Comparing Privacy-Preserving Revocation for the EUDI Wallet

Andrea Flamini
Hasso Plattner Institute, University of Potsdam
andrea.flamini@hpi.de

Anja Lehmann
Hasso Plattner Institute, University of Potsdam
anja.lehmann@hpi.de

Giada Sciarretta
Center for Cybersecurity, Fondazione Bruno Kessler
g.sciarretta@fbk.eu

Mario Scuro
Center for Cybersecurity, Fondazione Bruno Kessler
mscuro@fbk.eu

Nicola Smaniotto
University of Trento
nicola.smaniotto@unitn.it

Alessandro Tomasi
Center for Cybersecurity, Fondazione Bruno Kessler,
altomasi@fbk.eu

Silvio Ranise
Center for Cybersecurity, Fondazione Bruno Kessler
University of Trento
ranise@fbk.eu

Abstract

The European Digital Identity Wallet has integrated anonymous credentials into its technical specifications, and singles out four constructions for privacy-preserving revocation, drawn from two families: positive dynamic accumulators and signed-pairs. The two families are described in the literature in substantially different terms, and no common basis for comparing them exists, which currently prevents informed and quantitative decision making. In this work, we give a unified treatment of both families, showing that signed-pairs, despite their very different presentation, can be expressed in the standard accumulator syntax. We use this to define a single revocation mechanism that any of the four constructions instantiates, which in turn allows us to compare the resulting mechanisms both at the protocol level and empirically. We measure the performance of all four across the full revocation lifecycle, on server-class hardware for the Status Manager and on a smartphone for the Holder and Verifier, with parameters taken from a live national eID scheme. No construction dominates in every aspect, and we make the resulting trade-offs explicit, showing which construction suits which deployment, and identify promising avenues for further improvement at the protocol level.

Keywords

Accumulators, Revocation, Digital Identity

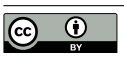
1 Introduction

Anonymous credentials are the privacy-preserving variant of classical attribute-based credentials. They enable a credential Holder to present the Issuer-attested attributes in a selective and unlinkable manner. These credential schemes are well-understood in the academic literature and are finally being seriously considered for deployment at national scale. The European Digital Identity (EUDI) Wallet has been extended with two technical specifications describing how anonymous credentials and zero-knowledge proofs can be implemented in the wallet, one based on multi-message signatures (TS14) [21] and one on arithmetic circuits (TS13) over ECDSA [20].

Privacy, however, must not come at the price of security: an anonymous credential system has to provide everything its non-private counterpart does. This is hardest for *revocation*, i.e., invalidating credentials that are lost, or no longer accurate. Classically the Issuer simply publishes the identifiers of the credentials that are no longer valid, and the Verifier looks up the one it was shown. But such an identifier is exactly what an anonymous credential must keep hidden. Instead, the Holder has to prove in zero-knowledge that its credential is not among the revoked ones. Over the past two decades a range of solutions has been proposed, from short-lived credentials and periodic re-issuance [14], through n-times unlinkable proofs [13], to cryptographic accumulators [15] and signed pairs [37]. From these, the European Commission has singled out, in TS14 [21, Section 9.6], four constructions across two families: positive dynamic accumulators and signed-pairs[1].

In *positive dynamic accumulators*, the Status Manager, the entity in charge of managing the status of credentials, publishes a compact digest of the set of valid credentials, and each Holder maintains a private witness of membership; the specifications consider an RSA-based and a pairing-based instantiation. In *signed-pairs*, the Status Manager publishes a signed list of the intervals between consecutive revoked identifiers, and a Holder proves that its own identifier lies strictly inside one of them; here the instantiations differ in the signature scheme and proof system used, namely BBS with Bulletproofs++, or ECDSA with a SNARK circuit. While the TS13 specification focuses on signed-pairs using ECDSA, TS14 recommends signed-pairs combined with short-lived attestations as a short-term strategy, leaving the optimal long-term choice open.

Making an informed choice among them requires a common understanding and a comparable basis, and to the best of our knowledge, neither exists yet. The difficulty is compounded by the fact that the two families are described in the literature in substantially different terms — one as a set-membership primitive with witnesses and update messages, the other as a signature-and-range-proof construction — so that there is no common syntax in which their guarantees can be compared, and no security notion stated once and shown to be met by both. Nor is there much empirical guidance.



This work is licensed under a Creative Commons Attribution 4.0 International License.

[1] The solution to manage revocation proposed for ZKP on arithmetic circuits in TS13 [20] is based on signed-pairs, signed with ECDSA.

1.1 Our Contribution

We give a unified treatment and comparison of privacy-preserving revocation for the EUDI Wallet, at both the protocol and the implementation level. We first bring the four candidate constructions of revocation schemes (Section 3) into a common description, which requires recognising that signed-pairs — despite their very different presentation in the literature — fit the same syntax as accumulators. On this basis we define a single revocation mechanism (Section 4) that any of them can instantiate, which lets us compare them conceptually, at the level of protocols and security (Section 4.1), and empirically (Section 5), through an implementation and benchmarks of all four across the full revocation lifecycle.

Common syntax and mechanism. We observe that signed-pairs, although never presented this way, can be expressed in exactly the standard accumulator syntax, with no algorithm requiring information that is not given. The fit is degenerate in an informative way — witness generation reads no secret key, and the update log stays permanently empty — and these two degeneracies turn out to predict much of what we later measure. On top of the syntax we define a *revocation mechanism* parameterised by an arbitrary *revocation scheme*. To keep the focus on revocation, we build on the committed disclosure framework of [33], binding the proof of non-revocation to the credential proof through a commitment, so that the two can be chosen and analysed entirely independently.

Comparison and benchmarks. At the protocol level, we identify where the two families genuinely differ and have their individual strengths: in what constrains the epoch length, in what a Holder leaks by fetching only the part of the revocation state it needs, in how tightly each scheme's security reduces to its underlying assumption (a loss of up to 40 bits at nation-scale query counts for the accumulators), and in how reinstatement is handled.

Empirically, we implement and measure all four mechanisms across all six phases, on a laptop for the Status Manager and on a Pixel 10 Pro for the Holder and Verifier, with parameters grounded in published figures from the Italian eID scheme, CIE. Our benchmarks and our RSA accumulator implementation are publicly available.

1.2 Related Works

We now reference some relevant papers testing revocation mechanisms, even if in a different setting, and some revocation schemes mentioned in TS14 that we do not analyze because of the low privacy guarantees they provide to users. We stress that the focus of this paper is on the revocation schemes proposed for adoption in the EUDI Wallet in TS14 [21, Section 9.6].

1.2.1 Works benchmarking revocation schemes. The most directly relevant measurement study [32] predates the recent optimized constructions [6, 30, 31, 46] and does not cover pairing-based schemes at all, while the recent work by Flamini et al. [23] compares accumulators against certificate revocation lists and OCSP, but for *non-anonymous* credentials and without considering signed-pairs. [35] surveys revocation mechanisms for blockchain applications, and test accumulators without optimal updates, though. [39] defines an API and benchmarks some dynamic accumulators in the context of Microsoft U-Prove, focusing only on accumulator-based revocation. No prior work, then, describes both families under a single syntax, and none reports a like-for-like performance comparison between them.

Other schemes considered in TS14. Other solutions that have been considered for adoption in TS14 [21] are short-lived attestations [9, 14], and Verifier local revocation [10]. While the former can be considered a viable option in contexts where the number of users is limited, it quickly becomes impractical in a nationwide context where revocation of a credential must be enforced in a short time, for instance, a day (e.g., national age verification credentials)[2]. The latter does not provide the same privacy guarantees as signed-pairs and accumulators, as presentations are not unlinkable against colluding Issuers, Verifiers, and Status Managers.

2 Anonymous Revocation

This section fixes the setting that the rest of the paper works in. We describe the roles involved in revocation and the objects they exchange, explain how we separate the proof of non-revocation from the credential presentation it is part of, give the six-phase lifecycle of a revocation mechanism in Section 2.1, and state the security notion it must satisfy in Section 2.2. Everything here is deliberately independent of any particular cryptographic construction we analyze, as is common to all of them: we say what a revocation scheme must provide, and defer to Section 3 the concrete schemes that provide it.

Entities. Revocation involves four roles. The **Issuer** signs credentials containing an identifier used to manage the credential's status. The **Status Manager**[3] maintains the status of many credentials by providing Holders and Verifiers with the information necessary to prove and verify credential status e.g., by publishing lists. The **Holder** stores the credential and possibly some cryptographic material received from the Status Manager and presents them to prove claims about its identity to the **Verifier**, who checks the Holder's presentations.

Revocation handles, material, and state. At issuance the Issuer embeds a unique identifier $h \in D$, the *revocation handle*, as an attribute of the credential. The domain D is fixed by the revocation scheme. The handle h is the means by which the credential's revocation is managed: informally, the credential is valid exactly when h belongs to the set of valid handles, and is revoked otherwise. For this reason we say interchangeably that a handle is revoked or that the credential carrying it is revoked.

Time is divided into *epochs*, indexed by t. For the current epoch the Status Manager publishes a *public revocation state* $(V_t, upmsg_t)$, consisting of a state value V_t, which identifies the set of valid handles, and an update log $upmsg_t$, which carries whatever a Holder needs in order to move its own material from one epoch to the next.

[2] Even if producing a signature took a time in the order of a millisecond, which is the case for BBS signatures [24, Figure 2], issuing a credential to tens of millions of users daily would take the Issuer hours to generate them.

[3] We distinguish between Issuer and Status Manager as e.g., in [26], though in practice their roles may be played by the same entity. The distinction is made for clarity of functions and associated costs borne by each, and to be consistent with separating the credential scheme handled by the Issuer and the revocation scheme handled by the Status Manager.

Alongside it, the Status Manager publishes a public key *mpk*, fixed at setup, against which presentations are verified. Each Holder keeps *revocation material* $W_{h|t}$ for its own handle, valid against V_t; this is what allows it to prove in zero-knowledge that h has not been revoked. To revoke a credential, the Status Manager revokes the associated handle and updates the public revocation state so that the affected Holders no longer hold valid material; Holders of surviving credentials fetch the new state, refresh $W_{h|t}$, and carry on generating presentations. Table 4 collects this notation.

Abstracting away the anonymous credential scheme. Proof of non-revocation is one conjunct of a larger privacy-preserving presentation, in which the Holder proves knowledge of an Issuer's signature on an attribute vector, selectively reveals some of those attributes, and proves predicates over the others. The revocation handle is simply one more attribute of the credential — one that is *never* revealed, since revealing it would make every presentation of that credential linkable. Instead, the Holder proves that the hidden handle is not on the Status Manager's blocklist.

Following the committed disclosure framework of [33], we split revocation and the core credential proof into two statements, glued together through a commitment. The Holder computes a commitment C to its revocation handle[4] and produces two proofs: a *credential proof*, that the handle signed in its credential opens C; and a *revocation proof*, that the value committed in C is not revoked under the current revocation state. The shared commitment is the only interface between them.

The advantage is modularity on two levels. Cryptographically, the credential and revocation schemes can be chosen, analyzed, and replaced independently, provided that there is a commitment scheme compatible both with proofs of equality for the credential scheme and with proofs of non-revocation under the revocation scheme. Practically, the same holds for the implementations: a wallet can pair a credential library with a revocation library without either being aware of the other. We therefore describe our revocation schemes as operating on an already-committed handle, and omit the credential proof and the equality proof throughout.

2.1 Revocation Mechanisms

We describe how to design a *revocation mechanism* for anonymous credentials, providing desirable features to satisfy, and the algorithms that must be defined to enable its deployment.

Free addition and optimal update. We restrict attention throughout to revocation schemes with two properties. *Free addition*: issuing a credential does not change the public revocation state, so no other Holder has to do anything when a new credential is issued. *Optimal update*: the cost of bringing a Holder's revocation material up to date does not grow with the number of credentials in circulation. Two consequences shape the rest of the paper. First, a freshly issued credential is usable immediately, since the Holder need not wait for the next state update. Second, the only event that forces Holders to act is revocation, so it is the number of revocations, and not the number of issuances, that determines how much work the ecosystem does to stay synchronised.

[4]Depending on the scheme, this may be, for instance, a Pedersen, Damgård–Fujisaki, or hash-based commitment.

Epochs and batched revocation. Revocation is not enforced the instant it is requested. Revocations requested during an epoch take effect at its end, when the Status Manager publishes the updated public revocation state, Holders refresh their revocation material against it, and Verifiers begin checking presentations against it. Epoch length is therefore a deployment parameter with a direct trade-off: it bounds how long a revoked credential remains usable, and at the same time bounds how often the Status Manager must recompute the state and every Holder must fetch it. Because revocations are enforced in batches, the batch size r is the natural parameter along which to compare mechanisms, and it is the main independent variable of our evaluation in Section 5.

Scope: no reinstatement. All the schemes we describe support *reinstatement*, that is, making a previously revoked credential valid again, but we exclude it from our analysis. Doing so reduces the number of operations to define, improves the concrete security bounds of the schemes (Section 4.1), and it costs little in practice: where credential issuance does not require a high level of assurance, re-issuing a credential is not a significant burden on Holders or Issuers. We return in Section 4.1 to what reinstatement would add.

With these conventions in place, a revocation mechanism consists of six phases. These six names are the vocabulary used for the rest of the paper, and in particular, they are the phases along which we report every measurement in Section 5.

(i) Setup. The Status Manager sets up the revocation scheme RS, publishing its public key *mpk* and the initial public revocation state $(V_0, upmsg_0)$.

(ii) Addition. When at time t the Issuer issues a credential signing a revocation handle h, the Status Manager adds h to the set of valid handles, and the Holder, with access to the public revocation state, is provided with the initial revocation material $W_{h|t}$.

(iii) Revocation. At the end of an epoch (see Section 2.1) the Status Manager enforces the revocation of a batch J of handles: it updates the public revocation state to $(V_{t+1}, upmsg_{t+1})$, so that the Holders of the revoked credentials no longer hold valid revocation material, and makes it available for anyone to download.

(iv) Holder Update. Having fetched the latest public revocation state, a Holder, whose credential survives, updates its revocation material from $W_{h|t}$ to $W_{h|t+1}$.

(v) Presentation. The Holder generates a zero-knowledge proof that it knows revocation material for the handle committed in C, valid against the current state value V_t.[5]

(vi) Verification. The Verifier checks the presentation against V_t and *mpk*.

In Section 4, we will show how to design revocation mechanisms from revocation schemes. The distinction is one of scope: a revocation scheme defines the atomic operations available to the actors of the ecosystem, while the mechanism places those operations into context and specifies who runs what, and when. The mechanism is defined once, parameterized by the kind of revocation scheme (RS) it is built on, and any such scheme can be used to instantiate it.

[5]As discussed above, this proof is combined with a proof of knowledge of the credential and a proof that the handle signed in the credential is the value committed in C.

2.2 Security Notion

The revocation mechanism must ensure that a presentation is accepted only for a credential that is valid under the current public revocation state. Following the standard notions [6, 15], we ask that no efficient adversary, allowed to drive Addition and Revocation adaptively — choosing which credentials are issued and which handles are revoked in each epoch — can produce a Presentation that Verification accepts at epoch t for a handle that is not valid under V_t, whether because it was revoked in an earlier epoch or because it was never issued at all.

Both halves of this requirement are needed, and they are enforced by different components. That the handle was never issued is ruled out by the credential, since the presentation includes a proof of knowledge of the Issuer's signature on h. That the handle was revoked is ruled out by the revocation scheme. It is shown in [6, Theorem 1] that the combination suffices: a *non-adaptively* sound revocation scheme, tracking which handles are valid and which are revoked, together with an unforgeable signature scheme[6] used to sign the handle in the credential, yields a mechanism secure against the adaptive adversary above. This is why we require only non-adaptive soundness of the schemes analysed in Section 3, and why abstracting the credential away is a presentational simplification rather than a weakening: the credential carries one half of the security argument.

Finally, we observe that the mechanisms we describe are not specific to anonymous credentials, and can equally be used to manage the revocation of non-anonymous ones. In that setting it remains crucial that the revocation handle is included as a credential attribute that is committed and never revealed, as described in [23].

3 Revocation Schemes

In this section we describe the revocation schemes we analyse, all of which have been considered for adoption for privacy-preserving revocation in the TS14 [21]. They come from two families: *positive dynamic accumulators with optimal update*, of which we consider an RSA-based and a pairing-based instantiation, and the *signed-pairs* technique, of which we consider a BBS-based and an ECDSA-based instantiation.

We present both families under a single syntax, namely the standard accumulator syntax of [6, 15, 31] with batched deletion. That signed-pairs fit it is not obvious — the technique is usually described in quite different terms, and its list of signed intervals is nothing like a compact accumulator value. The benefit of presenting both with the same syntax is that the zero-knowledge presentation of Section 3.3 and the revocation mechanism of Section 4 can each be defined once, over an arbitrary revocation scheme, and instantiated with any of the four schemes below.

Definition 1 (Revocation scheme). *A revocation scheme* RS *consists of the following algorithms, the first three run by the Status Manager and the last two by the Holder.*

$\mathsf{Gen}(1^\lambda) \xrightarrow{\$} (D, mpk, msk, V_0, upmsg_0)$ *produces the domain D of admissible revocation handles, a public/secret key pair, an initial state value V_0, and an initially empty public update log $upmsg_0$. The pair $(V_t, upmsg_t)$ is the public revocation state of Section 2, published alongside mpk.*

$\mathsf{GenWit}(msk, V_t, h) \to W_{h|t}$ *issues, for a newly admitted handle h, the revocation witness with which the Holder will prove non-revocation. The state value is not modified: addition is free, so no other Holder is affected.*

$\mathsf{Del}(msk, V_t, upmsg_t, J) \to (V_{t+1}, upmsg_{t+1})$ *revokes the batch J of handles, producing the new state value and extending the update log with whatever surviving Holders need in order to refresh their witness.*

$\mathsf{WitUp}(V_{t+1}, upmsg_{t+1}, upmsg_t, h, W_{h|t}, mpk) \to W_{h|t+1}$ *is run by a Holder whose handle survived the deletion, and turns a witness valid under V_t into a witness valid under V_{t+1}, using only public information published since its last update.*

$\mathsf{VerMem}(V_t, h, W_{h|t}, mpk) \to 0/1$ *checks that $W_{h|t}$ is valid for h under V_t. This algorithm is never run directly in our setting, since revealing h would destroy unlinkability; it is the relation the Holder proves in zero-knowledge instead (Section 3.3).*

Remark 1 (Batched deletion). *Since revocations are enforced in batches at epoch boundaries (Section 2.1), we let* Del *take a set J rather than a single handle, departing from the standard definitions [6, 15, 31]. Each of the schemes below exploits the batch, though in different ways, and unifying that in the syntax enables a meaningful comparison across families.*

Two axes of classification. Our definition of revocation schemes admits instantiations that behave very differently. Two properties are worth naming here, because they account for most of what Section 5 measures:

Public witness generation: A scheme has public witness generation if GenWit does not read *msk*, so that a Holder can derive its own witness from the public revocation state alone. Signed-pairs have this property; accumulators do not. The consequence is felt at the mechanism level: issuance requires no interaction with the Status Manager and costs it no computation, which is why the Addition phase is essentially free for one type and not for the other.

Log-based versus state-based update: A scheme is log-based if WitUp derives the new witness from the old witness together with the update log, and state-based if it derives it from V_{t+1} directly, ignoring *upmsg* and $W_{h|t}$. Accumulators are log-based, and *upmsg* carries one epoch's revocations; signed-pairs are state-based, and *upmsg* stays empty because the whole state value, containing all the witnesses, is republished in its place. This is the source of the bandwidth asymmetry analyzed in Section 5.3: a log-based Holder downloads an amount proportional to recent revocations, a state-based one an amount proportional to all revocations so far.

We will further compare the two approaches in Section 4.1. Now, we describe the relevant constructions we will analyze.

3.1 Accumulators

A *positive dynamic accumulator with optimal update* instantiates Definition 1 with a state value that is a compact digest of the set of valid handles. It is *positive* because the accumulated set is

[6] Note that an unforgeable signature scheme can be used to define a positive, additive and adaptively sound accumulator.

the allowlist, so a Holder proves membership rather than non-membership; [7] *dynamic* because handles may be added and removed after setup; and *optimal update* because the work a Holder does to keep its witness current is independent of the number of accumulated handles.

3.1.1 RSA-based Construction. RSA accumulators were first proposed in [8] and later improved in [7]. Authors in [15] formalize the concept of accumulator and propose a dynamic and positive accumulator that supports zero-knowledge proofs, making it applicable to the revocation of anonymous credentials. A universal accumulator based on RSA is described in [34], and in [6, Section 4.1] the authors propose an accumulator with free addition, i.e., the accumulator value does not need to be updated on add operations - in other words, when using the accumulator as an allow list of valid credentials, it does not need to be updated whenever a new credential is issued, only when one is revoked. We focus on the latter, referred to as CL-RSA-B, and defined by the following algorithms, starting from those executed by the Status Manager and then those executed by the Holder.

Construction 1 (RSA-based Accumulator [6]). *The accumulator is defined by the following algorithms:*

$(D, mpk, msk, V_0, upmsg_0) \xleftarrow{\$} \mathsf{Gen}(1^\lambda)$. *The Status Manager samples two safe primes of $\lambda/2$ bits $p = 2p' + 1, q = 2q' + 1$, sets $mpk \leftarrow pq = n$, and $msk \leftarrow (p', q')$. Then it samples a random generator for the group of quadratic residues module n: $V_0 \xleftarrow{\$} QR(n)$[8], sets $upmsg_0 \leftarrow []$, and sets D to be the set of prime numbers.*

$W_{h|t} \leftarrow \mathsf{GenWit}(msk, V_t, h)$. *The accumulator manager, when it adds a new element h to the accumulator, computes the associated witness $W_{h|t} \leftarrow V_t^{(h)^{-1} \bmod \frac{\phi(n)}{4}}$*[9].

$(upmsg_{t+1}, V_{t+1}) \leftarrow \mathsf{Del}(msk, V_t, upmsg_t, J)$ *To revoke the revocation handles in J, the Status Manager parses $\{h_i\}_{i\in[|J|]} \leftarrow J$, and performs, for all $i \in [|J|]$, sequentially, the following operations:*

- *updates the accumulator value $V_{t,i} \leftarrow V_t^{(h_i)^{-1} \bmod \frac{\phi(n)}{4}}$*
- *appends $upmsg_{t,i} \leftarrow upmsg_t.\mathsf{Append}(h_i, V_{t,i})$.*
- *if $i < |J|$ sets $V_t \leftarrow V_{t,i}$, $upmsg_t \leftarrow upmsg_{t,i}$*

Status Manager sets $V_{t+1} \leftarrow V_{t,|J|}$ and $upmsg_{t+1} \leftarrow upmsg_{t,|J|}$.

$W_{h|t+1} \leftarrow \mathsf{WitUp}(V_{t+1}, upmsg_{t+1}, upmsg_t, h, W_{h|t}, mpk)$. *The Holder, who is in possession of a witness for V_t, parses $[(h_1, V_{t,1}), \ldots, (h_{|J|}, V_{t,|J|} = V_{t+1})] \leftarrow upmsg_{t+1} \setminus upmsg_t$ and for each $i \in [|J|]$:*

- *computes the Bézout coefficients b_i, c_i such that $b_i h + c_i h_i = 1$.*
- *sets $W_{h|t,i} \leftarrow W_{h|t}^{c_i} \cdot V_{t,i}^{b_i} \bmod n$.*
- *if $i < |J|$, sets $W_{h|t} \leftarrow W_{h|t,i}$.*

The Holder sets $W_{h|t+1} \leftarrow W_{h|t,|J|}$.

$0/1 \leftarrow \mathsf{VerMem}(V_t, h, W_{h|t}, mpk)$. *The Holder verifies that its witness is indeed valid for its element h checking that $V_t \stackrel{?}{=} W_{h|t}^h \bmod n$.*

Batching Updates. As noted in [15], it is possible to batch multiple updates: in this case Del outputs the singleton $upmsg = [(V_{t,|J|}, \Delta)]$, where $\Delta = \prod_{i=1}^{|J|} h_i$ is the product of all the elements removed since the previous update. This benefits both the Status Manager, which does not need to compute all the intermediate accumulator values, and the Holder, who does not need to download them.

3.1.2 Pairing-based Construction. Originally proposed in [38] as a dynamic accumulator, in [17] is presented as a positive accumulator with zero-knowledge proofs. This accumulator was subsequently extended with proofs of non-membership in zero-knowledge in [5], making it universal. The construction is proven secure under the q-strong Diffie-Hellman assumption. Efficient batch operations and the use of Type-III pairings have been introduced in [46], and the algorithms to accumulate values with free addition are proposed in [31]. We refer to the latter as KB21, which is defined by the following algorithms

Construction 2 (EC-based Accumulator [31]). *Let $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, G_1, \tilde{G}_2, \mathbf{e})$ be a bilinear group of order p, a λ-bits prime number. Notation-wise, elliptic curve points in $\mathbb{G}_1$ are represented with capital letters (e.g. G), and those in $\mathbb{G}_2$ with capital letters and a tilde (e.g. $\tilde{G}$).*

$(D, mpk, msk, V_0, upmsg_0) \xleftarrow{\$} \mathsf{Gen}(1^\lambda)$. *The Status Manager samples $msk = x \xleftarrow{\$} \mathbb{Z}_p^*$, sets $mpk \leftarrow \tilde{G}_2^x$. Then it samples $u_0 \xleftarrow{\$} \mathbb{Z}_p^*$ and sets $V_0 \leftarrow G_1^{u_0}$, sets $upmsg_0 \leftarrow []$, and sets $D = \mathbb{Z}_p \setminus \{-msk\}$.*

$W_{h|t} \leftarrow \mathsf{GenWit}(msk, V_t, h)$. *The accumulator manager, when it adds a new element h to the accumulator, computes the associated witness $W_{h|t} \leftarrow V_t^{\frac{1}{msk+h}}$.*

$(upmsg_{t+1}, V_{t+1}) \leftarrow \mathsf{Del}(msk, V_t, upmsg_t, J)$ *To revoke the revocation handles in J, the Status Manager parses $\{h_i\}_{i\in[|J|]} \leftarrow J$, and performs, for all $i \in [|J|]$, sequentially, the following operations:*

- *updates the accumulator value $V_{t,i} \leftarrow V_t^{\frac{1}{msk+h_i}}$.*
- *appends $upmsg_{t,i} \leftarrow upmsg_t.\mathsf{Append}(h_i, V_{t,i})$.*
- *if $i < |J|$ sets $V_t \leftarrow V_{t,i}$, $upmsg_t \leftarrow upmsg_{t,i}$, increases i.*

Status Manager sets $V_{t+1} \leftarrow V_{t,|J|}$ and $upmsg_{t+1} \leftarrow upmsg_{t,|J|}$.

$W_{h|t+1} \leftarrow \mathsf{WitUp}(V_{t+1}, upmsg_{t+1}, upmsg_t, h, W_{h|t}, mpk)$. *The Holder, who is in possession of a witness for V_t, parses $[(h_1, V_{t,1}), \ldots, (h_{|J|}, V_{t,|J|} = V_{t+1})] \leftarrow upmsg_{t+1} \setminus upmsg_t$ and for each $i \in [|J|]$:*

- *sets $W_{h|t,i} \leftarrow (W_{h|t} \cdot V_{t,i}^{-1})^{\frac{1}{h_i - h}}$.*
- *if $i < |J|$, sets $W_{h|t} \leftarrow W_{h|t,i}$.*

The Holder sets $W_{h|t+1} \leftarrow W_{h|t,|J|}$.

$0/1 \leftarrow \mathsf{VerMem}(V_t, h, W_{h|t}, mpk)$. *The Holder verifies that its witness is indeed valid for its element h, checking that $\mathbf{e}(V_t, \tilde{G}_2) \stackrel{?}{=} \mathbf{e}(W_{h|t}, \tilde{G}_2^h\, mpk)$*

[7] The dual notion, accumulating revoked handles, is a *negative* accumulator, whereas one supporting both membership and non-membership proofs is *universal*. Since verifiers mainly just want to confirm a credential is valid, rather than keeping track of revoked ones, relying on a non-membership proof does not make much practical sense here.

[8] V_0 is a quadratic residue modulo n if there exists $a \in \mathbb{Z}_n$ such that $V_0 = a^2 \bmod n$.

[9] Where $\frac{\phi(n)}{4}$ is the product $p'q'$ only known to the Status Manager.

Batching Updates. Given the revocation witness at epoch t to be $[(h_1, V_{t,1}), \ldots, (h_{|J|}, V_{t,|J|})]$, applying WitUp sequentially $|J|$-times we obtain the witness at time t:

$$W_{h|t} \tag{1}$$

$$= ((W_{h|t-1} \cdot V_{t,1}^{-1})^{(h_1-h)^{-1}} \cdots V_{t,|J|}^{-1})^{(h_{|J|}-h)^{-1}} \tag{2}$$

$$= W_{h|t-1}^{\prod_{j=1}^{|J|}(h_j-h)^{-1}} \prod_{i=1}^{|J|} V_{t,i}^{-\prod_{j=i}^{|J|}(h_j-h)^{-1}} \tag{3}$$

The problem above takes the name of Multi-Scalar Exponentiation, which is efficiently solvable by adopting Pippenger's algorithm [42]. The key point is that the larger the *upmsg*, the more efficiently we can solve that product.

3.2 Signed-Pairs

Signed-pairs [37] take a different approach to representing the valid set. Rather than a compact digest against which each Holder holds a private witness, the Status Manager publishes the revoked handles explicitly, as an ordered list, and signs the gap between each pair of consecutive entries. A handle is valid exactly when it lies strictly inside one of these gaps, and the Holder's revocation witness is the signed gap bracketing it, an entry of the public list. To prove non-revocation, the Holder proves knowledge of a signature on a gap together with the fact that its handle lies inside it, revealing neither the handle nor which gap was used.

Two features of the technique are worth stating before the construction. First, since the published state V already contains every Holder's witness, as generated on the previous Del, GenWit has nothing to compute: signed-pairs have public witness generation in the sense above. Second, stale entries must not remain usable once their gap has been split, so each signature covers the epoch index t alongside the two endpoints, and t is revealed at presentation. This forces every surviving gap to be re-signed at every epoch, which we will see dominates the Status Manager's cost in Section 5.

Construction 3 (Signed Pairs [37]). *Let* DS *be a multi-message signature scheme supporting efficient zero-knowledge proofs of knowledge of a signature and of range relations over signed messages.*

$(D, mpk, msk, V_0, upmsg_0) \xleftarrow{\$} \mathsf{Gen}(1^\lambda)$. *The Status Manager samples* $(msk, mpk) = \mathsf{DS.KGen}(\lambda)$. *Then it sets* $D = [N]$, *where* N *is the maximum number of revocation handles, computes* $\sigma_{0,N+1}^{(0)} \xleftarrow{\$} \mathsf{DS.Sign}((0, 0, N+1), msk)$, *sets* $upmsg \leftarrow []$, *and sets the accumulator value* $V_0 \leftarrow (0, \{(0, N+1, \sigma_{0,N+1}^{(0)})\})$.

$W_{h|t} \leftarrow \mathsf{GenWit}(msk, V_t, h)$. *The Status Manager does not generate a new witness but simply fetches one of the signed-pairs from the already published accumulator value* V_t, *generated during the last* Del *operation. Note that while msk has been left as an input to this interface for compatibility with the accumulator syntax, it is not involved in a simple look-up.*[10]

$(upmsg_{t+1}, V_{t+1}) \leftarrow \mathsf{Del}(msk, V_t, upmsg_t, J)$ *To revoke the revocation handles in* J, *parse* $\{h_i\}_{i \in [|J|]} \leftarrow J$, *parses*

$$\left(t, \left\{\left((i_j, i_k), \sigma_{i_j,i_k}^{(t)}\right)\right\}_{(i_j,i_k)}\right) \leftarrow V_t,$$

sets $R_t \leftarrow \{i_j \in [N] : \exists i_k : (i_j, i_k) \in V_t\} \cup \{0, N+1\}$, $R_{t+1} \leftarrow R_t \cup J$ *and turns it into a ordered array where elements are included in* ascending order $A_{t+1} \leftarrow [h_i]_{i \in R_{t+1}}$. *Then it sets* $S \leftarrow \emptyset$ *and, for all consecutive elements* (i_j, i_k) *in* A_{t+1}

- *computes* $\sigma_{i_j,i_k}^{(t+1)} \xleftarrow{\$} \mathsf{DS.Sign}(t+1, i_j, i_k, msk)$.
- *sets* $S \leftarrow S \cup \{(i_j, i_k, \sigma_{i_j,i_k}^{(t+1)})\}$.

Finally it sets $V_{t+1} \leftarrow (t+1, S)$ *and* $upmsg_{t+1} \leftarrow []$.

$W_{h|t+1} \leftarrow \mathsf{WitUp}(V_{t+1}, upmsg_{t+1}, upmsg_t, h, W_{h|t}, mpk)$. *The Holder does not update its witness but simply fetches it from the already published accumulator value* V_{t+1}.

$0/1 \leftarrow \mathsf{VerMem}(V_t, h, W_{h|t}, mpk)$. *The Holder verifies that its witness* $W_{h|t} \rightarrow (t, (i_j, i_k), \sigma_{i_j,i_k}^{(t)})$ *is indeed valid for its element* h *under the accumulator value* V_t[11], *verifying that the signature* $\sigma_{i_j,i_k}^{(t)}$ *on messages* (t, i_j, i_k) *verifies under mpk and that* $i_j < h < i_k$.

The construction fits our joint syntax of Definition 1 in a way that is worth making explicit, since it is the crux of the unification: WitUp must produce a fresh witness at every epoch, because every signature is re-issued, and it can do so only because V_{t+1} is already among its inputs. Signed-pairs therefore use the syntax in a degenerate but entirely legitimate manner, with *upmsg* permanently empty and the whole state value republished in its place. This is precisely the state-based update mode identified above, and the cost of the choice is quantified in Section 5.3.

Instantiations. The original scheme uses BBS+ signatures [4], but any DS admitting a proof of knowledge of a signature together with range proofs over signed messages will do. We consider two. The first, BBS-BP, uses BBS [45] with Bulletproofs++ [19] for the two range conditions. The second, ECDSA-LF, uses ECDSA viewed as a multi-message signature scheme, with the circuit-based approach of [25], in which one circuit encodes both the signature verification and the range conditions. Section 3.3 gives the corresponding proof systems.

Non-adaptive Soundness (Proof Sketch). Signed-pairs satisfy the non-adaptive soundness required in Section 2.2, by reduction to the unforgeability of DS. A reduction $\mathcal{B}$ receives the public key pk from the UF-CMA challenger C and uses it as mpk in the soundness experiment. Since the experiment is non-adaptive, $\mathcal{B}$ knows in advance which handles will be revoked, and can therefore obtain from C exactly the signatures needed to build V_t at every epoch t. Suppose the adversary outputs a handle $h^\star$ that is not accumulated as of epoch t, together with material accepted by $\mathsf{VerMem}(V_t, h^\star, \cdot, mpk)$.[12] That material is a valid signature on some (t, u, v) with $u < h^\star < v$. But $h^\star$ being revoked at epoch t makes it a boundary of V_t, so no gap of

[10] We have chosen to include the list of signed-pairs as part of the accumulator value instead of as part of the update messages because otherwise GenWit would have needed to take also *upmsg* as an additional input.

[11] Actually, to verify it is sufficient to know the value t, not the whole accumulator value that includes the list of signatures.

[12] Since signed-pairs admit every handle of D at setup, a non-accumulated handle here is necessarily one that has been revoked, and never one that was simply never issued — unlike the accumulators above. The mechanism rules the latter case out through the credential instead; see Section 2.2.

V_t strictly contains it, and $\mathcal{B}$ never requested a signature on such a triple: either the endpoints do not bracket $h^\star$ in V_t, or the epoch index does not match.[13] The output is therefore a forgery.

Two caveats. The argument establishes soundness in the sense of Definition 1, where the witness is given in the clear; in the mechanism it is instead proven in zero-knowledge (Section 3.3), so the full argument additionally requires extracting the signature and the endpoints from the proof, and the tightness of the resulting bound is that of the extractor and not of the signature reduction alone. We return to this in Section 4.1. Second, the reduction relies on the endpoints proven in zero-knowledge being the ones actually signed; Section 3.3 describes how each instantiation enforces that binding.

3.3 Zero-Knowledge Presentation

Now that we have described positive, dynamic accumulators with optimal updates and signed-pairs using the same syntax, we introduce the essential operations that enable their adoption in the design of privacy-preserving revocation mechanisms, namely the ability to prove in zero-knowledge that the prover knows a witness for its revocation handle h committed in C, which according to the committed disclosure framework is indeed the same revocation handle signed in the credential of the user. This requires defining the following two algorithms:

- $\pi \xleftarrow{\$} \mathsf{GenZKP}(C, h, l, W_{h|t}, V_t, mpk)$. Given the commitment C of h and opening material l compute a proof of knowledge of the pair $(h, W_{h|t})$, such that h is committed in C, and $W_{h|t}$ is a valid witness for h under V_t. This can be done designing a ZKP for the following relation

$$\mathcal{R} = \Big\{ \underbrace{(h, l, W_{h|t})}_{\text{Holder's secret}}, \underbrace{(C, V_t, mpk)}_{\text{statement}} : \mathsf{Open}(C, h, l) = 1 \wedge \mathsf{VerMem}(V_t, h, W_{h|t}, mpk) \Big\}. \quad (4)$$

- $0/1 \leftarrow \mathsf{VerZKP}(\pi, C, V_t, mpk)$. The presentation can be verified running the verification algorithm associated with the adopted ZKP for $\mathcal{R}$.

Below we describe which ZKP we have considered for our analysis of the solutions presented in the previous sections.

Zero-Knowledge Accumulators. The algorithms GenZKP and VerZKP are also supported by the accumulators presented in Section 3.1.1 and Section 3.1.2. For the RSA accumulator we consider the ZKP introduced in [15] and revisited in [43] [14], while for the EC accumulator, we consider the ZKP from [13] for Boneh-Boyen Signatures, adapted to the accumulator set membership relation. We recall in Section B.2 both the ZKP for the RSA accumulator, where the revocation handle is revealed in committed form, using Damgård-Fujisaki [16] commitment, and the ZKP for the EC accumulator, where the revocation handle is committed using Pedersen commitments [41]. For both constructions, the NIZKP is generated by turning a sigma protocol for the relation $\mathcal{R}$ into a non-interactive scheme using the Fiat-Shamir transform [22].

[13] The epoch index closes the second case. Without it, a signature on a gap of an *earlier* list, which did strictly contain $h^\star$, would still verify.

[14] We add a few additional corrections to the generation of C_r, as described in [23].

Zero-Knowledge Signed Pairs. For signed-pairs instantiated using the signature scheme DS, the witness $W_{h|t}$ is derived from V_t, selecting the signature $\sigma^{(t)}_{h_i,h_{i+1}}$ of (t, h_i, h_{i+1}) s.t. $h_i < h < h_{i+1}$. In this case, the relation $\mathcal{R}$ described in Equation (4) can be rewritten, making the conditions defining VerMem for signed-pairs more explicit, obtaining the relation $\mathcal{R}'$.

$$\mathcal{R}' = \Big\{ \Big((h,\ l,\ h_i,\ h_{i+1},\ \sigma^{(t)}_{h_i,h_{i+1}}),\ \underbrace{(C,\ V_t,\ mpk)}_{\text{statement}} \Big) : \mathsf{DS.Verify}(mpk, (t, h_i, h_{i+1}), \sigma^{(t)}_{h_i,h_{i+1}}) = 1 \wedge \wedge\ h_i < h < h_{i+1} \wedge \mathsf{Open}(C, h, l) = 1 \Big\} \quad (5)$$

The proof combines a ZKPoK of the signature $\sigma^{(t)}_{h_i,h_{i+1}}$ on the tuple $(t,\ h_i,\ h_{i+1})$ and two range proofs encoding the strict inequalities $h_i < h$ and $h < h_{i+1}$. When the signed pair revocation scheme is instantiated using BBS [45], we have a sigma protocol to prove knowledge of the signature $\sigma^{(t)}_{h_i,h_{i+1}}$ revealing the first attribute, and we can use Bulletproofs++ [19] to prove the range proof relations. The handle of the prover and those signed in the witness are committed using Pedersen commitments [41]. Another popular instantiation of signed-pairs is the one adopted in [25], where the signature scheme is ECDSA, and the proof system to prove knowledge of the signature and the range relation is done using a circuit-based approach. In this case, the revocation handle can be committed using hash-based commitments [11].

4 Instantiating Revocation Mechanisms

Now we can proceed with the general description of the privacy preserving revocation mechanisms built on top of the *zero-knowledge* positive dynamic accumulators with optimal updates, and *zero-knowledge* signed-pairs, as revocation schemes. In the following, we will refer to the general revocation scheme with the symbol RS.

Definition 2 (Revocation Mechanism from RS). *When the Issuer issues a new credential, it includes, as one of the attributes, a random unique revocation handle $h \in D$, where D is the set of all possible revocation handles.*

(i) Setup. *The Status Manager runs*

$$\mathsf{RS.Gen}(1^\lambda) \rightarrow (D, mpk, msk, V_0, upmsg_0),$$

and publishes its public key mpk and the public revocation state *given by* $(V_0, upmsg_0)$.

(ii) Addition. *For every issued credential with revocation handle h, the Status Manager runs* $\mathsf{RS.GenWit}(msk, V_t, h)$ *making available the Holder with the necessary initial* private revocation material $W_{h|t}$.

(iii) Revocation. *When at the end of an epoch t the revocation of a set J of revocation handles must be enforced, the Status Manager runs* $\mathsf{RS.Del}(msk, V_t, upmsg_t, J)$*, and publishes the updated public revocation state* $(V_{t+1}, upmsg_{t+1})$.

(iv) Holder Update. *The Holder fetches the latest public revocation state* $(upmsg_{t+r}, V_{t+r})$ *and updates its local revocation material as follows: being* $(upmsg_t, V_t)$ *the last witness update*

performed by the Holder, it runs sequentially for $i \in [r]$*:*

$$\mathsf{RS.WitUp}(V_{t+i}, upmsg_{t+i}, upmsg_{t+i-1}, h, W_{h|t+i-1}, mpk)$$

and outputs the updated local revocation material $W_{h|t+r}$*.*

(v) Presentation. *The Holder generates a zero-knowledge proof* π *that its revocation handle* h*, committed in* C *(using randomess* l*) belongs to the set of valid handles, running*

$$\mathsf{RS.GenZKP}(C, h, l, W_{h|t}, V_t, mpk),$$

and sends it to the Verifier.

(vi) Verification. *The Verifier checks the proof against the current accumulator value* V_t *running* $\mathsf{RS.VerZKP}(\pi, C, V_t, mpk)$*.*

4.1 Functional Comparison Between Instantiations

We now describe some relevant distinctions between the instantiation of the revocation mechanism in Definition 2 with accumulators and signed-pairs as revocation schemes due to their design.

Hardware Support for msk. Use cases demanding a high level of assurance typically require that private key material be held in certified hardware, and this is one respect in which the two families start from different positions. ECDSA is a signature scheme with widespread support in HSMs, so a signed-pairs instantiation built on ECDSA has a clear head start. That head start is narrower than it first appears, however: certification requirements extend to the entire cryptographic stack, not only the signature scheme. A mechanism whose non-revocation proof is a circuit-based argument therefore needs the circuit and the proof system certified as well, which remains a challenging and likely time-consuming task.

For the accumulators the situation is less clear-cut, and we are not aware of hardware offerings that support either directly. The RSA-based construction is the closer of the two to widely deployed capability, though its Status Manager operations are exponentiations with an exponent derived from $\phi(N)$, rather than the RSA signing operation that HSM interfaces expose; pairing-friendly curves remain rare in certified hardware.

It is worth being precise about what such hardware would protect here. The Status Manager's key is not a credential-issuing key: an adversary who compromises it can produce revocation material for a handle that has been revoked — reinstating a revoked credential for a colluding Holder — or ban a valid credential. However, compromise of the Status Manager's does not allow to forge a credential, since that requires the Issuer's signing key. Whether revocation warrants the same protection as the issuing key is a deployment decision, and one that may reasonably be answered differently from the reflex of protecting every long-term key equally.

Fetching less data to update the witness. While standard privacy properties (such as multi-show unlinkability and untraceability [12]), are satisfied based only on the cryptographic material included in presentations, additional privacy threats may still arise at the architectural or network level. For example an important consideration when comparing cryptographic mechanisms is that users might download the information needed to update their witness right before generating a presentation. This means that, for efficiency, they might try to reduce the information they download to the bare minimum rather than downloading the entire public revocation state. This behavior results in leaking to the Status Manager, who hosts the public revocation state, information on who the Holder downloading part of the public revocation state is. But then, if the Status Manager colludes with the Verifier, this leaked information can be exploited to make the Holder's presentations linkable (not because of the data included in the presentation, but because of a misuse of the revocation mechanism).

For the signed-pairs, the minimum information Holders would download are the only relevant signature on the indices bracketing the a Holder's handle, leaking to the Status Manager the interval where the accumulated element lies, while for accumulators this would be the update messages and intermediate accumulator values since the last time a Holder has updated its witness, leaking the time of the Holder's last witness update. While for signed-pairs, downloading only the Holder's relevant signature before a presentation always reduces the privacy of the user, for accumulators, if we assume that the majority of the users will download the public revocation state at every epoch, then instructing users to download only the updates they miss degrades their privacy mildly.

On the concrete security of the constructions. When analyzing the security of a cryptographic primitive, the common path is to relate the hardness of breaking the security of that primitive to the hardness of a well-studied problem by designing reductions. In this paragraph, we argue why signed-pairs appear to provide better security guarantees compared to the accumulator constructions we have presented. Even if we do not show it formally, it should be clear from the sketch in Section 3.2 that breaking the non-adaptive security of the signed-pairs is at least as hard as breaking the unforgeability of the signature schemes used to sign the pairs. Instead, the security proofs of the accumulators that we analyse both require the reduction to their corresponding assumption[15] to make a random guess that will be successful with probability $\frac{1}{q^2}$ where q is the number of queries to add and remove an element from the accumulator during the execution of the non-adaptive security experiment, and this degrades the concrete security of the scheme by a factor q^2. In cases where the number of additions and revocations is not large, this does not have a significant impact on the security of the scheme. However, in nationwide applications, the numbers can grow, and considering an adversary who can perform up to 2^{20} queries is more than appropriate. This means that the scheme loses up to 40 bits of security due to the random guesses.

This problem can be possibly solved by designing tighter proofs of security for accumulators, showing that this security loss is not inherent to the construction itself but an artifact of the reduction, or, if the first is not possible, by using multiple instances of an accumulator to reduce the number of operations each accumulator manager must perform. However, the latter approach sacrifices the privacy of the users, as they would have to specify where their handle is being accumulated.

Adding reinstatement. If we consider reinstatement of revoked credentials, for accumulators this happens as a regular addition, where the Status Manager gives the Holder the new witness for its handle. For signed-pairs, the Holder must wait until the end of the

[15]The strong RSA assumption for the RSA-based accumulator [6], and q-strong Diffie Hellman assumption for the EC-based accumulator [31]

current epoch t before being able to use the reinstated credential associated to the revocation handle h, unless the Status Manager sends to the Holder (or publishes) a the signature $\sigma^{(t)}_{h-1,h+1}$, of the tuple $(t, h-1, h+1)$.

5 Experimental Evaluation

The comparison of Section 4.1 identifies, at the protocol level, where accumulators and signed-pairs diverge and why: in what bounds the epoch length, in what a partial fetch of the revocation state leaks, and in how tightly each scheme's security reduces to its underlying assumption. What that discussion cannot say is how large these differences actually are once realistic parameters are plugged in, and whether a difference that looks decisive in the abstract still matters at the scale of an actual deployment. This is the gap this section closes: we test and benchmark them across their full lifecycle. In Section 5.1, we describe the experimental set-up, and we describe the use case we analyse, which is the revocation management in a nationwide application (e.g. national age verification credentials), for which we consider the data on issuance and revocation of the CIE, the Italian identity card. Then in Section 5.2 we analyze the operations performed by the Status Manager, and in Section 5.3 those executed by Holder and Verifier. Finally, we summarize our analysis in Section 5.4.

5.1 Experimental Set-Up

Software. To allow a repeatable experiment, we make available a repository — `anonymous-revocation-benchmark`[16] — containing our benchmarks and our own RSA accumulator implementation; the code is written mainly in Rust, with a few bash scripts to facilitate cross-compilation and execution on mobile devices.

The CL-RSA-B implementation we provide is based on `unknown_order` [3]. It supports some of the most popular pure-Rust multiple-precision libraries, as well as the more mature `GMP` [2] and `OpenSSL` [40]. We prioritize performance, adopting the latter as backend.

For KB21, BBS, and Bulletproofs++ implementations we rely on `docknetwork` [18]. Meanwhile, for ECDSA and its zk-SNARK circuit we rely on Google's official `longfellow-zk` [28]. All four implementations can be considered quite mature in terms of primitive performance, with the important caveat that `docknetwork` is not optimized for mobile architectures, possibly incurring some overhead.

Hardware. The four mechanisms we benchmark split cleanly by who runs them. The Status Manager runs Setup, Add, and Revocation (i–iii), so we measure those on a high-end Dell Latitude 7450 laptop with an Intel Core Ultra 7 165H CPU and 32 GB of RAM. Holder Update, Presentation, and Verification (iv–vi) instead run on whatever hardware the Holder and Verifier have, which in the EUDI Wallet is typically a smartphone. We report these three phases (Section 5.3) on a Pixel 10 Pro (Android 16) with an octa-core, tri-cluster CPU (1×3.78 GHz Cortex-X4, 5×3.05 GHz Cortex-A725, and 2×2.25 GHz Cortex-A520) and 16 GB of RAM. Since user identification and authentication are latency-sensitive, the OS routes them to the primary performance core, so we pin our measurements to that core. Note that while some operations may benefit more than others from parallel computation, the libraries we considered do not properly support multi-core on smartphone architectures, so pinning to the primary core is the most realistic choice.

Methodology. We used `criterion-rs` [1] for Status Manager-side operations, but running Criterion directly on the phone may introduce additional latency: sustained load triggers thermal throttling that may skew the results. So for the mobile-side phases we use a small custom runner instead: 200 repetitions per operation, temperature monitored throughout, with a pause inserted whenever it climbs too high. For longfellow-zk specifically we run fewer repetitions, spaced out with a delay between them, under Google's own C++ Benchmark framework [27] rather than our runner.

Security level. KB21, BBS, and Bulletproofs++ are all based on the widely used BLS12-381 curve, generally chosen to target a security level of 128 bits, with an embedding degree of 12 over a 381-bit prime field [44]. For a broadly comparable setting, we instantiate the RSA accumulator with a 3072-bit modulus. Longfellow targets a similar security level as well.

Input Parameters. We selected the parameters for data collection based on publicly available data from the Italian eID, CIE [36]. This lets us ground the comparison against real data from a live, national eID scheme, since no comparable data exists yet for a credential ecosystem like the EUDI Wallet. On average, a CIE CRL is 20 MB in size and contains approximately 440,000 revoked certificates; on a daily basis, 19,178 new certificates are issued, and 548 are revoked. We sweep a range of batch sizes in powers of two to show how these mechanisms scale, specifically $a \in \{1, 1024, 4096, 16384, 65536\}$ additions and $r \in \{1, 8, 64, 512, 4096\}$ revocations per epoch, which brackets the CIE's daily averages.

For signed-pairs, Revocation also depends on how many revoked handles already exist in the list, since the Status Manager must re-sign every one of them, not just the batch being revoked in the current epoch. We fix the size of the starting list at 10^5 revoked handles for every Revocation and Bandwidth measurement in the paper. Re-signing cost scales linearly with the number of gaps (Section 5.2), so results at other list sizes can be obtained by simple proportion without re-benchmarking. Additionally, a way to reduce the number of signed-pairs within the public revocation state is to use credentials with a limited time validity, so that the manager must sign only the handles of revoked credentials whose validity has not expired. For instance, if we consider a revocation rate of 548 credentials a day, and credentials whose validity is 6 months, we obtain a public revocation state with approximately 10^5 signatures. Regarding the size of the revocation handle, we assume a length of 4 B, which is more than enough to count the number of revocations typically included in a CRL. Except for CL-RSA-B, where the domain is composed exclusively of prime numbers, and for ECDSA-LF, as a consequence of the circuit design, where we set this value to 32 B.

5.2 Manager-Side Costs: Setup, Add, Revocation

Table 1 and Figure 1 give the overall picture across all six phases: what the Status Manager pays for (i–iii), and what the Holder and Verifier pay for on a phone (iv–vi).

[16] https://github.com/aleph-fbk/anonymous-revocation-benchmark

Table 1: Median latency (ms) by mechanism and phase. Setup, Add, and Revocation (i–iii) are Status Manager-side, measured on the laptop; Holder Update, Presentation, and Verification (iv–vi) are Holder-/Verifier-side, measured on mobile. Revocation and Holder Update are measured for the number of revocations $r = 512$ and $r = 4096$, the grid points bracketing the CIE's real daily average of 548 revoked; as it is crucial for signed-pairs measurement of the Revoke operation, the number R of already revoked certificates is set to 10^5; Add considers as number of additions $a = 1024$ and $a = 16384$, the largest being the closest grid point below the CIE's real daily average of 19178 new certificates issued. Setup, Presentation, and Verification don't depend on batch size.

	Setup	Add		Revoke		Holder Update		Present	Verify
Mechanism		$a = 1024$	$a = 16384$	$r = 512$	$r = 4096$	$r = 512$	$r = 4096$		
KB21	1.52	1.32×10^2	2.06×10^3	1.28×10^1	4.99×10^2	4.79×10^1	8.85×10^1	4.75	1.05×10^1
CL-RSA-B	6.55×10^3	4.63×10^3	7.40×10^4	1.38×10^1	1.39×10^2	1.97×10^2	1.97×10^3	8.93×10^1	8.14×10^1
BBS-BP	3.99	–	–	4.85×10^4	5.01×10^4	9.71	9.68	8.16×10^1	4.11×10^1
ECDSA-LF	8.70×10^{-2}	–	–	1.03×10^4	1.06×10^4	2.03	1.91	1.58×10^2	9.42×10^1

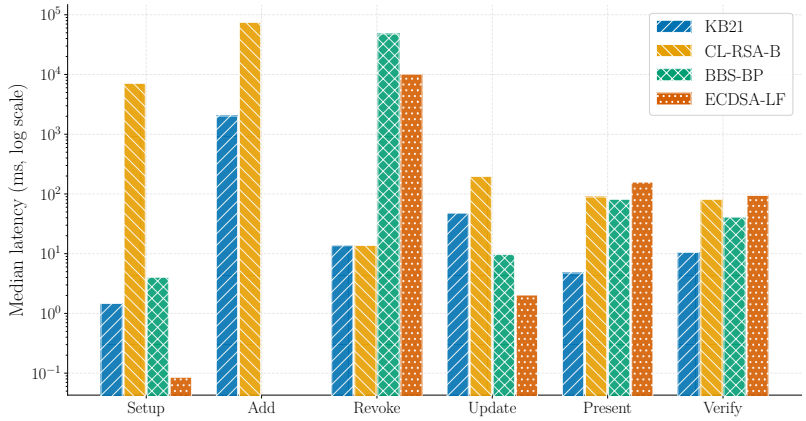


Figure 1: Median latency of all six lifecycle phases, grouped by mechanism, on a logarithmic scale. Setup, Add, and Revocation (i–iii) are Status Manager-side, measured on the laptop; Add and Revocation show 16,384 new certificates issued and 512 revoked, respectively. The number R of already revoked certificates, crucial for signed-pairs measurement of the Revoke operation, is set to 10^5. Holder Update, Presentation, and Verification (iv–vi) are on the Pixel 10 Pro; Holder Update shows 512 revoked.

Phase (i): Setup. Except for CL-RSA-B, where generating a large prime dominates the cost, every mechanism is cheap here: it is just a handful of parameters generated once, at initialization.

Phase (ii): Add. For the accumulators, adding a credential costs a little on the Status Manager, who adds the new element and issues its witness (the revocation material). This cost grows with the batch size a, at the largest batch, $a = 65{,}536$, CL-RSA-B takes about 296 s against KB21's 8.2 s – about 37× slower. The gap already shows up at realistic volumes: the CIE issues about 19,178 new certificates a day, close to our $a = 16{,}384$ grid point, where KB21 finishes in 2.06 s and CL-RSA-B needs 74 s. signed-pairs, in contrast, has no Add cost at all: issuing a credential does not require any computation from the Status Manager. We leave it out of Table 1, which only cover the batch-scaling accumulators.

Phase (iii): Revocation. For accumulators, the Status Manager computes an update message, *upmsg*, at each epoch. This cost grows with the number of revocations, and is fairly cheap for both EC and RSA accumulators. For signed-pairs, the Status Manager must re-sign every surviving gap, every epoch. As explained in Section 5.1, we fix the starting list at 10^5 revocations, but since the cost is linear in the number of signatures the Status Manager must generate, the performance can be easily adapted to other initial list sizes. Between the two signed-pairs instantiations, BBS-BP is the more expensive one: about 4.7× ECDSA-LF. At realistic daily volumes, signed-pairs' cost comes almost entirely from the list's existing size, not that day's new revocations, and even for lists with 10^5 elements, these exceed by far the computational complexity of accumulators. Our 10^5 revoked elements list can be seen as a conservative choice: the CIE's CRL already holds about 4.4×10^5 revoked certificates, 4.4× our benchmarked list.

5.3 Holder-Side Costs: Update, Presentation and Verification

Phases (iv)–(vi) numbers in this section are from the Pixel 10 Pro (see Table 1 above).

Phase (iv): Holder Update. A KB21 Holder applies some cheap scalar multiplication per deleted element, so its cost grows with the number of revocations: from 2.49 ms at $r = 1$ up to 88.47 ms at $r = 4096$. A CL-RSA-B Holder instead runs the Extended Euclidean Algorithm with input its own element and the whole batch product in order to compute the Bézout coefficients b_i, c_i. This is a much heavier operation whose cost also grows with the batch size. This

makes CL-RSA-B the slowest of the four mechanisms at this phase: 1,965 ms at $r = 4096$, against KB21's 88.47 ms.

The signed-pairs Holder does almost no work here: it scans a list for the gap that contains its handle, which is a handful of integer comparisons, and verifies one signature. This cost changes little with r: about 9.7 ms for BBS-BP and about 2 ms for ECDSA-LF, regardless of the batch size. At the CIE's daily rate of 548 revocations, both signed-pairs mechanisms beat both accumulators: ECDSA-LF (2.03 ms) and BBS-BP (9.71 ms) are faster than KB21 (47.87 ms) and much faster than CL-RSA-B (197 ms).

So far, we have compared the latency of the operations between different revocation schemes, now we also consider the communication cost for the Holder who must fetch the update public revocation state. A summary for the size comparison can be found in Table 3. For signed-pairs, during each epoch, the Holder must download the entire list. In practice, this size is dominated by the initial size of the list, not by the number r of revocations in the latest epoch. Each entry carries a signature over the gap (for BBS: one point plus one handle, $48 + 32 = 80$ B; for ECDSA: a pair of scalars, $32 + 32 = 64$ B) plus the signed messages. For ECDSA-LF these bounds are pinned by longfellow-zk's actual signed span format: the circuit hardcodes the handles as 256-bit values, so each bound costs 32 B regardless of how many bits the identifier actually needs. For BBS-BP there is no such implemented format, we only need to cover the range of identifiers actually in use, which at the scale of the CIE's population (and with room to spare) fits in 32 bits, so we size each bound at 4 B. That gives 64 B for the signed elements for ECDSA-LF and 8 B for BBS-BP. Summing the signature size, this yields 128 B per gap for ECDSA-LF and 88 B per gap for BBS-BP. This means that our 10^5-gap list already weighs 12.8 MB for ECDSA-LF and 8.8 MB for BBS-BP. Scaling this to the CIE's real list of 440,000 certificates gives 56.3 MB and 38.7 MB, a 2.8× and 1.9× increase over the CIE's actual 20 MB CRL. The reason for that increase is simple: a standard revocation list needs only a compact ∼48 B entry (a DER-encoded serial number and revocation date, with minimal ASN.1 overhead), while a signed-pairs gap entry must also carry a full signature.

If for accumulators we are happy to let the Holders download the updates to the public revocation state corresponding to the latest epoch (see Section 4.1), KB21's and CL-RSA-B's respective *upmsg* stay far smaller than either full list, because they only carry that epoch's new revocations, not the whole history. But between the two accumulators, KB21 is consistently the larger of the two: its per-element cost is 48 + 4 B (new accumulator and relative handle) against CL-RSA-B's 32 B (only the handle), so KB21's message is almost twice as long as CL-RSA-B's. Note that in this computation we are not considering the new value of the accumulator for CL-RSA-B (384 B), since for the Holders to update they don't need the intermediate values, only the last one.

If with signed-pairs we enable Holders to download only their relevant signature, accepting the privacy implications discussed in Section 4.1, this clearly drastically reduces the bandwidth cost.

Phase (v): Presentation. Presentation is the phase that most needs to be fast, since this is when a user is actively waiting to access a service. Proof size also matters, and it does not depend on the device: small presentation size is particularly crucial when presentations happen over a constrained proximity channel, such as NFC or Bluetooth Low Energy, where each kilobyte adds a non-trivial amount of time.

Presentation time. KB21 is fastest at 4.75 ms. BBS-BP follows at 81.60 ms (17.2× KB21), then CL-RSA-B at 89.26 ms (18.8×), and ECDSA-LF is slowest at 157.85 ms (33.2×). One caveat specific to ECDSA-LF: we do not count the circuit-compilation step, since it is a one-time cost that can be cached instead of repeated at every presentation. On the phone, this step takes about 1.26 s, or about 6.9 MiB of precomputed material to cache to disk; we use the cached version, in line with the one-time-cost point above.

Proof size. KB21 produces the smallest proof, 0.208 KiB, with BBS-BP behind at 1.78 KiB. CL-RSA-B is bigger, at 4.84 KiB, from the structure of its RSA membership witness. ECDSA-LF is far larger than all of them at 202.5 KiB – about 1010× KB21's proof and 114× BBS-BP's.

This means that, even an NFC reader with that reads 50 KiB/s [29], downloading an ECDSA-LF proof would take more than 4 seconds.

Phase (vi): Verification. Verification is cheaper than presentation for three of the four mechanisms, but KB21 is the exception. BBS-BP's verification costs 50% of its own proving time, ECDSA-LF's costs 60%, CL-RSA-B's costs 91% – but KB21's verification actually costs *more* than its own presentation, 221%.

Remark 2. *A few more details on the ZKPs we benchmark. For KB21, the library implements a refined version of the standard proof [46], by proving knowledge of a weak-BB signature [13].*

The BBS/Bulletproofs++ library does not natively support signed-pairs, so we built a custom composite proof. First, we generate a BBS Proof of Knowledge (PoK) for the revocation attestation, which reveals the epoch time t. This comes with a proof of equality to the message committed in Pedersen commitments C_i and C_{i+1}, committing to the revocation handles h_i and h_{i+1}, and with a commitment C_h to the Holder's revocation handle h that would be proven equal to the revocation handle included in the presented credential.

Second, we apply Bulletproofs++ for the range checks: we prove $h_i < h$ over $D_1 = C_h - C_i - G$, and $h < h_{i+1}$ over $D_2 = C_{i+1} - C_h - G$. Since h_i and h_{i+1} stay hidden inside C_i and C_{i+1}, they are never shown to the Verifier; the homomorphic properties of D_1 and D_2 let us connect these hidden commitments directly to the Bulletproofs++ inputs.

For ECDSA-LF, the same kind of binding holds for the bounds: the circuit slices h_i and h_{i+1} directly out of the SHA-256 preimage that also produces the signed challenge, and checks the reconstructed challenge against the one fed to the ECDSA Verifier. So h_i and h_{i+1} are provably the values that were actually signed.

5.4 Experimental Assessment Summary

The clearest pattern on the Status Manager side (Section 5.2) is where each mechanism puts its cost. Accumulators pay at issuance: Add grows with the number of new members and can get expensive, especially for CL-RSA-B's independent per-witness modular exponentiation (74 s at the CIE's real daily add rate, vs. KB21's 2.06 s), while Revocation stays cheap for both (499 ms KB21, 139 ms CL-RSA-B even at the largest batch we test). signed-pairs is the mirror image: Add is essentially free, but Revocation dominates everything

Table 2: Summary comparison of the four revocation mechanisms across protocol-level properties (Section 4.1).

Criterion	KB21	CL-RSA-B	BBS-BP	ECDSA-LF
Public witness generation	○	○	●	●
Manager update cost independent of revocation history[a]	●	●	○	○
Immediate reinstatement	●	●	◐	◐
Manager Hardware support	○	◐	○	●
Privacy under partial-state fetch	◐	◐	○	○
Proximity channel (e.g. NFC) practicality	●	●	●	◐

[a] Log-based vs. state-based update, as defined in Sec. 3.
●Fully satisfied ◐Partially satisfied ○Not satisfied.

Table 3: Communication complexity, measured in KiB, depends on the size of the proof that is sent to the Verifier and the size of the public revocation state that the Holder must download. In the table, we describe the three kinds of downloads the Holder can perform, that progressively degrade its privacy, namely the whole revocation status, the difference between the revocation status at epoch $t-1$ and at the current epoch t (Δ Update Message), and only its witness (if provided by the Status Manager). Here, R denotes the number of revocations executed by the Status Manager until epoch $t-1$, and r the revocations in the last epoch t. We provide concrete numbers for $R = 10^5$, $r = 512$, and $t = 195 \approx \frac{10^5}{512}$.

Mechanism	Proof size	Public revocation state size	Δ Update Message size	Witness size
KB21	2.08×10^{-1}	$5.2 \times 10^{-2}(R+r)$	$5.2 \times 10^{-2} r$	4.80×10^{-2}
($R = 10^5$, $r = 512$)		5.23×10^{3}	2.66×10^{1}	
CL-RSA-B	4.84	$3.84 \times 10^{-1} t + 3.2 \times 10^{-2}(R+r)$	$3.84 \times 10^{-1} + 3.2 \times 10^{-2} r$	3.84×10^{-1}
($R = 10^5$, $r = 512$, $t = 195$)		3.29×10^{3}	1.68×10^{1}	
BBS-BP	1.78	$8.80 \times 10^{-2}(R+r)$	$8.80 \times 10^{-2}(R+r)$	8.80×10^{-2}
($R = 10^5$, $r = 512$)		8.85×10^{3}	8.85×10^{3}	
ECDSA-LF	2.03×10^{2}	$1.28 \times 10^{-1}(R+r)$	$1.28 \times 10^{-1}(R+r)$	1.28×10^{-1}
($R = 10^5$, $r = 512$)		1.29×10^{4}	1.29×10^{4}	

else in this comparison – the Status Manager must re-sign the entire steady-state list every epoch, not just that day's new revocations. At the CIE's real list length this reaches an estimated 210 s (BBS-BP) and 44 s (ECDSA-LF), by far the largest cost anywhere in either table.

Within the accumulators, KB21 is cheaper than CL-RSA-B almost everywhere: setup, Holder witness update, and the ZKP, both generation and verification. The one exception is size: CL-RSA-B's *upmsg* is smaller than KB21's per element, even though applying it costs the Holder far more work.

Between the two families, KB21 and signed-pairs each win on a different axis. KB21 is faster at presentation – the phase a user actually waits on – beating ECDSA-LF by 33.2×. Signed-pairs, in exchange, keeps the Holder update to a lightweight scan and a single signature check: at the CIE's daily revocation rate ($r = 512$), KB21's Holder update is 4.9× BBS-BP's, and that gap only widens at larger batches – 9.1× at $r = 4096$. Neither family dominates once the full lifecycle is considered.

6 Conclusion

Our work places both revocation families currently considered for the EUDI wallet under one syntax, and provides a clear and comprehensive comparison. Neither family dominates on all measures, and the concrete costs separate by orders of magnitude rather than by constant factors (see Figure 1). We summarize the results of our comparison, which includes an experimental evaluation of the schemes in Table 1 a protocol-level comparison in Table 2, and the communication complexity reported in Table 3. Which of the properties of the schemes matters depends on the deployment, and our unified description and benchmarks are intended to let that choice be made on evidence. The comparison also shows where the current constructions fall short of what a deployment at this scale will demand — in the bandwidth a Holder spends to stay synchronised, and in the concrete security margins that erode as query counts grow — and we regard closing these gaps as promising directions for future work.

Acknowledgments

This work has been partially supported by the joint laboratory between the Bruno Kessler Foundation (FBK) and the Italian Government Printing Office and Mint (IPZS), and by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU. Work of A.F. and A.L. was partially funded by SPRIND, the Federal Agency for Breakthrough Innovation, and Common Codes. The authors are solely responsible for the content of this publication; the positions presented here do not reflect the views of SPRIND and Common Codes.

References

[1] 2026. Criterion.rs. https://bheisler.github.io/criterion.rs/book/. Retrieved 2026-07-16 from https://bheisler.github.io/criterion.rs/book/

[2] 2026. The GNU Multiple Precision Arithmetic Library. https://gmplib.org/. Retrieved 2026-07-16 from https://gmplib.org/

[3] 2026. unknown_order. https://crates.io/crates/unknown_order. Retrieved 2026-07-16 from https://crates.io/crates/unknown_order

[4] Man Ho Au, Willy Susilo, and Yi Mu. 2006. Constant-Size Dynamic k-TAA. In *Security and Cryptography for Networks*, Roberto De Prisco and Moti Yung (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 111–125.

[5] Man Ho Au, Patrick P. Tsang, Willy Susilo, and Yi Mu. 2009. Dynamic Universal Accumulators for DDH Groups and Their Application to Attribute-Based Anonymous Credential Systems. In *CT-RSA*, Marc Fischlin (Ed.). Springer, 295–308. doi:10.1007/978-3-642-00862-7_20

[6] Foteini Baldimtsi, Jan Camenisch, Maria Dubovitskaya, Anna Lysyanskaya, Leonid Reyzin, Kai Samelin, and Sophia Yakoubov. 2017. Accumulators with Applications to Anonymity-Preserving Revocation. In *EuroS&P*. IEEE, 301–315. doi:10.1109/EUROSP.2017.13

[7] Niko Barić and Birgit Pfitzmann. 1997. Collision-Free Accumulators and Fail-Stop Signature Schemes Without Trees. In *EUROCRYPT '97*, Walter Fumy (Ed.). Springer, 480–494. doi:10.1007/3-540-69053-0_33

[8] Josh Benaloh and Michael de Mare. 1994. One-way accumulators: A decentralized alternative to digital signatures. In *EUROCRYPT '93*, Tor Helleseth (Ed.). Springer, 274–285. doi:10.1007/3-540-48285-7_24

[9] Patrik Bichsel, Jan Camenisch, Maria Dubovitskaya, R Enderlein, Stephan Krenn, Ioannis Krontiris, Anja Lehmann, Gregory Neven, Janus Dam Nielsen, Christian Paquin, et al. 2014. D2. 2 Architecture for attribute-based credential technologies-final version. *ABC4TRUST project deliverable. Available online at https://abc4trust. eu/index. php/pub* (2014).

[10] Dan Boneh and Hovav Shacham. 2004. Group signatures with Verifier-local revocation. In *Proceedings of the 11th ACM conference on Computer and communications security*. 168–177.

[11] Dan Boneh and Victor Shoup. 2023. A graduate course in applied cryptography. *Draft 0.6* (2023). https://toc.cryptobook.us/

[12] Christian Bormann and Anja Lehmann. 2026. SoK: Anonymous Credentials for Digital Identity Wallets. Cryptology ePrint Archive, Paper 2026/330. https://eprint.iacr.org/2026/330

[13] Jan Camenisch, Manu Drijvers, and Jan Hajny. 2016. Scalable revocation scheme for anonymous credentials based on n-times unlinkable proofs. In *Proceedings of the 2016 ACM on Workshop on Privacy in the Electronic Society*. 123–133.

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

[15] Jan Camenisch and Anna Lysyanskaya. 2002. Dynamic Accumulators and Application to Efficient Revocation of Anonymous Credentials. In *CRYPTO 2002 (Lecture Notes in Computer Science, Vol. 2442)*, Moti Yung (Ed.). Springer, 61–76. doi:10.1007/3-540-45708-9_5

[16] Ivan Damgård and Eiichiro Fujisaki. 2002. A Statistically-Hiding Integer Commitment Scheme Based on Groups with Hidden Order. In *ASIACRYPT 2002*, Yuliang Zheng (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 125–142. doi:10.1007/3-540-36178-2_8

[17] Ivan Damgård and Nikos Triandopoulos. 2008. Supporting Non-membership Proofs with Bilinear-map Accumulators. *IACR Cryptol. ePrint Arch.* (2008). http://ia.cr/2008/538

[18] Dock.io. 2026. docknetwork/crypto. https://github.com/docknetwork/crypto. Retrieved 2026-07-16 from https://github.com/docknetwork/crypto

[19] Liam Eagen, Sanket Kanjalkar, Tim Ruffing, and Jonas Nick. 2024. Bulletproofs++: Next generation confidential transactions via reciprocal set membership arguments. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 249–279.

[20] European Commission. 2026. Specification for the implementation of Zero-Knowledge Proofs based on arithmetic circuits in the EUDI Wallet. https://github.com/eu-digital-identity-wallet/eudi-doc-standards-and-technical-specifications/blob/3cd8a8100a0f3be545d9412e5ffc985deba35cb3/docs/technical-specifications/ts13-zksnarks.md. Version 1.0.1.

[21] European Commission. 2026. Specification for the Implementation of Zero-Knowledge Proofs Based on Multi-Message Signatures in the EUDI Wallet. https://github.com/eu-digital-identity-wallet/eudi-doc-standards-and-technical-specifications/blob/3cd8a8100a0f3be545d9412e5ffc985deba35cb3/docs/technical-specifications/ts14-zkps-from-mms.md. Version 1.0.

[22] Amos Fiat and Adi Shamir. 1987. How To Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Advances in Cryptology — CRYPTO' 86*, Andrew M. Odlyzko (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 186–194.

[23] Andrea Flamini, Silvio Ranise, Giada Sciarretta, Mario Scuro, Nicola Smaniotto, and Alessandro Tomasi. 2025. Public key accumulators for revocation of non-anonymous credentials. *Cryptology ePrint Archive* (2025).

[24] Andrea Flamini, Giada Sciarretta, Mario Scuro, Amir Sharif, Alessandro Tomasi, and Silvio Ranise. 2024. On cryptographic mechanisms for the selective disclosure of verifiable credentials. *Journal of Information Security and Applications* 83 (2024), 103789. doi:10.1016/j.jisa.2024.103789

[25] Matteo Frigo and abhi shelat. 2024. Anonymous credentials from ECDSA. Cryptology ePrint Archive, Paper 2024/2010. https://eprint.iacr.org/2024/2010

[26] Riccardo Germenia, Salvatore Manfredi, Giada Sciarretta, Mario Scuro, and Alessandro Tomasi. 2025. Comparison of Credential Status Mechanisms for the Digital Wallet Ecosystem. In *22nd International Conference on Security and Cryptography (SECRYPT)*. doi:10.13140/RG.2.2.24161.01128

[27] Google. 2026. Google Benchmark. https://github.com/google/benchmark. Retrieved 2026-07-16 from https://github.com/google/benchmark

[28] Google. 2026. longfellow-zk. https://github.com/google/longfellow-zk. Retrieved 2026-07-16 from https://github.com/google/longfellow-zk

[29] ISO/IEC. 2023. *Information technology – Telecommunications and information exchange between systems – Near Field Communication – Interface and Protocol (NFCIP-1)*. Standard ISO/IEC 18092:2023. International Organization for Standardization / International Electrotechnical Commission, Geneva, Switzerland.

[30] Samuel Jaques, Michael Lodder, and Hart Montgomery. 2022. ALLOSAUR: Accumulator with Low-Latency Oblivious Sublinear Anonymous credential Updates with Revocations. *IACR Cryptol. ePrint Arch.* (2022), 1362. https://ia.cr/2022/1362

[31] Ioanna Karantaidou and Foteini Baldimtsi. 2021. Efficient Constructions of Pairing Based Accumulators. In *CSF*. IEEE, 1–16. doi:10.1109/CSF51468.2021.00033

[32] Amrit Kumar, Pascal Lafourcade, and Cédric Lauradoux. 2014. Performances of cryptographic accumulators. In *39th Annual IEEE Conference on Local Computer Networks*. 366–369. doi:10.1109/LCN.2014.6925793

[33] Anja Lehmann, Andrey Sidorenko, and Alexandros Zacharakis. 2026. Vision: A Modular Framework for Anonymous Credential Systems. In *Security Standardisation Research*, Henrich C. Pöhls and Chris J. Mitchell (Eds.). Springer Nature Switzerland, Cham, 46–68.

[34] Jiangtao Li, Ninghui Li, and Rui Xue. 2007. Universal Accumulators with Efficient Nonmembership Proofs. In *Applied Cryptography and Network Security*, Jonathan Katz and Moti Yung (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 253–269. doi:10.1007/978-3-540-72738-5_17

[35] Matteo Loporchio, Anna Bernasconi, Damiano Di Francesco Maesa, and Laura Ricci. 2023. A survey of set accumulators for blockchain systems. *Computer Science Review* 49 (2023), 100570.

[36] Ministero dell'Interno. [n. d.]. CIE Certification Authority. Retrieved 2024-10-11 from https://www.cartaidentita.interno.gov.it/en/public-and-business-administration/certification-autority/

[37] Toru Nakanishi, Hiroki Fujii, Yuta Hira, and Nobuo Funabiki. 2010. Revocable Group Signature Schemes with Constant Costs for Signing and Verifying. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* E93-A, 1 (jan 2010), 50–62. doi:10.1587/transfun.E93.A.50

[38] Lan Nguyen. 2005. Accumulators from Bilinear Pairings and Applications. In *CT-RSA 2005*, Alfred Menezes (Ed.). Springer, 275–292. doi:10.1007/978-3-540-30574-3_19

[39] Lan Nguyen and Tolga Acar. 2013. Accumulators and U-Prove Revocation. In *Financial Cryptography'13 (LNCS, Vol. 7859)*. Springer Verlag, 189–196. https://www.microsoft.com/en-us/research/publication/accumulators-and-u-prove-revocation/

[40] OpenSSL Project. 2026. OpenSSL BN – Multi-precision Integer Arithmetics. https://www.openssl.org/docs/man1.0.2/man3/bn.html. Retrieved 2026-07-16 from https://www.openssl.org/docs/man1.0.2/man3/bn.html

[41] Torben Pryds Pedersen. 1991. Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual international cryptology conference*. Springer, 129–140.

[42] Nicholas Pippenger. 1980. On the Evaluation of Powers and Monomials. *SIAM J. Comput.* 9, 2 (1980), 230–250. arXiv:https://doi.org/10.1137/0209022 doi:10.1137/0209022

[43] Sietse Ringers. 2023. Using the RSA or RSA-B accumulator in anonymous credential schemes. *IACR Cryptol. ePrint Arch.* (2023), 11. https://ia.cr/2023/011

[44] Yumi Sakemi, Tetsutaro Kobayashi, Tsunekazu Saito, and Riad S. Wahby. 2022. *Pairing-Friendly Curves*. Internet-Draft draft-irtf-cfrg-pairing-friendly-curves-11. Internet Engineering Task Force. Work in Progress. https://datatracker.ietf.org/doc/draft-irtf-cfrg-pairing-friendly-curves/11/

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

[46] Giuseppe Vitto and Alex Biryukov. 2022. Dynamic Universal Accumulator with Batch Update over Bilinear Groups. In *CT-RSA (Lecture Notes in Computer Science, Vol. 13161)*, Steven D. Galbraith (Ed.). Springer, 395–426. doi:10.1007/978-3-030-95312-6_17

$h \in D$	revocation handle, from the scheme's domain
t	epoch index
V_t	public revocation state value at epoch t
$upmsg_t$	public update log at epoch t
$W_{h\|t}$	Holder's revocation material for h at epoch t
mpk, msk	Status Manager's public and secret key
C	commitment to the revocation handle h
J, $r = \|J\|$	handles revoked in an epoch, and their number

Table 4: Notation used throughout the paper.

A Notation

We report in Table 4 the notation we use throughout the paper.

B Deep-Dive: Accumulators

B.1 Security Notions for Accumulators

Security notions for accumulators. There are two main security notions associated to positive dynamic accumulators, namely *adaptive soundness* and *non-adaptive soundness*. Informally, an accumulator scheme is *adaptively sound* if no polynomial time adversary $\mathcal{A}$ can win the following experiment. $\mathcal{A}$ takes in input an accumulator value V, and can perform a training where it sends a polynomial number of queries to the challenger to add or delete elements of its choice, choosing them adaptively. Let V be the value of the accumulator after the training: we say that $\mathcal{A}$ wins the experiment if it can generate a pair (h, W) such that $\mathsf{VerMem}(V, h, W) = 1$ and h is not accumulated. An accumulator is *non-adaptively sound* if the adversary chooses in advance the set of elements $\mathcal{S}$ to add and remove in its queries, before the publication of the accumulator V. Then the addition and deletions performed during the training by the adversary are chosen adaptively but must come from $\mathcal{S}$. For a formal security definition of adaptive and non-adaptive soundness see [6, Section 2.2].

B.2 Zero Knowledge Proofs of Membership for Accumulators

In this section, we describe the ZKP protocols that allow a Holder to prove knowledge of an element-witness pair (h, W), where h is committed in C.

We present these protocols for the RSA and EC accumulators described respectively in Sections 3.1.1 and 3.1.2. Our construction build on the standard proof of knowledge of an accumulated element and its witness: the RSA-based proof first introduced in [15] and revisited in [43] (with few additional corrections to the generation of C_r), and the EC-based proof from [13] for Boneh-Boyen Signatures.

RSA accumulator.

- $\pi \xleftarrow{\$} \mathsf{GenZKP}(C, h, l, W_{h|t}, V_t, mpk)$. Given the commitment $C \leftarrow G^h H^l$ of h and opening material l compute a proof of knowledge of the pair $(h, W_{h|t})$, such that h is committed in C, and $W_{h|t}$ is a valid witness for h under V_t. This can be computed as follows, according to [43]
  (1) generate random values $r_w, s \xleftarrow{\$} \mathbb{Z}_{\lfloor n/4 \rfloor}$ for the auxiliary Damgård-Fujisaki commitments [16] $C_w = wH^{r_w}, C_r = H^{r_w} G^s$;
  (2) prove knowledge of the pair $(h, W_{h|t})$ computing a proof:

$$\pi' \in PK\Big\{(h, l, r_w, s, hr_w, hs) : C = G^h H^l \wedge C_r = H^{r_w} G^s \wedge V = C_w^h \left(\frac{1}{H}\right)^{hr_w} \wedge 1 = C_r^h \left(\frac{1}{H}\right)^{hr_w} \left(\frac{1}{G}\right)^{xs}\Big\} \tag{6}$$

  (3) output the proof $\pi = (C_w, C_r, \pi')$.
- $0/1 \leftarrow \mathsf{VerZKP}(\pi, C, V_t, mpk)$

EC accumulator. This ZKP is presented in [13, Section 2.4] as a ZKP for Boneh-Boyen signatures, but can be adapted also to prove knowledge of an accumulated element and its witness for a KB21 accumulator.

- $\pi \xleftarrow{\$} \mathsf{GenZKP}(C, h, l, W_{h|t}, V_t, mpk)$. Given the commitment $C \leftarrow G^h H^l$ of h and opening material l compute a proof of knowledge of the pair $(h, W_{h|t})$, such that h is committed in C, and $W_{h|t}$ is a valid witness for h under V_t. This can be done sampling $r \xleftarrow{\$} \mathbb{Z}_p^*$ and setting $\bar{A} \leftarrow W_{h|t}^r$, then the prover sets $\bar{B} \leftarrow \bar{A}^{-h} V_t^r$ and computes

$$\pi' \in PK\Big\{(h, l, r) : C = G^h H^l \wedge \bar{B} = \bar{A}^{-h} V_t^r\Big\} \tag{7}$$

  Output the proof $\pi = (\bar{A}, \bar{B}, \pi')$.
- $0/1 \leftarrow \mathsf{VerZKP}(\pi, C, V_t, mpk)$. Parse $\pi \rightarrow (\bar{A}, \bar{B}, \pi')$, verify π' and check that $\mathbf{e}(\bar{B}, \tilde{G}_2) = \mathbf{e}(\bar{A}, mpk)$.