2017 IEEE European Symposium on Security and Privacy

Accumulators with Applications to Anonymity-Preserving Revocation*

Foteini Baldimtsi[‖], Jan Camenisch[¶], Maria Dubovitskaya[¶], Anna Lysyanskaya[‡], Leonid Reyzin[§], Kai Samelin[¶,**], Sophia Yakoubov[§]
[‖] *George Mason University, Fairfax, USA*
foteini@gmu.edu
[¶] *IBM Research – Zurich, Rüschlikon, Switzerland*
{*jca, mdu, ksa*}*@zurich.ibm.com*
[‡] *Brown University, Providence, Rhode Island, USA*
anna@cs.brown.edu
[§]*Boston University, Boston, USA*
{*reyzin, sonka*}*@bu.edu*
[**] *TU Darmstadt, Darmstadt, Germany*

***Abstract*—Membership revocation is essential for cryptographic applications, from traditional PKIs to group signatures and anonymous credentials. Of the various solutions for the revocation problem that have been explored, dynamic accumulators are one of the most promising. We propose Braavos, a new, RSA-based, dynamic accumulator. It has optimal communication complexity and, when combined with efficient zero-knowledge proofs, provides an ideal solution for anonymous revocation. For the construction of Braavos we use a modular approach: we show how to build an accumulator with better functionality and security from accumulators with fewer features and weaker security guarantees. We then describe an anonymous revocation component (ARC) that can be instantiated using any dynamic accumulator. ARC can be added to any anonymous system, such as anonymous credentials or group signatures, in order to equip it with a revocation functionality. Finally, we implement ARC with Braavos and plug it into Idemix, the leading implementation of anonymous credentials. This work resolves, for the first time, the problem of practical revocation for anonymous credential systems.**

1. Introduction

Authentication of users is vital to most of the electronic systems we use today. It is usually achieved by giving the user a token, or *credential*, that the user must present to prove that she has permission to access a service. An important challenge that such systems face is how to *revoke* a user's privileges in case she misbehaves or her credential gets compromised.

Achieving revocation in practice has been shown to be a very complex problem. The two obvious approaches are *whitelists*, where a user is valid if her public key or identity is on a whitelist, and *blacklists*, where anyone not on the blacklist can be presumed to be a valid user. Both of these solutions are problematic, because the party that manages revocation (hereafter called the *revocation authority*) needs to distribute large lists and update them continuously.

In anonymous settings, where user authentication must not reveal the user's identity, things get even more complicated. In order for a user to show that she hasn't been revoked, she must prove that she is on the whitelist (or is not on the blacklist) in *zero knowledge*, which requires work linear in the size of the list. Having to perform linear work in the number of whitelisted (or blacklisted) users is far from practical; thus, we look into solutions where the users and verifiers in charge of authentication only have to do a *constant* amount of work.

One of the most promising solutions to revocation in anonymous settings has been the use of *cryptographic accumulators* [2], [3], [4]. An accumulator is a binding (but not necessarily hiding) commitment to a set S of elements. This set can change over time, as elements are added and deleted; accumulators that support both additions and deletions are called *dynamic*. Known dynamic accumulators are based on Merkle hash trees [5], variants of RSA [6], [2], [7], and bilinear maps [3], [8], [9]. Upon the addition of a new element x to the set, a *witness* w, or *proof of membership*, is generated for the element. This witness is later used, in conjunction with the accumulator value, to demonstrate that the element is, in fact, in the set. The witnesses may evolve as the accumulated set changes.

A dynamic accumulator can be used as an anonymous revocation mechanism [2], [4]. The accumulator contains the whitelist S of users. The revocation authority then only needs to maintain and distribute a short accumulator value, not an entire large whitelist. When a user is issued a credential, she is given a membership witness w for her element x. While she hasn't been revoked, she can prove in zero-knowledge that her x is in the current accumulator, i.e., that she knows (x, w) such that x corresponds to her identity and w is a valid membership witness for x. The complexity of this proof is independent of the numbers of

** This is an extended abstract. The full version can be found in the IACR ePrint Archive [1].*

DOI 10.1109/EuroSP.2017.13

added and revoked users. When the user is revoked, x is removed from the accumulator.

However, there are two downsides to using existing accumulator constructions for revocation in this way. The first is the high communication cost. Each user has to update her membership witness every time the accumulator value changes, which typically happens with every addition or deletion. In order to give the users the information they need for these updates, the revocation authority has to send a broadcast message to all users. The frequency of these broadcasts is unattractive, especially when constrained devices are used.

The second downside is linkability: users who receive these broadcast messages might be able to link revocations of their fellow users with the corresponding additions. That is, when a revocation occurs, the associated broadcast message might contain information related to a previous addition broadcast message, allowing recipients to determine that the revoked user is the same user who was added at a certain previous point. This join-revoke linkability is a breach in anonymity.

1.1. Our Contributions

In this work, we focus on building accumulators that avoid the two downsides described above, namely those of communication overhead and join-revoke linkability. Our contributions are threefold. First, we provide a new modular definitional view of accumulators. Second, we use this modular view to build a new dynamic accumulator with optimal communication cost and join-revoke unlinkability. Finally, we explain how our proposed accumulator can be used to instantiate an anonymous revocation component (ARC) and showcase that our scheme can provide efficient revocation for anonymous credential systems by implementing it for Idemix [10] (IBM's anonymous credential system).

Compositional Accumulator Framework. In Section 2, we give a systematic view of accumulator properties, describing them in a modular way. In Section 3, we leverage our modular view to show simple ways of obtaining new accumulators with better security by combining accumulators that possess only some of the desired properties. (In the full version of this paper [1], we also show how to obtain accumulators with more functionality - adding and deleting, proving membership and non-membership - by combining accumulators with less functionality.) This view of accumulators also sheds light on trade-offs between functionality and efficiency.

Construction. Utilizing our compositional framework, in Section 4 we introduce a new dynamic accumulator which we call Braavos.[1] All previously known dynamic accumulators with strong security guarantees require that the accumulator and each membership witness be updated *both* every time a new element is added to the accumulator, and every time an element is deleted from the accumulator. In Braavos, the accumulator value and membership witnesses are updated *only* when an element is deleted, achieving optimal communication complexity. In particular, this means that the accumulator manager does not need to broadcast any information to the witness holders when a new element addition occurs, making it practical to support a high element addition rate. Because element addition does not involve the communication of any information to other witness holders, Braavos also ensures that a revocation event is not linkable to a prior addition event, achieving join-revoke unlinkability.

Our Braavos accumulator builds upon the dynamic CL-RSA-B accumulator, which was informally introduced as a brief remark by Camenisch and Lysyanskaya [2]. CL-RSA-B has the communication optimality of Braavos, but has a weaker security guarantee which makes it usable only for the accumulation of random elements. CL-RSA-B is only secure against non-adaptive attacks, as described in Section 2.2.[2] Braavos uses CL-RSA-B to accumulate random elements, and uses digital signatures to bind the random elements to the actual elements to be accumulated. Because neither CL-RSA-B nor digital signatures require witness updates upon additions, Braavos does not either.

Application and Implementation. In Section 5, we describe an anonymous revocation component (ARC) which uses accumulators to enable revocation in a variety of existing (as well as new) anonymous schemes. Our definition of the ARC is an extension of the revocation component described by Camenisch et. al [11] that allows the addition of users after the initial setup, as well as their re-addition after they have been revoked. When Braavos is the accumulator used to instantiate an ARC, members need to update their witnesses only when another member is revoked (but not when another member is added), which is key since the number of revocations in a typical system is much lower than the number of additions. Consider, for instance, PKI certificates. Around 8% of unexpired certificates are revoked [12]. This number rose from 1% in May 2014 because of the Heartbleed vulnerability, and is expected to eventually return to 1%. However, it is vital to be able to revoke this 1%, since the compromised keys can be used to inflict a lot of damage. Moreover, this has to be done without flooding certificate holders with instructions to update their witnesses roughly 100 times more frequently than necessary.

In Section 5.2, we show how one can directly plug an ARC with Braavos into the leading implementation of anonymous credentials (Idemix) [10]) with no modifications on either side. This compatibility with an existing working system resolves, for the first time, the problem of practical revocation for anonymous credentials.

We measure the performance of the resulting system, and demonstrate that, for realistic credential presentation policies, Braavos adds only a $\approx 10\%$ runtime overhead, both for generation and for verification. In more detail, even for reasonably high security parameter sizes (2,048

1. The Faceless Men of Braavos, from the Game of Thrones, are able to assume unrecognizable personas, unlinkable to their other personas.

2. As a side contribution, in Section 4.1 we formalize and prove the security of CL-RSA-B.

bits), credential generation without revocation takes ≈1.6s, while with revocation it takes ≈1.8s. For verification these numbers become even closer: without revocation we need ≈1.7s, while with revocation we need ≈1.8s. Clearly, this is a small price to pay for the benefits of revocation.

2. Definitions: A Modular View of Accumulator Functionality

In this section, we provide a unified view of accumulator properties. We separately consider accumulators that support additions, deletions or both, and accumulators that support membership proofs, non-membership proofs or both. We have a different focus than Derler et al. [13], who also discuss the unification of accumulators. Namely, unlike Derler et al., we emphasize the modularity of our definitions: subsets of the properties we define can be easily combined, resulting in a broader range of accumulator types than previously described. To limit our scope, we only consider accumulator properties pertaining to functionality and soundness in this section.

Various flavors of accumulator functionality definitions have been restated in literature a number of times. We leverage the definitions provided by Reyzin and Yakoubov [14], but reformulate them to offer a more complete view of the accumulator space. We begin by introducing four basic kinds of accumulator primitives.

- *static accumulator*: represents a fixed set.
- *additive accumulator*: supports only additions.
- *subtractive accumulator*: supports only deletions.
- *dynamic accumulator*: supports both additions and deletions (as defined in [2]).

Deletions and additions can, of course, be performed simply by re-instantiating the accumulator with the updated set. This takes at most a polynomial amount of time in the number of element additions or deletions which have been performed up until that point, but is not, in general, practical. A dynamic accumulator should support both additions and deletions in time which is either independent of the number of operations performed altogether, or is sublinear in this number.

We also describe accumulators in terms of the kinds of proofs (membership proofs, non-membership proofs, or both) they support.

- *positive accumulator*: supports membership proofs.
- *negative accumulator*: supports non-membership proofs.
- *universal accumulator [7]*: supports both.

2.1. Accumulator Algorithms

Next, we describe the algorithms used by all of these accumulator primitives. For convenience, Figure 1 enumerates and explains all algorithm input and output parameters.

We consider three types of parties: (1) an accumulator manager, (2) an entity responsible for an element and its corresponding witness (from hereon-out referred to as *witness holder*), and (3) a third party (e.g. a verifier, who is given any relevant witnesses by the witness holder at the time of verification). Parameters which are omitted in some schemes have a bar on top.

The following are algorithms performed by the accumulator manager:

- $\mathsf{Gen}(1^\lambda, S_0) \rightarrow (\overline{sk}, a_0, \overline{m_0})$ instantiates the accumulator manager's secret key sk, the accumulator a_0 (representing the initial set $S_0 \subseteq D$ of elements in the accumulator, where D is the domain of the accumulator), and the auxiliary value m_0 necessary for the maintenance of the accumulator. m_t can be thought of as the accumulator manager's memory or storage at time t. The allowable S_0 sets vary from accumulator to accumulator. There are accumulators that support only $S_0 = \emptyset$; others support any polynomial-size S_0, and yet others support any S_0 that can be expressed as a polynomial number of ranges.
- $\mathsf{Add}(\overline{sk}, a_t, \overline{m_t}, x) \rightarrow (a_{t+1}, \overline{m_{t+1}}, w^x_{t+1}, \mathsf{upmsg}_{t+1})$ (for additive and dynamic accumulators) adds the element $x \in D$ to the accumulator.
- $\mathsf{Del}(\overline{sk}, a_t, \overline{m_t}, x) \rightarrow (a_{t+1}, \overline{m_{t+1}}, \overline{u^x_{t+1}}, \mathsf{upmsg}_{t+1})$ (for subtractive and dynamic accumulators) deletes the element $x \in D$ from the accumulator.

A positive or universal accumulator additionally has the following algorithms:

- $\mathsf{VerMem}(a_t, x, w^x_t) \rightarrow \{0, 1\}$ (executed by any third party) verifies the membership of x in the accumulator using its membership witness w^x_t.
- $\mathsf{MemWitUpOnAdd}(x, w^x_t, \mathsf{upmsg}_{t+1}) \rightarrow w^x_{t+1}$ (executed by a witness holder for additive and dynamic accumulators) updates the membership witness for element x after y is added to the accumulator.

The membership witness update algorithm MemWitUpOnDel (for subtractive and dynamic accumulators) is defined analogously to MemWitUpOnAdd. Notice that the accumulator manager can eliminate the need for these algorithms by sending each witness holder a fresh witness on demand; however, this is unreasonable to ask of the accumulator manager, who might then have to do additional work *per witness holder* per addition or deletion. Instead, the accumulator manager broadcasts a single update message upmsg which witness holders use to bring their witnesses up to date.[3]

If an element x is in S_0, there might be a need to create a membership witness for x independently of the Add algorithm. To this end, there exists a membership witness creation algorithm MemWitCreate. (In this paper, we focus on witness generation during Add, so MemWitCreate will not appear in future sections.)

- $\mathsf{MemWitCreate}(\overline{sk}, \overline{a_t}, \overline{m_t}, x, \overline{(\mathsf{upmsg}_1, \ldots, \mathsf{upmsg}_t)}) \rightarrow w^x_t$ (executed by the accumulator manager or any third party) generates a membership witness w^x_t for x outside the element addition protocol Add.

3. The witness holders can also process the update messages in batches if they choose.

λ:	The security parameter.
D:	The domain of the accumulator (the set of elements that the accumulator can accumulate). Often, D includes all elements (e.g., $\{0,1\}^*$). Sometimes, D is more limited (e.g., primes of a certain size).
sk:	The accumulator manager's secret key or trapdoor. (The corresponding public key, if one exists, is not modeled here as it can be considered to be a part of the accumulator itself.)
t:	A discrete time / operation counter.
a_t:	The accumulator at time t.
m_t:	Any auxiliary values which might be necessary for the maintenance of the accumulator. These are typically held by the accumulator manager. Note that while the accumulator itself should be constant (or at least sub-linear) in size, m may be larger.
S_t:	The set of elements in the accumulated set at time t. Note that S_0 can be instantiated to be different, based on the initial sets supported by the accumulator in question. Most accumulators assume $S_0 = \emptyset$.
x, y:	Elements which might be added to the accumulator.
w_t^x, u_t^x:	The witness that element x is (respectively, is not) in accumulator a_t at time t.
$\mathsf{upmsgwh}_t$:	A broadcast message sent (by the accumulator manager, if one exists) at time t to all witness holders immediately after the accumulator has been updated. This message is meant to enable all witness holders to update the witnesses they hold for consistency with the new accumulator. It will often contain the new accumulator a_t, and the nature of the update itself (e.g., "x has been added and witness w_t^x has been produced"). It may also contain other information.
$\mathsf{upmsgtp}_t$:	A broadcast message sent (by the accumulator manager, if one exists) at time t to all third party verifiers immediately after the accumulator has been updated. This message is meant to provide the third parties with up to date values (the accumulator a_t) to verify proofs of membership or non-membership against. It may also contain other information.
upmsg_t:	All of the information broadcast (by the accumulator manager, if one exists) at time t. $\mathsf{upmsg}_t = (\mathsf{upmsgwh}_t, \mathsf{upmsgtp}_t)$. We often use upmsg instead of upmsgwh and upmsgtp, since these are frequently the same.

Figure 1. Accumulator algorithm input and output parameters.

	Static			Additive			Subtractive			Dynamic		
	Pos	Neg	Uni	Pos	Neg	Uni	Pos	Neg	Uni	Pos	Neg	Uni
Accumulator Manager Algorithms												
Gen	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Add				✓	✓	✓				✓	✓	✓
Del							✓	✓	✓	✓	✓	✓
MemWitCreate	✓		✓	✓		✓	✓		✓	✓		✓
NonMemWitCreate		✓	✓		✓	✓		✓	✓		✓	✓
Witness Holder Algorithms												
MemWitUpOnAdd				✓		✓				✓		✓
MemWitUpOnDel							✓		✓	✓		✓
NonMemWitUpOnAdd					✓	✓					✓	✓
NonMemWitUpOnDel								✓	✓		✓	✓
Third Party Algorithms												
VerMem	✓		✓	✓		✓	✓		✓	✓		✓
VerNonMem		✓	✓		✓	✓		✓	✓		✓	✓

Figure 2. Accumulator Algorithms.

For negative and universal accumulators, the non-membership witness creation algorithm NonMemWitCreate, the non-membership verification algorithm VerNonMem, and the non-membership witness update algorithms NonMemWitUpOnAdd and NonMemWitUpOnDel are defined analogously to MemWitCreate, VerMem, MemWitUpOnAdd, and MemWitUpOnDel, respectively.

The presence or absence of all of these algorithms is simple to infer from the accumulator type. For convenience, Figure 2 describes the algorithms corresponding to each accumulator type.

2.2. Accumulator Security Properties

In this section, we describe accumulator security properties. Though in Section 2.1 we described many different types of accumulators, here we limit ourselves to positive dynamic accumulators, because our focal construction Braavos is positive and dynamic. Additionally, for simplicity, we assume that the initial set S_0 is empty. If desired, our definitions can be extended to other initial sets, and other accumulator types, in a straightforward way.

Naturally, accumulators must be both correct and sound. We refer to Reyzin and Yakoubov [14] for formal definitions of correctness. Informally, in a positive accumulator, *correctness* requires that for every element in the accumulator,

an honest membership witness holder can always prove membership.

In a positive accumulator, *soundness* (also referred to as *security* or *collision-freeness*) requires that for every element not in the accumulator it is infeasible to prove membership. We present two definitions of soundness for positive dynamic accumulators with $S_0 = \emptyset$ (that is, positive dynamic accumulators which start empty): *adaptive* soundness (Definition 1), and *non-adaptive* soundness (Definition 2). Adaptive soundness is the standard definition of accumulator soundness; it is referred to simply as soundness. We introduce non-adaptive soundness as a building block for our new accumulator, Braavos, as described in Section 3. Informally, in non-adaptive soundness, the adversary must commit to his choice of elements to add in advance, whereas in adaptive soundness the adversary can choose these elements on the fly.

Definition 1. *A positive dynamic accumulator is* adaptively sound *(or simply* sound*) if for all security parameters λ, for all stateful probabilistic polynomial-time adversaries $\mathcal{A}$ with black-box access to* Add *and* Del *oracles (which take elements x) on accumulator a, it holds that:*

$$\Pr\left[\begin{array}{l}(\overline{sk}, a_0, \overline{m_0}) \leftarrow \mathsf{Gen}(1^\lambda, \emptyset);\\ (x, w) \leftarrow \mathcal{A}^{\mathsf{Add},\mathsf{Del}}(a_0, \overline{m_0});\\ x \notin S:\\ \mathsf{VerMem}(a, x, w) = 1\end{array}\right] \leq \nu(\lambda)\,,$$

where ν is a negligible function in the security parameter, x is an element that is not currently a member ($x \notin S$, where S started out empty and was updated with every call to Add *and* Del*), and a is the accumulator after the adversary made all of his calls to* Add *and* Del*.*

Our definition assumes that inputs to Add and Del are in D and that Del does nothing when called on an element that is not in the accumulated set. Some external mechanisms must ensure that this is indeed the case, or else soundness is not guaranteed. This notion of soundness is sufficient for many scenarios, including anonymity-preserving revocation described in this paper.

We also define a weaker notion of soundness in which the elements are picked in advance (though the ordering of their additions and deletions may still be adaptive).

Definition 2. *A positive dynamic accumulator is* non-adaptively sound *(NA-sound) if the conditions in Definition 1 hold with the following modification: before* Gen *is run, the adversary $\mathcal{A}$ produces a set of elements $x_1, \ldots, x_q \in D$, and queries to* Add *and* Del *must come from this set.*

This weaker notion of soundness suffices to ensure soundness for randomly chosen elements, because it does not matter when they are chosen.

3. Modular Accumulator Constructions

In this section, we introduce the idea of combining different accumulators to obtain new accumulators with different properties. This technique can lead to the creation of more efficient accumulators, such as the Braavos accumulator described in Section 4.

We focus here on obtaining an accumulator with strong security and functionality properties from two weaker ones. We describe other modular constructions, which achieve rich functionality by combining simpler accumulators, in the full version of this paper [1].

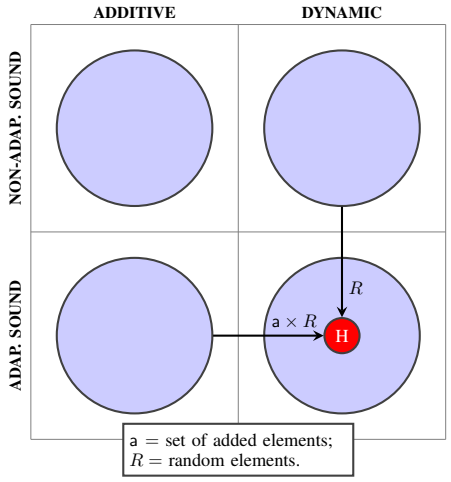

Figure 3. Our Modular Accumulator Construction H. All accumulators in this diagram are positive. R denotes random elements. A specific (particularly efficient) instantiation of this construction is discussed further in Section 4.

Given an (adaptively) sound positive additive accumulator $\mathsf{ACC_A}$ and non-adaptively sound (NA-sound) positive dynamic accumulator $\mathsf{ACC_{NA}}$, we can build an adaptively sound dynamic accumulator ACC, as shown in Figure 3.[4] We call this construction "Construction H". Its algorithms are described in Figures 4 through 6. When an element x is added, the accumulator manager selects a random element r from the domain D of $\mathsf{ACC_{NA}}$. She then adds r to $\mathsf{ACC_{NA}}$, and (x, r) to $\mathsf{ACC_A}$. (Recall that random elements can always be safely accumulated in non-adaptively sound accumulators, since those random elements can be chosen without using any information about the accumulator.) When deleting x, the accumulator manager removes r from $\mathsf{ACC_{NA}}$. Proving the membership of x in ACC consists of producing an r and proving that $(r \in \mathsf{ACC_{NA}}) \wedge ((x, r) \in \mathsf{ACC_A})$.

Note that, in order to support deletions, the accumulator manager must store a mapping from every element x to the corresponding r. This can be avoided by having the accumulator manager use a pseudorandom function F_s (where s is the secret pseudorandom function seed) to select an r corresponding to a given x: $r = F_s(x)$. Even though this causes elements added to $\mathsf{ACC_{NA}}$ to be computed rather than chosen at random (therefore seemingly requiring

4. Shamir and Tauman [15] achieve a similar goal of construct chosen message unforgeable signatures from random message unforgeable ones by using a different technique.

$\mathsf{Gen}(1^\lambda, \emptyset)$:

1) $(\mathsf{ACC_A}.a, \mathsf{ACC_A}.m, \mathsf{ACC_A}.sk) \leftarrow \mathsf{ACC_A.Gen}(1^\lambda, \emptyset)$
2) $(\mathsf{ACC_{NA}}.a, \mathsf{ACC_{NA}}.m, \mathsf{ACC_{NA}}.sk) \leftarrow \mathsf{ACC_{NA}.Gen}(1^\lambda, \emptyset)$
3) Let $\{F_s\}_{s\in\{0,1\}^\lambda}$ be a pseudorandom function family whose range is the domain of $\mathsf{ACC_{NA}}$; pick $s \in \{0,1\}^\lambda$ at random.
4) Let $sk = (\mathsf{ACC_A}.sk, \mathsf{ACC_{NA}}.sk, s)$.
5) Let $a = (\mathsf{ACC_A}.a, \mathsf{ACC_{NA}}.a)$.
6) Let $m = (\mathsf{ACC_A}.m, \mathsf{ACC_{NA}}.m)$.
7) Return (sk, a, m).

$\mathsf{Add}(sk, a, m, x)$:

1) Set $r = F_s(x)$.
2) $(\mathsf{ACC_A}.a, \mathsf{ACC_A}.m, \mathsf{ACC_A}.w, \mathsf{ACC_A.upmsg}) \leftarrow \mathsf{ACC_A.Add}(\mathsf{ACC_A}.sk, \mathsf{ACC_A}.a, \mathsf{ACC_A}.m, (x, r))$.
3) $(\mathsf{ACC_{NA}}.a, \mathsf{ACC_{NA}}.m, \mathsf{ACC_{NA}}.w, \mathsf{ACC_{NA}.upmsg}) \leftarrow \mathsf{ACC_{NA}.Add}(\mathsf{ACC_{NA}}.sk, \mathsf{ACC_{NA}}.a, \mathsf{ACC_{NA}}.m, r)$.
4) Let $w = (\mathsf{ACC_A}.w, \mathsf{ACC_{NA}}.w)$.
5) Let $\mathsf{upmsg} = (\mathsf{ACC_A.upmsg}, \mathsf{ACC_{NA}.upmsg})$.
6) Return $(a, m, w, \mathsf{upmsg})$.

$\mathsf{Del}(sk, a, m, x)$:

1) Set $r = F_s(x)$.
2) Let $(\mathsf{ACC_{NA}}.a, \mathsf{ACC_{NA}}.m, \mathsf{ACC_{NA}.upmsg}) \leftarrow \mathsf{ACC_{NA}.Del}(\mathsf{ACC_{NA}}.sk, \mathsf{ACC_{NA}}.a, \mathsf{ACC_{NA}}.m, r)$.
3) Return $(a, m, \mathsf{ACC_{NA}.upmsg})$.

Figure 4. Construction H from Figure 3 accumulator manager algorithms (Gen, Add, Del and MemWitCreate), in terms of the underlying positive adaptively sound accumulator $\mathsf{ACC_A}$ and positive dynamic non-adaptively sound accumulator $\mathsf{ACC_{NA}}$. In all of these algorithms, we assume that $sk = (\mathsf{ACC_A}.sk, \mathsf{ACC_{NA}}.sk, s)$, $a = (\mathsf{ACC_A}.a, \mathsf{ACC_{NA}}.a)$, and $m = (\mathsf{ACC_A}.m, \mathsf{ACC_{NA}}.m)$.

$\mathsf{MemWitUpOnAdd}(a, x, w, \mathsf{upmsg})$:

1) Parse $(\mathsf{ACC_A.upmsg}, \mathsf{ACC_{NA}.upmsg}) = \mathsf{upmsg}$.
2) Parse $(\mathsf{ACC_A}.w, \mathsf{ACC_{NA}}.w) = w$.
3) $\mathsf{ACC_A}.w \leftarrow \mathsf{ACC_A.MemWitUpOnAdd}(x, \mathsf{ACC_A}.w, \mathsf{ACC_A.upmsg})$.
4) $\mathsf{ACC_{NA}}.w \leftarrow \mathsf{ACC_{NA}.MemWitUpOnAdd}(x, \mathsf{ACC_{NA}}.w, \mathsf{ACC_{NA}.upmsg})$.
5) Return $w = (\mathsf{ACC_A}.w, \mathsf{ACC_{NA}}.w)$.

$\mathsf{MemWitUpOnDel}(a, x, w, \mathsf{upmsg})$:

1) Parse $(\mathsf{ACC_A}.w, \mathsf{ACC_{NA}}.w) = w$.
2) $\mathsf{ACC_{NA}}.w \leftarrow \mathsf{ACC_{NA}.MemWitUpOnAdd}(x, \mathsf{ACC_{NA}}.w, \mathsf{upmsg})$.
3) Return $w = (\mathsf{ACC_A}.w, \mathsf{ACC_{NA}}.w)$.

The witness holder can run BatchMemWitUpOnDel immediately before producing a proof.

Figure 5. Construction H from Figure 3 witness holder algorithms (MemWitUpOnAdd and MemWitUpOnDel), in terms of the underlying positive adaptively sound accumulator $\mathsf{ACC_A}$ and positive dynamic non-adaptively sound accumulator $\mathsf{ACC_{NA}}$. Note that when Construction H is instantiated with a digital signature scheme as the positive accumulator $\mathsf{ACC_A}$, there is no need for a MemWitUpOnAdd algorithm.

adaptive soundness rather than non-adaptive soundness), non-adaptive soundness is still sufficient because of the indstinguishability of the pseudorandom and random cases.

The correctness of Construction H follows by inspection.

Theorem 1. *Construction H is an adaptively sound positive dynamic accumulator if* $\mathsf{ACC_A}$ *is a positive additive adaptively sound accumulator,* $\mathsf{ACC_{NA}}$ *is a positive dynamic non-adaptively sound accumulator, and* F_s *is a pseudorandom function.*

$\mathsf{VerMem}(a, x, w)$:

1) Parse $(\mathsf{ACC_A}.a, \mathsf{ACC_{NA}}.a) = a$.
2) Parse $(\mathsf{ACC_A}.w, \mathsf{ACC_{NA}}.w) = w$.
3) Let $b_1 \leftarrow \mathsf{ACC_A.VerMem}(\mathsf{ACC_A}.a, x, \mathsf{ACC_A}.w)$.
4) Let $b_2 \leftarrow \mathsf{ACC_{NA}.VerMem}(\mathsf{ACC_{NA}}.a, x, \mathsf{ACC_{NA}}.w)$.
5) Return 1 if $b_1 = b_2 = 1$, and return 0 otherwise.

Figure 6. Construction H from Figure 3 third party algorithms (VerMem), in terms of the underlying positive adaptively sound accumulator $\mathsf{ACC_A}$ and positive dynamic non-adaptively sound accumulator $\mathsf{ACC_{NA}}$.

Proof. The proof consists of a reduction to the adaptive soundness of $\mathsf{ACC_A}$, or the non-adaptive soundness of $\mathsf{ACC_{NA}}$, or the pseudorandomness of F_s. We give only the outline here, because the formal reductions are simple exercises. If an adversary produces a witness for an element that is not a member of the accumulated set, then there are two cases: either the element was never added, or it was deleted. In the first case, the adversary has succeeded in breaking the adaptive soundness of $\mathsf{ACC_A}$. In the second case, the adversary has succeeded in forging a witness for a pseudo-random element in $\mathsf{ACC_{NA}}$. We know that if truly random elements were used in $\mathsf{ACC_{NA}}$, then this would break the non-adaptive soundness of $\mathsf{ACC_{NA}}$, since random elements could have been chosen non-adaptively. Thus, if the adversary is able to forge a witness for $\mathsf{ACC_{NA}}$ with pseudorandom elements, the adversary breaks either the non-adaptive soundness of $\mathsf{ACC_{NA}}$ or the pseudorandomness of F_s. □

4. Braavos: A Communication-Optimal Adaptively Sound Dynamic Accumulator

In this section we introduce the Braavos accumulator, which is an instantiation of construction H from Figure 3. Braavos is an adaptively sound positive dynamic accumulator derived from an adaptively sound positive additive accumulator $\mathsf{ACC_A}$ and an non-adaptively sound positive dynamic accumulator $\mathsf{ACC_{NA}}$.

We aim for Braavos to have two properties: communication optimality [16] and efficient zero knowledge proofs, as described in Section 4.4. Our choice of underlying adaptively sound accumulator $\mathsf{ACC_A}$ in Braavos is the CL signature scheme [17], because it supports efficient zero knowledge proofs of knowledge of a signature on a committed value. Note that though construction H has a MemWitUpOnAdd algorithm (described in Figure 5), this algorithm is not used by Braavos, since signatures do not require witness updates when additions take place.

The challenge that remains is finding a communication-optimal, dynamic, non-adaptively sound accumulator

$\mathsf{ACC}_{\mathsf{NA}}$. $\mathsf{ACC}_{\mathsf{NA}}$ should only require membership witness updates upon element deletions, not element additions. In Section 4.1, we describe CL-RSA-B, which is exactly such an accumulator.

4.1. CL-RSA-B: A Communication-Optimal Non-Adaptively Sound Dynamic Accumulator

In this section, we formally describe the CL-RSA-B accumulator, which was informally introduced by Camenisch and Lysyanskaya [2] in a remark on page 12. The CL-RSA-B accumulator is similar to the standard RSA accumulator [2], which evolves the accumulator value (as well as all membership witnesses) with every addition and deletion. The CL-RSA-B accumulator, unlike the RSA accumulator, evolves the accumulator value with every deletion only. However, the price is that, as far as we can tell, the CL-RSA-B accumulator is only non-adaptively sound.

The RSA Accumulator. In order to understand the CL-RSA-B accumulator, it helps to understand the RSA accumulator first. Its value is a quadratic residue a modulo n, where n is an RSA modulus: $n = pq$, where $p = 2p' + 1$ and $q = 2q' + 1$ for prime p, p', q, and q'. The domain D of the RSA accumulator consists of all odd positive prime integers x.[5]

During the addition of x to the accumulator, the new accumulator value is computed as $a_{t+1} = a_t^x \bmod n$. The membership witness w for x is then defined to be the old accumulator value a_t. A membership verification consists of checking that $a = w^x \bmod n$. When another element y is added to the accumulator, the membership witness for x is updated by taking $w_{t+1} = w_t^y \bmod n$.

When an element y is deleted, the accumulator manager (who knows the trapdoor $p'q'$) computes the new accumulator as $a_{t+1} = a_t^{y^{-1} \bmod p'q'} \bmod n$. The membership witness w for x can then be updated using the Bezout coefficients b and c such that $bx + cy = 1$. (Recall that the domain D of the accumulator contains only odd prime numbers, so such b and c are guaranteed to exist.) The new witness is computed as $w_{t+1} = w_t^c a_{t+1}^b \bmod n$.

The CL-RSA-B Accumulator. The CL-RSA-B accumulator preserves the relationship between the accumulator value and the witnesses, but avoids computing a new accumulator value and updating witnesses during each addition. Instead, during the addition of odd prime x the accumulator manager keeps the accumulator constant, and computes the membership witness w for x as $w = a^{x^{-1} \bmod p'q'} \bmod n$. Notice that this eliminates the need for updating existing membership witnesses during additions. The process for proving membership and for deletions is the same as in the RSA accumulator. The algorithms of the CL-RSA-B accumulator are detailed in Figure 7.

5. Note that p' or q' cannot themselves be accumulated, since $(p')^{-1} \bmod p'q'$ and $(q')^{-1} \bmod p'q'$ do not exist; however, that only happens with negligible probability in the adaptive soundness game in Definition 1, since if the adversary finds p' or q', he or she has succeeded in factoring n.

$\mathsf{Gen}(1^\lambda, \emptyset)$:

1) Select two λ-bit safe primes $p = 2p' + 1$ and $q = 2q' + 1$ where p' and q' are also prime, and let $n = pq$. (Consider n to be public knowledge from hereon out; it is actually a part of the accumulator value a, but for simplicity we will not refer to it as such.)
2) Let $sk = p'q'$.
3) Select a random integer $a' \leftarrow \mathbb{Z}_n^*$.
4) Let $a = (a')^2 \bmod n$.
5) Return (sk, a).

$\mathsf{Add}(sk, a, x)$:

1) Check that $x \in D$ (that is, that x is an odd prime). If not, FAIL.
2) Let $w = a^{x^{-1} \bmod p'q'} \bmod n$.
3) Return $(a, w, \mathsf{upmsg} = \perp)$.

$\mathsf{Del}(sk, a, x)$:

1) Check that $x \in D$ (that is, that x is an odd prime). If not, FAIL.
2) Let $a = a^{x^{-1} \bmod p'q'} \bmod n$.
3) Let $\mathsf{upmsg} = (a, x)$.
4) Return (a, upmsg).

$\mathsf{MemWitUpOnDel}(a, x, w, \mathsf{upmsg})$:

1) Parse $(a, y) = \mathsf{upmsg}$.
2) Compute Bezout coefficients b, c such that $bx + cy = 1$. (Given that $x \neq y$, since both x and y are prime, such b and c are guaranteed to exist.)
3) Let $w = w^c a^b \bmod n$.
4) Return w.

$\mathsf{VerMem}(a, x, w)$:

1) Return 1 if $a = w^x \bmod n$.
2) Return 0 otherwise.

Figure 7. CL-RSA-B algorithms.

CL-RSA-B Soundness. The RSA accumulator is adaptively sound, meaning that an adversary cannot find a membership witness for an element that is not a member even if she chooses which elements should be added, optionally based on accumulator and witness values she has previously seen.

The CL-RSA-B accumulator is non-adaptively sound, meaning that an adversary cannot find a membership witness for an element that is not a member if she chooses all elements to add prior to seeing any accumulator information. In particular, the CL-RSA-B accumulator is sound when only random elements are added to the accumulator, since those can be chosen prior to seeing any accumulator or witness values.[6] This holds under the *strong RSA assumption* [18].

Assumption 1 (Strong RSA)**.** *For any probabilistic polynomial-time adversary $\mathcal{A}$,*

$$\Pr[p, q \leftarrow \{\lambda\text{-bit safe primes}\}; n = pq; t \leftarrow \mathbb{Z}_n^*; (r, e) \leftarrow \mathcal{A}(n, t) : t = r^e \bmod n \wedge e \text{ is prime}] = \nu(\lambda)$$

For some negligible function ν.

6. We are not certain whether CL-RSA-B is also adaptively sound. Proving that it is or is not is an open problem. It is adaptively sound when the when a polynomial-size subset of D is used as the domain; however, this is a very limiting restriction.

Theorem 2. *The* CL-RSA-B *accumulator with a domain D consisting of odd primes is non-adaptively sound under the strong RSA assumption.*

Proof. In Figure 8, we reduce the non-adaptive soundness of the CL-RSA-B accumulator to the strong RSA assumption. The reduction $\mathcal{R}$ takes in an RSA integer n and a random value $t \in \mathbb{Z}_n^*$, and returns r, e such that $t = r^e \bmod n$. $\mathcal{R}$ leverages an adversary $\mathcal{A}$ which can break the non-adaptive soundness of the CL-RSA-B accumulator; that is, after making addition (Add) and deletion (Del) queries on elements chosen before seeing the initial state of the accumulator, $\mathcal{A}$ can produce an odd prime x and a witness w such that $a = w^x \bmod n$, and x is not in the accumulator.

$\mathcal{R}$ must be able to answer two types of queries from $\mathcal{A}$: Add queries on the non-adaptively chosen elements, and Del queries on the same elements. Let q_{Add} be an upper bound on the number of Add queries, and q_{Del} be an upper bound on the number of Del queries $\mathcal{A}$ can make. During the setup phase, having received the elements $x_1, \ldots, x_{q_{\mathsf{Add}}}$ from $\mathcal{A}$, the reduction $\mathcal{R}$ creates an accumulator for which it can answer Add and Del queries on elements $x_1, \ldots, x_{q_{\mathsf{Add}}}$. It does so by starting with $a = t^2 \bmod n$, and raising a to the power of the elements. By raising a to the power of $x_j^{q_{\mathsf{Del}}}$, $\mathcal{R}$ creates an accumulator value for which it is able to answer Del and Add queries on x_j even if $\mathcal{A}$ spends all of its Del queries on that one element. However, if $\mathcal{A}$ forges a witness w for x_j (after having added and deleted it fewer than q_{Del} times), the reduction won't be able to use w to break the strong RSA assumption, since it already knows w! For that reason, $\mathcal{R}$ guesses a "target" element x_j from among $x_1, \ldots, x_{q_{\mathsf{Add}}}$, and the number e_j of times that x_j will be added and deleted before the forgery (which can be anywhere from 0 to q_{Del}), and only raises t to the power of $x_j^{e_j}$, not $x_j^{q_{\mathsf{Del}}}$. Figure 8 shows the details of how the reduction picks an accumulator value based on $x_1, \ldots, x_{q_{\mathsf{Add}}}$, how it answers Add and Del queries, and how it then uses the output of $\mathcal{A}$ to break the strong RSA assumption.

This reduction succeeds as long as:

1) During the query phase, $\mathcal{R}$ does not output FAIL. $\mathcal{R}$ does not output FAIL if the target exponent e_i was chosen correctly, which happens with probability $\frac{1}{q_{\mathsf{Del}}+1}$.
2) During the output phase, $\mathcal{R}$ does not output FAIL. If $\mathcal{A}$ outputs a witness for an element $x_i \in \{x_1, \ldots, x_{q_{\mathsf{Add}}}\}$, $\mathcal{R}$ does not output FAIL as long as:
   a) $\mathcal{R}$ makes x the "target" prime (that is, $j = i$). This happens with probability $\frac{1}{q_{\mathsf{Add}}}$.
   b) $\mathcal{R}$ correctly chooses the target exponent e_i for x. This happens with probability $\frac{1}{q_{\mathsf{Del}}+1}$. However, this is already accounted for in item 1.
3) $\mathcal{A}$ succeeds in breaking the security of the CL-RSA-B accumulator, which we assume happens with non-negligible probability ϵ.

As long as $\mathcal{R}$ does not output FAIL, $\mathcal{A}$ sees the same transcript it would when interacting with a real accumulator manager. The probability of the reduction $\mathcal{R}$ succeeding is $\frac{1}{q_{\mathsf{Add}}} \frac{1}{q_{\mathsf{Del}}+1} \epsilon$, which is non-negligible.

$\mathsf{Setup}(n, t, q_{\mathsf{Add}}, q_{\mathsf{Del}})$:

1) Let $x_1, \ldots, x_{q_{\mathsf{Add}}}$ be the distinct odd primes provided by the adversary $\mathcal{A}$.
2) Let $a = t^2$ (so as to make a a quadratic residue).
3) Let $e_c = 2$ be the current exponent linking t to a. (So, $a = t^{e_c} \bmod n$ is an invariant.)
4) Pick a random index $j \leftarrow \{1, \ldots, q_{\mathsf{Add}}\}$.
5) For $i \in [1, \ldots, q_{\mathsf{Add}}]$:
   a) If $i = j$: pick a random $e_i \leftarrow \{0, \ldots, q_{\mathsf{Del}}\}$.
   b) Else: $e_i = q_{\mathsf{Del}}$.
   c) Let $a = a^{(x_i^{e_i})} \bmod n$.
   d) Let $e_c = e_c x_i^{e_i}$.
6) Return $a = e_i^{e_c} \bmod n$ to the adversary $\mathcal{A}$.

$\mathsf{Add}(x_i)$:

1) If $e_i = 0$: FAIL.
2) Let $w = t^{e_c/x_i} \bmod n$. (Note that e_c must be divisible by x_i, since e_c has a factor of $x_i^{e_i}$, and $e_i > 0$.)
3) Return w.

$\mathsf{Del}(x_i)$:

1) If $e_i = 0$: FAIL.
2) Let $e_c = e_c/x_i$. (Note that e_c must be divisible by x_i, since e_c has a factor of $x_i^{e_i}$, and $e_i > 0$.)
3) Let $e_i = e_i - 1$.
4) Let $a = t^{e_c} \bmod n$.
5) Let $\mathsf{upmsg} = (a, x_i)$.
6) Return (a, upmsg).

$\mathsf{Output}(e, w)$:

1) Check that $a = w^e \bmod n$. If not, FAIL.
2) If $e = x_i$ for $i \in \{1, \ldots, q_{\mathsf{Add}}\}$ and $(i \neq j$ or $e_i > 0)$: FAIL.
3) We know that e and e_c must be relatively prime; e_c has no factors outside of $x_1, \ldots, x_{q_{\mathsf{Add}}}$, and those factors have powers e_i. So, $\mathcal{R}$ can compute Bezout coefficients b, c such that $be + ce_c = 1$.
4) Let $r = t^b w^c \bmod n$.
   (Let $y = t^{e^{-1} \bmod p'q'} \bmod n$; equivalently, $y^e \bmod n = t$. Since $w^e \bmod n = a$ and $t^{e_c} \bmod n = (y^e)^{e_c} \bmod n = a$, it follows that $w = y^{e_c} \bmod n$. So, $r = t^b w^c = (y^e)^b (y^{e_c})^c = y^1 = y$.)
5) Return (r, e).

Figure 8. Reduction $\mathcal{R}$ from the non-adaptive soundness of CL-RSA-B to the strong RSA assumption.

□

Other Approaches for Expanding the Domain and Getting Adaptive Soundness for CL-RSA-B. The domain D of CL-RSA-B consists of odd primes. Such a limited domain is not a problem for our main application, because the Braavos accumulator manager can choose a (psuedo)random prime r when a new element is added to the accumulator for the first time, as described in Construction H. In fact, Construction H can be viewed as one approach to expanding the domain of CL-RSA-B and obtaining adaptive security for it. Here we briefly mention other approaches. Let D' be the desired domain. Let f be a mapping from D' to λ-bit odd primes. To add $x \in D'$ to the accumulator, add $f(x)$ instead. We can obtain adaptive soundness in the following ways:

- We can model f as a random oracle (the proof is straightforward).

- We can avoid the random oracle by making a different strong assumption instead: namely, the assumption that f is collision-resistant, and the very strong "adaptive strong-RSA assumption". Informally, the adaptive strong-RSA assumption states that even given an oracle that can take roots modulo n, it is difficult to find new roots whose power is relatively prime to those of the roots produced by the oracle.
- We can get somewhat better assumptions by having f be a randomized mapping, and include the randomness R as part of the witness. Then, assuming that for every two elements x_1 and x_2, the distributions $f(x_1; R)$ and $f(x_2; R)$ (over random choices of R) are statistically close, we can use the technique from [19]. To do so, we need to assume that the strong RSA assumption (Assumption 1) also holds in a model where there exists an oracle $\mathcal{O}$ that on input x, p returns a random R such that $f(R; x) = p$.
- Alternatively, we can use the strong-RSA assumption without modification if f is a trapdoor hash function, following the technique of [15].

All of these approaches require f that maps to primes. A way to build such f is described in [20, Section 3.2] (see also [21, Section 7]).

4.2. Braavos Soundness

The Braavos accumulator uses Camenisch-Lysyanskaya (CL) signatures [17] as the underlying positive accumulator $\mathsf{ACC_A}$, and the CL-RSA-B accumulator as the underlying dynamic positive non-adaptively sound accumulator $\mathsf{ACC_{NA}}$. CL signatures are existentially unforgeable under the strong RSA assumption. Recall that according to Theorem 2, the CL-RSA-B accumulator is non-adaptively sound under the same assumption. By Theorem 1, this implies that the Braavos accumulator is an adaptively sound positive dynamic accumulator under the strong RSA assumption.

Properties other than adaptive soundness (such as correctness) are self-evident.

4.3. Comparison with Other Constructions

The Braavos accumulator is a positive, dynamic accumulator with efficient (constant-time) membership witness generation, and no membership witness updates upon element additions — only upon element deletions. In particular, for a fixed security parameter λ, Braavos achieves the total communication lower bound shown by Camacho [16]. (Total communication refers to the sum of the sizes of all upmsg messages sent by the accumulator manager to the witness holders after $|\mathsf{a}|$ additions and $|\mathsf{d}|$ deletions.) In Appendix A of the full version of this paper [1] , we prove that adding universality would necessarily degrade the total communication of Braavos.

In Figure 9, we compare Braavos to prior constructions in terms of the properties introduced in Section 2. We compare it to digital signatures, and to the three other primary lines of work on accumulators: the RSA construction [6], [2], [7], the bilinear map constructions [3], [8], [9], and the Merkle tree constructions [5].[7] In our comparison we also include CL-RSA-B, which is used in Braavos and described in Section 4.1.

Though Figure 9 includes some of the most well known accumulator constructions to compare with Braavos , we would like to note that there exists a large number of other dynamic accumulator constructions in the literature [7], [8], [4], [9], [23], [24]. To the best of our knowledge, these constructions do not achieve the efficiency we aim for.

4.4. Adding Zero Knowledge to Braavos

So far, we have only discussed the functionality of accumulators, ignoring potential privacy concerns. There typically exist three primary privacy goals in the context of accumulators: hiding the membership (or non-membership) witness, hiding the element whose membership (or non-membership) is being demonstrated as well as the witness, and hiding all information about the accumulated set [25]. For our application of anonymous credential revocation (discussed in Section 5), we mostly care about zero knowledge proofs of member knowledge, which hide not only the witness, but the member element itself.

The Braavos accumulator supports efficient zero-knowledge proofs of member knowledge. Given that Braavos is composed of two accumulators $\mathsf{ACC_A}$ and $\mathsf{ACC_{NA}}$, in order for a witness holder to produce a zero-knowledge proof of member knowledge in Braavos, she would have to produce a conjunction of proofs of member knowledge in both $\mathsf{ACC_A}$ and $\mathsf{ACC_{NA}}$ and a proof that those members have the correct relationship. More concretely, she would have to compute the following zero-knowledge proof (described using Camenisch-Stadler [26] notation):

$$\begin{aligned}&\mathsf{ZKP}[(x, r, \mathsf{ACC_A}.w, \mathsf{ACC_{NA}}.w) : \\ &\quad \wedge\ \mathsf{ACC_A}.\mathsf{VerMem}(\mathsf{ACC_A}.a, (x, r), \mathsf{ACC_A}.w) \\ &\quad \wedge\ \mathsf{ACC_{NA}}.\mathsf{VerMem}(\mathsf{ACC_{NA}}.a, r, \mathsf{ACC_{NA}}.w) \\ &\quad](\mathsf{ACC_{NA}}.a, \mathsf{ACC_A}.a)\end{aligned}$$

Where $\mathsf{ACC_A}$ is the signature scheme $\mathsf{SIG}_{CL} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ due to Camenisch and Lysyanskaya [17], and $\mathsf{ACC_{NA}}$ is the CL-RSA-B accumulator.

For integration into larger systems, it might be important to be able to link the witnesses used in the proof to other statements, while still keeping the elements and witnesses private. To this end, commitments to the witnesses can be used. Let $\mathsf{Com} = (\mathsf{Commit}, \mathsf{Verify})$ be a commitment scheme; to integrate commitments into the zero knowledge proof, a witness holder computes commitments to the membership witnesses $\mathsf{ACC_A}.w$ and $\mathsf{ACC_{NA}}.w$: $(C_1, o_1) = \mathsf{Com.Commit}(\mathsf{ACC_A}.w)$ and $(C_2, o_2) =$

7. For those interested in a more concrete comparison, Lapon et al. [22] provide concrete running time measurements of bilinear map accumulator constructions and the RSA construction. We discuss the concrete running times of Braavos in Section 5.2.

$\mathsf{Com.Commit}(\mathsf{ACC_{NA}}.w)$, where o_1 and o_2 are decommitment values. The proof is then enhanced, as follows:

$$\begin{aligned}&\mathsf{ZKP}[(x, r, \mathsf{ACC_A}.w, \mathsf{ACC_{NA}}.w, o_1, o_2) :\\ &\quad \mathsf{Com.Verify}(C_1, \mathsf{ACC_A}.w, o_1)\\ &\quad \wedge \mathsf{Com.Verify}(C_2, \mathsf{ACC_{NA}}.w, o_2)\\ &\quad \wedge \mathsf{ACC_A.VerMem}(\mathsf{ACC_{NA}}.a, (x, r), \mathsf{ACC_A}.w)\\ &\quad \wedge \mathsf{ACC_{NA}.VerMem}(\mathsf{ACC_{NA}}.a, r, \mathsf{ACC_{NA}}.w)\\ &\quad](\mathsf{ACC_{NA}}.a, \mathsf{ACC_A}.a, C_1, C_2)\end{aligned}$$

For concrete descriptions of the individual clauses of this proof using the commitment scheme due to Fujisaki and Okamoto [27], please refer to Fujisaki and Okamoto [27] and Camenisch and Lysyanskaya [17], [2].

5. Anonymous Revocation from Accumulators

In this section, we show how accumulators can be used in practice. Accumulators combined with zero knowledge proofs are a perfect solution for providing revocation in a system where preserving users' privacy is crucial.

As an example, consider an anonymous credential system where transactions involving the same credential need to be unlinkable. An anonymous credential system is comprised of users, issuers, and verifiers. An issuer certifies a user's attributes in the form of a credential. To authenticate a user, a verifier first sends her a presentation policy that describes which statements she should prove about her credentials. Based on the policy, the user derives a fresh unlinkable proof (or *token*) from her credentials and sends it to the verifier. The verifier then determines whether the token is valid with respect to the policy.

To make credentials *revocable*, the user needs to be able to prove that the credential, on which the token is based, was not revoked. This must be done in a privacy-preserving fashion, i.e., without destroying unlinkability. Camenisch et al. [11] describe a generic revocation component, which can be added to any anonymous system, including anonymous credentials and group signature schemes. We refer to this component as the anonymous revocation component (ARC).

An ARC requires an additional entity called a revocation authority (RA). The RA assists the issuer with adding new users to the system, maintains the necessary revocation information, and changes the revocation status of any user in the system. (The role of the RA can optionally be played by an issuer or a verifier.) Camenisch et al. [11] describe the necessary interfaces and definitions for an ARC, and show how to instantiate it with the revocation scheme of Nakanishi et al. [28].

In Section 5.1, we show how to instantiate an ARC with accumulators and zero knowledge proofs, and extend the definition of an ARC to include the addition of users after the initial setup and the re-addition of users after they have been revoked. In Section 5.2, we provide performance measurements of our ARC with Braavos in the Identity Mixer anonymous credential system (idemix) [10].

8. $|\mathsf{a}|$ and $|\mathsf{d}|$ refer to the number of elements added and deleted *after* the addition of the element whose witness updates are being discussed.

5.1. Anonymous Revocation Component (ARC) with Accumulators

ARC *Syntax.* In an ARC, revocation is done via a special value called a *revocation handle* (rh) that can be embedded into the revocable object, i.e., as a special attribute in a credential. rh is bound to the revocable object with a signature. By using a commitment to rh, a proof that rh has not been revoked can be easily combined with any other proof about rh– for example, that rh was signed in a credential, as shown in Section 4.4.

As described by Camenisch et al. [11], an ARC consists of the following algorithms: ARC.SPGen, ARC.RKGen, ARC.RevTokenGen, ARC.Revoke, and ARC.RevTokenVer.

- Revocation parameters are generated using $\mathsf{ARC.SPGen}(spar_g) \rightarrow spar_r$, and then added to the global system parameters $spar_g$.
- The revocation authority RA runs $\mathsf{ARC.RKGen}(spar_r) \rightarrow (rsk, rpk, RI)$ to generate the RA's secret and public keys (rsk, rpk) and the initial revocation information RI. RI contains all public data that parties need in order to generate and verify proofs of non-revocation. RI can also be supplemented by privately held witnesses.
- The RA can revoke a user based on her revocation handle rh by updating the revocation information RI: $\mathsf{ARC.Revoke}(rh, rsk, RI) \rightarrow RI'$.
- A user who has a valid credential can generate a publicly verifiable token rt proving that her revocation handle rh has not been revoked and that C is a commitment to rh. $\mathsf{ARC.RevTokenGen}(rh, C, o, RI, rpk) \rightarrow rt$.
  (For each new revocation token rt, the user generates a fresh commitment to rh in order to avoid making her tokens linkable. (C, o, rh) can also be used in other proofs - for example, the user should also prove that rh is an attribute of her credential.)
- A verifier can check such a token by running $\mathsf{ARC.RevTokenVer}(rt, C, RI, rpk) \rightarrow \{0, 1\}$.

Note that ARC.RevTokenGen and ARC.RevTokenVer can be integrated into an interactive protocol if interactive zero knowledge proofs are used instead of non-interactive ones.

Accumulator-Based ARC. Let $\mathsf{ACC_P}$ and $\mathsf{ACC_N}$ be a positive and a negative dynamic accumulator, respectively. Let $\mathsf{ZKP} = (\mathsf{Prove}, \mathsf{Verify})$ be a zero knowledge proof of knowledge system (as described in Section 4.4), $\mathsf{SIG} = (\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ be a signature scheme, and $\mathsf{Com} = (\mathsf{Commit}, \mathsf{Verify})$ be a commitment scheme.

Let $\mathcal{RS}$ be a set of supported revocation handles, $\mathcal{W}$ be a list of all witnesses, and $\mathcal{M}$ be a list of all update messages that are contained in the revocation information RI and are necessary for the witness updates.

In Figure 10, we describe our accumulator-based ARC. For clarity we describe two approaches in parallel: the blacklist **[BL]** approach and the whitelist **[WL]** one. In the blacklist approach, only the revoked users' revocation

Protocol Runtimes						
Accumulator	**Sigs**	**RSA**	**BM**	**Merkle**	CL-RSA-B	Braavos
Add	1	1	1	$\log \lvert\mathsf{a}\rvert$	1	1
Del	–	1	1	$\log \lvert\mathsf{a}\rvert$	1	1
NonMemWitCreate (without sk)	–	$\lvert S\rvert$	$\lvert S\rvert$	$\log \lvert\mathsf{a}\rvert$	–	–
MemWitUpOnAdd	0	1	1	$\log \lvert\mathsf{a}\rvert$	0	0
MemWitUpOnDel	–	1	1	$\log \lvert\mathsf{a}\rvert$	1	1
NonMemWitUpOnAdd	–	1	1	$\log \lvert\mathsf{a}\rvert$	–	–
NonMemWitUpOnDel	–	1	1	$\log \lvert\mathsf{a}\rvert$	–	–
VerMem	1	1	1	$\log \lvert\mathsf{a}\rvert$	1	1
VerNonMem	–	1	1	$\log \lvert\mathsf{a}\rvert$	–	–
Storage						
Accumulator	**Sigs**	**RSA**	**BM**	**Merkle**	CL-RSA-B	Braavos
Accumulator size	1	1	1	1	1	1
Witness size	1	1	1	$\log \lvert\mathsf{a}\rvert$	1	1
Manager storage ($\lvert m\rvert$)	1	$\lvert S\rvert$	$\lvert S\rvert$	$\lvert\mathsf{a}\rvert$	1	1
Properties						
Accumulator	**Sigs**	**RSA**	**BM**	**Merkle**	CL-RSA-B	Braavos
Additive?	✓	✓	✓	✓	✓	✓
Subtractive?		✓	✓	✓	✓	✓
Positive?	✓	✓	✓	✓	✓	✓
Negative?		✓	✓	✓		
Total communication to Verifier [8]	0	$\lvert\mathsf{a}\rvert + \lvert\mathsf{d}\rvert$	$\lvert\mathsf{a}\rvert + \lvert\mathsf{d}\rvert$	$(\lvert\mathsf{a}\rvert + \lvert\mathsf{d}\rvert) \log \lvert\mathsf{a}\rvert$	$\lvert\mathsf{d}\rvert$	$\lvert\mathsf{d}\rvert$
Total communication to Member	0	$\lvert\mathsf{a}\rvert + \lvert\mathsf{d}\rvert$	$\lvert\mathsf{a}\rvert + \lvert\mathsf{d}\rvert$	$(\lvert\mathsf{a}\rvert + \lvert\mathsf{d}\rvert) \log \lvert\mathsf{a}\rvert$	$\lvert\mathsf{d}\rvert$	$\lvert\mathsf{d}\rvert$
Efficient ZKPs?	✓	✓	✓		✓	✓
CMA-sound?	✓	✓	✓	✓		✓

Figure 9. Various Accumulators and their Protocol Runtimes, Storage Requirements, and Properties. We let $\lvert\mathsf{a}\rvert$ denote the number of elements added to the accumulator, $\lvert\mathsf{d}\rvert$ denote the number of elements deleted from the accumulator, and $\lvert S\rvert$ denote the total number of member elements in the accumulator. (Note that $\lvert S\rvert$ is $\lvert\mathsf{a}\rvert - \lvert\mathsf{d}\rvert$.) The Braavos accumulator is the first CMA-sound dynamic (additive and subtractive) accumulator to have the optimal total communication of $O(\lvert\mathsf{d}\rvert)$. Sigs represents any digital signature scheme. The RSA Construction is due to [6], [2], [7]. The BM (bilinear map) construction is due to [3], [8], [9]. The Merkle tree construction is due to [5]. CL-RSA-B and Braavos were described earlier in this section. A logarithmic factor is omitted everywhere; it is implicit as the size of our elements. Big-O notation is omitted from this table in the interest of brevity.

handles are added to the accumulator. To prove that her credential has not been revoked, a user proves that her revocation handle rh is not in the accumulator (by means of a non-membership witness). In the whitelist approach all users' revocation handles are added to the accumulator when their credentials are issued, and are removed from the accumulator upon revocation. To prove that her credential has not been revoked, a user proves that her revocation handle rh is in the accumulator (by means of a membership witness). Naturally, a positive accumulator realizes the whitelist approach and a negative accumulator realizes the blacklist one.

5.1.1. Security Analysis of ARC With Accumulators.

We now analyze the security of the accumulator-based ARC constructions described in Figure 10.

Correctness. Correctness requires that whenever an honestly computed revocation information RI is used, an honest user is able to successfully generate valid tokens. See the full version for a formal definition [1].

Theorem 3. *The* ARC *described in Figure 10 is correct.*

The proof follows immediately from the correctness of the accumulator scheme and the properties of ZKP.

Soundness. Revocation soundness captures the following: to make the verifier accept, the user must know the revocation handle contained in the commitment it computes a revocation token for. Further, nobody except for the revocation authority can come up with a new valid revocation information, i.e., the revocation information is always authentic. Finally, this revocation handle must not have been revoked in an earlier revocation step. See the full version for a formal definition [1].

Theorem 4. *Assuming that the accumulators* $\mathsf{ACC_P}$ *and* $\mathsf{ACC_N}$ *are adaptively sound, the signature scheme* SIG *is existentially unforgeable, and the zero knowledge proof system* ZKP *is sound, then the* ARC *described in Figure 10 is sound in the random oracle model.*

Proof. We prove this theorem by showing that a prover can only convince the verifier with negligible probability for each of the three given winning conditions.

Case a: Assume that the adversary outputs $(RI_{\mathcal{A}}, rt, C)$ such that the verifier accepts, but commitment verification fails. By construction, a revocation token rt is a zero knowledge proof of knowledge of rh, o such that $\mathsf{Com.Verify}(rh, C, o) = 1$. Therefore, it follows from the soundness of ZKP that $\mathsf{Com.Verify}(rh, C, o) =$

$\mathsf{ARC.SPGen}(spar_g)$**:** Using global system parameters $spar_g$ (group descriptions, parameters for ZKP, etc.), generate revocation system parameters $spar_r = (spar_g, \mathcal{RS})$, where $\mathcal{RS}$ specifies the set of supported revocation handles. The revocation system parameters can be given to any algorithm of the revocation framework.

$\mathsf{ARC.RKGen}(spar_r)$**:**

- Generate the initial accumulator value:
  [WL]: $(sk, a, m) \leftarrow \mathsf{ACC_P.Gen}(1^\lambda, \emptyset)$.
  [BL]: $(sk, a, m) \leftarrow \mathsf{ACC_N.Gen}(1^\lambda, \emptyset)$.
- Generate witnesses for every $rh \in \mathcal{RS}$:
  [WL]: $(a, m, w^{rh}, \mathsf{upmsg}) \leftarrow \mathsf{ACC_P.Add}(sk, a, m, rh)$. Let $\mathcal{M} = \mathcal{M} \cup \{\mathsf{upmsg}\}, \mathcal{W} = \mathcal{W} \cup \{w^{rh}\}$.
  [BL]: $w^{rh} \leftarrow \mathsf{ACC_N.NonMemWitCreate}(\overline{sk}, \overline{a}, \overline{m}, rh)$. Let $\mathcal{W} = \mathcal{W} \cup \{w^{rh}\}, \mathcal{M} = \emptyset$.
- Generate signing keys $(sgk, vk) \leftarrow \mathsf{SIG.KeyGen}(spar_g)$.
- Sign the revocation information: $\sigma \leftarrow \mathsf{SIG.Sign}(sgk, (a, \mathcal{W}, \mathcal{M}))$.
- Output $rpk = vk$, $rsk = (sgk, sk, m)$, and $RI = (a, \mathcal{W}, \mathcal{M}, \sigma)$.

$\mathsf{ARC.Revoke}(rh, rsk, RI)$**:**

- Parse RI as $(a, \mathcal{W}, \mathcal{M}, \sigma)$ and rsk as (sgk, sk, m). Abort if $\mathsf{SIG.Verify}(vk, \sigma, (a, \mathcal{W}, \mathcal{M})) = 0$.
- Update the accumulator value:
  [WL]: $(a, m, \mathsf{upmsg}) \leftarrow \mathsf{ACC_P.Del}(sk, a, m, rh)$.
  [BL]: $(a, m, w, \mathsf{upmsg}) \leftarrow \mathsf{ACC_N.Add}(sk, a, m, rh)$.
- Let $\mathcal{M} = \mathcal{M} \cup \{\mathsf{upmsg}\}$.
- Remove the corresponding witness from the revocation information: $\mathcal{W} = \mathcal{W} \backslash \{w^{rh}\}$.
- Sign the updated revocation information: $\sigma \leftarrow \mathsf{SIG.sign}(sgk, (a, \mathcal{W}, \mathcal{M})$ and append the signature to $RI : RI = (a, \mathcal{W}, \mathcal{M}, \sigma)$.
- Output the updated RI.

$\mathsf{ARC.RevTokenGen}(rh, C, o, RI, rpk)$**:**

- Parse RI as $(a, \mathcal{W}, \mathcal{M}, \sigma)$ and rpk as vk. Abort if $\mathsf{SIG.Verify}(vk, \sigma, (a, \mathcal{W}, \mathcal{M})) = 0$.
- Update the accumulator witness w^{rh}:
  For every new $\mathsf{upmsg} \in \mathcal{M}$ since the last witness update:
  [WL]: $w^{rh} \leftarrow \mathsf{ACC_P.MemWitUpOnDel}(rh, w^{rh}, \mathsf{upmsg})$
  [BL]: $w^{rh} \leftarrow \mathsf{ACC_N.NonMemWitUpOnAdd}(rh, w^{rh}, \mathsf{upmsg})$.
- Prove knowledge of w^{rh} and rh such that w^{rh} is a witness that rh is (not) in the accumulator a:
  [WL]: $rt \leftarrow \mathsf{ZKP.Prove}[(rh, o, w^{rh}) : \mathsf{Com.Verify}(rh, C, o) = 1 \wedge \mathsf{ACC_P.VerMem}(a, rh, w^{rh}) = 1](C, a)$.
  [BL]: $rt \leftarrow \mathsf{ZKP.Prove}[(rh, o, w^{rh}) : \mathsf{Com.Verify}(rh, C, o) = 1 \wedge \mathsf{ACC_N.VerNonMem}(a, rh, w^{rh}) = 1](C, a)$.
- Output rt.

$\mathsf{ARC.RevTokenVer}(rt, C, RI, rpk)$**:**

- Parse RI as $(a, \mathcal{W}, \mathcal{M}, \sigma)$ and rpk as vk. Abort if $\mathsf{SIG.Verify}(vk, \sigma, (a, \mathcal{W}, \mathcal{M})) = 0$.
- Verify the proof:
  [WL]: $b \leftarrow \mathsf{ZKP.Verify}[(rh, o, w^{rh}) : \mathsf{Com.Verify}(rh, C, o) = 1 \wedge \mathsf{ACC_P.VerMem}(a, rh, w^{rh}) = 1](C, a, rt)$.
  [BL]: $b \leftarrow \mathsf{ZKP.Verify}[(rh, o, w^{rh}) : \mathsf{Com.Verify}(rh, C, o) = 1 \wedge \mathsf{ACC_N.VerNonMem}(a, rh, w^{rh}) = 1](C, a, rt)$.
- Output b.

Figure 10. ARC algorithms using accumulators, for both the whitelisting and blacklisting approaches.

$0 \wedge \mathsf{ARC.RevTokenVer}(rt, C, RI, rpk) = 1$ can only happen with negligible probability.

We note that here we consider the revocation component in isolation, without bridging it with any signature or credential scheme. Therefore, it is not necessary to require the binding property from the commitment scheme. When the revocation handle is used in any other proof (of a valid signature, etc.) the binding property is required for the security of the overall system.

Case b: Assume that the adversary outputs $(RI_{\mathcal{A}}, rt, C)$ such that the verifier accepts although the given revocation information was never generated by the revocation authority. Then the following algorithm $\mathcal{R}$ can be used to break the unforgeability of the underlying signature scheme SIG. Briefly, $\mathcal{R}$ behaves as follows.

- It runs ARC.SPGen and $\mathsf{ARC.RKGen}(spar_r)$ as described in Figure 10, obtaining values rpk, RI and par_r.
- It then calls $\mathcal{A}$ on input (rpk, RI, par_r).
- For every call (Revoke, rh) to $\mathcal{O}^{\mathsf{Revoke}}$, $\mathcal{R}$ computes the updated revocation information by requesting the required signatures from the signing oracle.
- When $\mathcal{A}$ outputs $(RI_{\mathcal{A}}, rt, C)$, $\mathcal{R}$ extracts the signature component $\sigma_{\mathcal{A}}$ of $RI_{\mathcal{A}}$ (and removes the signature from the $RI_{\mathcal{A}}$ value itself.. If the signature $\sigma_{\mathcal{A}}$ does not verify on the signature-less $RI_{\mathcal{A}}$ or if the signature $\sigma_{\mathcal{A}}$ was previously returned by the signing oracle, $\mathcal{R}$ returns $\perp$. Otherwise, $\mathcal{R}$ returns $(RI_{\mathcal{A}}, \sigma_{\mathcal{A}})$.

It is easy to see that $\mathcal{R}$ succeeds in forging a signature as long as it does not output $\perp$, which occurs with non-negligible probability if $\mathcal{A}$ succeeds with non-negligible probability.

Case c: Assume that the adversary outputs $(RI_{\mathcal{A}}, rt, C)$ such that the verifier accepts although the revocation handle was previously revoked. Then through a sequence of games we can build a reduction to the adaptive soundness property of the underlying accumulator scheme and the soundness of ZKP. Briefly, $\mathcal{R}$ uses the adaptive soundness oracles to add and delete revocation handles from the accumulator, and then uses the ZKPK extractor to extract the revocation handle and the corresponding membership witness. It then outputs these revocation handle and witness as a forgery to the adaptive soundness challenger.

□

Privacy. Revocation privacy ensures that no adversary can tell which of two unrevoked revocation handles rh_0, rh_1 underlies a revocation token. See the full version for a formal definition [1].

Theorem 5. *Assuming that the commitment scheme* Com *is hiding and that the zero knowledge proof system* ZKP *is zero knowledge, the* ARC *described in Figure 10 is private.*

As revocation tokens are zero knowledge, they do not leak any information about the revocation handle, and the claim follows immediately.

5.1.2. Enabling Dynamic Revocation. In most large systems, it is insufficient to start with a fixed set of users, and revoke users over time. Other users might need to be added to the system, or users' revocation status might need to be changed back to "un-revoked". We extend the syntax described by Camenisch et al. [11] to include this additional functionality; that is, we add a Join algorithm that allows the RA to add users to the system after the initial setup has been performed.

For the whitelist approach, this reduces the computational complexity of the RA key generation algorithm ARC.RKGen and the size of the revocation information RI, since the RA no longer needs to include all potential users' revocation handles in the initial whitelist accumulator. Users' witnesses are no longer a part of RI, since they are given to their intended holders when a Join is executed. Additionally, RevTokenGen algorithm takes a witness as a separate input.

Let $\mathcal{RU}$ be a set of the revocation handles that are already in use. $\mathcal{RU}$ is initialized in ARC.RKGen, at which time it is empty since we assume that initially there are no users in the system – they are now only added via the Join algorithm. We describe the details of the Join algorithm in Figure 11.

$\mathsf{ARC.Join}(rsk, rpk, RI, rh') \rightarrow (w^{rh}, RI', rh)$**:** RA optionally receives a revocation handle rh' (for a previously revoked user) as input. (If the user is joining for the first time, the input rh' is $\perp$ and RA picks a fresh rh, which is returned to the user.) RA then does the following:

- Parse RI as $(a, \mathcal{M}, \sigma)$, rpk as vk, and rsk as (sgk, sk, m). Abort if $\mathsf{SIG.Verify}(vk, \sigma, (a, \mathcal{M})) = 0$ or if $rh' \notin \mathcal{RS}$.
- If $rh' = \perp$, pick a fresh revocation handle $rh' \leftarrow \mathcal{RS} \backslash \mathcal{RU}$.
- Let $rh = rh'$.
- Add rh to $\mathcal{RU}$.
- Update the accumulator value:
  [WL]: $(a, m, w^{rh}, \mathsf{upmsg}) \leftarrow \mathsf{ACC_P.Add}(sk, a, m, rh)$.
  [BL]: $(a, m, u^{rh}, \mathsf{upmsg}) \leftarrow \mathsf{ACC_N.Del}(sk, a, m, rh)$.
- Let $\mathcal{M} = \mathcal{M} \cup \{\mathsf{upmsg}\}$. $w^{rh} \leftarrow \mathsf{ACC_N.NonMemWitCreate}(\overline{sk}, \overline{a}, \overline{m}, rh)$.
- Sign the updated revocation information: $\sigma \leftarrow \mathsf{SIG.sign}(sgk, (a, \mathcal{M}))$.
- Let $RI = (a, \mathcal{M}, \sigma)$.
- Output w^{rh} and rh privately to the party that invoked the algorithm, and output RI publicly.

Figure 11. Join algorithm.

The RA is usually asked to execute the Join algorithm by the issuer. When issuing a revocable credential, the issuer requests a membership witness from the RA. If the user in question has previously joined - and subsequently been revoked from - the system, the issuer gives the RA her revocation handle rh. Otherwise, the RA generates a fresh revocation handle for the user. The issuer signs the revocation handle inside the credential, and sends the credential together with rh and the witness to the user.

If the Join algorithm changes the accumulator value, users need to update their witnesses to account for the additions. Therefore, the $\mathsf{ACC_P}$.MemWitUpOnAdd (in the whitelist approach) or $\mathsf{ACC_N}$.NonMemWitUpOnDel (in the blacklist approach) algorithms are used in ARC.RevTokenGen to bring the witness up to date.

Theorem 6. *The* ARC *extended with the* Join *algorithm (Figure 11) is correct, sound and private if the commitment scheme* Com *is hiding, the zero knowledge proof system* ZKP *is sound and zero knowledge, the accumulators* $\mathsf{ACC_P}$ *and* $\mathsf{ACC_N}$ *are adaptively sound, and the signature scheme* SIG *is existentially unforgeable.*

The proof is very similar to the one from the previous section and, therefore, omitted.

Join-Revoke Unlinkability. Before adding the Join algorithm, we did not have to worry about a user addition being linkable with a user revocation, because we had no user additions. Now that we do have a Join algorithm, though, this is a real concern; the revocation information could allow others to determine that the user revoked just now was the user who joined two hours ago, and not the user who joined four hours ago.

More formally, join-revoke unlinkability ensures that no adversary can determine which joining session a revocation corresponds to. This is similar to the blindness of blind signature schemes. The adversary should be unable to guess which user out of two has joined, even if it can choose the revocation handles itself, can arbitrarily join and revoke users, and can generate revocation tokens for all participants. Note, however, the revocation authority parameters must be generated honestly, and thus this definition does not imply privacy. We provide a formal definition for the join-revoke unlinkability in the full version of this paper [1].

Theorem 7. *The* ARC *extended with the* Join *algorithm and instantiated with* Braavos *is join-revoke unlinkable.*

Using ARC with Braavos precludes join-revoke linkability, because joins are not reflected at all in the revocation information.

5.2. Revocation for Anonymous Credentials Using Braavos: Performance Evaluation

In the previous section we discussed how one can construct an ARC using any dynamic accumulator that supports zero-knowledge proofs of member knowledge (as well as any secure commitment scheme and any existentially unforgeable signature scheme). In particular, the Braavos accumulator can be used, together with the zero knowledge mechanisms described in Section 4.4 and Fujisaki-Okamoto commitments. Using Braavos is especially efficient because during ARC.RevTokenGen, users only need to run MemWitUpOnDel, not MemWitUpOnAdd.

We now discuss the performance of a real anonymous credential system that uses an ARC with Braavos to support revocation. The system we have tested is idemix [10].[9] We left out key generation, as this is only relevant at setup once.

Idemix binds multiple user attributes (e.g., name, age, citizenship status, or employer) into a single credential. It

9. https://abc4trust.eu/idemix

allows a credential to be used in a number of ways, which are described in detail by Camenisch et al. [29]. Briefly, during credential presentation, a user can either simply prove that she has a valid credential ("proof of possession"), she can reveal one or more attributes ("opening", i.e., she can disclose that she is a citizen), or she can describe attributes in a range ("range proof", i.e., she can disclose that she is between 21 and 65 years old, without revealing her actual age).

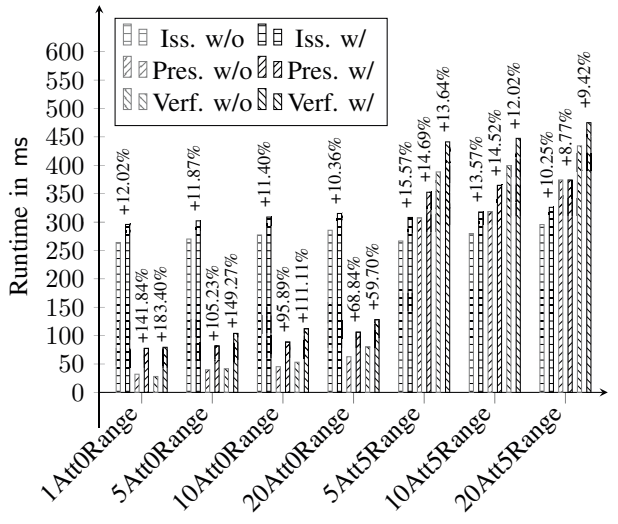

Figure 12. 1,024Bit Measurements

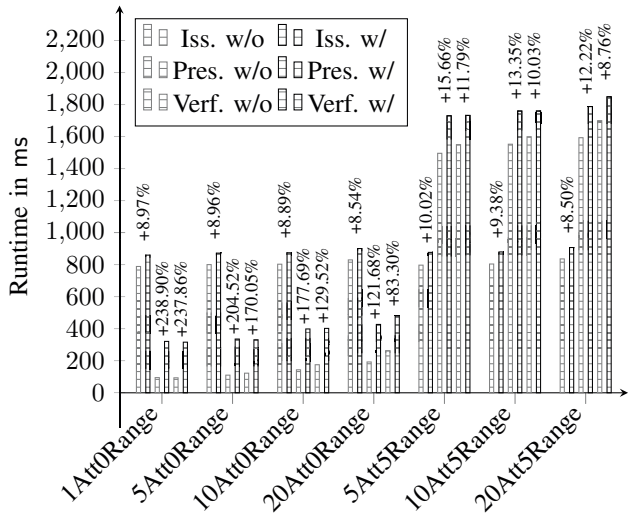

Figure 13. 2,048Bit Measurements

In Figures 12 and 13, we show the timings of credential issuance ("Iss."), presentation ("Pres.") and verification ("Verf."), with ("w/") and without ("w/o") the anonymous revocation component (ARC) with a 1024-bit and 2048-bit RSA modulus, respectively. We use credentials with 1, 5, 10, and 20 attributes. For all of those credentials, we measure presentation consisting of a simple proof of possession. For credentials with 5, 10, and 20 attributes, we also measure presentation consisting of 5 range proofs, to show that using our ARC in conjunction with more realistic presentation policies only adds a marginal amount of run-time. On the x-axis, xAtt$y$Range means that the credential has x attributes and y range proofs are performed during presentation.

The measurements were performed locally on a machine with a Intel Quad-Core CPU with 2.70GHz, 16GB of RAM and Java8u77. In total, 1,000 runs where taken and the figures represent the average run-time. We provide a table of the exact timings in the full version of this paper [1].

Conclusions. The revocation overhead for issuance is the same for all credential types, and is very small. For presentation and verification, the revocation overhead is only significant for simple proofs of possession with few attributes. In these cases, the absolute numbers are very small anyway, so the overhead is not a practical problem. We think that having more attributes inside a credential and more complex presentations (e.g., involving range proofs) is more realistic, and in the case of credentials with 20 attributes and 5 range proofs, the overhead for a presentation and verification is less than 10%. Thus, anonymity-preserving revocation using ARC with Braavos is practical, as it does not impact performance significantly. The advantages one gains by achieving revocability is clearly worth the small price of 10% runtime overhead.

6. Acknowledgements

Jan Camenisch, Maria Dubovitskaya, and Kai Samelin were supported by the European Research Council under grant agreement number 321310 (PERCY). The work of Foteini Baldimtsi, Leonid Reyzin, and Sophia Yakoubov was supported, in part, by US NSF grants 1012798, 1012910, and 1422965. Foteini Baldimtsi performed this work while at Boston University. Leonid Reyzin is grateful for the hospitality and support of IST Austria, where part of this work was performed. Anna Lysyanskaya's work was supported by US NSF grant 1422361.

References

[1] F. Baldimtsi, J. Camenisch, M. Dubovitskaya, A. Lysyanskaya, L. Reyzin, K. Samelin, and S. Yakoubov, "Accumulators with applications to anonymity-preserving revocation," Cryptology ePrint Archive, Report 2017/43, 2017, http://eprint.iacr.org/2017/043.

[2] J. Camenisch and A. Lysyanskaya, "Dynamic accumulators and application to efficient revocation of anonymous credentials," in *Advances in Cryptology – CRYPTO 2002*, ser. Lecture Notes in Computer Science, M. Yung, Ed., vol. 2442. Springer, Heidelberg, Aug. 2002, pp. 61–76.

[3] L. Nguyen, "Accumulators from bilinear pairings and applications," in *Topics in Cryptology – CT-RSA 2005*, ser. Lecture Notes in Computer Science, A. Menezes, Ed., vol. 3376. Springer, Heidelberg, Feb. 2005, pp. 275–292.

[4] J. Camenisch, M. Kohlweiss, and C. Soriente, "An accumulator based on bilinear maps and efficient revocation for anonymous credentials," in *PKC 2009: 12th International Conference on Theory and Practice of Public Key Cryptography*, ser. Lecture Notes in Computer Science, S. Jarecki and G. Tsudik, Eds., vol. 5443. Springer, Heidelberg, Mar. 2009, pp. 481–500.

[5] P. Camacho, A. Hevia, M. A. Kiwi, and R. Opazo, "Strong accumulators from collision-resistant hashing," in *ISC 2008: 11th International Conference on Information Security*, ser. Lecture Notes in Computer Science, T.-C. Wu, C.-L. Lei, V. Rijmen, and D.-T. Lee, Eds., vol. 5222. Springer, Heidelberg, Sep. 2008, pp. 471–486.

[6] J. C. Benaloh and M. de Mare, "One-way accumulators: A decentralized alternative to digital sinatures (extended abstract)," in *Advances in Cryptology – EUROCRYPT'93*, ser. Lecture Notes in Computer Science, T. Helleseth, Ed., vol. 765. Springer, Heidelberg, May 1994, pp. 274–285.

[7] J. Li, N. Li, and R. Xue, "Universal accumulators with efficient nonmembership proofs," in *ACNS 07: 5th International Conference on Applied Cryptography and Network Security*, ser. Lecture Notes in Computer Science, J. Katz and M. Yung, Eds., vol. 4521. Springer, Heidelberg, Jun. 2007, pp. 253–269.

[8] I. Damgård and N. Triandopoulos, "Supporting non-membership proofs with bilinear-map accumulators," Cryptology ePrint Archive, Report 2008/538, 2008, http://eprint.iacr.org/2008/538.

[9] M. H. Au, P. P. Tsang, W. Susilo, and Y. Mu, "Dynamic universal accumulators for DDH groups and their application to attribute-based anonymous credential systems," in *Topics in Cryptology – CT-RSA 2009*, ser. Lecture Notes in Computer Science, M. Fischlin, Ed., vol. 5473. Springer, Heidelberg, Apr. 2009, pp. 295–308.

[10] J. Camenisch and E. Van Herreweghen, "Design and implementation of the idemix anonymous credential system," in *ACM CCS 02: 9th Conference on Computer and Communications Security*, V. Atluri, Ed. ACM Press, Nov. 2002, pp. 21–30.

[11] J. Camenisch, S. Krenn, A. Lehmann, G. L. Mikkelsen, G. Neven, and M. Ø. Pedersen, "Formal treatment of privacy-enhancing credential systems," in *Selected Areas in Cryptography - SAC 2015 - 22nd International Conference, Sackville, NB, Canada, August 12-14, 2015, Revised Selected Papers*, ser. Lecture Notes in Computer Science, vol. 9566. Springer, 2015, pp. 3–24.

[12] Y. Liu, W. Tome, L. Zhang, D. R. Choffnes, D. Levin, B. M. Maggs, A. Mislove, A. Schulman, and C. Wilson, "An end-to-end measurement of certificate revocation in the web's PKI," in *Proceedings of the 2015 ACM Internet Measurement Conference, IMC 2015, Tokyo, Japan, October 28-30, 2015*. ACM, 2015, pp. 183–196.

[13] D. Derler, C. Hanser, and D. Slamanig, "Revisiting cryptographic accumulators, additional properties and relations to other primitives," in *Topics in Cryptology – CT-RSA 2015*, ser. Lecture Notes in Computer Science, K. Nyberg, Ed., vol. 9048. Springer, Heidelberg, Apr. 2015, pp. 127–144.

[14] L. Reyzin and S. Yakoubov, "Efficient asynchronous accumulators for distributed PKI," Cryptology ePrint Archive, Report 2015/718, 2015, http://eprint.iacr.org/2015/718.

[15] A. Shamir and Y. Tauman, "Improved online/offline signature schemes," in *Advances in Cryptology – CRYPTO 2001*, ser. Lecture Notes in Computer Science, J. Kilian, Ed., vol. 2139. Springer, Heidelberg, Aug. 2001, pp. 355–367.

[16] P. Camacho, "On the impossibility of batch update for cryptographic accumulators," Cryptology ePrint Archive, Report 2009/612, 2009, http://eprint.iacr.org/2009/612.

[17] J. Camenisch and A. Lysyanskaya, "A signature scheme with efficient protocols," in *SCN 02: 3rd International Conference on Security in Communication Networks*, ser. Lecture Notes in Computer Science, S. Cimato, C. Galdi, and G. Persiano, Eds., vol. 2576. Springer, Heidelberg, Sep. 2003, pp. 268–289.

[18] N. Bari and B. Pfitzmann, "Collision-free accumulators and fail-stop signature schemes without trees," in *Advances in Cryptology – EUROCRYPT'97*, ser. Lecture Notes in Computer Science, W. Fumy, Ed., vol. 1233. Springer, Heidelberg, May 1997, pp. 480–494.

[19] R. Gennaro, S. Halevi, and T. Rabin, "Secure hash-and-sign signatures without the random oracle," in *Advances in Cryptology – EUROCRYPT'99*, ser. Lecture Notes in Computer Science, J. Stern, Ed., vol. 1592. Springer, Heidelberg, May 1999, pp. 123–139.

[20] C. Cachin, S. Micali, and M. Stadler, "Computationally private information retrieval with polylogarithmic communication," in *Advances in Cryptology – EUROCRYPT'99*, ser. Lecture Notes in Computer Science, J. Stern, Ed., vol. 1592. Springer, Heidelberg, May 1999, pp. 402–414.

[21] S. Micali, M. O. Rabin, and S. P. Vadhan, "Verifiable random functions," in *40th Annual Symposium on Foundations of Computer Science*. IEEE Computer Society Press, Oct. 1999, pp. 120–130.

[22] J. Lapon, M. Kohlweiss, B. D. Decker, and V. Naessens, "Performance analysis of accumulator-based revocation mechanisms," in *Security and Privacy - Silver Linings in the Cloud - 25th IFIP TC-11 International Information Security Conference, SEC 2010, Held as Part of WCC 2010, Brisbane, Australia, September 20-23, 2010. Proceedings*, ser. IFIP Advances in Information and Communication Technology, vol. 330. Springer, 2010, pp. 289–301.

[23] D. Catalano and D. Fiore, "Vector commitments and their applications," in *PKC 2013: 16th International Conference on Theory and Practice of Public Key Cryptography*, ser. Lecture Notes in Computer Science, K. Kurosawa and G. Hanaoka, Eds., vol. 7778. Springer, Heidelberg, Feb. / Mar. 2013, pp. 55–72.

[24] D. Derler, C. Hanser, and D. Slamanig, "Revisiting cryptographic accumulators, additional properties and relations to other primitives," Cryptology ePrint Archive, Report 2015/087, 2015, http://eprint.iacr.org/2015/087.

[25] E. Ghosh, O. Ohrimenko, D. Papadopoulos, R. Tamassia, and N. Triandopoulos, "Zero-knowledge accumulators and set operations," Cryptology ePrint Archive, Report 2015/404, 2015, http://eprint.iacr.org/2015/404.

[26] J. Camenisch and M. Stadler, "Efficient group signature schemes for large groups (extended abstract)," in *Advances in Cryptology – CRYPTO'97*, ser. Lecture Notes in Computer Science, B. S. Kaliski Jr., Ed., vol. 1294. Springer, Heidelberg, Aug. 1997, pp. 410–424.

[27] E. Fujisaki and T. Okamoto, "Statistical zero knowledge protocols to prove modular polynomial relations," in *Advances in Cryptology – CRYPTO'97*, ser. Lecture Notes in Computer Science, B. S. Kaliski Jr., Ed., vol. 1294. Springer, Heidelberg, Aug. 1997, pp. 16–30.

[28] T. Nakanishi, H. Fujii, Y. Hira, and N. Funabiki, "Revocable group signature schemes with constant costs for signing and verifying," in *PKC 2009: 12th International Conference on Theory and Practice of Public Key Cryptography*, ser. Lecture Notes in Computer Science, S. Jarecki and G. Tsudik, Eds., vol. 5443. Springer, Heidelberg, Mar. 2009, pp. 463–480.

[29] J. Camenisch, M. Dubovitskaya, R. R. Enderlein, A. Lehmann, G. Neven, C. Paquin, and F. Preiss, "Concepts and languages for privacy-preserving attribute-based authentication," *J. Inf. Sec. Appl.*, vol. 19, no. 1, pp. 25–44, 2014. [Online]. Available: http://dx.doi.org/10.1016/j.jisa.2014.03.004