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

Wouter Lueks[1✉], Gergely Alpár[1 2], Jaap-Henk Hoepman[1], and Pim Vullers[1]

[1] Digital Security, Institute for Computing and Information Sciences,
Radboud University, Toernooiveld 212, Nijmegen, The Netherlands
{lueks, gergely, jhh, pim}@cs.ru.nl
[2] Faculty of Management, Science & Technology,
Open University of the Netherlands, Valkenburgerweg 177, Heerlen, The Netherlands

**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 such as 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.

**Keywords:** revocation, attribute-based credentials, privacy-enhancing technologies, privacy, smart cards, efficiency

## 1 Introduction

More and more governments are issuing electronic identity (eID) cards to their citizens [25,32,34]. These eID cards can be used both offline and online for secure authentication with the government and sometimes with other parties, such as shops. Attribute-based credentials (ABCs) [10] are an emerging technology for implementing eID cards because of their flexibility and strong privacy guarantees, and because they can be fully

* This paper is the extended version of Lueks et al. [28]. In particular, it adds a security model and security proofs (see Section 5), a discussion about using multiple generators (see Section 6), and an extended implementation discussion (see Section 8). Furthermore, it now explicitly shows how the system—rather than a user herself—can revoke credentials. 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`).

implemented on smart cards [39]. Every credential contains attributes that the user can either reveal or keep hidden. Such attributes describe properties of a person, such as her name and age. ABCs enable a range of scenarios from fully-identifying to fully-anonymous.[3] 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 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 [9] 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).[4] 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 [39]. 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. We achieve this by generalizing the DAA (direct anonymous attestation) domain-specific pseudonyms. We model the security of our scheme and prove that our scheme is secure in this model—see Section 5. To mitigate the linkability within an epoch, we explore the idea of using multiple generators in Section 6. Our revocation scheme works with most credential schemes. As an example, we instantiate it for Idemix [21] in Section 7.

[3] This is why we prefer the term 'attribute-based credentials' over the more traditional term 'anonymous credentials'. The ABC4Trust project emphasizes the privacy aspects even more by using the term 'privacy-enhancing attribute-based credentials' [10].

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

In Section 8, we give pointers for implementing our scheme in practice, and give experimental results as evidence of the feasibility of our scheme. Finally, we review related work in Section 9 and conclude our paper in Section 10.

## 2 The idea

Our scheme enables efficient and privacy-friendly revocation of credentials. As our scheme resembles verifier-local revocation (VLR) schemes [1,4,7], we describe those first.

### 2.1 Verifier-local revocation

The mathematical 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 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

$$(h, R) = (h, h^r) \tag{1}$$

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 7 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 9—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 is a generalization of the direct anonymous attestation approach of creating domain specific pseudonyms based on a per-verifier generator, see Section 9. Using one per-epoch per-verifier generator 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 the raw revocation values of revoked users, and provide the verifier with a revocation list:

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

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 epoch and verifier specific generator is reused, the corresponding activities of the user become linkable (but only within that epoch and at that verifier).

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 $\varepsilon$ equal $H(\varepsilon \parallel V)$, where $H$ is a hash function from the strings to the group $G$ and the epoch $\varepsilon$ is derived from the current time.

*How to revoke a credential?* In this paper we consider two different approaches to revoke a credential. Revocation is either *user-initiated* or *system-initiated*. When revocation is always user-initiated, only the owner can revoke a credential, and will do so when the credential (carrier) is lost or stolen. When revocation can also be system-initiated, the system can revoke a credential when the credential's owner loses the right to use that credential, for example, when the owner does not pay for a subscription or when the owner abuses the credential. In our system, the choice boils down to the question: who knows the revocation value $r$?

When revocations are always user-initiated, only the owner needs to know the revocation value $r$. When her credential carrier is lost or stolen, she provides the revocation value $r$ to the revocation authority. The revocation authority can then use this value to revoke the corresponding credential.

When revocation can (also) be system-initiated, the revocation authority needs to be able to obtain the revocation value of revoked credentials credentials. To this end, we introduce a trusted *escrow agent* that stores all the revocation values of all credentials.

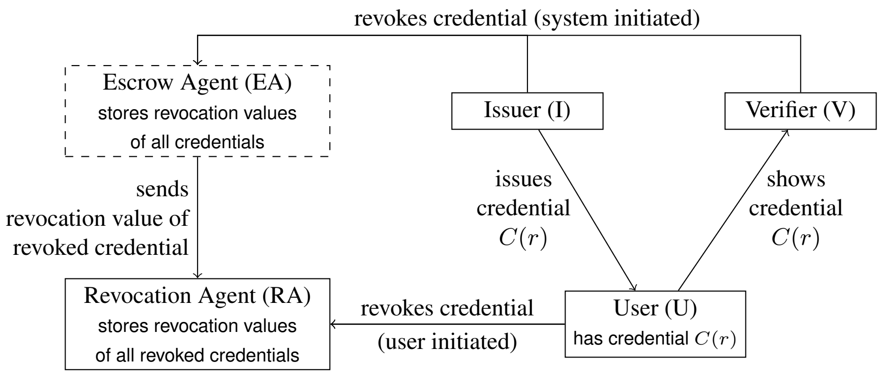
revokes credential (system initiated)
Escrow Agent (EA)
stores revocation values of all credentials
Issuer (I)
Verifier (V)
sends revocation value of revoked credential
issues credential $C(r)$
shows credential $C(r)$
Revocation Agent (RA)
stores revocation values of all revoked credentials
revokes credential (user initiated)
User (U)
has credential $C(r)$

**Fig. 1.** Interactions between parties in our revocation system. The escrow agent (EA) is only present when the system supports system-initiated revocation. There is typically only one revocation agent and one escrow agent, but there may be multiple issuers, verifiers and users.

Then, when a party (for example, an issuer or a verifier) wants to revoke a credential, that party provides evidence to the escrow agent why it wants to revoke that credential. If the escrow agent accepts the evidence, it looks up the corresponding revocation value and sends it to the revocation authority. The revocation authority will use the revocation value as before to revoke the credential. In this case, no cooperation of the user is required. Also see Figure 1 for how the parties interact.

The downside of the system-initiated approach is that the escrow agent has a lot of power: it can use the revocation values to link a user's actions, even if that credential is not yet revoked. This is why we consider both approaches in tandem.

## 3 A primer on ABCs

Attribute-based credentials are a cryptographic alternative to traditional credentials such as 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. If system-initiated revocation is supported, we assume that issuers internally assign an identifier $id$ to each credential that they issue. This allows issuers to later refer to credentials when they want to revoke them. (This identifier is of course never revealed to a verifier.)

**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.

In a credential scheme with revocation, a revocation authority and, if system-initiated revocation is supported, an escrow agent, are also present. Figure 1 shows how the parties interact.

**Revocation Authority** The revocation authority is responsible for revoking credentials. It revokes credentials on request of users and, when present, the escrow agent. The revocation agent keeps track of all the revoked credentials. If necessary, it sends revocation information to users and verifiers.

**Escrow Agent** The escrow agent can initiate the revocation of credentials upon request of users (when they lack the information to revoke their credentials directly), verifiers and issuers. The escrow agent stores all information necessary to fulfil this task. If it decides to grant a revocation request, it sends the required information to the revocation agent.

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.[5] This value can be used to identify a revoked credential. 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. Depending on whether the system supports system-initiated revocation, the revocation value is blindly supplied by the user or the escrow agent. Most credential schemes support the required forms of blind issuing. In Section 7 we show this for Idemix.
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.

Our paper focuses on credentials that are multi-show unlinkable, i.e., a verifier cannot link multiple showings of the same credential. When credentials are only single-show unlinkable the situation is different. Single-show unlinkable credentials are either used multiple times and are then naturally linkable (and hence easier to revoke) or used only once to remain unlinkable. Our methods also apply to this latter case of single-show credentials that are used only once. To compensate, a user has multiple version of the same credentials over the same attributes. If every version of the credential contains the same revocation value, then they can all be revoked simultaneously using our techniques.

[5] 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 (although in this case, extra care should be taken when using system-initiated revocation).

## 4 The full scheme

We now describe the full scheme. It expands on the intuition described in Section 2 by explicitly stating how the parties interact. Section 8 shows how to implement this scheme. Our scheme always supports user-initiated revocation. In addition, it can be configured to support system-initiated revocation.

The revocation authority runs the $\mathsf{SetupRA}$ algorithm once.

**SetupRA**$(1^\ell)$ This algorithm takes as input a security parameter $1^\ell$. 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 $\ell$ 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 $\varepsilon$, which it initializes to 0, and the initially empty master revocation list $\mathsf{MRL}$ containing revoked credentials identified by their revocation values.

If system-initiated revocation is supported, the escrow agent is present. The escrow agent runs the following setup algorithm.

**SetupEA**$()$ The escrow agent (EA) keeps track of a list $\mathsf{RV}$ containing a tuple $(id, r_{id})$ for each issued credential with identifier $id$ storing the corresponding revocation value $r_{id}$.

Users and verifiers run the algorithms $\mathsf{SetupU}$ and $\mathsf{SetupV}$ respectively.

**SetupU**$()$ The user keeps track of the current epoch $\varepsilon$. 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 $\mathsf{GetRevocationList}$ to get an initial revocation list from the revocation authority—see below. It also keeps track of the current epoch $\varepsilon$.

At the beginning of a new epoch, all parties increase the current epoch $\varepsilon$ by 1. In particular, we assume that all users know the current epoch.[6] At the start of a new epoch, users additionally clear the lists $\mathcal{T}_C$ of verifiers that have seen credential $C$ in this epoch. Every verifier $V$ runs the $\mathsf{GetRevocationList}$ 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 $V$ identifies itself to the revocation authority.
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 $V$.

[6] As we explain in Section 8.2, 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.

Sorting the revocation lists $RL_{\varepsilon,V}$ ensures that unlinkability is preserved for all previous activities, even for revoked users (if $|\mathsf{MRL}| > 1$).[7]

To obtain a credential, the user and the issuer follow the $\mathsf{ObtainCredential}$ protocol. This protocol describes two variants, one for user-initiated revocation (in which case the user generates the revocation value $r$), and one for system-initiated revocation (in which case the EA generates $r$).

**ObtainCredential**$(I)$ This protocol is run between a user $U$ wishing to obtain a credential, an issuer $I$, and, only if the system is configured to support system-initiated revocation, the escrow agent. The parties proceed as follows.

- If the system is not configured with system-initiated revocation, the user $U$ and issuer $I$ run the normal issuance protocol. As part of this protocol, the user generates its revocation value $r \in_R \mathbb{Z}_q$, on which the issuer blindly issues a credential $C(r)$. This credential might contain other attributes, but as we explained in Section 3, we omit these, as our focus is on the revocation scheme.
- If the system is configured with system-initiated revocation, the escrow agent generates the revocation value $r \in_R \mathbb{Z}_q$ for the credential, and sends it to the user. The user and the EA then jointly run the issuance protocol with the issuer. The EA blindly provides the revocation value $r$ (and the issuer checks that this value does indeed originate from the EA), while the user provides the other attributes (if any) as in the normal issuance protocol. Again, the user obtains the credential $C(r)$. Internally, the issuer assigns an identifier $id$ to the credential, provides this identifier to the AE, and stores a record of the issuance process. The EA adds the tuple $(id, r)$ to its list $\mathsf{RV}$.

Most credential schemes can be extended to support the second configuration form issuance where the RA determines the revocation value, in Section 7 we show how to do so for Idemix.

If the user wishes to revoke one of her credentials with revocation value $r$, she runs the following protocol.

**UserInitiatedRevoke**$(r)$ When the revocation authority is asked to revoke a credential with revocation value $r$, it adds $r$ to the master revocation list $\mathsf{MRL}$.

If, on the other hand, another party in the system wishes to revoke a credential they can run the following protocol with the escrow agent.

**SystemInitiatedRevoke**$(id, (R, \bar{g}), \texttt{evidence})$ The escrow agent can be asked to revoke a credential by supplying either the credential's identifier $id$ or a corresponding revocation token $R$ generated using generator $\bar{g}$. The caller additionally supplies some evidence `evidence`. The EA first examines `evidence` to determine if this is sufficient grounds for revocation. If the EA decides to grant the request, it recovers the revocation value $r_{id}$ as follows.

- If the credential identifier $id$ is supplied, it looks up the pair $(id, r_{id}) \in \mathsf{RV}$ to find $r_{id}$.

[7] 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.

  - If the tuple $(R, \bar{g})$ is supplied, the EA iterates over $(id, r_{id}) \in \mathsf{RV}$ to find the revocation value $r_{id}$ such that $\bar{g}^{r_{id}} = R$.

  If the AE cannot find the revocation value it returns an error. Otherwise, it makes a $\mathsf{UserInitiatedRevoke}(r_{id})$ request to the RA.

The idea of using the revocation token $R$ to identify the corresponding credential is identical to the tracing mechanism provided by Boneh and Shacham [4]. In Section 8.1 we discuss some extra options for revoking credentials in practice.

When showing a credential, the user and the verifier follow the ShowCredential protocol. The user takes as input the verifier $V$ and a credential. She sends the 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 by a user taking as input a credential $C$ and a verifier $V$. The user interacts with a verifier.[8] The protocol proceeds as follows.
1. The user immediately aborts if $V \in \mathcal{T}_C$ (before contacting $V$).
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 in zero-knowledge 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 7 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 $\varepsilon$ 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 8.5.

## 5 Security model and proofs

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. In this section,

[8] For our revocation scheme it is not necessary that the user authenticates the verifier, she relies on her own input of the verifier $V$ to determine the generator. However, for a credential scheme in general, it is essential to authenticate the verifier, lest the user reveal attributes that she did not want to reveal.

we prove that our scheme has these two properties, which we call unlinkability and unavoidability.

For both security definitions, we use credentials that are indistinguishable from one another. Most credential schemes provide this type of unlinkability, as long as the revealed attributes (if any) are the same. For unavoidability, we also require that credentials are unforgeable, but all credential schemes that we know of satisfy this property.

## 5.1 Unlinkability game

We say that a revocation scheme is unlinkable if no adversary can win the following game. This game is very similar to the anonymity games defined for group signatures [4,29] in general, and the backward unlinkable anonymity game of Nakanishi and Funabiki [31] in particular. Naturally, we added the restriction that the challenge credentials were not shown to the challenge verifier in the challenge epoch.

**Definition 1 (The unlinkability game).** *In the unlinkability game, the adversary's goal is to determine which of two credentials is shown to him. Let $\mathcal{S}$ be a credential scheme, $n$ the number of credentials in the system, and $\ell$ the security parameter.*

**Setup** *The challenger sets up the system by running the* SetupRA$(1^\ell)$ *algorithm on the RA and the* SetupEA$()$ *algorithm on the EA. It sets up the credential system $\mathcal{S}$ and initializes $n$ users using* SetupU$()$*. Then, it issues corresponding credentials $C_1, \ldots, C_n$ containing revocation values $r_1, \ldots, r_n$ using the* ObtainCredential *protocol to these users. Finally, it initializes the current epoch $\varepsilon$ to 0. The adversary is responsible for setting up the verifiers.*

**Queries** *The adversary may issue the following queries.*

- **CorruptCredential**$(i)$ *The adversary can request credential $C_i$ to be corrupted. It receives the revocation value $r_i$ and the entire internal state of the credential.*
- **Verify**$(V, i)$ *The adversary can request to act as verifier V for credential $C_i$ in the* ShowCredential$(C_i, V)$ *protocol.*
- **Revoke**$(i)$ *The adversary can ask to revoke credential $C_i$. The challenger calls the* UserInitiatedRevoke$(r_i)$ *protocol on the RA.*[9]
- **NextEpoch** *The adversary requests to move to the next epoch. The challenger ensures that the revocation authority and the users move to the next epoch and updates its own epoch $\varepsilon$ as well. The adversary is responsible for moving the epoch of the verifiers.*

*The verifiers under the adversary's control can always make* GetRevocationList *queries calls to the RA.*

**Challenge** *Let the current epoch be $\varepsilon^*$. The adversary selects two credentials with identifiers $i_0$ and $i_1$ and a verifier $V^*$ such that linking is not trivial, i.e.,*

1. *neither credential $i_0$ nor $i_1$ was revoked in $\varepsilon^*$ or earlier,*
2. *neither credential $i_0$ nor $i_1$ was corrupted, and*
3. *verifier $V^*$ did not verify the credentials $i_0$ and $i_1$ during epoch $\varepsilon^*$.*

[9] This oracle call also captures the abilities given by regular SystemInitiatedRevoke calls, as the adversary can always identify the credentials (except in the challenge phase).

*The challenger then picks a bit $b \in \{0,1\}$ at random and runs* Verify$(V^*, i_b)$ *(as defined above) with the adversary.*

**Restricted queries** *The adversary can make* CorruptCredential, Verify, Revoke *and* NextEpoch *queries as in the query phase. However, the adversary is not allowed to call* CorruptCredential *on the challenge credentials $i_0$ and $i_1$. Furthermore, during the challenge epoch $\varepsilon^*$, the adversary is not allowed to revoke the challenge credentials $i_0$ and $i_1$ nor make* Verify$(V^*, i_0)$ *and* Verify$(V^*, i_1)$ *queries on them.*

**Output** *The adversary outputs a bit $b'$. It wins if $b = b'$.*

*The advantage of an adversary $\mathcal{A}$ is given by $\mathbf{Adv}^{LINK}_{\mathcal{A},\mathcal{S}}(1^\ell) = 2|Pr[b = b'] - 1/2|$ where the probability is over the random bits of the challenger and the adversary. The revocation scheme for credential scheme $\mathcal{S}$ is unlinkable if $\mathbf{Adv}^{LINK}_{\mathcal{A},\mathcal{S}}(1^\ell)$ is negligible for every PPT algorithm $\mathcal{A}$.*

This game models the fact that the revocation authority is trusted; it will not give the raw revocation values to the adversary. The game also models the forward security of the scheme. The adversary is allowed to revoke the challenge credentials $i_0$ and $i_1$ in epochs beyond $\varepsilon^*$, but should nevertheless have a negligible advantage in distinguishing the credentials in epoch $\varepsilon^*$.

In the following reduction we let the revocation token $R$ of the challenge users depend on a DDH instance. As a result, we do not know its discrete logarithm, so we cannot create the equality proof required in the protocol. Instead, we require that we can forge this proof in the reduction. In most applications, this proof will be a non-interactive zero-knowledge proof resulting from the Fiat-Shamir heuristic [19]. In this case, our reduction can forge these proofs assuming a random oracle. (For regular zero-knowledge proofs we can use rewinding techniques.)

**Theorem 1.** *Our credential scheme with our revocation scheme is unlinkable (in the sense of Definition 1) in the random oracle model provided that the DDH problem is hard in the group $G$ and the underlying credential system is unlinkable.*

*Proof.* We reduce the security of our revocation scheme to the hardness of the DDH problem. We encode the DDH instance into the revocation token of two specific users. We do this in such a way that in the challenge epoch $\varepsilon^*$ we give the correct token if $c = ab$ and a random one otherwise. Any distinguisher thus breaks DDH.

More precisely, we construct a new game, which we call the random-challenge-token game, in which the revocation token $R = g_{\varepsilon^*,V^*}^{r_{i_b}}$ of the challenge user $i_b$ (picked by the challenger) is replaced by a random token $R \in_R \mathbb{Z}_q$. Our claim is that no adversary can distinguish the normal unlinkability game from this random-challenge-token game.

If this claim holds, we can replace the revocation token in the challenge of the regular indistinguishability game with a random token without being detected. Clearly, if the revocation token is random, the adversary has no extra information with respect to the regular unlinkability game for credentials. Therefore, since the credentials themselves are unlinkable, the adversary has, by definition, no chance of winning the random-challenge-token game. As a result, the adversary also cannot win the unlinkability game for credential schemes extended with our revocation mechanism.

We now prove that no adversary $\mathcal{A}$ can distinguish the normal unlinkability game from the random-challenge-token game. Suppose such an adversary $\mathcal{A}$ does exist. We will construct a challenger $\mathcal{B}$ that breaks the DDH assumption. As input, the challenger takes a DDH problem $(g, A = g^a, B = g^b, C = g^c)$, and $\mathcal{B}$'s task is to determine whether $c = ab$ or $c \in_R \mathbb{Z}_q$.

First, challenger $\mathcal{B}$ guesses the challenge epoch $\varepsilon^*$, the challenge verifier $V^*$ and the special credentials $i_0$ and $i_1$. Challenger $\mathcal{B}$ generates $n-2$ revocation values $r_i \in_R \mathbb{Z}_q$ for all $i \neq i_0, i_1$ and issues $n-2$ credentials $C(r_i)$ corresponding to these values. For credentials $i_0, i_1$, it picks random values $y_0, y_1, z_0, z_1 \in_R \mathbb{Z}_q$, and it will act as if credential $i_b$'s revocation value $r_{i_b}$ equals $a \cdot z_b$ for $b \in \{0,1\}$. It also creates corresponding credentials $C(y_0)$ and $C(y_1)$. It does not know $r_{i_0}$ or $r_{i_1}$ themselves, and since $y_b \neq r_{i_b}$, challenger $\mathcal{B}$ always needs to fake the equality proofs involving $C(y_b)$.

Note that the challenger can answer almost all queries honestly. The only changes that we make are how it answers hash queries, and how it answers the challenge query.

The challenger changes the random oracle $H$ to choose the generators $g_{\varepsilon,V}$. A judicious choice of the generators makes it possible to create the revocation token for credentials $i_0, i_1$, even though $a$ is unknown. For every epoch $\varepsilon$ and verifier $V$ the challenger chooses an exponent $x_{\varepsilon,V} \in_R \mathbb{Z}_q$. For verifier $V = V^*$ in epoch $\varepsilon = \varepsilon^*$ the challenger sets $g_{\varepsilon^*,V^*} = H(\varepsilon^* \parallel V^*) := B^{x_{\varepsilon^*,V^*}}$. For all other $\varepsilon$, $V$ pairs, it sets $g_{\varepsilon,V} = H(\varepsilon \parallel V) := g^{x_{\varepsilon,V}}$. The challenger runs the adversary $\mathcal{A}$, and honestly answers all queries not involving credentials $i_0, i_1$. For credential $i_b$ it creates the revocation token as

$$R_{\varepsilon,V,i_b} = g_{\varepsilon,V}^{r_{i_b}} = (g^{x_{\varepsilon,V}})^{a \cdot z_b} = (g^a)^{x_{\varepsilon,V} \cdot z_b} = A^{x_{\varepsilon,V} \cdot z_b}$$

unless $V = V^*$ and $\varepsilon = \varepsilon^*$. For $b \in \{0,1\}$, if the adversary ever makes a $\mathsf{Verify}(V^*, i_b)$ query in epoch $\varepsilon^*$, corrupts credential $i_b$, or revokes credential $i_b$ at or before epoch $\varepsilon^*$[10] challenger $\mathcal{B}$ aborts. In all cases, the challenger forges the proof of equality of the revocation value and credential $C(y_b)$ using its random oracle.

Eventually, the adversary makes its challenge query for credentials $\hat{i}_0, \hat{i}_1$ at verifier $V$ in epoch $\varepsilon$. If $\mathcal{B}$ did not guess these correctly, i.e., if $\varepsilon^* \neq \varepsilon$, $\{\hat{i}_0, \hat{i}_1\} \neq \{i_0, i_1\}$, or $V^* \neq V$, it aborts. Otherwise, it picks a bit $b \in_R \{0,1\}$ and answers with $R = C^{x_{\varepsilon^*,V^*} \cdot z_b}$ (and forges the corresponding equality proof). If $c = ab$ then $R$ belongs to credential $i_b$ (because then $C = B^a$) and if $c \in_R \mathbb{Z}_q$ then $R$ is random. If $\mathcal{A}$ indicates that it plays the normal unlinkability game, then $\mathcal{B}$ will answer that $c = ab$ and if $\mathcal{A}$ answers indicates the random-challenge-token game, then $\mathcal{B}$ will answer that $c \neq ab$. Any non-negligible advantage that $\mathcal{A}$ has in distinguishing the two games results in challenger $\mathcal{B}$ having a non-negligible advantage for solving the DDH problem. $\square$

## 5.2 Unavoidability game

A credential which has been revoked should no longer be usable in the $\mathsf{ShowCredential}$ protocol. This requirement is expressed by the unavoidability game. As this notion is very close to traceability in group signatures (see for example Manulis et al. [29]), we express it using a similar game.

[10] Even though we do not know $r_{i_b}$ we can still revoke credential $i^*$ in later epochs because we *can* calculate the revocation token $R_{\varepsilon,V,i^*}$ as shown.

The general idea of the game is that we revoke every credential that the adversary obtains. The goal of the adversary is to use these credentials to sidestep the revocation mechanism. The adversary has two options to obtain a credential: it can corrupt an honest user's credential or it can request a credential to be issued to him. In the former situation, the game models a call to $\mathsf{UserInitiatedRevoke}$ to model that the honest user revokes his credential. In the latter case, we *must* use system-initiated revocation (a malicious user cannot be trusted to voluntarily revoke its credentials), and so the game models a call to $\mathsf{SystemInitiatedRevoke}$. (As a direct consequence of this, if the system does not support system-initiated revocation, then the adversary is not allowed to obtain any new credentials in this game.)

**Definition 2 (The unavoidability game).** *In the unavoidability game, the adversary's goal is to convince the challenger's verifier that it has a valid and unrevoked credential while in fact all credentials he has* are *revoked. Let $\mathcal{S}$ be a credential system, $\ell$ the security parameter and $n$ the number of honest users' credentials in the system.*

**Setup** *The challenger sets up the system by running the $\mathsf{SetupRA}(1^\ell)$ algorithm on the RA and $\mathsf{SetupEA}()$ on the EA. It also sets up an issuer $I$ for credential system $\mathcal{S}$ to construct credentials and a verifier for its own use. The challenger sets up the honest users by running $\mathsf{SetupU}()$ for each of them. The challenger also runs $\mathsf{ObtainCredential}(I)$ on behalf of each honest user $i$, so that they obtain a credential $C_i$ with revocation value $r_i$. The adversary is responsible for setting up the users it controls.*

**Queries** *The adversary may issue the following queries.*

- **$\mathsf{CorruptCredential}(i)$** *The adversary can request the corruption of credential $C_i$. Then, $r_i$ is the revocation value corresponding to this credential. The challenger sends the complete credential $C_i$ to the adversary. Thereafter, the challenger calls $\mathsf{UserInitiatedRevoke}(r)$ to revoke the credential.*
- **$\mathsf{ObtainCredential}$** *The adversary can request a credential to be issued to a user it controls. To this end, the challenger lets the adversary run the system-initiated revocation variant of the $\mathsf{ObtainCredential}(I)$ protocol with the escrow agent and the issuer (the challenger controls the latter two entities). At the end of this protocol, the adversary's user now has a new credential $C(r)$. Immediately after, the challenger makes a $\mathsf{SystemInitiatedRevoke}(id, \perp, \perp)$ call to the EA to revoke this new credential.*
- **$\mathsf{Verify}(V, i)$** *The adversary can request to act as verifier $V$ for credential $C_i$ in the $\mathsf{ShowCredential}(C_i, V)$ protocol.*
- **$\mathsf{NextEpoch}$** *The adversary requests to move to the next epoch. The challenger ensures that the revocation authority, and the users it controls move to the next epoch and updates its own epoch $\varepsilon$ as well. The adversary is responsible for moving the epoch of the verifiers and the users it has created or corrupted.*

**Challenge** *The challenger sets its verifier $V$ to the current epoch, and subsequently runs the $\mathsf{GetRevocationList}$ algorithm to get the latest revocation list. The adversary will run the $\mathsf{ShowCredential}$ protocol with this verifier. The adversary wins if the verifier accepts.*

*The advantage of adversary $\mathcal{A}$ is given by $\mathbf{Adv}_{\mathcal{S}}^{AVOID}(1^\ell) = Pr[V \text{ accepts}]$ where the probability is over the random bits of the challenger and the adversary. The revocation* *scheme for credential scheme $\mathcal{S}$ is unavoidable if $\mathbf{Adv}_{\mathcal{S}}^{AVOID}(1^\ell)$ is negligible for every PPT algorithm $\mathcal{A}$.*

For simplicity we do not give the challenger the option to revoke credentials nor to obtain the revocation list since all this information is already encoded into its credentials.

**Theorem 2.** *Our revocation scheme is unavoidable (in the sense Definition 3) in the random oracle model provided that the underlying credential scheme is unforgeable.*

We first provide a sketch of this proof.

*Proof (sketch).* Suppose the challenger's verifier accepts the credential $C(r)$ that is shown by the adversary. Since credentials are unforgeable, the adversary obtained this credential using either a ObtainCredential or a CorruptCredential query, thus the embedded revocation token $r$ is on the master revocation list MRL. Let $g_V$ be the verifier's generator. The equality proof guarantees that the revocation token $R$ presented by the adversary is of the form $g_V^r$. Since $g_V^r$ is on the verifier's revocation list, the verifier will never accept the adversary's proof. $\square$

To give a more formal security proof, we first define the unforgeability of a credential scheme.

**Definition 3 (The unforgeability game).** *In the unforgeability game for a credential system $\mathcal{S}$, the adversary's goal is to show possession of a credential with attributes that were not previously issued by the issuer. Let $\ell$ be the security parameter and $L$ the number of attributes contained within each credential.*

**Setup** *The challenger sets up the system by setting up the credential system $\mathcal{S}$ and a corresponding issuer $I$.*
**Queries** *The adversary may issue the following queries.*
- **ObtainCredential**$(a_1, \dots, a_L)$ *The adversary can request a credential on the tuple of attributes $(a_1, \dots, a_L)$. The challenger engages in the issue protocol with the adversary to issue to the adversary a credential with these attributes. The challenger controls the issuer $I$.*
- **VerifyCredential**$(a_1, \dots, a_L)$ *The adversary can also request that a credential with attributes $(a_1, \dots, a_L)$ is shown to him. To do so, the challenger issues himself a credential $C(a_1, \dots, a_L)$, and runs the showing protocol for this credential with the adversary acting as verifier.*

**Challenge** *The challenger sets up a verifier $V$ and engages in the showing protocol with the adversary. Let $\mathcal{D} \subset \{1, \dots, L\}$ be the indices of the attributes that were disclosed by the adverary. The adversary wins if the verifier accepts and the adversary never made a ObtainCredential$(a'_1, \dots, a'_L)$ query where $a'_i = a_i$ for all $i \in \mathcal{D}$ (note that if $\mathcal{D} = \emptyset$, this implies that the adversary did not make any ObtainCredential queries).*

*The advantage of adversary $\mathcal{A}$ is given by $\mathbf{Adv}_{\mathcal{S}}^{FORGE}(1^\ell) = Pr[\mathcal{A}\ wins]$ where the probability is over the random bits of the challenger and the adversary. The credential scheme $\mathcal{S}$ is unforgeable if $\mathbf{Adv}_{\mathcal{S}}^{FORGE}(1^\ell)$ is negligible for every PPT algorithm $\mathcal{A}$.*

We can now formally prove Theorem 2.

*Proof (of Theorem 2).* Assume that an adversary $\mathcal{A}$ can break the unavoidability of our revocation scheme as applied to a credential system $\mathcal{S}$. We show how to construct an adversary $\mathcal{B}$ that breaks the unforgeability of credentials in $\mathcal{S}$.

Adversary $\mathcal{B}$ plays the role of challenger in the unavoidability game, and the role of adversary in the unforgeability game for the credential system. Adversary $\mathcal{B}$ runs the setup and query phases of the unavoidability game honestly. However, it does not construct its own issuer, instead it makes use of its oracle access to the unforgeability challenger. In particular:

- Adversary $\mathcal{B}$ picks revocation values $r_i$ for each honest user $i$ during the setup phase, but does not request credentials for them.
- Whenever the adversary $\mathcal{A}$ makes a $\mathsf{Verify}(V, i)$ query, adversary $\mathcal{B}$ makes use of its $\mathsf{VerifyCredential}(r_i)$ oracle to respond to the request.
- If adversary $\mathcal{A}$ makes a $\mathsf{CorruptCredential}(i)$ query, adversary first makes $\mathcal{B}$ makes an $\mathsf{ObtainCredential}(r_i)$ query to its oracles to obtain a credential $C(r_i)$. It then sends this credential to adversary $\mathcal{A}$. As part of the game, adversary $\mathcal{B}$ also makes a $\mathsf{UserInitiatedRevoke}(r_i)$ query.
- If adversary $\mathcal{A}$ makes a $\mathsf{ObtainCredential}$ query, adversary $\mathcal{B}$ generates a revocation value $r$ (acting as the EA) and makes a $\mathsf{ObtainCredential}(r)$ call to its challenger and relays the messages to $\mathcal{A}$ as in the normal system-initiated variant of the $\mathsf{ObtainCredential}$ protocol. Let $id$ be the identifier of this credential, then $\mathcal{B}$ makes a call $\mathsf{SystemInitiatedRevoke}(id)$ to revoke it.

During the challenge phase, adversary $\mathcal{A}$ runs the $\mathsf{ShowCredential}$ protocol with $\mathcal{B}$. Assume that $\mathcal{A}$ wins, i.e., $\mathcal{B}$ accepts the credential that is shown to him (if $\mathcal{A}$ looses, adversary $\mathcal{B}$ aborts). Since $\mathcal{B}$ accepts, the proof of knowledge produced by $\mathcal{A}$ shows that there is a revocation value $r$ such that (1) the credential $C(r)$ shown by $\mathcal{A}$ is valid and (2) the revocation token $R$ is of the form $R = g_{\varepsilon^*,V}^{r}$. Finally, since $\mathcal{B}$ accepts, $R \notin RL_{\varepsilon,V}$.

Since $R \notin RL_{\varepsilon,V}$ we know that $\mathcal{B}$ never made a request to issue a credential with $r$ as attribute (note that every time it did make a $\mathsf{ObtainCredential}$ request, the corresponding revocation value ended up on the $\mathsf{MRL}$ as a result of a subsequent revocation call). Hence, the credential $C(r)$ is a forgery.

To use this credential as a forgery, adversary $\mathcal{B}$ runs the knowledge extractor on the proof of knowledge of $\mathcal{A}$ to obtain the credential $C$ and the revocation value $r$. Adversary $\mathcal{B}$ then shows this credential, with the revocation value $r$ as part of the disclosed attributes, to its challenger. Since $r$ was never issued as part of a credential, $\mathcal{B}$ wins the unforgeability game.

To conclude, if an adversary breaks the unavoidability of the revocation scheme, then it must also break the unforgeability of the underlying credential system. □

## 6 Multiple generators

The single generator protocol we described above is secure and efficient. Yet, a user is linkable in an exceptional case: when she uses a credential multiple times for the same verifier within one epoch (which is then too long). Also, the load on the revocation authority can become quite high (it needs to create a revocation list for each verifier). In this section, we make a detour to explore the question whether multiple generators—shared among the verifiers[11], or even per verifier—alleviate these minor problems. The answer is positive, however, using multiple generators can make the user somewhat linkable (but, less linkable than if the user would have reused the single generator in the original scheme). Before we explain this linkability, we extend our scheme with multiple generators.

## 6.1 Multiple generators for revocation

We propose two methods for creating multiple generators: the global and the local. In the global method there are $m$ generators (per epoch) that are shared among the verifiers. In the local method there are $m$ generators (per epoch) for each verifier individually. In the latter case $m$ can be smaller than in the former.

Global generators ensure that the user has $m$ different generators to choose from. These can be spent at any verifier, several different generators can even be used at the same verifier. If $m$ is smaller than the number of verifiers, using global generators reduces the load on the revocation authority as well. Local generators give the user $m$ generators per verifier instead of only one.

Instead of generating per-epoch per-verifier generators, we now create the $i$th generator, with $1 \leq i \leq m$ as follows:

$$g_{\varepsilon,i} = \begin{cases} H(\varepsilon \parallel i) & \text{if mode is global} \\ H(\varepsilon \parallel V \parallel i) & \text{if mode is local.} \end{cases}$$

The verifier is implicit for local generators. The verifier can now request revocation lists for each of theses generators (for global generators the RA will cache the responses). During showing, a user randomly picks one of the unused generators (or aborts if she cannot). To this end, she keeps track of the indexes of generators used in this epoch for global generators, or pairs $(V, i)$ when she used the $i$th verifier of verifier $V$. She informs the verifier of her generator choice, and proves that she created her revocation token $R$ with respect to this generator.

This method was inspired by ideas for traceable signatures [16], where signatures can be traced because the signer can only produce a finite number of unique tags. When tracing a signature the tracing agent produces all these tags, much like we generate revocation tokens for all the generators. The traceable signature scheme does not suffer from the problem we describe next because the (inefficient) proofs of knowledge hide which generators are used.

## 6.2 Distinguishing credentials

The unlinkability game is easily extended to the above setting. We first note some positive results. For local generators with $m = 1$ we exactly have the same scheme as

[11] This approach using global generators is similar to the starting point taken in the revocation technique by Verheul [38].

before, with the same security requirements. Similarly, if two credentials have never been used in this epoch (at this verifier, for local generators), it can be shown that the adversary has no chance of linking them.

However, when a credential is used multiple times, before the challenge phase, the user creates an internal state—the generators that she has already used—that can be recognized by the verifier with a non-negligible probability. This attack works independent of the mode.

The verifier only needs two credentials $C_0$ and $C_1$ to have an advantage in the unlinkability game. It makes $m-1$ verify queries to $C_0$. It can observe which generators $C_0$ chooses, let $\tilde{g}$ be the generator it did *not* yet use. The adversary makes no queries to $C_1$. In the challenge phase it again requests credentials $C_0$ and $C_1$. If the credential uses generator $\tilde{g}$ the adversary guesses it is communicating with $C_0$, otherwise it guesses $C_1$. Since $C_0$ always uses $\tilde{g}$ and $C_1$ uses it with probability $1/m$ the adversary is correct with probability $1 - 1/(2m)$. Note that this attack does not work if $m = 1$. A similar attack works for any two credentials for which the internal state $\mathcal{T}_C$ differs.

While it is not nice to have credentials that are linkable in this way, the effect of this attack—that does not identify credentials directly, nor makes them fully linkable—may be acceptable to either significantly reduce the load on the RA for global generators, or allow the options of multiple authentications with the same verifier within an epoch at a small loss of privacy. (The original scheme makes the credentials fully linkable in this case.)

### 6.3 Making multiple generators work

The essential difficulty in the multiple-generator scheme we sketched above is that the user reveals the generator she used. This is not necessary. Instead, given a set of generators $g_{\varepsilon,1}, \ldots, g_{\varepsilon,m}$ the user with revocation value $r$ can make a zero-knowledge proof that

$$R = g_{\varepsilon,1}^r \vee R = g_{\varepsilon,2}^r \vee \cdots \vee R = g_{\varepsilon,m}^r.$$

The verifier then checks whether $R$ is on any of its revocation lists (corresponding to the $m$ generators. It can be shown that this variant is secure.

This zero-knowledge proof is not complicated, but it is computationally intensive for the user: its complexity is $O(m)$. However, it provides us with a trade-off between efficiency and perfect unlinkability. Moreover, when $m$ is small, we still outperform other fast solutions (like accumulators, see Section 9) without requiring updates to the user.

## 7 Issuing and showing protocols for Idemix

In this section, we give a brief overview of the Idemix [21] attribute-based credential system and how our revocation can be combined with it to enable revocation. We focus on the way our scheme can be incorporated and omit some of the cryptographic details.

In Idemix, a credential is a Camenisch–Lysyanskaya [13] signature $(A, e, v)$ on the block of messages consisting of the user's private key $sk_U$ and the attributes $a_1, \ldots, a_L$.

By the credential's mathematical construction, we can easily insert the revocation value $r$ as an extra attribute into the signature:

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

where the credential issuer's public key consists of the integers $Z, S, R_K, R_R, R_1, \ldots, R_L, n$. The issuer's private key is the prime factorization of $n$: $p, q$. Both $sk_U$ and $r$ are 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 from the issuer.

To incorporate the revocation mechanism, we describe first the way a credential is issued, then the way a credential is verified. As described in Section 4, the system can provide user-initiated and system-initiated revocation. In the former case the user embeds the revocation value in the credential, while in the latter case the escrow agent does. We discuss both options.

*Issuance in the case of user-initiated revocation.* In a system in which only the user knows the revocation value $r$, the revocation value is blindly signed by the issuer, in the same way that the issuer already signed the blinded secret key $sk_U$. In particular, the user commits to both $r$ and $sk_U$: $U := S^{v'} R_K^{sk_U} R_R^r \pmod{n}$, where $v'$ is a random integer from a large interval (for the exact size of the interval see the Idemix specification [21]). The user then sends the commitment $U$ to the issuer along with a zero-knowledge proof of knowledge of the exponents $v'$, $sk_U$ and $r$. Using this commitment, the issuer can create a transient signature $(A, e, v'')$ by computing

$$A = \left( \frac{Z}{U S^{v''} \prod_{i=1}^L R_i^{a_i}} \right)^{1/e \pmod{\varphi(n)}} \pmod{n},$$

where $e$ is a random prime and $v''$ is a random integer from some specified intervals. The issuer sends $(A, e, v'')$ along with a proof of knowledge of $1/e \pmod{\varphi(n)}$ to the user. Finally, the user constructs the signature $(A, e, v)$ by computing $v := v' + v''$.

*Issuance in the case of system-initiated revocation.* In the case of system-initiated revocation, the escrow agent generates the revocation value and supplies it to the issuer. Moreover, the EA and the user share the generation of the random value $v' = v'_{EA} + v'_U$. First, both the EA and the user create a commitment: the EA commits to the revocation value $r$ using $U_{EA} := S^{v'_{EA}} R_R^r \pmod{n}$, while the user commits to the secret key $sk_U$ using $U_U := S^{v'_U} R_K^{sk_U} \pmod{n}$. They send their commitments $U_{EA}$ and $U_U$ to the issuer together with the corresponding zero-knowledge proofs of knowledge. Furthermore, the EA sends the commitment $U_{EA}$ as well as the exponents $v'_{EA}, r$ to the user. Similarly to the other case, the issuer constructs the transient signature $(A, e, v'')$ by computing

$$A = \left( \frac{Z}{U_{EA} U_U S^{v''} \prod_{i=1}^L R_i^{a_i}} \right)^{1/e \pmod{\varphi(n)}} \pmod{n},$$

with random values $e, v''$. Finally, upon receiving $(A, e, v'')$ with the corresponding proof of knowledge, the user constructs the signature $(A, e, v)$ by computing $v := v'_{EA} + v'_U + v''$.

*Verification.* The Idemix credential verification is extended to prove that the credential is actually not revoked. 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

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

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 $(\hat{A}, e, \hat{v})$, where $\hat{A} := A \cdot S^{-\varrho} \pmod n$, $\hat{v} := v + e\varrho$ for any randomly chosen $\varrho$ (in some large interval), is also a valid signature on these values. 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:[12]

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

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.

## 8 Implementation

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

[12] We use a simplified version of the Camenisch–Stadler notation [15] 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 omitted the range proofs; see the Idemix specification [21] for details.

## 8.1 How to revoke a credential

In the preceding part of this paper we considered two options for revoking credentials: user-initiated and system-initiated. In this section we revisit these options.

To revoke a credential one needs to know its revocation value. However, this value also poses a privacy risk: the party that stores it could revoke the credential and hence detect its use. Many revocation schemes suffer from the same problem, see Section 9. Clearly, user-initiated revocation offers better privacy, as the user controls who she gives the revocation value to. However, she loses this control when she wants to revoke her credential.

**More privacy for user-initiated revocation** If the user does not trust the RA to only use her revocation value $r$ to revoke future uses of her credential, she can instead adopt the following approach. Rather than sending the raw revocation value $r$ to the RA, the user calculates the revocation tokens herself for all verifiers and all remaining epochs in which the credential is valid (assuming credentials expire)—she uses the fact that the generators can be calculated in advance.

This is costly, but does give forward-privacy for the user without trusting the RA. To reduce this cost, the RA can add structure to the generators of a single epoch as follows. It picks $z_{\varepsilon,V} \in_R \mathbb{Z}_q$ and sets $g_{\varepsilon,V} = H(\varepsilon)^{z_{\varepsilon,V}}$. The user only needs to do one exponentiation per epoch—she calculates $H(\varepsilon)^r$—and the RA creates the per-verifier specific values (as $R_V^r = (H(\varepsilon)^r)^{z_{\varepsilon,V}}$). The user is no longer able to check the verifier specific generators herself (she does not know $z_{\varepsilon,V}$), instead, the RA issues certificates on the verifiers' generators.[13] To ensure that the user does not reuse generators (verifiers might collude and provide each other's signed generators to the user), the user stores the used, verified generators instead.

**Managing revocation values** If the system uses only user-initiated revocation, users need to store their revocation values so that they are available in case the credential needs to be revoked. This is not the case when the credentials (and thus the revocation values) are stored on a smart card. So, when the card is lost or stolen, the revocation values needed to revoke the credentials need to be available elsewhere. One option would be to use a trusted terminal to print the (card-generated) revocation values (for example as a QR code) when a credential is issued. The user can then store the revocation values separately from the card. Note that if system-initiated revocation is enabled, the user can always use that avenue to revoke her own credentials.

**Escrowing revocation information** An alternative to storing all revocation values at the escrow agent, is to escrow the revocation information during the showing protocol. Similar to identity escrow [22], the user provides a verifiable encryption of her revocation value (encrypted with the public key of the escrow agent) to the verifier. While

[13] To prevent the RA from being able to link credentials across epochs (this specific construction of the generators already allow it to do so within epochs), the certificate should also include a proof that the RA knows $z_{\varepsilon,V}$ such that $g_{\varepsilon,V} = H(\varepsilon)^{z_{\varepsilon,V}}$.

this does not protect against lost or stolen credentials, it does allow verifiers to ask for revocation of a credential when it can present sufficient grounds to do so. Since the user attaches the encrypted revocation value to each showing, the escrow agent does not need to store the revocation values anymore.

However, the escrow agent can still identify a user based on the encrypted revocation value, just like it can for our system-initiated revocation approach by checking the revocation token against the list of revocation values. Therefore, we do not gain any security, and lose efficiency, because verifiable encryption is computationally intensive.

Once again, using an escrow agent trades privacy of the user for a better security of the system as a whole. It depends on the application of the system whether trusting the escrow agent is better than simply allowing some abuse.

## 8.2 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 $\varepsilon$ 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, such as the new European passport [8]. 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.

## 8.3 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 the lists 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.[14] Using time to instantiate epochs (as described in Section 8.2) allows us to use verifier-specific epochs easily.

## 8.4 Experiments

We did two experiments to prove the validity of our scheme: we estimated the performance impact on our 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 [39] in the IRMA project. To assess the performance of the implementation, we compare it to its version without revocation. As described in Section 7, we add an extra attribute to every credential to hold the revocation value.

As group $G$, we use a subgroup of $\mathbb{Z}_p$ of prime order $q$.[15] Here $p$ is a 1024-bit prime such that $p - 1 = bq$. This choice of $p$ is somewhat small, but matches the security level used in the implementation of Vullers and Alpár [39]. The group $G$ is cyclic and the DDH problem is hard. Furthermore, hashing onto this group is rather easy. It takes five 256-bit hash calculations to get a (statically uniformly) random element in $\mathbb{Z}_p$ and one exponentiation to the power $b$, the cofactor, to obtain an element in $G$. The exponentiation 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 [35]. This is very practical. Since showing a credential takes 1.0–1.5 seconds, the overhead is limited too.

We estimate that the cost of verifying the epoch certificate (recall that this certificate is necessary to correctly keep track of time on a smart card) is approximately 150 milliseconds.

**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.

[14] 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.

[15] In a previous version of this paper we used the quadratic residues group, but in that group it is easier to calculate discrete logarithms if the factorisation of the modulus is known.

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.

Idemix [11,21] uses a modular arithmetic setting for their credential scheme. One option is to reuse this setting to create a cyclic group $G$, as described in the previous section. We created a (non-optimized) test application, built using the GMP big number library[16] to get an estimate for the time required to build the revocation list. Our application calculates approximately 7 500 revocation tokens per second on a single core of a first generation mobile Intel Core i7 at 2.66 GHz. This is already acceptable in a system with a small number of users and service providers, for example with 450 service providers and 10 000 revoked users all lists can be generated in just 10 minutes.

However, nothing prevents us from choosing a more efficient group. It does not matter for the proof of knowledge. The only impediment might be that the smart card may not support this group. For reference, we also created an optimized implementation using the ECC library by Bernstein et al. [2]. The authors of this library already went to great lengths to create fast exponentiation for a fixed generator. We extended this library somewhat to also support dynamic generators (and do the pre-computation on the fly). This implementation performs about 50 000 exponentiations per second, on a single core 2.53 GHz machine. This should be fast enough for even nationwide deployment of the system.

There is one technicality that one has to take care of when using an ECC library such as the one by Bernstein et al. Often, points are represented internally in projective coordinates. This saves an expensive inversion operation in the underlying field. However, it also means that points do not have a unique representation. Such a unique representation is, however, essential to our fast revocation check. We normalize the representation by using Montgomery's trick [30] to calculate the inverses. By using this trick we only require 1 inversion and $3n$ multiplications to calculate $n$ inverses. This causes a significant speedup over the naive approach of normalizing each element separately. The cost of inversion has been taken into account in the performance measures given above.

The specific curve we used above is generally not available on smart cards, but other curves are; see for example Hein et al. [20]. Finally, our results with the optimized ECC library suggest that also in the modular arithmetic setting serious improvements in speed can still be obtained.

### 8.5 The size of a revocation list

Our scheme requires the distribution of revocation lists. It might seem that when the revocation list contains many items, the size of these revocation lists could become prohibitive. We will show that this is not the case.

Throughout, let $\nu$ be the number of items on the revocation list. The list contains group elements, therefore their size depends on the group. We consider two types of cyclic groups, both of prime order $q$ of about 256-bits. We follow the 2012 ECRYPT advisory [18] in selecting sizes that give long term protection.

[16] The GNU multiple precision big number arithmetic library: `http://gmplib.org/`.

– A cyclic subgroup of the integers modulo a prime $p$. For a group order of 256-bits, $p$ itself needs to have 3248-bits. A group element is thus 406 bytes.
– An elliptic curve group of order $q$ such that $q$ is about 256 bits. Only the $x$ coordinate and one bit for the $y$ coordinate need to be stored. Thus a group element takes about 32 bytes to represent.

Table 1 compares the storage requirements for a single revocation list, for $\nu = 2^{15} = 32\,768$, $\nu = 2^{18} = 262\,144$ and $\nu = 2^{21} = 2\,097\,152$ elements. We see that especially for the integers modulo $p$ the storage requirements are considerable.

For traditional $O(1)$ access structures, the verifier needs to store at least the entire list itself, and additionally some overhead. Since we only test membership of the revocation list, and do not calculate with the elements on the revocation list, it suffices to store the hashes of the elements. This reduces the storage requirements immediately, see Table 1. However, if we accept a very small error probability, we can do better by applying Bloom filters.

**Bloom filters** Bloom filters are a probabilistic data structure that can very efficiently store revocation tokens, at a constant number of bits per item, independent of the size of the item itself [3]. The number of bits per item is so small that a Bloom filter gives a one or two orders of magnitude improvement over storing the elements directly.

This increased efficiency comes at a price: the filter can give false positives, i.e., it can claim that an element is on the revocation list, while in fact it is not. However, the false positive rate can be made small. We think that in this setting a small (in the order of $10^{-6}$) false positive rate is acceptable for two reasons. One, in a practical system the error probability due to other means (such as intermittent connections and user error) is probably much higher, and two, when sufficient generators are available the user can easily retry with a fresh one (the probability that both fail is extremely small). In fact, we might even accept a higher false positive rate if the verifier can always fall back to an online check with the RA (that then does need to store the full list).

A Bloom filter is constructed as follows. It consists of a bit array of length $\kappa$ together with $\lambda$ hash functions $H_i$ that map strings into indices of this array, i.e., they map into the range $\{1, \ldots, \kappa\}$. To store an item $m$ in the filter, calculate $H_1(m), \ldots, H_\lambda(m)$, and set those bits in the array to one. To check if an item $m$ is on the list, calculate $H_1(m), \ldots, H_\lambda(m)$. If all these indices are set, the item is most likely in the filter.

It can be shown that the probability $P$ of a false positive for a Bloom filter storing $\nu$ items is given by

$$P \approx \left(1 - e^{\frac{-\lambda\nu}{\kappa}}\right)^{\lambda} .$$

This probability is minimal for $\lambda = \ln(2)\kappa/\nu$. Table 1 shows that a Bloom filter uses at least an order of magnitude less storage than a traditional solution at acceptable false positive rates.

The number of hash function calls is small too. The biggest filter with $\kappa/\nu = 32$ and $\nu = 2^{21}$ contains $2^{26}$ items. We need $\lambda = \lfloor \ln(2)32 \rfloor = 22$ hash functions with a 26-bit output. If we make one SHA256 call, we get 256 bits, therefore we need to make 3 SHA256 calls (with appropriate padding to get different hash functions) for every

**Table 1.** Comparison of storage requirements for a single revocation list. We consider different sizes of the revocation list, and two types of groups: the integers modulo $p$, with elements of 406 bytes and an elliptic curve, with elements of 32 bytes. Verifiers in our scheme only do membership tests with the revocation list, so instead of storing the elements themselves it suffices to hash them (with SHA256 in this case), or to store them in a Bloom filter. The data are parameterized by the false positive probability $P$ of the Bloom filter (based on $\lambda = \lfloor \ln(2)\kappa/\nu \rfloor$ hash functions), the length $\kappa$ of the filter and $\nu$ the number of revoked items.

| | Nr. of revoked items ($\nu$) | | |
|---|---|---|---|
| | $2^{15}$ | $2^{18}$ | $2^{21}$ |
| Integers modulo $p$ | 13 MiB | 102 MiB | 812 MiB |
| Elliptic curve | 1 MiB | 8 MiB | 64 MiB |
| Hashes of elements | 1 MiB | 8 MiB | 64 MiB |
| Bloom filter | | | |
| $P = 4.6 \cdot 10^{-4}, \kappa/\nu = 16$ | 64 KiB | 512 KiB | 4 MiB |
| $P = 9.9 \cdot 10^{-6}, \kappa/\nu = 24$ | 96 KiB | 768 KiB | 6 MiB |
| $P = 2.1 \cdot 10^{-7}, \kappa/\nu = 32$ | 128 KiB | 1 MiB | 8 MiB |

item.

## 9 Related work

Revocation has been widely studied in the literature; we refer to, for example, Lapon et al. [24] for a nice overview of current revocation techniques for attribute-based (Idemix) credentials. Traditional revocation techniques, such as CRLs and OCSPs, require credentials to have a unique identifier that is always visible to the verifier. A certificate revocation list (CRL) [17] 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) [36]. 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 [5,21,23] 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.

A final trick would be to use verifiable encryptions, see for example Camenisch and Shoup [14], to encode the revocation information. A trusted third party can then decrypt the ciphertext and check whether the credential has been revoked. Clearly this party is privy to too much information, and such a solution should thus be avoided.

We now focus our attention on solutions that do offer sufficient privacy guarantees for the user. Table 2 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

**Table 2.** We compare CRLs [17], accumulators [9,12,33], traditional VLR schemes [1,4,7], VLR schemes with backward unlinkability (VLR-BU) [31], blacklistable anonymous credentials (BLAC) [37], and our scheme. We compare the complexity of the operations and data transfers. We consider the transfer of data per epoch, as well as the extra data that needs to be send to update the verifier to include one extra revocation. 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 | ✓ | $\times$ | ✓ | ✓ | ✓ | ✓ |
| Data to verifier | | | | | | |
| per epoch | $\lvert RL\rvert$ | 1 | $\lvert RL\rvert$ | $\lvert RL\rvert$ | $\lvert RL\rvert$ | $\lvert RL\rvert$ |
| per update | 1 | 1 | 1 | 1 | 1 | 1 |
| Proving (time) | 1 | 1 | 1 | 1 | $\lvert RL\rvert$ | 1 |
| Verifying (time) | 1 | 1 | $\lvert RL\rvert$ | $\lvert RL\rvert$ | $\lvert RL\rvert$ | 1 |
| Security | - | + | +/- | + | + | + |

the accumulator comes with a witness, which enables efficient membership checks. Camenisch and Lysyanskaya [12] 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 [26,33].

Accumulators change. For whitelists, this is after an addition; for blacklists, this is after a revocation. Thus users need to receive updates (for schemes such as Camenisch et al. [9], these updates are public and can be provided by the verifier) and process them, inducing extra load on carriers such as smart cards. Additionally, the (non-)membership proofs are expensive. Lapon et al. [24] show an overhead of 300% in the showing protocol. Other schemes, such as Libert et al. [27] 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) [1,4,7] 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 [31] proposed a VLR scheme that is backward unlinkable, such as 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 solutions as well as our solution.

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.

Direct anonymous attestation (DAA) [6,7] uses the same technique as Boneh and Shacham's VLR scheme to revoke tags—that is, a tag creates a tuple $(h, h^{sk_U})$ for a random generator $h$, a rogue tag's tuples can then be recognized because its private key $sk_U$ is known, see Section 2. In DAA this same mechanism is also used to derive domain-specific pseudonyms. To do so, the tag creates the generator $h$ as the hash of the verifier's basename. We generalize this by creating a generator not only based on the verifier's identity, but also on the current epoch. Choosing generators in this fashion drastically limits the linkability that a user would otherwise incur, also see above.

Finally, blacklistable anonymous credentials (BLAC) [37] 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.

## 10 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. We showed that we can remove this trust assumption when the users calculate the revocation tokens themselves. Finally, we showed that by using multiple generators we can even limit the linkability within an epoch.

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.F., Camenisch, J., Chen, L.: Direct anonymous attestation. In: Atluri, V., Pfitzmann, B., McDaniel, P.D. (eds.) Proceedings of the 11th ACM Conference on Computer and Communications Security, CCS 2004, Washington, DC, USA, October 25-29, 2004. pp. 132–145. ACM (2004), `http://doi.acm.org/10.1145/1030083.1030103`
7. 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)
8. 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)
9. 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)
10. 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)
11. Camenisch, J., Lysyanskaya, A.: An Efficient System for Non-transferable Anonymous Credentials with Optional Anonymity Revocation. In: EUROCRYPT 2001. pp. 93–118. LNCS 2045, Springer (2001)
12. Camenisch, J., Lysyanskaya, A.: Dynamic Accumulators and Application to Efficient Revocation of Anonymous Credentials. In: CRYPTO 2002. pp. 61–76. LNCS 2442, Springer (2002)
13. Camenisch, J., Lysyanskaya, A.: A Signature Scheme with Efficient Protocols. In: SCN 2002. pp. 268–289. LNCS 2576, Springer (2003)
14. Camenisch, J., Shoup, V.: Practical Verifiable Encryption and Decryption of Discrete Logarithms. In: CRYPTO 2003. pp. 126–144. LNCS 2729, Springer (2003)
15. Camenisch, J., Stadler, M.: Efficient Group Signature Schemes for Large Groups. In: CRYPTO 1997, pp. 410–424. LNCS 1294, Springer (1997)
16. Chow, S.S.M.: Real Traceable Signatures. In: SAC 2009. pp. 92–107. LNCS 5867, Springer (2009)
17. 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
18. ECRYPT II: Yearly Report on Algorithms and Key Lengths (2012), revision 1.0
19. Fiat, A., Shamir, A.: How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In: CRYPTO 1986. pp. 186–194. LNCS 263, Springer (1987)
20. Hein, D.M., Wolkerstorfer, J., Felber, N.: ECC Is Ready for RFID — A Proof in Silicon. In: Proc. 15th Workshop on Selected Areas in Cryptography (SAC 2008), Sackville, New Brunswick, Canada, August 14-15, 2008. pp. 401–413. LNCS 5381, Springer (2009)
21. IBM Research Zürich Security Team: Specification of the Identity Mixer Cryptographic Library, version 2.3.4. Tech. rep., IBM Research, Zürich (Feb 2012)
22. Kilian, J., Petrank, E.: Identity Escrow. In: Proc. 18th Annual Int. Cryptology Conf. (CRYPTO 1998), Santa Barbara, California, USA, August 23-27, 1998. pp. 169–185. LNCS 1462, Springer (1998)
23. Kutylowski, M., Krzywiecki, L., Kubiak, P., Koza, M.: Restricted Identification Scheme and Diffie-Hellman Linking Problem. In: INTRUST 2011. pp. 221–238 (2011)
24. 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)

25. 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)
26. Li, J., Li, N., Xue, R.: Universal Accumulators with Efficient Nonmembership Proofs. In: ACNS 2007. pp. 253–269 (2007)
27. Libert, B., Peters, T., Yung, M.: Group Signatures with Almost-for-Free Revocation. In: CRYPTO 2012. pp. 571–589 (2012)
28. Lueks, W., Alpár, G., Hoepman, J.H., Vullers, P.: Fast Revocation of Attribute-Based Credentials for Both Users and Verifiers. In: SEC 2015. pp. 463–478. IFIP AICT 455, Springer (2015)
29. Manulis, M., Fleischhacker, N., Günther, F., Kiefer, F., Poettrering, B.: Group Signatures: Authentication with Privacy. Tech. rep., Bundesamt für Sicherheit in der Informationstechnik (2012)
30. Montgomery, P.L.: Speeding the Pollard and Elliptic Curve Methods of Cactorization. Mathematics of Computation 48(177), 243–264 (1987)
31. 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)
32. Naumann, I., Hogben, G.: Privacy features of European eID card specifications. Network Security 2008(8), 9–13 (2008)
33. Nguyen, L., Paquin, C.: U-Prove Designated-Verifier Accumulator Revocation Extension. Tech. Rep. MSR-TR-2014-85, Microsoft Research (June 2014)
34. OECD: National Strategies and Policies for Digital Identity Management in OECD Countries (2011)
35. De la Piedra, A., Hoepman, J.H., Vullers, P.: Towards a Full-Featured Implementation of Attribute Based Credentials on Smart Cards. In: CANS 2014. pp. 270–289. LNCS 8813, Springer (2014)
36. 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)
37. 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)
38. Verheul, E.R.: Practical backward unlinkable revocation in FIDO, German e-ID, Idemix and U-Prove. Cryptology ePrint Archive, Report 2016/217 (2016), `http://eprint.iacr.org/2016/217`
39. Vullers, P., Alpár, G.: Efficient Selective Disclosure on Smart Cards Using Idemix. In: IDMAN 2013. pp. 53–67. IFIP AICT 396, Springer (2013)