Towards a general purpose identity card

Jan Vossaert *Pieter Verhaeghe* *Bart De Decker*
Vincent Naessens

Report CW 587, June 2010

Katholieke Universiteit Leuven
Department of Computer Science
Celestijnenlaan 200A – B-3001 Heverlee (Belgium)

Towards a general purpose identity card

Jan Vossaert *Pieter Verhaeghe* *Bart De Decker*
Vincent Naessens

Report CW 587, June 2010

Department of Computer Science, K.U.Leuven

Abstract

Many countries are currently designing or even rolling out electronic identity cards. Simultaneously, eID applications are developed. In many cases, the eID technology is initially integrated in governmental applications. Thereafter, the technology is adopted by other domains (i.e. the financial sector, eHealth services, social networking, corporate environments, ...). However, security, privacy, performance and/or scalability restrictions in current eID architectures may constrain the enormous potential of eID cards for electronic authentication. This report gives an overview of the major shortcomings of current eID designs and presents a general purpose electronic identity card that reconciles crucial concerns.

Keywords : eID technology, privacy, security.

Towards a general purpose identity card

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] Katholieke Universiteit Leuven, Department of Computer Science,
Celestijnenlaan 200A, 3001 Heverlee, Belgium
firstname.lastname@cs.kuleuven.be

Abstract

Many countries are currently designing or even rolling out electronic identity cards. Simultaneously, eID applications are developed. In many cases, the eID technology is initially integrated in governmental applications. Thereafter, the technology is adopted by other domains (i.e. the financial sector, eHealth services, social networking, corporate environments ...). However, security, privacy, performance and/or scalability restrictions in current eID architectures may constrain the enormous potential of eID cards for electronic authentication. This paper gives an overview of the major shortcomings of current eID designs and presents a general purpose electronic identity card that reconciles crucial concerns.

Keywords : eID technology, privacy, security

1 Introduction

Many European countries are currently laying the foundations for national eID technology [6]. In this context, smart cards are being increasingly used for identification, authentication and digital signatures. They are usually called electronic identity cards. Some countries are already past the design phase and are currently rolling out the eID infrastructure. Early eID applications aim at facilitating communication between citizens and governmental institutions. But eID cards are, consequently, also gradually introduced in other domains (e-health domain, financial, commercial domain, etc.). The architecture of current eID cards, however, was driven by concerns that typically apply in the governmental domain. Examples are ease-of-use, strong authentication, etc. The requirements for on- and offline applications in different domains, however, can exhibit a wide diversity. Whereas for some services, a higher level of security is required (e.g. online banking), in other areas the protection of the card holder's privacy should be the first priority (e.g. social networking). Moreover, by introducing the eID technology in diverse domains, scalability becomes a major concern.

This paper presents the design and evaluation of a general purpose identity card that tackles the security requirements that arise when the card is used in multiple domains and that protects the privacy of the card holder, hereby considering reasonable trust assumptions. Moreover, the proposed eID technology is scalable and the security and privacy measures do not have a unfavourable impact on the performance. We thereby mainly focus on *authentication* (rather than *identification* or *digital signatures*). The rest of this paper is structured as follows. Section 2 evaluates other eID card initiatives. The requirements and notations are respectively discussed in section 3 and 4. Section 5 gives an overview of the protocols. The design and prototype are evaluated and compared to other solutions in section 6. This paper ends with general conclusions.

2 Related work

Many European countries are currently taking initiatives to roll-out electronic identity technology. The approaches can be classified into five categories, namely solutions based on (either contact or contactless) smart cards, password based solutions, solutions in which citizens can manage electronic identities on mobile devices, technology independent solutions and combinations thereof. Smart cards are by far the most widely adopted. Electronic identity cards allow individuals to *identify*, to *authenticate* and/or to *sign electronic documents*. Two cryptographic architectures for electronic identity cards are very popular. Either each card shares one or more personal certificates (and corresponding private keys) in which attributes are embedded or multiple cards keep a common asymmetric key pair and a set of (non-)certified attributes. Many countries (like Belgium) apply the former strategy. The latter approach is adopted by the German eID card. The rest of this section evaluates the opportunities and constraints of the design of the Belgian and German eID card. We thereby mainly focus on *electronic authentication*. These initiatives are representative for the two categories mentioned above (although many variants exist).

The Belgian electronic identity (BeID) [10, 4] card contains personal information of the card owner, written in three files: an identity file (with the card holder's name, NRN[3], place and date of birth and some card related information), a picture and an address file (with the current place of residence). Each file is signed by the government. The BeID card also stores two personal asymmetric key pairs (and corresponding certificates). The private keys SK_{sig} and SK_{auth} are stored in a tamper-resistant part of the chip. SK_{sig} is used for signing electronic documents; SK_{auth} is used for authentication. Some personal information, such as the NRN and name, is also included in the card owner's certificates. Any application can read out the identity, picture and address files and the certificates without the user's consent (i.e. this operations does not require the user to enter a PIN). SK_{sig} and SK_{auth} are activated by a PIN code. Many Belgian eID applications support mutual authentication: the service provider authenticates to the browser and the card authenticates to the server. The BeID card, however, cannot verify the authenticity of the service provider. This may lead to man-in-the-middle-attacks as discussed in [9]. Moreover, SK_{auth} remains activated after authentication. Malicious applications can, therefore, authenticate to several service providers, from which sensitive personal information such as tax declaration info, bank account information, etc.) can be retrieved. These consecutive authentications remain transparent to the user. Moreover, the card has weak privacy properties: during authentication, users can be uniquely identified since their name and NRN is included in the authentication certificate. This also allows collaborating service providers to link authentication sessions and user profiles. Moreover, the data that are released during and after authentication are certified (i.e. the information in the authentication certificate and identity file is signed, hence, endorsed by the government). Easy deployability and low costs are the major benefits. The government does not need to certify service providers and citizen CRLs are maintained by the government. Hence, it is easy to deploy BeID applications. However, the card should not be used for applications with high privacy and security risks, such as providing access to medical records, bank transactions, social networking applications, etc. A similar approach has been adopted by many other European countries [2, 1].

A totally different approach is used by the German eID [3]. A common key pair (SK_{Co}, PK_{Co}) is shared by a large set of cards. SK_{Co} cannot be read out. The key pair is used to set up a secure channel with service providers.

[3] National Registry Number (i.e. a unique identifier for each citizen)

In contrast to the Belgian eID, service providers also authenticate to the card themselves. This avoids man-in-the-middle attacks. The service provider can only query a limited subset of the available personal information on the card. This subset is defined by the government and stored in the service provider's certificate $Cert_{SP}$. Moreover, service provider specific nyms are generated by the card, based on a personal master secret in the card and data in $Cert_{SP}$. Hence, the card has better privacy properties compared to the BeID: users can authenticate pseudonymously/anonymously to service providers, collaborating service providers cannot generate extensive citizen profiles and the veracity of the released information cannot be proven to third parties (i.e. attributes are not certified). However, the government needs to know the master secret of each eID to generate service specific revocation lists (i.e. a separate revocation list is distributed to each service provider). The latter implies a high infrastructural cost.

3 Requirements

The requirements of a general purpose electronic identity card are classified into three categories. The security and privacy requirements are defined by ENISA and extensively motivated in [8]:

- **Security requirements:**
  - S_1: mutual and strong authentication mechanism between the card and service provider/government.
  - S_2: access control to the card (i.e. based on rights/privileges)
  - S_3: secure communication between card and service provider/government
- **Privacy requirements:**
  - P_1: selective disclosure based on the service provider
  - P_2: service provider specific pseudonyms
  - P_3: unlinkability of pseudonyms (even by government)
  - P_4: support for conditional anonymity during authentication[4]
- **Performance and scalability:**
  - O_1: acceptable performance (authentication/release of information)
  - O_2: scalable certification/validation/revocation mechanisms
  - O_3: support for *online* and *offline* services

4 Basic Terminology

Five different roles are defined in the system. The *user (U)* is the owner/holder of an *electronic identity card* (*SC*). *M* represents the software (e.g. middleware) running on the PC with the card reader. A *service provider (SP)* can control access to his services and offer personalized services by requiring user authentication with the eID card. The *government (G)* has three major tasks. First, *G* issues identity cards to citizens (i.e. users). Each card contains a set of public and private keys, certificates and attributes of the card and its owner. Examples of attributes are `chip_number`, `lastValidationTime`, `birth_date`, `address_info`, etc. Second, *G* (re)validates or blocks identity cards at regular times. During (re)validation, the `lastValidationTime` is updated. Third, *G* issues certificates to service providers. The keys and certificates that are maintained in the eID eco-system are listed below:

[4] P_4 allows identification of suspects in case of abuse.

- Each card holds a *unique master secret*, K_U, that is used to generate session keys K_s (see further) and service specific pseudonyms. K_U can either be user-specific or card-specific. The former strategy allows citizens to reuse K_U in multiple eID cards (e.g. when the previous card is defect or lost). However, a secure backup of K_U is required (e.g. by using a key escrow mechanism). If K_U is card-specific, a mechanism must be designed to link (old and new) pseudonyms of the same citizen generated by two different cards.
- A governmental key pair (SK_G, PK_G) is used to set up a secure authenticated communication channel between identity cards and the government. PK_G is placed on each identity card during initialization, the corresponding secret key is only known by the involved governmental service. The card can be updated with a new governmental key by sending the corresponding certificate to the card. This requires proper authentication of G to the card.
- A common key pair (SK_{Co}, PK_{Co}) is equal for a large set of eID 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 government and service providers to verify that a genuine eID is used (without revealing unique identifiers).
- Each service provider has an asymmetric key pair (SK_{SP}, PK_{SP}). SK_{SP} is certified by the government (i.e. $Cert_{SP}$). $Cert_{SP}$ contains at least the service provider's unique name, a validity period and a set of queries (i.e. a set of attributes or properties thereof) that the *SP* can submit to the card. SK_{SP} and $Cert_{SP}$ are used to authenticate to eID cards.
- A session key K_s is used to securely transmit data between a card and a service provider (including governmental services). This session key is generated based upon the hash of the *master secret* K_U and a *counter*.

$\mathcal{H}(.)$ represents a universal hash function. Arrows ($\rightarrow$ or $\leftarrow$) represent the direction of communication. We assume that during a protocol run, the same connection is used. Dashed arrows (`--→` or `←--`) 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. "`String`" represent string constants. Interactions with the card holder are indicated with boxed U.

5 Protocols

This section discusses four protocols, namely card (re)validation, service provider authentication, card authentication, and controlled release of information. The card's validation time can be updated whenever the card is online. The other protocols are typically combined if a user requests access to a service. The user needs to enter his PIN code for each separate service provider.

5.1 Updating the card's `lastValidationTime`

Table 1 illustrates card (re)validation protocol. It results in a new time value on the card that represents the most recent validation time. The time is used during authentication with service providers to prove that the card was valid at that time. An anonymous communication channel can be setup between the middleware and G in order to hide the whereabouts of the card holder.

When the eID card has not recently been revalidated, the user is requested[5] to start the revalidation protocol (1–6). The card first receives a challenge from the government and signs it with SK_{Co} (7–9) resulting in a signature, *sig*. The card then generates a new symmetric key K_s and encrypts it with the public

[5] Note that steps 4–5 are optional. If they are omitted, the card will be automatically revalidated or blocked.

key of the government, PK_G (10–11). Next, *sig*, $Cert_{Co}$ and the `chip_number` are encrypted with K_s (12). The encrypted message E_{msg} and key E_{key} are then sent to the government (13). The government then decrypts E_{key} and the encrypted message, verifies the signature *sig* and checks the status of the `chip_number` (14–18). If the card is still valid, the government encrypts the current time (and its hash) with K_s and sends it to the card (19–21). If the card is not valid, G sends a `"blockCard"` command[6] to the card (18). Finally, upon receiving the encrypted time, the card decrypts it, verifies the integrity, and updates its `lastValidationTime` (22–24). If a block command is received, the card rejects any further authentication requests.

		revalidateCard():
(1)	SC	inserted in reader
(2)	$SC \leftarrow M$	`"Hello"`, *currentTime*
(3)	$SC \rightarrow M$	*reqRevalidation* := (`lastValidationTime` < *currentTime* - δ)
(4)	$M \rightarrow \boxed{U}$	**if** (*reqRevalidation*) showRevalWindow() **else** abort()
(5)	$M \leftarrow \boxed{U}$	**response** [*assume* `Yes`; otherwise abort()]
(6)	$M \dashrightarrow G$	`"RevalidationRequest"`
(7)	G	c := genChallenge()
(8)	$SC \leftarrow M \dashleftarrow G$	`"RevalidateCard"`, c
(9)	SC	*sig* := sign($\mathcal{H}$(c ||"`Revalidation`"|| `chip_number`), SK_{Co})
(10)	SC	K_s := genNewSymKey($\mathcal{H}$(K_U || "`Key`" || `counter++`))
(11)	SC	E_{key} := asymEncrypt(K_s, PK_G)
(12)	SC	E_{msg} := symEncrypt([$Cert_{Co}$, *sig*, `chip_number`], K_s)
(13)	$SC \rightarrow M \dashrightarrow G$	E_{msg}, E_{key}
(14)	G	K_s := asymDecrypt(E_{key}, SK_G)
(15)	G	[$Cert_{Co}$, *sig*, `chip_number`] := symDecrypt(E_{msg}, K_s)
(16)	G	**if** (verifyCert($Cert_{Co}$) == **false**) abort()
(17)	G	**if** (verifySig(*sig*, $\mathcal{H}$(c ||"`Revalidation`"|| `chip_number`), PK_{Co}) == **false**) abort()
(18)	G	**if** (stillValid(`chip_number`) == **false**) sendBlockCommand()
(19)	G	*time* := getCurrentTime()
(20)	G	E_{time} := symEncrypt([*time*, $\mathcal{H}$(*time*)], K_s)
(21)	$SC \leftarrow M \dashleftarrow G$	E_{time}
(22)	SC	[*time*, *hashTime*] := symDecrypt(E_{time}, K_s)
(23)	SC	**if** ($\mathcal{H}$(*time*) != *hashTime*)abort()
(24)	SC	`lastValidationTime` := *time*

Table 1. The card is regularly revalidated by the government.

5.2 Authentication of the service provider

During *SP* authentication (see table 2), the card checks the trustworthiness of the service provider and stores which queries may be performed by the service provider. Note that the communication between the middleware and the service provider can also happen over an anonymous channel.

First, the service provider (*SP*) sends a request for authentication and its certificate $Cert_{SP}$, which still needs to be valid after the most recent card validation time (1–3). The card starts a new session (*sesId*), and copies the *SP*'s name and maximum rights for that session (4–6). Next, the card generates a *session key*, K_s, and a fresh *challenge*, c, encrypts K_s with the *SP*'s public key, and the challenge (with some added redundancy) with K_s, and sends both cyphers to the service provider (7–13) where they are decrypted and verified (14–16). The

[6] The command includes the `chip_number` and its hash, both encrypted with K_s.

incremented challenge, the oldest acceptable validation time and its hash, are reencrypted with the session key and sent to the card (17–18). The card then decrypts and verifies the response (19–21) and checks whether the card is still acceptable to the *SP* (i.e. whether revalidated recently enough) (22).

		authenticateServiceProvider():
(1)	$SC \leftarrow M \leftarrow SP$:	"AuthenticateSP", $Cert_{SP}$
(2)	SC :	**if** (verifyCert($Cert_{SP}$)==**false**) abort()
(3)	SC :	**if** ($Cert_{SP}$.validEndTime < lastValidationTime) abort()
(4)	SC :	$sesId$:= genNewSessionID();
(5)	SC :	session[$sesId$].maxRights := $Cert_{SP}$.maxRights
(6)	SC :	session[$sesId$].Subject := $Cert_{SP}$.Subject
(7)	SC :	K_s := genNewSymKey($\mathcal{H}$(K_U|| "Key" || counter))
(8)	SC :	session[$sesId$].Ks := K_s
(9)	SC :	c := genChallenge($\mathcal{H}$(K_U || "Challenge" || counter++))
(10)	SC :	session[$sesId$].chal := c
(11)	SC :	E_{key} := asymEncrypt(K_s, $Cert_{SP}$.PK)
(12)	SC :	E_{msg} := symEncrypt([c, $Cert_{SP}$.Subject, $sesId$], K_s)
(13)	$SC \rightarrow M \rightarrow SP$:	$sesId$, E_{Key}, E_{msg}
(14)	SP :	K_s := asymDecrypt(E_{key}, SK_{SP})
(15)	SP :	[c, $subject$, ses] := symDecrypt(E_{msg}, K_s)
(16)	SP :	**if** (($subject$!= SP) || (ses != $sesId$)) abort()
(17)	SP :	E_{resp} := symEncrypt([$c+1$, **accValTime**, $\mathcal{H}$(**accValTime**)], K_s)
(18)	$SC \leftarrow M \leftarrow SP$:	$sesId$, E_{resp}
(19)	SC :	[$resp$, **accValTime**, $hashTime$] := symDecrypt(E_{resp}, session[$sesId$].Ks)
(20)	SC :	**if** ($resp$!= session[$sesId$].chal+1) abort()
(21)	SC :	**if** ($\mathcal{H}$(***accValTime***) != $hashTime$) abort()
(22)	SC :	**if** (lastValidationTime < **accValTime**) abort($sesId$)
(23)	SC :	session[$sesId$].auth = **true**
(24)	SC :	session[$sesId$].timeout = τ

Table 2. The service provider authenticates towards the card.

5.3 Authentication of the card

During card authentication, the service provider verifies the trustworthiness of the card. Note that the communication between the middleware and the service provider can also happen over an anonymous channel.

First, the service provider sends an encrypted challenge via the middleware to the card (1–3). The card then decrypts the challenge and the current state of the session (4–6), signs the challenge with the common SK_{Co}, encrypts the outcome and its certificate with the session key and sends it to *SP* (7–9). Finally, the service provider decrypts the card's response and verifies the certificate and the signature (10–12).

5.4 Disclosure of card holder's attributes (or properties thereof)

This protocol (see table 4) allows the service provider to query a subset of the attributes from the card based on privileges (maxRights) assigned by the government.

First, the service provider sends an attributes query to the card, containing identifiers of the attributes/properties he wishes to obtain (1). The middleware intercepts the query and requests the card holder's consent; the card holder can modify (reduce) the query and gives his consent by entering the PIN code of the

		authenticateCard():
(1)	SP :	c := genChallenge()
(2)	SP :	E_{msg} := symEncrypt([c, $sesId$], K_s)
(3)	$SC \leftarrow M \leftarrow SP$:	"AuthenticateCard", $sesId$, E_{msg}
(4)	SC :	[c, ses] := symDecrypt(E_{msg}, session[$sesId$].K_s)
(5)	SC :	**if** ((ses != $sesId$) || closed($sesId$)) abort()
(6)	SC :	**if** (session[$sesId$].auth == **false**) abort()
(7)	SC :	sig := sign($\mathcal{H}$(c || "Auth"), SK_{Co})
(8)	SC :	E_{msg} := symEncrypt([$Cert_{Co}$, sig], session[$sesId$].K_s)
(9)	$SC \rightarrow M \rightarrow SP$:	$sesId$, E_{msg}
(10)	SP :	[$Cert_{Co}$, sig] := symDecrypt(E_{msg}, K_s)
(11)	SP :	**if** (verifyCert($Cert_{Co}$) == **false**) abort()
(12)	SP :	**if** (verifySig(sig, $\mathcal{H}$(c || "Auth"), $Cert_{Co}$.PK) == **false**) abort()

Table 3. The card authenticates towards the service provider.

card (2–3). The possibly modified query and PIN code are forwarded to the card, which checks the PIN code and verifies whether (a) the session is still open, (b) the *SP* has been authenticated properly and (c) the query is consistent with the privileges assigned by the government to the service provider (4–8). If a (service provider specific) pseudonym is required[7], it is generated by the card (9) and the outcome of the query (10) is encrypted with the session key (11–12) and sent to the *SP* which will decrypt it, verify its integrity and process the personal data (13–14). The *SP* can close a session by sending a "LogOff(*sesId*)" command to the card. Any active sessions are also automatically terminated upon removal of the card from the card reader since session data is stored in transient memory.

		releaseAttributes():
(1)	$M \leftarrow SP$:	"attributeQuery", $sesId$, $query$
(2)	$M \rightarrow$ U :	showQueryWin(SP, $query$)
(3)	$M \leftarrow$ U :	**response** [Assume **OK** [$query^*$, **PIN**]; otherwise abort()]
(4)	$SC \leftarrow M$:	"attributeQuery", $sesId$, $query^*$, **PIN**
(5)	SC :	**if** (PINincorrect(**PIN**)) handleWrongPIN()
(6)	SC :	**if** (closed($sesId$)) abort()
(7)	SC :	**if** (session($sesId$).auth == **false**) abort()
(8)	SC :	**if** ($query^* \not\cong$ session[$sesId$].maxRights) abort()
(9)	SC :	nym_{SP} := $\mathcal{H}$(K_U|| session[$sesId$].Subject)
(10)	SC :	$queryResults$:= solveQuery($query^*$)
(11)	SC :	$data$:= [nym_{SP}, $queryResults$, $\mathcal{H}$(nym_{SP} || $queryResults$)]
(12)	$SC \rightarrow M \rightarrow SP$:	$E_{attributes}$:= symEncrypt($data$, session[$sesId$].Ks)
(13)	SP :	[nym_{SP}, $queryResults$, $hashData$] := symDecrypt($E_{attributes}$, K_s)
(14)	SP :	**if** ($\mathcal{H}$($data$) != $hashData$) abort()

Table 4. The card releases attributes to the authenticated service provider.

6 Discussion

This section evaluates the performance properties, security/privacy properties and usability/deployability properties against the requirements discussed in section 3. We also compare our approach with existing alternatives.

[7] Otherwise, this step is omitted.

Performance properties The prototype is developed on a TOP IM GX4 [5] smart card. The performance measurements are based on 1024 and 2048 bit RSA keys. Table 5 clearly shows that asymmetric private key operations are expensive compared to other crypto operations. Omitting the communication overhead, the performance of the protocols is, therefore, linear to the number of private key operations. Table 6 shows the number of required cryptographic operations

	RSA		**AES**			**SHA1**
Key length in bits	1024	2048	128	192	256	-
Verification	26	60	-	-	-	-
Signing	555	1655	-	-	-	-
Encryption	24	60	19	21	23	-
Decryption	553	1654	24	27	30	-
Hashing	-	-	-	-	-	2

Table 5. Average timing results for cryptographic operations on the TOP IM GX4 smart card. Tests are done with 128 bytes data, times are in ms.

for the Belgian eID and the general purpose eID. Card validation and service consumption are evaluated separately. Our protocols require considerably more cryptographic operations on the card compared to the Belgian eID. However, only one signature is performed on the card (which is equal to the Belgian eID). Since this is by far the most time consuming operation, the total execution time does not considerably increase. Service consumption does imply a considerable computational overhead towards the service provider compared to the Belgian eID. This is, however, similar to initiating an TLS/SSL session, which is widely adopted by service providers for secure transfer of data. Moreover, with the Belgian eID, OCSP [7] is often used to validate user certificates, which introduces extra communication overhead.

	Belgian eID			**General Purpose eID**			
	Service consumption			**Card validation**		**Service consumption**	
Actors	SrvPr.	Gov.	Card	Gov.	Card	SrvPr.	Card
Verify/asEncrypt	3	0	0	2	1	2	2
Sign/asDecrypt	0	1	1	1	1	1	1
Symm. operation	0	0	0	2	2	5	5
Hashes	0	0	0	3	3	4	7

Table 6. Overview of the required operations during authenticaton.

We reserved 6500 bytes for identity files (including a small picture). Multiple keys also need to be stored in persistent memory: one key pair (SK_{Co}, PK_{Co}) requires 472 bytes, two public keys (PK_G and $PK_{SP_{CA}}$) require 2*172 bytes and a certificate ($Cert_{Co}$) estimated at 600 bytes. Finally, a `master secret` and `counter` are also stored on the card, each requiring 20 bytes. The sum is 7956 bytes (excluding the program code). Multiple attributes are stored for each session, typically in transient memory, defining the context of that session. The storage for each session is lower than 100 bytes.

Security properties First, according to S_1, *strong mutual authentication* is realized between the service provider (or government) and the card. This prevents man-in-the-middle attacks. Second, service providers have *restricted access* to the personal information stored on the card (cfr. S_2). The government can update the `lastValidationTime` and block revoked cards. Only certified service

providers can access the card holder's personal information. The government defines which attributes or properties thereof can be queried by each service provider. In contrast to the BeID, a user must reenter his PIN code each time when personal information needs to be disclosed to a *SP*. Third, the approach provides *a flexible and efficient revocation mechanism* (cfr. O_2) (see also *privacy properties*). No CRL/OCSP checks are required by the service provider nor the card. At the service provider's side, this is realized with certificates that have a short lifetime. It is, however, possible to check the validity of the server certificate with an OCSP check initiated by the card (i.e. the card can forward – via the middleware – the serial number to the government OCSP server over a secure connection). Alternatively, although less secure, the OCSP check can also be initiated by the middleware. A card does not need to be revalidated before each authentication session. The policy of the service provider decides the validity interval for `lastValidationTime`. Short intervals are more secure and appropriate for online services (i.e. retrieving a new `lastValidationTime` from the government requires no substantial overhead and can be initiated by client middleware as soon as it detects that an eID card is inserted in a reader). Offline services (e.g. vending machines for alcoholic beverages or cigarettes) typically allow a larger validity interval. A reasonable trade-off between security and usability must be found. Finally, *no real random generator* is required on the card: SK_{Co} and the *master secret*, K_U, are generated at set-up (by means of a certified/trusted module); *session keys* and *challenges* are based on K_U and a *counter*.

Privacy properties Access policies (cfr. P_1) are defined by the government. A dedicated attribute in each *SP*'s certificate defines the maximal information that can be retrieved from the card. Users can decide to release only a subset of information but then some limited trust is required in the client middleware or an advanced card reader which enables the user to restrict the query of the *SP*[8]. Every release of personal information requires the user to enter the PIN. Every query for personal information, therefore, requires explicit consent of the user.

No link between the user's identity and the transaction at the service provider is possible, even if government and service provider collaborate (unless uniquely identifying information is released). We assume that the master secret, K_U, is only known by the card. Similar to the German eID, collaborating service providers cannot link information to the same user (unless uniquely identifying info is released) (cfr. P_2 and P_3). Conditional anonymity (cfr. P_4) is also possible. The service provider can request a probabilistic encryption of the user's real name, SSN or serial number with the public key of a trusted third party (TTP). The cyphertext can then be decrypted by that TTP. The public key of the TTP can be embedded in the certificate of the service provider. Hence, each service can have its own TTP (or deanonymizer) if it is approved by the government.

Usability and scalability properties Although our approach has similar privacy properties compared to the German eID, our solution provides a better separation of concerns and, hence, is more scalable (cfr. O_2). First, the revalidation service can be administered by an external entity. It replaces CRL/OCSP checks by the service provider. Note that OCSP checks may link service requests to serial numbers (and indirectly to identities). Second, the certificates of service providers have a relatively short validity period. This implies a short window of vulnerability.
The complexity of revocation strategy is linear to the number of citizens whereas,

[8] It is impossible to increase the *SP*'s rights; hence, the only abuse that is possible, is that the card holder's restrictions are ignored.

in the German eID, a separate revocation list must be generated for each service provider (i.e. linear to number of citizen multiplied by number of services).
The proposed approach can be applied to a wide variety of *online services* (such as webservices) and *offline services* (such as cigarette vending machines) (cfr. O_3). The user only needs to update the `lastValidationTime` regularly. For this service, no trust is required in the client workstation on which the card is inserted. The `lastValidationTime` can be updated each time a user has Internet access. For offline services, an administrator does not need to update CRLs at each vending machine (which is a serious maintainance barrier in the German eID architecture). No PIN-pad reader is necessary for card revalidation. However, it prevents PIN caching. A screen can display info kept in $Cert_{SP}$ (i.e. ensure the card holder that the right service provider is contacted) and that only the minimal amount of personal information will be disclosed. Such a reader reduces the level of trust required in client middleware. Finally, the proposed solution is easily extensible to prove attribute values or properties thereof that are even not stored at the card itself. For instance, a *car rental company* may request the card holder to prove to be the owner of a `valid driver's license`; an *insurance company* may request `family status information` (such as marital status, number and age of children ...). Some of these attributes are not stored at the card due to limited storage space or because they may change too frequently. $Cert_{SP}$ can list a set of queries $Q_1, ..., Q_N$ to other service providers SP_1, ..., SP_N that cannot be answered by the card alone. If so, the card will, after validation of $Cert_{SP}$ and explicit consent of the user, initiate a connection to SP_1, ..., SP_N to retrieve the requested values and forward them to the initial service provider SP. This approach is more reliable and scalable than the one that is currently applied in Belgium (and in many other countries). For instance, the Crossroads Bank for Social Insurance organizes and executes data transfer between multiple institutions and service providers. A service provider typically sets up a connection to other service providers. However, in many cases, no explicit user consent is required. Our solution also allows data transfer between service providers and has two major benefits. First, the access policy is kept in $Cert_{SP}$ (i.e. no central policy decision point is required). Second, the user's explicit consent is required.

7 Conclusion

This paper presents the design of a general purpose electronic identity card. The approach is based on a time service that updates (blocks) valid (revoked) cards. We extensively argued the benefits of this approach compared to solutions that are currently developed or even rolled out. More specifically, the proposed solution compromises a strong level of *security and privacy* and high *scalability and performance*. The former is required when the card is used outside the governmental domain (e.g. banking services and social networking). The latter is required as the number of services increases.

References

1. Estonian identity card. `http://www.id.ee/`.
2. Portuguese citizen card. `http://www.cartaodocidadao.pt/`.
3. Advanced security mechanisms for machine readable travel documents - extended access control (eac) and password authenticated connection establishment (pace). Technical Guideline TR-03110, 2008.
4. Danny De Cock, Christopher Wolf, and Bart Preneel. The Belgian Electronic Identity Card (Overview). In Jana Dittmann, editor, *Sicherheit 2005: Sicherheit* -

Schutz und Zuverlässigkeit, Beiträge der 3rd Jahrestagung des Fachbereichs Sicherheit der Gesellschaft für Informatik e.v. (GI), volume LNI P-77 of *Lecture Notes in Informatics (LNI)*, pages 298–301, Magdeburg,DE, 2006. Bonner Köllen Verlag.
5. Gemalto. Top gx4, product information. `http://www.gemalto.com/products/top_javacard/download/TOP_GX4_Nov08.pdf`, 2008.
6. Giles Hogben Ingo Naumann. Privacy Features of European eID Card Specifications. Technical report, ENISA, 2009.
7. A.Malpani S. Galperin C. Adams M. Meyers, R. Ankney. Rfc2560: X.509 internet public key infrastructure, online certificate status protocol - ocsp. `http://tools.ietf.org/html/rfc2560`, June 1999.
8. Ingo Naumann. Privacy and Security Risks when Authenticating on the Internet with European eID Cards. Technical report, ENISA, 2009.
9. B. De Decker V. Naessens P. Verhaeghe, J. Lapon and K. Verslype. Security and privacy improvements for the belgian eid technology. pages 237–248, May 2009. Proceedings of the 24th IFIP International Information Security Conference.
10. Marc Stern. *Belgian eID Card content.* Zetes, CSC, 2.2 edition, 2003.