Fast Revocation of Attribute-Based Credentials for Both Users and Verifiers

Wouter Lueks, Gergely Alpár, Jaap- Henk Hoepman, Pim Vullers

Fast Revocation of Attribute-Based Credentials for Both Users and Verifiers*

Wouter Lueks, Gergely Alpár, Jaap-Henk Hoepman, and Pim Vullers

Radboud University, Nijmegen, The Netherlands
{lueks, gergely, jhh, pim}@cs.ru.nl

Abstract. Attribute-based credentials allow a user to prove properties about herself anonymously. Revoking such credentials, which requires singling them out, is hard because it is at odds with anonymity. All revocation schemes proposed to date either sacrifice anonymity altogether, require the parties to be online, or put high load on the user or the verifier. As a result, these schemes are either too complicated for low-powered devices like smart cards or they do not scale. We propose a new revocation scheme that has a very low computational cost for users and verifiers, and does not require users to process updates. We trade only a limited, but well-defined, amount of anonymity to make the first practical revocation scheme that is efficient at large scales and fast enough for smart cards.

1 Introduction

More and more governments are issuing electronic identity (eID) cards to their citizens [18,23,25]. These eID cards can be used both offline and online for secure authentication with the government and sometimes with other parties, like shops. Attribute-based credentials (ABCs) [9] are an emerging technology for implementing eID cards because of their flexibility and strong privacy guarantees, and because they can be fully implemented on smart cards [29]. Every credential contains attributes that the user can either reveal or keep hidden. Such attributes describe properties of a person, like her name and age. ABCs enable a range of scenarios from fully-identifying to fully-anonymous.[1] When using a credential fully anonymously (i.e., without revealing any identifying attributes), proper ABC technologies guarantee that the credential is unlinkable: it is not possible to connect multiple uses of the same credential.

When ABCs are applied, the carriers on which the credentials are stored (for example, smart cards) can be lost or stolen. In such cases, it is important that users can revoke

* An extendend version of this paper is available in the Cryptology ePrint Archive [21]. The work described in this paper has been supported under the ICT theme of the Cooperation Programme of the 7th Framework Programme of the European Commission, GA number 318424 (FutureID) and the research program Sentinels (www.sentinels.nl) as project 'Mobile IDM' (10522) and 'Revocable Privacy' (10532). Sentinels is being financed by Technology Foundation STW, the Netherlands Organization for Scientific Research (NWO), and the Dutch Ministry of Economic Affairs. This research is conducted within the Privacy and Identity Lab (PI.lab) and funded by SIDN.nl (http://www.sidn.nl).

[1] This is why we prefer the term 'attribute-based credentials' over the more traditional term 'anonymous credentials'.

these credentials to ensure that they can no longer be (ab)used. This is also necessary when the owner of the credential herself abuses it. Revocation may, in fact, happen often. As an example, the nationwide Belgian eID system's revocation list contains more than 375 000 credentials [8] for just over 10 million citizens. A practical revocation scheme must therefore efficiently deal with such large revocation lists.

Unfortunately, the unlinkability of ABCs precludes the use of standard, identity-based revocation. There exist many privacy-friendly revocation schemes, with different trade-offs in terms of efficiency (both for users and verifiers), connectivity requirements, and anonymity. It turns out to be hard to satisfy all of these simultaneously. In particular, all revocation schemes proposed so far suffer from at least one of the following two problems: (1) they rely on computationally powerful users, making the scheme unsuitable for smart cards, the obvious carrier for a national eID; or (2) they place a high load on verifiers, resulting in long transaction times.

The IRMA Project. This research is part of the ongoing research project "I Reveal My Attributes" (IRMA).[2] The goal of this project is to demonstrate the practicality of attribute-based credentials. We implemented the entire user-side of the credentials on a smart card [29]. In this paper we focus on this setting.

Our contribution. Our contribution is a new revocation scheme that has very low computational cost for users and verifiers alike; it is efficient even in the smart card setting, and can therefore be used in practice. We introduce the main idea in Section 2, introduce ABCs in Section 3, and describe the full scheme in Section 4. In our scheme, verifiers need only constant time on average to check revocation status, making it as fast as traditional non-anonymous revocation schemes. Furthermore, the users' computational overhead is small (and updates to reflect revocations are *not* necessary). Our scheme is based on *epochs* that divide time in short (configurable) intervals. Our scheme is unlinkable, except if the user uses her credential more than once per epoch at the same verifier. Our revocation scheme works with most credential schemes. As an example, we instantiate it for Idemix [15] in Section 5. We give pointers for implementing our scheme in practice, and give experimental results as evidence of feasibility of our scheme in Section 6. Finally, we review related work in Section 7 and conclude our paper in Section 8.

2 The Idea

Our scheme enables efficient and privacy-friendly revocation of credentials. As it resembles verifier-local revocation (VLR) schemes [1,4,6], we describe those first.

2.1 Verifier-local revocation

The setting is a cyclic group G with prime order q. Every credential encodes a random *revocation value* $r \in \mathbb{Z}_q$. If a credential has to be revoked, its revocation value r is added to the global revocation list RL. When the user shows the credential to a verifier, the

[2] See `http://www.irmacard.org`.

verifier needs to check whether the user's revocation value r appears on the revocation list RL. To facilitate this check without revealing r itself, the user chooses a random revocation generator $h \in_R G$, calculates the *revocation token* $R = h^r$, and sends

$$(h, R) \tag{1}$$

to the verifier during showing. The user also proves that the revocation value r embedded into R corresponds to credential she is showing. This proof depends on the type of credential—see Section 5 for an example. Each verifier holds a copy of the revocation list $RL = \{r_1, \ldots, r_k\}$. To check whether the credential is still valid, the verifier checks whether $h^{r_j} = R$ for each $r_j \in RL$ and rejects the credential if one of these equalities holds. This form of verifier-local revocation has some problems in practice:

1. Because the user chooses the revocation generator h at random, the work for the verifier increases linearly with the number of items on the revocation list. This quickly causes performance problems.
2. The scheme is not forward-secure. Once the verifier obtains a revocation value r_i, the verifier can link all past and future interactions involving this value, if it stores the tuples (h, R) from (1). Some solutions have been proposed to solve this problem—see Section 7—but they are not efficient enough for our purposes.

Our scheme addresses these two disadvantages.

2.2 Our scheme

We propose to split time into epochs and to use one generator per epoch and per verifier. This limits the user to one showing per verifier per epoch if she wants to remain unlinkable (which is not a problem when epochs are small) but makes revocation checking very fast for the verifier. The user uses the per-epoch per-verifier generator $g_{\varepsilon,V}$ to create the values in (1). In particular, she sends $R = g_{\varepsilon,V}^r$ to the verifier.

To check whether the credential is revoked the verifier does not need to know the raw revocation values. Instead, a semi-trusted party, the revocation authority (RA), can store these, and provide the verifier with a revocation list:

$$RL_{\varepsilon,V} = \{g_{\varepsilon,V}^{r_1}, \ldots, g_{\varepsilon,V}^{r_k}\}.$$

The credential is revoked if $R \in RL_{\varepsilon,V}$. This operation takes only $O(1)$ time on average using associative arrays. The average time complexity thus decreases from linear to constant in the length of the revocation list $RL_{\varepsilon,V}$. While some computation load shifts to the RA, the RA does no more work creating the list than a verifier in the VLR scheme does for *every* verification. Also, the verifier can no longer link transactions in different epochs since it does not have the bare revocation values.

Epochs and generators. The length of an epoch must be sufficiently short so that a user normally never shows her credential twice within the same epoch to the same verifier. If the generator is reused, the corresponding activities of the user become linkable.

The generators form an attack vector for a malicious adversary to link users' activities. It is not sufficient for the user to keep track of the generators she used before.

A malicious verifier could take one fixed generator $g_{\varepsilon,V}$, and then create a new one by picking a random exponent $x \in_R \mathbb{Z}_q$ and sending $g_{\varepsilon,V}^x$ to the user. All revocation tokens are then easily reduced to the same base $g_{\varepsilon,V}$, without the user ever seeing a similar generator.

To prevent this attack, users should calculate the generators themselves. The easiest method—and the one we propose here—is to use a hash function and let the generator $g_{\varepsilon,V}$ for a verifier V and epoch ε equal $H(\varepsilon \parallel V)$, where H is a hash function from the strings to the group G and the epoch ε is derived from the current time.

3 A primer on ABCs

Attribute-based credentials are a cryptographic alternative to traditional credentials like driver's licenses and passports. ABCs contain a set of attributes, typically encoded as numbers, that a user can selectively reveal to the verifier. Even when attributes are hidden, the verifier can still assess the validity of the credential.

A typical attribute-based credential scheme comprises the following three parties.

Issuer The issuer issues credentials to users. It ensures that the correct data are stored in the credential. A typical credential scheme has multiple issuers.

User The user holds a set of credentials, obtained from one or more issuers. She can disclose a (user defined) selection of attributes from any number of her credentials to a verifier to obtain a service.

Verifier The verifier, sometimes called relying party or service provider, checks that the credential is valid, the revealed attributes are as required, and the credential is not revoked. Based on the outcome, it may provide a service to the user.

When the credential scheme supports revocation, another party is present.

Revocation Authority The revocation authority is responsible for revoking credentials. It determines when to revoke, and stores all information necessary to do so. If necessary, it sends revocation information to users and verifiers.

Our scheme is independent of the choice of credential scheme, but we impose three restrictions on it:

1. The credential must be able to encode a revocation value r from a sufficiently large set.[3] This value can identify a credential if it is revoked. We use the notation $C(r)$ to denote a credential that contains the revocation value r. Depending on the type of credential, other attributes may be present.
2. The issuer should be able to issue a credential $C(r)$ without learning the revocation value r. Otherwise, the issuer can use it to trace credentials. Most credential schemes support blind issuing, which makes this possible.

[3] For simplicity, we focus on attribute-based credentials, but this is not strictly necessary. Any credential scheme that can encode the revocation value and that satisfies the second restriction can be used with our scheme. One example would be to use the user's private key as the revocation value.

3. The showing protocol must be extendible to provide the verifier with the revocation token $R = g_{\varepsilon,V}^{r}$ and a proof that R and $C(r)$ contain the same revocation value r. Fortunately, most credential schemes already rely on zero-knowledge proofs, and these can readily be extended to include the required proof of equality.

Furthermore, we assume that the credential scheme authenticates the verifiers. (This is without loss of generality. It is easy to add such a layer if it is missing.)

4 The full scheme

We now describe the full scheme. It expands on the intuition described in Section 2 by explicitly stating how the revocation authority (RA) operates and how it deals with verifiers. Section 6 shows how to implement this scheme.

The revocation authority runs the SetupRA algorithm once.

SetupRA(1^ℓ) This algorithm takes as input a security parameter 1^ℓ. It chooses a cyclic group G of prime order q with generator g such that the DDH problem is hard in G and q has ℓ bits. Furthermore, it picks a hash function $H : \{0,1\}^* \to G$ that maps strings onto this group. It outputs (G, g, q, H). These parameters are public and known to all other parties. The RA keeps track of the current epoch ε, which it initializes to 0, and the initially empty master revocation list MRL containing revoked credentials identified by their revocation values.

Users and verifiers run the algorithms SetupU and SetupV respectively.

SetupU() The user keeps track of the current epoch ε. She also stores sets $\mathcal{T}_C$ of the verifiers that she has shown credential C to in this epoch. Initially, $\mathcal{T}_C = \emptyset$.

SetupV() The verifier calls GetRevocationList to get an initial revocation list from the revocation authority—see below. It also keeps track of the current epoch ε.

At the beginning of a new epoch, all parties increase the current epoch ε by 1. In particular, we assume that all users know the current epoch.[4] At the start of a new epoch, users additionally clear the list $\mathcal{T}_C$ of verifiers that have seen credential C in this epoch. Every verifier V runs the GetRevocationList protocol with the revocation authority to get its revocation list for the current epoch.

GetRevocationList() This protocol is run between a verifier V and the revocation authority. The parties execute the following steps:

1. The verifier V authenticates itself to the revocation authority.[5]
2. The revocation authority
   (a) calculates the generator $g_{\varepsilon,V} = H(\varepsilon \parallel V) \in G$ for verifier V;
   (b) computes the sorted list $RL_{\varepsilon,V} = \mathsf{sort}(\{g_{\varepsilon,V}^{r} \mid r \in \mathsf{MRL}\})$; and
   (c) sends $RL_{\varepsilon,V}$ to verifier V.

[4] As we explain in Section 6.2, epochs are represented as time intervals. Users test their knowledge of the current time against this interval to make sure the interval is not in the past.

[5] The verifier reuses the authentication mechanism in the credential scheme.

Sorting the revocation lists $RL_{\varepsilon,V}$ ensures that unlinkability is preserved for all previous activities, even for revoked users (if $|\mathsf{MRL}| > 1$).[6]

To revoke a credential, the Revoke protocol is run with the revocation authority.

Revoke(r) When the revocation authority is asked to revoke a credential with revocation value r, it adds r to the master revocation list MRL.

In a deployed system, it is the RA's responsibility to decide whether to grant the revocation request. In Section 6.1 we discuss how a credential can be revoked in practice.

When showing a credential, the user and the verifier follow the ShowCredential protocol. Using this protocol, the user first authenticates the verifier. Then, she gives a revocation token to the verifier and proves that she has a corresponding credential. The verifier checks the validity of the credential and whether it has been revoked.

ShowCredential(C, V) This protocol is run between a user holding credential C and a verifier V. It proceeds as follows.

1. The verifier authenticates itself to the user. The user aborts if the authentication is unsuccessful or if $V \in \mathcal{T}_C$.
2. The user calculates the verifier (and epoch) specific generator $g_{\varepsilon,V} = H(\varepsilon \parallel V)$, and adds V to the list of seen verifiers $\mathcal{T}_C$.
3. The user sends its revocation token $R = g_{\varepsilon,V}^{r}$ to the verifier. Here, r is the revocation value encoded into the user's credential $C(r)$.
4. The user and the verifier run the normal showing protocol for the user's credential $C(r)$, but in addition the user proves that its revocation token R is well-formed, i.e., that the exponent r is the same as the revocation value encoded in the credential. Section 5 shows an example of such a proof for Idemix.
5. The verifier checks the validity of the credential and whether R is well-formed. Finally, it confirms that R is not on its revocation list $RL_{\varepsilon,V}$ for the current epoch. It aborts if any of these checks fail.

The list $\mathcal{T}_C$ and the epoch ε uniquely determine the generators that the user has used for credential C in this epoch. The checks above ensure that the user never reuses a generator. Also, the user always calculates the generators herself. This prevents the verifier from cheating with the generators.

Checking that $R \notin RL_{\varepsilon,V}$ can be done in constant time (on average) if the verifier processes the revocation list $RL_{\varepsilon,V}$ into an associative array. Some tricks help keep the size of the revocation lists manageable—see Section 6.5.

4.1 Sketch of the security of the scheme

A good revocation scheme needs to satisfy two properties: (1) non-revoked credentials are still unlinkable, and (2) a revoked credential is no longer usable. Our scheme satisfies these two properties in a reasonable security model.

To model unlinkability, the adversary can interact with credentials in the system as often as it wants, in any epoch and as any verifier. It can also revoke credentials. In the

[6] For this purpose, it suffices to sort on the representation of the elements. All that matters is that the order depends only on information in the list itself.

challenge phase, it picks a verifier, an epoch and two credentials i_0 and i_1 that were not revoked in any earlier epoch nor shown to it in the chosen epoch (since our scheme does not guarantee unlinkability when a credential is shown twice in one epoch to to the same verifier). It is shown one of these credentials; it must determine which one. In the full version of this paper, we show that, assuming the random oracle model and the hardness of the Diffie-Hellman problem, it is not possible to make this distinction [21]. Because credentials can be revoked in epochs past the challenge epoch, this implies forward security.

It is also not possible for an adversary to avoid revocation. The proof in step 4 of ShowCredential binds the revocation token to the revocation value in the credential. Since credentials cannot be forged (by assumption on the credential scheme), the adversary cannot change its revocation value, and hence cannot avoid revocation. A full security model and proof is in the full version of this paper [21].

5 Showing Protocol for Idemix

In this section, we give a brief overview of the Idemix [15] attribute-based credential system and how our revocation scheme can be incorporated into it to enable revocation without losing anonymity. We focus on the way our revocation scheme can be incorporated and omit some of the cryptographic details.

In Idemix, a credential is a Camenisch–Lysyanskaya [11] signature (A, e, v) on the block of messages consisting of the user's private key sk_U and the attributes $a_1, \ldots, a_L$. We can easily incorporate an extra attribute containing the revocation value r into the signature:

$$A \equiv \left(\frac{Z}{S^v \cdot R_K^{sk_U} \cdot R_R^r \cdot \prod_{i=1}^{L} R_i^{a_i}} \right)^{1/e} \pmod{n},$$

where the credential issuer's public key consists of the integers $Z, S, R_K, R_R, R_1, \ldots, R_L, n$. Both sk_U and r should be chosen from a large set. The construction of the signature guarantees that the user cannot change any of the values in the exponents. In the issuing protocol, the revocation value r and the private key sk_U should be hidden.

Given a block of messages sk_U, r and $a_1, \ldots, a_L$ the validity of the signature can be verified by checking that

$$Z \stackrel{?}{\equiv} A^e \cdot S^v \cdot R_K^{sk_U} \cdot R_R^r \cdot \prod_{i=1}^{L} R_i^{a_i} \pmod{n}.$$

When the signature is part of a credential scheme, some of these values can never be shown to the verifier as they would make the credential linkable. Instead, during verification the user uses the following two functions to show a credential anonymously. First, the user randomizes the signature to ensure unlinkability. Second, the user selectively discloses only those attributes appropriate for the application (the private key and the revocation value are never revealed).

A user randomizes the value A of a signature (A, e, v) as follows. If (A, e, v) is a valid signature on sk_U, r and $a_1, \ldots, a_L$, then $(\hat{A}, e, \hat{v})$ is also a valid signature where

$\hat{A} := A \cdot S^{-\varrho} \pmod{n}$, $\hat{v} := v + e\varrho$ for any randomly chosen ϱ (in some large interval). This does not yet provide unlinkability by itself—e remains unchanged—but the selective disclosure proof described below also hides the value e.

The selective disclosure protocol is a (non-interactive) zero-knowledge proof constructed by the user. Such a proof reveals a subset of the attributes determined by the index set $\mathcal{D}$ and proves that a (randomised) signature contains these attribute values. To make revocation possible, we also include a predicate that demonstrates that (a) the revocation token R was honestly computed using the generator $g_{\varepsilon,V}$ and (b) the revocation value r corresponds to this credential. The proof is as follows:[7]

$$PK\Big\{(e, \hat{v}, sk_U, r, (a_i)_{i\notin\mathcal{D}}) : Z \prod_{i\in\mathcal{D}} R_i^{-a_i} \equiv \hat{A}^e S^{\hat{v}} R_K^{sk_U} R_R^r \prod_{i\notin\mathcal{D}} R_i^{a_i} \pmod{n} \\ \wedge\ R = g_{\varepsilon,V}^r \text{ in } G\Big\}.$$

In the congruence above, all the exponents on the left-hand side are known to the verifier (selectively disclosed attributes $(a_i)_{i\in\mathcal{D}}$), while the exponents on the right-hand side remain hidden and the user only proves knowledge of them. The above proof realizes the user's side of steps 4 and 5 in the ShowCredential algorithm—see Section 4.

6 Implementation

We now address some implementation challenges when using our revocation scheme.

6.1 Obtaining revocation information

To revoke a credential, one needs to know its revocation value. However, this value also introduces a privacy risk: the party that stores it could revoke the credential and hence detect its use. Many revocation schemes suffer from this problem, see Section 7. We discuss three options for storing and using revocation values.

In all cases, the user herself generates the revocation value. The issuer will include this revocation value in the credential without ever learning its value.[8]

Option 1: only the user knows the revocation value. The privacy of the credential owner is best protected when only she knows the revocation value. When she loses her credential, she reveals the revocation value to the RA, who then uses this to revoke the credential. To get even better privacy, she can calculate the future revocation tokens herself. This is computationally intensive, but lowers the trust in the RA significantly.

When a smart card acts on behalf of the user, there is another difficulty to overcome: how does the user access the revocation values when the card is lost or stolen? We

[7] We use a simplified version of the Camenisch–Stadler notation [13] for zero-knowledge proofs of knowledge. Only the prover knows the values in front of ':', other values are also known by the verifier. We also omited the range proofs; see the Idemix specification [15] for details.

[8] The method for encoding these attributes is similar to the 'blind' encoding of the user's private key in a credential. Idemix and U-Prove support this.

propose to use a trusted terminal to print the (card-generated) revocation values (for example, as a QR code) when the credential is issued. The user can then store the revocation values separately from the card and use them to revoke credentials.

Option 2: a trusted third party stores revocation information. A second option is for a (possibly distributed) trusted third party (TTP) to store the revocation values. During issuance, the user creates not only a revocation value, but also a verifiable encryption [12] of the revocation value (for example, Idemix supports this). The user proves that the TTP can decrypt its revocation value, and the issuer verifies this proof before issuing the credential to the user. Hence, the user cannot avoid revocation later on.

In theory, this allows the TTP to revoke a credential when it is abused. However, the credentials do still provide anonymity, so it is not easy to pinpoint the abuser. This is where the final option comes in.

Option 3: revocation information is escrowed during showing. For the third option, the user escrows the revocation value during a showing proof using verifiable encryption. When abuse is detected, the ciphertext can be used to recover the revocation value and thus revoke the user.

The latter two solutions do reduce the efficiency of the underlying credential scheme, and introduce a lot of trust in the third party. In practice, one has to weigh whether it is better to accept some abuse or to decrease efficiency and privacy.

6.2 Instantiating epochs

To keep the protocol description simple, we assumed that all parties are aware of the current epoch. To achieve this, epochs are, in practice, based on time. The revocation authority determines the length of an epoch, by specifying its start time t_s and end time t_e, so the current epoch ε is modelled by the tuple $\varepsilon = (t_s, t_e)$.

In step 2 of the ShowCredential protocol, the user checks that $t_s \leq t \leq t_e$ where t is the current time. If this equation is not correct, the user aborts. In this way, users always use the correct generator.

Embedded devices The above description does not suffice for smart cards, our target platform, as they lack a built-in clock, and thus have no notion of time. Nevertheless, an embedded device must also be able to calculate the generators itself, to prevent a verifier from adversarially choosing them.

We propose the following solution, similar to the method used in Machine Readable Travel Documents, like the new European passport [7]. The embedded device keeps track of an estimate t^* of the current time. The estimate is always at or before the current time. Every time the embedded device interacts with a verifier, it

1. receives a description of the current epoch (t_s, t_e) signed by the RA;
2. confirms that the epoch (t_s, t_e) is possible given its time estimate t^* by checking that $t^* \leq t_e$ (this is done in step 1 of the ShowCredential protocol); and
3. updates its estimate $t^* \leftarrow \max(t_s, t^*)$ if the signature is valid.

The signature by the revocation authority on the epoch makes it impossible for verifiers to trick the device into creating a too futuristic estimate t^* of the current time.

6.3 How to choose the epochs

Epochs determine during what period a credential is linkable. Ideally, at most one showing happens at each verifier within an epoch. The period between two showings wildly differs among applications. For example, a citizen credential may be used only a couple of times a year for filing tax returns with the government, while it may be used weekly to prove having reached legal drinking age in a pub or a store. A credential for accessing an online newspaper subscription could even be used daily.

At the same time, computing revocation lists for every epoch can become computationally intensive and transferring uses bandwidth. Therefore, we propose not to have a global epoch, but instead create epochs per verifier. The length of the epoch should be chosen in such a way that no credential is normally reused within the epoch for that particular verifier.[9] Using time to instantiate epochs (as described in Section 6.2) allows us to use verifier-specific epochs easily.

6.4 Experiments

We did two experiments to prove the validity of our scheme: we estimated the performance impact on an existing smart card implementation and tested the impact on the revocation authority. As the extra work for the verifier is extremely small, we did not measure its overhead.

Fast smart card implementation We estimate the efficiency of this scheme based on the work by Vullers and Alpár [29] in the IRMA project. To assess the performance of the implementation, we compare it to its version without revocation. As described in Section 5, we add an extra attribute to every credential to hold the revocation value.

As group G, we use the quadratic residues modulo a 1024-bit safe prime (this is somewhat small, but matches the security level used in the implementation of Vullers and Alpár [29]). This group is cyclic and the DDH problem is hard. Furthermore, hashing onto this group is easy. It takes five 256-bit hash calculations (to get a statically uniformly random element) and a squaring (which can be precomputed as part of the revocation value). Calculating a 256-bit hash takes about 10 milliseconds. We estimate a total extra time of 390 milliseconds for including the revocation value as an attribute, generating the revocation token and adding the equality proof [26]. This is very practical. Since showing a credential takes 1.0–1.5 seconds, the overhead is limited too.

We did not implement the verification of the certificate for the epoch yet, but we believe that the cost of doing this to be approximately 150 milliseconds.

[9] Note that when a user *does* use her credential more often within the same epoch a lot of anonymity remains. The uses within this epoch are linkable, but they are still unlinkable to uses in other epochs or at other verifiers. In particular, this will usually not reveal the user's identity.

Fast revocation list calculation The main remaining burden of the revocation scheme is on the revocation authority, which has to generate revocation lists for all verifiers, and has to do so for each epoch. This can amount to a large number of exponentiations. However, the reader should be aware that the amount of work the revocation authority has to do per generator (i.e., per epoch and per verifier) equals the work that a verifier has to do for *every verification* in the standard VLR setting.

We implemented the calculation of the revocation list to confirm that this approach is valid. The efficiency of this calculation depends on the group. For the Idemix setting, we created a (non-optimized) implementation that calculates about 7 500 revocation tokens per second. However, one can do much better, as is shown by our optimized implementation for the ECC library by Bernstein et al. [2]. This implementation calculates about 50 000 revocations per second. These results show that even for large scale systems revocation lists can be generated sufficiently fast.

6.5 The size of a revocation list

At the start of every epoch, verifiers retrieve new revocation lists. It might seem that when the revocation lists are big, the storage and transfer costs become prohibitive. This is not the case. Since our scheme does not do group operations on revocation tokens, it suffices to store their hashes. Furthermore, if one is willing to accept a false positive rate of 10^{-7}, Bloom filters [3] reduce the storage requirements by another order of magnitude. This allows, for example, 250 000 elements to be stored in only 1 MiB.

7 Related Work

Revocation has been widely studied in the literature; we refer to, for example, Lapon et al. [17] for a nice overview of current revocation techniques for attribute-based (Idemix) credentials. Traditional revocation techniques, like CRLs and OCSPs, require credentials to have a unique identifier that is always visible to the verifier. A certificate revocation list (CRL) [14] is a list of revoked credential identifiers, published by the issuer. Alternatively, the verifier can ask the issuer if a credential is still valid using the Online Certificate Status Protocol (OCSP) [27]. Both situations require the credential to be recognizable, which is undesirable for ABCs. However, revocation is fast: there is no extra work required on the side of the user, and the verifier can test validity in constant time.

Domain-specific pseudonyms [5,15,16] only slightly improve the situation: instead of being globally linkable, different uses are only linkable by the same verifier, but not across different verifiers. We believe this still weakens the unlinkability too much.

We now focus our attention on solutions that do offer sufficient privacy guarantees for the user. Table 1 compares these schemes with our scheme and the CRL scheme. A digital accumulator is a constant-sized representation of a set of values. Every value in the accumulator comes with a witness, which enables efficient membership checks. Camenisch and Lysyanskaya [10] proposed an updatable accumulator that can be used for revocation. A credential is unrevoked as long as it appears on the whitelist, represented by the accumulator. Another approach is to accumulate revoked credentials to create a blacklist. A credential is unrevoked if it is not on this blacklist [19,24].

Table 1. We compare CRLs [14], accumulators [8,10,24], traditional VLR schemes [1,4,6], VLR schemes with backward unlinkability (VLR-BU) [22], blacklistable anonymous credentials (BLAC) [28], and our scheme. We compare the complexity of the operations and data transfers. A proving time of 1 means that it is constant, while a proving time of $|RL|$ means that it scales linearly with the size of the revocation list. Of all the constant-time proving schemes, the accumulator has the biggest overhead. Our scheme is the only privacy-friendly scheme that has constant-time proving and verification while users do not need to receive updates.

	CRL	Accumulators	VLR	VLR-BU	BLAC	Our scheme
User can be offline	✓	×	✓	✓	✓	✓
Data to verifier						
per epoch	$\|RL\|$	1	$\|RL\|$	$\|RL\|$	$\|RL\|$	$\|RL\|$
per update	1	1	1	1	1	1
Proving (time)	1	1	1	1	$\|RL\|$	1
Verifying (time)	1	1	$\|RL\|$	$\|RL\|$	$\|RL\|$	1
Security	-	+	+/-	+	+	+

Accumulators change. For whitelists, this is after an addition; for blacklists, this is after a revocation. Thus users need to receive updates (for schemes like Camenisch et al. [8], these updates are public and can be provided by the verifier) and process them, inducing extra load on carriers like smart cards. Additionally, the (non-)membership proofs are expensive. Lapon et al. [17] show an overhead of 300% in the showing protocol. Other schemes, like Libert et al. [20] are equally inefficient, making them impractical.

Where accumulators place the load on the users—who need to get new witnesses after revocations or additions—and the revocation authority—who needs to create those witnesses—verifier-local revocation (VLR) [1,4,6] places the majority of the load at the verifier. As we saw in Section 2, the verifier needs to do a check that is linear in the length of the revocation list, however, apart from sending the extra revocation token, the extra work for the user is minimal.

A downside of traditional VLR schemes is that once a user is revoked, all of its transactions (also past ones) become linkable. Nakanishi and Funabiki [22] proposed a VLR scheme that is backward unlinkable, like our scheme. Similar to our scheme, they create different revocation tokens per epoch, so that verifiers cannot use the revocation token for the current epoch and apply it to earlier ones. However, their scheme is still linear in the number of revoked users, and needs to perform a pairing operation per revoked user. This makes it less efficient than previous and our solutions. The security of their scheme hinges on the fact that the per-epoch revocation tokens are maintained by a trusted party. It thus requires the same trusted party as our scheme does.

Finally, blacklistable anonymous credentials (BLAC) [28] take a different approach to revocation: misbehaving users can be blacklisted without requiring a TTP to provide a revocation token. In every transaction, the user provides a ticket, similar to our revocation token, that is bound to the user. To blacklist a user, the verifier places this ticket on the blacklist. In the second step of the authentication, the user proves that her ticket

is not on the blacklist. The complexity of this proof is linear in the number of items on the blacklist, so this scheme places a high load on the user. Even if a user's credential is revoked, the verifier does not learn her identity, nor can the verifier trace her.

8 Discussion and Conclusion

Our revocation scheme is fast. It can be combined with ABC showing protocols and can be *fully* implemented on a smart card. It incurs minimal overhead, while at the same time the revocation check can be performed efficiently by the verifier. We created a security model for our scheme and proved that our scheme is forward secure as long as the revocation authority is trusted. The proofs are included in the full version of this paper [21]. We showed that we can remove this trust assumption when the users calculate the revocation tokens themselves.

To obtain this speedup, we traded some traceability, but with an appropriate choice of epoch length this should not be a problem in practice. The fact that this enables us to create a revocation system that is truly practical makes this a worthwhile trade-off.

We believe our scheme is a valuable contribution to making large scale attribute-based credentials possible. It would be interesting to investigate protocols that further reduce the trust assumption on the revocation authority.

References

1. Ateniese, G., Song, D.X., Tsudik, G.: Quasi-Efficient Revocation in Group Signatures. In: Financial Cryptography 2002. pp. 183–197. LNCS 2357, Springer (2003)
2. Bernstein, D.J., Duif, N., Lange, T., Schwabe, P., Yang, B.Y.: High-speed high-security signatures. Journal of Cryptographic Engineering 2(2), 77–89 (2012)
3. Bloom, B.H.: Space/Time Trade-offs in Hash Coding with Allowable Errors. Communications of the ACM 13(7), 422–426 (Jul 1970)
4. Boneh, D., Shacham, H.: Group Signatures with Verifier-Local Revocation. In: CCS 2004. pp. 168–177. ACM (2004)
5. Brands, S., Demuynck, L., Decker, B.D.: A Practical System for Globally Revoking the Unlinkable Pseudonyms of Unknown Users. In: Australasian Conf. on Information Security and Privacy (ACISP 2007). pp. 400–415. LNCS 4586, Springer (2007)
6. Brickell, E., Camenisch, J., Chen, L.: The DAA scheme in context. In: Mitchell, C.J. (ed.) Trusted Computing, Professional Applications of Computing, vol. 6, chap. 5, pp. 143–174. Institution of Electrical Engineers (2005)
7. BSI: Advanced security mechanisms for machine readable travel documents – extended access control (eac). Tech. Rep. TR-03110, Bundesamt für Sicherheit in der Informationstechnik (BSI), Bonn, Germany (2006)
8. Camenisch, J., Kohlweiss, M., Soriente, C.: An Accumulator Based on Bilinear Maps and Efficient Revocation for Anonymous Credentials. In: PKC 2009. pp. 481–500. LNCS 5443, Springer (2009)
9. Camenisch, J., Krontiris, I., Lehmann, A., Neven, G., Paquin, C., Rannenberg, K., Zwingelberg, H.: D2.1 Architecture for Attribute-based Credential Technologies. Tech. rep., ABC4Trust (2011)
10. Camenisch, J., Lysyanskaya, A.: Dynamic Accumulators and Application to Efficient Revocation of Anonymous Credentials. In: CRYPTO 2002. pp. 61–76. LNCS 2442, Springer (2002)

11. Camenisch, J., Lysyanskaya, A.: A Signature Scheme with Efficient Protocols. In: SCN 2002. pp. 268–289. LNCS 2576, Springer (2003)
12. Camenisch, J., Shoup, V.: Practical Verifiable Encryption and Decryption of Discrete Logarithms. In: CRYPTO 2003. pp. 126–144. LNCS 2729, Springer (2003)
13. Camenisch, J., Stadler, M.: Efficient Group Signature Schemes for Large Groups. In: CRYPTO 1997, pp. 410–424. LNCS 1294, Springer (1997)
14. Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., Polk, W.: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280 (Proposed Standard) (May 2008), updated by RFC 6818
15. IBM Research Zürich Security Team: Specification of the Identity Mixer cryptographic library, version 2.3.4. Tech. rep., IBM Research, Zürich (Feb 2012)
16. Kutylowski, M., Krzywiecki, L., Kubiak, P., Koza, M.: Restricted identification scheme and diffie-hellman linking problem. In: INTRUST 2011. pp. 221–238 (2011)
17. Lapon, J., Kohlweiss, M., de Decker, B., Naessens, V.: Analysis of Revocation Strategies for Anonymous Idemix Credentials. In: CMS 2011. pp. 3–17. LNCS 7025, Springer (2011)
18. Lehmann, A., Bichsel, P., Bichsel, P., Bruegger, B., Camenisch, J., Garcia, A.C., Gross, T., Gutwirth, A., Horsch, M., Houdeau, D., Hühnlein, D., Kamm, F.M., Krenn, S., Neven, G., Rodriguez, C.B., Schmölz, J., Bolliger, C.: Survey and Analysis of Existing eID and Credential Systems. Tech. Rep. Deliverable D32.1, FutureID (2013)
19. Li, J., Li, N., Xue, R.: Universal accumulators with efficient nonmembership proofs. In: ACNS 2007. pp. 253–269 (2007)
20. Libert, B., Peters, T., Yung, M.: Group signatures with almost-for-free revocation. In: CRYPTO 2012. pp. 571–589 (2012)
21. Lueks, W., Alpár, G., Hoepman, J.H., Vullers, P.: Fast revocation of attribute-based credentials for both users and verifiers. Cryptology ePrint Archive, Report 2015/237 (2015), `http://eprint.iacr.org/`
22. Nakanishi, T., Funabiki, N.: Verifier-Local Revocation Group Signature Schemes with Backward Unlinkability from Bilinear Maps. In: ASIACRYPT 2005. pp. 533–548. LNCS 3788, Springer (2005)
23. Naumann, I., Hogben, G.: Privacy features of European eID card specifications. Network Security 2008(8), 9–13 (2008)
24. Nguyen, L., Paquin, C.: U-prove designated-verifier accumulator revocation extension. Tech. Rep. MSR-TR-2014-85, Microsoft Research (June 2014)
25. OECD: National Strategies and Policies for Digital Identity Management in OECD Countries (2011)
26. De la Piedra, A., Hoepman, J.H., Vullers, P.: Towards a Full-Featured Implementation of Attribute Based Credentials on Smart Cards. In: CANS 2014. Springer (2014), (To appear)
27. Santesson, S., Myers, M., Ankney, R., Malpani, A., Galperin, S., Adams, C.: X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. RFC 6960 (Proposed Standard) (Jun 2013)
28. Tsang, P.P., Au, M.H., Kapadia, A., Smith, S.W.: Blacklistable Anonymous Credentials: Blocking Misbehaving Users without TTPs. In: CCS 2007. pp. 72–81. ACM (2007)
29. Vullers, P., Alpár, G.: Efficient Selective Disclosure on Smart Cards Using Idemix. In: IDMAN 2013. pp. 53–67. IFIP AICT 396 (2013)