Ubiquitous Privacy-Preserving Identity Managment

Kristof Verslype and Bart De Decker

Abstract The increasing use of digital credentials undermines the owner's privacy. Anonymous credentials offer a powerful means to improve this. However, more is needed w.r.t. usability. A user will indeed have to manage dozens of credentials in the future: sporting club credentials, a digital driving license, e-tickets, etc. The owner will want to use these anytime at any place. The credentials must remain manageable as well and, in case of theft or loss, they must become unusable by others and recoverable by the legitimate owner. A possible solution based on smart card or SIM tokens is presented, in which user privacy is maximized. An evaluation reveals both strengths and future challenges.

1 Introduction

A credential is a piece of information attesting to the integrity of certain stated facts: properties about or rights of its owner. Examples are a driving license, money, an identity card and a ticket.

Traditional digital credentials (e.g. X.509 certificates) pose a threat to the privacy of the owner since they generally contain a unique identifier together with other personal data. This data is registered in databases, potentially with other data (shopping behaviour, medical records, etc.). These data are not only interesting to the entity to which the user shows the credential, but also to insurance companies, to marketeers, etc. Databases containing personal data are thus very valuable and a point of attraction for (internal or external) attackers. They can also get lost, and possibly fall in the wrong hands, as we saw recently in the UK.

Moreover, a user will have to manage dozens of credentials in the future: a sporting club credential, a digital driving license, digital prescriptions, cinema e-tickets, etc. The problem of loss of privacy and identity theft will thus aggravate if we do

Department of Computer Science, K.U.Leuven, Celestijnenlaan 200A, B-3001 Leuven, Belgium
e-mail: firstname.lastname@cs.kuleuven.be

not offer the proper techniques to the users in a practical way. At the same time, the user needs access to these credentials anywhere at any place. E.g., it is not acceptable that the user has a smart card for each credential or that the credentials can only be used on a single computer. The credentials must thus remain easily manageable. In case of theft or loss, credentials must be useless for others and recoverable by the legitimate owner.

Privacy enhancing credentials are being developed and implemented. These allow the user to select what data will be released. However, more is needed. This paper examines how a user can manage and use credentials such that the above requirements hold. Therefore, a portable user-unique token (e.g. a smart card) is introduced, as well as an online server where credentials can be stored in a privacy-preserving way, while preventing loss and exposure of credentials and related credential data. This paper is the result of an exercise in which we tried to maximize the privacy of the user, while still taking into account the other requirements. The exercise revealed future problems and challenges that must be tackled in order to have a deployable system.

Section 2 touches cryptographic and technology related aspects. Section 3 discusses the storage and management of credentials. Section 4 presents the roles and high level interactions. Section 5 presents the requirements. The protocols are described in section 6, and evaluated in section 7. Section 8 refers to related work. Section 9 concludes and discusses future work.

2 Technologies

This section briefly touches aspects about modular exponentiations, zero-knowledge proofs and key lengths and discusses anonymous credentials.

2.1 Some Cryptographic Aspects

A **modular exponentiation** (modex) has the form $h \leftarrow g^a \bmod n$. Finding a out of h, g and n is infeasible for sufficiently large numbers (DL assumption).

A **zero-knowledge proof** proves some knowledge of the prover to a verifier, without revealing any other information. The notation in [9] will be used in this paper: $PK\{\alpha : y = g^\alpha \bmod n\}$ denotes a *"zero-knowledge proof of knowledge of an integer* α *such that* $y = g^\alpha \bmod n$ *with* y, g *and* n *publicly known"*. A message can be added to the proof: $PK\{\alpha : y = g^\alpha \bmod n\}(message)$. The proof can only be correctly verified if the message is not modified.

As technology is evolving, 1024 bits **modulus length** will soon be insufficient; 2048 bits, both for DL and RSA, will suffice till 2022 and symmetric keys need to have a length of at least 109 bits to be secure till 2050 [12].

2.2 Anonymous Credentials

Anonymous credentials were introduced by Chaum [10]. Idemix [8] and U-Prove [7] are two credential systems that are being developed. They allow for anonymous yet accountable transactions between users and organizations and allow to show properties of some credential attributes while hiding the others. E.g. using an anonymous credential containing one's name, date of birth and address, one can prove that he is older than 18, without revealing anything else. Credentials can have features such as an expiry date, the number of times it can be shown and the possibility to be revoked. A mix network ([15], [11]) is required for network layer anonymity. The two most important (simplified) anonymous credential protocols are $getCred()$ and $showCred()$.

- In $cred \leftarrow getCred(attributes, features)$ an *issuer* issues a new credential *cred* to the *receiver*. The credential attributes and features are given as input.
- In $trans \leftarrow showCred(cred, props, [msg])$, the *prover* shows properties *props* of credential *cred* to the *verifier* resulting for the verifier in a transcript that can serve as proof in case of disputes. By giving message *msg* as (optional) input, the prover additionally signs *msg* anonymously: the verifier knows that the signer fulfills the revealed properties.

A U-Prove modification was proposed [7] to enforce the collaboration of a smart card or SIM token, containing a credential secret, during a credential show by a device. A similar Idemix modification exists. The notation becomes:

- $(secret; cred) \leftarrow getCred(attributes, features)$**.** The credential issue results in a credential on the device and a secret on the token.
- $trans \leftarrow showCred(secret; cred, props, [msg])$**.** The credential is shown by the user's computer, with the help of the token, which needs to know the corresponding secret. The secret never leaves the token in cleartext.

Anonymous credential systems heavily rely on complex zero-knowledge proofs an thus on modular exponentiations. Both $getCred()$ and $showCred()$ require only a single modex on the token if the token is involved.

3 Credential Manager and Credential Repository

We distinguish between the Credential Repository and the Credential Manager. The former stores the user's credential data such as credentials, but also transcripts, while the latter enables usage and management of this data.

The **Credential Repository** can be situated locally at the user's side, on a remote server or even a hybrid combination is possible. A purely local credential store, on a device such as a USB-stick, smart card, PDA or mobile phone, can have dramatic consequences in case of loss, theft or damage of the device. If a server-based solution is applied, tampering, deletion, reading or use of credential data and linkage of credential data to the owner by the server must be prevented. Also, an Internet connection is required, which will not always be available, and which potentially slows

down the system. Therefore, a hybrid solution is presented; permanent remote storage is combined with local caching.

The **Credential Manager** runs the credential protocols and therefore, it must be trusted by the user; e.g. it should never show a credential to another party without the user's consent. The Credential Manager is the only place where credential data may exist in cleartext. If the Credential Manager runs on a server, it cannot be trusted by the user, as it is completely outside the user's control. Also, network access is required. If the Credential Manager runs locally on a device which is a user's PDA, mobile phone or PC, it is relatively trustworthy, but a Trojan horse or malware cannot be excluded in case of a software implementation. A Credential Manager that runs on a computer outside the user's control is much more dangerous (e.g. a computer in a public library or shop). If the Credential Manager runs entirely on a secure token such as a smart card, the Credential Manager is trustworthy. This paper focuses on a more realistic approach, where most of the computation is outsourced by the token to the client device to which the token is connected.

4 Roles and Interactions

An overview of the roles and their most important interactions is given in figure 1. Token *T* is a SIM or smart card owned by the user who inserts it in a client device *D* in order to manage or use his credentials. Token *T* locally caches a part of the credential data. The client device *D* will do most of the computations (outsourced by *T*). *D* is also responsible for the network connection and for the token-user interactions. The dashed part of *D* is trusted by the user. This part can for instance be a sealed smart card reader with limited user interaction capabilities (e.g. small screen) on a public computer. The Online Repository *OR* stores almost all the user's credential data. The credential issuer *I* issues new credentials to the user. These credentials can be shown to a service provider *SP* and stored on the *OR*. During token issuance and initialisation, as well as during token recovery, a token issuer *TI* and and a notary[1] are needed.

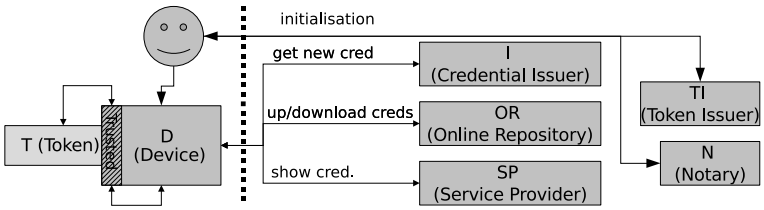

Fig. 1 Overview of the different roles and their interactions.

5 Requirements

The requirements on which we focus are now summed up.

[1] A trusted intermediary in contract signing, testaments, etc. Exists in many countries.

- **User privacy.** The *OR* must be unable to link actions or data stored on the *OR* to an identifiable user (P1). The anonymity set w.r.t. the *OR* must thus also be kept as large as possible, i.e. profiling must be minimized. The amount of personal data the (potentially untrustworthy) device *D* can extract must be minimized (P2). Eavesdroppers and attackers must not be able to derive any personal information (P3).
- **Integrity.** The *OR* must be unable to tamper with uploaded credential data (I1). This also means that the *OR* is unable to add or delete credential data. Abuse must be provable (dispute handling). It must be impossible to tamper with messages by external attackers (I3) or by the intermediary *D* (I2).
- **Confidentiality.** It must be impossible for anyone to get hold of the sensitive data such as protocol keys and credential secrets (C1).
- **Access Control.** In order to use/manage credential data, the user needs to authenticate to the token, using a sufficiently strong user authentication mechanism (A1). If there is a possibility that a thief obtains access to the owner's token (e.g. using the proper PIN), the legitimate owner should be able to revoke further usage of the token and access to the *OR* (A2). The *OR* must be ensured that the user indeed has the right to use its services and that the user is indeed the owner of the *OR* record he wants to access (A3). The user must be ensured that he is communicating with the right *I*, *OR* and *SP* and vice versa (A4).
- **Efficiency.** Computational, interaction and storage efficiency are especially important w.r.t. the token, which has limited capabilities (E1).
- **Functionality.** Even without Internet connection, the user must be able to use a limited set of credentials (e.g. a one show digital cinema ticket) (F1). The *OR* must be able to limit the amount of storage space that can be used by a single user (F2). Loss, theft or damage of the token should not result in loss of any credentials or other credential data (F3).

6 Protocols

We start by presenting the most important data structures; next we describe the protocols for issuing credentials, uploading credentials to the *OR*, downloading credentials from the *OR* and showing a credential to a service provider *SP*. We also discuss some other protocols and aspects in lesser detail.

6.1 Data structures

This section shows the data structures on the online repository and token.

Each user has exactly one **token** T, which contains:
- An unextractable user-specific *user master secret S*. This secret is required to access the user's credential data (both locally and on the *OR*).
- A credential $cred_T$ and a corresponding secret $secret_T$. These allow to authenticate anonymously to the *OR*. To do so, $cred_T$ is off-loaded to the device *D*. $secret_T$ never leaves the token. This way, the credential is linked to the token.

- X.509 certificates $cert_{OR}^{sig}$ and $cert_{OR}^{auth}$ of the *OR* which allows to verify signatures from and authentications of the *OR*.
- Locally cached credentials.
- The credential *index file* $file_{index}$, containing for each credential a tuple $(i, desc, receipt)$:

- i is an index for the credential, used only inside the token.
- $desc$ contains a user-friendly credential name, a short description, and the credential attribute names. This allows T to locally search for the proper credential. E.g. with what credential is it possible to prove that $age > 18$.
- $receipt$ is a proof attesting that a credential is stored by the *OR*.

The **Online Repository (OR)** contains:
- Secret keys SK_{OR}^{sig} and SK_{OR}^{auth} and the corresponding certificates $cert_{OR}^{sig}$ and $cert_{OR}^{auth}$ for signing and authentication purposes.
- For each upload credential, an $(id, cred^E, receipt_{OR})$-tuple is stored:

- id is the unique index of the *OR*-record where the credential is stored. Each credential of each user has its unique index.
- $cred^E$ is the credential and credential (token) secret encrypted by T.
- $receipt_{OR}$ is the receipt, anonymously signed with $cred_T$. The *OR* can use it as a proof in case of dispute. The signature (i.e. a transcript) can be deanonymizable by a trusted third party.

6.2 Assumptions and Notation

We assume that (1) the user is already authenticated to the token, (2) all secret data loaded in T's volatile memory will be removed from that memory by the T as soon as it is no longer required in the protocol session and (3) the user is informed by T about the protocol status at the end.

All the network connections involving D are integrity and confidentiality preserving in which OR, I and SP identify and authenticate to D, while the client stays anonymous. $user \leftrightarrow D$ and $D \leftrightarrow T$ connections are direct connections. We assume that during one protocol execution, the same (secure) connection is used between two parties, introducing linkabilities of actions during the protocol.

All the ciphers are integrity preserving. This can easily be done by adding a MAC before encryption. The numbers g and p are generated by OR, publicly available and stored by T. p is prime and g is a generator of a multiplicative group with order q with $q|p-1$ and p and q sufficiently large.

The method $genKey(seed)$ generates a symmetric key. The method $sendEncrypted(PK, data)$ sends data to a receiver, via the potentially untrusted device D. Therefore, the data is encrypted using the public key of the receiver.

Superscripts ^{E} and ^{S} denote a cipher and a signed message: $M^E \leftarrow encrypt(K, M)$, $M \leftarrow decrypt(K, M^E)$), and $M^S \leftarrow (M)sig_K$. The initiating entity is put in bold: $\mathbf{X} \rightarrow Y$. Optional steps are between square brackets $[\ldots]$.

6.3 Token Issuance and Deployment

We focus on the realistic business model where the token issuer *TI* and the online repository *OR* are two roles which may collaborate in order to obtain user information; e.g., the *TI* and *OR* can be owned by the same company.

In order to obtain a token for getting access to the OR, the user first needs to register to the *OR*. Therefore, payment and/or identification might be necessary. What exactly is required, depends upon the applied business model. E.g. the government can offer for free access to the *OR* for all its citizens and thus, it issues exactly one token to each citizen. Therefore, the citizens need to prove their identity, e.g. with their eID card.

To guarantee the user's privacy and security, the user secret *S* on the token must not be known by the *OR* or the *TI* and must thus be generated at the user side. On the other hand, it is unacceptable that loss of *S* (e.g. damaged token) obliges the user to renew all his - potentially expensive - credentials. Therefore, a secret sharing scheme can be applied. E.g. *TI* could generate half of *S*, store it together with the user id and put it on the token *T*. During token initialisation by the user, the other half is generated on the token and sent to a user-chosen notary. Using both secret halves, *S* is generated (e.g. by xoring) and the TI generated secret part is removed from *T*.

Different parties can deploy our system. The government can issue eID cards, combined with the credential management functionality. The government can manage the *OR* itself, or can outsource it to a commercial company. Other potential token issuers are banks (bank card issuers), and GSM operators (issuers of SIM tokens for mobile phones).

Although we focus on anonymous credentials, it is possible to extend the protocols to support other credential types as well, although this will have a negative impact on the privacy. Computationally, it will be less intensive.

6.4 Receiving a Credential

In table 1, a credential is issued to the user (1). This results in a credential on the device *D* and a corresponding secret on *T*. The credential is transfered by *D* to *T* (2). Then, *T* generates a local index *i* for the new credential (3). This *i* is used together with the user's master secret *S* to generate a credential specific symmetric key *K* (4). This *K* is used to encrypt both the credential and the corresponding secret (5). The resulting cipher is stored (6). Potentially with the help of the user, a description of the credential is made (7) and together with *i* added to the index file (8). The *null*-value indicates that there is no receipt; the credential has not yet been uploaded to the *OR*.

6.5 Upload Credential

Now, the credential is only stored on the token, making it vulnerable to loss. Therefore, it is uploaded (see table 2). Afterwards, the credential can be deleted from the token.

1	$T \leftrightarrow \mathbf{D} \leftrightarrow I$	$(secret; cred, \ldots) \leftarrow getCred(\ldots)$
2	$T \leftarrow \mathbf{D}$	$send(cred)$
3	$\mathbf{T}$	$i \leftarrow getFreeLocalIndex()$
4	$\mathbf{T}$	$K \leftarrow genKey(i\|\|S)$
5	$\mathbf{T}$	$cred^E \leftarrow encrypt(K, (secret, cred))$
6	$\mathbf{T}$	$store(cred^E)$
7	$\mathbf{T}[\leftrightarrow U]$	$desc \leftarrow composeDescription(cred)$
8	$\mathbf{T}$	$addToIndexFile(i, desc, null)$

Table 1 The 'Receive Credential' protocol

To upload the credential to the *OR*, the credential cipher stored by T is retrieved from T's local storage (1). The corresponding symmetric key K is calculated (2). Based on this K, the *OR* specific, global *id* for that credential is calculated (3). The user proves with the help of both T and D that he is allowed to use the *OR* by showing $cred_T$ (4, 5). T now sends *id* and $cred^E$ to *OR* (6, 7) and proves that he is the owner of the *OR*-record with index *id* (8). By linking the zero knowledge proof with $cred^E$, the *OR* is sure that D did not tamper with the credential cipher.

The *OR* generates for the user a receipt (9), stating that at a given moment, an encrypted credential with a certain hash value was uploaded to record *id*. Newer receipts invalidate older ones. The signed receipt is sent to and verified by T with $cert_{OR}^{sig}$ (10, 11).

T now signs anonymously that receipt with $cred_T$ (12). The resulting transcript (signature) is stored by the *OR*, together with the credential cipher, *id* and *receipt* (13). T adds the receipt to the index file (14). Both user and *OR* now have a proof that can be used in case of dispute (e.g. user claims that a credential has been removed by *OR*.

1	$\mathbf{T}$	$cred^E \leftarrow retrieveLocalCred(i)$
2	$\mathbf{T}$	$K \leftarrow genKey(i\|\|S)$
3	$\mathbf{T}$	$id \leftarrow g^K \bmod n$
4	$\mathbf{T} \rightarrow D$	$send(cred_T)$
5	$T \leftrightarrow \mathbf{D} \rightarrow OR$	$showCred(secret_T; cred_T, possession)$
6	$\mathbf{T} \rightarrow D \rightarrow OR$	$sendEncrypted(PK_{OR}, id)$
7	$\mathbf{T} \rightarrow D \rightarrow OR$	$send(cred^E)$
8	$\mathbf{T} \rightarrow D \rightarrow OR$	$PK\{K : id == g^K \bmod n\}(cred^E)$
9	$\mathbf{OR}$	$receipt \leftarrow (id, H(cred^E), timestamp_{OR})sig_{OR}$
10	$T \leftarrow D \leftarrow \mathbf{OR}$	$send(receipt)$
11	$\mathbf{T}$	$verify(cert_{OR}^{sig}, receipt)$
12	$T \leftrightarrow \mathbf{D} \rightarrow OR$	$trans_{OR} \leftarrow showCred(secret_T; cred_T, possession, receipt)$
13	$\mathbf{OR}$	$store(id, cred^E, receipt, trans_{OR})$
14	$\mathbf{T}$	$updateIndexFile(i, receipt)$

Table 2 The 'Upload Credential' protocol

6.6 Show Credential

Table 3 shows how a credential is shown after it has been retrieved from the *OR*. First, T searches for a credential able to prove the requested properties $properties_{show}$ (1,2). If more credentials can be shown, the user selects the most

appropriate one. The credential decryption key K is calculated and the credential's OR-index *id* is retrieved from the receipt (3,4). D and T show together $cred_T$ to the *OR* (5,6) to prove that the user is allowed to use the *OR*. Now the *OR*-index *id* of the required credential is sent to the *OR* (7). The *OR* replies with the credential cipher and the corresponding receipt (8). We will later argue why this receipt is required although it is not used in this protocol. T decrypts the credential cipher. This results in a credential and its corresponding secret. No proof of possession of the *OR*-index *id* is required as no changes on that record are performed and only the owner of the record can decrypt the content. Finally, the credential is shown (11, 12) after the user has given his consent (10).

1	$T \leftarrow D \leftarrow \mathbf{SP}$	$send(properties_{show})$
2	$\mathbf{T}[\leftrightarrow U]$	$(i, desc, receipt) \leftarrow find(properties_{show})$
3	$\mathbf{T}$	$K \leftarrow genKey(i\|\|S)$
4	$\mathbf{T}$	$id \leftarrow receipt_T.id$
5	$\mathbf{T} \rightarrow D$	$send(cred_T)$
6	$T \leftrightarrow \mathbf{D} \rightarrow OR$	$trans_{OR} \leftarrow showCred(secret_T; cred_T, possession)$
7	$\mathbf{T} \rightarrow D \rightarrow OR$	$sendEncrypted(PK_{OR}, id)$
8	$\mathbf{T} \leftarrow D \leftarrow OR$	$send(cred^E, receipt')$
9	$\mathbf{T}$	$(secret; cred) \leftarrow decrypt(K, cred^E)$
10	$U \leftrightarrow \mathbf{T}$	*user gives permission*
11	$\mathbf{T} \rightarrow D$	$send(cred)$
12	$T \leftrightarrow \mathbf{D} \rightarrow SP$	$showCred(secret; cred, properties_{show})$

Table 3 The 'Show Credential' protocol

6.7 Other Relevant Aspects and Protocols

We touch *token recovery* and *limiting the usage of the OR*. Other protocols such as $file_{index}$ *upload* are not discussed in this paper.

Token recovery. The user requests a new token from the token issuer TI, which then revokes the previous $cred_T$ and puts a new one on a new token, as well as one half of the user master secret S. After having contacted the notary, S is regenerated. If the last version of the index file is not uploaded as a cipher to the *OR*, it can be recovered by requesting for each *id* owned by that user the credential data from the *OR*, as it is done in our show credential protocol, where the *OR* not only returns the credential cipher, but the receipt as well. The index file can thus be regenerated/updated. For each credential recovery, a new connection with the *OR* needs to be made to avoid linkabilities. The credential data that were not yet uploaded to the *OR* are lost. Thus, some credentials potentially need to be revoked and reissued.

Limiting usage of the OR. As a result of the unlinkability of *OR* records, the user has a potentially unlimited online storage space. Limiting the size of a record is one part of the solution. Secondly, the number of records per user can be limited. The token issuer TI can issue two credentials $cred_T^\infty$ and $cred_T^k$ instead of a single $cred_T$. The former is an unlimited show credential, the latter a k-show. $cred_T^k$ is only used for step 5 in the upload protocol, in all the other situations, $cred_T^\infty$ is used. If the user removes an *OR* record, he obtains a proof thereof, blindly signed by *OR*. Later, the

user can use these proofs to update (reload) the k-show credential.

Keeping track of your anonymity. Different shows of the same U-Prove credential are linkable, as well as shows of Idemix credentials over the same nym. The user's willingness to reveal personal data to a service provider might depend upon the amount of data that has been revealed previously. Therefore, the user can decide to store a profile tuple $(id_{SP}, spec, i_{cred}, timestamp)$ on the token. *spec* describes (a simplified $properties_{prove}$) what properties of the credential referred to by i_{cred} were shown. As the tokens usually do not have a clock, timestamp must be provided by the client device *D*. For each service provider whereof the user stores such tuples, a different *OR*-record is needed. Each such profile record thus contains a set of profile tuples. This can be uploaded in a similar way as the credentials. However, each time a tuple is added, the record changes and thus a new receipt must be generated. This receipt has the form $(id, nb, acc, timestamp)$. *nb* is the number of tuples, acc has the form $H(...H(H(tuple_1), tuple_2, ..., tuple_{nb})$ and avoids tampering by the *OR*. Later, the user can merge tuples, but this must be done in a trusted environment.

7 Evaluation

In this section, the requirements listed in section 5 are evaluated.

7.1 Privacy

The level of privacy is evaluated by analysing what personal information the different entities can or cannot obtain. An overview is given in table 4.

OR	1.1 Size and time of action on *OR*-record with id *id* 1.2 Number of tuples per profile record 1.3 No inter-credential or credential-owner linkability
Eavesdropper	2.1 No linkability of packets 2.2 No information leakage on network layer.
Device	3.1 When, for how long are what entities contacted. 3.2 Credential and show specification, but no secrets or keys. 3.3 [I/O via client] Interception of commands, PINs, etc. 3.4 More properties can be revealed during a single show.
Illegal token access	4.1 [$cred_T$ valid] Access to all credential data, not to secrets, keys. 4.2 [$cred_T$ revoked] Access to local credential data, not to secrets, keys.

Table 4 Overview of the data the different entities can obtain.

P1 - During authentication, the *OR* first receives a $cred_T$ credential show. If $cred_T$ is an Idemix credential, the *OR* cannot link different shows thereof. The proof of possession of the record with index *id* does not reveal anything else. The *OR* can thus not link records to each other or to the same user. A mix-network is required and delay might be necessary to avoid time-based linking. The *OR* cannot distinguish between credential retrieval for recovery or show purposes. We thus can theoretically have total unlinkability of credential records stored by the *OR* (1.3).

OR knows the index *id*, receipts and ciphers of credentials and privacy tuples. The only evidence it can collect is the time actions on records are performed and the cipher sizes (1.1). This can be reduced by increasing the amount of credential data

cached by T and by padding uploaded data. *OR* also knows the number of uploaded tuples per profile record (1.2). The *OR* thus only obtains a very limited usage profile per credential record and per profile record.

P2 - The device D can get hold of credentials and show specifications during the protocol executions (3.2). D also knows when actions are performed, and to whom (3.1). D can thus get hold of many personal data. Users should be aware of this when they use potentially untrusted computers.

In the absence of a sealed token reader with user I/O capabilities, D can eavesdrop on the interactions between user and T, however this does not reveal new personal attributes (except the user's PIN!) (3.3). D can never get hold of *id*s, preventing it from collaborating with *OR*, nor does it ever see credential secrets or the master secret. However, by synchronizing and collaborating with *OR*, linkabilities can be revealed.

D can show more (or other) properties to the *SP* than what is required in $properties_{show}$. Showing insufficient properties to D will be detected because the user will not obtain the expected privileges. The required user consent to T before a credential is shown and the corresponding secret on T prevents D from surreptitiously showing a credential (3.4).

P3 - Eavesdroppers or external entities cannot link different sessions if a mix network is used (2.1). Secure network connections prevent eavesdropping (or tampering) on the sent data (2.2).

If an attacker obtains access to T (e.g. by obtaining the PIN), the privacy is evidently further reduced because he can access many locally stored data (4.2). If $cred_T$ is not yet revoked, the attacker can get hold of the other, *OR*-stored, data as well (4.1). As long as the tamper resistance is maintained, the attacker cannot get hold of keys or secrets.

7.2 Integrity

I1 - Tampering with individual ciphers stored by the *OR* will be detected as the ciphers are integrity preserving. Deletion of an individual cipher in a profile record can be detected by verifying the receipt. The *OR* also stores an (anonymous) signature on the receipt from T, to prevent charges based on out-dated receipts of T. By slightly modifying the protocol, the proof generated in step 8 in the upload credential protocol can serve as a pre-proof that can be used by *OR* in case the protocol is interrupted after step 11.

An unsolved problem is that token (and thus $cred_T$) renewal results in an invalidation of the user's anonymous signatures on the receipts, invalidating the *OR*'s evidence. Renewal of other certificates such as $cert_{OR}^{sig}$ must also be possible in a transparent way.

I2 - T cannot be sure whether there is indeed a secure connection between D on the one side and *OR*, I or *SP* on the other side. Therefore, *OR*, I and *SP* need to have and enforce policies that require D to set up such a connection.

I3 - If there is no sealed card reader with I/O capabilities, the communications between T and user need to pass D, which can do modifications. This is a typical

smart card/SIM token problem. Everything else sent or forwarded by D to T or an external party is integrity protected. However, T is unable to check the integrity of anonymous credentials due to T's computational limitations. *OR* signatures (e.g. on receipts) can be verified by T as it has the right *OR* certificate, but still, D is needed to check the validity of that certificate. Even if $properties_{show}$ is signed and T has the corresponding certificate, T cannot verify whether the certificate corresponds to the right *SP*.

7.3 Access control

A1 - A PIN or biometric data can be used for user authentication to the token. No credential (including $cred_T$) secret ever leaves T in an unencrypted form. The token is thus required to access the *OR* or credentials. However, if no sealed token reader with user I/O capabilities is used, D can intercept the PIN, giving it complete control over the usage of T.
A2 - By revoking $cred_T$, further access to the *OR* by an illegitimate user is prevented. However, this user can still access and use the locally cached credential data. These credentials need to be revoked as well. Quickly revoking $cred_T$ and all the locally cached credentials is thus of utmost importance. Caching too many credentials locally should be discouraged. Access by an illegitimate user to the locally cached profile tuples or the index file cannot be prevented if he knows the PIN.
A3 - Only users having a valid token (i.e. containing a valid $cred_T$) can use the *OR*. If a user wants to change something in *OR*-record with index id, he needs to prove that his token knows the corresponding key K. Thus, only the owner of that record can do changes. Because the user's token is the only entity that knows K, only the owner can decrypt the data in the *OR*-record. A proof of ownership is not necessary in this case.
A4 - D is trusted to connect to the proper I, *SP*. If it connects to the wrong (fake) *OR*, that *OR* will not be able to issue valid receipts or show the right data.

7.4 Confidentiality.

C1 - We assume that the notary does not collaborate with the *OR* and that the token is tamper proof such that secret data cannot be extracted. Each record on the *OR* is encrypted using another key. If such a credential-specific key is leaked, only a single credential record is compromised.

7.5 Storage

Our reference credential has 7 attributes. Tests showed that Idemix needs about 4.5Kb and about 7KB when using respectively 1024 and 2048 bit keys. This included the credential itself, an XML description and the public key data.

Based on the assumptions in Figure 2 on the observation that T stores $file_{desc}$, $cred_{OR}$, $secret_{OR}$, S, and a two *OR* certificates, about 50KB is needed to manage 50 credentials stored on the *OR*.

Additionally managing 25 profile records (25 different *SP*s) requires about 7KB as the receipts are about the same size as the credential receipts. The size of a single privacy tuple will be dominated by the size of *spec*, which will seldom be larger than 0.5KB if expressed in a compact way.

Thus, a realistic token with 100KB storage space satisfies in our setting.

Cred. Description	
Cred. desc.	64 byte
att. names	7 * 32 byte
TOTAL:	288 byte

Receipt	
id	128 byte
hash	20 byte
$time_{OR}$	8 byte
sig_{OR}	128 byte
TOTAL:	284 byte

$file_{index}$ **entry**	
counter	2 byte
desc	288 byte
receipt	284 byte
TOTAL:	572 byte

Other	
single character	1 byte
master secret *S*	256 byte
small cert.	< 5 KB

Fig. 2 Estimated size of a credential description, a receipt, an index file entry and other data.

7.6 Performance

Token *T* will be the bottleneck in the protocols. Especially modex operations are cumbersome. RSA signature verification as well as RSA encryption can be done efficiently if the exponent is well chosen. Modex operations with a small, numerical, values (e.g. numerical attributes) are no problem either. The Chinese Remainder Theorem can never be applied in our protocols. Based on these observations, only the underlined step in the protocols require a 'hard' modex by *T*: credential receive, upload and show, need respectively one, four and two modex. This can be done in 174, 696, and 348 ms for 2048 bit moduli by a state-of-the-art smart cards (Infineon SLE 88CFX4002P). Less expensive, but still considered as fast, smart cards require much more time. E.g. the ST22N144 requires 1.7, 6.8 and 3.4 seconds to do the same operations. The remaining operations on *T* are lightweight: sending, receiving, storing, reading, en- and decryption of small amounts of data, user interactions and generation of symmetric keys.

The *showCred*() and *getCred*() methods will dominate the steps performed by the other entities. The time required by client and *I*, *OR* or *SP* to issue a credential or to prove properties highly depends upon the involved attributes and properties. Issuing our reference credentials required 25 modex for Idemix at 2048 bit. Show tests required between 12 and 106 hard modex. Even the latter was only proving that one's age is in a given interval. We did not consider the use of Idemix pseudonyms. Usually, the big half of the modex are on the user side. An Intel 1.83GHz CPU needed on average about 80 ms for a 2048 bit modex.

The proposed protocols are thus computationally feasible and acceptable on ths fastest smart cards. Theoretically, the prover part of very simple credential shows can be done solely on these smart cards. However, even for these cards, still help of the client device is indispensable to do more complex operations on credentials. Future improvements (more powerful tokens, more efficient implementations, etc.) will improve this, enhancing the security and privacy. Then, a secure connection could be established between *T* and *OR*, *I* or *SP*. Still, a trusted sealed token reader is required for user I/O.

Only small packets are transfered. The use of mix networks will likely introduce most delay on the network level. By caching, network interactions are reduced. An

implementation is required to test the real-life performance and feasibility of our proposed protocols.

8 Related Work

Online Credential Repositories can be categorized in mechanism-aware or mechanism-neutral systems [6]. Mechanism-aware repositories (e.g. MyProxy [13], CredEx [16]) can support mechanism-specific protocols for credential retrieval. The disadvantage of such systems is that only few types of credentials are supported. Furthermore, the repository can access the credentials. Our repository can store every type of credential and does not know the credential data. Several credential repositories (Entrust Roaming PKI [1], Verisign roaming [5], etc.) are described in [14], which lists some problems that most current repositories suffer from. First, compromising the credential repository allows the attacker to perform an offline attack on each credential [6, 14]. This can be a serious threat if the credentials are encrypted using a password. Our approach uses strong encryption for credentials. Furthermore, because the users's credentials are unlinkable, it is infeasible to gain access to all the credentials of one particular user. Second, most repositories use a potentially untrusted client that can directly access the credentials. On public workstations, this of course poses a big security risk. Our approach minimizes trust put in the client.

Multiple **Identity Management Systems** exist. *Microsoft CardSpace* [3] enables the user to request from identity providers security tokens asserting claims (e.g. age > 21). Although computationally less demanding, each show of a security token requires a new interaction with the identity provider if unlinkability of shows has to be achieved. More trust is put in the identity provider (the issuer). This entity must also be trusted not to collaborate with service providers. *Liberty alliance* [4] is based on federated identity management; service providers exchange personal data about the user to facilitate user's authentication. The default setup is thus not privacy-friendly. However, the user can setup his own identity provider or strong privacy preference can be set to the identity provider. *The Higgins framework* [2], of which Idemix is part of, is a young project aimed at creating a common interface layer that will allow various existing identity management systems to interoperate.

9 Conclusion and Future Work

This paper tried to maximize the user's privacy, as well as the usability of his credentials. Although feasible on the fastest smart cards, a number of problems appeared, that are likely to appear in other similar solutions as well. E.g. the trust put in the client device. The paper revealed those challenges that will need to be tackled in order to have a deployable system that satisfies the user's needs.

A prototype implementation is necessary and aspects such as the use of Idemix nyms and mix network latency need to be looked at. A study of the evolution of token crypto co-processors w.r.t. the available computer power would reveal

whether it will once become possible to run the protocols always entirely on the token. Other challenges are updating the security parameters, using ECC and examining to what extent Trusted Computing Base (TCB) can offer a solution for the untrustworthy client problem. Development of similar protocols for e.g. Microsoft CardSpace are likely to be possible.

Acknowledgements. This research is a contribution to the European PRIME project and is partially funded by the Interuniversity Attraction Poles Programme Belgian State, Belgian Science Policy, the Research Fund K.U.Leuven and the IWT-SBO project (ADAPID) "Advanced Applications for Electronic Identity Cards in Flanders".

References

1. Entrust authority roaming server. http://www.entrust.com/pki/roaming/index.htm.
2. Higgins trust framework project home. http://www.eclipse.org/higgins/.
3. Introducing windows cardspace. http://msdn2.microsoft.com/en-us/library/aa480189.aspx.
4. Liberty allicance project. http://www.projectliberty.org/.
5. Verisign roaming. http://www.verisign.com/products-services/security-services/pki/pki-security/wireless-roaming/index.html.
6. J. Basney, W. Yurcik, R. Bonilla, and A. Slagell. The credential wallet: A classification of credential repositories highlighting myproxy. In *Proceedings of the 31st Research Conference on Communication, Information and Internet Policy*, 2003.
7. Stefan A. Brands. *Rethinking Public Key Infrastructures and Digital Certificates: Building in Privacy*. MIT Press, Cambridge, MA, USA, 2000.
8. Jan Camenisch and Anna Lysyanskaya. An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In *EUROCRYPT '01: Proceedings of the International Conference on the Theory and Application of Cryptographic Techniques*, pages 93–118, London, UK, 2001. Springer-Verlag.
9. Jan Camenisch and Markus Stadler. Efficient group signature schemes for large groups (extended abstract). In *CRYPTO '97: Proceedings of the 17th Annual International Cryptology Conference on Advances in Cryptology*, pages 410–424, London, UK, 1997. Springer-Verlag.
10. David Chaum. Security without identification: transaction systems to make big brother obsolete. *Commun. ACM*, 28(10):1030–1044, 1985.
11. Matt Hooks and Jadrian Miles. Onion routing and online anonymity. *CS182S*, 2006.
12. Arjen K. Lenstra and Eric R. Verheul. Selecting cryptographic key sizes. *Journal of Cryptology: the journal of the International Association for Cryptologic Research*, 14(4):255–293, 2001.
13. J. Novotny, S. Tuecke, and V. Welch. An online credential repository for the grid: Myproxy. In *Proceedings of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10)*. IEEE Press, 2001.
14. G. Sarbari. Security characteristics of cryptographic mobility solutions. In *Proceedings of the 1 Annual PKI Research Workshop, Gaithersburg, Maryland*, 2002.
15. Paul F. Syverson, David M. Goldschlag, and Michael G. Reed. Anonymous connections and onion routing. In *SP '97: Proceedings of the 1997 IEEE Symposium on Security and Privacy*, page 44, Washington, DC, USA, 1997. IEEE Computer Society.
16. David Del Vecchio, Marty Humphrey, Jim Basney, and Nataraj Nagaratnam. Credex: User-centric credential management for grid and web services. In *ICWS '05: Proceedings of the IEEE International Conference on Web Services (ICWS'05)*, pages 149–156, Washington, DC, USA, 2005. IEEE Computer Society.