k_0, x) \oplus Eval(k_1, x) = y. \end{cases} \tag{6.1}$$

A DPF must satisfy *correctness* and *secrecy*. Correctness ensures that *Eval* on input an element $x' \in \{0,1\}^*$, and two honestly-generated keys for a point function $F_{x,y}$, returns additive shares of $F_{x,y}(x')$. Secrecy ensures that each key does not reveal anything about x, y (except for the input and output domains). In the following, we report the formal definitions of completeness and secrecy given in [GI14].

Completeness: for all x, x' s.t. $|x| = |x'|$:

$$\Pr[(k_0, k_1) \leftarrow Gen(x, y) : Eval(k_0, x') \oplus Eval(k_1, x') = F_{x,y}(x')] = 1.$$

Secrecy: for $x, y \in \{0,1\}^*$, and $b \in \{0,1\}$, let $D_{b,x,y}$ be the probability distribution of k_b induced by $Gen(x, y)$. Then, there exists a PPT algorithm *Sim* such that the following distributions are *indistinguishable*:

- $\{Sim(b, |x|, |y|)\}$
- $\{D_{b,x,y}\}_{b \in \{0,1\}, x,y \in \{0,1\}^*}$

As noted by Gilboa and Ishai [GI14], a trivial DPF can be implemented by letting k_0, k_1 be random vectors in $(\mathbb{F}_{2^{|y|}})^{2^{|x|}}$, such that $k_0[x'] + k_1[x'] = 0$ for all $x' \in \{0,1\}^{|x|}$ *s.t.* $x' \neq x$, and $k_0[x] + k_1[x] = y$. This DPF is perfectly hiding, as both keys are completely random. However, the key size is *exponential* in x, y.

The authors address this problem by recursively replacing parts of the

[2]In this thesis, we consider the case where the number of derived shares is 2, but in general we can have more shares.

keys k_1, k_2 with appositely selected seeds of a pseudo-random function. The recursive evaluation algorithm *Eval* uses a pseudo-random generator to expand each seed and compute the answer. The seeds for the two keys are chosen such that *Eval* behaves exactly as a DPF (i.e., as by eq. (6.1)). The author's solution compresses the key size up to $4n(\lambda - 1)$ bits, where λ is the security parameter, and $N = 2^n$ is the number of users. This compression comes at the cost of making the scheme *computationally* hiding, meaning that the derived keys are only computationally indistinguishable from random strings. For more details we refer to the original paper.

In [BGI16], the authors additionally shrink the key size by a factor of 4. They also design an optimized algorithm for *full-domain evaluation* (i.e., computing the DPF over all possible input elements), which is useful for PIR applications, as we explain the next section. The *Eval* function traverses a binary tree represented by the input key, each input element x' determines a different root-to-leaf path according to its binary representation. To compute the evaluation correctly, at each intermediate node a *correction word* is applied by one of the two server. To correctly synchronize this process, the authors add a single bit $b \in \{0, 1\}$ as additional argument to the input of *Eval*. The clients sets $b = 0$ when he queries the first sever, and $b = 1$ when querying the second.

On an database of $N = 2^n$ one bit entries and a security parameter λ, the authors achieve the following upper bounds:

- **key size**: $\lambda(n - \log \lambda)$ bits;
- **key generation**: $2(n - \log \lambda)$ AES operations;
- **single root-leaf evaluation**: $n - \log \lambda$ AES operations,
- **full-domain evaluation**: $N \log \lambda$ AES operations,

where AES is the pseudo-random generator used for expanding the key. We underline that the key size is *logarithmic* in the number of users.

In the next section, we explain how DPFs can be used in a 2-server PIR scheme.

6.3.2 2-Server PIR from DPFs

DPFs can naturally be used as a building-block for multi-server PIR schemes. Let us consider a 2-server PIR setting, where each server has a *binary* database $D = \{d_1, \ldots, d_N\}$, with each $d_i \in \mathbb{F}_2$. A client who is interested in the i-th bit, generates a DPF for the point function $F_{i,1}$ using the construction in [BGI16].

The generation algorithm *Gen* outputs two shares k_0, k_1 such that:

$$\begin{cases} Eval(0, k_0, j) \oplus Eval(1, k_1, j) = 0 & \forall j \neq i \\ Eval(0, k_0, i) \oplus Eval(1, k_1, i) = 1. \end{cases} \tag{6.2}$$

The client queries each of the two servers, with a different pair (b, k_b). Then, each server computes the answer as follows:

$$a_b = \bigoplus_{j=1}^{N} Eval(b, k_b, j) \cdot d_j,$$

and sends $a_b \in \mathbb{F}_2$ back to the client.

Thanks to the properties of DPFs, the client can reconstruct the target entry by simply computing $a_1 \oplus a_2$. In fact, from eq. (6.2) it follows:

$$\begin{aligned} a_1 \oplus a_2 &= \bigoplus_{j=1}^{N} (Eval(0, k_0, j) \cdot d_j) \oplus (Eval(1, k_1, j) \cdot d_j) \\ &= \bigoplus_{j=1}^{N} d_j \cdot (Eval(0, k_0, j) \oplus (Eval(1, k_1, j)) \\ &= d_i \end{aligned} \tag{6.3}$$

Due to DPF secrecy, the client's query leaks no information on the *specific position* of the target bit. Such assumption holds as long as the two server do not collude.

The costs for running the above PIR scheme are:

- **client:** send two DPF keys k_1, k_2, and download the two answers $a_1, a_2 \in \mathbb{F}_2$. The result is reconstud with a *single* xor operation;

- **server:** download one DPF key k_i, and send the answer a_i. To compute the answer the server performs a *full-domain evaluation* (i.e., computes *Eval* for all inputs $j \in [1, N]$), an *inner product* between the results of the evaluation and the respective database element, and a final xor between all results;

6.3.3 Using 2-Server PIR for Anonymous Witness Update

In this section, we apply the 2-server PIR scheme to achieve anonymous witness updates for the accumulator defined in Chapter 4.

We assume to have $N = 2^n$ credential holders, and 2 servers maintaining a database of witnesses $D = \{A_1, \ldots, A_n\}$, where we denote as A_i the witness associated with the i-th holder.

Remember that each witness is an element of $\mathbb{G}_1 \subset E(\mathbb{F}_q)$, and over the BLS12-381 curve it can be represented by $|q| = 381$ bits. In this regard, call **Compress** the (invertible) function that maps a point $P \in \mathbb{G}_1$, into a scalar $p \in \mathbb{F}_q$.

Using the efficient construction in [BGI16, Section 3.2], we define a DPF for the point function $F_{i,1} : \{0,1\}^n \to \mathbb{F}_q$, which returns 0 on every input

$j \in \{0,1\}^n$ such that $j \neq i$, and 1 on input i. This corresponds to a pair $(Gen, Eval)$ such that:

$$(k_0, k_1) \leftarrow Gen(1^\lambda, i, 1, \mathbb{F}_q),$$

and for every input $j \in \{0,1\}^n$:

$$\begin{cases} Eval(0, k_0, j) + Eval(1, k_1, j) = 0 & \text{if } j \neq i, \\ Eval(0, k_0, i) + Eval(1, k_1, i) = 1. \end{cases}$$

Note that, in our previous construction (Section 6.3.3), the output domain of *Eval* was $\mathbb{F}_2$. Therefore, we would have needed one distinct query for each bit of the target witness, as we could only retrieve *single* bits at a time. Instead, by choosing $\mathbb{F}_q$ as co-domain, we retrieve the full witness with a single query.

Using the 2-server PIR protocol presented in Section 6.3.3, a client interested in witness A_i queries each server with a different pair (b, k_b), with $b \in \{0,1\}$. On input a query bit b, and the respective key k_b, each server computes the result as as follows:

$$a_b = \sum_{i=1}^{2^n} Eval(b, k_b, i) \cdot \mathbf{Compress}(A_i), \tag{6.4}$$

with $a_b \in \mathbb{F}_q$. The client reconstruct the result as follows:

$$A_i = \mathbf{Compress}^{-1}(a_0 + a_1).$$

Correctness can be shown similarly as in the previous section (eq. (6.3)), while secrecy follows from secrecy of the DPF.

6.3.4 Experimental Results

In this section we provide experimental results of an implementation for the 2-server PIR protocol for witness retrieval presented in the previous section.

As for Chapter 4, the results are generated after 30 independent runs on a Mac M1 chip with 16GB of RAM. Remember that each witness has size 48 bytes.

Holder In Table 6.1, we measure the holder's computational and communication costs for retrieving a single witness. The column "*Gen*" contains the computational cost for generating the DPF keys k_0, k_1, while column "*Reconstruct*" indicates the cost for extracting the target witness from the two answers $a_0, a_1 \in \mathbb{F}_p$ (i.e., $A = \mathbf{Compress}^{-1}(a_0 + a_1)$).

The upload cost, is determined by the dimension of the DPF key-pair (k_0, k_1), while the download cost is given by the two answers a_0, a_1. Up to

$\log N$	*Gen*	*Reconstruct*	Upload	Download
10	12.83 μs	1.30 ms	426 B	96 B
12	15.10 μs	1.24 ms	498 B	96 B
14	17.60 μs	1.32 ms	570 B	96 B
16	19.87 μs	1.31 ms	642 B	96 B
18	22.65 μs	1.35 ms	714 B	96 B
20	24.54 μs	1.36 ms	786 B	96 B
21	25.68 μs	1.40 ms	822 B	96 B
22	26.94 μs	1.40 ms	858 B	96 B
23	28.30 μs	1.18 ms	894 B	96 B

Table 6.1: Holder's computational and communication costs for updating witnesses with increasing number of users.

$\log N$	*Answer*	Download	Upload
10	0.79 ms	213 B	48 B
12	3.07 ms	249 B	48 B
14	12.37 ms	285 B	48 B
16	49.82 ms	321 B	48 B
18	197.53 ms	357 B	48 B
20	791.83 ms	393 B	48 B
21	1.59 s	411 B	48 B
22	3.17 s	429 B	48 B
23	6.34 s	447 B	48 B

Table 6.2: Issuer's computational and communication costs for answering queries with increasing number of users.

$n = 23$ (i.e., $N = 8\,388\,608$), the total computational costs remains *under* 2 ms, while the maximum communication cost is around 1KB. Note that both values are *independent* of the number of revocations that happened since the holder last updated. This makes updating with 2-pir extremely efficient on the holder side.

Issuer In Table 6.2, we show the issuer's computational and communication cost. The *Answer* column indicates the computational time required for answering each individual query (as by eq. (6.4)). The server run-time is determined by computation of the full-domain evaluation using the holder's key, followed by calculating the inner product of each result with its corresponding database element. Due to time constraints, we tested a non-optimized version of *Eval*, which does not implement the *early termination* optimization described in [BGI16]. As a result, computing the full-domain evaluation of the database *dominates* the issuer's run-time. For instance, when $N = 2^{23}$, the full domain evaluation costs around 6.1s, while the inner product only takes the remaining 0.2s.

Reducing Issuer's Answer Time Our first observation is that the dimension database does not necessarily need to reflect the *actual* number of holders. This is because each holder is *anonymous* with respect to all other holders in the *same* database (i.e., each database defines an *anonymity set*). Assuming to have 10M holders, by splitting their witnesses into 10 different databases we achieve $\approx 10\times$ improvement in server performances, while still maintaining anonymity sets of 1M.

Additionally, we can reduce the cost of full domain evaluation by *restricting* the domain size from N to N/k, at the cost of increasing the holder's download size by a factor of k. To achieve this, we can view a database containing $N = 2^{20}$ witnesses as a $k \times (N/k)$ matrix. Each holder decomposes the index of the target witness into a pair (i_{row}, i_{col}), and computes the DPF keys embedding the point function $F_{i_{col},1}$ (i.e., for retrieving the i_{col}-th element of each row). To generate the answer, each server computes a full evaluation on an input domain of size N/k (instead of N). The results are used for computing k inner-products (one for each row), generating k different answers $a_{b,1}, \ldots, a_{b,k} \in \mathbb{F}_q$. After downloading the answers, the client keeps only the one corresponding to the row containing his witness (i.e., $a_{b,i_{row}}$), and discards the rest. Then, he reconstructs his witness as usual: $A_i = \textbf{Compress}^{-1}(a_{0,i_{row}} + a_{1,i_{row}})$.

In Table 6.3 we show the improvements for a database of size $N = 2^{20}$, with different values of k. We notice that for $k = 2^8$, the issuer's answer time is one *order of magnitude* faster than in Table 6.2, while the holder's download size of 25 kilobytes is still very practical.

	Computation			Communication	
$\log k$	Domain Evaluation	Inner Product	Total	Query	Answer
10	0.73 ms	46.78 ms	46.23 ms	213B	49 152B
9	1.46 ms	45.36 ms	46.82 ms	231B	24 576B
8	2.97 ms	45.98 ms	48.95 ms	249B	12 288B
7	5.89 ms	45.99 ms	51.88 ms	267B	6144B
6	11.70 ms	45.68 ms	57.38 ms	285B	3072B
5	23.51 ms	45.52 ms	69.03 ms	303B	1536B
4	47.27 ms	46.40 ms	93.67 ms	321B	768B
3	94.37 ms	47.65 ms	142.02 ms	339B	384B
2	185.84 ms	45.97 ms	231.81 ms	357B	192B
1	372.82 ms	47.84 ms	420.66 ms	375B	96B

Table 6.3: Issuer's communication costs and total run-time for ansewering a single query with $1M$ users and decreasing values of k. The total answer time is split in full-domain evaluation, and computation of the inner product.

As we can observe from the run-time, the improvements result only from reducing the input domain size of the evaluation from N to N/k, while the number of multiplications required for the inner product remains constant (i.e., N).

We highlight that the inner product computation is perfectly parallelizable: we can split the database rows between t CPU cores, and reduce the answer time by almost a factor of t. For instance, with $t = 2^4$ cores, and $k = 2^8$ we would achieve a total answer time of around 5.84 ms for each query.

6.3.5 An Alternative Update Strategy

From the results presented in the previous section, we conclude that 2-server PIR has the potential of being a scalable solution for achieving *anonymous* witness updates. With respect to directly fetching an up-to-date witness, 2-server PIR only increases the number of downloaded bits by a factor of 2, while the number of uploaded bits (i.e., the two DPF keys), is logarithmic in the number of holders. In comparison, the accumulator's update algorithm $\mathbf{WitUp}_{\mathcal{H}}$, has a *linear* communication overhead in the number of revocations.

Therefore, we could think of integrating PIR in the accumulator scheme presented in Chapter 4. Obviously, we would require a different threat model with (at least) two non-colluding servers. This scenario could be realistic when considering a wider e-ID ecosystem, where other digital credentials are issued on top of the government-issued e-ID by private service providers. In this setting, one of the two servers can be hosted by the government, and the other by the private issuer. Additional (semi-)trusted entities can be involved depending on the use case.

Assuming to have 10M users, an issuer adopting the naive PIR solution presented in Table 6.2 would take around 2.5 years to answer every client. Splitting the database into 10 smaller databases of 1M entries and using the optimizations discussed in the last section, we would reduce the issuer's total computational to 13 hours, or to 48 minutes using 16 cores. By using more cores and parallellizing the domain-evaluation, we can achieve better results.

Due to the considerable server costs for answering client queries, our 2-server PIR scheme would probably not be suitable for satisfying every update request. However, we could think of it as a replacement for revocation epochs. This could work as follows:

- the issuer sets a threshold u on the *maximum* update size that holders can afford to compute using the regular accumulator's update algorithm. When the number of updates reaches the threshold, the issuer sends up-to-date witnesses to the PIR servers.

- when some holder has too many updates to compute, he queries the PIR servers and efficiently retrieve a new witness. The witness is always less than u revocations behind the current accumulator value, therefore the holder computes updates for at most u revocations.

Note that holders queries are *indistinguishable* from random: even when a *single* holder performs a query, each server do not learn anything from the query itself. The advantage of 2-PIR compared to revocation epochs is that PIR queries can be performed *on-demand* and only by holders that have a large number of updates to compute.

As a final remark, we note that our exploration of multi-server PIR techniques was intended to assess the feasibility of the approach, rather than to identify the optimal solution for our use case. We have already mentioned that our implementation does not optimize the *Eval* algorithm with the *early termination* technique introduced in [BGI16]. As a comparison, the authors in [KOR19] have produced an optimized open-source C++ library [3], which seems to require less than 0.32 *ms* for a full domain evaluation with a database of 1M entries. Note that, as the cost of computing the inner product would become the bottleneck, the final server answer time would still be around 46 *ms*, with the advantage that holders would download a single answer per server (i.e., 98B in total).

A more in-depth study on 2-server PIR schemes and their application to anonymous witness update is left as interesting future work.

[3]`https://github.com/dkales/dpf-cpp/tree/master`

Chapter 7

Conclusion

In this chapter, we first give a brief summary of our improvements compared to some related accumulator schemes (Section 7.1). Then, we conclude this thesis with some final considerations (Section 7.2).

7.1 Improvements to Related Works

In this section, we highlight the improvements introduced by our revocation proposals compared to the related work on pairing based accumulators ([Ngu05; VB22; KB21; JLM22]).

Efficient membership proofs the accumulator constructions in [VB22; JLM22] adopt the zero-knowledge membership proof introduced in [Ngu05], which require 4 pairings operations. On the contrary, our proofs are based on recent sigma protocols for BBS disclosure, which do not require any pairing operation (see Section 4.3). In a software implementation, we showed that our scheme reduces the prover's run-time by $10\times$ with respect to previous approaches.

Efficient Batch Update some related works [VB22; JLM22] use polynomials to batch multiple updates. Evaluating the polynomials on some holder's input yields the update coefficients. However, in both [VB22], and [JLM22] (single-server construction), the holder evaluates the polynomials by definition, incurring in a overhead of m point multiplication. In Section 4.4.3, we applied Pippenger's approach for MSM to speed up polynomial evaluation. For increasingly large batch sizes, we demonstrated that our approach yields a performance increase of 3 to $4\times$, considerably improving the holder's runtime.

Aggregating Multiple Batches the works of Vitto et al. [VB22], and Jacques et al. [JLM22] do not offer any efficient way for aggregating subsequent batch updates. To update from time t to time $t + n$, holders have two alternatives: 1. sequentially apply the n update polynomials that were published after each batch-deletion, 2. ask the issuer to compute a polynomial batching all updates "on the fly". Each strategy comes with its own drawbacks:

1. computing update polynomials has quadratic complexity, hence creating them on-demand introduces high overhead on the issuer. The authors in [JLM22] partially address this problem by setting an upper-bound on the maximum degree of each polynomial. However, limiting the polynomial size also *reduces* the speed-up achieved by using Pippenger's method, which is proportional to the batch size;

2. the issuer might need to wait before publishing an update, so that he can batch more deletions in a single polynomial and allow for more efficient client evaluation (e.g., using Pippenger's method). However, when revocations need to be timely enforced, the issuer has less time to incorporate deletions in the batch, rendering the update more costly for holders;

In our accumulator scheme the issuer does not need to compute *any* update polynomial. Instead, he can instantly publish individual updates produced by the single deletion algorithm, and publish the respective update messages. Clients can then aggregate all the individual update messages in a single polynomial, at the cost of only m additional multiplication in $\mathbb{F}_p$ (see Section 4.4.4). In the library we used for our implementation, a single scalar multiplication takes around 18 ns, so the induced overhead remains negligible for any reasonable update size (e.g., around $10ms$ for $m \simeq 500\,000$). As a result, we both *eliminate* the quadratic complexity of computing update polynomials, and allow clients to *always* update in a single batch.

Limiting Holder Updates None of the related work[1] proposes a way to limit the maximum number of updates that holders need to compute. In large credential systems or with high revocation rates, this can impose considerable overhead on holders that have been offline for long periods. Depending on the trust assumptions, we proposed two solutions:

- in the single-server setting, we proposed the introduction of revocation epochs (Section 4.5). At the end of each revocation epoch, the issuer efficiently computes up-to-date witnesses for every user, and distribute them as a periodic update (e.g., once per week). As a result, before

[1]Except for the multi-server solution in [JLM22], which outsources the update using MPC

presenting a credential, users need to update only for revocations that occurred inside the current epoch, independently on when they last used their credential.

- assuming to have (at least) two non-colluding servers, we proposed to replace revocation epochs with PIR. Two PIR servers maintain a list of up-to-date witnesses, that are always updated after a given number of revocations occurred, say 5000. Thanks to the hiding property of PIR schemes, clients can just query their witness on-demand instead of periodically fetching an up-to-date witness. Then, they only need to update for the number of revocations after the retrieved witness was last updated (i.e., at most 5000). In our implementation (Section 6.3.4), we showed that with 2^{23} users, fetching an up-to-date witness requires less than 1 kilobyte *independently* on the batch size, and around 1.5 ms for any tested number of users.

Integrating accumulators-based revocation and BBS credential in Chapter 5, we proposed a protocol to bind accumulator membership proofs to BBS disclosure proofs. In this way, we integrate our revocation scheme to BBS credentials and increase the security properties (i.e., soundness) of our accumulator, without requiring the additional static signature used in [Ngu05; KB21; JLM22].

7.2 Conclusions

The purpose of this thesis was to assess whether it is possible to enforce privacy-preserving revocation of anonymous credentials on a national scale. Throughout the thesis, we mainly focused on cryptographic accumulators, as they enforce *instant* revocation and support *unlinkable* non-revocation proofs. The main argument against using accumulators in e-ID systems is that, after a credential is revoked, every holder in the system must perform computationally expensive update operations.

In Chapter 4, we improved existing accumulator solutions, achieving better update performances. Considering a worst-case scenario with 10M credentials and a 2% yearly revocation rate[2], we reduced the overhead of updating witnesses after a *month* of inactivity (i.e., 16.5K revocations) to less than 0.4s. In terms of communication, the latter scenario would require downloading around 1.3MB of updates. Despite such overhead could be considered acceptable, we proposed two methods to limit the maximum number of holder updates, namely revocation epochs and two-server PIR. We can safely conclude that cryptographic accumulators can already be

[2]Consider that even in a less secured environment as the web PKI, the revocation rate is normally less than 1%

considered a *practical* revocation method in a national e-ID setting, offering a viable alternative to the popular (but linkable) list-based solutions.

Beyond the scope of this thesis, we could consider even larger use cases. For instance, we could ask ourselves whether accumulator-based revocation could scale sufficiently to handle revocation for all credentials within the European Union. There are currently around 450M people living in the EU[3], assuming that everyone gets a credentials and a 2% yearly revocation rate, we would have around 25K daily revocations. Handling all these revocations with a single accumulator would probably be unfeasible, especially considering the high communication costs. Furthermore this solution would introduce governance problems: if the accumulator is shared (i.e., every country knows the trapdoor) states could potentially revoke credentials issued by other states.

In a more realistic scenario, each country would issue its own national e-ID and use a dedicated accumulator. Germany, the largest country in Europe, has around 85 million inhabitants. Managing this number of credentials with a single accumulator could be challenging. However, large nations can manage their credential space by using multiple accumulators (e.g., by randomly assigning each credential to an accumulator). This is because each holder remains anonymous among all holders associated with a specific accumulator, which defines an *anonymity set*. In Germany's use-case, using 8 different accumulators would achieve anonymity sets of size comparable to that of the Swiss e-ID, with similar overheads to those discussed in this thesis.

In conclusion, the scalability potential of accumulator-based revocation is not limited to medium or small countries; with adequate infrastructure, it could also be extended to larger nations and more complex settings, such as the European Union. Further research on the types of infrastructure that would best suit these settings could be an interesting topic for future work.

[3]`https://european-union.europa.eu/principles-countries-history/key-facts-and-figures/life-eu_en`

Acknowledgments

To Dr. Martin Burkhart for guiding me with his continuous advice and precious feedback.

To Prof. Serge Vaudenay for supporting me with his valuable insights and extreme cordiality.

To Dr. Simone Colombo for suggesting multi-server PIR and for his precious advice.

To Dr. Francesca Falzon for the interesting discussions on encrypted databases.

To Mr. Jonas Niestroj, and Mr. Christian Heimann for their insights on the Swiss e-ID.

To my brother Riccardo who inspired me to follow this path.

To my family and friends for being always there.

To energy drinks, for keeping me awake through this entire journey.

Without each and every one of you, this achievement would have never been possible. Grazie!

Bibliography

[Bal+17] Foteini Baldimtsi et al. “Accumulators with Applications to Anonymity-Preserving Revocation”. In: *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*. 2017, pp. 301–315. DOI: `10.1109/EuroSP.2017.13`.

[Bau+24] Carsten Baum et al. *Cryptographers’ Feedback on the EU Digital Identity’s ARF*. 2024.

[BD93] Josh Benaloh and Michael De Mare. “One-way accumulators: A decentralized alternative to digital signatures”. In: *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer. 1993, pp. 274–285.

[Ber+12] Daniel J Bernstein et al. “Faster batch forgery identification”. In: *Progress in Cryptology-INDOCRYPT 2012: 13th International Conference on Cryptology in India, Kolkata, India, December 9-12, 2012. Proceedings 13*. Springer. 2012, pp. 454–473.

[BGI16] Elette Boyle, Niv Gilboa, and Yuval Ishai. “Function secret sharing: Improvements and extensions”. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 2016, pp. 1292–1303.

[Blo70] Burton H. Bloom. “Space/time trade-offs in hash coding with allowable errors”. In: *Commun. ACM* 13.7 (July 1970), pp. 422–426. ISSN: 0001-0782. DOI: `10.1145/362686.362692`. URL: `https://doi.org/10.1145/362686.362692`.

[BS23] Matthias Babel and Johannes Sedlmeir. “Bringing data minimization to digital wallets at scale with general-purpose zero-knowledge proofs”. In: *arXiv preprint arXiv:2301.00823* (2023).

[CDH16] Jan Camenisch, Manu Drijvers, and Jan Hajny. “Scalable Revocation Scheme for Anonymous Credentials Based on n-times Unlinkable Proofs”. In: Oct. 2016, pp. 123–133. DOI: `10.1145/2994620.2994625`.

[CH10] Philippe Camacho and Alejandro Hevia. “On the impossibility of batch update for cryptographic accumulators”. In: *Progress* *in Cryptology–LATINCRYPT 2010: First International Conference on Cryptology and Information Security in Latin America, Puebla, Mexico, August 8-11, 2010, proceedings 1*. Springer. 2010, pp. 178–188.

[CKS09] Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente. "An accumulator based on bilinear maps and efficient revocation for anonymous credentials". In: *Public Key Cryptography–PKC 2009: 12th International Conference on Practice and Theory in Public Key Cryptography, Irvine, CA, USA, March 18-20, 2009. Proceedings 12*. Springer. 2009, pp. 481–500.

[CKS10] Jan Camenisch, Markulf Kohlweiss, and Claudio Soriente. "Solving revocation with efficient update of anonymous credentials". In: *International Conference on Security and Cryptography for Networks*. Springer. 2010, pp. 454–471.

[CL02] Jan Camenisch and Anna Lysyanskaya. "Dynamic accumulators and application to efficient revocation of anonymous credentials". In: *Advances in Cryptology—CRYPTO 2002: 22nd Annual International Cryptology Conference Santa Barbara, California, USA, August 18–22, 2002 Proceedings 22*. Springer. 2002, pp. 61–76.

[CV02] Jan Camenisch and Els Van Herreweghen. "Design and implementation of the idemix anonymous credential system". In: *Proceedings of the 9th ACM Conference on Computer and Communications Security*. 2002, pp. 21–30.

[DPC23] Alex Davidson, Gonçalo Pestana, and Sofía Celi. "Frodopir: Simple, scalable, single-server private information retrieval". In: *Proceedings on Privacy Enhancing Technologies* (2023).

[Dun22] Paul Dunphy. "A note on the blockchain trilemma for decentralized identity: Learning from experiments with hyperledger indy". In: *arXiv preprint arXiv:2204.05784* (2022).

[Eur23] European Digital Identity Wallet Consortium. *EUDI Architecture and Reference Framework v1.4.0*. 2023. URL: `https://eu-digital-identity-wallet.github.io/eudi-doc-architecture-and-reference-framework/1.4.0/`.

[FYC24] Daniel Fett, Kristina Yasuda, and Brian Campbell. *Selective Disclosure for JWTs (SD-JWT)*. Internet-Draft draft-ietf-oauth-selective-disclosure-jwt-08. Work in Progress. Internet Engineering Task Force, Mar. 2024. 84 pp. URL: `https://datatracker.ietf.org/doc/draft-ietf-oauth-selective-disclosure-jwt/08/`.

[GI14] Niv Gilboa and Yuval Ishai. “Distributed point functions and their applications”. In: *Advances in Cryptology–EUROCRYPT 2014: 33rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Copenhagen, Denmark, May 11-15, 2014. Proceedings 33*. Springer. 2014, pp. 640–658.

[Hen+23] Alexandra Henzinger et al. “One Server for the Price of Two: Simple and Fast {Single-Server} Private Information Retrieval”. In: *32nd USENIX Security Symposium (USENIX Security 23)*. 2023, pp. 3889–3905.

[Hop+16] Daira Hopwood et al. “Zcash protocol specification”. In: *GitHub: San Francisco, CA, USA* 4.220 (2016), p. 32.

[JLM22] Samuel Jaques, Michael Lodder, and Hart Montgomery. “ALLOSAUR: Accumulator with Low-Latency Oblivious Sublinear Anonymous credential Updates with Revocations”. In: *Cryptology ePrint Archive* (2022).

[KB21] Ioanna Karantaidou and Foteini Baldimtsi. “Efficient constructions of pairing based accumulators”. In: *2021 IEEE 34th Computer Security Foundations Symposium (CSF)*. IEEE. 2021, pp. 1–16.

[KC21] Nikita Korzhitskii and Niklas Carlsson. “Revocation Statuses on the Internet”. In: *Passive and Active Measurement: 22nd International Conference, PAM 2021, Virtual Event, March 29 – April 1, 2021, Proceedings*. Cottbus, Germany: Springer-Verlag, 2021, pp. 175–191. ISBN: 978-3-030-72581-5. DOI: `10.1007/978-3-030-72582-2_11`. URL: `https://doi.org/10.1007/978-3-030-72582-2_11`.

[KL24] Victor Youdom Kemmoe and Anna Lysyanskaya. “RSA-Based Dynamic Accumulator without Hashing into Primes”. In: *Cryptology ePrint Archive* (2024).

[KOR19] Daniel Kales, Olamide Omolola, and Sebastian Ramacher. “Revisiting user privacy for certificate transparency”. In: *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE. 2019, pp. 432–447.

[Lar+17] James Larisch et al. “CRLite: A scalable system for pushing all TLS revocations to all browsers”. In: *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2017, pp. 539–556.

[Liu+15] Yabing Liu et al. “An end-to-end measurement of certificate revocation in the web’s PKI”. In: *Proceedings of the 2015 Internet Measurement Conference*. 2015, pp. 183–196.

[Loo+23] Tobias Looker et al. *The BBS Signature Scheme.* Internet-Draft draft-irtf-cfrg-bbs-signatures-05. Work in Progress. Internet Engineering Task Force, Dec. 2023. 115 pp. URL: `https://datatracker.ietf.org/doc/draft-irtf-cfrg-bbs-signatures/05/`.

[Ngu05] Lan Nguyen. "Accumulators from bilinear pairings and applications". In: *Topics in Cryptology–CT-RSA 2005: The Cryptographers' Track at the RSA Conference 2005, San Francisco, CA, USA, February 14-18, 2005. Proceedings.* Springer. 2005, pp. 275–292.

[Par14] European Parliament. *Regulation (EU) No 910/2014 of the European Parliament and of the Council of 23 July 2014 on electronic identification and trust services for electronic transactions in the internal market and repealing Directive 1999/93/EC.* `https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32014R0910`. 2014.

[Reg09] Oded Regev. "On lattices, learning with errors, random linear codes, and cryptography". In: *Journal of the ACM (JACM)* 56.6 (2009), pp. 1–40.

[Ros+23] Michael Rosenberg et al. "zk-creds: Flexible anonymous credentials from zksnarks and existing identity infrastructure". In: *2023 IEEE Symposium on Security and Privacy (SP).* IEEE. 2023, pp. 790–808.

[Sco19] Michael Scott. "Pairing implementation revisited". In: *Cryptology ePrint Archive* (2019).

[Spo+20] Manu Sporny et al. *JSON-LD 1.1.* Tech. rep. W3C, July 2020. URL: `https://www.w3.org/TR/2020/REC-json-ld11-20200716`.

[TZ23] Stefano Tessaro and Chenzhi Zhu. "Revisiting BBS signatures". In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques.* Springer. 2023, pp. 691–721.

[VB22] Giuseppe Vitto and Alex Biryukov. "Dynamic universal accumulator with batch update over bilinear groups". In: *Cryptographers' Track at the RSA Conference.* Springer. 2022, pp. 395–426.