THESIS/DIPLOMARBEIT

PRIVACY ENHANCING TECHNOLOGY IN HEALTH CARE BY USE OF SMART CARDS

by

Silke Christina Lieber

FACHHOCHSCHULE ULM,
FACHBEREICH INFORMATIK, MEDIZINISCHE DOKUMENTATION UND INFORMATIK

with cooperation of the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

SOMMERSEMESTER 2001

June 30, 2001

Author_____
Silke Christina Lieber
June 30, 2001

Certified
by_____
Prof. Dr. Peter Szolovits
Professor of Electrical Engineering and Computer Science

Accepted
by_____
Prof.Dr. Gerhard Kongehl
Professor of Data-Privacy, IT-Security and Technological Assessment

Table of content

List of figures

1. Abstract

This thesis explores the use of Smart Card technology to help support secure authentication. This is done in the context of a Web-based health study system, SHARE, whose security relies critically on reliable authentication of several adminstrators. The tools developed support the use of SDSI certificates as well as more commonly-used certificates.

2. Introduction

This is the modern age of information theft on the Internet. Stories of credit card numbers being stolen from databases come to our ear. To prevent this from happening, Smart Cards can be a tool of safeguarding such sensitive information. It is even more important to use the same safeguards to protect the privacy of the medical records of people who, for example, may have psychiatric care histories, or who may be under lifelong treatment for HIV.

The World Wide Web provides great possibilities to enhance the accessibility and therefore the quality of health care studies. The downside is that data, which are accessible by the WWW, are vulnerable to tampering. If this happens, the privacy of patients is violated.

To ensure that sensitive information remains on a need-to-know-basis only, and that it stays out of the reach of those who would use that information wrongly, we must have reliable security measures in place.

Information technology businesses, the government, and healthcare organizations continue to move towards storing and releasing information via networks, intranets, extranets, and the Internet. These new techniques offer great possibilities, but are also risky. This change requires improved security of network transactions, to assume that data are not intercepted or altered in transit and that communications takes place only with intended individuals and institutions. Smart Cards are one recently developed technology to help to meet these requirements, and their use is investigated in this thesis.

Using Smart Cards, this information can be made readily available to those who need it, while at the same time they protect the privacy of individuals and keep their informational assets safe from being hacked into and other unwanted intrusions. Therefore, Smart Cards offer:

- Secure logon and authentication of users to PCs and networks
- Storage of digital certificates, credentials and passwords
- Encryption and decryption of sensitive data

Smart Cards are an evolution of the credit card. They incorporate features and functions of a computer and their built-in cryptographic functions present a high level of security. A Smart Card can be used for the same purpose as a credit card but because of its enhanced capabilities, it can also be used for owner authentication and secure transactions.

Smart Cards could revolutionize the way business is conducted over the Internet and might very well lead to a truly cashless society. At the same time, it would make such a system less liable to fraud and could make all communication via a public network such as the Internet secure.

The goal is to prevent any medical information from becoming open to the public, or to anyone who would use private medical information as a means of discrimination, blackmail, or just to have fun knowing that they know. One possible solution to privacy enhancing technology in health care is the implementation of the micro-processing Smart Card.

3. What is a Smart Card?

A Smart Card looks like a credit card but that is about the only thing that they have in common. The most advanced Smart Card, a microprocessor Smart Card, has more similarities to a PC than to a plastic card containing a magnetic stripe. A microprocessor card is able to store data and execute commands; it contains a single-chip microprocessor with a size of 25 mm^2. Smart Cards provide data portability and security.

In microprocessor cards, the information on the card can be protected by a PIN code and can also be read-write protected. The card cannot be cloned. It is possible to perform encryption on the card. The private key for the encryption never leaves the card and therefore it is difficult to tamper with it from outside the card. Each card has its own serial number and is therefore unique. By use of a Smart Card reader, the card can communicate with the computer. It is possible to update information on the card with the reader.

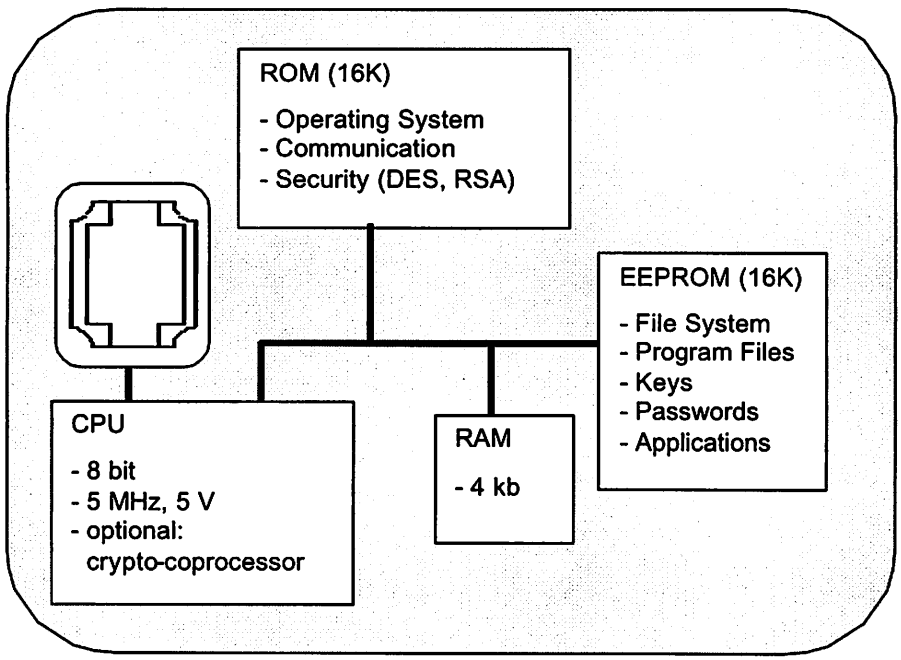

Figure 1: example of a Smart Card chip and its components

Table 1 describes different Smart Card technologies that provide various levels of capability. For the current work, the microprocessor card is the minimum needed. The drawback of a memory card is that it is able to store only limited information, whereas micro-processing cards have up to 16 K of EEPROM. Furthermore, micro-processing cards can have their information changed freely (EEPROM), but ROM Cards are limited because the information is stored in them during production.

Simple memory chip cards	Only electronic memory EPROM, EEPROM or RAM -> direct access
Intelligent chip cards	Memory EPROM or EEPROM Access only through a connected security logic, use: telephone cards
Processor chip cards	Smart Cards: Have a whole PC in there - Microprocessor, ROM (EEPROM, EPROM), and RAM. Free security functions possible. Implementation of Challenge-Response-Protocols and cryptographic things on the card possible. Multifunction cards
Superchip cards	Processor chip cards with built in display, keyboards and batteries. Functions without any card terminals possible. Price and robustness are problematic. Multiple functions
Chip cards with contacts	Through gold contact areas -> contact to Smart Card reader
Contact-less chip cards	Are passed near an antenna to carry out a transaction

Table 1: the different types of Smart Cards

3.1. What makes a Smart Card smart?

For our purposes, a Smart Card is "smart" because it has a microprocessor embedded. It can ensure that the user of the card really is the person to whom the card belongs as long as he or she does not divulge the PIN. A credit card does not guarantee that the person who uses it is the person to whom the card has been issued, as information used to authenticate the user (e.g. maiden name, birthday, address) can easily be obtained through other channels. The Smart Card, with its encrypted information, guarantees that authentication.

3.2. Security Issues of Smart Cards

Smart Cards are a barrier against fraud. Data contained in a Smart Card can only be accessed by encryption and decryption. The most common alternatives to Smart Card based themes for authentication and secure access are based on passwords or pass-phrases. Passwords can be stolen or forwarded freely. Therefore, many companies look for schemes that are based on Smart Cards or biometric methods.

If a Smart Card is stolen together with its PIN, an illegal authentication is possible. This is in contrast to biometric methods where it is hard to believe that the "hardware" can be stolen.

Usually the Smart Card requires the owner to authenticate himself by using a password. This is sometimes referred to as "two-factor" authentication, combining "what you have" (the physical Smart Card) with "what you know" (the password needed to use the Smart Card). Two-factor authentication can provide a higher level of security than authentication based on a password or on a hardware token alone.

A password is not always sent on a secure telnet; this is like sending a password out in the clear. Using SSL and TLS to communicate, a secure session can be established with the Smart Card because information can be encrypted before sending and decrypted upon receiving.

3.3. Smart Card Software

The Smart Card applications can be divided into two parts: on- and off card applications. The off-card application is the part that resides on the computer connected to the Smart Card. The on-card part is stored on the chip of the Smart Card. This code is executed by the Smart Card operating system. So-called Java Cards use Java as the language for on-card application. This is a good basis for multi-application cards. Java is the platform but not the operating system of the card. The Java language subset for the card has some limitations in comparison with the Java language for PCs and networks. Some differences are:

- Dynamic class loading is not supported
- The JVM does not support multiple threads
- Objects cannot be cloned

Nevertheless, a common Java compiler can be used to compile Java Card sources to byte code.

3.4. GemSAFE from Gemplus

For this thesis, we have chosen to work with Smart Card technology as implemented by GemSAFE, corporation of Gemplus.

GemSAFE was developed to provide secure access to networks for organizations to use Smart Cards in their public key infrastructures. GemSAFE provides a personal network safeguard by using digital certificates stored on Smart Cards for accessing corporate intranets, extranets, websites and electronic mail systems. Designed for plug-and-play installation and set up, GemSAFE combines a Gemplus Smart Card, Smart Card reader and software for integrating with Microsoft and Netscape software suites.

GemSAFE works with Microsoft Internet Explorer 4.x, Outlook 98 and Outlook Express via Microsoft's Crypto API, as well as Netscape Navigator 4.x and Messenger via RSA PKCS#11. When these applications are being used, GemSAFE provides SSLv3 client authentication for requesting web servers and secure S/MIME email exchange via the user's digital certificate and private key stored on a Smart Card. The user's private key never leaves the Smart Card and is inaccessible from outside the card. All cryptographic functions requiring the private key are handled on the card by the onboard microprocessor.

GemSAFE also works out of the box with Microsoft Windows 2000 logon, and GemSAFE is fully PC/SC compliant. No extra drivers are needed to get up and running. Furthermore, GemSAFE Smart Cards can be used in any industry-standard PC/SC-compliant Smart Card reader.

GemSAFE is designed to work with all certifying authorities, whether an organization chooses to take on this task themselves or outsource it to an external trusted third party such as VeriSign, Baltimore or GTE Cybertrust. Organizations can also choose to personalize GemSAFE Smart Cards themselves using the Gemplus card management system, a complete solution for issuing and managing cards for employee IDs, etc. This makes it easy for security officers to generate, manage and recover Smart Card-based digital certificates and keys in the enterprise.

3.4.1. Properties

The main properties of the GemSAFE card are:

1) Store a private key and digital certificate
2) Tamper resistance
3) Electronic passport that can be used on any GemSAFE system
4) X.509v3 certificate storage
5) Onboard Key Generation
   - Every time a new certificate on the card is enrolled, a new key pair is generated on the card. Therefore it is not limited to use the same key pair for every certificate that somebody enrolls.
   - One significant advantage of onboard key generation is the ability to monitor and control the life span of RSA key pairs.
6) Increased Certificate Storage
   - 4 key pairs and 4 digital certificates can be stored on the GemSAFE Card, i.e. use one certificate and key encryption 1024 bit and one certificate and key encryption 512 bit.

The GemSAFE system has four main components:

1) Manager
   - It allows in house card management (card issuance, user support, integration of certificates for authentication (CAs) and card deactivation).
   - It makes it possible to have multiple CAs (e.g. Microsoft certificate for internal network and Baltimore certificate to support partners and supplier transactions).
2) Workstation
   - It provides login authentication, encryption (email & files).
   - Private key never leaves the card.
   - Smart Card is hardware-based security.
   - PIN code protects key use.
   - GemSAFE Enterprise Workstation is portable.
   - Secure X.509v3 certificate storage.

3) Entrust
   - Enables integration of Smart Cards into existing and new Entrust PKI based applications.
   - Strengthens the security of Entrust PKI Smart Cards and performs all sensitive operations in the Smart Card.
4) Software Development Kit
   - Hardware and software package for Smart Cards enables developing and customization of applications.

The advantages of the GemSAFE system are that the:

- Private key never leaves the card.
- Smart Card is hardware-based security.
- PIN code protects the key.

4. Cryptography

4.1. Introduction

Encryption is the process of disguising information as "ciphertext", to make data unintelligible to an unauthorized person. Manual encryption has been used since Roman times, but nowadays the term has become associated with the disguising of information via electronic computers. Encryption is a process basic to cryptography.

4.2. Security aspects of web technology

The World Wide Web has become the de facto interface for consumer oriented electronic commerce. So far, the interaction between consumers and merchants is more or less limited to providing information about products and credit card based payments for mail orders. This is largely due to the lack of security currently available for commercial transactions. Furthermore, the future of the WWW is not limited to electronic commerce. Using the huge possibilities of the WWW for healthcare studies makes data privacy even more important. Currently the only security mechanism present in most browsers is the Secure Socket Layer (SSL), which is limited to authentication and encryption of the HTTP session. It does not aim to secure transactions.

Computers encrypt data by applying a set of procedures or instructions for performing a specified task to a block of data. A personal encryption key or name, known only to the transmitter of the message and its intended receiver, is being used to control the algorithm's encryption of the data, thus yielding unique ciphertext that can be decrypted only by using the key.

Since the late 1970s, two types of encryption have emerged. Conventional symmetric encryption requires the same key for both encryption and decryption. Asymmetric encryption, or public-key cryptography, requires a pair of keys, one for encryption and one for decryption. It allows disguised data to be transferred between allied parties at different locations without also having to transfer the (not encrypted) key. A common asymmetric encryption standard is the RSA (Rivest-Shamir-Adleman) algorithm. Briefly, the asymmetric encryption works like this: the so-called public key is a "one-way" function, which means that it is not reversible. A text, which has been encrypted with this algorithm, is computationally so difficult to decrypt as to be practically impossible. Therefore, the key can be made public, comparable to a phone book. Should Alice want to send a message to Bob, she looks up his public key, encrypts the message with Bob's public key and sends it to him. Even though Bob's public key is known, nobody can decrypt the message – except Bob. Bob has a "private key", which is an algorithm that is associated with his public key. With his private key, which he has to keep secret, he can decrypt the message again.

4.3. Cryptography in Networks

Intranets, extranets, and the Internet: they are all based on the same technology. All of them are computer networks that use common protocols to communicate with each other and they are all susceptible to the same security risks.

Even simple retail transactions pose threat, and personal email communications entail risk if intercepted. More sensitive areas of personal transactions and most business transactions pose a much greater risk, especially when conducted via an insecure channel like public networks. These require far greater assurances of authenticity, privacy, integrity and acceptance. Today, the most popular way to establish identity is to use a password. Still, they cannot be relied upon for real proof of identity for many reasons.

Passwords are often sent through networks without any encryption, making them highly susceptible to interception and misuse. Users generally need to remember many passwords, and often end up using the same password for many systems, often using easy-to-remember (and easy to crack) passwords, or writing down the passwords where they are easily accessible. On the back end, all passwords must be stored in a single file, and although encrypted, there are numerous cases of these files being stolen and unlocked. Further, these password files are expensive to support and maintain, with much of the expense resulting from users forgetting their passwords. The bottom line is: a password offers little proof of who is actually using it.

Cryptography instead provides a set of techniques for encoding data and messages in a way that the data and messages can be stored and transmitted securely.

- Cryptography can be used to achieve secure communications, even when the transmission media (for example, the Internet) is untrustworthy. It can be used to encrypt sensitive files, so that an intruder cannot understand them.
- Cryptography can be used to ensure data integrity as well as maintain secrecy.
- Using cryptography, it becomes possible to verify the origin of data and messages. This is done using digital signatures.
- When using cryptographic methods, the only part that must remain secret is the cryptographic keys. The algorithms, the key sizes, and file formats can be made public without compromising security.

Cryptography allows a series of operations or actions on data. The two fundamental operations are encryption (with decryption as its inverse) and signing (with verification of signature as its matching operation). Encryption is comparable to enclosing data in an opaque envelope; decryption is comparable to removing it from the envelope. Signature is similar to physically signing a document, and initialing each section to show that no part of the document has been changed. Verification of signature is roughly equivalent to

matching the signature to a "signature on file" card, and verifying that no part of the document has been changed. Certificates are signed documents, which match public keys to other information.

4.4. Digital Signatures

The aim of a digital signature is similar to hand-written signatures:

- Verify the origin of documents
- Proof of identity
- Endorsement of a document
- Possibility of verification of the signature

Digital signatures are based on an asymmetrical key pair. The private key of the signing person cannot be accessed by anybody else. The associated public key is needed to verify the truth of the private key that was used to sign the digital document. The public key is distributed in a certificate that contains an expiration date.

4.5. Certificates

Digital certificates were created to leave behind the general anonymity afforded by unsecured networks like the Internet by providing a reliable and trustworthy proof of identity similar to passports and driver's licenses. Used in conjunction with modern web browsers, email software and other applications, digital certificates (and the public key technology they are based on) offer the potential for ensuring secure electronic commerce and transactions through these networks.

4.5.1. SDSI certificate

Ronald L. Rivest and Butler Lampson of MIT's Laboratory for Computer Science designed a Simple Distributed Security Infrastructure (SDSI) as a new, simplified public key infrastructure. In SDSI, principals are equated with their certificates. SDSI also relies strongly on the use of local names to identify individuals, groups and resources. Certificates can assign such local names, attest to membership of an individual in a group, and legitimize the use of a resource by an individual or group. SDSI certificates have a specific publicly defined format. They are used for authentication and authorization in SHARE.

For SDSI, the most important feature of public-key cryptography is the ability to sign things. To create a signature on a message, the private key scrambles the message in its own unique way - much like a handwritten signature is always identifiable and unique, but is still a little different on each document that has been signed.

Once a message is signed, anyone who knows the public key of the author can verify that signature by unscrambling it and comparing that to the message. Since the public key will only correctly unscramble signatures that were provided by its matched private key, a correct signature (and therefore, the message) can be believed to be authentic. The programs PGP and SSH utilize public-key cryptography.

4.5.2. The Problem with Digital Certificates

With a digital certificate, anyone with access to the private key is assumed to have rightful ownership of the certificate. Thus, while digital certificates can associate an identity with a public key, the digital certificate alone cannot confirm that the individual presenting the certificate as proof of identity is actually the rightful owner.

Consequently, protecting the private key is the single most important aspect of using digital certificates because if others know the private key, it is possible for them to assume that identity and engage in fraudulent use of the certificate. Most digital certificates today, and more importantly their associated private keys, are simply encrypted with a password and stored on the owner's PC hard disk drive where it may be vulnerable to attack either directly or through the network. The private key is vulnerable to many of the same password-related problems mentioned earlier, and several programs are available to either divert PC files or attack password mechanisms.

As a result, although digital certificates can provide digital authentication, they are not fully secure without strong user authentication. Without strong user authentication, a digital certificate is about as much good as a passport without a photograph of its owner attached. A passport may attest to its owner's identity and be an official document issued by a government agency; but without a photograph, it is impossible for anyone to confirm whether or not the person presenting the passport is in fact the owner.

4.6. Smart Cards in Authentication

As mentioned above, the protection of the private key must be at least as strong as the security of the messages encrypted with the key. Single-factor authentication such as a password is simply not good enough.

Smart Cards offer good protection for private keys because they require not only a password (PIN) but also physical possession of the card as well to gain use of the private key. This kind of two-factor authentication offers significantly stronger security than passwords, and ensures that only its rightful, intended owner uses the digital certificate. The most common alternatives to Smart Card based schemes are based on biometric methods.

By using a Smart Card, the private key never leaves the card and is completely inaccessible from outside the card. All cryptographic functions requiring use of the private key for secure Internet browser and electronic mail transactions - digital signatures and decryption of the session keys - take place on the Smart Card by the onboard microprocessor, and only the results are passed back to the host PC. Because the private key does not leave the card, the corresponding public key has to be generated on the card, but will be sent out to the network.

The Smart Card itself is easy to use, portable, unique and cannot be cloned. Its use is PIN protected, and it becomes completely unusable after a specified number of failed access attempts. The user has fewer passwords to remember and IT departments have fewer problems with lost or stolen passwords.

Smart Cards offer this protection by locking the digital certificates in a highly secure, removable medium, and by making them inaccessible to anyone but their rightful owner. Without knowledge of this critical information, would-be hackers and thieves are unable to violate the rightful owner's identity and use it to gain access to confidential information or conduct transactions.

5. SHARE: Secure Health Information Sharing System

SHARE is a web based set of tools to support secure sharing of data for multi-institutional health studies. Such studies need efficient access to patient records, but they also need to protect the privacy of the patients who have agreed to participate in the studies. Especially because several institutions are normally involved, it is particularly difficult to guarantee the patient's privacy. The researchers need patient data from different data sources (e.g. hospitals and medical labs) without disclosing the identity of the patient. To enable the sharing and linking of data it is common practice to tag the data with the identity of the patient. Identifiers can be either name, address, phone number, birth date, or social security number. Unfortunately, the privacy of the patient cannot be guaranteed.

The simplest way to keep the privacy of the patient is to remove all patient specific information before her data are shared. However, the information might be useless because further information cannot be added anymore. It is also possible that the data of one patient are submitted several times if she cannot be identified anymore. Furthermore, it is questionable whether all information that can be used to track down the identity of the patient is sufficiently removed.

To solve this problem, SHARE was developed. It is based on the formerly developed system HIIDIT (Health Information Identification and De-identification Toolkit). HIIDIT permits the appropriate linkage of multiple records and also protects patient privacy. The medical records are being linked in a secure way. The name space is decentralized, based on simple distributed security infrastructure (SDSI). SHARE provides a practical implementation of some of the ideas developed in HIIDIT. In general there exist three kinds of sites that participate in the exchange of data:

- The *"generation site"* (a secure web site where an authorized study generator can design a new health study)
- The *"study site"* (a secure study institution which collects health data from source sites and performs the health study)
- The *"source site"* (a clinical institution that provides health data from its databases)

6. Implementation

6.1. Introduction

In order to restrict access to SHARE to properly authorized individuals, a system needs to be implemented to verify and authenticate the identity of individuals wanting to gain access to the system. One possible solution is the use of Smart Cards and SDSI certificates.

6.2. Ideal system design

The ideal design is that the implementation will work with any certificate, with any browser, and with any Smart Card. To enable the authentication a SDSI certificate has to be stored on the Smart Card. Three possible solutions exist to do this:

1) A servlet and an applet on the browser where:
   - An applet retrieves the certificate from the Smart Card and sends it to the servlet.
   - An applet answers the random challenge from the servlet.
   - A servlet verifies the certificate and confirms the challenge.
   - A servlet sets cookie on the clients browser.

   In figure 2 a scheme of this solution is shown.

2) A plug in for the browser, which can:
   - Receive a challenge from the server and send it to the card for decrypting.
   - Receive a certificate request from the server and retrieve it from the card.

3) Three stand alone applications
   - A server.

- An application to write the SDSI certificate to the Smart Card.
- An application to communicate with the server.

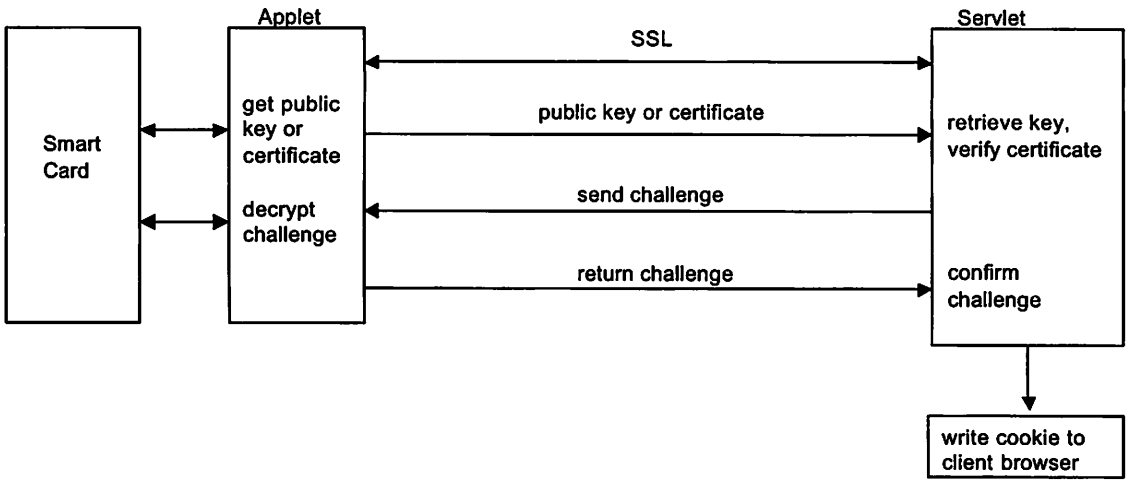

Figure 2: architecture of the applet/servlet solution

6.3. Why did I use GemSAFE

Two different Smart Cards that could be chosen are in existence: the Java Card and the GemSAFE Card. The choice of the card is not relevant from the application point of view; the application developer can rely on standardized API. The API of the Java Card is OCF. PC/SC is used by Microsoft for Windows environment.

Use of PC/SC is spreading rapidly, therefore this is promising. OCF however is stagnating even though it is more generic. It also defines Java services. However, only a few vendors offer packages of this kind. Furthermore, they are not inter-operable. Therefore, OCF was discarded for this project.

For the Smart Card itself, the application developer can rely on a standardized API. The card is responsible for the authentication process i.e. cryptographic processing. In this case, the choices are:

- CAPI/CSP - defined by Microsoft. Internet Explorer 5.0 uses this interface.
- PKCS#11 - defined by RSA Labs. Netscape uses this interface.

In order to use the Smart Card in conjunction with a browser, i.e. the Smart Card is accessed by an applet, the appropriate library would have to be used. Therefore one applet would be needed for Netscape users and one for Internet Explorer users. Because a stand-alone application was used, the more convenient library for Smart Card access was used. In this case PKCS#11 because actually documentation made this library usable to a certain extent, whereas a Java library for CAPI/CSP could not be found nor could any comprehensive documentation for it be found. Another aspect of CAPI/CSP is that the CSP libraries have to be signed by Microsoft. This restricts the use of those libraries to the US and Canada because Microsoft will not sign libraries that do not comply with US export regulation. This means that outside of the US or Canada CAPI/CSP offers very little cryptographic protection.

By using such interfaces, any kind of cryptographic tokens i.e. Smart Cards, USB tokens, PCMCIA cards etc. can be used to fulfill the authentication needs of a standardized protocol like SSL. The problem is that the SDSI certificate was used in this thesis and that this certificate has never before been applied on Smart Cards.

By using a GemSAFE card, it is possible to communicate with the card through PKCS#11 and CAPI/CSP interfaces. By this means, it cannot be detected whether the card has been implemented by a Java card or a proprietary card.

6.4. Solutions

Using the three stand-alone applications, the SDSI certificate was stored on the Smart Card. This enables the connection to the JSHARE card and the server. In doing so, it was also decided to separate the application into two parts. One to generate the public-private key pair on the card and generate a SDSI certificate from it, and to put that also on the card, and another one to register the certificate with the server and obtain a session key from the server.
The servlet/applet solution did not work because of security policy problems in Java. In general, it should be possible to get rid of these problems but this could not be done cause of the limited time of this thesis. The development of a plug in was also too time consuming.

6.5. Actual system design

This system uses a challenge-response protocol that is based on the client's certificate. After a client has registered its certificate with the server, the server uses the public key to encrypt a challenge for the client. If the client correctly decrypts the challenge, it has proven to be in possession of the private key that works in conjunction with the public key embedded in the certificate. Once the server received the decrypted challenge and verified it to be correct, it can grant the client access to the protected data. If the client failed to decrypt the challenge correctly, it is denied that access.

6.5.1. Communication between Card Reader and Application

The card reader contains a PKCS#11 library, which communicates with the driver. If for example the application calls the Decrypt Method, the Decrypt Method communicates with the driver and exchanges the relevant information. The driver then exchanges information

with the card reader and the Smart Card. Finally the library returns the decrypted message to the application.

Standard Crypto Libraries do not communicate with Smart Cards, but do en- and decryption, key management, etc.

PKCS#11 is needed because it talks to Smart Cards and the Smart Cards take care of all encrypting, decrypting and key management (excluding SDSI certificate verification). A certificate on the hard drive has to be moved to the Smart Card. This is also accomplished with PKCS#11.

6.5.2. Writing the SDSI Certificate to the Smart Card

This following window is launched when the user wants to write the SDSI certificate to the Smart Card:

Figure 3: screen shot of the window to write the SDSI certificate

Before issuing the Smart Card to the user, the administrator has to write a SDSI certificate to the Smart Card. The administrator does this so the card contains the proper certificate, is tested and ready for use. The steps involved therein are illustrated in the figure below.

The SDSI application is a Java application with a toolbar, a text field for the PIN, a text field for the name of the certificate holder, a button to load the current SDSI certificate to the Smart Card and a text field to show the filename of the currently selected SDSI certificate.

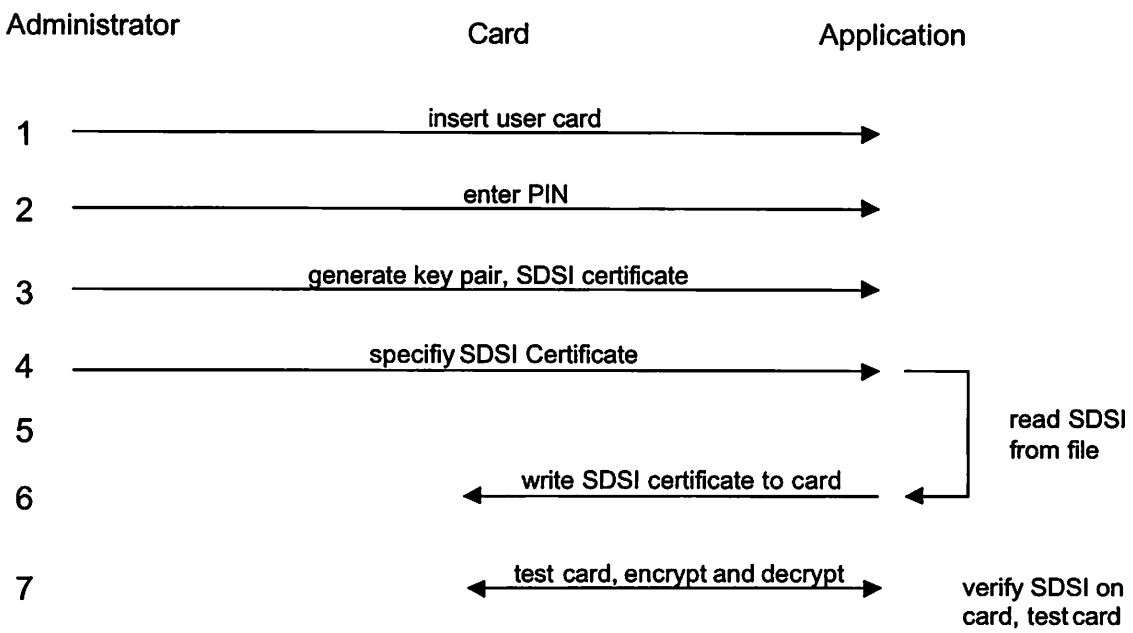

Figure 4: time flow diagram for application to write SDSI to a Smart Card

Using the Open Document and Create Document buttons, a new SDSI certificate can be created or selected to write to the Smart Card. The filename of the currently selected certificate to be written to the Smart Card is shown in the Current Certificate text field. To actually write the certificate to the Smart Card, the Load SDSI Certificate button has to be pressed. The Delete First Item on Card button removes the first item stored on the card, the List Items on Card button displays a list of item on the card in the console. In order to delete the second item in the list, the first item will also have to be deleted. If the last item has to be deleted, all items on the card have to be deleted.

Initial registration of certificate:

A) First time user uses the card, the authentication application has to do the following:

1. Get the certificate from the card, sign it and send it to the server with the login.
2. Server verifies that the certificate is authentic and stores it.
3. The server determines the authenticity with the signature and any other parameters embedded in the certificate that tells the server that the certificate is authorized to access the confidential data. Such access authority could be encoded in the certificate in various ways. Simple possession of a SDSI certificate that was signed by a certain principal might be sufficient, or the certificate could have an extra s-expression that implies access authority to SHARE. Other s-expressions could imply access to other projects in that case.

The challenge response architecture:

B) Subsequent use of the card:

1. Application tells the server it wants access to confidential data by sending only the Login.
2. Server sends a challenge, which works like this
   2.1. Server generates random string 'A'
   2.2. Server encrypts random string 'A' with public key from the certificate, resulting in encrypted string 'B'
   2.3 Server sends encrypted string 'B' to the application
3. Application upon reception of challenge will decrypt string 'B', resulting in string 'C'
4. Application signs string 'C' and sends the signed string 'C' to the server
5. Server compares 'A' and 'C', if 'A' equals 'C' the server will send a session key which is encrypted with the public key to the client allowing it to access the confidential data.

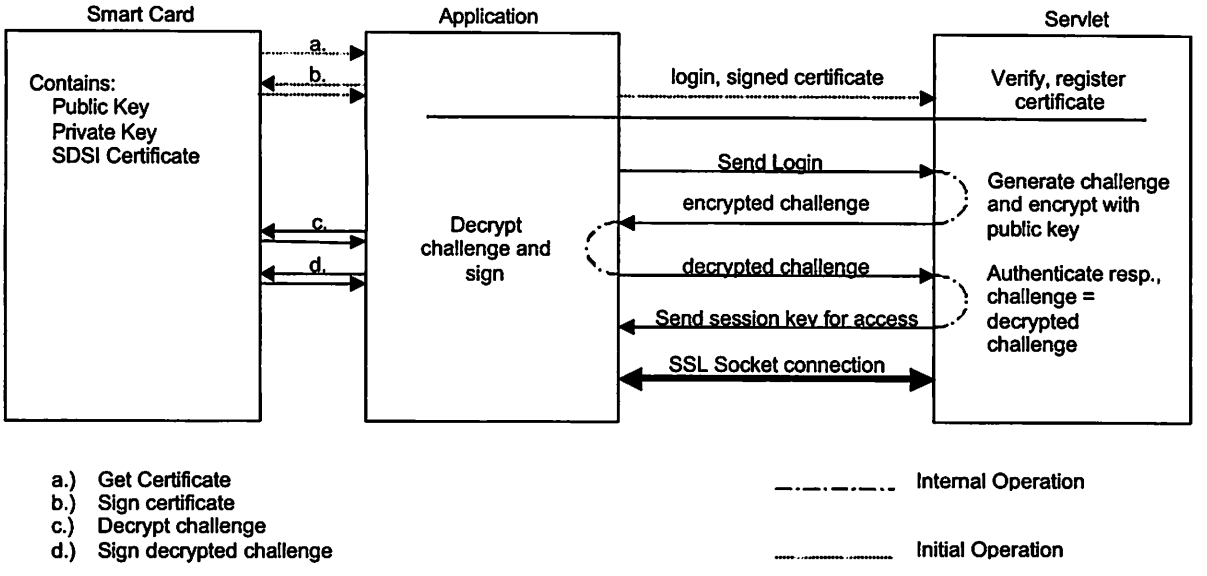

Figure 5: authentication protocol

The SDSI certificate can be read from the Smart Card and digital signing and decryption based on the private key from the Smart Card can be performed. Access is granted after the Smart Card is successfully verified. To write a certificate on the card the create object method from the PKCS#11Session class was used.

The server has to extract the public key from the certificate and use that to encrypt the challenge.

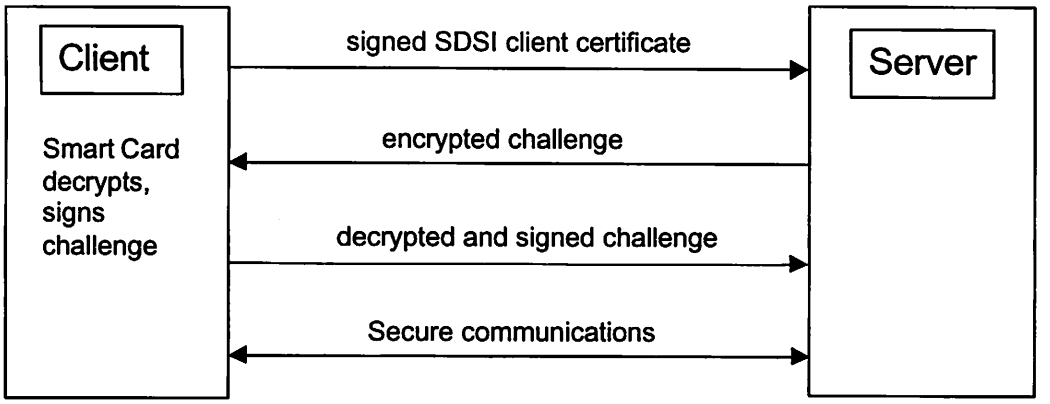

Figure 6: challenge response protocol

In order to communicate all data safely, both the server and the application are set up to use SSL sockets and set up their own secure connection over the network. SSL requires at a minimum that the server has a X.509 certificate to authenticate itself. The client does not need such a certificate, as it uses SDSI certificates to authenticate itself to the server.

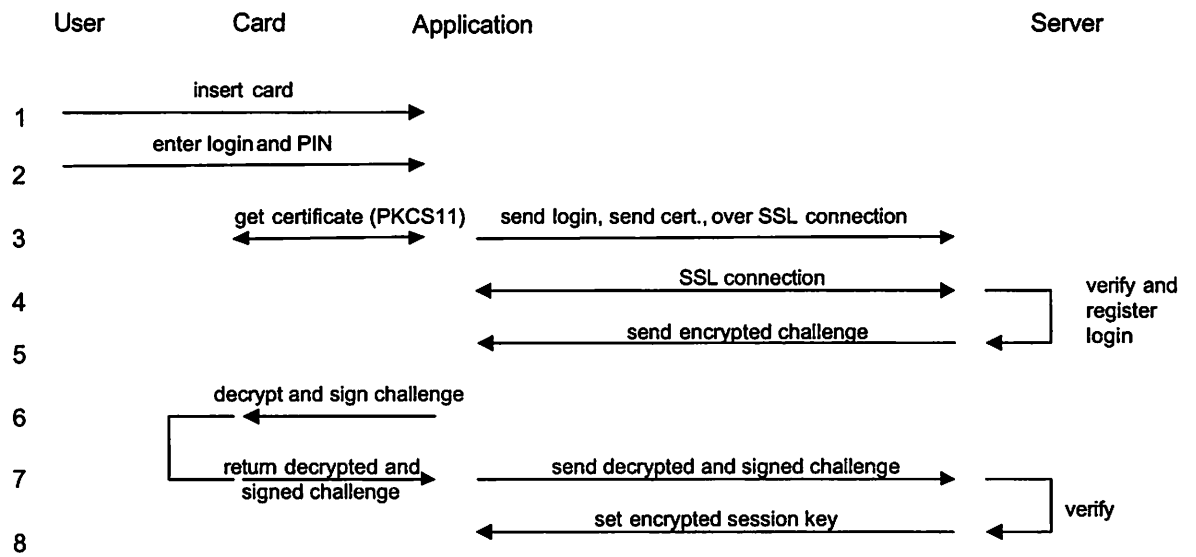

Figure 7: time flow diagram for authentication application

6.5.3. How do I do it?

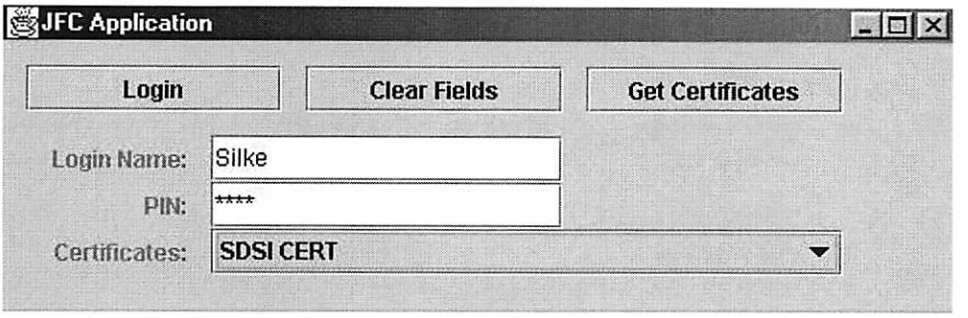

Figure 8: screen shot of the JSHARECard application

The application provides two input fields for a Login name and a PIN. It also has a combo box to list all the certificates available on the card. Available buttons are Login, Clear Fields and Get Certificates. The PIN field is being used to enable access to the Smart Card because there are no Smart Card functions available unless the correct PIN has been entered. The application asks for a PIN number, and with it will be able to retrieve the certificate from the card, sign the certificate or the response or will be able to decrypt the challenge from the server (if the PIN is correct).

The Login name is a unique identifier for the user of a particular Smart Card, using a particular certificate from that card.

When Get Certificates is pressed the application will query the Smart Card and put all the labels of the certificates on the Smart Card into the combo box.

Clear fields will clear the Login field, the PIN field and the certificate list.

When pressing the Login button, the application will do the following, depending on whether the certificate list has been filled or not. If a certificate has been selected in the certificate list, it will retrieve the certificate, sign it and send it with the Login to the server. This will cause the server to store the certificate and bind it to the Login, if the server successfully authenticates the certificate. Using the public key of the certificate to verify the signature makes this possible. The server should also check some other information embedded in the certificate to check if the owner of the certificate has permission to

access the confidential data in the first place. If this Login has already been linked to a certificate or the authentication of the certificate failed the server will simply ignore the certificate.
In any case, the application will then send the Login only. This will trigger the challenge response protocol. The server will check if the Login has been linked to a certificate.
If no certificate has been found, the server will disconnect. If a certificate for the Login is found, the server will generate a random challenge and encrypt it using the Public key found in the certificate. It will send that back to the application. The application then will access the Smart Card and decrypt the challenge.
Using the Smart Card, it will then sign the solution and send it back to the server. The server will then check the digital signature again and compare the solution with the original challenge. If the solution and original challenge are the same and the message is determined to be genuine by means of the signature, the server will send a session key to the client giving it permission to access the confidential data.
This session key is only temporary and should become invalid if the client disconnects from the server.

The first stage is to obtain the certificate of a principal (user) and put it on the Smart Card. The user starts the application on her computer. The certificate will then be sent to the server (communicating with a server), which checks if the certificate is valid, and if so, stores the certificate. This is being done over an SSL connection, which is set up between the server and the application.

Secondly, when the user attempts to use the server from another (untrusted) client, the user starts the application, identifying herself through a login. The server waits in the background, and the client has the application running. The server generates a random challenge and sends it to the application. The application asks for a PIN, and passes the challenge to the Smart Card, which calculates a response. The server verifies the response, and if it is correct, it grants access to the server.

6.5.4. Concerns

The research conducted raised the following concerns:

- The original intention was to have an applet for authentication. This did not work, as the applet would have to load a dll from the client hard drive to talk to the Smart Card. Due to Java applet security restrictions, this is not practicable. The dll would somehow have to be loaded from the server. Whether this can be accomplished using the current implementation of PKCS#11 is not known. In a sense, PKCS#11 violates the Java paradigm of write once, run anywhere because such a dll should not be loaded by Java at runtime but instead by the operating system at startup. When the authentication is being done through a Java standalone application, a problem connecting to the servlet turns up. Therefore, the servlet has been replaced with a server.
- The problem of loading the dll for PKCS#11 from the client when running an applet is not insurmountable though. By changing the security parameters of the browser with respect to Java, the applet might be able to access the local file system and load the dll and therefore get access to the Smart Card. How this problem can be successfully addressed is unfortunately not exactly known.
- PKCS does not seem to work with the Internet Explorer. We would have to make the code be aware of which browser the client is using and have the code call PKCS methods or CAPI methods accordingly. At present, a suitable library for CAPI was not found.
- Installation of necessary libraries and executables so that the clients can execute the Application. This has to be done before the application can be used on client machines.
- The server needs to know which valid certificates are acceptable and which ones are not.
- The Smart Card is limited in the amount of data it can store. Even though the Smart Card has 8 or 16 kilobytes of memory, only one kilobyte can be used to store the SDSI certificate. The rest is reserved for private public key pairs, symmetric keys and X.509 certificates. Because of the Card not recognizing the SDSI certificate as anything but miscellaneous data, we are limited to the maximum data storage capacity of 1 kilobyte on the card.

Using the servlet and the applet, the applet had to load a file from the local machine to access the Smart Card reader. This was fine in the debugger, but once the applet runs in a web page, loading this file is not possible, as applets are not allowed access to the local file system, they are only allowed access to the file system on the server from which they originate. This is one of the security features of Java. It might be feasible to load the required file from the server, but the easier solution was to rewrite the applet as a stand-alone application. The servlet was also abandoned in favor of a stand-alone server because of problems encountered while trying to communicate with the servlet and with setting up a web server to use SSLSockets.

6.5.5. More details on the problem containing the Applet

The Java console generates the following output when running the authentication applet:

at Java.lang.System.loadLibrary(Compiled Code)
at com.ibm.pkcs11.nat.NativePKCS11.<init>(Compiled Code)

This shows where the problems are caused. The line in the code looks as follows:

```
PKCS11 pkcs11 = new NativePKCS11 ("gclib"/*args [0]*/);
```

According to the documentation, the "gclib" argument is the dll (without extension) that needs to be loaded for PKCS#11 to work with the Smart Card and resides in the windows system directory on the local hard drive.

6.6. Conclusion

In order to ensure that only authorized clients have access to the confidential data in SHARE, the system needs a way of authentication clients in a secure manner. This is necessary to prevent access by any unauthorized clients, whether malicious or accidental. Through proper authentication, access could also be logged and if an authorized client misuses data obtained from SHARE, the client can be found out by checking on who accessed the data that was misused.

Because the system uses Smart Cards, it is more reliable. The private key is inaccessibly stored on the card and therefore cannot be extracted. This prevents the impersonation of the owner of the card and makes possession of the card necessary to access the confidential data. There is no possibility of impersonating the owner without the card because the authentication with the challenge-response protocol will not work without the card.

Because of the necessity of having a Smart Card, the issuing authority of the Smart Cards has better control on who gets one in the first place. It also makes access authority exclusive, because a Smart Card owner will lose his or her access privileges if they transfer the card to somebody else. The issuing authority could also program the keys on the Smart Card in such a way that they have a time limit, this would ensure that the cards are only usable for a certain period of time, after which they would have to be programmed once more with a new key pair and certificate. Finally, the access to the data could only be done through certain PCs, as a PC without a card reader cannot be used to access the data. This could add another level of security if the issuing authority keeps strict control of those PC that have card readers and the server is set up in such a way to ignore any connection attempts from a PC other than those set up with card readers by the issuing authority. This would prevent access through a PC with a card reader that the issuing authority has no control over.

One of the disadvantages of the system would be additional cost. The Smart Cards and Smart Card readers add initially additional costs to the system, plus the necessary labor in order to install the equipment and the need to train user in the use of the system also affect the system cost over a time period until it is up and running.

7. Summary

This thesis discusses the secure authentication for an administrator using Smart Cards. To guarantee a secure access by Smart Cards a SDSI certificate is stored on it. To do this, three applications were written: one server application and two stand-alone applications. One stand-alone application communicates with the server, the other one generates a SDSI certificate for the Smart Card with the public key stored on the same. The aim of this study is to use this implementation in the web based health study system SHARE.

The SDSIWriter program generates the necessary public-private key pair on the Smart Card and the necessary SDSI certificate, which is written onto the card.

The JSHARECard program reads the certificate from the card and sends it to SSLServer when the Smart Card is used for the first time and takes care of decrypting and digital signing of the challenge from SSLServer on subsequent uses.

The SSLServer program verifies and records the SDSI certificates sent to it and generates the challenge used to authenticate any JSHARECard client that requests access to the confidential data. If a JSHARECard client successfully answers the challenge, it will send a session key to that client.

8. Conclusions

8.1. Summary

As this is only a proof of concept, there are parts of the current implementation that need to be worked out in greater detail. As these parts require intimate knowledge of the use of cryptographic libraries, they ought to be done by somebody better versed in such matters. The current program was abstracted in such a way as to allow implementation of the cryptographic methods without having to know other parts of the program.
Although it would have been nice to continue the implementation into SHARE, it is considered that this is beyond the scope of this thesis. The logic had to be solid before the results of the project could be implemented into SHARE.

8.2. Goals and future work

For the proper implementation of the proposed framework, it would be necessary to continue as follows:
The SDSIWriter application currently only generates an artificial SDSI certificate. This would have to change to allow dynamic generation of true SDSI certificates that incorporate the user's public key, which is generated on the card, and maybe should also incorporate the principal's public key, which would be read from the local hard drive.
The JSHARECard application needs to properly decrypt the challenge. The challenge is currently sent from the server in an unencrypted state and therefore decrypting it would yield a response that would not be accepted by the server, as the server would fail the comparison and therefore deny the client the authentication. In order to prevent unauthorized access to the data, the server has to determine that the client is:

a.) authorized to access the data and

b.) not pretending to be somebody who has been authorized to access the data.

A valid certificate establishes authority to access the data, but in order to prove that the certificate holder really is the certificate holder and not an imposter, the client has to answer the challenge. If the client successfully decrypts the challenge, it means that the client can access the private key and decrypting functions on the smart card and therefore is the true owner of the certificate and the server can allow it to access the data. If the client fails the challenge, it means it's an imposter and data access will be denied.

The client application would also need to handle the session key it receives from the server in a different manner, as this key is only temporary and would not be a simple string message but an integer to be used for continued communications between the server and the client. This key would be unique every time a new connection is established. It would also encode a time limitation and could be used to compute a value for verification of its validity. Finally, the session key would have to be destroyed every time the connection is terminated for any reason.

The greatest work is left on the server end. Upon receiving a signed certificate, the server would have to extract the signature and the public keys and with the public key, verify the signature. With the users public key, then it should encrypt the challenge before sending it off to the client. The response in that case would have a signature with the user's public key. The server would then retrieve the users public key from the certificate and use it to verify if the signature for the decrypted challenge is genuine. If the challenge is correct and the signature is successfully verified, the server would send a random number that also encodes a time limitation and obeys some formula to ascertain the authenticity of the session key (e.g. the number encodes the current date and a random element in such a way to yield a prime which has a certain modulus or exponent). Subsequently any request from that client would have to be checked for the session key before any processing of that request is done.

The certificate should only be sent once for initial registration in order to limit its exposure to any possible interception attempts. Then to authenticate the client, only the challenge is needed because its successful decryption by the client proves that the client is in possession of the private key that belongs to the public key from the certificate. The

signature in this case provides another safeguard in this respect in case the challenge was correctly guessed by an unauthorized client that tried to impersonate the owner of a particular certificate.

Another interesting aspect would be to attack the system and test it for its security holes, privacy and confidentiality.

9. Appendix

9.1. Manual

Here is a brief description, on how to use the three applications:

9.1.1. Using the SDSIWriter

Ensure that the proper library is in the Java class path, the library being PKCS#11.

- Start the SDSIWriter from the Java console.
- Use the list and delete buttons to clear the card, the list function will display the items on the card in the console.
- If the list shows no objects on the card, it can be used to write a new key pair and certificate.
- Press on the new document button in the toolbar to generate a new key pair.
- Press on the open document button in the toolbar and in the popup dialog select write SDSI certificate to the card.
- Press the open button in the dialog. The dialog should then close.
- Press the write certificate button. A new certificate will be written to the card.
- Do a list object and the Java console should show that the card now has a public key, a private key and a SDSI certificate.
- Terminate the program by clicking on the close button in the top right corner.

9.1.2. Starting the SSLServer

- Ensure that the proper libraries are in the Java class path, the libraries being jsafe, sslj and certj from RSA. In addition, the class files for AppApplet, StdInRetriever, GUIRetriever, Retriever and TextAreaOutputStream have to be in the class path.
- Start the SSLServer from the Java console.
- The diagnostic output from SSLServer will be shown in the Java console window.

9.1.3. Using JSHARECard

- Ensure that the proper libraries are in the Java class path, the libraries being jsafe, sslj and certj from RSA. In addition, the PKCS#11 library has to be in the class path.
- Start the JSHARECard from the Java console.
- Enter a PIN and press enter on the keyboard.
- Enter a login name and press enter on the keyboard.
- Press the get cert button.
- Press the login button, the program will now try to connect with the server and send the certificate to the server. Once that is done, it will also try a challenge-response for verification.
- Press the clear fields button (if for some reason it doesn't work, restart the program).
- Enter a PIN and press enter on the keyboard.
- Enter a login name and press enter on the keyboard.
- Press the login button, the program will now try to connect with the server and do a challenge-response only.
- Terminate the program.
- Terminate SSLServer by closing the DOS window of the SSLSserver.
- The diagnostic messages from JSHARECard will be shown in the console output.

9.2. Messages

9.2.1. JSHARECard

When the stand alone application talks to the server, the JSHARECard program generates the following output messages. These messages were generated for diagnostic and illustrative purposes and would not be shown under normal circumstances:

```
loading JSHARECard.class for debugging...
JSHARECard.class successfully loaded
Loading CA certificates:
Loaded certificates
Starting signing, original message length: 499
Doing a digest operation on msg: (cert (i...
Signed message.
Sig:
(*æs¡¢kÓ:R·˜H3"Ù»åU'Ú'C[;^ìK8 Lw˜H|1õSöI,ß-ƒ ¶'„5?Ôü©?®p¢B¢±âÆOé
…µ#øÓÕñ<ÓÈjÍ}ÜÚx …I %ü-Ñ¾#©´

Sig length, bytes: 128  128
Creating SSLSocket.
dhreg-380/18.30.1.50  8888
Creating output stream.
Creating input stream.
Sending data to server.
 certi;silke;endlogin;(cert (issuer (name (public-key (rsa-pkcs1-
md5 (e #010001#) (n
|AMyTgZj6crpD+spNz0DA5D6wnwo0PwuVJmF6+M1nURCOtlA7/184ksWLgpjEG/q0E
egVrA3jntJMbFafpkALoLlFRJEb0yGR7purG6WNXm0EkY0OR/zSJjrWSQ+eGgyRjAA
nZktP8Tmwq+WVvzCqE+X64KzZGnM1S6c/x7ka4m8J|))) "silke")) (subject
(name (public-key (rsa-pkcs1-md5 (e #010001#) (n
```

|AMyTgZj6crpD+spNz0DA5D6wnwo0PwuVJmF6+M1nURCOtlA7/184ksWLgpjEG/q0E
egVrA3jntJMbFafpkALoLlFRJEb0yGR7purG6WNXm0EkY0OR/zSJjrWSQ+eGgyRjAA
nZktP8Tmwq+WVvzCqE+X64KzZGnM1S6c/x7ka4m8J|))) SDSI
Test)))(*æs¡¢kÓ:R·ˇH3"Ù»åU'Ú'C[;^ìK8 LwˇH|1õSöI,ß-ƒ ¶'„5?Ôü©?®p¢B¢
±âÆOé
…µ#øÓÕñ<ÓÈjÍ}ÜÚx …I %ü-Ñ¼#©´

Response received from server: Certificate accepted.
Server accepted the certificate.
Connection close. Exiting.
Creating SSLSocket.
dhreg-380/18.30.1.50 8888
Creating output stream.
Creating input stream.
Sending data to server.
 login;silke;;;
Challenge received: challenge;hŸ×°…në¤
challenge, length: hŸ×°…në¤;;;8
Doing a digest operation on msg: hŸ×°…në¤...
Signed message.
Sig: T Æ(A< xƒ ª-£Yw.QF*ù Û§¶é Ó\ê„Ð _èµR
 ï?†ªúÌáÏ""ˇHÄ© T÷ëjSâî•n¸éB:5:‡Ÿ÷§ ®µ«a Š°
ä™éµ†,ÞÛ £-t,°Ísª$'¬.c¨è
V>-ÏýÍýÏR¼™] 1?Tø
Sig length, bytes: 128 128
Challenge response from server: Key;This is a session key from the server.
Connection close. Exiting.
Creating SSLSocket.
dhreg-380/18.30.1.50 8888
Creating output stream.
Creating input stream.

```
Sending data to server.
 login;silke;;;
Challenge received: challenge;hŸx°…pë¤
challenge, length: hŸx°…pë¤;;;8
Doing a digest operation on msg: hŸx°…pë¤...
Signed message.
Sig: T  Æ(A< xƒª -£Yw.QF*ù Û§¶é Ó\ê„Ð _èµR
 ï?†ª úÌáÏ"”˜HÄ© T÷ëjSâî•n¸éB:5:‡Ÿ÷§ ®µ«a Š°
ä™éµt,ÞÛ £-t,°Ísª$'¬.c¨è
V>-ÏýÍýÏR¼™] 1?Tø
Sig length, bytes: 128  128
Challenge response from server: Key;This is a session key from the
server.
Connection close.  Exiting.

The communications channel closed.
```

9.2.2. SSLServer

When the stand alone application talks to the server, the SSLServer generates the following output messages. These messages were generated for diagnostic and illustrative purposes and would not be shown under normal circumstances:

```
[SSLServer]  Beginning SSL Server sample.
Done loading certificates.
Waiting for a connection.
ServerSocket[addr=dhreg-380/18.30.1.50,port=0,localport=8888]
[SSLServer]  Connection received.
[SSLServer]
Creating output stream.
```

```
Creating input stream.
attempting to read data.
incomming data: certi;silke;endlogin;(cert (issuer (name (public-
key (rsa-pkcs1-md5 (e #010001#) (n
|AMyTgZj6crpD+spNz0DA5D6wnwo0PwuVJmF6+M1nURCOtlA7/184ksWLgpjEG/q0E
egVrA3jntJMbFafpkALoLlFRJEb0yGR7purG6WNXm0EkY0OR/zSJjrWSQ+eGgyRjAA
nZktP8Tmwq+WVvzCqE+X64KzZGnM1S6c/x7ka4m8J|))) "silke")) (subject
(name (public-key (rsa-pkcs1-md5 (e #010001#) (n
|AMyTgZj6crpD+spNz0DA5D6wnwo0PwuVJmF6+M1nURCOtlA7/184ksWLgpjEG/q0E
egVrA3jntJMbFafpkALoLlFRJEb0yGR7purG6WNXm0EkY0OR/zSJjrWSQ+eGgyRjAA
nZktP8Tmwq+WVvzCqE+X64KzZGnM1S6c/x7ka4m8J|))) SDSI
Test)))(*æs¡¢kÓ:R·ˇH3"Ù»åU'Ú'C[;^ìK Lwˇ H|1õSöI,ß-ƒ ¶'„5?Ôü©?®p¢B¢
±âÆOé
…µ#øÓÕñ<ÓÈjÍ}ÜÚx …I %ü-Ñ¾#©´ ];† Z4 6H - °E'?J<Å
 í>Àjé%‡'ÚæÉ¸Finished;;;
Saving cert to file: silke
attempting to read data.
Caught EOF on primary read.
incomming data:
Waiting for a connection.
ServerSocket[addr=dhreg-380/18.30.1.50,port=0,localport=8888]
[SSLServer]  Connection received.
[SSLServer]
Creating output stream.
Creating input stream.
attempting to read data.
incomming data: login;silke;;;
Cert loaded.
Sending challenge.
incomming data, second read: respo;hŸx°…pë¤T  Æ(A< ×ƒª -
£Yw.QF*ù Û§¶é Ó\ê„Ð _èµR
 ï?†ªúÌáÏ""ˇHÄ© T÷ëjSâî•n¸éB:5:‡Ÿ÷§ ®µ«a Š°
```

```
ä™éµt,ÞÛ £-t,°Ísª$'¬.c"è
V>-ÏýÍýÏR¼™] 1?TøFinished;;;
Signature verification success.
Challenge successfully answered.
Challenge reply is good.
attempting to read data.
Caught EOF on primary read.
incomming data:
Waiting for a connection.
ServerSocket[addr=dhreg-380/18.30.1.50,port=0,localport=8888]
[SSLServer]  Connection received.
[SSLServer]
Creating output stream.
Creating input stream.
attempting to read data.
incomming data: login;silke;;;
Cert loaded.
Sending challenge.
incomming data, second read: respo;hŸx°…pë¤T  Æ(A< ×ƒª -
£Yw.QF*ù Û§¶é Ó\ê„Ð _èµR
 ï?tªúÌáÏ""˜HÄ© T÷ëjSâî•n¸éB:5:‡Ÿ÷§ ®µ«a Š°
ä™éµt,ÞÛ £-t,°Ísª$'¬.c"è
V>-ÏýÍýÏR¼™] 1?TøFinished;;;
Signature verification success.
Challenge successfully answered.
Challenge reply is good.
attempting to read data.
Caught EOF on primary read.
incomming data:
Waiting for a connection.
ServerSocket[addr=dhreg-380/18.30.1.50,port=0,localport=8888]
```

9.3. Storing of Certificates

These are examples of an X.509 and a SDSI certificate, which can be stored on a Smart Card:

X.509:

Object:
CLASS: CERTIFICATE
TOKEN: true
PRIVATE: false
LABEL: Silke Lieber's GemSAFE Canada Inc. ID
VALUE: 30 82 04 F7 30 82 04 60 A0 03 02 ... (1275 bytes)
CERTIFICATE_TYPE: X_509
...

Currently the only *CERTIFICATE_TYPE* defined in the Java libraries is X_509.

SDSI:

3 object:
CLASS: DATA
TOKEN: true
PRIVATE: true
LABEL: SDSI CERT
APPLICATION:
VALUE: 28 63 65 72 74 20 28 69 73 73 75 ... (512 bytes)
MODIFIABLE: true
SIZE: 528

9.4. UML-Diagrams

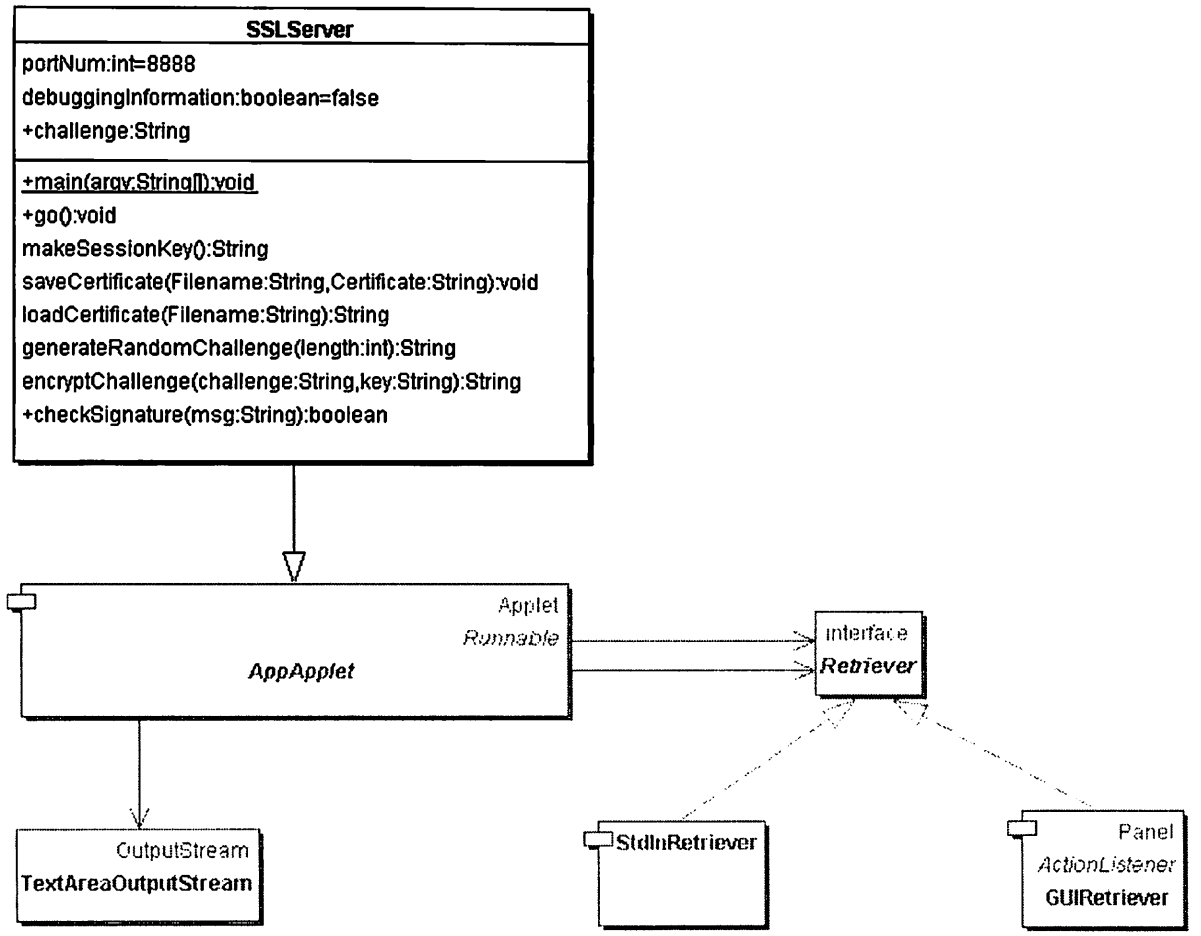

Figure 9: UML diagram of the SSLServer

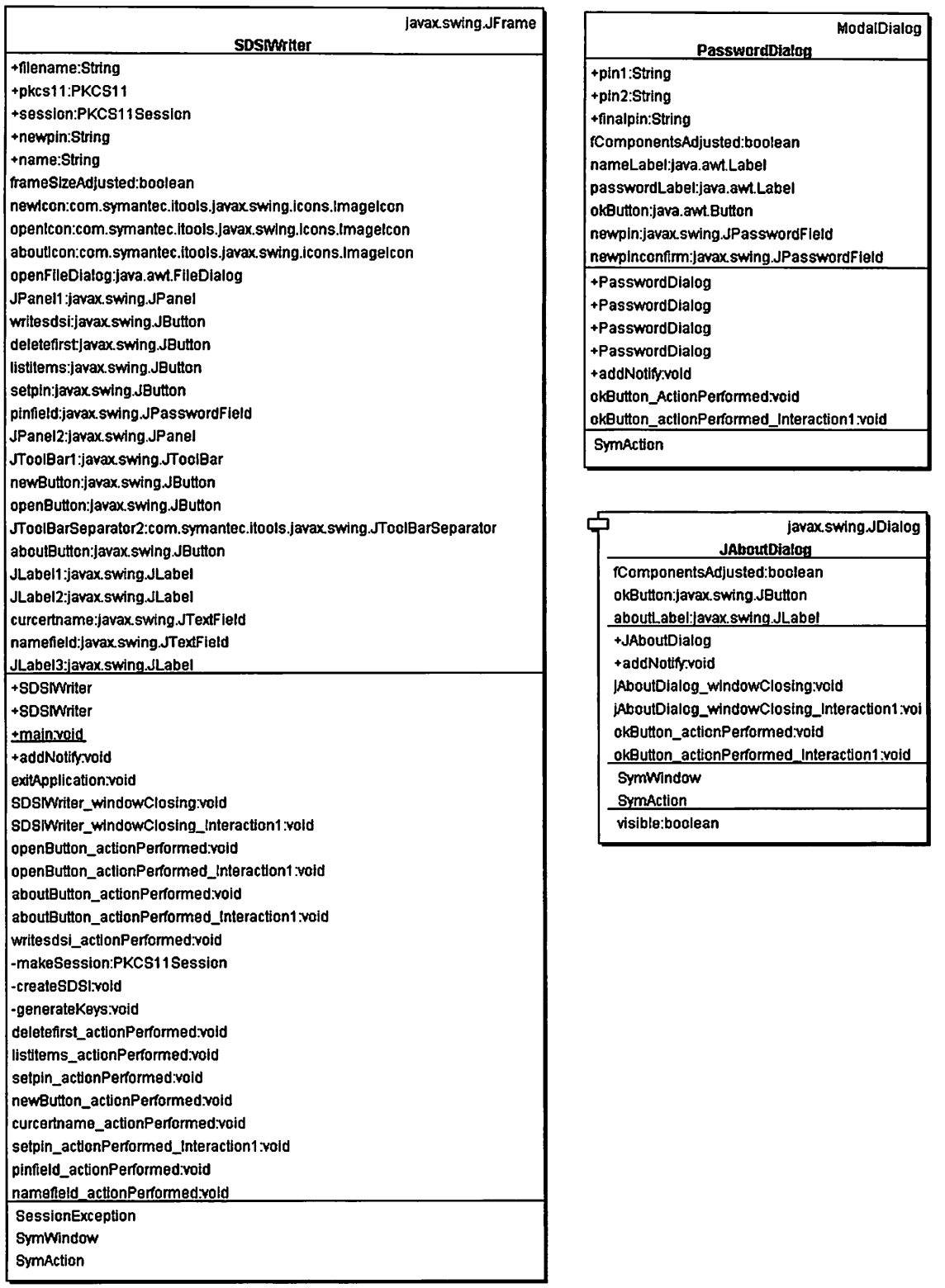

Figure 10: UML diagram of the SDSI Writer

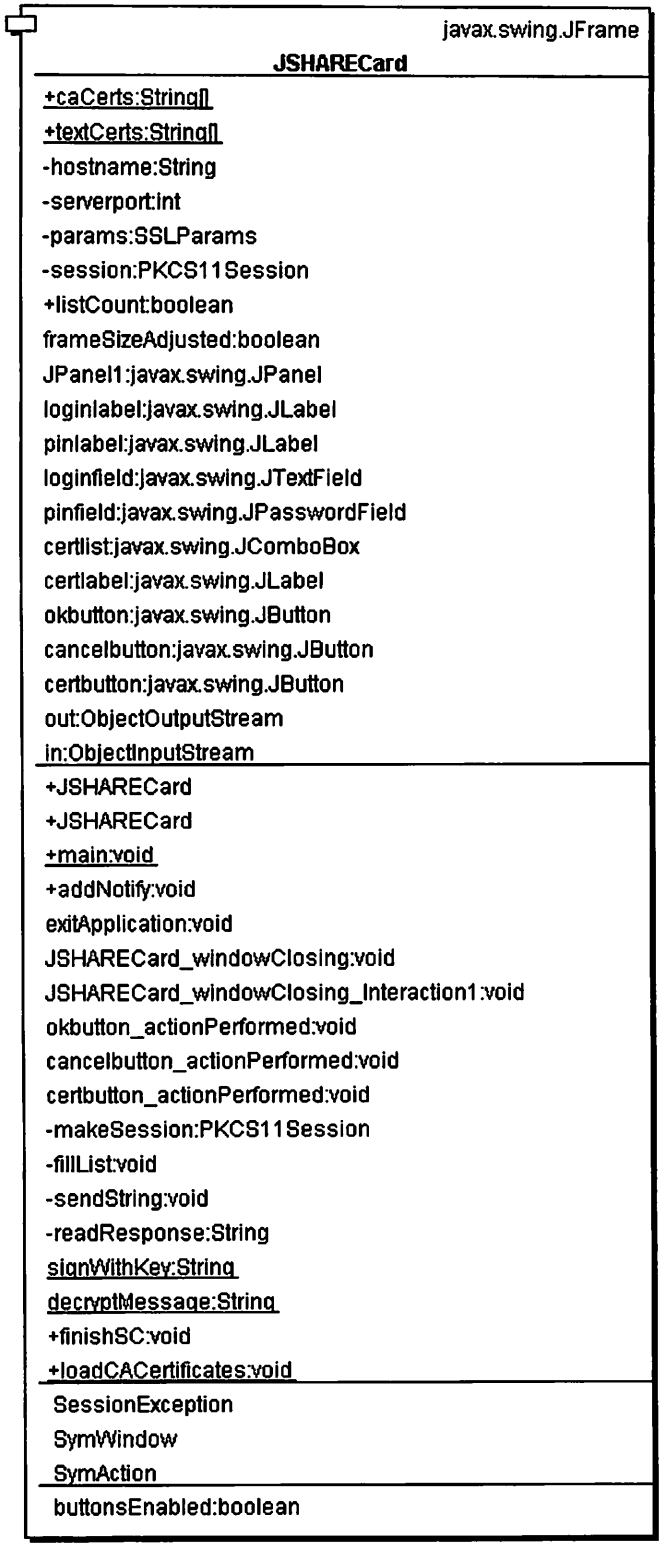

Figure 11: UML diagram of the JSHAREServer

10. References

A. Beutelsbacher, J. Schwenk, K.-D. Wolfenstetter, Moderne Verfahren der Kryptographie, von RSA zu Zero-Knowledge, 3rd edition, Vieweg, 1999

B. Eckel, Thinking in Java, www.bruceckel.com

Foldoc – computer dictionary can be found at http://foldoc.doc.ic.ac.uk/

Links to GemPLUS and GemSAFE can be found at
http://www.gemplus.com and http://www.gemsafe.com

U. Hansmann, M.S. Nicklous, T. Schäck, F. Seliger, Smart Card, Application Development Using Java, Springer, 2000

J. Hunter, W. Crawford, Java Servlet Programming, O'Reilly publisher, 1998

S. Oaks, JAVA Security (Java 1.2), O'Reilly publisher, 1998

W. Rankl, W. Effing, Smart Card Handbook, John Wiley and Sons, 1997

R. Rivest, A. Shamir, L. Adleman, A method for obtaining digital signature and public key cryptosystems. Communications of the ACM, 21:120-126, 1978

RSA user's manual can be found at www.rsasecurity.com

B. Schneier, Applied Cryptography, 2nd Edition, John Wiley and Sons, 1996

Links to SDSI and SPKI materials can be found at http://theory.lcs.mit.edu/cis/sdsi.html

USENIX Workshop on Smart Card Technology can be found at http://www.usenix.org

M. Wu, Secure Health Information Sharing System (SHARE), Thesis, MIT 2001

M. Wu, L. Mui, M. Mohtashemi, P. Szolovits, Secure Health Information Sharing System: SHARE, 2001 Fall Symposium of the American Medical Informatics Association

11. Glossary and Abbreviations

API (Application Program Interface),

Is the interface by which an application program accesses the operating system and other services. An API is defined at source code level and provides a level of abstraction between the application and the kernel (or other privileged utilities) to ensure the portability of the code. An API can also provide an interface between a high level language and lower level utilities and services, which were written without consideration for the calling conventions supported by compiled languages. In this case, the API's main task may be the translation of parameter lists from one format to another and the interpretation of call-by-value and call-by-reference arguments in one or both directions.

Applet,

Is a Java program, which can be distributed as an attachment in a World-Wide Web document and executed a Java-enabled web browser such as Sun's HotJava or Netscape Navigator version 2.0 or later.

Navigator severely restricts the applet's file system and network access in order to prevent accidental or deliberate security violations. Full Java applications, which run outside of the browser, do not have these restrictions.

Web browsers can also be extended with plug-ins though these differ from applets in that they usually require manual installation and are platform-specific. Various other languages can now be embedded within HTML documents, the most common being JavaScript.

Client,

Is a computer system or process that requests a service of another computer system or process (a "server") using some kind of protocol and accepts the server's responses. A client is part of a client-server software architecture. For example, a

workstation requesting the contents of a file from a file server is a client of the file server.

EPROM, EEPROM, ROM, and RAM,

Are forms of memory. RAM (Random Access Memory) allows the computer to store and retrieve data. ROM is read-only-memory that cannot be changed and is initialized by the manufacturer. The end user can write to EPROM once and EEPROM can be written to several times by the end user.

JVM (Java Virtual Machine),

Is a specification for software, which interprets Java programs that have been compiled into byte-codes, and usually stored in a "class" file. The JVM instruction set is stack-oriented, with variable instruction length. Unlike some other instruction sets, the JVM's supports object-oriented programming directly by including instructions for object method invocation (similar to subroutine call in other instruction sets).

The JVM itself is written in C and so can be ported to run on most platforms. It needs thread support and I/O (for dynamic class loading). The Java byte-code is independent of the platform.

OCF: Open Card Framework

PC/SC: Personal Computer / Smart Card Interface

PGP (Pretty Good Privacy),

Is a high security RSA public-key encryption application for MS-DOS, Unix, VAX/VMS, and other computers. PGP was distributed as "guerrilla freeware". PGP uses a public-key encryption algorithm. PGP allows people to exchange files or messages with privacy and authentication. Privacy and authentication are provided without managing the keys associated with conventional cryptographic software. No secure channels are needed to exchange keys between users, which makes PGP much easier to use. This is because PGP is based on public-key cryptography.

PKCS (Public Key Cryptographic Specification)

Is a set of specifications defined by RSA for the encryption, decryption, etc. of data.

Server,

1. A program, which provides some service to other (client) programs. The connection between client and server is normally by means of message passing, often over a network, and uses some protocol to encode the client's requests and the server's responses. The server may run continuously (as a daemon), waiting for requests to arrive or it may be invoked by some higher-level daemon, which controls a number of specific servers. There are many servers associated with the Internet, such as those for Network File System, Network Information Service (NIS), Domain Name System (DNS), FTP, news, finger and Network Time Protocol. On Unix, a long list can be found in /etc/services or in the NIS database "services". See client-server.

2. A computer, which provides some service for other computers, connected to it via a network. The most common example is a file server, which has a local disk and services requests from remote clients to read and write files on that disk, often using Sun's Network File System (NFS) protocol or Novell Netware on IBM PCs.

Servlet (Java Servlet),

Is a Java program that runs as part of a network service, typically an HTTP server and responds to requests from clients.

The most common use for a servlet is to extend a web server by generating web content dynamically. For example, a client may need information from a database; a servlet can be written that receives the request, gets and processes the data, as needed by the client and then returns the result to the client.

Applets are also written in Java but run inside the JVM of a HTML browser on the client. Servlets and applets allow the server and client to be extended in a modular way by dynamically loading code, which communicates with the main program via a standard programming interface.

Servlets are more flexible than CGI scripts and, being written in Java, more portable.

SMIME: Secure mail

SSH (Secure Shell),

Is a Unix shell program for logging into and executing commands on a remote computer. SSH is intended to provide secure encrypted communications between two untrusted hosts over an insecure network. X11 connections and arbitrary TCP/IP ports can also be forwarded over the secure channel.

SSL (Secure Sockets Layer),

Is a protocol designed by Netscape Communications Corporation to provide encrypted communications on the Internet. SSL is layered beneath application protocols such as HTTP, SMTP, Telnet, FTP, Gopher, and NNTP and is layered above the connection protocol TCP/IP. It is used by the HTTPS access method.

TLS: Transport Layer Security

12. Acknowledgement

This thesis was completed in cooperation of the Fachhochschule Ulm, the Massachusetts Institute of Technology, and Gemplus. I like to thank my Professors Kongehl and Szolovits and all the people who helped me with this thesis. I also thank Gemplus and the Ulmer Akademie für Datenschutz und IT-Sicherheit UDIS e. V for the financial support.

My thanks goes also to my parents for the nice DELL laptop and for taking care of my goldfishes during my time here in Boston.

I also would like to thank my roommate Roberto for taking care of my nutrition and teaching me, what "al dente" means.