Extending the Belgian eID technology with mobile security functionality

Jorn Lapon[1], Bram Verdegem[1], Pieter Verhaeghe[2],
Vincent Naessens[1], Bart De Decker[2]

[1] Katholieke Hogeschool Sint-Lieven, Department of Industrial Engineering
Gebroeders Desmetstraat 1, 9000 Gent, Belgium
[2] Katholieke Universiteit Leuven, Department of Computer Science,
Celestijnenlaan 200A, 3001 Heverlee, Belgium

Abstract. The Belgian Electronic Identity Card was introduced in 2002. The card enables Belgian citizens to prove their identity digitally and to sign electronic documents. Today, only a limited number of citizens really use the card in electronic applications. A major reason is the lack of killer functionality and killer applications.

This paper presents two reusable extensions to the Belgian eID technology that opens up new opportunities for application developers. First, a secure and ubiquitously accessible remote storage service is presented. Second, we show how the eID card can be used to issue new certificates. To demonstrate the applicability and feasibility of both extensions, they are combined in the development of a secure e-mail application. The proposed solution offers strong privacy, security and key management properties while increasing the accessibility of confidential e-mail compared to existing solutions (such as PGP and S/MIME).

Key words: Identity Technology, Security, Privacy, Mobile Access

1 Introduction

In 2002, Belgium has introduced an electronic identity card (eID) [1] as one of the first countries in Europe. The card enables individuals to prove their identity digitally and to sign electronic documents. The Belgian eID card opens up new opportunities for the government, their citizens, service providers and application developers. Although many eID applications have been developed, the success of the Belgian eID technology is still limited. A major reason is the lack of essential functionality and, more importantly, real killer applications. Moreover, the use of the current eID card involves a few security and privacy hazards [2, 3].

This paper presents two enhancements to the current Belgian eID technology while addressing certain privacy shortcomings. A first extension defines a service that allows users to store and update sensitive data (such as passwords, keys, tickets, . . .) securely at a remote location. The service is ubiquitously accessible with the Belgian eID card. Moreover, the data that is kept at the server is useless to internal and external attackers. A second extension defines the creation and use of proxy certificates that are certified by means of the Belgian eID card. Hence, an individual can create new certificates that can be used to send confidential messages, to setup a mutually authenticated secure channel between two individuals, etc. An external certificate authority is no longer required to issue these certificates.

Both extensions are bootstrapped by means of the Belgian eID Card and can be integrated in many applications. To demonstrate their usefulness, both extensions have been incorporated in a ubiquitously accessible secure e-mail service. This service allows individuals to send confidential (encrypted) and signed messages using certificates that can be validated by certificate chains. The approach is more secure than PGP (which is based on trust levels) and tackles some key management problems in S/MIME (i.e. confidential e-mail can be retrieved from any location at which a web browser, Internet access and a card reader is available).

The paper is structured as follows. Section 2 gives an overview of the Belgian eID card technology. Section 3 describes the notation used in the rest of the paper. Two extensions to the Belgian eID technology are proposed in section 4 and section 5. Both extensions are evaluated in section 6 and validated through the development of a secure e-mail application in section 7. Finally, the paper draws some conclusions and describes directions for future research.

2 Belgian Electronic Identity Card technology

The *Belgian eID card* is a smart card that allows Belgian citizens to prove their identity visually and digitally and to sign electronic documents [4]. The eID card contains three files: (1) a digital picture of the citizen, (2) an identity file which contains the basic identity information and a hash value of the picture file; this file is signed by the National Registry, (3) an address file which contains the citizen's current residence; it is signed by the National Registry together with the identity file to guarantee the link between both files.

Two private keys SK_{Auth} and SK_{Sig} are stored in the eID card. These keys are used for digital authentication and signing respectively. They are stored in a tamper-proof part of the chip and can be activated with a PIN code. Each corresponding public key (PK_{Auth} and PK_{Sig}) is certified by a certificate. Each certificate also keeps the name of the card holder and his nation-wide identification number (i.e. the National Registry Number or NRN).

The Belgian government offers a *middleware* package [5, 6] to facilitate interaction with the eID card. The middleware contains a GUI to enable end-users to read the files and the certificates that are stored in the eID card and to change the PIN code. Moreover, the middleware acts as an intermediary for all accesses to the eID card by other applications. If a document has to be signed, the middleware passes a hash of the document to the card. Similarly, a hash of the challenge is passed to the card for authentication purposes. When an application wants to authenticate or sign a document with the eID card, the middleware asks the user for a PIN code and forwards it to the eID card. The middleware can also check the validity of certificates (using CRL or OCSP). Note that the middleware is not essential: an application can also implement the middleware functionality and directly interact with the card.

The certificates on the eID card are part of a larger hierarchical infrastructure, the *Belgian Public Key Infrastructure* [7]. The signature and authentication certificates are issued by a *Citizen_CA* and the certificates of each *Citizen_CA* are issued by the *Belgium_Root_CA*, that is found at the top of the eID hierarchy. In addition to the CA hierarchy, the PKI defines Certificate Revocation Lists [8] that contain the serial numbers of revoked certificates issued by that CA.

The government aims at encouraging the use of the eID card in both *e-government and commercial applications.* Currently, most applications use the Belgian eID card for setting up an SSL connection with mutual authentication. Many other applications (such as physical access control) only retrieve the identity information stored on the card.

3 Notations

The following notation is used throughout this paper:

- $\mathrm{P}_1 \leftrightarrow \mathrm{P}_2$: $NRN \leftarrow \mathsf{authenticate}_{\mathsf{eID}}()$ represents an interactive protocol, in which P_1 uses SK_{Auth} in the eID card to authenticate to P_2. As a result, P_2 obtains P_1's NRN.
- P : $sig \leftarrow \mathsf{sign}_{\mathsf{eID}}(\mathsf{hash}(M))$ denotes a user signing the hash of message M, with SK_{Sig} in the eID card.
- P : $K \leftarrow \mathsf{createSymmetricKey}(PRG, seed)$ denotes the generation of a secret/symmetric key, based on a pseudorandom number generator PRG and a *seed.* Use of the same PRG and *seed* will always generate the same key.
- P : $\mathsf{store}(data; index)$ means that *data* is stored in a database at location *index.*

4 Mobile access to secrets

A major challenge in today's society is to enable access to sensitive data (e.g. personal information, secret keys, ...) from various locations. Data must be stored securely on an easily accessible remote server. Data is typically encrypted by the owner before it is stored on the server. Hence, the encrypted data is useless to the server or any adversary that may have access to it. In this section a scheme is discussed based on the Belgian eID card for securely storing sensitive data. A trivial solution consists of encrypting the data with the public key of the authentication certificate. Whenever needed, the ciphertext is fetched from the server and decrypted using the corresponding private key.

However, encryption/decryption with the Belgian eID card is not possible. Moreover, when the card is lost or renewed, decryption of previously encrypted information is no longer possible. Therefore, a new mechanism is proposed that uses the Belgian eID card as a bootstrap. A secret/symmetric key is derived from a signature generated by the eID card. This secret key is used for encrypting data before it is stored. When a user wants to retrieve his confidential information, the encrypted information is fetched from the server and decrypted with the secret key, regenerated using the same eID card. In the following paragraphs, the protocols for storing and retrieving sensitive data are discussed in more detail.

$\mathsf{storeSensitiveData}(data,\ tag)$:

(1)	U	:	$H_{keyGen} \leftarrow \mathsf{hash}_{\mathsf{A}}('KEYGEN' \Vert NRN \Vert CardNumber \Vert otherUserInfo)$
(2)	U $\leftrightarrow$ C	:	$sig \leftarrow \mathsf{sign}_{\mathsf{eID}}(H_{keyGen})$
(3)	U	:	$K_S \leftarrow \mathsf{createSymmetricKey}(PRG, \mathsf{hash}_{\mathsf{B}}(sig))$
(4)	U	:	$E_{tag} \leftarrow \mathsf{encrypt}(tag, K_S)$
(5)	U	:	$R \leftarrow \mathsf{generateRandom}()$
(6)	U	:	$K_D \leftarrow \mathsf{createSymmetricKey}(PRG, \mathsf{hash}_{\mathsf{C}}(sig \Vert R))$
(7)	U	:	$E_{data} \leftarrow \mathsf{encrypt}(data, K_D)$
(8)	U $\leftrightarrow$ S	:	$NRN \leftarrow \mathsf{authenticate}_{\mathsf{eID}}()$
(9)	U $\rightarrow$ S	:	(E_{data}, R, E_{tag})
(10)	S	:	$index \leftarrow \mathsf{hash}_{\mathsf{D}}(NRN \Vert E_{tag})$
(11)	S	:	$\mathsf{store}([E_{data}, R]; index)$
(12)	U $\leftarrow$ S	:	$(Timestamp, \mathsf{sign}_{\mathsf{S}}(\mathsf{hash}_{\mathsf{E}}([Timestamp, E_{tag}, E_{data}, R])))$

Table 1. Storing sensitive data remotely.

Storing sensitive data. Table 1 shows the steps for storing sensitive data. The user provides the *data* to be stored and a secret *tag* that serves as a name or alias of the data.

First, the Belgian eID card is used to generate secret keys as follows. A message with a fixed format is hashed using the middleware (1). The message consists of a header *'KEYGEN'*, the *NRN*, the serial number of the eID card and possibly some other user information, making the message card specific. The hash, H_{keyGen}, is then signed with the signature key on the eID card resulting in a 1024 bits signature *sig* (2). The fixed format prevents that an adversary obtains *sig* by requesting a signature on a forged message; any message to be signed by the eID card should not match this format, except in this protocol. Subsequently, *sig* is used as the seed for generating two new symmetric keys, namely K_S and K_D. To ensure that the same keys are generated, independent of the platform, a specific pseudo-random generator *PRG* is passed as a parameter of the $\mathsf{createSymmetricKey}$ function; also, the hash-functions used should be fixed. K_S is used for encrypting the secret *tag* associated with the data. The key is derived from the hash of *sig* (3-4). The encrypted tag E_{tag} is used in step 10 of the protocol to derive an *index* to a record in the server database. K_D is used for encrypting the *data*. Each time information is stored (i.e. each time the protocol is invoked), a new random number R is associated with it (5). From the hash of R and *sig* the encryption key K_D is derived (6) with which the *data* is encrypted into the ciphertext E_{data} (7). In each execution of $\mathsf{storeSensitiveData}$ the data will be encrypted with another key; this prevents linking of the same encrypted data that is stored in more than one location.

Next, the encrypted data E_{data} is stored on a remote server S. First, the user

authenticates with his eID, to ensure that later only the owner U can retrieve his data (8). Next, U sends the encrypted tag E_{tag} and the encrypted data E_{data} to S (9). The NRN, disclosed during the authentication, is hashed together with E_{tag} and will be used as an *index* to the information that is stored in the server database (10). The use of the encrypted secret tag, E_{tag}, ensures the user's privacy, while making the *index* tag specific. A different *tag** will result in another index. If the server S is trustworthy and does not store the user's NRN nor the E_{tag}, dictionary attacks on the *index* are no longer feasible. An adversary with full access to the server data cannot link any data to a particular citizen.
Finally, the database stores a record with the encrypted data and the random number R at location *index* (11). Although useless to the server or any other adversary, the random number R is necessary for the owner of the data to derive the correct symmetric key for decrypting E_{data}. A receipt is sent to the user, certifying the proper storage of the encrypted data (12).

$\mathsf{retrieveSensitiveData}(tag)$:

(1) U	:	$H_{keyGen} \leftarrow \mathsf{hash_A}('KEYGEN' \| NRN \| CardNumber \| otherUserInfo)$
(2) U ↔ C	:	$sig \leftarrow \mathsf{sign_{eID}}(H_{keyGen})$
(3) U	:	$K_S \leftarrow \mathsf{createSymmetricKey}(PRG, \mathsf{hash_B}(sig))$
(4) U	:	$E_{tag} \leftarrow \mathsf{encrypt}(tag, K_S)$
(5) U ↔ S	:	$NRN \leftarrow \mathsf{authenticate_{eID}}()$
(6) U → S	:	$\mathsf{requestRecord}(E_{tag})$
(7) U ← S	:	$record \leftarrow \mathsf{getRecord}(\mathsf{hash_D}(NRN \| E_{tag}))$
(8) U	:	$K_D \leftarrow \mathsf{createSymmetricKey}(PRG, \mathsf{hash_C}(sig \| record.R))$
(9) U	:	$data \leftarrow \mathsf{decrypt}(record.E_{data}, K_D)$

Table 2. Retrieving sensitive data remotely.

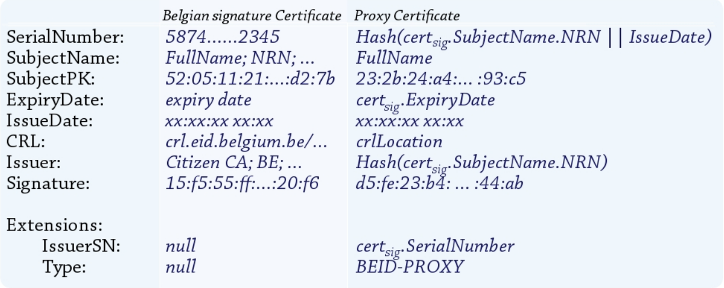
Retrieving sensitive data. When the data has been stored on the remote server, it can be retrieved by the owner from anywhere (see table 2). First, the hash H_{keyGen} is regenerated (1) and signed with the eID card (2). With the signature sig, the secret key K_S (3) is regenerated for encrypting the secret tag (4). Next, the user authenticates with his eID card (5). The encrypted secret tag, E_{tag} is sent to the server S and the corresponding record requested (6). The server S fetches the record with $index = \mathsf{hash_D}(NRN \| E_{tag})$ from his database; the record, comprising of encrypted data and the random number, is sent to U (7). U can now regenerate the data specific secret key K_D from the hash of sig and R (8). Finally, U decrypts E_{data} with the secret key K_D (9).

Recovery of keys. This scheme exploits the property that a deterministic signature algorithm is implemented in the eID card. When creating a signature with the eID card, the same input (H_{keyGen}), always results in the same signature sig. As such, using the same PRG and the same hash-functions ($\mathsf{hash_A}$ .. $\mathsf{hash_D}$), the same secret keys K_S and K_D (for a certain R) are always regenerated. However, in case of loss or renewal of the eID card, PK_{Sig}, SK_{Sig} and *CardNumber* will have changed and the generated signature *sig** no longer matches the signature sig (generated by the previous eID card), impeding the localization and decryption of the stored information. Therefore, to protect important data, a secured backup of the signature *sig* is created the first time this signature is generated. In case of loss or renewal, *sig* is restored from the backup and the keys can be recovered. This secured backup could be provided by a key escrow service.

5 Proxying the Belgian eID

As illustrated above, it is now possible to perform symmetric encryption based on the Belgian eID card. However, when other parties are involved, asymmetric encryption may be required. Therefore, a second encryption scheme based on the eID card is proposed.

Proxy certificates. Although the eID card itself cannot encrypt nor decrypt data, the right to do this can be delegated to the host. This restricted delegation and proxying to another entity is achieved through proxy certificates [9]. A proxy certificate is an extended version of a normal X.509 certificate. Proxy certificates can be derived from and signed by a normal X.509 certificate or by another proxy certificate. Once a proxy certificate is created, the proxy certificate and its corresponding private key can be used for asymmetric encryption resp. decryption.
Modified standard. The standards for proxy certificates, as defined in the RFC 3820, impose some problems. Therefore, some modifications have to be made (see figure 1). First, according to the RFC, proxy certificates should be issued by the authentication certificate, since only this certificate of the Belgian eID contains the required *key-usage* attribute. The *key-usage* attribute defines the purpose of the public key contained in the certificate. However, using the authentication key SK_{Auth} is less secure than using the signature key SK_{Sig} since the PIN is only required for the first authentication (Single Sign On feature), while it is required for every signature. Therefore, SK_{Sig} is used to issue proxy certificates. Second, the Belgian legislation prohibits to store the *NRN*. However, the *NRN* is stored in the subject field of the eID certificates. This implies that the eID certificates may not be stored. However, the proxy certificate standard defines that the subject of the issuing certificate is copied into the issuer and subject fields of new proxy certificates. To solve this problem, the name of the owner is copied into the subject field and the hash of the *NRN* is copied into the issuer field instead of the subject of the eID certificate. For validation purposes, the serial number of the issuer certificate is also included in the certificate (i.e. *issuerSN*). Additionally, another extra attribute indicates the type: `BEID-PROXY`. In this scheme, we assume that when the eID certificate is revoked (e.g. in case of loss or theft), the issued proxy certificates are no longer valid. Moreover, the proxy certificate must expire before the expiration date of the eID certificate. The protocol in table 3 demonstrates the creation of a BeID proxy certificate. The user *U* generates an asymmetric key-pair (1). A serial number is generated from the hash of the *NRN* and the *issueDate* of the new proxy certificate (2-3). The *NRN* in the hash avoids collisions, while the *issueDate* enables users to have



	Belgian signature Certificate	*Proxy Certificate*
SerialNumber:	*5874......2345*	*Hash(*$cert_{sig}$*.SubjectName.NRN* $\Vert$ *IssueDate)*
SubjectName:	*FullName; NRN; ...*	*FullName*
SubjectPK:	*52:05:11:21:....:d2:7b*	*23:2b:24:a4:... :93:c5*
ExpiryDate:	*expiry date*	$cert_{sig}$*.ExpiryDate*
IssueDate:	*xx:xx:xx xx:xx*	*xx:xx:xx xx:xx*
CRL:	*crl.eid.belgium.be/...*	*crlLocation*
Issuer:	*Citizen CA; BE; ...*	*Hash(*$cert_{sig}$*.SubjectName.NRN)*
Signature:	*15:f5:55:ff:...:20:f6*	*d5:fe:23:b4: ... :44:ab*
Extensions:		
IssuerSN:	*null*	$cert_{sig}$*.SerialNumber*
Type:	*null*	*BEID-PROXY*

Fig. 1. Content of the modified proxy certificate

more than one proxy certificate. A proxy certificate *proxyCert* is then generated and certified with SK_{Sig} of the eID card (4).

```
createBeIDProxy([attributes]):
(1) U    :  $(SK_U, PK_U) \leftarrow \mathsf{generateKeyPair}()$
(2) U    :  $issueDate \leftarrow \mathsf{getDate}()$
(3) U    :  $serialNb \leftarrow \mathsf{hash}(NRN||issueDate)$
(4) U    :  $proxyCert \leftarrow \mathsf{generateProxy}(\texttt{BEID-PROXY}, cert_{Sig}, serialNb,$
               $PK_U, issueDate, crlLocation, [attributes]; SK_{Sig})$
```

Table 3. Create a new proxy certificate.

Revocation. A special purpose server R_p can publish revoked proxy CRLs. To revoke a proxy certificate (cfr. table 4), the user authenticates with his eID card (1) and sends the proxy certificate he wants to revoke (2). If the issuer corresponds to the eID signing certificate (i.e. $\mathsf{hash}(cert_{Sig}.NRN) = proxyCert.Issuer$), the proxy certificate is revoked by adding its serial number to the latest CRL.

```
revokeBeIDProxy(proxyCert):
(1) $U \leftrightarrow R_p$  :  $NRN \leftarrow \mathsf{authenticate_{eID}}()$
(2) $U \rightarrow R_p$      :  $\mathsf{revokeCertificate}(proxyCert.serialNumber)$
(3) S                        :  $\mathsf{if}\ (\mathsf{hash}(cert_{Sig}.SubjectName.NRN) \neq proxyCert.Issuer)\ \mathsf{abort}$
(4) $U \leftarrow S$         :  $true \leftarrow \mathsf{addToCRL}(proxyCert.serial)$
```

Table 4. Revoke a proxy certificate account.

Validation. A receiver validates a new proxy certificate by checking the validity period, its revocation status and by verifying the rest of the certificate chain. Since the hash of *NRN*is kept in the issuer field, name chaining (cfr. RFC 3280 [10]) for certification path validation will fail. However, the extra attribute *issuerSN* included in the certificate binds the eID certificate to the proxy certificate. The first step in creating the certification path is thus modified. The serial number of the eID certificate must match the *issuerSN* in the proxy certificate. To comply with Belgian legislation, the eID certificate is removed after validation. Hence, future validation is not possible. However, verifying the validity period and the revocation status suffices. This can be performed as the proxy certificate is stored at a trusted location. Additionally, the revocation status of the eID certificate can be verified by checking the *issuerSN* of the proxy certificate in the CRLs of the Belgian eID.

Belgian citizens can now create legitimate *proxy certificates* themselves, that can be used in many applications. Moreover, once a proxy certificate has been created, the Belgian eID is no longer required.

6 Discussion

The extensions discussed above promise new opportunities for the Belgian eID technology. Not only can the eID card be used for digital signatures or authentication, the eID technology can also be used to store and to retrieve sensitive data. The user only needs his card to access the encrypted data that is stored on the remote server. However, adversaries might try to retrieve the secret keys by continuously sending challenges to the eID card. Our solution tackles the thread by using the signature key in the eID card. The latter requires a PIN for every signature in contrast to the authentication key, which only requires a PIN once. Moreover, the fixed format of the message allows to detect trojan horses or malicious applications that request users to sign a certain message.

To support recovery of sensitive data if the eID card is lost or invalid, a secure backup of the signature *sig* needs to be created the first time this signature is generated. However, the user remains responsible for making the secure backup. An alternative approach is to support a key-escrow mechanism [11, 12]. The secret *sig* is then split into n parts using a secret sharing algorithm and each part is stored on a different escrow server. To reconstruct the secret, all n parts are retrieved from the escrow servers and the interpolation of the parts results in the secret. To ensure that users only obtain their own keys, eID authentication can be used with the escrow servers. Additionally, a hash of *NRN* and the serial number of the eID card can be combined as an index to store a part of the secret. Every citizen can –online– lookup his current and previous card numbers at the National Registry.

The *proxy certificate* mechanism allows owners of an eID card to setup mutually authenticated secure channels without the need for a trusted third party. Secure communication is even no longer restricted to SSL. The proposed system with proxy certificates makes it more flexible and extensible. Although not completely complying with the standards, the proposed scheme supports asymmetric *encryption* with the eID card. Moreover, the proxy certificates can be used for asynchronous communication (i.e. recipients can decrypt confidential messages after the communication channel is closed). Once the sender has deleted the eID certificate (as imposed by Belgian legislation), he can still check the validity of the proxy certificate and the revocation status of the eID certificate. Although other certificates in the validation path (i.e. *Citizen_CA*, *Belgium_Root_CA*, . . .) may have been revoked, the proxy certificate can include the issuer of the eID certificate as an extra attribute to verify the validity of the rest of the certificate chain. Note that optional attributes in the proxy certificate may further restrict its use.

7 A mobile and secure e-mail client

Both extensions discussed above are combined into a proof-of-concept application. An e-mail client has been developed. It supports exchanging confidential e-mail messages using the BeID proxy mechanism described in section 5. Individuals can use the e-mail service at any location as the necessary key material and contact information are stored securely on an easy accessible remote server.

7.1 Requirements

- R_1: The secure e-mail application is simple to use.
- R_2: The e-mail service is ubiquitously accessible.
- R_3: Certificates can be revoked (using CRL, OCSP).
- R_4: Contact information is kept private.
- R_5: Data stored on a remote server is useless to any third party.

7.2 Protocols

In this discussion, abstraction is made of the message format and the e-mail system that is responsible for the transport of e-mail messages. The protocols are designed to be compatible with existing e-mail systems.

setupProxy-Protocol (cfr. table 5). This protocol defines the creation and storage of an *e-mail* proxy certificate. The user U generates a proxy certificate with his e-mail address as an extra attribute (1). Next, a list of already existing credentials *creds* (i.e. certificates and corresponding private keys) is fetched from the remote secure store (2). The new credential consisting of SK_{proxy} and *proxyCert* is added to the credential list (3). Finally, the updated list is uploaded to the remote store (4).

setupProxy():

(1) U	:	$(SK_{proxy},\ proxyCert) \leftarrow \mathsf{createBeIDProxy}([E - mail : U.email])$
(2) U ← S	:	$creds \leftarrow \mathsf{retrieveSensitiveData}(U.email + ".creds")$
(3) U	:	$creds{*} \leftarrow \mathsf{addToCredentials}(creds, [SK_{Proxy}, proxyCert])$
(4) U → S	:	$\mathsf{storeSensitiveData}(creds{*}, U.email + ".creds")$

Table 5. Creating and remote storage of a proxy.

receiveBeIDProxy-Protocol (cfr. table 6) defines how a user $U1$ requests a valid *proxyCert* from another user $U2$ and adds it to his contacts in the remote secure store. Storing the contact certificates online is required to enable remote access from hosts on other locations. $U1$ requests the proxy certificate of $U2$ (1). Then, $U2$ fetches his certificate from his secure credential store (2-3) and sends it to $U1$ (4). $U1$ verifies the validity of the proxy certificate, taking into account the modified certificate path generation (5). To comply with the Belgian legislation, $eID_2.cert_{sig}$ is deleted after validation (6). Finally, $U1$ adds the proxy certificate to his secure contact store indexed with the e-mail address (7-9).

receiveBeIDProxy():

(1) U1 → U2	:	requestCertificate()
(2) U2 ← S2	:	$creds$ ← retrieveSensitiveData($U2.email$ + ”$.creds$”)
(3) U2	:	$proxyCert$ ← lookup($creds, U2.email$)
(4) U1 ← U2	:	$(proxyCert, eID_2.cert_{sig}, cert_{Citizen_{CA}}, cert_{BelgiumRoot_{CA}})$
(5) U1	:	if (!isValidCert($proxyCert$, [$eID_2.cert_{sig}, \ldots$])) abort
(6) U1	:	delete($eID_2.cert_{sig}$)
(7) U1 ← S1	:	$contacts$ ← retrieveSensitiveData($U1.email$ + ”$.contacts$”)
(8) U1	:	$contacts$* ← addToContacts($contacts$, [$proxyCert, certChain/eID_2.cert_sig$])
(9) U1 → S1	:	storeSensitiveData($contacts$*, $U1.email$ + ”$.contacts$”)

Table 6. Retrieve the proxy certificate of a contact.

sendEncryptedEmail-Protocol. Table 7 shows the protocol to send a confidential message between two users. First, $U1$ fetches the proxy certificate $proxyCert_{U2}$, which he received earlier, from his contact store (1-2). $U1$ verifies the validity of the $proxyCert_{U2}$ (3). To improve the performance, a symmetric encryption scheme is used to encrypt the message. Therefore, a new random symmetric key *sK* is created (4-5). Next, the symmetric key *sK* itself is encrypted with the public key $proxyCert_{U2}.PK$ in the proxy certificate of the contact $U2$ (6). Both the encrypted message and encrypted symmetric key are mailed to $U2$ (7). To read the confidential message, $U2$ fetches the corresponding secret key SK_{U2} (8-9)) to decrypt the symmetric key *sK** (10). Finally, the message is decrypted with *sK** (11).

sendSecureEmail(message):

(1) U1 ← S	:	$contacts_{U1}$ ← retrieveSensitiveData($U1.email$ + ”$.contacts$”)
(2) U1	:	$(proxyCert_{U2}, certChain)$ ← lookup($contacts_{U1}, U2.email$)
(3) U1	:	if (!isValidProxy($proxyCert_{U2}, certChain$))abort
(4) U1	:	sK ← createSymmetricKey()
(5) U1	:	E_{data} ← encrypt(msg, sK)
(6) U1	:	E_{sK} ← encrypt($sK, proxyCert_{U2}.PK$)
(7) U1 → U2	:	sendEmail($E_{data}, E_{sK}, U2.email$)
(8) U2 ← S	:	$creds$ ← retrieveSensitiveData($U2.email$ + ”$.creds$”)
(9) U2	:	SK_{U2} ← lookupKey($U2.email, creds_{U2}$)
(10) U2	:	$symKey$* ← decrypt(E_{sK}, SK_{U2})
(11) U2	:	$message$* ← decrypt(E_{data},$symKey$*)

Table 7. Sending and receiving an encrypted message

7.3 Evaluation

This paragraph first evaluates the initial requirements. Next, our solution is compared to other approaches for securing e-mail services.

- R_1: Only the setup and request of a proxy certificate may imply some additional user interaction. The other protocols can be processed transparently. Of course when accessing the remote secure store or creating new proxy certificates, two PINs of the eID card are required: for authentication and signing (key generation). This may seem awkward. However, a possible solution may be to detach the fetching and storing of secure data (i.e. retrieveSensitiveData resp. storeSensitiveData) from the protocols above and only perform them at the initialization and closing of the e-mail client: the user will have to enter his authentication PIN once and his signing PIN twice (of which one can be avoided if the application caches the signature temporarily).

- R_2: Confidential e-mails can be sent and received from any host with a card reader.
- R_3: A separate server maintains CRLs of revoked proxy certificates. Moreover, a sender can verify that the eID certificate of the owner of a proxy certificate is (not) revoked.
- R_4: Access to contact information is only possible using the Belgian eID. Adversaries do not have access to the proxy certificates of the contacts.
- R_5: All information stored on the remote secure storage server is encrypted and if the server is trustworthy (i.e. the server does not store the encrypted tag nor the NRN), an adversary cannot link any data or even discover the presence of data of a particular individual. Moreover, only the owner can get access to his encrypted data.

PGP [13] has a different trust model compared to our approach. PGP was initially based on chains of trust, while our solution is based on valid certificate chains. The trust level of public keys in PGP depends on the number of signatures on these keys by other users. Verifying the validity of those keys is not trivial and less reliable. Users have to interpret trust levels themselves. In the more recent OpenPGP specification [14], trust signatures can be used to support the creation of certificate authorities.

S/MIME [15] also offers a solution to send confidential messages based on certificate chains. S/MIME typically stores keys and certificates on a user workstation which implies that e-mail cannot be sent or retrieved on different hosts. Some attempts have been made to store keys and certificates for e-mail services on a remote server. Those solutions mainly use password based encryption to support ubiquitous access. Our eID based solution provides stronger security while maximizing the availability.

Moreover, the creation of a BeID proxy certificate does not require a complex registration procedure. The individual can even create a proxy certificate off-line. Some certificate authorities offer free e-mail certificates for exclusive S/MIME usage on their web site. Users must sign up for an account. However, this does not automatically allow usage of one's name in the certificate. For that, one has to prove ones identity in person to at least two Thawte notaries that are part of their Web of Trust.

PGP and S/MIME also support digital signatures and mechanisms to ensure the integrity of e-mails. Since the eID card can directly be used to sign e-mail messages, we have omitted the description of this functionality of our e-mail client.

8 Conclusion

This paper presents two reusable extensions to the Belgian eID technology, namely a *ubiquitously accessible remote secure storage service* and *a mechanism to issue proxy certificates*. The former allows Belgian citizens to manage sensitive personal data such as contact information, passwords, keys, tickets, etc. The latter can be used to self-certify asymmetric encryption keys. The validation of the proxy certificates slightly deviates from the standard, because of the current design of the eID certificates and restrictions imposed by the Belgian legislation. However, this problem can easily be solved by redesigning the eID certificates. Hence, this paper also offers some guidelines for countries that consider introducing an eID card in the future. As a proof-of-concept, both extensions have been incorporated in a secure e-mail client. The approach is more secure than PGP and avoids some key management problems in S/MIME.

Acknowledgements

This research is partially funded by the Interuniversity Attraction Poles Programme Belgian State, Belgian Science Policy and the Research Fund K.U.Leuven, the IWT-SBO project ADAPID and the IWT-Tetra project e-IDea.

References

1. De Cock, D. , Wolf, C., Preneel, B. , The Belgian Electronic Identity Card (Overview), LNI, vol. P-77, pp. 298–301. Bonner Köllen Verlag (2006)
2. Verhaeghe, P., Lapon, J., De Decker, B., Naessens, V., Verslype, K., Security and privacy improvements for the belgian eid technology, In: 24th IFIP International Information Security Conference (SEC), Springer (2009)
3. Dumortier, J., eID en de paradoks van het rijksregisternummer, (2005)
4. Stern, M., Belgian Electronic Identity Card content, 2.2 ed., CSC, Zetes (2003)
5. Andries, P., eID Middleware Architecture Document, 1.0 ed., Zetes (2003)
6. Rommelaere, J., Belgian Electronic Identity Card Middleware Programmers Guide, 1.40 ed., Zetes (2003)
7. Ramlot, G., eID Hierarchy and Certificate Profiles, 3.1 ed., Zetes – Certipost (2006)
8. Belgian certificate revocation list, http://status.eid.belgium.be
9. Tuecke, S., Welch, V., Engert, D., Pearlman, L., Thompson, M., Rfc 3820 - Internet x.509 public key infrastructure (pki) proxy certificate profile, (2004)
10. Housley, R., Polk, W., Ford, W., Solo, D., Rfc 3280 - Internet x.509 public key infrastructure certificate (pki) and certificate revocation list (crl) profile, (2002)
11. Shamir, A., How to share a secret, vol. 22, no. 11, pp. 612–613. Commun. ACM (1979)
12. Bellare, M., Goldwasser, S., Verifiable partial key escrow, In: CCS '97: Proceedings of the 4th ACM conference on Computer and communications security, pp. 78–91. ACM, (1997)
13. Garfinkel, S., PGP: Pretty Good Privacy, O'Reilly Media (1994)
14. Callas, J., Donnerhacke, L., Finney, H., Shaw, D., Thayer, R., Rfc 4880 (Proposed Standard) – OpenPGP Message Format, (2007)
15. Ramsdell, B., Rfc 2633 – s/mime version 3 message specification, (1999)