2000

Smart Card-enabled security services to support secure telemedicine applications

Monoreet Mutsuddi
West Virginia University

Smart Card enabled security services to support Secure Telemedicine Applications

Monoreet Mutsuddi

Thesis Submitted to the College of Engineering & Mineral Resources
at
West Virginia University
in partial fulfillment of the requirements for
the degree of

Master of Science
In
Computer Science

V.Jagannathan, Ph.D.,Chair
Sumitra Reddy, Ph.D.
Ramana Reddy, Ph.D.

Department of Computer Science & Electrical Engineering

Morgantown, West Virginia
2000

Keywords : RSA, Digital Certificates, Encryption,Authentication,SmartCard

Abstract

Smart Card enabled security services to support Secure Telemedicine Applications

Monoreet Mutsuddi

The Internet is quickly evolving into a standard and widely used mode of information transfer. Therefore internet's information transfer capabilities could be exploited by Secure Telemedicine Applications to securely transfer medical information . However the current day internet protocol stack does not provide much in terms of security for the data being transferred across it. In this thesis we explore how Cryptography along with Smart Cards can be used to implement application layer protocols, to enhance the security of data flowing across the internet in a Secure Telemedicine environment. This thesis also discusses the design and implementation of various system software components required to implement such an application layer protocol.

Acknowledgement

I thank my advisor, Dr Juggy Jagannathan, for his support, guidance and patience through out my research work at Concurrent Engineering Research Center, West Virginia University.

I also thank my committee members : Dr Sumitra Reddy and Dr Ramana Reddy for their help and guidance.

I would also like to thank Mr. Ravi Raman for his invaluable guidance and insight. He motivated me to work hard and exposed me to new technologies.

I am also grateful to Srivatsan Kannan ,William Hunt and Chaoxin Sima for their support through out my research work at Concurrent Engineering Research Center.

Table Of Contents

List Of Figures

1. Introduction

1.1 Description of the Problem

The Secure Telemedicine Model is a system which allows information both medical and administrative to be distributed over a public network and tries to bring them together to the advantage of all , namely doctors, patients, nurses , administrative professionals etc. It tries to make medical information easily accesible regardless of the geographic location of the patient or the doctor. Needless to say the internet is exploited to carry out transfer of medical and administrative information from place to place. These information could be patient records or other medical information shared by doctors and/or patients. Information could also be classified as multimedia or pure data. As the name "Secure Telemdicine" suggests, security is one of the key issues. Medical information is inherently sensitive and hence must be available only to people who are supposed to have it. Providing security for medical information can be broken down further into providing a secure storage for medical information, providing secure transport of medical information and providing access to medical information only after proper authentication.

The current internet does not provide support for the above. The internet is inherently insecure and data travelling across it can be easily sniffed.

Proper authentication measures are essential in a secure telemedicine system as two distant entities can no longer rely on their physical senses to authenticate each other. Existing technologies like the public key cryptography can be exploited to implement strong authentication measures. A public key infrastructure consists of a certificate which binds an entity to a

public key, a private key corresponding to the public key and a trusted certifying authority. Some of the key issues relating to this kind of authentication are

1. How to verify that the other entity has the private key corresponding to the presented certificate.
2. How to protect against third party attacks, that is making sure that the received certificate indeed belongs to the other legitimate entity and not to some third party who has somehow managed to sniff internet packets and replaced the legitimate certificate with his own.
3. How to securely exchange certificates in the absence of some central certificate repository.

One of the key problems in public key cryptography is mobility. That is a persons public/private key and certificate is typically stored in a file on his computer. This means a person cannot use his keys if he logs on to some other computer. One of the aims of secure telemedicine is to achieve a high degree of mobility.

Another problem is security of the keys themselves. Smart Cards can help to solve both of the above problems. Storing the public/private key pair and the certificate on the smart card means the owner of the smart card can use his keys on any computer as long as the computer has a smart card aware cryptographic service provider. Smart Card also enhances security. Storing and generating keys on the smart card means that the keys are always with the user and not on the computer. Access to keys on smart cards are protected by pins and hence provide a higher level of security.

1.2 Objectives

The objective of this thesis are

1. To show how public key cryptography enhanced with smart cards can be used to provide strong authentication between two distant entities in a secure telemedicine environment.
2. To demonstrate the use of smart cards as mobile containers for carrying certificate , public/private key pairs and logon information in a secure telemedicine environment.

In order for the above objectives to be met the following components are to be developed.

1. A smart card aware cryptographic service provider . (CSP)
2. Native(C ++) routines which implement the necessary authentication steps and other data retrieval steps from the smart cards.
3. A Java Native Interface to provide a interface between the calling java program and the authentication and other data retrieval routines.
4. A CORBA service to coordinate the authentication mechanism.
5. An implementation of the GINA.dll to allow smart card aware logon/locking facility.

1.3 Preview of Chapters

Rest of the thesis is organized as follows. Chapter 2 discusses the various cryptographic concepts and protocols. It also gives a brief background on Smart Cards. Chapter 3 discusses design issues. Chapter 4 discusses the implementation details. Finally chapter 5 presents conclusion and future work.

2. Background

This chapter covers two major areas :

Cryptography

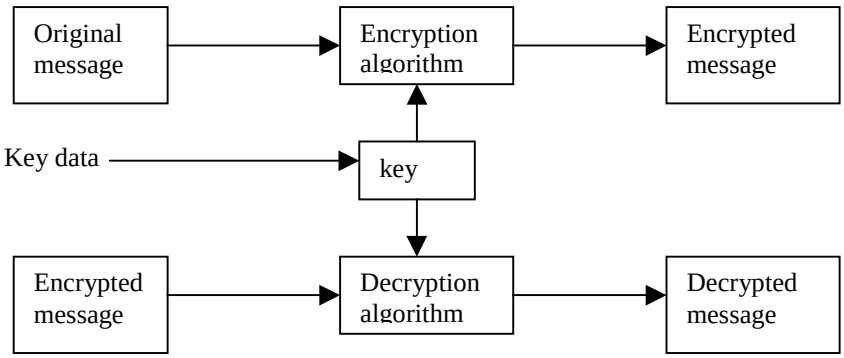
Smart Card Technology

2.1 Cryptography

2.1.1 Symmetric Key Cryptography

In symmetric key cryptography the, the same key is used to encrypt as well as decrypt the data.

Figure 1. Symmetric Key Operations



Symmetric key algorithms can be classified into block ciphers and stream ciphers. Block ciphers encrypt by breaking up the input data into blocks whereas stream ciphers encrypt data one unit at a time where a unit can be bit or a byte. Hence stream ciphers can encrypt variable length data.

2.1.1.1 Block Ciphers

Block ciphers can in turn be divided into 2 categories. Electronic Code Book (ECB) and Cipher Block Chaining (CBC). ECB encrypts each block of data independent of the previous block. Hence same data will always yield the same cipher in this mode. CBC on the other hand will modify each block of input by XORing it with the previous block of output before encrypting it. An initialization vector is required to encrypt the first block. Hence if the IV is different the encrypted cipher will be different for the same input data.

Figure 2. ECB Mode

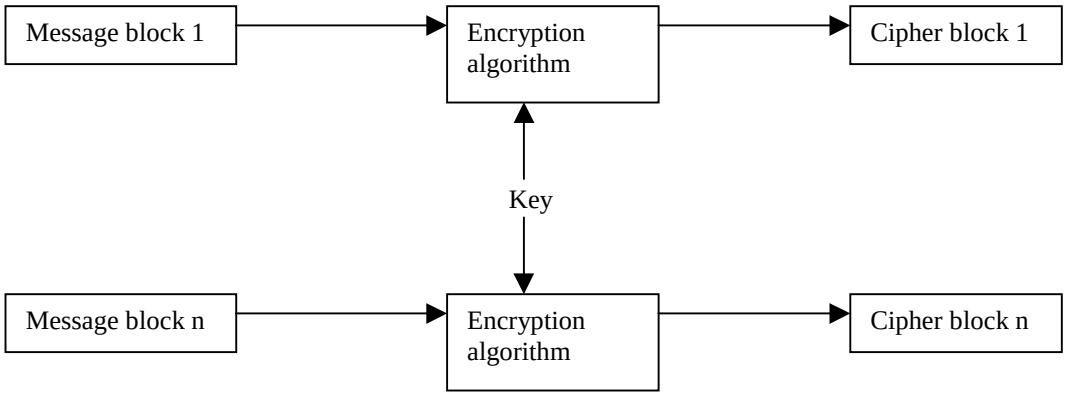
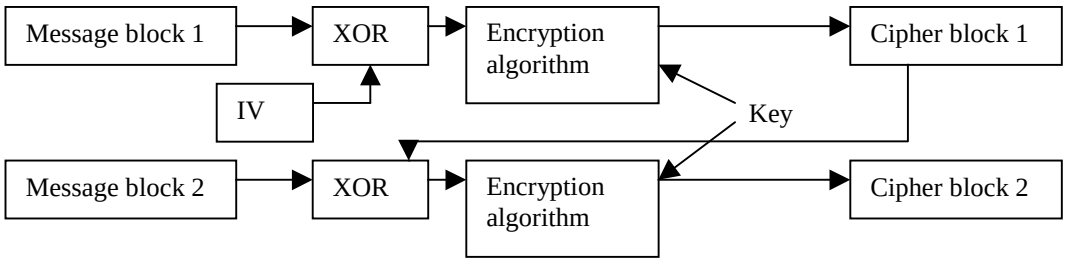


Figure 3. CBC mode



Some of the block ciphers algorithms are DES, Triple DES, RC2, RC5, DESX.

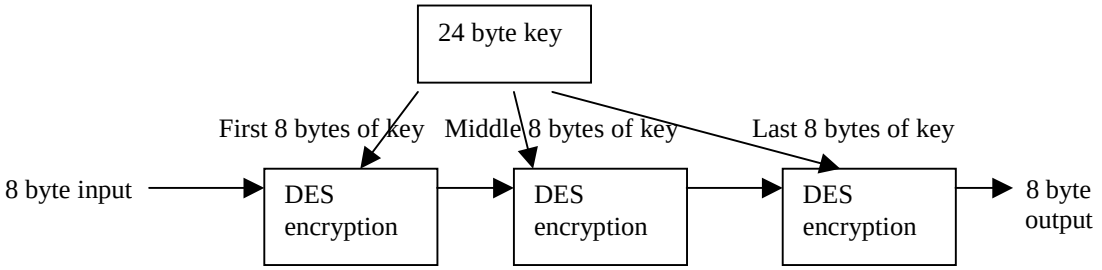
DES

DES divides the input into 8 byte blocks and produces 8 byte long output. The key length is 8 bytes.

Triple DES

Triple DES executes DES 3 times. Key length is 24 bytes

Figure 4. Triple DES Operation



DESX

DESX is an RSA data security extension of DES encryption algorithm. It increases effective key bits from 56 bits to 120 bits.

2.1.1.2 Message Digests

Message digest algorithms map input data into fixed length output digests. This algorithm is not random and digesting the same input will always produce the same output. Some of the essential features of a message digest algorithm are

1. It produces a fixed length output.
2. It accepts data of any length
3. Its computationally infeasible to guess an input data that would produce a specific output.
4. Its computationally infeasible to produce 2 different input blocks produce the same digest.

Since minor changes in data will produce different digests, message digests can be used to authenticate data. Also it finds use in digital signatures where its infeasible to encrypt a large chunk of data using public key crypto systems.

2.1.2 Public-Key Cryptography

Public key cryptography was invented in 1976 by Stanford graduate student Whitfield Diffie and Stanford professor Martin Hellman. In this system each person holds a pair of keys, called the private key and the public key. The key pair's owner publishes the public key and keeps the private key secret.

In this system if Alice wants to send a message to bob, she would encrypt the message using bobs public key. Now this encrypted message cannot be decrypted by any one else, but the person who holds the corresponding private key, which in this case would be Bob. Unlike symmetric - key cryptography the key used to encrypt data will not decrypt the message. Hence knowledge of Bob's public key would not help anyone decrypt the message.

2.1.2.1 The RSA Algorithm

RSA is a public key crypto system for both encryption and authentication that MIT professors Ronal L. Rivest, Adi Shamir and Leonard M. Adleman invented in 1977.

The RSA algorithm employs prime numbers and modulur mathematics.

Modulur Math

For any mod n, the only numbers under consideration are 0 to (n-1).

Example if modulus =5

3*3 is not 9, but 3*3 = 4 mod 5

2+4 is not 6, but 2+4 = 1 mod 5

2 numbers are inverse of each other in modulus n if their product equals 1 in modulus n.

Example if moduls =5

3*2 = 1 mod 5. Hence 3 and 2 are inverses of each other in modulus 5.

In the simplest form the public key consists of a modulus and a public exponent (n,e) and the private key consists of a modulus the same as the public modulus and a private exponent (n,d).

They are calculated as follows

1. Choose a public exponent e.
2. Choose 2 large prime numbers p and q such that pq > e, and e is relatively prime to (p-1)(q-1)
3. Calculate d such that ed = 1 mod (p-1)(q-1)
4. n = pq

The RSA algorithm works as follows

To compute cipher text c from plain text message m, find

$c = m^e \bmod n$

To decrypt, compute

$m = c^d \bmod n$

The relationship between e and d ensures that the algorithm recovers the original message m, since

$c^d = (m^e)^d = m^{ed} = m^1 = m \bmod n$

The security of the system relies on the fact that if you know p,q and e its easy to compute d, but if you know only n and e its difficult to determine d.

Security

Its possible to find out d, if one knows n and e. But for that one has to factor n to get p,q and its extremely time consuming to factor large numbers. This is why the RSA algorithm is secure.

2.1.2.2 Digital Envelopes

The downside of RSA algorithm is that its time consuming to encrypt and decrypt large amounts of data and for some applications might be too slow for any practical use. Symmetric key cryptography is much faster, but transferring the symmetric key across is a problem. Digital envelopes which combine public key cryptography with symmetric cryptography is the solution to this problem. The sender encrypts the data using a symmetric key. Then encrypts the symmetric key with the recipients public key. The recipient can then get the symmetric key by decrypting with his private key. Once the recipient has the symmetric key , the recipient can decrypt the message.

Figure 5. Sealing Operation

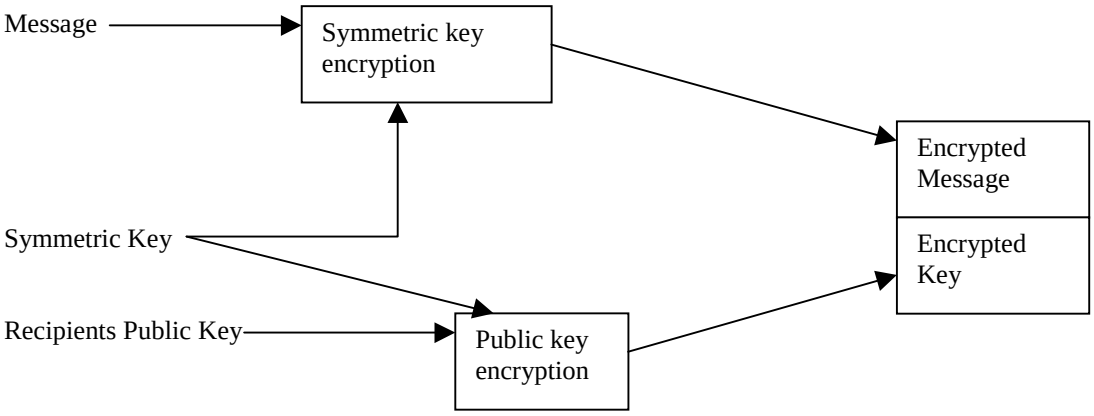
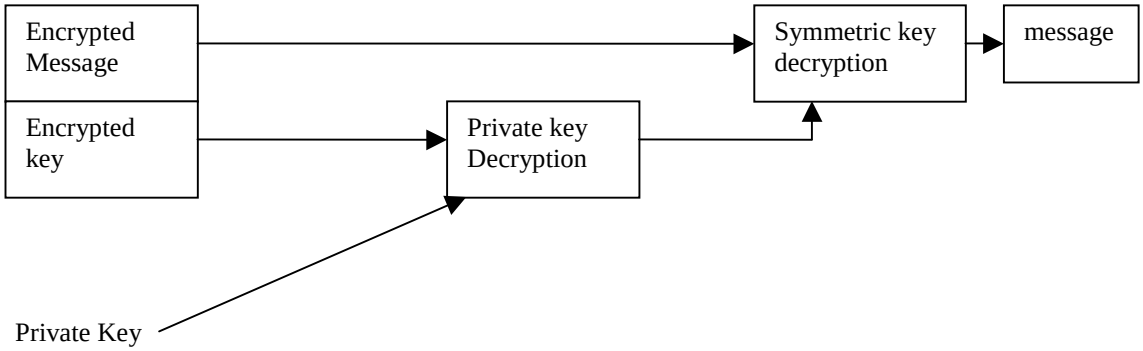


Figure 6. Envelope Opening Operation



2.1.2.3 Digital Signatures

The process of digital signature consists of the following steps.

1. Hash the data using some hashing algorithm like MD5,MD2 or SHA1
2. Encrypt the hash produced with the private key.

Digital signature can be used to authenticate as well as testing to see whether a document has been altered.

Alice and bob compose a contract in digital format. This contract is digitally signed by each parties private key. The contract now consist of the file and 2 copies of the signature. This contract can be distributed to alice and bob. Now suppose at some later time, bob comes up with a file which is different from alice's. We can easily find out which one of the above changed the contract. All we have to do is hash alice's and bobs files. Then we decrypt the signature copies of both alice's and bobs. One of their decrypted signature will not match with hashed files and hence we can find out whose file has been changed.

2.1.2.4 Authentication.

By authentication we mean is someone really is, what he/she claims to be. To verify a persons identity a public key can be acquired belonging to the person from a third party certifying authority. This public key can be used to encrypt some random data generated by the verifying side. This encrypted data would be sent across to the person. This person would decrypt the data using his/her private key, and send the decrypted information back to the verifying side. If the information sent by the person is the same as the information encrypted on the verifying side, the person can be authenticated.

2.1.2.5 Digital Certificates

Digital certificates are the standard way to publish ones public keys. Digital certificates bind a persons identity to his/hers public key. Digital certificates are issued by trusted third party certificate issuers.

Certificates are issues as follows:

1. Alice sends a certification request containing her name and her public key to the certifying authority.
2. The certifying authority forms a special message from alice's request and signs it with its own private keys. The certifying authority returns the message and the signature to alice; the 2 parts together form a certificate.
3. Alice sends the certificate to bob to convey trust in her public key.
4. Bob verifies the certificate by verifying the signature contained in the certificate with the certifying authorities public key. If the signature is valid, bob accepts alice's public key.

This assumes that bob has the certifying authorities certificate. Bob can develop trust in the certifying authority's public key recursively, if he has a certificate containing the certifying authorities public key signed by a superior certifying authority.

A broader application of digital certification not only includes alice's name and public key but also other information.

2.1.3 Cryptographic Protocols

2.1.3.1 Key Exchange

A common technique for keeping a message from the hands of malicious individual's is to encrypt the message with a session key. Session keys are destroyed after the message has been successfully communicated. One major challenge to this approach is : How to safely let the conversants know the session key.

2.1.3.1.1 Key Exchange with Symmetric Cryptography

This protocol assumes that both Alice and Bob share a secret key with a trusted central authority on the network called the key distribution authority(KDC).

1. Alice calls KDC and asks for a session key to communicate with Bob.
2. KDC generates a random key, makes 2 copies of it. Encrypts one copy with Alice's key and the other with Bob's key. KDC sends both copies to Alice.
3. Alice receives the 2 encrypted keys. He decrypts his copy of the session key. He sends the other encrypted copy to Bob.
4. Bob decrypts his copy of the session key.
5. Now both Alice and Bob share a secret key which can be used to communicate between them.

This protocol relies very heavily on the security of KDC. If a malicious individual Mallory breaks into KDC, he has access to all the secrets shared between KDC and all individuals. Thus he can carry out a man-in-the middle attack very easily. For example after KDC has encrypted the session key on Alice's request. Mallory can

sniff the encrypted message decrypt the key, substitute a random key which he generates, make 2 copies of it , encrypt it with Alice and Bob's session key. Then send it to Alice. Alice has no way of knowing that the message came from Mallory and not from KDC.

2.1.3.1.2 Key Exchange with Public Key Cryptography

This is a hybrid crypto system where in which Alice and Bob use public key cryptography to agree on a session key.

1. Alice acquires Bob's public key from a KDC.
2. Alice generates a random session key and encrypts it using Bob's public key. She sends the encrypted key to Bob.
3. Bob retrieves the session key by decrypting the message from Alice with his private key.
4. Bob Alice and Bob can communicate using the same session key.

The above protocol suffers from a man-in-the middle threat.

Mallory could substitute his public key for both Alice's an Bob's.

Now when Alice encrypts message using Bob's public key (actually Mallory's public key). Mallory can sniff the message, decrypt it using her private key. Read it . Encrypt it using Bob's public key and send it to Bob. He can do the same when Bob sends a encrypted message to Alice.

Alice and Bob have no way of verifying whether they are talking to each other.

2.1.3.1.3 Interlock Protocol

This protocol invented by Ron Rivest and Adi Shamir, tries to foil the the main in the middle attack.

1. Alice sends Bob her public key.
2. Bob sends Alice her public key.

3. Alice encrypts her message using Bob's public key. She sends half of the encrypted messaege to Bob.
4. Bob encrypts his message using Alice's public key. He sends half of the encrypted messaege to Alice.
5. Alice sends the rest of the encrypted message to Bob.
6. Bob concatenates the two halves and decrypts it using his private key. Bob sends the other half of the encrypted message to Alice.
7. Alice concatenates the two halves of Bob's message together and decrypts it with her private key.

This approach causes problems for the man in the middle attack. Mallory can still substitute his public key for Alice and Bob's. But when he gets Alice's first message, she cannot decrypt it as its only half the message. Mallory has to invent a totally new message. Encrypt it using Bob's public key and send half of it to Bob. He has the same problem when Bob sends a message. By the time he receives the second half of the messages its too late for him to change his new message. Mallory can still get away if she knows both Alice and Bob well enough to mimic a relevant conversation between them.

2.1.3.1.4 Key Exchange with Digital Signatures

The man in the middle attack can be avoided by enhancing the key exchange protocols with digital signatures. In this protocol the KDC signs both Alice's and Bob's public keys. Before KDC signs their public keys, he makes sure that they have the private key corresponding to the public key they are providing. When Alice and Bob receive the keys, they each verify KDC's signature. Then they can proceed with the key exchange protocol. This causes Mallory a lot of problems. He cannot pose as Alice or Bob as he does not have their private keys. He also cannot substitute his keys for Alice or Bob as its not signed as being Alice or Bob. This protocol will fail if

Mallory breaks into KDC, acquires its private key and creates phony signed keys to fool Alice and Bob. Then he could replace the real keys in the database with these phony keys. This enables him to launch a man in the middle attack and read messages.

2.1.3.2 Authentication

2.1.3.2.1 Authentication using One-Way Functions

Roger Needham and Mike Guy realized that the host need not store a table of passwords to authenticate users. Instead they store a table containing the results of one-way function on the passwords. This is how it works.

1. Alice sends her password to the host.
2. The host calculates the result of the one - way function on the password.
3. The host compares this value with the value previously stored.

Since the host does not store a list of passwords, threat of someone breaking into the system and stealing every one's passwords is nulled. The host only stores the results of one - way functions. Since one - way functions cannot be reversed, its very difficult for some one to get the passwords from this table.

This approach is however vulnerable to the Dictionary Attack. Mallory can make up a file of commonly used passwords. Then she could perform the one - way function on each of these passwords and generate a file containing the encrypted passwords. If she could steal the encrypted password from the host and compare the files, she could find out a few passwords. This attack could be thwarted by using a salt value. The host would pad the password supplied by the user with a random value of a certain length. Then it would perform the one-way function on this padded password. The result of the one way function and the salt could be stored in a file on the host. When a user

wants to log on the host simply pads the supplied password with the corresponding salt value, performs the one way function and then compares the results. This creates a lot of problems for Mallory has to do a trial encryption of each password every time she wants to break another person's password.

2.1.3.2.2 SKEY

This protocol also works on one way functions. To start up Alice enters a random number R . The host calculates f(R), f(f(R)), f(f(f(R))) for a fixed number of times say n. Then the host prints these values x[1]...x[n], where x[1] = f(R), x[2] = f(f(R)) and so on.Alice keeps them safely. The host also calculates x[n+1]. When Alice logs on next, she presents x[n] to the host. The host calculates f(x[n]) and compares it with x[n+1]. If they match Alice is logged in and the host replaces x[n+1] with x[n]. Alice deletes x[n] from her list and uses x[n-1] to logon the next time.

2.1.3.2.3 Authentication Using Public Key Cryptography

Both of the authentication protocol discussed above suffer from the same weakness. The weakness is that the password is sent out in the open. If the host is several networks away, Mallory could operate any one of the intermediate machines and sniff the password from the network. To avoid this problem the following protocol could be used

1. The host sends Alice a random string.
2. Alice encrypts the random string with her private key and sends it to the host.
3. The host decrypts the message received with Alice's public key.
4. The host compares the decrypted data with the original random string. If they match Alice is authenticated and Alice is logged in.

The above protocol suffers from the fundamental risk that Alice is using her private key to sign some unknown random string sent to her by some one else. The above random string could be "I Alice agree to pay Mallory 100000000$ ". Once Alice signs it, she cannot go back and is trapped. Thus the above protocol needs to be enhanced such that Alice does not sign some unknown random string sent by some one else.

2.1.3.2.4 SKID3

This protocols uses a secret key K. It can be used for mutual authentication. The protocols assume that both Alice and Bob know K.

1. Alice chooses a random number R_A. She sends it to Bob.
2. Bob also chooses a random number R_B. Then he calculates $H_k(R_A, R_B, B)$. H_k is the MAC. B is Bob's name. He sends Alice $R_B, H_k(R_A, R_B, B)$.
3. Alice computes $H_k(R_A, R_B, B)$ and compares it with the message she received from Bob. If they match Bob is authenticated.
4. Alice sends Bob $H_k(R_B, A)$
5. Bob computes $H_k(R_B, A)$. If it matches with what he received from Alice, then Alice is authenticated.

2.1.3.3 Authentication and Key Exchange

These protocols attempt to solve the problem of exchanging a session key and assuring the two ends on the network that they are indeed talking to each other and not Mallory.

2.1.3.3.1 Wide-Mouth Frog

Its one of the simplest protocol and it depends heavily on a trusted server (Trent) . The protocol assumes that both Alice and Bob share a secret key with Bob and that

Alice is competent enough to generate a random session key. The session key which Alice and Bob share with Trent is not used to encrypt actual messages going from Alice to Bob and vice versa, but to establish a session key.

1. Alice forms a string by concatenating a timestamp, Bob's name and a random session key. She encrypts this whole string with the key she shares with Trent. She sends this encrypted string to Trent along with her name.
2. Trent receives this message from Alice. Decrypts it. Retrieves the random session key. Forms a string by concatenating a new timestamp , Alice's name and the random session key. Trent encrypts this string with the key he shares with Bob. Trent sends the message to Bob. On receiving the message, Bob decrypts it with the key he shares with Trent. Thus he has the random session key which Alice generated. This can be used to encrypt communications between Alice and Bob.

Mallory can be in the middle and intercept the message sent by Alice to Trent. But she cannot do much, as she does not know the secret key shared between Alice and Trent. Neither can she read the session key, nor can she act as Alice.

2.1.3.3.2 Yahalom

This protocol again assumes that both Alice and Trent share secret keys with Trent a trusted server.

1. Alice generates a message $A,R_{A.}$ Where A is Alice's name and R_A is a random number generated by Alice. Alice sends this message to Bob.
2. Bob generates the following message B, $E_B(A,R_{A,}R_B)$ where B is Bob's name, R_B is a random number generated by Bob. E_B is the encryption key Bob shares with Trent. Bob sends this message to Trent.
3. Trent generates $E_A(B,K,R_{A,}R_B)$ and $E_B(A,K)$ where K is a random session key generated by Trent. E_A is the secret key it shares with Alice. Trent sends both these messages to Alice.

4. Alice decrypts the first part using her session key. She verifies R_A to be the random number she had generated. She also extracts K. She sends Bob two messages. $E_B(A,K)$ and $E_k(R_B)$.
5. Bob decrypts the first message and extracts K. He uses that to retrieve the random number from the second message and verifies it against the random number he had generated.

2.1.3.3.3 Needham-Schroeder

This protocol also uses symmetric key cryptography.

1. Alice sends to Trent A,B,R_A where A is Alice's name, B is Bob's name and R_A is a random number generated by A.
2. Trent generates the message $E_A(R_A,B,K,E_B(K,A))$. He sends this to Alice. Alice decrypts the message using her secret key. She verifies R_A. She also retrieves K. Then she sends Bob $E_B(K,A)$.
3. Bob decrypts the message , retrieves K. He then generates $E_k(R_B)$ and sends it to Alice.
4. Alice generates $E_k(R_B-1)$ and sends it to Bob.
5. Bob verifies R_B

2.1.3.3.4 Kerberos

Kerberos is a variant of the previous protocol. This protocol also assumes that Alice and Bob , both share a secret key with Trent.

1. Alice sends the following message to Trent . A,B where A is Alice's name and B is Bob's name.
2. Trent generates the following messages $E_A(T,L,K,B)$ and $E_B(T,L,K,A)$ where T is a timestamp, L is a lifetime and K is a random session key. Trent sends both these messages to Alice.

3. Alice decrypts the first message and retrieves K. Then she sends Bob $E_k(A,T)$ and $E_B(T,L,K,A)$.
4. Bob retrieves K and sends to Alice $E_k(T+1)$

This protocol assumes that everyone's clock is synchronized to Trent's.

2.1.3.3.5 A Brief Description of The SSL Protocol

SSL provides secure transmission of data at the network layer. It runs over TCP/IP and below protocols like HTTP, Telnet, FTP etc. Thus all of the above applications can use SSL to enhance security. It also allows a SSL enabled server to authenticate to a SSL enabled client and vice versa.

The SSL protocol includes two layers. 1) The SSL record protocol and 2) The SSL handshake. The former defines the format used to transmit data. The latter uses the former to exchange a sequence of messages between the server and the client to achieve the following

1. Authenticate the server to the client.
2. Allow the client and the server to negotiate a set of cryptographic algorithms.
3. Optionally authenticate the client to the server.
4. Use public key cryptography to agree upon a shared secret key.
5. Establish a secure , encrypted connection.

The SSL Handshake :

The SSL protocol uses a hybrid crypto system. It exploits the fact that symmetric key algorithms are much faster for encrypting bulk data, yet public key algorithms provide stronger authentication. A SSL session always starts with a SSL handshake which allows the server to authenticate itself to the client using public key cryptography and then allows the client and the server to mutually agree upon a symmetric key used for

encryption and decryption of messages. The SSL handshake consists of the following steps:

1. The client sends to the server the client side SSL version, cipher settings, randomly generated data and other information the server might need to set up a SSL connection.
2. The server sends back to the client, the server side SSL version, cipher settings, randomly generated data and other information the client may need to set up a SSL connection. The server also sends its own certificate and if client authentication is enabled the server requests for the client certificate.
3. The client uses the information sent by the server to authenticate the server. Server authentication has been discussed below. If the server did not fulfill the authentication steps, the user is informed.
4. The client generates a premaster secret key, encrypts it with the server public key and sends it to the server.
5. If the server requests client authentication the client also signs some data, unique to the session with its private key and attaches it along with its certificate to the above message.
6. If the server has requested client authentication , the server tries to authenticate the client as discussed below. If the authentication fails, the session is terminated. Otherwise the server decrypts the encrypted premaster secret and generates a master key from it. The client also does the same and generates a master key.
7. The client notifies the server that all future messages from the client would be encrypted with the master key and then it sends a separate encrypted message notifying the server that the client portion of the SSL handshake is over.
8. The server does the same.
9. The SSL handshake concludes.

Server Authentication

In order for server to authenticate itself to a SSL enabled client , the server sends its certificate to the client as described in step 2. The client goes through the following steps in order to authenticate the server.

1. It checks for the validity period. That is the current date should be within the certificates validity period. If that’s not true the client stops, else it proceeds to the next step.
2. A client maintains a list of trusted CA's. If the CA specified in the server certificate is present in the list of trusted CA's for the client , the client goes to the next step.
3. Next the client verifies the CA's digital signature present in the server certificate. It uses the public key present in the CA's certificate to do the above. If the signature cannot be validated, it means either the private key used to sign the certificate did not correspond to the public key in the CA certificate, or else the certificate was tampered. In both these cases the authentication fails.
4. Next the client checks if the server is actually located at the network address as specified in the server certificate. This step is important to protect against man-in-the-middle attack. If the network address matches with the address specified in the server certificate the client goes to the next step. If any of the above steps fail, the client browser informs the user that the authentication failed and that it cannot establish a encrypted and authenticated connection with the server.

Client Authentication

SSL enabled servers can be configured to require client authentication. When configured this way, the client sends the server its certificate along with a signed message which is signed by the private key corresponding to the certificate. The server uses this signed data to verify the public key in the certificate. SSL requires the client to digitally sign a piece of data generated randomly during the handshake

and known only to the server and the client. To authenticate a client the server goes through the following steps :

1. The server verifies the signature generated by the client with the public key present in the client certificate. If the verification succeeds, the private key used to sign the message matches the public key in the certificate.
2. The server checks to make sure that the current time and data is within the certifcate's validity period.
3. The server checks to see whether the issuing CA for the client certificate , is in the list of trusted CA's maintained by the server.
4. The server uses the public key from the issuing CA's certificate, to verify the CA's digital signature on the client certificate. If this succeeds the server goes on to step 6 of the protocol handshake.

2.2 Smart Cards

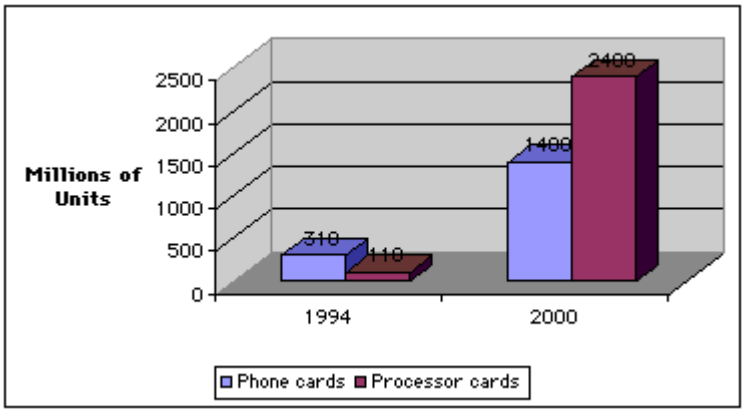
A smart card is just like a credit-card ,but with an embedded integrated circuit chip that makes it "smart". This combination of a convenient plastic card and a microprocessor allows an immense amount of information to be stored, accessed and processed either online or offline. Present day smart cards in commercial use can store up to 16K of application data but cards with much higher memory exists . The information or application stored in the IC chip is transferred through a communication interface. This communication interface is based on a standard ,half duplex, ISO T=0 protocol. Depending on the type of the embedded chip, smart cards can be either memory cards or processor cards.

Memory Cards These cards are not "smart enough". Though they have a processor. This processor is simply used to store information. They can store a huge amount of information.

Processor Cards These cards have a full fledged processor on board. Depending on the degree of sophistication these cards can perform file management, access control mechanisms for these files, perform encryption and decryption, generate keys etc. Older cards had support for encryption algorithms like Triple DES and performed internal and external authentication. Newer cards like cryptoflex cards have on board RSA key generation, digital signature generation and verification mechanisms. Most stored-value cards integrated with identification, security and information purposes are processor cards. Only processor cards are truly smart enough to offer the flexibility and multifunctionality desired in the networked economy.

The following figure shows the expected growth of smart cards :

Figure 7. Smart Card Usage



The Smart Card (ICC) is an intrinsically secure computing platform ideally suited to providing enhanced security and privacy functionality for applications running within general purpose computing environments such as the personal computer (PC). Due to support for access control mechanisms Smart cards are capable of providing secure storage facilities for sensitive information such as:

- Private keys.
- Account numbers.
- Passwords.
- Medical information.

Since the processing provided by smart cards are isolated , it provides a secure environment for carrying out sensitive computations unlike a PC where it is at potential risk from hostile code (viruses, Trojan horses, and so on). This becomes critically important for certain operations such as:

- Generation of digital signatures, using private keys, for personal identification.
- Network authentication based on stored secrets.
- Maintenance of electronic representations of value (that is, prepaid purchase credits).

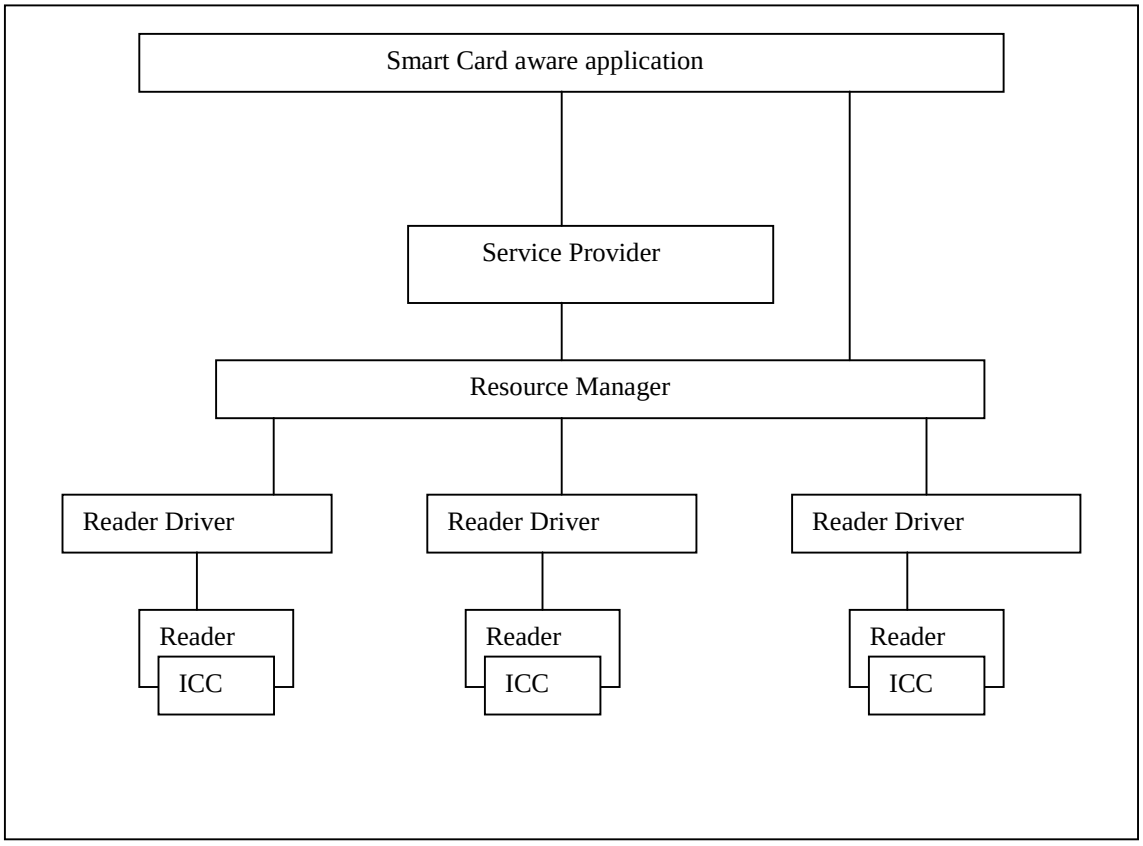
Until recent years the growth in the use of smart cards was severely hampered by a lack of interoperability at several layers There was no standard for interfacing PC's with the readers. This meant that an application had to be written for a specific reader. This lead to applications which would only work with the specified reader for which it was written for. Secondly there was no high level abstraction for exploiting the common functionality's of the ICC itself. This too led to a lack in interoperability as the applications were written to a specific smart card operating system instruction set. Thirdly standards for effective sharing of the smart card by more than one applications were not present.

Thus the PC/SC standards were developed. These standards focus on the above issues.

The architecture defined by this specification, in terms of software and hardware components, is shown in the next figure.

Figure 8. PC/SC Architecture

.



2.2.1 Integrated Circuit Card (ICC)

This specification deals with contact-type ICCs(smart cards) as specified by ISO/IEC 7816. Electrical connection is made between a reader (IFD) and a smart card (ICC) through electrical contacts connected to various pins on the microprocessor .These electrical connections are responsible for powering the card and also for the transfer of binary information between the ICC and the IFD.

An ICC compliant with this interoperability specification will conform physically and electrically to the ISO 7816-1, 7816-2, and 7816-3 standards. In addition, an ICC that is compliant with the ISO 7816-10 draft specification for synchronous cards can be supported by this specification. These standards provide a detailed definition of the physical form factor and the electrical characteristics of a compliant ICC.

2.2.2 Interface Device (IFD)

The IFD or the smart card reader is responsible for setting up communication between the PC and the ICC. As discussed above the IFD sets up electrical connection through electrical contacts connected to various pins on the microprocessor . The IFD is responsible for supplying DC power to the ICC through these connections.. Also through these electrical connections, the IFD provides a clock signal, which is used to step the program counter of the microprocessor, as well as an I/O line through which digital information may be passed between the IFD and the ICC.

An IFD is free to use various modes of physical connection with the PC like the keyboard port, a serial line port, or a PC Card (PCMCIA) port. In the future, Universal Serial Bus (USB) devices will likely become common.

The IFD should conform to the ISO 7816-1, 7816-2, and 7816-3 standards. In addition, an IFD may support the ISO 7816-10-draft specification for synchronous cards. IFD implementations may vary. The simplest IFD may just provide basic bit transfer mechanisms and power up capabilities. On the other hand a sophisticated IFD could implement a portion of the data link layer on its side thus reducing the work of the IFD handler.

2.2.3 Interface Device Handler (IFD Handler)

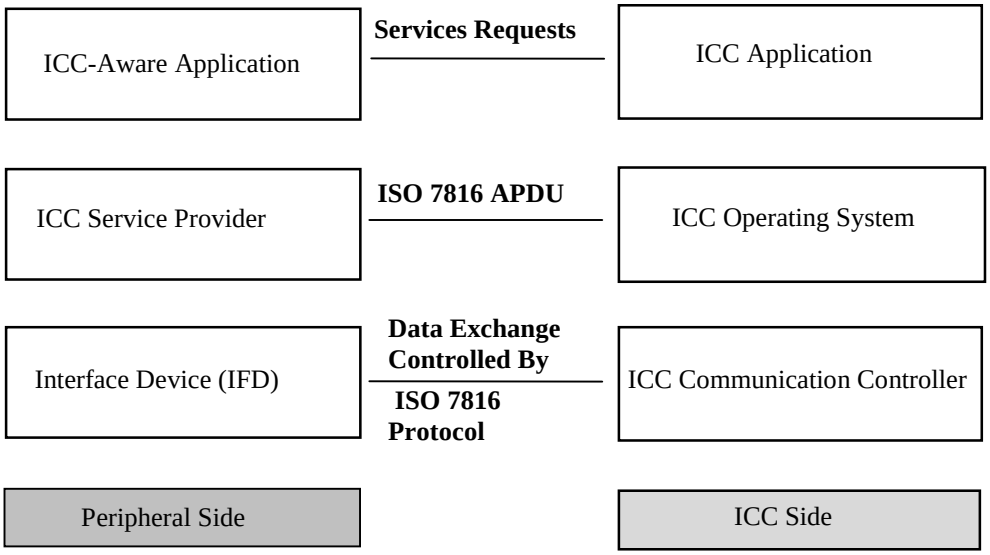
The IFD handler implements the IFD Handler interface as defined by the PC/SC standards. It provides a mapping between this higher level interface and native IFD calls. This software is usually a lower level software and provides interfaces for reading and writing data to the reader. IFD Handler is the terminus (on the PC side) of the ISO 7816-3 defined ICC communication protocols (T=0, T=1), and the synchronous protocol specified by the ISO 7816-10 draft specification. At the IFD

Handler API, all differences between ICCs based on ISO protocol handling, whether synchronous or asynchronous, are hidden.

2.2.4 ICC Resource Manager

This component is responsible for managing ICC-relevant resources within the system and for supporting controlled access to IFDs and, through them, individual ICCs.This component must be present and would be provided by the operating system vendor. There should be only a single ICC Resource Manager within a given system. The major responsibilities of the resource manager are keeping a track of the various IFD's installed on a system, tracking the different ICC types along with their associated service providers, allocation of IFD's and ICC's across multiple applications either through exclusive or shared mode of operations. It also enforces transaction primitives for ICC services. This is extremely important, as current ICCs often require execution of multiple commands to complete a single function. Transactions allow multiple commands to be executed without interruption, ensuring that intermediate state information is not corrupted.

Figure 9. The communication model for PC/SC is shown below



3. Design

3.1 Introduction

The goal of this chapter is to explain and justify the needs of the different components required to support security in a telemedicine environment. It also talks about the their interdependence and how they fit together in the overall picture.

3.2 Design Requirements

Enhancing the security of a telemedicine based application was one of the fundamental design objectives of this exercise. For the security features to be truly useful and scaleable it should guarantee the health care professional secure access to the telemedicine application independent of the machine he uses to logon to the system. Smart cards which was already being used in the system provided an excellent solution to the above requirement. The design exploits the fact that the smart card could be used as a secure and mobile token to carry around security information essential for making the security system useful from any place. Smart cards are inherently mobile as they can fit into a wallet just like a credit card. They are inherently secure as the smart card operating system supports authenticated file access through pin verification mechanisms.

3.2.1 Choice of Security System

The hybrid key crypto system was used as necessary infrastructure was already in place and it was easier to make the system interoperable. Public key crypto system was not chosen because of the following reasons :

1. Public key algorithms are slow. Symmetric algorithms are generally 1000 times faster than public key algorithms.
2. Public key crypto systems are vulnerable to chosen - plaintext attacks. This is because if the possible values of plaintext messages are limited, the cryptanalyst can try to encrypt all the possible messages, compare the result with the encrypted message and hence can get to know the plaintext message.

In a hybrid key crypto system , public key cryptography is used to distribute the session keys and symmetric key cryptography is used to secure message traffic. The key advantage of this mechanism, is that its much faster. Moreover the existing certifying authority structure could be exploited to issue secure identification tokens to the users of the secure telemedicine system.

3.2.2 Smart Cards

Utilization of the smart card to be the primary and only carrier of security information from place to place in the secure telemedicine application in turn demanded the following functionality's.

1. Identification of the card holder to the card.
2. Identification of the card holder's profession.
3. Guarantee the security of the information stored on the card.
4. Generation of card holder's signature on the smart card.
5. Mutual authentication between two end points using keys stored on the smart card.
6. Secure transmission of sensitive data over the network.
7. Secure logon to a computer using a smart card.
8. Authentication to a secure website using keys stored on the smart card.
9. Seamless integration of the above with industry standard technologies like Internet Explorer 5.0 and WinNT 4.0 OS.

Some of the above features like points 1,2,3,6 have already been taken care of by previous work. Hence the design focus was to integrate new functionality's like mutual authentication , signature generation , client authentication using SSL, and secure logon mechanisms to the already existing functionality. The design and implementation of these modules were essential as most of the products available in the market supporting similar functionality could not be integrated into the existing smart card based secure telemedicince application being developed at Concurrent Engineering Research Center.

The above requirements require the following basic support from the smart card.

1. Ability to protect data on card using pins/keys.
2. Private key containers on the smart card.
3. Certificate containers on the card.
4. On board encryption/decryption/digital signature generation on the smart card.

Out of the above mentioned points only point 1 was supported by Schlumburger's Multiflex Card. Hence extra design considerations were necessary to support the rest. Transparent files were created on the card to store the private keys / certificates. The private keys could be protected by pins which were supported by the multiflex file access system. Thus private keys could only be used by someone who knew the pin and held the smart card at the same time.

3.3 System Overview

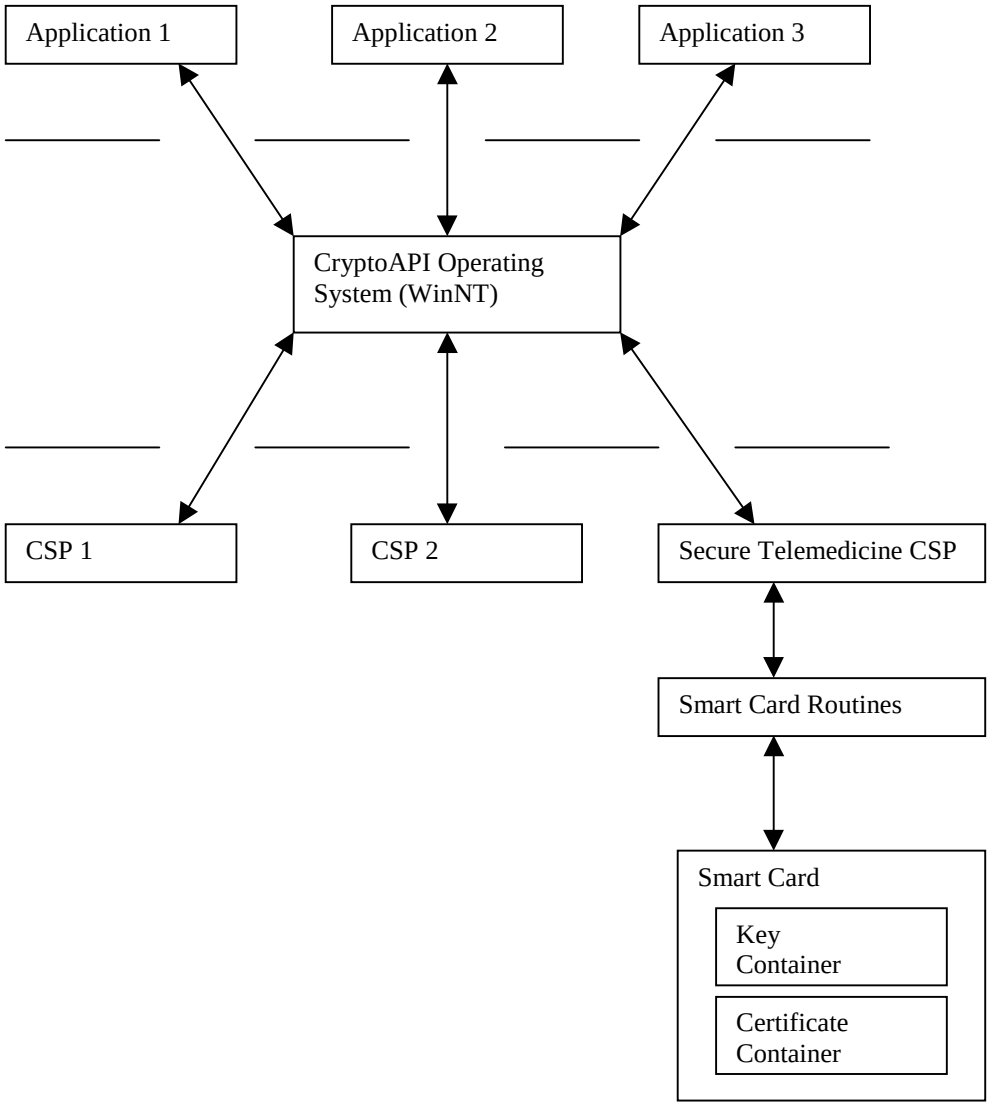
The system consists of the following major components to enhance security.

1. Cryptographic service provider
2. Mutual authentication module.
3. Graphical Identification and Authentication (GINA)
4. Smart Card Format

3.3.1 Cryptographic Service Provider

The Cryptographic service provider is a piece of software which provides implementations for various cryptographic computations and key management services required by an application program. In order to integrate various security routines with the existing software Microsoft CryptoAPI was used. MS CAPI is the standard programming interface provided by Microsoft for applications requiring cryptographic functionality's. MS CAPI gives the application the freedom to choose from the various CSP's available on the local machine. WinNT 4.0 comes with a set of standard CSP's.

Figure 10. Relationship between Applications/CryptoAPI/CSP's



The above figure gives a basic overview of where a CSP fits in the WinNT environment. An application is free to choose the CSP it wants to use. The above figure also justifies the implementation of a specialized CSP. The default microsoft CSP's do not store their keys and certificates on a smart card. Instead they use the registry to do so. That meant that certificates and keys are bound to a specific machine. This is highly undesirable in a secure telemedicine environment where mobility is one of the key issues. Smart cards which are mobile can provide a secure

storage for keys and certificates . Hence a new CSP was needed which would exploit the above mentioned advantages of a smart card.

The various reasons for implementing a smart card aware CSP are :

1. Need to store private/public keys and certificates on the smart card.
2. Need to able to generate the key-pair .
3. Need to able to use the key-pair and certificates on the smart card for carrying out authentication and other security mechanisms.
4. Need to be able to integrate the above functionality with the existing secure telemedicine system.

The last point was one of the most important reasons why a specialized CSP was implemented. There are a few smart card aware CSP's available in the market but none of them provided us the necessary support required for integrating them into the existing secure telemedicine application.

The smart card aware CSP developed, forms the core of all the security mechanisms in the secure telemedicine project. Some of the key points where it gets used are

1. Certificate request generation.

At this step the CSP is used to generate a RSA key pair, generate hashes and signatures required for presenting the request to the certificate server.

2. Certificate Storage

After the certificate request has been successfully processed by the certificate server the certificate is sent back to the client browser which uses the CSP to store the certificate onto the card.

3. SSL client authentication

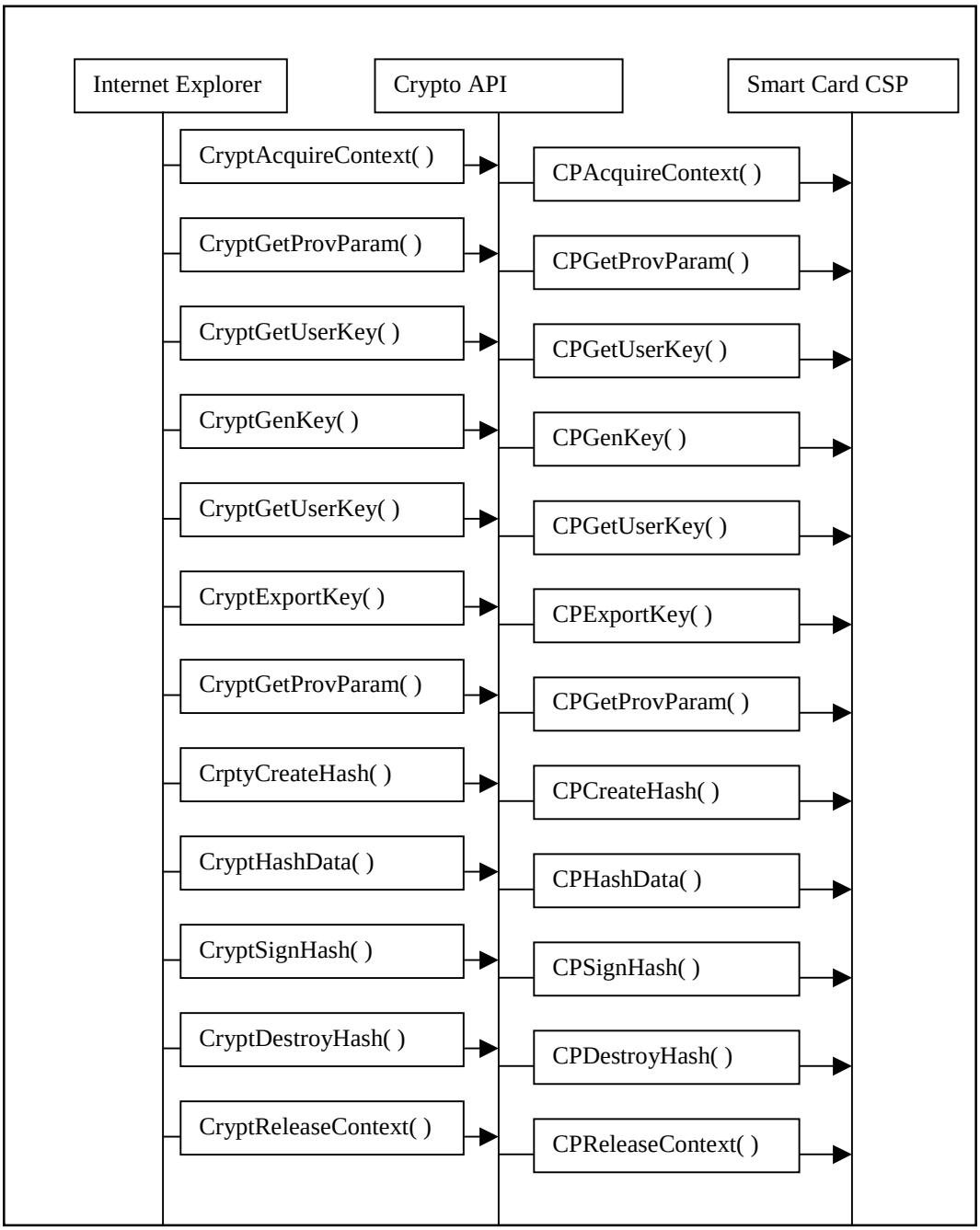
When a client browser wants to access a site requiring client authentication SSL requires signature generation which is done by the CSP using the private key from the card.

4. Mutual Authentication Process

Two doctors mutually authenticate each other through a handshake mechanism. All encryption, hashing and signature generation mechanism required for this is done by the CSP using the private key stored on the smart card.

3.3.1.1 Calling Sequences For CSP

Figure 11. Certificate request generation



1. CPAcquireContext(..new key set ..)

IE calls the CryptoAPI function CryptAcquireContext( ) . Crypto API loads up the smart card CSP DLL and calls the function CPAcquireContext( ). This function acquires a handle to the requested key container and returns the handle to CryptoAPI. This handle is used in the consequent calls to the CSP. If a new key set is requested a new container with the specified name is created.

2. CPGetProvParam(..dwParams = 4, dwFlag = 0..)

CPGetProvParam returns information about the CSP. IE calls this function in order to verify if its communicating with the right CSP. The above parameter combination return the CSP name.

3.CPGetUserKey(...keyspec = keyexchange ..)

This function returns a handle to one of the key pairs in the key container handle returned in CPAcquireContext( ). IE calls this function in order to check whether the key exchange key pair is present in the key container or not.

5.CPGenKey(...keyspec = keyexchange..)

Since the key container is new, the last function returns a FALSE. Hence IE calls this function to generate a pair of key exchange keys.

5. CPGetUserKey(...keyspec = keyexchange ..)

This time the function succeeds.

6. CPExportKey( )

IE calls this function in order to securely extract the key exchange public key from the key container. It does so because the public key is a part of the certificate request.

7. CPGetProvParams(dwParams = 1)

IE calls this function to enumerate the different algorithms supported by the CSP.

8. CPCreateHash( )

IE calls this function in order to initiate the hashing process. It returns a handle to the hash object.

9. CPHashData( )

IE calls this function in order to pass the data to be hashed. This data is usually the unsigned part of the certificate request.

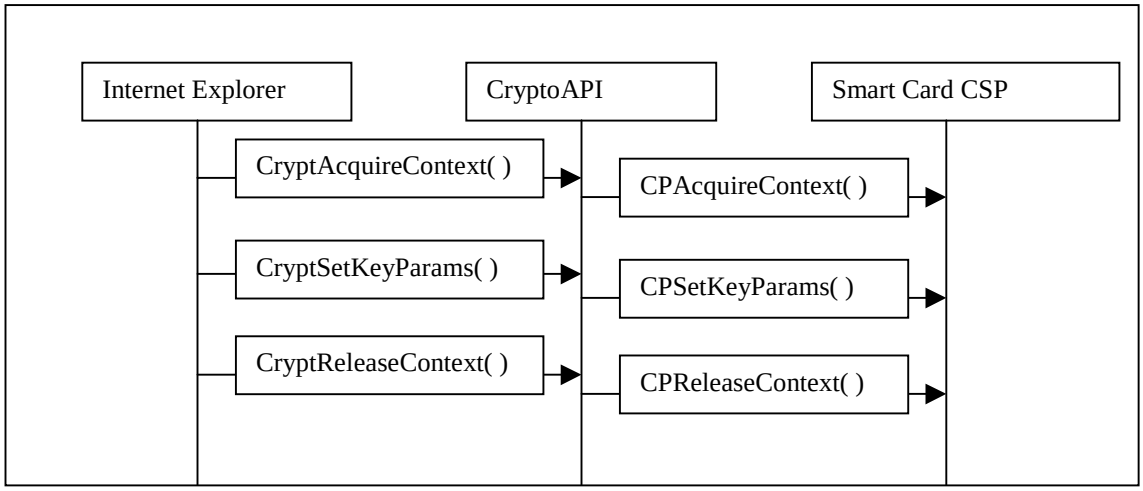
10. CPSignHash( )

This function is called by IE to retrieve the signature of the hashed data created in the previous step. This signed data is also added to the certificate request.

11. CPReleaseContext( )

This releases the handle to the key container in the CSP.

Figure 12. Certificate Storage



1. CPAcquireContext( )

This function is called by IE with the name of the new key container created during certificate request generation. It returns a handle to the key container.

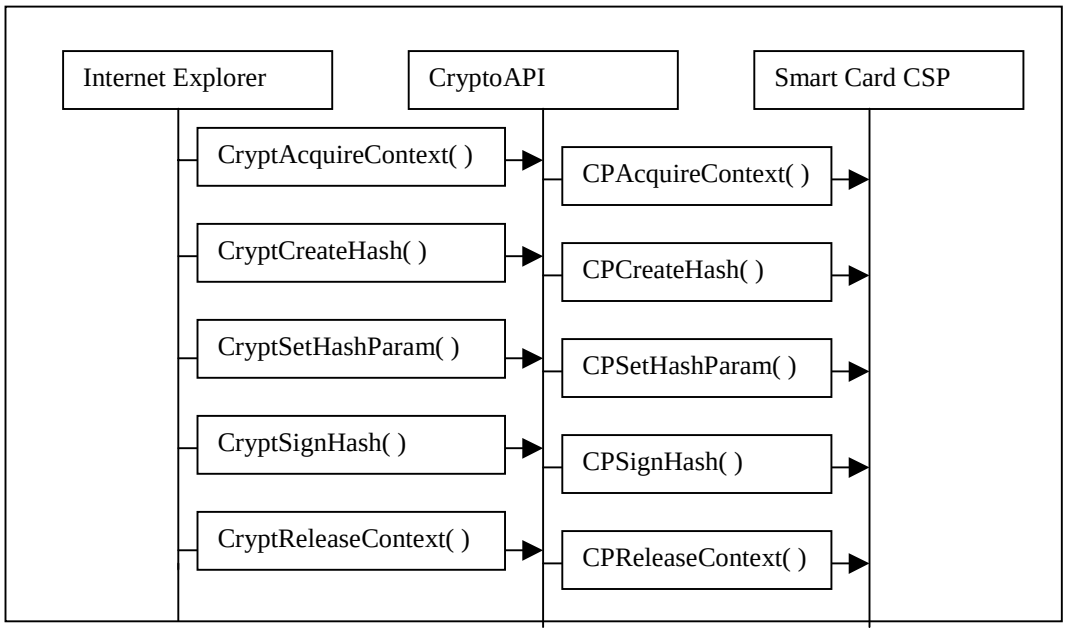
2. CPSetKeyParams(...KPCertificate..., Certificate Blob ...)

This function is called by IE to let the CSP store the certificate which could be a smart card.

3. CPReleaseContext( )

The handle to the key container is released.

Figure 13. SSL client authentication



1. CPAcquireContext( )

During client side SSL authentication , IE allows the user to choose the certificate to authenticate with. For each certificate WinNT maintains a context which among other details, contains information about the CSP to use, the key container within the CSP to use. Once the user selects a certificate IE calls CPAcquireContext( ) on the appropriate CSP with the appropriate container name. The CSP returns a handle to the key container.

2. CPCreateHash( )

Initiates the hash object.

3. CPSetHashParam( )

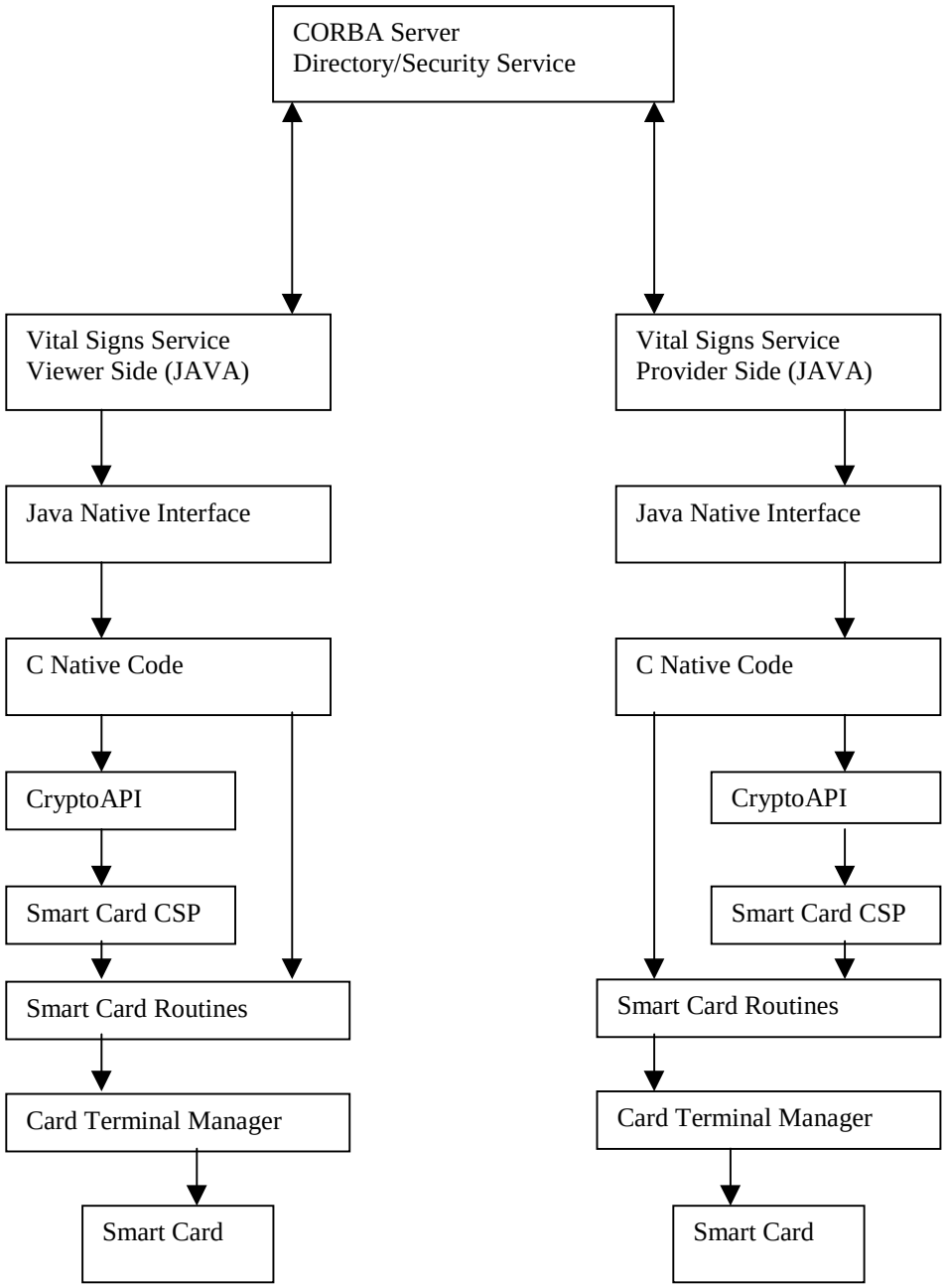
This function allows the caller to sign a piece of data without actually knowing the base data.IE calls this function to set the hash value directly. This hash consists of a concatenation of an MD5 hash and an SHA hash signed with an RSA or Diffie-Hellman private key. The hash is computed by the schannel.dll which does most of the work during SSL authentication.

4. CPSignHash( )

IE calls this function to get the above hash signed by the RSA private key stored on the smart card.

3.3.2 Mutual Authentication Of Two Entities Based On Their Public Key Certificates.

Figure 14. Mutual Authentication



The above diagram shows the various components implemented to support mutual authentication of two entities based on their private keys and public key certificates stored on the card.

To mutually authenticate two entities, both these entities take part in the following handshake. The following protocol assumes that there is no trusted repository of certificates.

3.3.2.1 Protocol

Nomenclature:

C_X : X's Certificate.

PuK_X : X's public key

E_X(M) : Encrypt message M with X's public key.

D_X(M) : Sign M using X's private key.

H(M) : Hash M.

T_X : X's Time Stamp.

A : viewer

B : provider (the source of information)

A To B : C_A

B To A : B verifies the certificate with the public key of the certifying authority. B extracts PuK_A from C_A. Generates a random no R_B and timestamp T_B Sends back to A

E_A(R_B+T_B) + C_B + D_B(H (E_A(R_B+T_B)))

A To B : A verifies the certificate with the public key of the certifying authority .A Retrieves R_B and T_B, Retrieves PuK_B from C_B , A verifies B's signature. B is authenticated. Generates a random no R_A and timestamp T_A. Sends back To B

E_B(R_A+R_B+T_A) + D_A(H(E_B(R_A + R_B + T_A)))

B : B verifies A's signature. Then verifies R_B . A is authenticated.

After all the steps of the above protocol have been successfully completed both entities are convinced that the the other entity is authentic, that is they are, who they claim to be.

Protection from man-in-the-middle attack :

Mallory the man-in-the-middle cannot do any harm. He could try to view the patient information by providing A's certificate , but that is of no use as he does not have the private key for A and the authentication would fail in step 4.

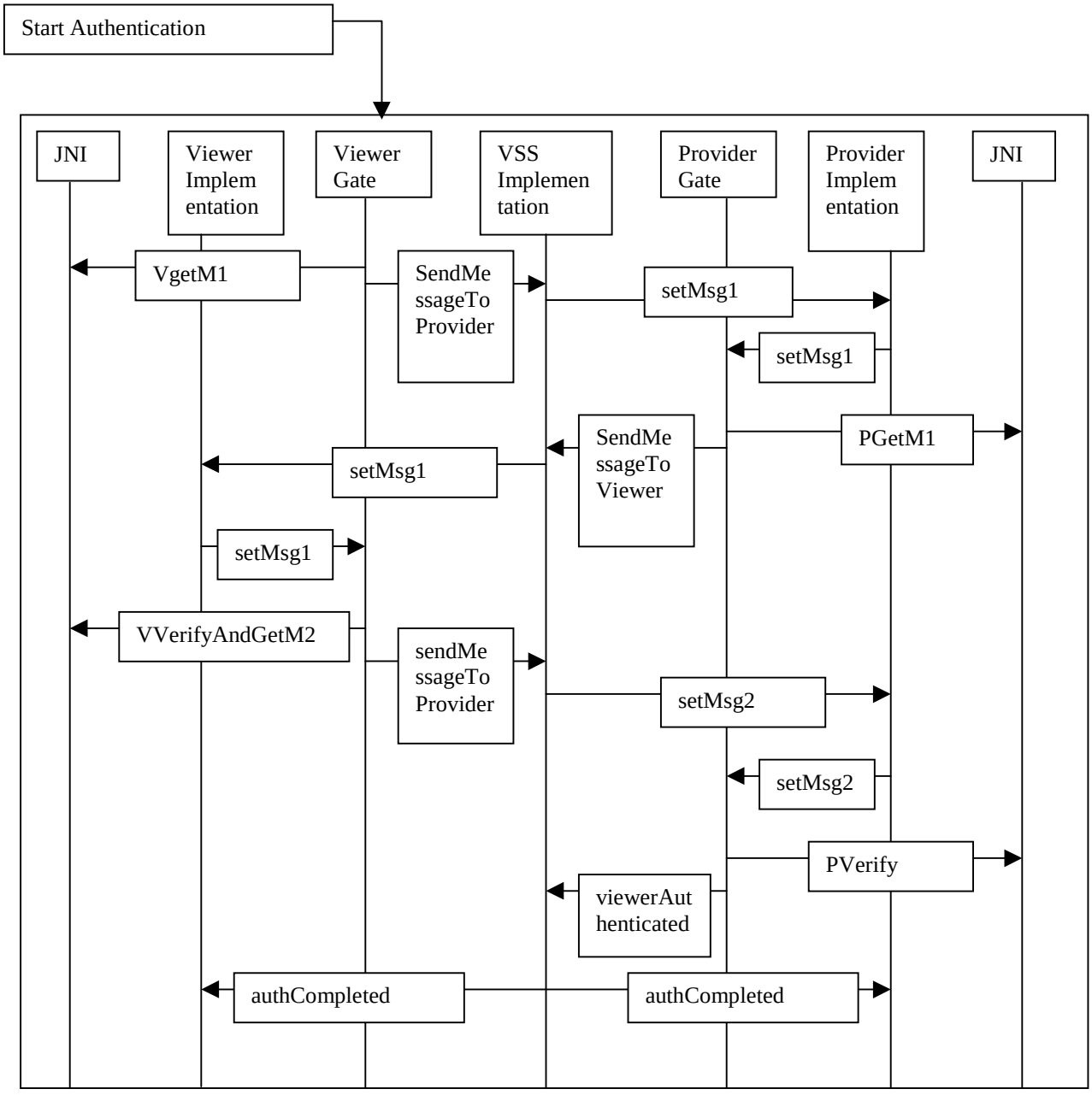
Protection from the replay attack :

Mallory could try to sniff the message from B to A at step 2, and later when A contacts B for information could pose as B and send the sniffed message. The signature would still match the public key in the certificate in the message. However the time stamp would have expired, and authentication would fail. However this protocol would fail to detect replay if the replay message is sent within the minimum interval. That is the time stamp in the replayed message is still fresh enough. In this case B could send wrong and malicious information about the patient to A. Replay attacks in which the viewer's identity is being compromised would fail automatically as Mallory would not be able to retrieve the random number and time stamp in step 3.

The strength of the protocol lies in the fact that it makes sure that the other entity has the private key for the certificate it is presenting, and that the certificate has been issued by trusted CA.

3.3.2.2 Calling Sequence

Figure 15. Calling Sequence For Mutual Authentication between Provider - Viewer



VGetM1 ( ): This function is implemented by the JNI interface. It retrieves the certificate stored on the viewer side physician card.

SendMessageToProvider (..PID..) : This function is implemented by the VSS implementation. It is called by the Viewer Gate to transfer a message to the provider side. The VSS implementation uses the PID to find a reference to the provider implementation stored in a directory. It uses this object reference to call the appropriate SetMsg ( ) on the provider implementation.

Provider Implementation : : SetMsg1 ( ): This function is called by the VSS implementation. Its used to pass the message in Step1 of the protocol to the provider side.

ProviderGate : : SetMsg1( ) : The provider implementation passes on the above message to the provider Gate by calling this method.

PgetM1 ( ) : This method is implemented by the JNI interface. It implements step 2 of the protocol and retrieves the message shown in step 2 of the protocol .It uses calls the smart card CSP functions through crypto API to generate the message.

SendMessageToViewer(..VID..) : This method is implemented by the VSS implementation. It uses the VID to retrieve the appropriate viewer implementation reference. It transfers the message in step 2 from the provider side to the viewer side.

ViewerImplementation : : SetMsg1 ( ) : SendMessageToViewer ( )calls this method on the viewer implementation to transfer the above message to the viewer side .

ViewerGate : : SetMsg1( ) : The viewer implementation passes on the above message to the Viewer Gate by calling this function.

VVerifyAndGetM2 ( ) : This function is implemented by the JNI. It implements step 3 of the protocol and retrieves the message shown in step 3 of the protocol .It uses calls the smart card CSP functions through crypto API to generate the message.

Provider Implementation : : SetMsg2 ( ): This function is called by the VSS implementation. Its used to pass the message in Step3 of the protocol to the provider side.

ProviderGate : : SetMsg2( ) : The provider implementation passes on the above message to the provider Gate by calling this method.

Pverify : This function is implemented by the JNI. It implements step 4 of the protocol .It uses calls the smart card CSP functions through crypto API to generate the message.

ViewerAuthenticated ( ) : This function is implemented by the VSS implementation. It is called by the provider side if Pverify( ) succeeds.

AuthCompleted( ) : This function is implemented by both the Provider and the Viewer implementations. It is used by the VSS server to convey the final state of the authentication process.

Following the conclusion of the above authentication steps. The viewer is allowed to view the vital signs of the patient on the provider side. This authentication mechanism can be used in other situations where a mutual authentication of a client and server side is required at the same time.

3.3.3 Graphical Identification and Authentication (GINA)

This module was developed in order to provide for secure logon mechanism on to a computer using a smart card. When a user boots up a computer running Microsoft WindowsNT , winlogon.exe is executed. This DLL takes care of the logon process. In turn winlogon.exe calls GINA to perform various duties. WinNT comes with the default MSGina.dll . It can be replaced by any other dll which wants to customize the logon process.

3.3.3.1 The reasons for implementing a smart card GINA are as follows :

1. It provides for a more secure logon process. Only someone with a smart card and the correct pin for the smart card can logon to a machine. In other words , stealing some users pin is not sufficient to logon as that user on to a computer.
2. It provides for logon to multiple accounts for a single user with a single pin. That means the user does not have to remember different passwords for different accounts. All he needs to remember is the pin.
3. It provides mechanisms for transparently creating a certificate context in the registry for the corresponding certificate on the smart card. This certificate context can be later on used for client authentication using SSL using the smart card CSP.
4. It provides mechanisms for supplying the user pin to programs requiring it for cryptographic calculations. The pin required to logon is the same as the pin required by the CSP for accessing the private key stored on the smart card. This mechanism activated by checking a check box on the unlock/logon screen. Once checked the GINA launches a program which uses Interprocess Communication to

transfer the pin to the CSP whenever it asks for it. Thus the user is spared the trouble of typing in the pin again and again during a cryptographic process.

3.3.3.2 Responsibilities of a GINA as specified by Microsoft.

A GINA .dll has the following responsibilities:

- **SAS Monitoring**

  GINA is responsible for monitoring the appropriate devices for a Secure Attention Sequence. The standard MSGina monitors for Ctrl + Alt + Del. The smart card aware GINA also monitors for smart card insertion and removal. For example the smart card aware GINA automatically prompts for screen lock on smart card removal from the reader, and automatically prompts for logon / unlock once it detects a smart card insertion into the reader.

- **SAS Processing**

  One of the reasons for modularizing Winlogon and making GINA replaceable is to provide alternative identification and authentication mechanisms. To do this, GINA presents all user interfaces resulting from the recognition of a SAS. When no user is logged on, GINA is responsible for collecting identification and authentication information. When a user is logged on, GINA is responsible for presenting all options or taking whatever actions are deemed appropriate. For example, in a system that includes a smart card, it may be appropriate to automatically lock the workstation if the user removes the smart card.

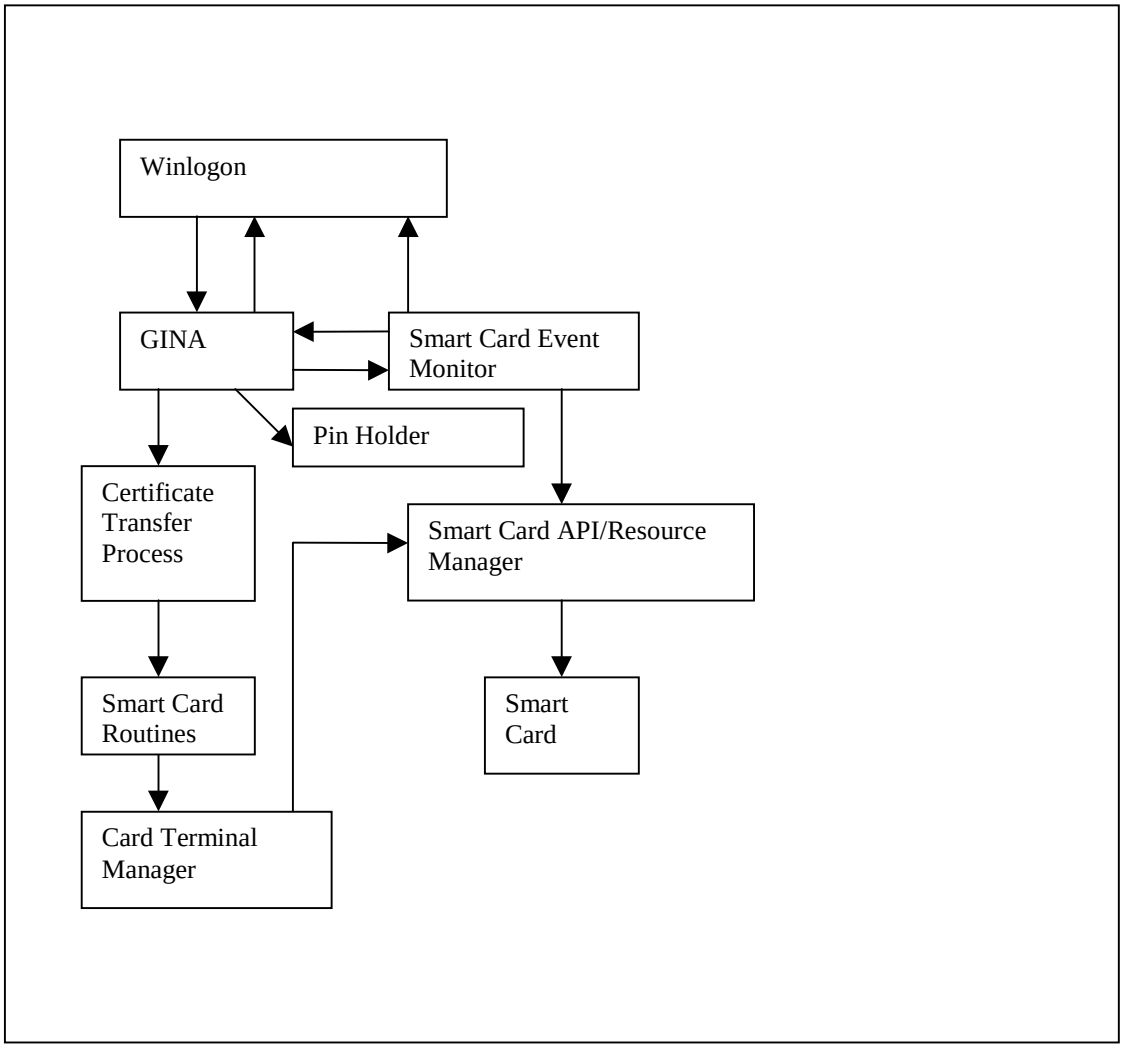
- **Shell Activation**

  When a user logs on, GINA is responsible for creating one or more initial processes for that user. One of the processes is the user shell. As part of shell activation, GINA must assign the newly logged-on user's token to the processes, so the initial process(es) should be created in a suspended state and

the token assigned before the process(es) can run. Winlogon provides a service to assist GINA in assigning the token.

3.3.3.3 GINA and related components for logon authentication.

Figure 16. GINA and related components for logon authentication

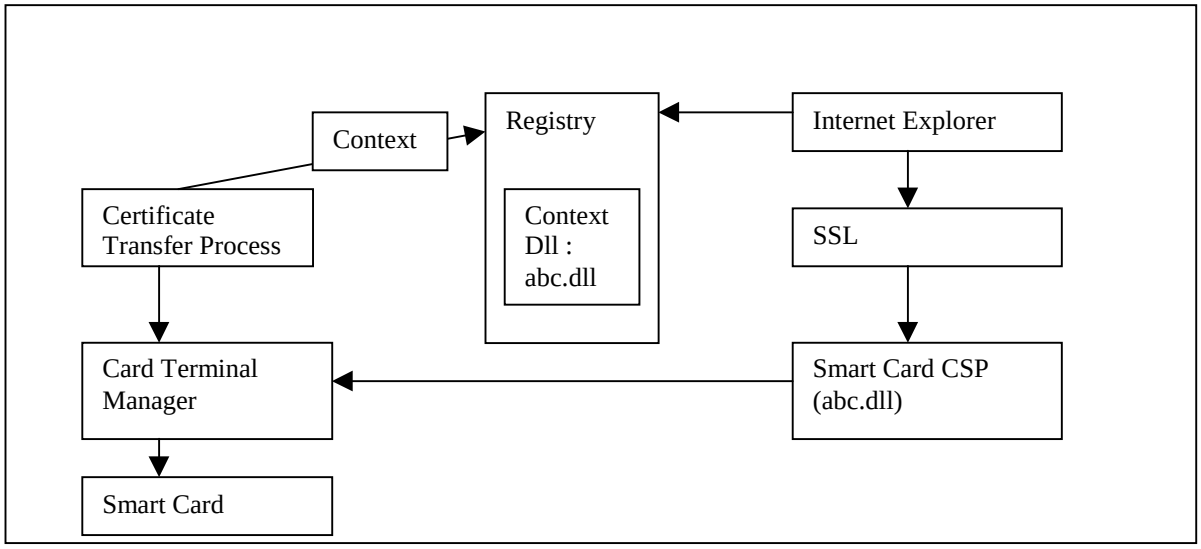


3.3.3.4 Certificate Transfer Process :

This process is launched by GINA when the user logs on and every time the user unlocks the workstation. The major responsibility of this process is to read the certificate present on the smart card, create a certificate context from it. Then it adds

this certificate context to the MY store in the registry. The context contains information about the certificate, the key usage, the raw certificate, the name of the Cryptographic service provider to load up when the certificate is in use etc. This process is important as it lets the physician use his certificate from a different computer than the one he used to obtain the certificate.This is because once the certificate context is created in the registry, IE adds it to the list of certificates available for the current user. The context also contains the name of the CSP dll to load up. Hence if the user chooses this newly added certificate to perform SSL client authentication, IE loads up the corresponding smart card CSP.

Figure 17. Certificate Transfer Process



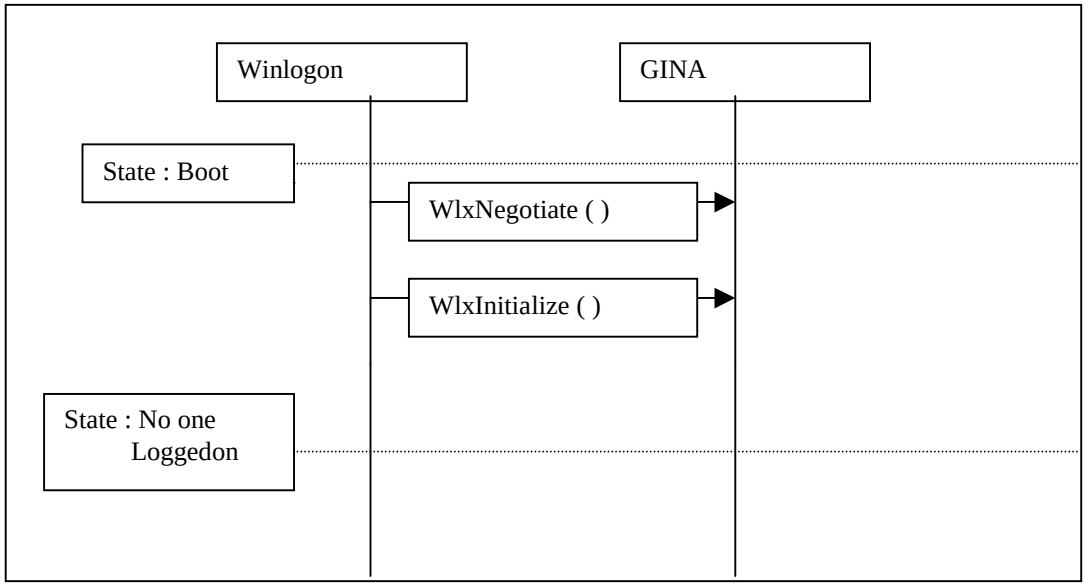
3.3.3.5 Pin Holder Process :

This process is a supplemental process which is launched by GINA when the user logs on or unlocks the computer. This process makes the use of a smart card CSP more convenient. The private keys stored on the smart card are protected by a pin, the same as the one used to logon or unlock the workstation. Hence whenever a CSP needs to use the private key, it prompts the user for a pin. A CSP could require the use of the private key quite a few times. This in turn would require the user to enter the pin

more than once within a short interval of time. To make it easier for the user, GINA launches a process called pin holder on the users request. This process stores the user pin. The CSP uses the pipe mechanism to extract the pin from this process, whenever needed. Thus the user is not prompted for pin entry each and every time, its needed by the CSP.

3.3.3.6 Interaction Between Winlogon and GINA

Figure 18. Winlogon - GINA interaction at Boot

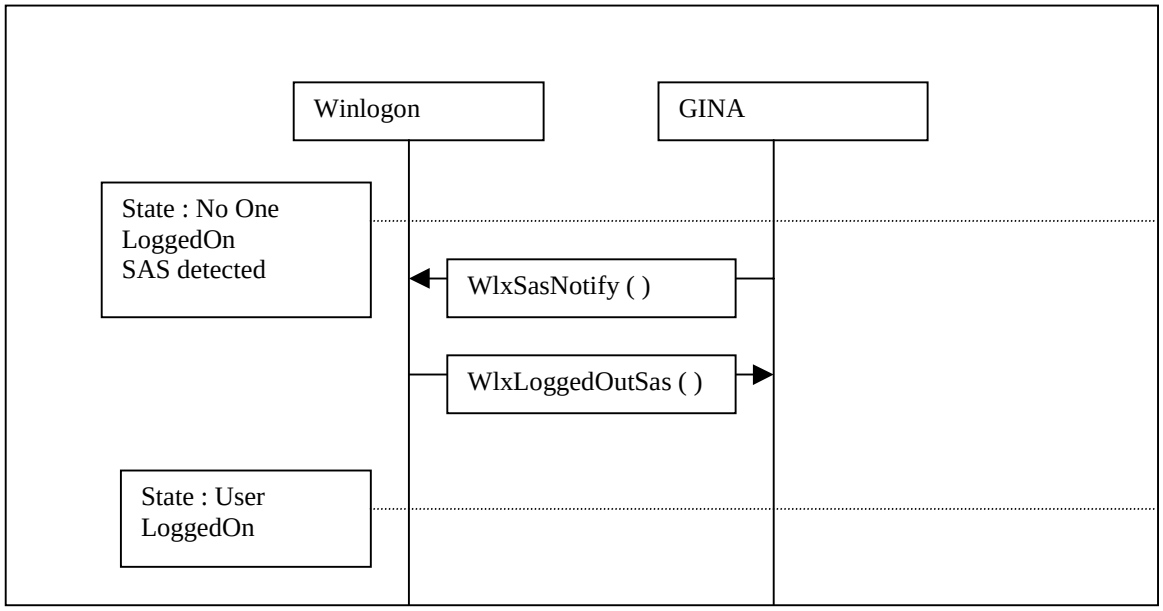


WlxNegotiate ( ) : This function is implemented by GINA. It allows GINA to verify whether it supports the version of Winlogon making the call.

WlxInitialize ( ) : This function is implemented by GINA. It allows GINA to receive a pointer to the function table supported by winlogon for performing various tasks. This function also creates the smart card state monitoring thread. This thread detects smart card insertion and removal, and notifies winlogon of the smart card SAS by calling

WlxNotifySas( ). It returns a handle to a context structure defined within GINA. This context is required for all consequent calls to GINA.

Figure 19. Winlogon - GINA interaction at Logon

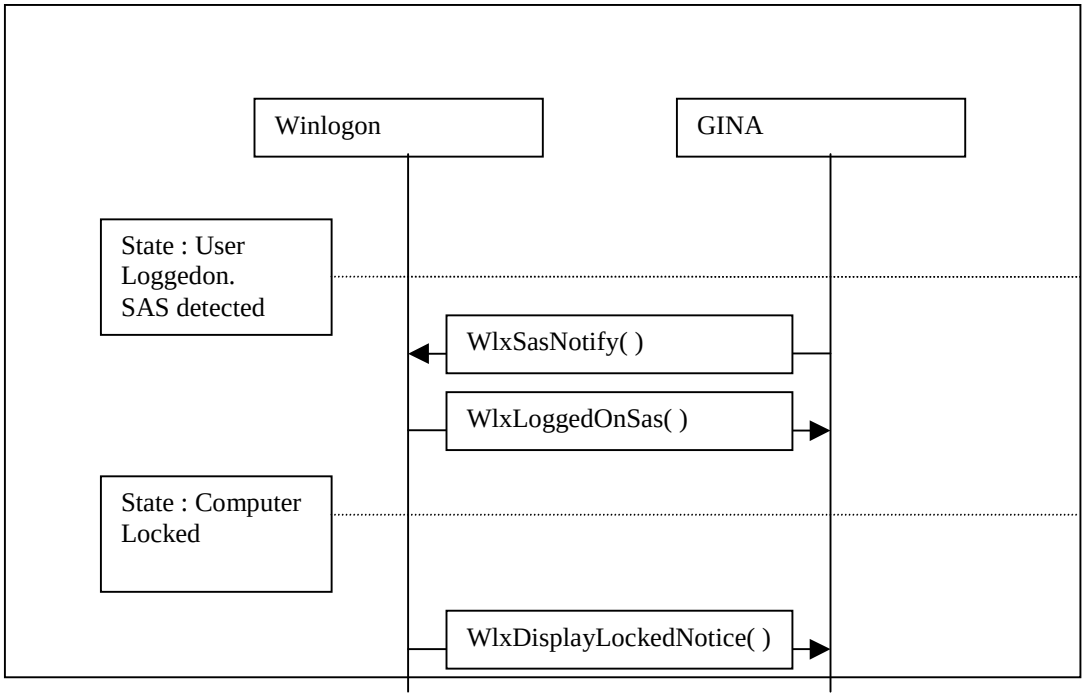


WlxSasNotify( ) : This function is called by GINA. Its implemented by winlogon.exe. Every time GINA detects a SAS , it lets winlogon know about it by calling this function.

WlxLoggedOutSas( ) : Winlogon delivers the SAS back to GINA by calling this function. GINA may configure winlogon to monitor for ctrl+alt+del. In that case GINA does not have to call WlxSasNotify( ) . Winlogn automatically detects ctrl+alt+del and calls GINA's WlxLoggedOutSas( ). This function has the responsibility for displaying the logon dialog box and other options. It captures the user input like name,domain and password, then tries to log the user on to the system by calling the appropriate WinAPI function. If the logon is successful, it tries to read the certificate of the smart card and create a context in the registry. This enables IE

to use the certificate on the smart card for future authentication purposes. This function also launches the pin holder process.

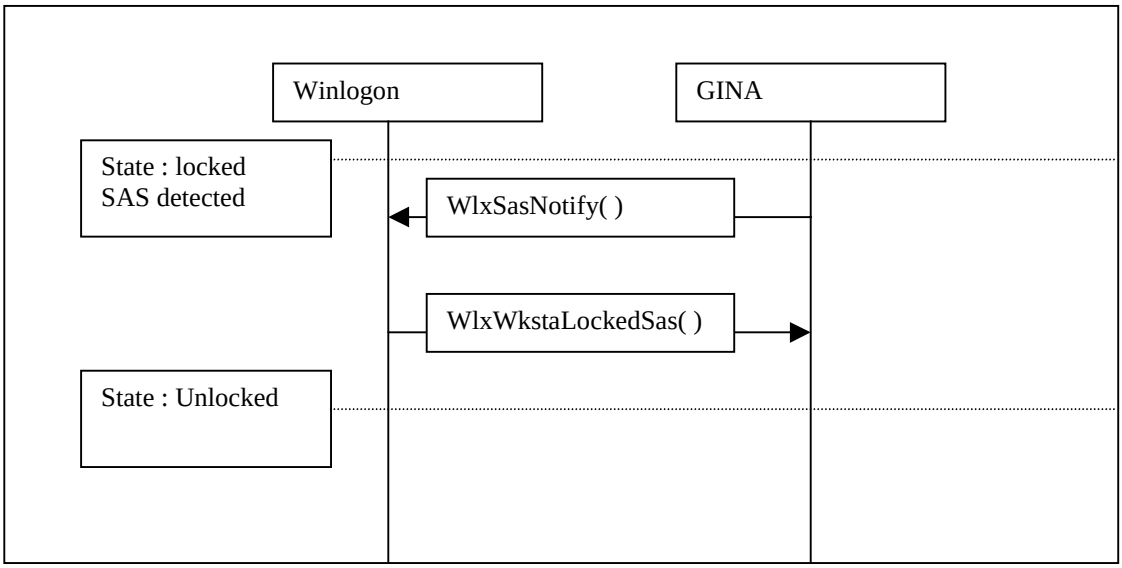
Figure 20. Winlogon - GINA interaction at Lock Workstation



WlxLoggedOnSas( ) : This function is implemented by the GINA. Winlogon calls this function when a SAS is detected while the user is logged on and the computer is not in a locked state. This function interacts with the user, providing user interfaces where necessary to let the user have an option of cancelling the lock mechanism, or going ahead with it. If the user chooses to lock the computer this function kills the pin holder process, and returns the appropriate value to winlogon.

WlxDisplayLockedNotice( ) : This function implemented by GINA is responsible for displaying the lock screen while the computer has been locked.

Figure 21. Winlogon - GINA interaction at Unlock Workstation.



WlxWkstaLockedSas( ) : Winlogon calls this GINA function when the workstation is locked, and it receives a SAS notification either through GINA or ctrl+alt+del. This function is responsible for interacting with the user and asking the user for the unlocking pin/password. If the unlock is successful, this function launches the certificate transfer process which would create a certificate context for the certificate stored on the smart card and store the context in the registry. This function also launches the pin holder process.

3.3.4 Smart Card Format.

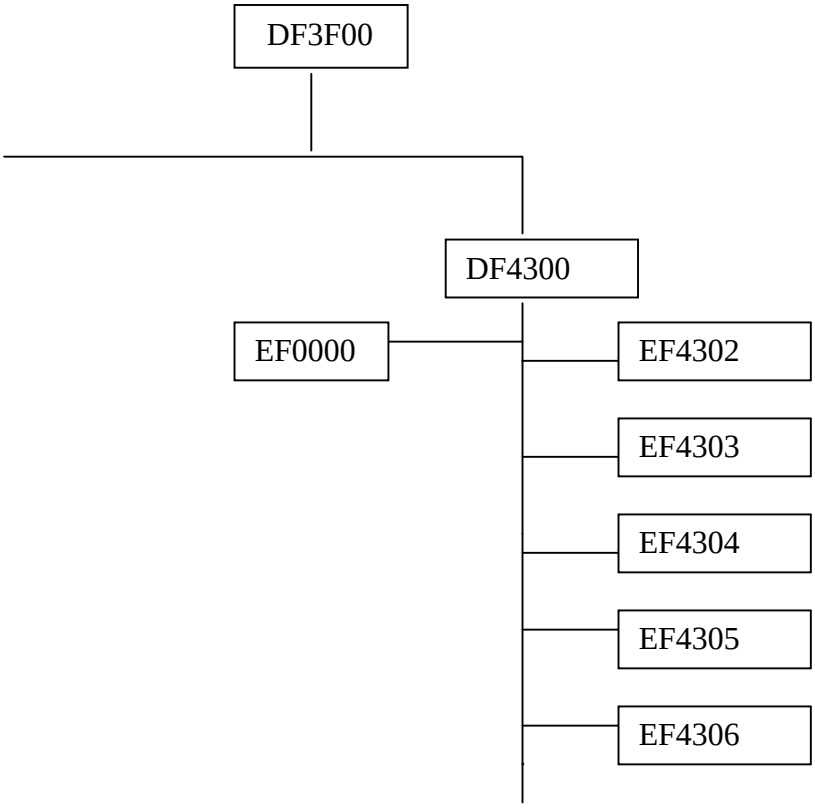
The smart card layout is the most fundamental aspect of the design.

Each and every module discussed so far understands the file layout and access conditions required in order to function properly. Originally the physician card consisted of files to sore physician information and files to store master key information in order to calculate patient card keys. Thus the card format had to be extended in order to allow for storage of certificate, RSA keys and logon information.

Necessary access conditions had to be defined in order to make the private key and logon information secure.

The following figure shows the extensions made to the health care professional card in order to support the security services required for a secure telemedicine application. An EF prefix before a file name specifies that it is an elementary file. Elementary files are like normal files. They are used for storing information. A DF prefix specifies that it is a dedicated file. Dedicated files are like directories.

Figure 22. Smart Card Format Extensions for supporting Security Services



DF4300 : This file is the directory which contains all of the files required for storing security information.

EF0000: This file contains the PIN

EF4302 : This file stores the private/public keys. It has space for storing 2 pairs of keys. One for key exchange and the other for digital signature. This file is read/write protected by the PIN.

EF4303 : This file stores the certificate required for client authentication.

EF4304 : This file stores the passwords for supporting logon mechanisms using smartcards. This file is read/write protected by the PIN.

EF4305 : This file stores the usernames for supporting logon mechanisms using smart cards.

EF4306 : This file stores the domain names for supporting logon mechanisms using smart cards.

.

4. Implementation

4.1 Introduction

This chapter briefly describes the various issues involved in the development of a smart card based security system for the secure telemedicine application. It justifies the use of one implementation technique over other implementation techniques.

4.2 Choice of Programming Language

The major part of the system was developed in C++. The components fully developed in C++ are the Cryptographic Service Provider , GINA and the smart card routines to extract certificates/keys/logon information from the card. The reason for using C++ were

1. The components depended heavily on the Card Terminal Manager developed previously by "Srivatsan Kannan" which was coded in C++. Thus writing the software in C++ avoided the overhead of developing interfaces like JNI/COM, which would have been required if the software was written in Java.
2. The support for developing these components on WinNT were much better in C++ than Java.

The higher level code for mutual authentication was written in Java. This is because the mutual authentication process was closely developed for a piece of software which had already been developed in Java by "Chaoxin Sima" . The mutual authentication software was just an add on which provided authentication between the provider and the viewer side. However this piece of software written in Java had to call Smart Card routines / Crypto API both of which had a C++ interface. Hence a JNI (Java

Native Interface) was provided which allowed functions implemented in C++ to be invoked from a Java Program.

4.3 Choice Of Smart Card and Smart Card Readers

Schlumburger's Multiflex card and Reflex72 readers were chosen for development of this system. Multiflex cards provided 8K of EEPROM for storing application data on the card. At the time, the project was started this card had the maximum memory. This card provided for on-board Triple-DES encryption. However functionality's required for public key cryptography was not supported on this card. Later Schlumburger released the Cryptoflex card which had on-board RSA key generation, signature mechanisms. However support for this card was not easily available. Java card which supported on-board applets in the form of cardlets was also released by Schlumburger. This card had 16K of memory. However storing cardlets on the card would reduce the memory available for applications drastically.

The original reader for this project was SCR60. However we changed over to Reflex 72 as it was PC/SC compliant. This meant that any application which was written on Microsofts Smart Card API could access the Smart Card reader. This also meant that if our application was written against Microsofts Smart Card API, it could access any reader which was PC/SC compliant. This made our software more interoperable as the software was no longer written for a particular reader driver but for a whole class of reader drivers (PC/SC reader drivers) .

4.4 Implementation of the Cryptographic Service Provider

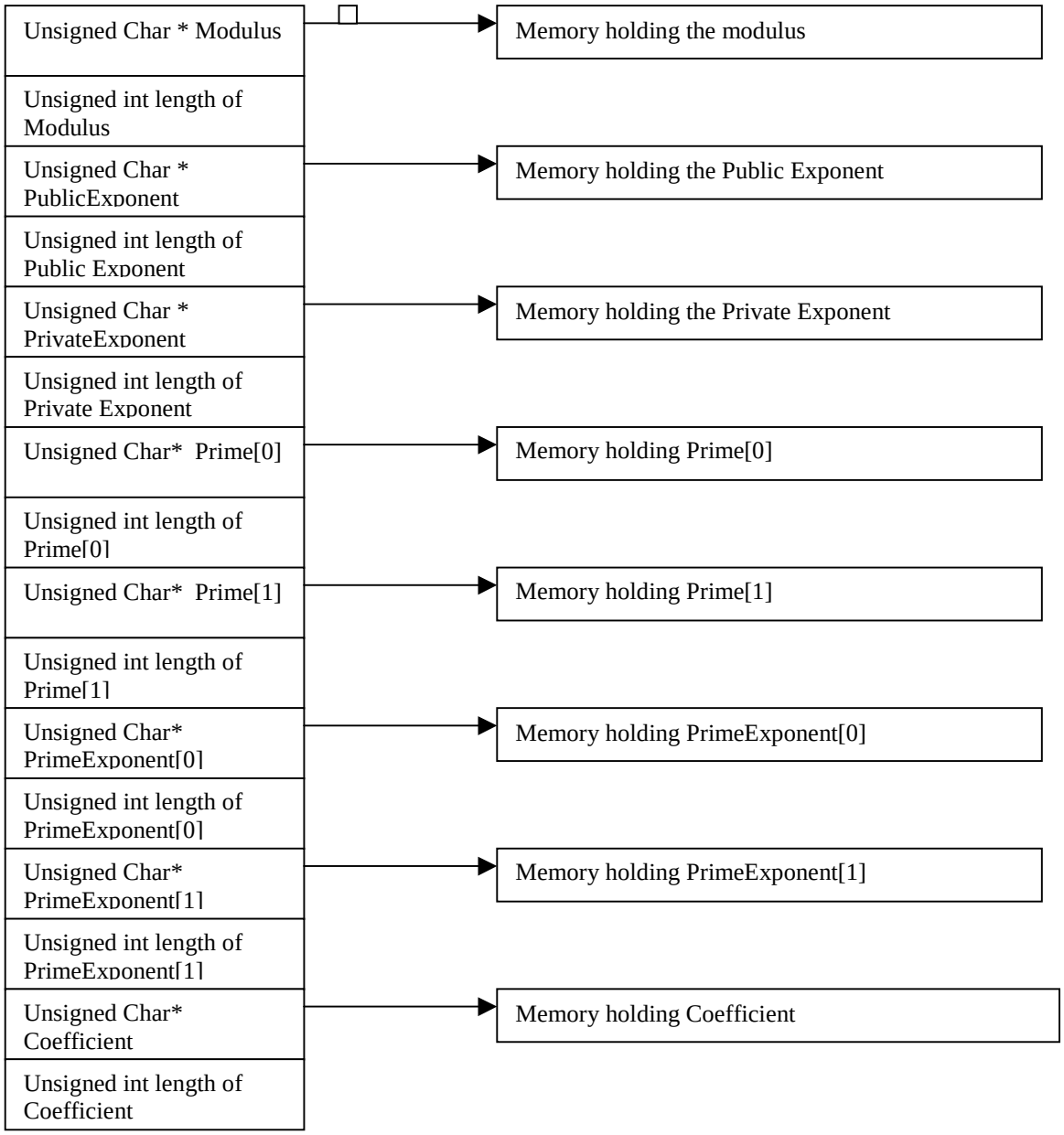
The Cryptographic Service Provider was implemented according to the specifications defined by the Microsofts CSP developers guide. This ensured that the CSP would be interoperable with Microsoft CSP's and could thus take part in SSL client authentication, certificate request generation etc with the Internet Explorer 5.0.
The CSP internally used BSAFE to do most of the cryptographic computations.

The Smart Card CSP performed the following major functions :

1. RSA key generation.
2. Key management.
3. Certificate management
4. Hash generation
5. Digital Signature generation.
6. Digital signature Verification.

The Smart Card CSP generates 1024 bit RSA key pairs. Since Multiflex cards did not have on-board key generation mechanisms, these were generated using RSA's BSAFE libraries. A public exponent of 65537 was used. After key generation the keys are stored on the smart card. Access to the keys are protected by PIN verification.

Figure 23. The Representation of the RSA key pairs in memory.



This format allows the key pair to be stored in the CRT format which results in faster cryptographic computations.

The keys were stored in a PIN protected elementary transparent file on the card. For storage onto the smart card the whole structure is interpreted as a sequence of bytes

and then stored in the TLV format. While reading the keys from the smart card the byte sequence is converted back to the above structure. The memory holding the keys were corrupted immediately after the key operation was over. This was done to prevent any malicious program from stealing the private key. Currently the key container was designed to hold two pairs of RSA key pairs. One for key exchange and the other for digital signatures.

The Smart Card CSP does not store the certificate in memory. The only instance when the CSP handles a certificate is when Internet Explorer calls CPSetKeyParams(...KPCertificate...). . The CSP simply takes the certificate blob passed to it from IE. Parses it based on the DER encoding mechanism to find the length of the blob. Then it calls the necessary smart card routines to store the certificate on to the card in the TLV format.

In order to achieve maximum interoperability with Microsoft CSP's the Smart Card CSP calls CryptoAPI functions to generate signatures and verify signatures. Hence the keys had to be exported to the Microsoft Enhanced CSP, which required that the keys be transformed from RSA's Big-Endian representations to Micrsofts Little-Endian format.

4.5 Implementation Of The Mutual Authentication Modules

Most of the important features of this module has been discussed extensively in the design section of this thesis.

The major chunk of the implementation lied in the implementation of the JNI routines which generated the messages and the coordination of this messages between the viewer and the provider side to achieve mutual authentication.

The message structure for transferring information between viewer and provider was based on the TLV format.

Figure 24. Message Format

Part1

Tag For Certificate (1 Byte)	Certificate Length (2 Bytes)	Certificate (Certificate Length Bytes)

Part2

+

Tag For Encrypted Data (1 Byte)	Encrypted Data Length (1 Byte)	Encrypted Data (Encrypted Data Length Bytes)

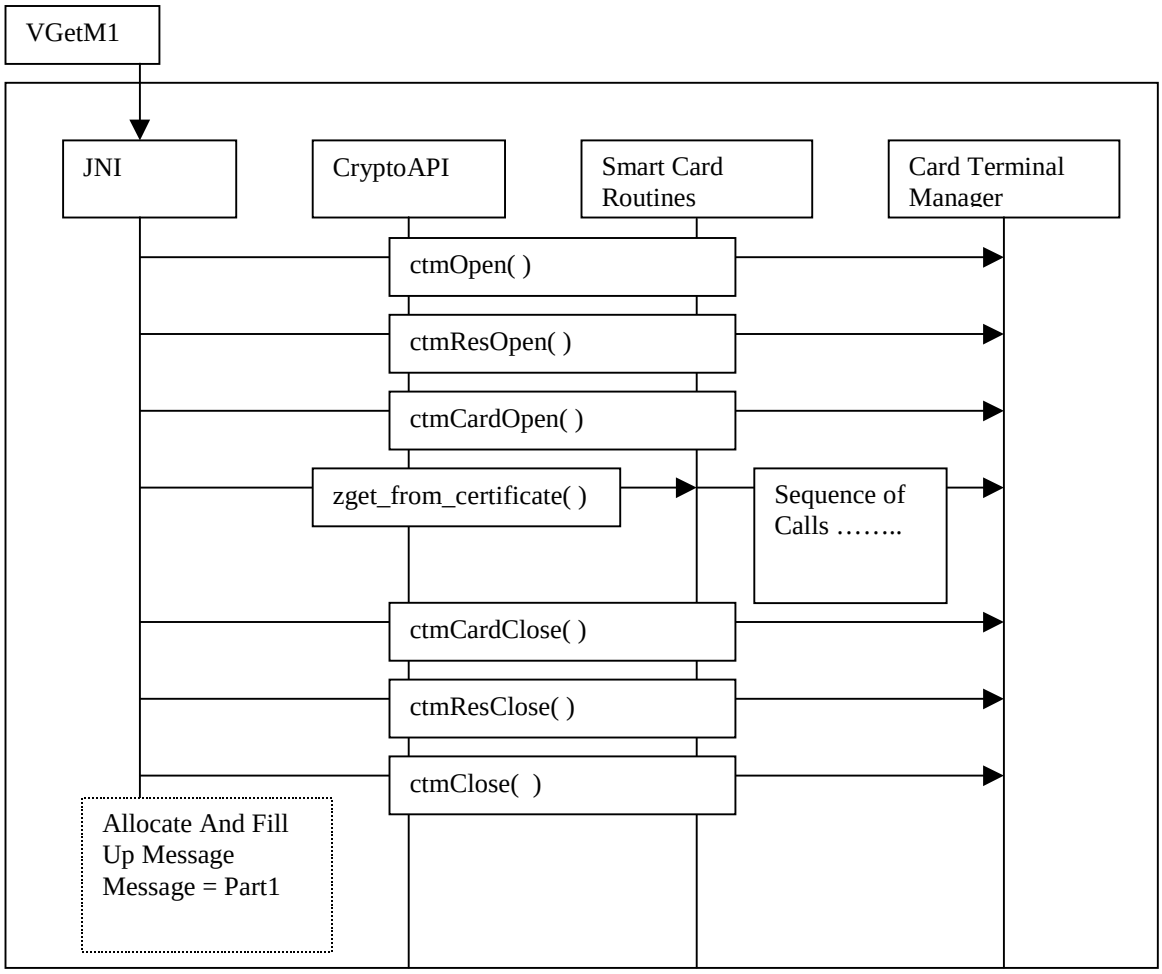
Part3

+

Tag For Signed Data (1 Byte)	Signed Data Length(1 Byte)	Signed Data (Signed Data Length Bytes)

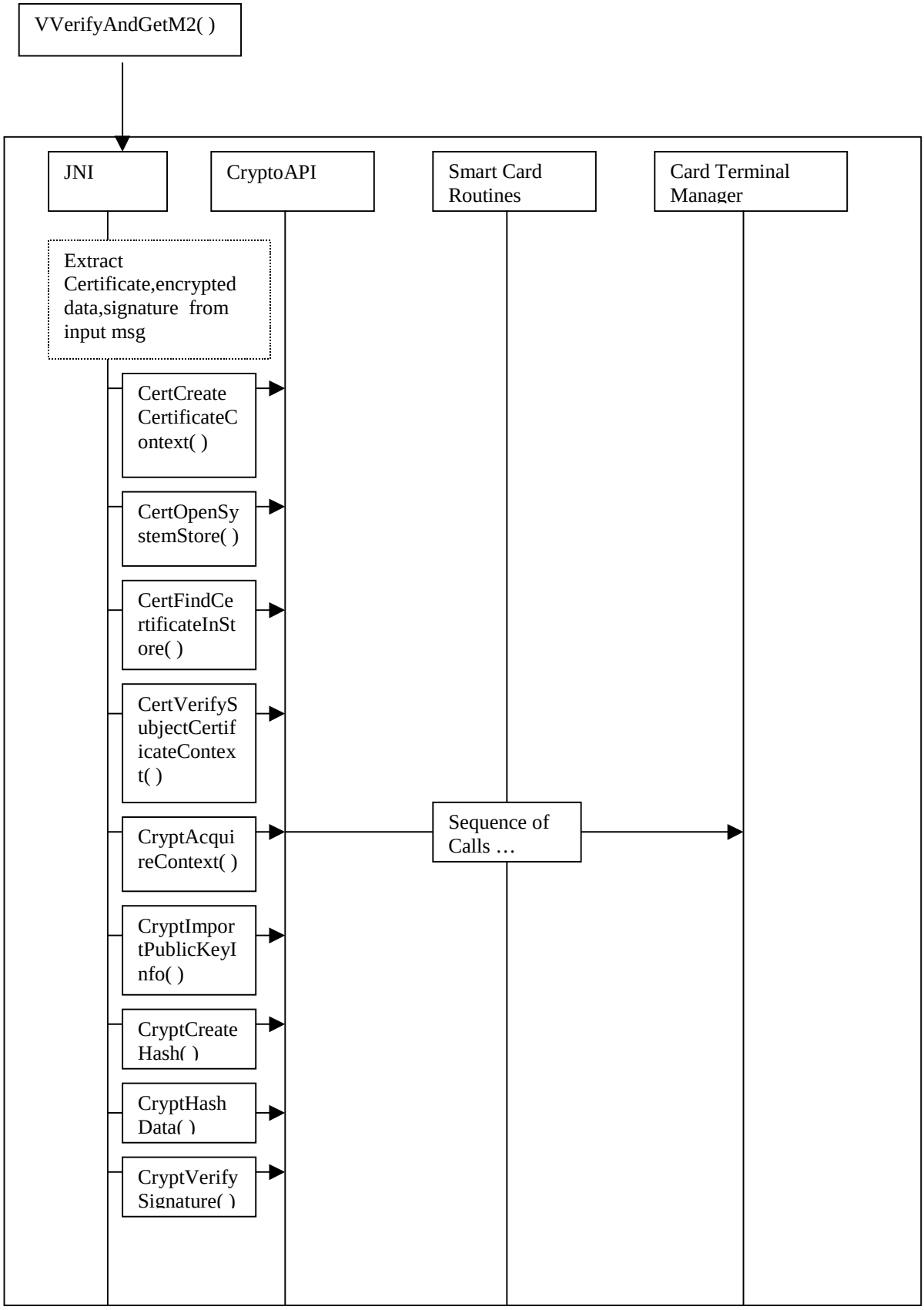
The JNI functions allocate memory for these message structures and fill them returning a pointer to them.

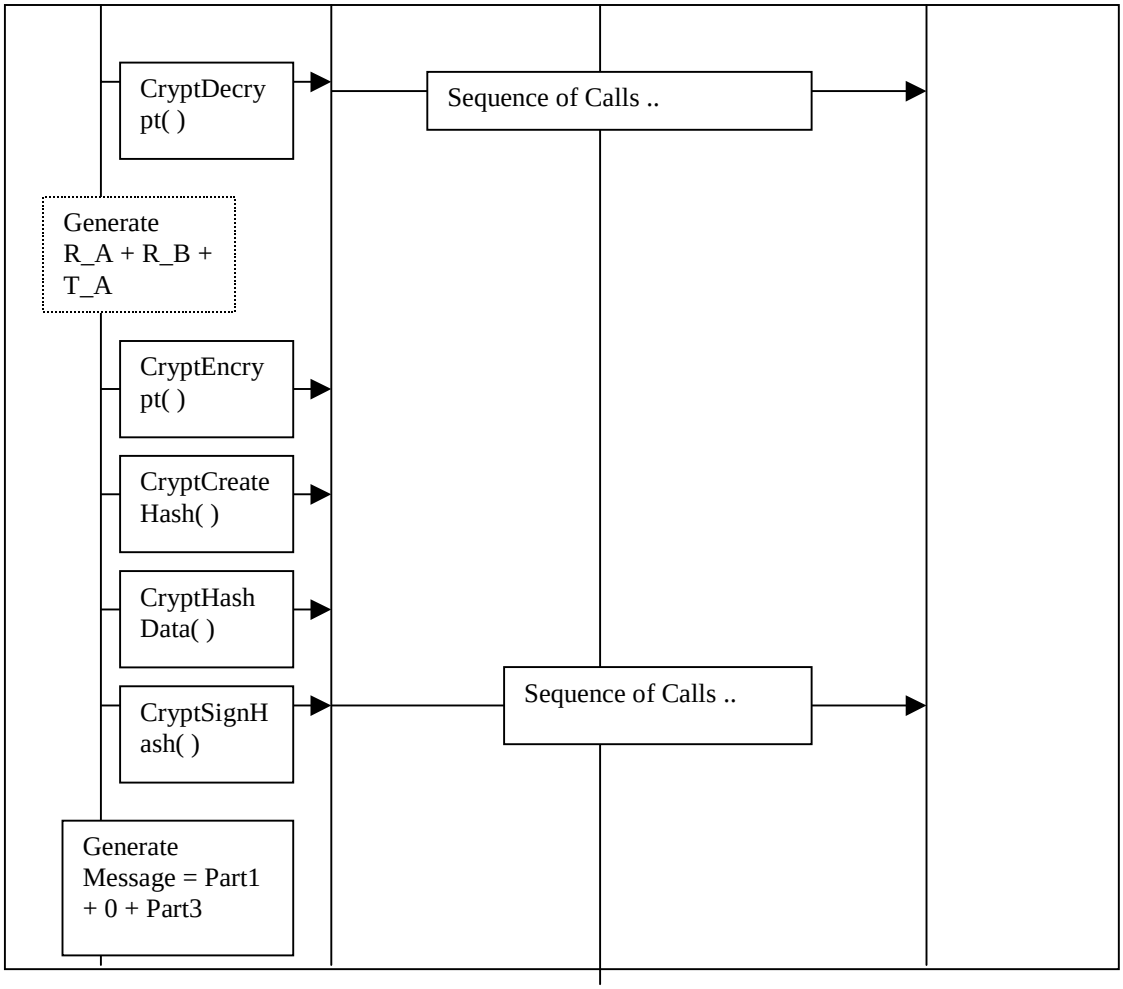
Figure 25. A closer look at two of the JNI implementations to support mutual authentication.



ctmOpen( ), ctmResOpen( ) , ctmCardOpen( ) , ctmCardClose( ), ctmResClose( ), ctmClose( ) are card terminal manager functions called to get access to and close the card.

zget_fromcertificate( ) retrieves the certificate from the card. This function in turn calls the card terminal manager to access the card.


VVerifyAndGetM2( )
JNI
CryptoAPI
Smart Card Routines
Card Terminal Manager
Extract Certificate,encrypted data,signature from input msg
CertCreate CertificateContext( )
CertOpenSystemStore( )
CertFindCertificateInStore( )
CertVerifySubjectCertificateContext( )
CryptAcquireContext( )
Sequence of Calls …
CryptImportPublicKeyInfo( )
CryptCreateHash( )
CryptHashData( )
CryptVerifySignature( )



Input to the function : E_A(R_B + T_B) + C_B + D_B (H (E_A (R_B + T_B)))

Output From Function : E_B(R_A + R_B + T_A) + D_A (H (E_B (R_A + R_B + T_A)))

The function parses the input into Part1 + Part2 + Part3

Part1 = C_B ; Part2 = E_A(R_B + T_B);

Part3 = D_B (H (E_A (R_B + T_B)));

CertCreateCertificateContext(.. C_B..) : This is a Crypto API call which creates a context in memory from certificate C_B. This has to be done to verify B's signature.

CertOpenSystemStore( ) : This function is called to open up a certificate store. The aim is to retrieve the CA certificate in order to verify the validity of C_B.

CertFindCertificateInStore( ) : This Crypto API call is used to search for the CA "Fayette" certificate in the certificate store opened in the last step.

CertVerifySubjectCertificateContext( ) : This CryptoAPI call actually validates the CA's signature present in C_B by using the public key from the CA certificate found in the last step.

CryptAcquireContext( ) : Used to acquire context to the Smart Card CSP.

CryptImportPublicKeyInfo( ) : Imports the public key from C_B into the CSP. This is done as the CSP needs to have the public key in order to verify the signature.

CryptCreateHash( ),CryptHashData( ),CryptVerifySignature( ) : verifies B's signature. D_B (H (E_A (R_B + T_B)))

CryptDecrypt( ) : This function uses the private key stored on the card to decrypt E_A (R_B + T_B) to retrieve R_B & T_B

The raw message R_A + R_B + T_A is generated by adding the time stamp to the retrieved value.

CryptEncrypt( ) : The public key imported into the CSP is used to produce E_B(R_A + R_B + T_A).

CryptCreateHash( ),CryptHashData( ),CryptSignHash( ) is used to sign the encrypted message with the private key stored on the smart card.

The encrypted message and its signature are concatenated together to produce the output for this function.

4.6 Problems Faced

Developing the GINA module was very tricky. It was very tedious to debug it. The only way to debug it was to run it and log variable values in log files. Moreover if the GINA dll being developed had bugs(especially memory problems) it would blue screen the computer while trying to logon. Once the computer blue screened there was no quick way to get back inside and repair the registry. The only solution was to repair WinNT by running the WinNT setup disks, a time consuming process!.

The other difficulty faced was while developing the Cryptographic Service Provider. Sometimes it got quite tricky trying to make the smart card CSP interoperable with existing Microsoft CSP's. One such problem was the endian problem. Microsoft stores its keys in little-endian byte order. Whereas RSA's BSAFE library used for generating keys in the CSP generated the keys in big-endian byte order. It took me some time to figure that one out. Other than this I was having problems while generating signatures. Due to some padding mechanism differences, the signatures generated using BSAFE libraries were not getting verified properly by Microsoft CSP's. I got around this problem by calling CryptoAPI and using Microsoft's enhanced provider to generate the signatures and verify the signatures.

5. Conclusions And Future Work.

5.1 Conclusions

The objective's set forth at the beginning of the work were completely satisfied. Mutual authentication using smart cards as well as using the smart cards as mobile containers for certificates,associated keys and logon information in a secure telemedicine environment was successfully demonstrated.

5.2 Future Work

The smart card used for this work was Schlumburger's Multiflex 8K card. This card did not have the capability of generating RSA key pairs, generating digital signatures and encrypting /decrypting using public/ private key respectively. Thus all these operations were done on the application side instead of on the smart card. With the availability of more sophisticated smart cards like Schlumburger's Cyberflex card, these activities could be moved on to the smart card. This would enhance the security of the system as the private keys don't have to come out of smart card at all. The presence of private keys on the PC increases the security risks as it could fall prey to rogue software designed to read sensitive and private information from the memory.

Presently the software is only interoperable with Microsoft products. That means the certificate stored on the card cannot be used for activities like SSL client authentication on other browser implementations like Netscape etc, a certificate cannot be downloaded to the card on any other platform but Microsofts, the mutual authentication mechanism of two physicians over the network will not work unless both sides are running Microsoft WinNT. This would hamper widescale growth of such security systems. Making this system work with other operating systems and browser technologies is left for future work.

References

1. Anderson, Ross J. *Why Cryptosystems Fail.* Communications of the ACM. November1994. pp 32-41
2. Bhimani, Anish. *Securing the commercial internet.* Communications of the ACM. June 1996 - Volume 36, Number 6.
3. Comer, Michael. *How passwords are cracked.* Computer Fraud & Security Bulletin, 7(1):1-10, November 1981.
4. Comer, Michael. *Password breaking.* Computer Fraud & Security Bulletin, 4(2):7-8, December 1981.
5. Comer, Michael. *Underground advice.* Computer Fraud & Security Bulletin, 4(2):8-11, December 1981.
6. Doorns, Leendert van. *Computer Break-ins: A Case Study.* Vrije University, Amsterdam. The Netherlands.
7. Elizabeth, Dorothy and Denning, Robling. *Cryptography and Data Security.* Addison-Wesley, Reading. MA. January 1983.
8. EU/G7 Healthcards - WG7, *Interoperability of Healthcard Systems, Part 3 Interoperability Specification.* http://www.clinical-info.co.uk/euhci.htm.
9. Fifield, Ken J. *Smartcards outsmart computer crime.* Computer and Security, 8(3):247-255, May 1989.
10. Frank, John N. *A Doubtful Diagnosis for Smart Health Cards.* Card Technology. May/June 1997.
11. IONA Technologies,Ltd. Dublin, Ireland. *Programmer's Guide for Orbix,* Version 1.2, 1994.
12. Java Card Forum. *Java Card API Version 2.0.* http://www.javacardforum.org/.
13. Kaliski Jr., Burton S. *A Layman's Guide to a Subset of ASN.1, BER*

and DER. Technical Note.

http://www.rsa.com/pub/pkcs/ascii/layman.asc.

14. Microsoft. CryptoAPI. http://www.msdn.microsoft.com/library/

15. Microsoft. Cryptographic Service Providers. http://www.msdn.microsoft.com/library/

16. Microsoft. Smart Card Reference. http://www.msdn.microsoft.com/library/

17. Microsoft. Winlogon User Interface. http://www.msdn.microsoft.com/library/

18. National Bureau of Standards. *Data Encryption Standard* FIPS PUB 46-1 (1987)

19. Netscape Communications. *Introduction to SSL*.

http://developer.netscape.com/docs/manuals/security/sslin/contents.htm

20. PC/SC Workgroup. *Interoperability Specification for ICC's and Personal Computer Systems.* PC/SC Workgroup Documents. http://www.smartcardsys.com/

21. RSA Data Security Inc. http://rsa.com

22. RSA Data Security, BSAFE, User's Manual. Version 3.0

23. RSA Data Security, BSAFE, Library Reference Manual. Version 3.0

24. Schlumberger, Inc. *Multiflex Card Reference Manual*. Sptember 25. 1996.

25. Schneier, Bruce. *Applied Cryptography*. John Wiley & Sons,1994.

24. Schneier, Bruce & Shostack, A . *Breaking Up Is Hard to Do: Modeling Security Threats for Smart Cards.* http://www.counterpane.com/smart-card-threats.html

25. Stroustrup, Bjarne. The C++ Programming Language. Addison-Wesley 1991. Second Edition.

26. TrustHealth 1, Trustworthy Health Telematics. Selection of Security Services and Interfaces. http://www.ehto.be/projects/trusthealth/deliver.html

27. TrustHealth 1, Trustworthy Health Telematics.Guidelines for Implementation of Security Services and Interfaces.

http://www.ehto.be/projects/trusthealth/deliver.html