A smart card based solution for user-centric identity management

Jan Vossaert[1], Pieter Verhaeghe[2], Bart De Decker[2], and Vincent Naessens[1]

[1] Katholieke Hogeschool Sint-Lieven, Department of Industrial Engineering
Gebroeders Desmetstraat 1, 9000 Ghent, Belgium
firstname.lastname@kahosl.be
[2] K.U.Leuven, Department of Computer Science, DistriNet,
Celestijnenlaan 200A, 3001 Heverlee, Belgium
firstname.lastname@cs.kuleuven.be

Abstract. This paper presents a prototype of a previously proposed user-centric identity management system using trusted modules. The trusted module, implemented using a smart card, can retrieve user attributes from identity providers and offer them to service providers, after authentication. This paper allows an evaluation of the practical feasibility of the identity management architecture and provides insight in several design decisions made during the prototype implementation. Also, the cryptographic protocols implemented in the prototype are discussed.

Key words: user-centric identity management, privacy, security

1 Introduction

Many service providers want to control access to their services and critical resources. To address accountability and support personalized services, many services require the user to disclose personal information during a registration phase. Existing federated identity management systems (FIMs) offer a straightforward solution, in which a trusted party manages and releases attributes of an individual. Unfortunately, the privacy of a user is often neglected or not dealt with appropriately. Growing concerns about the privacy of individuals require new solutions that give an answer to these requirements. On the other hand, electronic identity solutions are rolled out in many countries. Many approaches are based on smart cards. An electronic identity card typically stores a set of immutable attributes that can be released during authentication. However, electronic identity cards are combined with FIMs if mutable attributes are requested. Moreover, the user has no or limited control over the attributes that are released.

This paper presents an implementation of a user-centric identity management system based on trusted modules, presented in [15]. The prototype is implemented using a TOP IM GX4 smart card as trusted module. The contribution of this paper is threefold. First the paper allows for a evaluation of the practical feasibility of the proposed architecture. Second, this paper presents concrete protocols that realize the security and privacy requirements stated in [15]. Third,

this paper provides insight in the important design decisions made during the prototype implementation.

The rest of this paper is structured as follows. Section 2 discusses related work. Section 3 recapitulates the general approach presented in [15]. Section 4 presents the roles, key infrastructure, requirements and notations. The major protocols are discussed in section 5. Section 6 focuses on implementation details. A prototype evaluation is done in section 7. This paper ends with general conclusions and points to future research.

2 Related Work

In the federated identity management model [6, 1], of which Shibboleth [9], CardSpace [4] and OpenID [13] are common examples, a user is known to at least one organization (i.e., the identity provider) in the federation (i.e., a group of organizations with mutual trust agreements). If a user contacts a service provider, authentication is delegated to the identity provider of the user. The identity provider releases the personal data that are requested by the service provider. Therefore, user impersonization by identity providers is inherent to the FIM model. Moreover, attribute aggregation [7] is often not supported. Further, many identity providers still use password based authentication since often no infrastructure for strong authentication is available.

Several European countries are issuing governmental eID cards [10] to tackle these shortcomings. However, many designs are not flexible as service providers can only request attributes that are stored in the card itself. Attributes are typically stored in the card during the whole card's lifetime. This implies that only immutable attributes can be stored in the card. Our approach aims at eliminating the drawbacks of existing federated identity management systems and current eID initiatives

Strong authentication is also realized by Jøsang and Pope [8] who present a user-centric identity management model using a *personal authentication device (PAD)*. Each service provider can store an authentication token on the PAD of the user. Our work generalizes the PAD concept to a personal identification device with extended functionality (e.g., support for multiple identity providers, deanonymization) and a concrete implementation is presented. A similar approach is taken in PorKI [12] where users can delegate temporary proxy credentials to workstations using their mobile devices.

Suriadi et al. [14] propose a user-centric federated single sign-on system based on the private credential system proposed in Bangerter et al [2]. However, the computationally expensive nature of the credential scheme limits the feasibility in mobile solutions. Similar concerns apply to the identity management system [5] proposed in the PRIME project. Moreover, this system is less flexible since attributes are embedded in immutable credentials. Multiple credentials need to be used if not all the requested attributes are contained in a single credential.

3 General Approach

This paper proposes an implementation of a privacy friendly user-centric federated identity management approach based on a trusted smart card. The smart card is the mediator between identity providers and service providers. More precisely, an identity provider can store some of the user's personal attributes (or properties thereof) in her smart card. Information that is endorsed by the identity provider can then be disclosed to service providers. The latter use the information to provide fine-grained access control and offer personalized services.

The smart card controls access to identity information. Before the user is authenticated, the service provider first has to authenticate to the smart card and prove that it is authorized to access certain personal attributes. The smart card verifies the acceptability of the service provider's information request. This verification ensures that only information from identity providers is queried, for which the identity providers (or their representative) gave their consent. The authorization info is included in the certificate (or credential) of the service provider. Additionally, the user may further restrict access to personal information through a policy or an explicit consent. If the query is acceptable, the smart card forwards this request to the identity provider(s) that can provide the information.

4 Roles, Key Infrastructure, Requirements and Notation

Roles Nine different roles are distinguished in our solution. The *user (U)* is the owner of a *smart card (SC)*. The card is considered a trusted computing platform by the different actors in the system. It means – amongst others – that the user trusts that the card will never release more than the requested attributes (for which the service provider has been authorized) and that the service provider trusts the card to release genuine attributes. Further, the card also provides a safe environment for storing private information such as keys (i.e., no data can be directly extracted from the cards memory). A *service provider (SP)* can implement attributed-based access control and/or offer customized services by requiring user authentication with a valid smart card. An *identity provider (IP)* manages user attributes that can be retrieved by identity cards during authentication with service providers. *Deanonymization providers (DP)* are responsible for deanonymizing users in case of abuse. A *provider (P)* can either be an identity, service or deanonymization provider. *Certification authorities (CA)* issue card and server certificates. These certificates are used during mutual authentication between card and provider. The *card issuer (CI)* issues identity cards to users. The *revalidation center (R)*(re)validates or blocks cards at regular times. The *middleware (M)* is a software layer that enables communication between card and provider(s). The middleware also allows the user to monitor and influence the authentication process (e.g., view the service provider authentication certificate, view and modify the attribute query, enter PIN). The card also stores an internal private variable `lastValTime` which represents the time R last verified

the revocation status of the card (see section 5). This variable serves two goals. First, it is used by the card during authentication with a provider for verifying that the revocation status of the card was verified by R at a time after the last accepted validation time posed by the provider. Second, since smart cards do not have an embedded real-time clock, the `lastValTime` is used to verify the validity period of certificates.

Key Infrastructure Each card holds a *unique master secret*, K_U, that is used to generate service specific pseudonyms. K_U can either be user-specific or card-specific. The former strategy allows individuals to reuse K_U in multiple smart cards (e.g., when the previous card is defect or lost). If K_U is card-specific, it would be useful to have a mechanism that allows proving a link between (old and new) pseudonyms generated by two different cards of the same user. A revalidation key pair (SK_R, PK_R) is used to set up a secure authenticated communication channel between a smart card and the card revalidation authority. PK_R is stored in each smart card during initialization, the corresponding private key is only known by the revalidation center. A common key pair (SK_{Co}, PK_{Co}) is used by a large set of smart cards. PK_{Co} is embedded in a certificate $Cert_{Co}$. SK_{Co} and $Cert_{Co}$ are stored on the card during initialization. This allows the identity, service and revalidation providers to verify that a genuine smart card is used (without revealing unique identifiers). Each service provider, identity and deanonymization provider has an asymmetric key pair (SK_P, PK_P). SK_P and $Cert_P$ are used to authenticate to smart cards. SK_P is certified by a CA, resulting in $Cert_P$. Section 6 elaborates on additional data that is kept in $Cert_P$. The public keys of (root) certification authorities in the system are placed on the card during initialization. This allows smart cards to verify the validity of certificates. A newly generated session key K_s is used to securely transmit data between a card and providers.

Requirements The requirements to which the prototype system must satisfy are listed below:

- **Functional requirements:**
  - F_1: Service providers can retrieve personal attributes either stored in the identity card and/or managed by an identity provider.
  - F_2: Cards can be personalized (e.g., through privacy policies and preferences).
  - F_3: Adding new providers is straightforward.
  - F_4: Cards can be used online and offline.
- **Security and privacy requirements:**
  - S_1: Strong mutual authentication and secure communication between users and providers (including revalidation authorities).
  - S_2: Controlled access to user attributes (i.e., based on rights/privileges and personal preferences).
  - P_1: Service specific pseudonyms of a user are unlinkable (even by the card issuer). These pseudonyms also encompass pseudonyms of users towards

deanonymization and identity providers. One service provider can offer multiple services for which different pseudonyms are generated.
  - P_2: Support for conditional anonymity during authentication.
  - P_3: Support for anonymous subscriptions.
- **Performance and scalability:**
  - O_1: The prototype system should have straightforward and easy management functions.
  - O_2: Economic use of computationally expensive operations should result in acceptable performance of the prototype system.

Notation During the protocol listings, authenticated encryption is assumed. Authenticated encryption can be realized using several block cipher modes [11] or by explicitly adding a MAC to the message [3] (requiring an extra integrity key). If message integrity verification fails, an exception is thrown and retransmission of the previous message is requested. Arrows ($\rightarrow$ or $\leftarrow$) represent the direction of communication. We assume that during a protocol run, the same connection is used. Dashed arrows ($\dashrightarrow$ or $\dashleftarrow$) represent communication over an anonymous channel. Variables of the card are shown in `teletype` font; if the variable is underlined, it is stored in temporary memory.

5 Card Operations

This section discusses the most important operations performed by the card and gives the cryptographic protocols implemented in the prototype to achieve the security and privacy requirements.

Card Validation During card validation (Table 1), the `lastValTime` is updated with the current time. Prior to updating, the revocation status of the card is checked. This requires unique identification of the card. This is realized by setting up an secure authenticated channel between the card revalidation center and the card using $Cert_R$ and $Cert_{Co}$. Over this channel, the card releases its (unique) serial number which allows the revalidation authority to block the card (if it has been reported lost or stolen) or update its `lastValTime`. The `lastValTime` will be used during authentication with providers allowing them to trust the card's current revocation status without the card releasing any uniquely identifying information. A location hiding communication channel can be setup between the middleware and R in order to hide the whereabouts of the card holder.

Mutual Authentication between Card and Providers During mutual authentication (Table 2), the provider first authenticates to the card. This is done using $Cert_P$ and the public key of the CA stored on the card. Next, the card authenticates to the provider using the common certificate $Cert_{Co}$ and the `lastValTime`. $Cert_{Co}$ enables the verification of genuine smart cards (i.e., no uniquely identifying information is released) and the card checks its revocation status using `lastValTime` and a reference time from the service provider. During authentication, a session key is generated. This phase results in a secure

				revalidateCard():
(1)		M -->	R :	`"RevalidationRequest"`
(2)	SC <-	M <--	R :	c := genRandom()
(3)	SC		:	sig := sign(c , SK_{Co})
(4)	SC		:	K_s := genRandom()
(5)	SC		:	E_{key} := asymEncrypt(K_s, PK_R)
(6)	SC		:	E_{msg} := symEncrypt([$Cert_{Co}$, sig, `chip_number`], K_s)
(7)	SC ->	M -->	R :	E_{msg}, E_{key}
(8)			R :	K_s := asymDecrypt(E_{key}, SK_R)
(9)			R :	[$Cert_{Co}$, sig, `chip_number`] := symDecrypt(E_{msg}, K_s)
(10)			R :	**if** (verifyCert($Cert_{Co}$) == **false**) abort()
(11)			R :	**if** (verifySig(sig, c, PK_{Co}) == **false**) abort()
(12)			R :	$time$:= getCurrentTime()
(13)			R :	**if** (isValid(`chip_number`) == **false**) $time$:= -1
(14)	SC <-	M <--	R :	E_{time} := symEncrypt($time$, K_s)
(15)	SC		:	`lastValTime` := symDecrypt(E_{time}, K_s)

Table 1. The card is periodically revalidated by the revalidation center.

anonymous mutually authenticated communication channel[3]. Authentication of the provider is realized by generating and encrypting a session key with PK_P on the card and sending it to the provider. All communication between card and provider is now encrypted with the session key. Hereby, a cryptographic link between the authentication of the provider and card is established. The authentication of the card is then executed by signing a challenge from the provider on the card using SK_{Co}.

Release of User Attributes After mutual authentication, the card can disclose the user's attributes over the previously established secure channel. Attributes that are not stored on the card can be retrieved by the card from identity providers. This also requires authentication between card and identity provider. The certificates of the service and identity provider restrict the attributes that can be queried and provided respectively (see section 6). These restrictions are enforced by the card. Attributes can also be a service specific pseudonym or a deanonymization item (i.e., an encryption of an identifier of the user). The first allows users to have persistent identifiers with a service without revealing identifiers that can be linked to other services. This identifier is generated by taking a cryptographic hash of K_U with the identifier of the service provider (obtained from the certificate). The deanonymization entry is realized by probabilistically encrypting the concatenation of the service specific pseudonym of the user towards the selected deanonymization provider, the `lastValTime` and the conditions required for deanonymization with the key of the deanonymization authority (i.e., $\{hash(K_U || ID_{DP}) || \texttt{lastValTime} || Cert_{SP}.deanonCond\}_{PKDP}$). A list of trusted deanonymization providers is contained in the service providers' certificate.

Before users can use the services of a deanonymization provider, registration is required. During registration the deanonymization authority requires users

[3] We assume that several other requirements are met e.g., the identity set is sufficiently large, an anonymous communication channel is used.

			authenticate():
(1)	$SC \leftarrow M \leftarrow P$:		$Cert_P$
(2)	SC	:	**if** (verifyCert($Cert_P$)==**false**) abort()
(3)	SC	:	**if** ($Cert_P$.validEndTime < `lastValTime`) abort()
(4)	SC	:	$sesId$:= startNewSession();
(5)	SC	:	`session`[$sesId$].maxRights := $Cert_P$.maxRights
(6)	SC	:	`session`[$sesId$].Subject := $Cert_P$.Subject
(7)	SC	:	K_s := genRandom()
(8)	SC	:	`session`[$sesId$].Ks := K_s
(9)	SC	:	$c1$:= genRandom()
(10)	SC	:	`session`[$sesId$].chal := $c1$
(11)	SC	:	E_{key} := asymEncrypt(K_s, $Cert_P$.PK)
(12)	SC	:	E_{msg} := symEncrypt($c1$, K_s)
(13)	$SC \rightarrow M \dashrightarrow P$:		E_{Key}, E_{msg}, $sesId$
(14)		P :	K_s := asymDecrypt(E_{key}, SK_P)
(15)		P :	$c1$:= symDecrypt(E_{msg}, K_s)
(16)		P :	$c2$:= genRandom()
(17)		P :	E_{resp} := symEncrypt([$c1$+1, **accValTime**, $c2$], K_s)
(18)	$SC \leftarrow M \leftarrow P$:		$sesId$, E_{resp}
(19)	SC	:	[$resp$, **accValTime**, $c2$] := symDecrypt(E_{resp}, `session`[$sesId$].Ks)
(20)	SC	:	**if** ($resp$!= `session`[$sesId$].chal+1) abort()
(21)	SC	:	**if** (`lastValTime` < **accValTime**) abort()
(22)	SC	:	`session`[$sesId$].auth = **true**
(23)	SC	:	sig := sign($c2$, SK_{Co})
(24)	$SC \rightarrow M \dashrightarrow P$:		E_{msg} := symEncrypt([$Cert_{Co}$, sig], `session`[$sesId$].K_s)
(25)		P :	[$Cert_{Co}$, sig] := symDecrypt(E_{msg}, K_s)
(26)		P :	**if** (verifyCert($Cert_{Co}$) == **false**) abort()
(27)		P :	**if** (verifySig(sig, $c2$, $Cert_{Co}$.PK) == **false**) abort()

Table 2. Mutual authentication between provider and card.

to release some attributes and stores these in a database linked to the service specific pseudonym of the user. The card stores the public key of the authority together with the name and identifier. Once the deanonymization authority receives a deanonymization item, it can decrypt it and, consequently, link it to the unique identifiers stored in the database. Before revealing the identity of the user, the conditions contained in the deanonymization item are verified.

Although the card itself does not foresee a secure interaction mechanism with its holder, the query from the service provider can be displayed to the card holder using the middleware. The attributes are only released after the user's consent (e.g., after entering a PIN code). The user can also choose not to release some attributes requested by the service provider by removing them from the query.

6 Implementation Details

This section zooms in on several design decisions. The prototype is developed using the Java Card 2.2.1 framework and is deployed on a TOP IM GX4 smart card.

Certificates A hybrid certificate solution is used in the prototype. The card uses standard X.509 certificates to authenticate towards providers; a provider uses card verifiable certificates (CVCs) to authenticate towards a smart card. This strategy ensures interoperability (i.e., providers do not need to install a custom certificate verification module) while avoiding parsing complex certificate structures on the card. Each CVC contains standard information (issuer, subject, role, validity interval etc.). Moreover, each CVC that is issued to an identity provider contains an *attributeList* that lists the identifiers of attributes it may supply, together with a level of assurance (LOA). Authoritative identity providers typically guarantee a high level of assurance for many attributes. The CVC used by the deanonymization and service providers also contains an *attributeList*, which indicates the attributes that can be requested, together with the minimal level of assurance. Moreover, a *trustedIDPList* restricts the set of acceptable identity providers (or IDP groups) for the service. The card will only release attribute values to that particular service provider that were fetched from identity providers of the list (with an appropriate LOA). Note further that CVCs have a short lifetime. This is necessary to ensure a short window of vulnerability as revocation checks are not performed on the card. Each provider also has a X.509 certificate with a long lifetime. The latter is used to authenticate towards CAs that issue the CVCs.

Discovery of Identity Providers Each attribute in the system is represented by an *Attribute* object. Each of them has a unique identifier. Some attribute values are cached on the card. Other attribute values need to be fetched from identity providers. Therefore, each card keeps a list of *IdentityProvider* objects. They define the set of identity providers in which the owner has enrolled. Each *IdentityProvider* object maintains references to a set of attribute objects. They refer to the *Attribute* objects that can be retrieved from the respective provider.

When an attribute query is received, the query handler first looks for the attribute value in the cache. The cached attributes that meet the prerequisites (e.g., LOA, trusted identity provider) are selected. The remaining attributes are fetched from identity providers. The handler selects a minimal set of acceptable identity providers that can supply the remaining attributes, hence, ensuring that only a minimal set of connections to identity providers is required.

Memory Management Smart cards have limited memory. The card used for the prototype has around 70K bytes of available EEPROM. Moreover, the Java Card virtual machine does not implement a garbage collector, nor is it possible in Java to explicitly release memory. Therefore, all required memory should be allocated at the beginning of the program and continously reused.

Caching attributes. A fixed set of byte arrays of variable length is allocated to cache attribute values. These arrays are embedded in *AttributeValue* objects that also keep context information such as retention time, LOA, time of last usage, etc. For an optimal implementation, the distribution of the average length of each type of attribute should be calculated. This caching strategy is straightfor-

ward while limiting fragmentation. When an attribute value is fetched from an identity provider, it might be necessary to remove another attribute from the cache. The following selection strategy is applied. First, a predefined number of *AttributeValue* objects with the smallest memory footprint still large enough to keep the new attribute value are selected. Consequently, the least recently used attribute value is replaced. *Persistent attributes* (see section 6) are not considered by the cache update policy.

Static memory configuration. Since all memory allocations occur when the applet is deployed on the card, the attribute query length, the maximum number of supported identity providers and cached attributes . . . are fixed. Dynamically assigning memory increases flexibility. However, replacement strategies become more complex. Therefore, in the prototype, memory is configured statically. Hereby, the initializer can define the amount of memory assigned to the different parts of the program when installing the applet. For instance, the initializer can opt for allocating only a limited amount of memory for identity providers while increasing the attribute cache.

Personalized Policies The policy engine on the card enforces the policies of the service and identity providers. The policies are specified in the CVCs (cfr. ...*List*). The user can further restrict the access policy (i.e., he can update his policy using his PIN code). First, the user can select the attributes that will be cached on the card until their retention time has expired. Those attributes are marked as *persistent*. Moreover, the user can assign a trust level to service providers: `untrusted`, `default` and `trusted`. Requests from untrusted service providers are blocked. In the `default` policy, user confirmation is required before the attributes are released. If a service provider is `trusted`, the user is no longer involved in the attribute(s) disclosure. The query, however, is still verified using the privileges listed in the CVC. Moreover, the user can also mark an *Attribute* as `sensitive`. If so, user consent (i.e., PIN code) is always required if that attribute needs to be released. The card issuer determines a set of trusted CAs of which the public key is stored in the card during initialization. However, users can further restrict the list of trusted CAs by deactivating keys of untrusted CAs. Although the user cannot further expand this list of trusted CAs, previously deactivated keys can be reactivated. Analogously, keys from deanonymization providers can be deactivated (or even removed which allows registration with a new deanonymization organization if no empty slots were available) and reactivated.

Anonymous Subscriptions For some applications, e.g., news sites, entrance control in buildings,. . . , it is not necessary for a user to disclose a persistent identifier. For instance, news sites only need to verify whether the user has a subscription that allows him to view the requested content. To support this, a *Subscription* object is initialized during enrollment of the user with the service provider. The *Subscription* object contains the *id* of the service provider, a *validity period* and a *type*. The *type* allows the service provider to verify whether the

requested service (or content) is included in the subscription of the user. The *validity period* allows the service provider to specify subscriptions that expire after a specified time. Note that the actual *validity period* does not have to be released but can be verified by the card using a time provided by the service provider. The *type* field constraint can also be verified by the card but is less critical if released since typically only a limited number of subscription types are available. A pseudonym that allows the user to retain his subscription when re-enrolling with a new card (e.g., when the previous card was lost or expired) could be released during enrollment.

Cryptographic Parameters To realize the cryptographic functions defined in section 4 the cryptographic capabilities of the smart card are used. Since RSA is the only asymmetric cryptosystem available on the card, RSA keys are used for all the asymmetric keys in the system. Key lengths of 1024 bit were chosen to achieve good performance, section 7 discusses the performance impact of other key lengths and the potential use of other ciphers. For signing, a SHA-1 digest is encrypted using PKCS #1 padding. Asymmetric encryption uses OAEP padding and can, hence, also be used to realize the probabilistic encryption required for deanonymization.

For authenticated encryption, 128 bit AES keys are used to achieve confidentiality and a MAC is generated by encryption a SHA-1 digest with a 128 bit AES integrity key. The 256 bit session key K_s is divided in two parts to obtain the integrity and confidentiality key.

Random number generation is realized using the on-board random number generator of the chip.

7 Evaluation

Functionality Analysis The proposed solution combines the benefits of smart cards and federated identity management systems. The card is a trusted module to all players in the system. Common secrets allow providers to establish authenticated sessions with the card. Over these channels, service providers can send attribute queries which are fulfilled by the card. Possibly, the requested attributes are retrieved from identity providers that are trusted by the service providers (cfr. F_1).

Although certificates of service providers already restrict the type of attributes that can be queried, the user can be even more restrictive by restricting the attribute query. Further, users can mark attributes as sensitive, assign trust levels to providers, influence caching policy, etc. Hence, extensive personalization functions are available (cfr. F_2).

Adding new providers only requires them to retrieve an authentication certificate from a CA. This requires them to have an audit done that determines what attributes they can request or supply. Service providers also require permissions from identity providers (or groups of) for requesting attributes they provide. Hence, adding new providers introduces some overhead for the respective in-

stance and the certification authority. However, it is a clearly defined procedure transparent for the users and the majority of providers (cfr. F_3).

The caching policy maintains a set of attributes stored on the card. These cached attributes can also be used in offline settings. Hence, apart from authentication and identification also attributes queries are supported in offline environments if the required attributes are cached on the card (cfr. F_4).

Security and Privacy Analysis Reference protocols for realizing secure authenticated sessions between card and providers using the proposed key infrastructure are given in the section above. Providers are assured that the card was not revoked before a certain date/time (i.e., `accValTime`< `lastValTime`). If required, the card will execute the card revalidation protocol before proceeding with the authentication. Revocation of the CVCs of providers is not supported, however, as they are short-lived potential abuse is limited. Hence, S_1 is satisfied.

When a common key is compromised, the set of cards on which the keys are stored should be revoked. This can be done using certificate revocation lists or OCSP responders. The CRL will typically contain little entries since smart cards are designed with countermeasures against attempts to extract secret information. When a card is lost or stolen, this should be reported with the revalidation center, who will block the respective card. Hence, these cards should not be added to the CRL. However, stolen cards can be misused during a service specific time interval (i.e., `accValTime`< `lastValTime`) if the PIN is compromised.

A potential misuse of the revocation strategy is that providers require real-time revocation checks of the card (i.e., `accValTime`= `currentTime`) while this might not be necessary for the requested service. This decreases performance of the card as revalidation is required for each authentication. The CRLs, currently used for revocation checks, are also typically only periodically updated. This may also lead to timing attacks (i.e., linking a profile of a provider to the card number released to the revalidation center) if revalidation center and provider collude.

The public key of the revalidation center stored on the card can be updated by the revalidation center itself. After establishing a secure session with the card, the new certificate can be sent to the card. It will verify that the certificate was issued to the revalidation authority, that the old validity interval precedes the new and the validity of the signature (using the CA public key stored on the card). The revalidation authority can also deactivate keys of CAs that are no longer trusted over the secure channel. Other keys stored on the card cannot be updated or deactivated by the revalidation center.

Access to user attributes is controlled on three levels. First, an audit organization determines the set of user attributes relevant for the offered services. Second, identity providers control the set of service providers that can acquire attributes they provide. These restrictions are embedded in the service provider certificate and, hence, can be verified by the card. Third, users can implement policies or determine at runtime which attributes are released. Hence, extensive control over the release of attributes is enforced (cfr. S_2). The card is the policy enforcement point and should, hence, be trusted by both the user and providers.

After authentication, users can release service specific pseudonyms which are unique pseudonyms which providers nor card issuer can link (cfr. P_1).

Certain services may allow users to remain anonymous but require support for deanonymisation when abuse is detected. As mentioned in section 5, providers can request a probabilistic encryption of an identifier of the user using the attribute query. Hence, these providers can cooperate with the deanonymization service to identify users in case of abuse (cfr. P_2). The user can select supported (and hence trusted) deanonymization authorities by registering with the respective authorities. Further, the certification authorities should employ rigorous security requirements for the deanonymization authorities. This provides suitable security for the users and limits the number of deanonymization authorities. Further, different authorities could provide different levels of deanonymization, depending on the attributes that were released during registration.

During authentication, no unique identifiers are released. Hence, users can use the *subscription* functionality of the card to request access to content which does not necessarily requires identification (cfr. P_3).

Although a user can theoretically remain anonymous, note that the anonymity set for a user largely depends on the size of the set of identity cards with an identical private key and the frequency with which these cards are used. For instance in the case of subscriptions, if the anonymity set is too small and only one user with common certificate x has a certain subscription he is linkable. Hence, when determining the size of the set of identity cards with identical keys one has to consider both the impact of key revocation and potential linkable profiles.

Performance and Scalability Analysis Each party in the system has one or more clearly defined responsibilities. Further, the revocation strategy allows offline services without requiring manual updates of CRL on these devices. Card validation can be triggered by users, but if required, can be executed transparently. Hence O_1 is satisfied.

The rest of this section focuses on performance results. For the prototype implementation mutual authentication between card and provider takes, on average, 1040 ms. The influence of the communication delays between card, workstation and server are minimized by running the test applications locally. Processing the attribute queries takes around 180 ms for attribute sets of around 100 bytes. Hence, when all necessary attributes are cached on the card, the total required authentication time is around 1220 ms. For each identity provider that needs to be contacted, another 1220 ms is added to the required time. The card revalidation operation requires around 900 ms.

Table 3 lists the number of cryptographic operations required during each step. In Table 4 reference performance numbers for cryptographic operations on the prototype smart card are given. These tables illustrate that the asymmetric cryptographic operations are the major performance bottleneck. During authentication, the card performs one private and two public key operations. These operations alone, require around 615 ms, which is more than half of the total authentication time. However, the caching policy and the identity provider

selector minimize the number of identity providers required for satisfying the query (cfr. O_2). Further, replacing RSA with a more efficient algorithm provides an opportunity for significantly improving the performance of the entire system. For instance, elliptic curve cryptography (ECC) could be used to replace the private key operation on the card which could significantly increase performance, especially for increasing key lengths. The smart card used for the prototype, however, does not yet support ECC, hence, no concrete results can be given. Moreover, during a normal authentication procedure the card and service provider first mutually authenticate. Then user interaction is required (entering PIN, modifying – if necessary – the attribute query), after which the card retrieves the attributes from the identity providers and sends them to the service provider. The user interaction divides the waiting time in two smaller pieces and, hence, improves the perceived experience.

The instantiated applet on the card requires around 17000 bytes of memory. This includes temporary memory to store session data but excludes memory required for storing setting dependant information (e.g., cached attributes, registered identity providers, deanonymization authorities). However, as around 70k bytes of memory are available on the prototype smart card, sufficient memory is left to support large numbers of identity providers, cached attributes...

	Card revalidation		**Mutual authentication**		**Attribute exchange**	
	Card	Reval. Serv.	Card	Provider	Card	Provider
Verify/asymEnc	1	2	2	2	0	0
Sign/asymDec	1	1	1	1	0	0
Sym. Enc/Dec	2	2	4	4	2	2

Table 3. Number of cryptographic operations during the different stages of identification.

	RSA				**AES**					
Key length in bits	1024		2048		128		192		256	
	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
Verification	32,00	2,85	72,10	1,02	-	-	-	-	-	-
Signing	555,33	2,62	2318	1,64	-	-	-	-	-	-
Encryption	31,00	3,00	70,10	0,99	31,20	0,40	31,20	0,40	31,21	0,41
Decryption	554,48	3,52	2316	1,42	36,93	1,85	41,9	4,57	44,78	4,03

Table 4. Average (μ) timing results and standard deviation (σ) of 1000 runs of cryptographic operations on the TOP IM GX4 smart card. Tests are done with 128 byte input data, results in ms.

8 Conclusion

This paper presents an implementation and evaluation of a smart card based solution for user-centric identity management [15]. Several implementation details are given and an evaluation is performed, illustrating the practical feasibility of the system. Further research is currently being conducted in two directions. First, the security of the system could be improved by using group signature for

authenticating cards to providers. If the secret keys in one smart card are stolen, it suffices to revoke only that card. Also, support for signed attributes could give a stronger level of assurance which might be required for some services. Second, the usability could be improved by replacing RSA with a more efficient algorithm. The architecture is also being ported to mobile phones with secure elements. To validate the choices made during implementation (e.g., caching policy, selection of identity providers) several realistic use-cases are being implemented.

References

1. Gail-Joon Ahn and Moonam Ko. User-centric privacy management for federated identity management. In *COLCOM '07: Proceedings of the 2007 International Conference on Collaborative Computing: Networking, Applications and Worksharing*, pages 187–195, Washington, DC, USA, 2007. IEEE Computer Society.
2. Endre Bangerter, Jan Camenisch, and Anna Lysyanskaya. A cryptographic framework for the controlled release of certified data. In *Security Protocols Workshop*, pages 20–42, 2004.
3. Mihir Bellare and Chanathip Namprempre. Authenticated encryption: Relations among notions andanalysis of the generic composition paradigm. *Journal of Cryptology*, 21:469–491, 2008.
4. Vittorio Bertocci, Garrett Serack, and Caleb Baker. *Understanding windows cardspace: an introduction to the concepts and challenges of digital identities*. Addison-Wesley Professional, 2007.
5. Jan Camenisch, abhi shelat, Dieter Sommer, Simone Fischer-Hübner, Marit Hansen, Henry Krasemann, Gérard Lacoste, Ronald Leenes, and Jimmy Tseng. Privacy and identity management for everyone. In *DIM '05: Proceedings of the 2005 workshop on Digital identity management*, pages 20–27. ACM, 2005.
6. David W. Chadwick. Federated identity management. In *FOSAD*, 2008.
7. David W. Chadwick, George Inman, and Nate Klingenstein. A conceptual model for attribute aggregation. *Future Generation Computer Systems*, 26(7), 2010.
8. Audun Jøsang and Simon Pope. User centric identity management. In *Asia Pacific Information Technology Security Conference, AusCERT2005, Australia*, 2005.
9. R. L. Morgan, Scott Cantor, Steven Carmody, Walter Hoehn, and Ken Klingenstein. Federated security : The shibboleth approach. *EDUCAUSE Quarterly*, 2004.
10. Ingo Naumann and Giles Hogben. Privacy features of european eid card specifications. Technical report, ENISA, 2009.
11. NIST. Block cipher modes. `http://csrc.nist.gov/groups/ST/toolkit/BCM/current_modes.html`.
12. Massimiliano Pala, Sara Sinclair, and Sean Smith. Portable credentials via proxy certificates in web environments. In *Public Key Infrastructures, Services and Applications*, Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2011.
13. David Recordon and Drummond Reed. OpenID 2.0: a platform for user-centric identity management. In *DIM '06: Proceedings of the second ACM workshop on Digital identity management*, pages 11–16, New York, NY, USA, 2006. ACM.
14. Suriadi Suriadi, Ernest Foo, and Audun Jøsang. A user-centric federated single sign-on system. *Journal of Network and Computer Applications*, 32, 2009.
15. Jan Vossaert, Jorn Lapon, Bart De Decker, and Vincent Naessens. User-centric identity management using trusted modules. In *Public Key Infrastructures, Services and Applications*, Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2011.