2003

User authentication in distributed computing using proactive two-party signatures.

Arshad Ahmed. Shaikh
University of Windsor

User Authentication In Distributed Computing using Proactive Two-Party Signatures

By

Arshad A. Shaikh

A Thesis
Submitted to the Faculty of Graduate Studies and Research
through the School of Computer Science in Partial
Fulfillment of the Requirements for the Degree of
Master of Science at the University of Windsor

Windsor, Ontario, Canada
2003

Your file Votre référence
ISBN: 0-612-84561-3
Our file Notre référence
ISBN: 0-612-84561-3

Abstract

To address the challenges of user authentication in distributed systems, many protocols have been proposed. Among them is the Proactive Two-party Signatures Scheme (P2SS). P2SS has not so far been investigated in detail. Using P2SS, I have developed a system, called Proactive User-authentication in a Distributed Environment (PUDE) in which two parties (the client and server) use a 2-part signature algorithm, based on standard (one part) RSA algorithm. In this scheme the client and the server jointly produce the signatures. If the password of the user is compromised, the system can recover by changing the share of the private key and the password on the primary server only. To test PUDE, a basic multi-realm test-bed called the Secure Multi-realm System (SMS) has been designed. PUDE is able to bring out the usefulness of a proactive two-party signature scheme over different realms. Its security is based on the unacceptable high computational effort required for computing discrete logs. This is a widely-accepted basis of security for security protocols.

Dedication

To my Mom and brother Ali !

Acknowledgments

Words fail to express my indebtedness to Dr. A. K. Aggarwal for his guidance and enthusiasm and who made this thesis a very enjoyable endeavor for me, I am grateful to my advisor for his guidance and advice during the development of the research presented in this thesis. I have been fortunate to have Dr. A. K. Aggarwal as an advisor and to have the honor of working with him. I am also dedicating this thesis to him.

I would also like to thank Dr. R. D. Kent from department of Computer Science for his valuable statistical guidance, support, impeccable understanding and for his friendly encouragement that allowed me to write this thesis.

I would also like to thank Dr. Joan Morrissey from the department of Computer Science for her kindness and agreeing to join my thesis committee as an internal reader and for generously taking time to read an earlier version of this thesis proposal and for offering many insightful comments and suggestions for improvement.

I would also like to thank Dr. A. van den Hoven from department of French Studies for generously taking time to read an earlier version of this thesis

proposal and for his guidance for improvement.

My gratitude for Dr. Alioune Ngom from department of Computer Science for his kindness and agreeing to join my thesis committee as the chairman.

I would like to thank my family, my mom, my brother Ali, Mushtaq, Mansoor, my sisters Sarwar and Sheema, for their continuous support through out my education. Also my extended love for my nephews Ahmer, Mashad and niece Maria.

I am also grateful to my friends, Adlan Habed, Salim Murad, Aniss Zakaria, Aniruddha Bharadwaj Sridhar and Farrukh Siddiqui who encouraged me to put as much of myself into this work as humanly possible.

My special gratitude for my special friends, Kamran Choudhery and Arslan Khan.

Table of Contents

List of Figures

Chapter 1 Introduction

User authentication is a key feature of distributed computing security which deals with integrity of information. A secure system must ensure that the information, communicated, stored or transmitted could only be accessed by a legitimate user. This concept was first introduced in the 1960s along with the idea of time sharing, multi-user computing. For more than two decades distributed computing security was primarily a defence and military issue. It took center stage after the advent of Internet and electronic commerce, after which its applications became of commercial interest.

As the computing environment changed from a few centralized time-sharing machines to a large number of decentralized workstations, it became obvious that the security of the network could not be enforced by simple techniques based on host- based authentication, as the messages from hosts on the network could not be trusted anymore. That means, that any transaction flow from host to server transmitted in today's electronic world can be seen and probably modified by malicious and adverse element. So we have to take security measures that ensure that the information being processed, stored or communicated is reliable and available to those who are authorized recipients.

As a message passes through a number of routers or on the local area network, on its way from its origin to its destination, some malicious elements may acquire access it or they may modify it. Whether they are able to change it or not, they

can get a copy of the data traveling over a network. Numerous identity theft cases and credit card frauds provide an example of it.

Moreover it became necessary to develop techniques for securing privileged information traveling across the network.

1.1 Overview

The importance of securing the communications path between the distributed entities increases as our reliance on Internet technologies for valuable jobs becomes more. That security technology is not losing out to hackers is proved by the fact that transactions, that only a few years ago we would trust being done only over private networks, are being conducted today using Internet technology. Applications, such as securities transactions, payments, online banking, payroll and health care are routinely entrusted to travel over open networks. Obviously the data is extremely sensitive and all sorts of mischief could ensue if it were to be exposed to the wrong entity or if it were modified during transmission. However even though the security technologies are able to ensure the integrity and confidentiality of the jobs quite well, there does not exist an absolutely secure system. Therefore issues that improve the degree of security can be always studied. Or one may explore new security architectures that improve security.

With the introduction of distributed systems, the security technologies have to face bigger challenges. The Internet is a less than ideal medium to conduct such transactions. Internet protocol version 4 [Postel 1981] has a number of well-known security problems [Bellovin 1989], [Bellovin 1995]. To over-come these

problems, security features have been added to the new suit of protocol [Deering 1995], [Kent 1998]. Cryptography and authentication play a major role in the new systems.

1.2 Cryptography

Cryptography in Greek stands for "hidden writing." Cryptography is the art and science of transforming information into an intermediate form, which secures that information while it is in storage or in transit. Cryptography is used to encrypt a given “plaintext” into a “ciphertext” and to decrypt a “ciphertext” to “plaintext”, for the purpose of providing data security. Cryptographic algorithms can be used for entity and data origin authentication and for providing confidentiality and integrity.

Cryptography allows users to disguise data so that attackers gain no information from listening to the information during transmission. Thus cryptology provides a mean for two entities to have confidential communication.

Cryptography offers a set of security tools for resolving a variety of problems, ranging from securing data secrecy, through authenticating information and parties, to more complex multi-party security goals. Even though cryptographic techniques have been used since 1970s [Menezes 1997], the most notorious attacks till now, on cryptographic security mechanisms have been system attacks in which the cryptographic keys are directly exposed, rather than crypto analytical attacks which are the result of analyzing cipher texts.

There are four kinds of cryptographic algorithms:

- Symmetric Algorithms
- Asymmetric Algorithms
- Message Digest Algorithms
- Digital Signature Algorithms

1.2.1 Symmetric Algorithms

It is the oldest form of encryption. It is also known as 'secret key' cryptography. This secret key is the main tool for security and it is only known to the participants of the secure communication. This key has to be shared by the participants and is required to be kept as a secret. The requirement to share the key makes the symmetric algorithm vulnerable. Its security is immediately compromised if the secret-key is not secret any more. Symmetric cryptography can be further divided into two categories: stream ciphers and block ciphers.

- Stream Ciphers

Stream ciphers operate on pseudo-random data of plain text, i.e. encrypting and decrypting it one bit at a time.

- Block Ciphers

Block ciphers operate on a fixed-length block of plain text and encrypt it into cipher text of the same length. Decryption is conducted using the reverse transformation and the same secret key.

Digital Encryption Standard (DES) was the accepted symmetric key standard by NIST, USA (National Institute of Standards Technology) till 2001, when it was replaced by AES (Advanced Encryption Standard).

1.2.2 Asymmetric Algorithms

Asymmetric cryptography is also known as public-key cryptography. It is a relatively a new field of cryptography, compared to symmetric cryptography that is also called conventional cryptography. The encryption and decryption processes are done with two different keys. The decryption key cannot be calculated from the encryption key within a reasonable amount of time.

1.2.3 Message Digest Algorithms

Asymmetric algorithms can only encrypt data blocks smaller than their key size. Moreover the implementation process is quite slow. These problems can be avoided to some extent by creating message digests. The algorithms for creating message digests are more efficient than the algorithms for encryption using asymmetric key. A message digest can be created by applying a secure hash function.

In a one-way hash function, it is infeasible to find out the input data if the output hash value is given. Each hash function, H(M), takes an input, the original message, and gives the output, called message digest. A one-way hash function

operates on an arbitrary-length message and returns a fixed-length hash value, h. Each message digest algorithm usually determines the length of a hash, n.

$$h = H(M), \text{ where } h \text{ is of length } n.$$

The major benefit of hashing a message lies in the less computational overhead when signing a message digest than signing an original message.

Hash functions can be used for authentication and verification. Examples of message digest algorithms include MDx, Haval, RIPEMD, and SHA-1.

- MD5

MD5 (Message Digest) is mostly used for digital signature applications by applying an asymmetric algorithm, e.g. RSA to the hash obtained from MD5. MD5 processes the input data in 512-bit blocks, which are further divided into 16 32-bit blocks. Finally MD5 produces a 128-bit message digest.

- SHA-1

SHA-1 (Secure Hash Algorithm), developed by NSA (National Security Agency) and NIST (National Institute of Standards and Technology), is used in Digital Signature Standard (DSS). The SHA-1 algorithm takes the input of 512-bit data and produces a 160-bit message digest. SHA-1, in conjunction with Digital Signature Algorithm (DSA), can be used in electronic mail, electronic transactions, software distribution and data storage. SHA-1 is more resistant to brute-force attacks, compared to other peer algorithms such as MD5.

In my design and implementation, I chose the message digest algorithm SHA-1 because SHA-1 is widely used. It has a satisfactory computational efficiency. In

addition, SHA-1 is supported by quite a few software vendors, including Sun Microsystem, in the form of programming API.

1.2.4 Digital Signature Algorithm (DSA)

The NIST proposed the Digital Signature Algorithm (DSA) for use in their Digital Signature Standard (DSS), in August 1991.

The DSA generates a digital signature on a message. It can also verify the authenticity of a signature. DSA uses the asymmetric key system. To be more specific, the private key is used in signing a message, and the public key is used in the signature verification. First the message is hashed using SHA to improve the computational efficiency of signing.

DSA uses a variant of the Schnorr and ElGammal signature algorithms. In my experiment, I have used DSA to compare with the RSA P2SS implementation and I have shown that PUDE has a lower cost of computation overhead as compared to DSA.

1.3 Digital Signatures in Authentication

We use signatures in today's environment for identifying ourselves uniquely. A signature has several important properties. It is considered to be:

(i)Authentic – where the recipient is aware that the signer deliberately signed the document.

(ii) Unforgeable – where the signature is proof that the signer signed the document.

(iii) Not Reusable – where the signature is part of the document and cannot be copied onto another document.

(iv) Unalterable – where once a document is signed, it cannot be changed.

(v) Nonrepudiable – where the signer cannot claim that he/she didn't sign the document.

It is these properties that make signatures so indispensable.

Authentication allows clients and servers to securely determine each other's identities.

Authentication is the process of reliably verifying the identity of someone (or some computing entity). There are two types of authentications used in networks. One is when a computer is authenticating another computer. For example a print spooler may attempt to authenticate a printer. Another case is when a computer performs operations for a user where the user supplies a password.

Digital signatures depend upon the asymmetric system. It is due to the importance of digital signatures in digital economy, that Stalling said. "The development of Public-key cryptography is the greatest and perhaps the only revolution in the entire history of cryptography" [Stallings 1999].

Related work includes 2-party signature schemes, proactive cryptosystems and security provided by proactive method for maintaining the overall security of a system, even in cases where individual components are repeatedly broken into and manipulated by an attacker. The proactive method provides for automated recovery of the security of individual components, thus avoiding the use of expensive and inconvenient manual processes (except perhaps when an

ongoing attack is detected). Multi-party techniques call for the distribution of key components among several components. Pro-active methods use periodic refreshments of the private key data held by the servers. This way, the proactive multi-party method assures uninterrupted security provided that the client and the server are not both compromised. We describe the proactive approach in the context of our design of PUDE using 2-part RSA algorithm.

1.4 Our Approach and Contribution

This thesis explores the design of an authentication mechanism for User-Authentication in distributed computing (PUDE) using proactive two-party signature scheme. It is the first work that uses 2-part RSA algorithm in P2SS in distributed computing, with single-sign-on in different realms. For each user, the sign in is required by the default realm only. A secure channel has been used for all the communication between the client and the server. The framework, designed under the project, improves the strength of the authentication functionality. It provides a more flexible way for local security policy-making. The design and implementation of the PUDE has been completed, and its comparison with another related algorithm is made.

The Proactive Two-Party Signature mechanism would require the design and implementation of a Secure Multi-Realm System (SMS) for effectively using

P2SS with 2-part RSA. So the thesis also provides the detailed framework of SMS.

I have compared the efficiency of PUDE with DSA, the standard for digital signatures.

RSA has been used widely in industry and the NIST standard are largely based upon RSA. 2-part RSA algorithm should therefore prove to be a useful technology.

For implementing PUDE in SMS, technologies, including PBE (Password Based Encryption), SHA, and DES, have been used to enhance the overall security of the system.

1.5 Thesis Outline

This thesis starts with a brief introduction of the problem, that has been studied. Chapter 2 gives the background literature and related description of (one-part) RSA Digital signature system. Chapter 3 focuses on the motivation for developing P2SS with 2-part RSA and the design of PUDE. Chapter 4 concentrates on the design of the secure-multi realm (SMS) system for use with PUDE. Chapter 5 gives results of experiments and performance evaluation. Chapter 6 concludes the work with a few lines on the possibility of Future Work.

Chapter 2 Related Work

Introduction

A security system has to be designed to face specific type of threats. So the type of threats that are likely to be faced by a system have to be studied before an appropriate strategy for design is adopted. The chapter begins with a classification of the type of security threats faced by a system on the Internet. It is followed by a description of the standard RSA signature scheme. The related work on multi-party threshold systems and pro-active systems is reported in the next Chapter.

2.1 Types of Security Threats

A distributed system is susceptible to a variety of threats by intruders as well as legitimate users of the system. Indeed, legitimate users are more powerful adversaries, as they possess internal state information rarely available to an intruder. There are two general types of threats. The first one, host compromise, refers to the subversion of individual hosts in a system. Various degrees of subversion are possible, ranging from the relatively benign case of corrupting process state information to the extreme case of assuming total control of a host. Host compromise threats can be effectively dealt with through a combination of hardware techniques (like processor protection modes) and software techniques (like security kernel/reference monitor) [Denning 1982]. The second type, communication compromise, includes threats associated with message

communications. T. Woo and S. Lam [Woo 1992] have subdivided the communication threats into the following classes:

(T1) eavesdropping of messages transmitted over network links to extract information from private conversations;

(T2) arbitrary modification, insertion, and deletion of messages transmitted over network links to confuse a receiver into accepting fabricated messages; and

(T3) replay of old messages; this can be considered a combination of (T1) and (T2).

(T1) is a passive threat, while (T2) and (T3) are active threats. A passive threat does not affect the system being threatened, whereas an active threat does. Therefore, passive threats are inherently undetectable by the system, and can only be dealt with by using preventive measures. Active threats, on the other hand, are combated by a combination of prevention, detection, and recovery techniques.

As we are dealing with the second case threat (T2), we are going to focus on P2SS, based on 2-part RSA signature algorithm.

In the digital domain, where copying and modifying files is easy. Therefore it is desirable to use digital signatures, which have the five properties of signatures, stated in section 1.3. For example, when a user signs a document, the receiver would know that it was actually the user who signed it and that the document was not altered after it was signed.

2.2 RSA Digital Signature

Named after its three inventors, Ron Rivest, Adi Shamir, and Leonard Adleman, RSA is the best-known and most frequently used reversible asymmetric algorithm. The security of this algorithm is based on the difficulty in factoring large numbers [Francis 2002].

RSA has a variable length key (between 512 bits and 2048 bits). It also has a variable block size, which must be smaller than the key lengths. The length of the cipher text will be the length of the key. The public and private keys are functions of a pair of large prime numbers. Each participant will have to generate a public and corresponding private key. The public key consists of the product of two large primes p and q, and a fixed number e. The private key is a number, d.

The basic algorithm is to generate two large primes p and q, multiply them , and get the result n. The factors p and q will remain secret. A private key and a public key will be generated from p, q, and n.

Recovering the private key from the public key and the cipher text is conjectured, to be equivalent to factoring the product of the two primes. A summary of how to use RSA method is given below [Stallings 1999]:

Randomly choose two random large prime numbers, p and q (e.g. of 1024 bits each or higher).

To maximize security, choose p and q of nearly equal length.

Compute the product $n = pq$ [Stallings 1999]

Randomly choose the encryption key e such that e and (p-1)(q-1) are relatively prime. This means that they have no prime factors in common.

Compute the decryption function d, such that (de – 1) is evenly divisible by (p-1)(q-1). This can also be written as:

de = 1(mod (p-1)(q-1)) here d is called the multiplicative inverse of e.

d and n are also relatively prime. The numbers e and n constitute the public key; the number d is the private key.

To encrypt a message P, divide it into blocks smaller than n.

The encryption function is:

encrypt(P) = $((P^e) \bmod pq)$ where P is the plain text.

The decryption function is:

decrypt(C) = $((C^d) \bmod pq)$ where C is the ciphertext.

Using RSA, you are free to publish your public key , as there is no known method for calculating d, p, or q, given only your public key (pq, e). If p and q are relatively large (greater than 1024 bits), it may take hundreds of MIPs years before even the most powerful computers can factor the modulus into p and q. The real premise behind RSA's security is the assumption that factoring a big number is difficult. The best-known factoring methods are really slow. To factor a 2048-bit number with the best-known techniques would take about a million of MIPS- years [RSA 1999]. Ultimately, however, it is not far fetched that another method will be discovered to cryptanalyse RSA [Stallings 1999].

Public key algorithms, such as RSA, are much slower to compute than secret key algorithms, such as DES. As a result, RSA is not used for encrypting long

messages. It is used for digital signatures and for encrypting a secret key, which may be used to actually encrypt a long message.

Chapter 3 Proactive Two-Party Signatures Scheme

3.1 Motivation of P2SS

The Digital signature is one of the most widely used authentication technologies. This signature mechanism can be based on any one of the many algorithms, developed by cryptographists. The RSA algorithm for digital signatures is probably the best-known one in the digital industry.

The signature mechanisms are secure only if the security of the private key is not compromised. Unfortunately the systems, on which the keys are stored, may be attacked by intruders, hackers, or through software trapdoors using viruses or Trojan horses. Such attacks are frequently successful, owing to the fact that the environments and operating systems of today are complex systems. There is no design methodology, which can guarantee fully secure operating systems and environments. Since successful attacks on computer systems can lead to availability of private keys to hackers, one cannot depend only upon the security of digital signature algorithms for securing information. Moreover attackers try to avoid being detected at all costs. They would, more often than not, willingly give up control on a computer rather than risk being detected.

In the face of such an environment, a common approach for enhancing the security is the periodic refreshment of secrets. The time period after which this periodic refreshment is performed, can be set with the security refreshment in secure communication protocols such as IP-SEC [Carrel 1997] and SSL/TLS. This renders "old secrets" (i.e., secrets from before the refreshment) useless for

the attacker. Thus even if the attacker is able to get the private key, the information may be of no use because the private key may be modified before the attacker is able to use the information..

Another way to enhance security is to distribute cryptographic trust among several components or servers. This approach can best be seen in secret sharing algorithms [Shamir 1979], [Blakley 1979], and taken to another level through the notion of threshold cryptography [Desmedt 1989], [Gemmell l997]. Here a secret key is split into shares, with one share being given to each one of a group of servers. The servers engage in a protocol that "emulates" the behavior of the centralized solution (the case where the key is kept in one piece), as mentioned by Phlip Makenzie [Makenzie 2001]. The protocol ensures security as long as most of the servers ("threshold") stay un-compromised. Threshold cryptography improves the security against break-in attacks in many scenarios. However, it has its limitations. If he has the required time, an attacker can break into the servers one by one, thus eventually compromising the security of the system [Makenzie 2001]. This danger is particularly imminent in systems that must remain secure for long spans of time (such as certification authorities) or in cases where secure recovery proves to be difficult (such as with secure communication).

Proactive security is a mechanism designed to protect against such long-term attacks. It blends the approach calling for distribution of trust with the one of periodic refreshment [Frakel 1997]:

Proactive = Distributed + Refresh

First of all distribute the cryptographic capabilities among several servers. Next, have the servers periodically engage in a refreshment protocol. This protocol will create the possibility for servers to automatically recover from possible, undetected break-ins, and specifically will provide the servers with new shares of the sensitive data while keeping the sensitive data unmodified [Frakel 1997a]. The information gathered by an attacker before a refreshment period becomes futile for future attacks on the system even when attacker may be able to prevent detection. All in all, the security of the system will be guaranteed, provided not too many of the servers are broken into between two consecutive executions of the refreshment protocol [Herberz 1997]. With this approach, security can be maintained even when over a long period, every server is broken into at some time or another. In other words, a proactively secure system does not wait until a break-in is detected. Instead, it invokes the refreshment protocol periodically (and proactively) in order to maintain uninterrupted security or force detection [Herberz 1997].

For PUDE, periodically (say, every day/week) the servers will engage in a refreshment protocol, ensuring the security of the PUDE as long as (both halves) of the client and server are not compromised.

Earlier work on Proactive Security: Ostrovsky and Yung [Ostrovsky 1991] showed how a large class of multiparty protocol problems could be solved in a proactive way, in a setting where secure communication channels are available

[Ostrovsky 1991]. Their solution is based on the general paradigm of multiparty computation.

Using the proactive approach, repeated attacks from strong adversaries can be defended [Ostrovsky 1991]. As long as most parties are not compromised, the multi-party signature scheme remains secure. Most proactive signature methods are threshold schemes, which assume an honest majority of players and work on $n \geq 3$ players. Unfortunately, the longer the lifespan of the Public key, the more likely the threat. Proactive cryptosystems focus on the issue by allowing a potentially unbounded number of compromises, while a majority of them not happening simultaneously [Herzberg1997], [Ostrovsky 1991].

P2SS can also be seen as a *Key-insulated signature scheme* [Dodis 2002], in which the server helps the client to update its secrets periodically. The Public key remains the same, while the signature reflects the periods in which they were created. Thus the verifier can reject signatures produced in the period when the client was compromised. Server co-operation is not required for each signature. The client alone does the entire signature. In [Bellare 2001], Mihir Bellare and Ravi Sandhu presented the proof of security of two-party RSA.

In our scheme PUDE, all signatures go through the server and can be immediately recovered if compromised. In PUDE the actual secret does not change, only the key sharing is changed.

3.2 P2SS with RSA

The early work in this area was done by Antonio Nicolosi, who has used Schnorr's method to split the private key [Nicolosi 2003].

Schnorr's method requires a lot more computation than the method, which we are using. Moreover RSA has been used widely in industry and the NIST standard for digital signatures is largely based upon RSA. We are proposing a 1024 bit or larger key. With the experience gained from RSA we can safely say that our PUDE system will be quite secure.

In [Canetti 1994] the proactive approach as a security enhancement to centralized systems is considered, and a practical proactive pseudo-random generator with applications to secure sign-on is presented. Another basic task that has been 'proactivized' is secret sharing, and in particular verifiable secret sharing (i.e., secret sharing resilient against malicious faults) [Herzberg 1995]. This algorithm played a key role in proactive solutions for public-key cryptosystems, and in particular in proactive signature systems [Herzberg1997] (by extending the threshold signature of [Desmedt 1989]). Proactive solutions were found for the DSS signature algorithm [Gennaro 1996], [Herzberg1997] and for RSA [Frankel 1997a], [Frankel 1997]. They were used in [Canetti1997] to provide a proactive, automated solution to key refresh. [Canetti1997] shows how to use cryptography to ensure authenticated and secret communication among servers, with recovery from penetrations and key exposures. This provides an alternative to manual key refreshment.

Most previous work on two-party signature schemes focuses on generating the signature, which is capable of forming one party algorithm.

Let the client and the server each generate its own key pair-call them (Kc, Kc^{-1}) and (Ks, Ks^{-1}), respectively. The two-party public key then consists of the two public keys (Ks, Kc). A two-party signature of a message m consists of only two independent signatures of m using Kc^{-1} and Ks^{-1}. The first signature can be produced only by the client, and the second only by the server.

In a practical two party signatures scheme, M. Bellare and R. Sanhdu [Bellare 2001] and MacKenzie and Reiter [MacKenzie 2001] consider several variations on two-party generation of the RSA signatures, building on some previous less formal work, for example [Boyd 1989] and [GY1995]. The schemes are simple, elegant, and in most cases reducible to the basic RSA assumption. P. MacKenzie and M. Reiter [MR 2001a] also give a protocol for two-party generation of DSA signatures [FIPS 186]. Two-party signatures can also be viewed as a special case of general secure two-party computation [YAO 1986].

Chapter 4 Implementation of PUDE in SMS

4.1 Design of SMS

SMS has been designed for decentralized control and is a secure channel to transfer the information for authentication from client to server. The SMS client may register with a primary server in a particular realm. But it is able to access securely into multiple realms with a single sign-on.

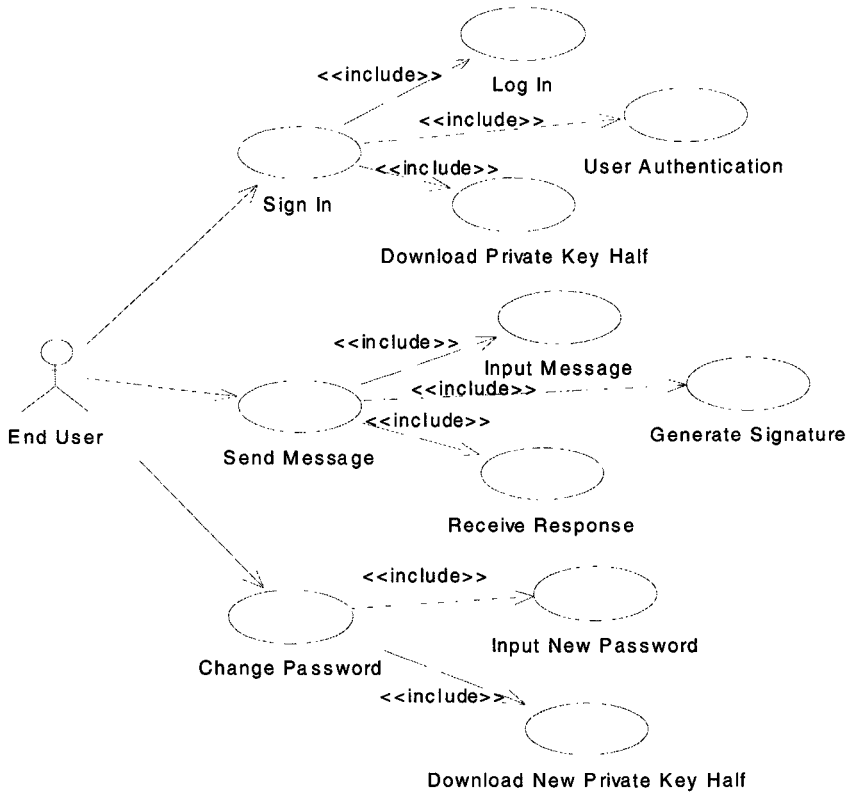


Figure 1 Use Case Diagram of SMS Client Side

Figure 1 illustrates the use cases of the SMS client side. In the requirement

view, there are only three use cases for end users: sign in, send message, and change password. In the design view, there are some sub user cases included in the each use case. The sign in use case includes log in, user authentication, and download private key half use cases. The send message consists of input message, generate signature, and receive response use cases. The change password use case comprises input new password and download new private key half use cases.

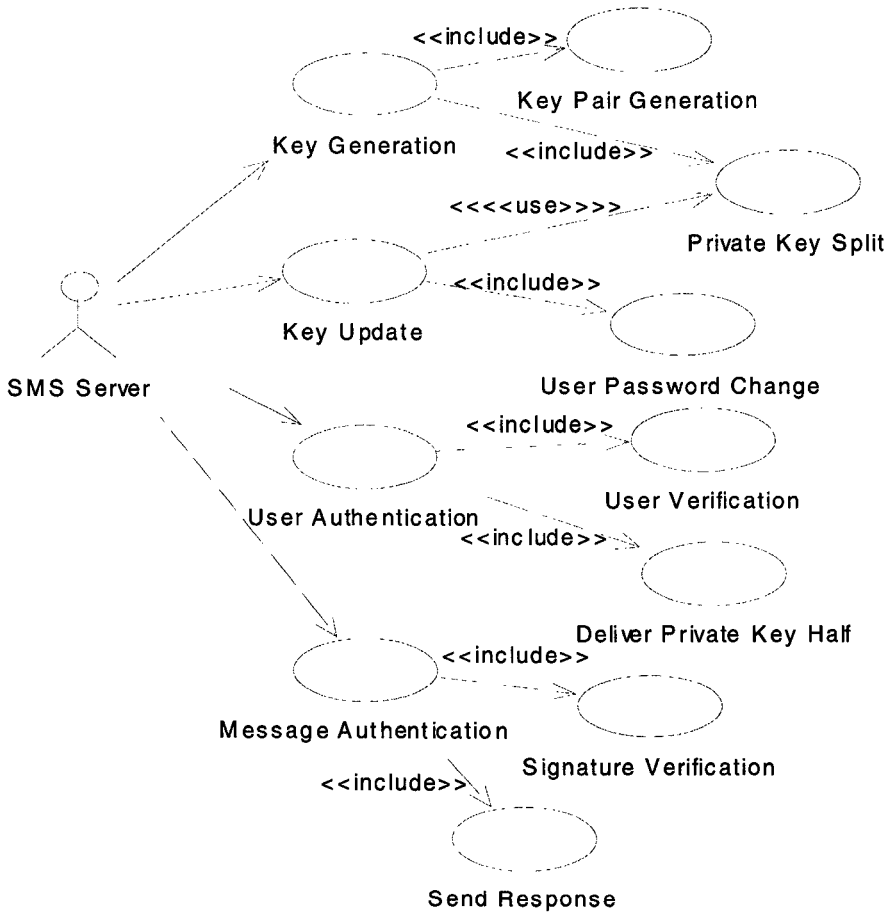


Figure 2 Use Case Diagram of SMS Server Side

Figure 2 shows the use cases of the SMS server side. There are four use cases on the server side: key generation, key update, use authentication, and message authentication, each of which can be divided into a couple of sub use cases. The key generation use case includes key pair generation and private key split use cases. The key update use case includes user password change and reuses the private key split use case. The user authentication use case is composed of user verification and delivers private key half use cases. The message authentication use case contains signature verification and send response use cases.

SMS environment is specifically designed as a test-bed for multi-realm authentication. In many respects, it is similar to the *Andrew File System* (AFS), including the authentication system where all passwords are verified with the main database. In our system the main database is in the default realm of the user. A user may belong to a particular realm. However cross realm authentication can be achieved in both. AFS is a distributed file system, which was designed to work over wide area networks. AFS is based on a distributed file system originally developed at the Information Technology Center at Carnegie-Mellon University [Howard 88].

Some useful features of AFS: In conjunction with *Kerberos* [Steiner 1988], AFS provides a global authentication system (all passwords are verified with a site-wide database). Access Control Lists (ACLs) provide more flexibility in setting file access permissions.

Users can access AFS files at remote sites, if given appropriate permissions [Howard 88]. Cross-realm authentication can also be achieved but requires the

co-operation from the administration of other realms. As it is based on Kerberos, it is not recommended to have the same password in different realms.

The closest to the SMS system is a part of Self-certifying File System (SFS) [Mazieres 2000]. Most secure file system come tightly coupled with a key management system that is similar to the popular secure shell (ssh) [Ylonen 96] or Kerberos [Steiner 1988]. SFS splits overall security into two pieces: file system security and key management. By registering the public key in different realms, a user digitally signs an authentication request with the corresponding private key; and the user can easily access the file from different realms without caring for administrative boundaries [Mazieres 2000].

The PUDE environment may be suitable for hardware based user-authentication using smart cards, as the cards can use their compute-power to easily calculate digital signatures.

4.1.1 Assumptions

The SMS system is implemented with the user authentication, based on the assumption that the users trust the servers. In other words, there is only one-way authentication, i.e. authenticating the messages from the users.

It is assumed that all the messages sent from the users are of small size.

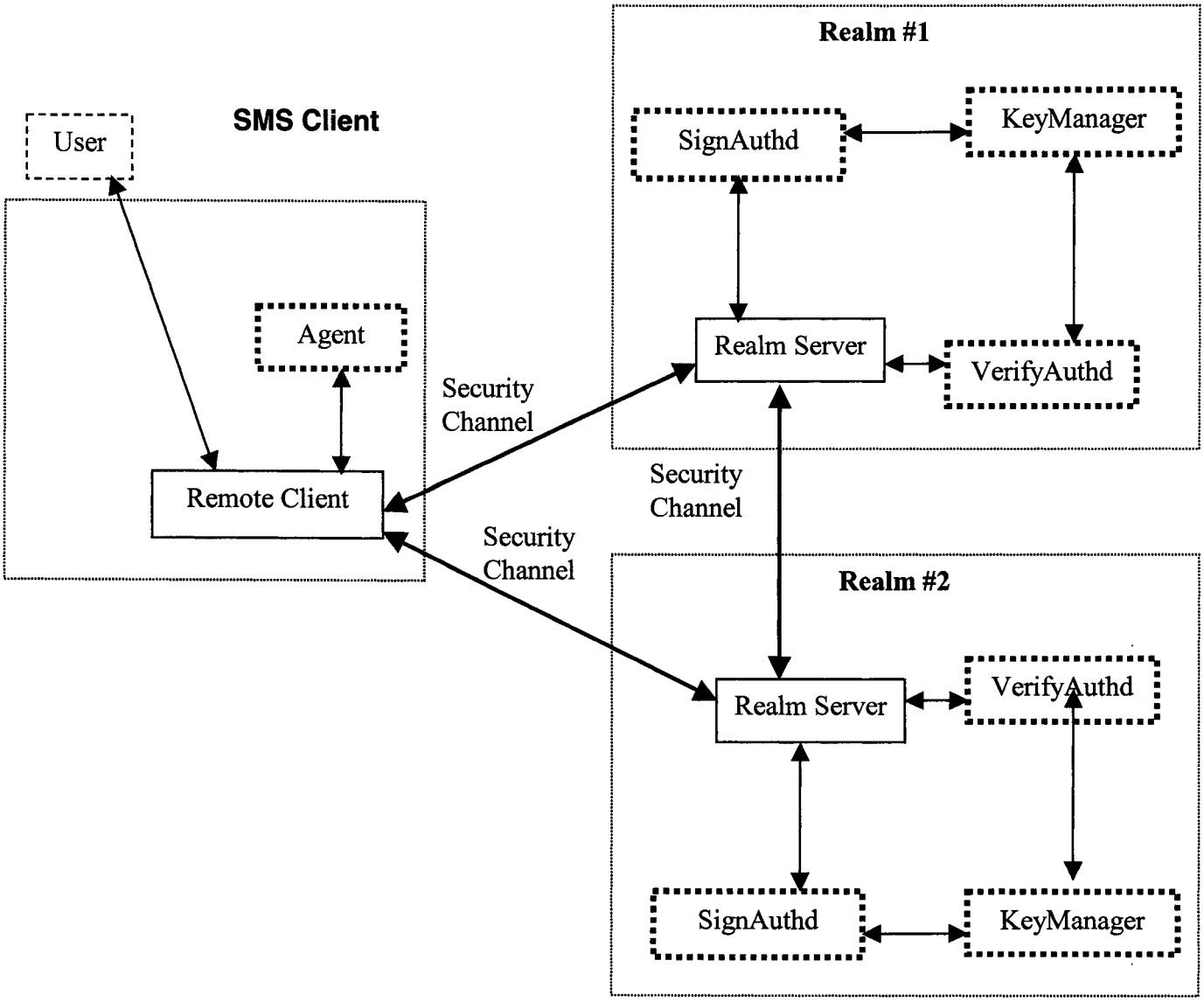


Figure 3: PUDE Architecture

From figure 3, we can see that the SMS is basically a client-server system in a distributed environment. Every user should register with one default Realm Server before the first-time sign in. After the successful sign in, the user obtains a

credential ID, which indicates that this user has been authenticated and can access the hosts within this realm and hosts within other realms. When the user accesses the hosts in the non-default realm, the user does not need to be authenticated again, by carrying the credential ID.

On the client side, the Remote Client is the essential component for communication, while the Agent is the major one taking care of all the authentication issues. The figure above just shows two realms for the purpose of illustration. Realm Server receives a request from the Remote Client, assigns the authentication work to SignAuthd or VerifyAuthd, and sends back the response to the Remote Client. The sign in authenticator, SignAuthd, is responsible for the user sign-in request. VerifyAuthd is a signature verifier used to authenticate the messages from users. The key factory and repository, Key Manager, works for key generation and key update.

The Security Channels provide a TCP connection with the traffic encrypted with Data Encryption Standard (DES). The DES, a Federal Information Processing Standard (FIPS) 46-2, is a very well publicly known cryptographic algorithm that converts plain text to cipher text and from cipher text to plain text using a 56-bit key.

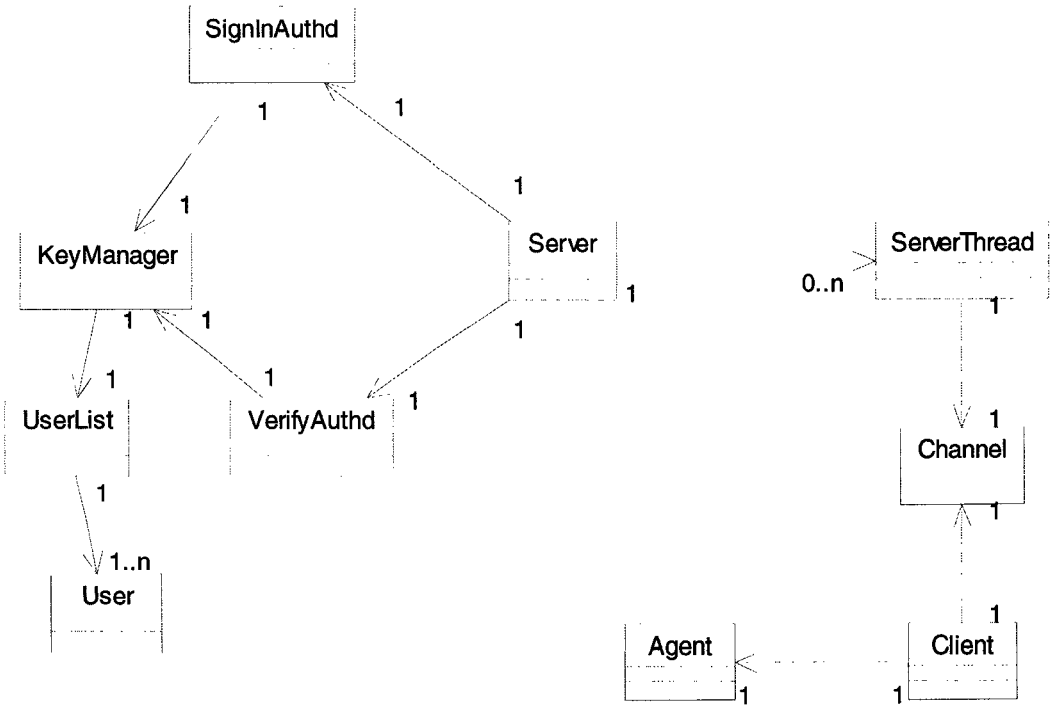


Figure 4: Conceptual Model of SMS

Ffigure 4 illustrates the entity classes in the SMS, their relationship and multiplicity. The roles that each of the entity classes plays are explained as below.

4.1.2 Remote Client

The Remote Client takes the commands from users, assigns the authentication tasks to the Agent, sends out the message through the Security Channel, and receives the messages through the Security Channel.

4.1.3 Agent

When the user signs in, the Agent generates a sign in with a random number along with the password, encrypts the user name, and then sends it through the Remote Client. After the user has been authenticated for the first-time sign in, the Agent gets the encrypted private key half from the Remote Client, decrypts the private key half with the user's password, and signs the messages being sent out by the Remote Client with the user's private key half. After the user changes the password and the new password is authenticated by the realm server, the Agent refreshes the private key half by downloading again from the Server.

4.1.4 Realm Server

The Realm Server validates the sign in request from the user, generates, restores and updates the users' keys and the related information, and receives the messages from the user and authenticates them.

4.1.5 SignAuthd

The SignAuthd validates the user's sign in requests by decrypting and verifying the user name with the related password and the updated salt using Password Based Encryption (PBE). As one of the public-key cryptography applications, PBE combines a password with a salt to

generate a secret key, which is in turn used to encrypt and decrypt the message data. The Appendix contains a brief description of PBE. The SignAuthd also retrieves the user's encrypted private key half from the KeyManager and sends it to the Realm Server for downloading to the client side.

4.1.6 VerifyAuthd

The VerifyAuth verifies the digital signature of the user by using the user's private key half for the server side, which is stored in the KeyManager. If a message is of a key update request, the VerifyAuthd retrieves the refreshed private key half from the KeyManager and sends it to the Realm Server.

4.1.7 KeyManager

The KeyManager generates the key pairs for users' digital signatures, selects a private key half for the client side and another half for the server side, updates the private key share as requested, maintains the mapping of users' public key and private key halves with users' IDs.

4.1.8 Security Channel

The Security Channel connects the Remote Client with the Realm Servers, and Realms Servers with each other over TCP. Each user sign-in session opens one channel. This channel is kept open for the whole of

the session. All messages being sent out need to be wrapped up, padded, and then encrypted with DES. All packages received need to be decrypted with DES and then unwrapped.

4.2.1 User Authentication

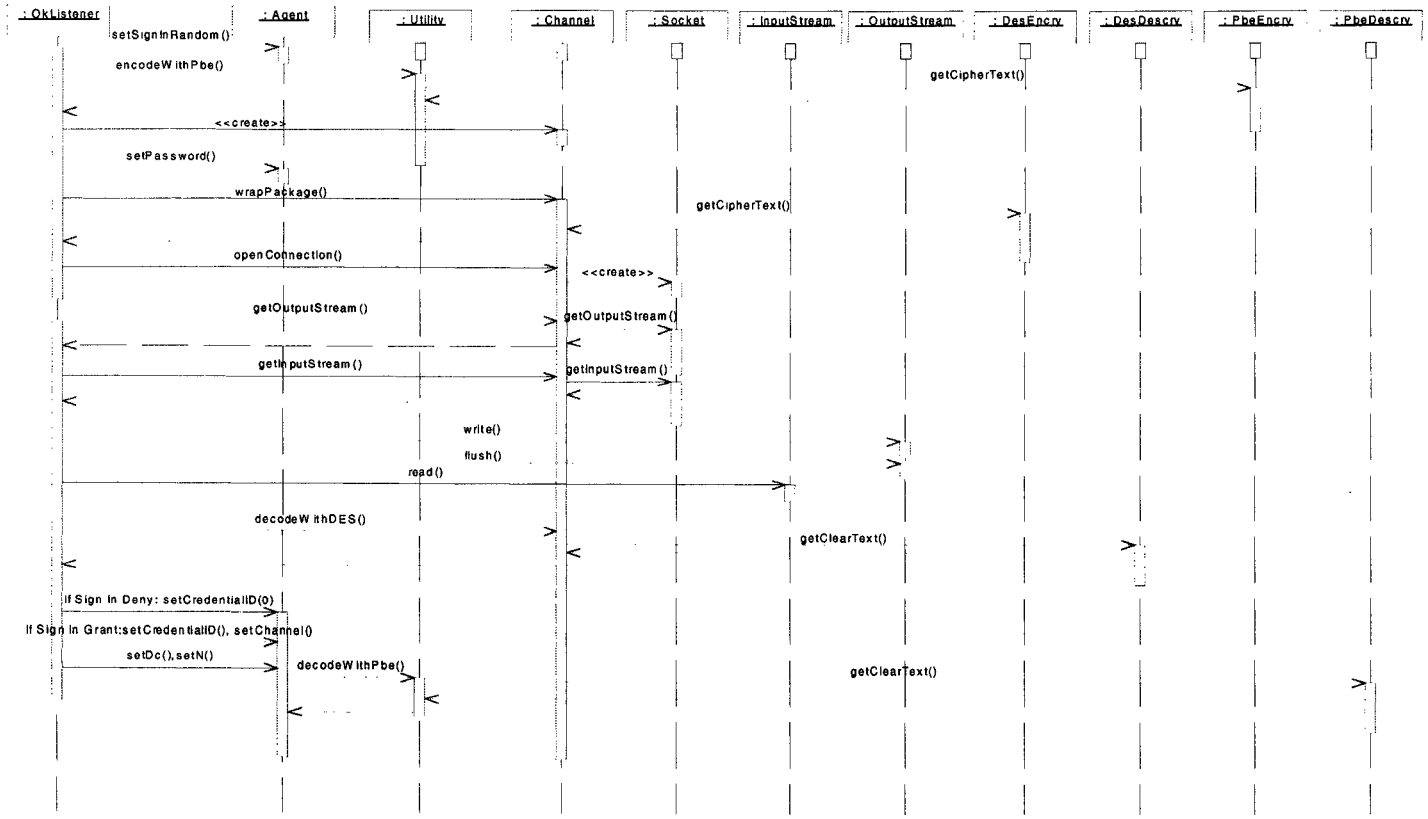


Figure 5: Sequence Diagram of Sign In from Client Side

Figure 5 illustrates the relations among the classes, which are used in sign in use case from the client side. In sign in use case, the OkListener class is responsible for performing the action after the user inputs the user name, password, server IP address etc. and clicks the OK button. A sign-in random number, used to generate the new salt for PBE, is created and set with the Agent. The user named is encrypted with the password using PBE. The user name clear text, user name cipher text, and the sign-in random number are encrypted and wrapped up in a package, which is going to be sent out to the server side. The OkListener class then gets the input stream and output stream from the Socket class and sends out the package. After the response package is retrieved from the server, the package is unwrapped. If the user sign-in is granted by the server, the user's private key half and credential ID are stored in the Agent class.

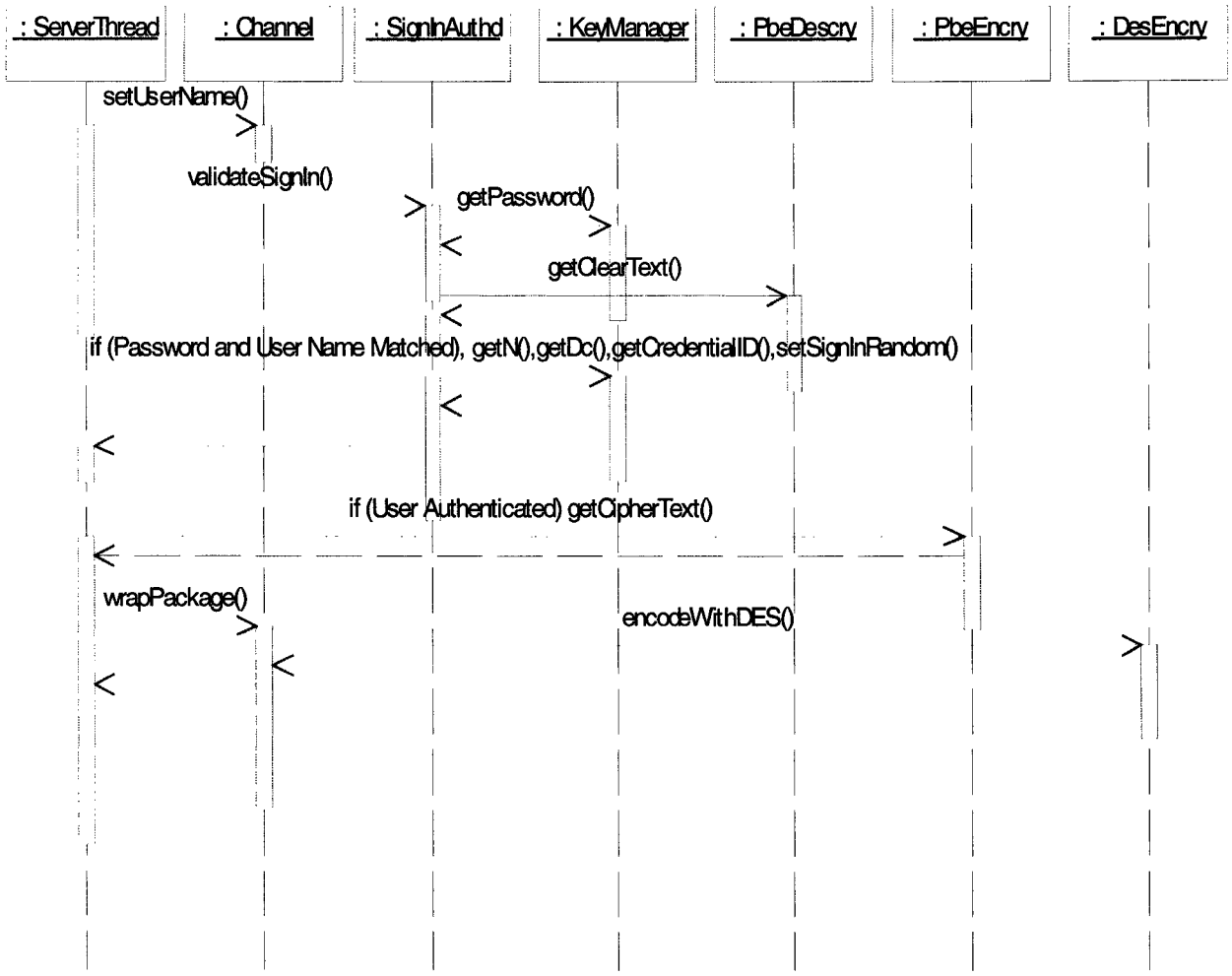


Figure 6: Sequence Diagram of User Authentication from Server Side

Figure 6 illustrates how the Realm Server authenticates the users by validating the password. The Server class creates one thread and one channel for each client, when the client sends out the user sign-in request. So the client is served by the ServerThread class. Since socket communication on the server side is the

same as on the client server side, there is no need to describe again the socket communication in the sequence diagram of the server side use case.

A user signs in to the default Realm Server by providing the realm ID, user ID and password. Once the Server class receives the open connection request from the client side, when the user first-time signs in, the Server class creates a new Channel object for this user and sets the user name with the Channel object. Then the Server assigns the sign in validation task to the SignAuthd class by calling the method validate Sign-In in the SignAuthd class. The SignAuthd retrieves the password from the KeyManager class and uses PbeDescry class to verify the the user name cipher. If verification is successful, the private key half (n, dc) is decrypted by calling PbeEncry class and wrapped up in the package and sent back to the client side. All these processes are illustrated in the following table.

Remote Client		Realm Server
• New Salt= Original Salt + Sign In Random Number • User Name Cipher=PE (User Name) • P1=E_{Kt}(User Name, User Name Cipher, Sign In Random Number)	P1 ---------->	• Selects p, q, then generates n,e, d_s,d_c,
		• P2=D_{Ks}(P1) • Extracts User Name Cipher • User Name from Client = PD (User Name Cipher) • Verifies User Name from Client ?=User Name from Server • If yes, sends out the encrypted n and d_c • N=PE(n), Dc=PE(d_c) P2= E_{Kt}(N, Dc)
	P2 <----------	

On the remote client side, the new salt is generated by adding the old salt with the newly created random number. The user name is encrypted with PBE. Then all the information is encrypted again with DES and packed in a package, which is sent out to the server side. After the Realm Server receives the package and

decrypted with DES, the user name cipher text is decrypted again via PBE with the user's password, which is stored in the KeyManager. If the user name clear text matches with the original user name, the server grants the user sign in. The private key half will be encrypted with PBE and put in the package. The package, is encrypted again with DES. This package will be sent back as a response to the user sign-in request.

4.2.2 Key Generation

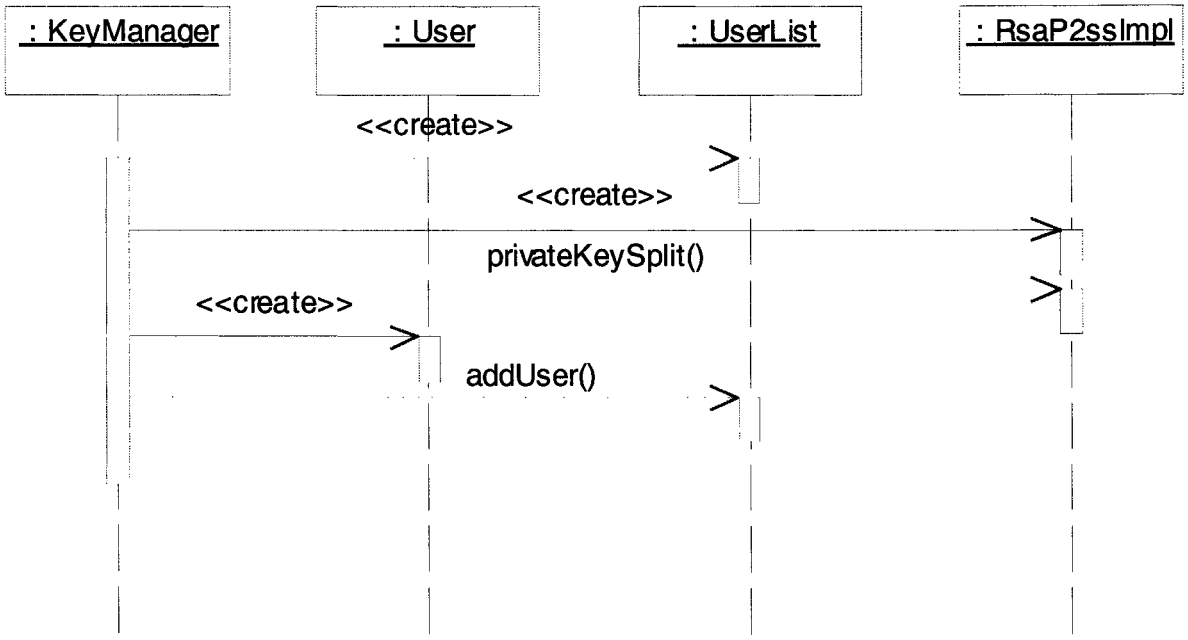


Figure 7: Sequence Diagram of Key Generation

Figure 7 shows the process of key generation, which is conducted by the KeyManger class on the server side. The KeyManager creates a UserList that is used to store the User objects, and a RsaP2ssImpl object that is used to

generate and split the keys for each User object. After the key pairs are generated and the private keys are split, the keys are stored in the User objects which are in turn stored in the UserList. In the whole SMS, there is only ONE public-private key pair for each user. The default Realm Server, with which a user registers, is responsible for computing the RSA public-private key pair: (n, e, d). Then the private exponent is split multiplicatively: $dcds \equiv d \pmod{\phi(n)}$. The default Prime Server selects a private half ds randomly from $Z^*\phi(n)$. Then the Server sets $dc \equiv d\ ds^{-1} \pmod{\phi(n)} \equiv ((d \bmod \phi(n))\ (ds^{-1} \pmod{\phi(n)})) \bmod \phi(n)$. If the private half of either side is ever compromised or exposed to any adversaries, it can be changed immediately on the primary or default server only. The original key won't change; only the share of that private key will be changed. In this way the system will select the new key from $Z^*\phi(n)$ and after selection and splitting of the key is done by the key manager, the private key will be downloaded on the client side whenever, after this process, it is successfully authenticated.

4.2.3 Signature Generation

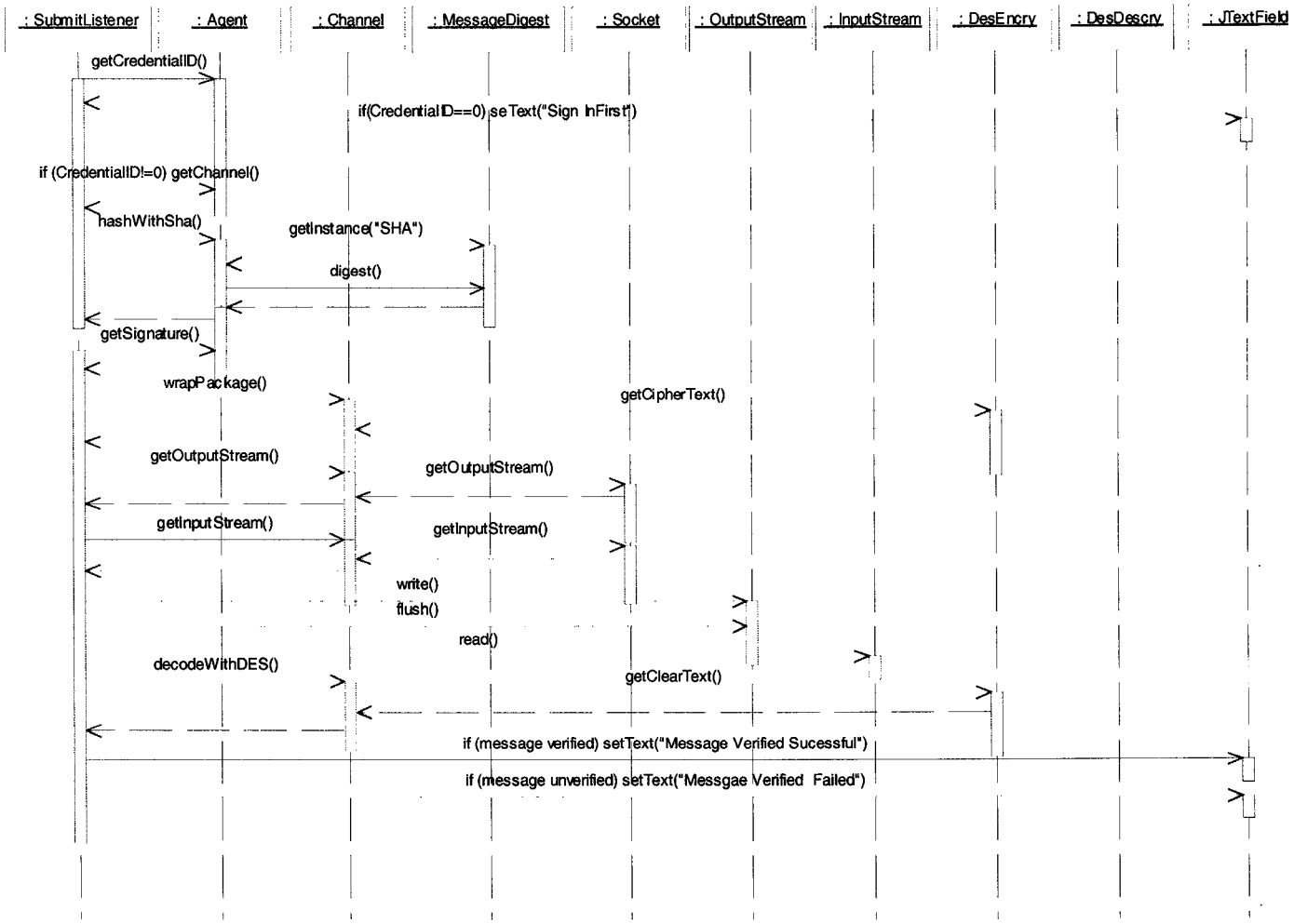


Figure 8: Sequence Diagram of Send Message

Figure 8 illustrates the process of signing a message that is going to be sent out by the client side. After the user first-time signs in and the user's private key half

is downloaded, every message sent should be signed with the this private key half based on RSA:

- y=H(M), y is the message digest based on SHA-1.
- $x_c=y^{dc} \mod (n)$, x is the user's signature.

After signing, the message digest y and signature x will be sent out via Security Channel. To accomplish this, the SubmitListener class will first check whether the user owns a credential ID, which indicates that this user's sign-in request has been granted by the server. If yes, the message will be hashed and signed via RSA with private key half. The message digest and the signature are sent to the server via Channel class. After the message and the signature are verified by the server, the response will be sent back to the client side.

4.2.4 Signature Verifier

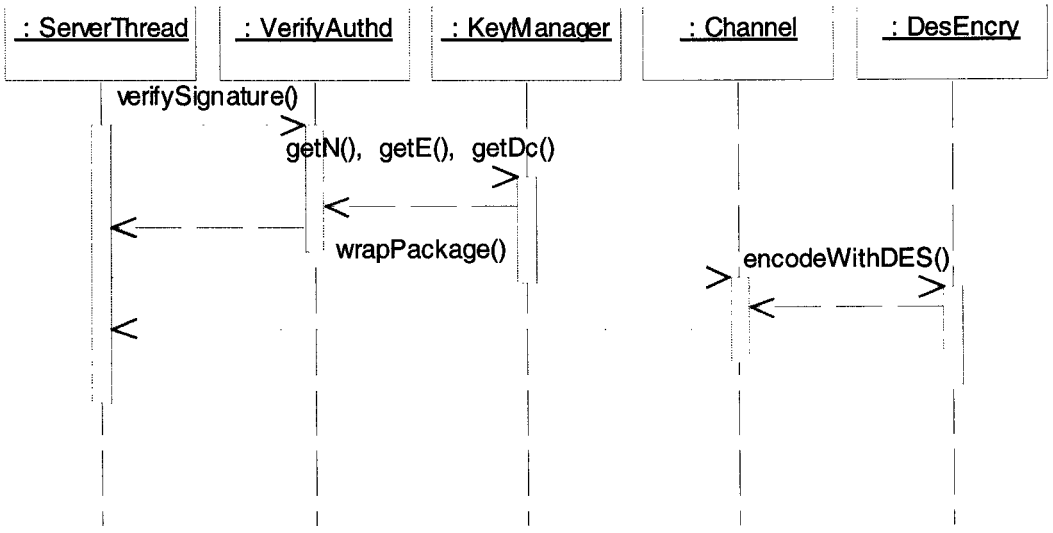


Figure 9: Sequence Diagram of Message Authentication

Figure 9 shows the process of verification of signature by the server. The ServerThread class assigns the signature verification task to the VerifyAuth class, which verifies the received message digest y with the signature attached x_c:

- $x=x_c^{ds} \bmod (n)$
- If $y = x^e \bmod (n)$, then the authentication is successful.

The n, e, and d_c used in the algorithm are retrieved by the VerifyAuth class from the KeyManger class. After the verification, the response package is encrypted and sent back to the client side.

4.2.5 Key Update

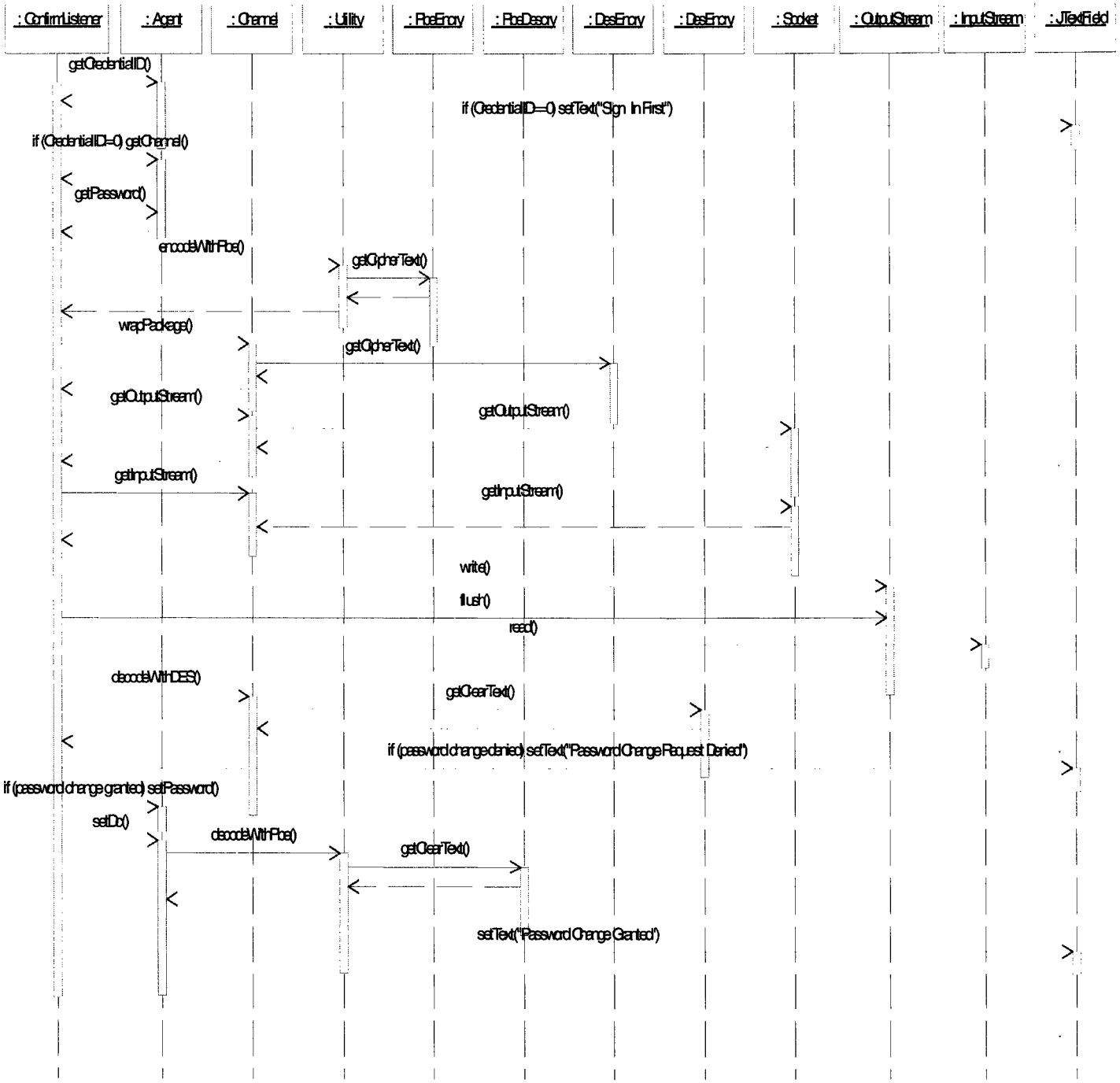


Figure 10: Sequence Diagram of Change Password from Client Side

Figure 10 illustrates the workflow of changing password form the client side. The ComfirmListener class will first check whether the user owns a credential ID, which indicates that this user's sign-in request has been granted by the server. If yes, the new password is encrypted with PBE and sent to the server. After the server validate the password change, the new private key half is downloaded to the client side via the encrypted response package.

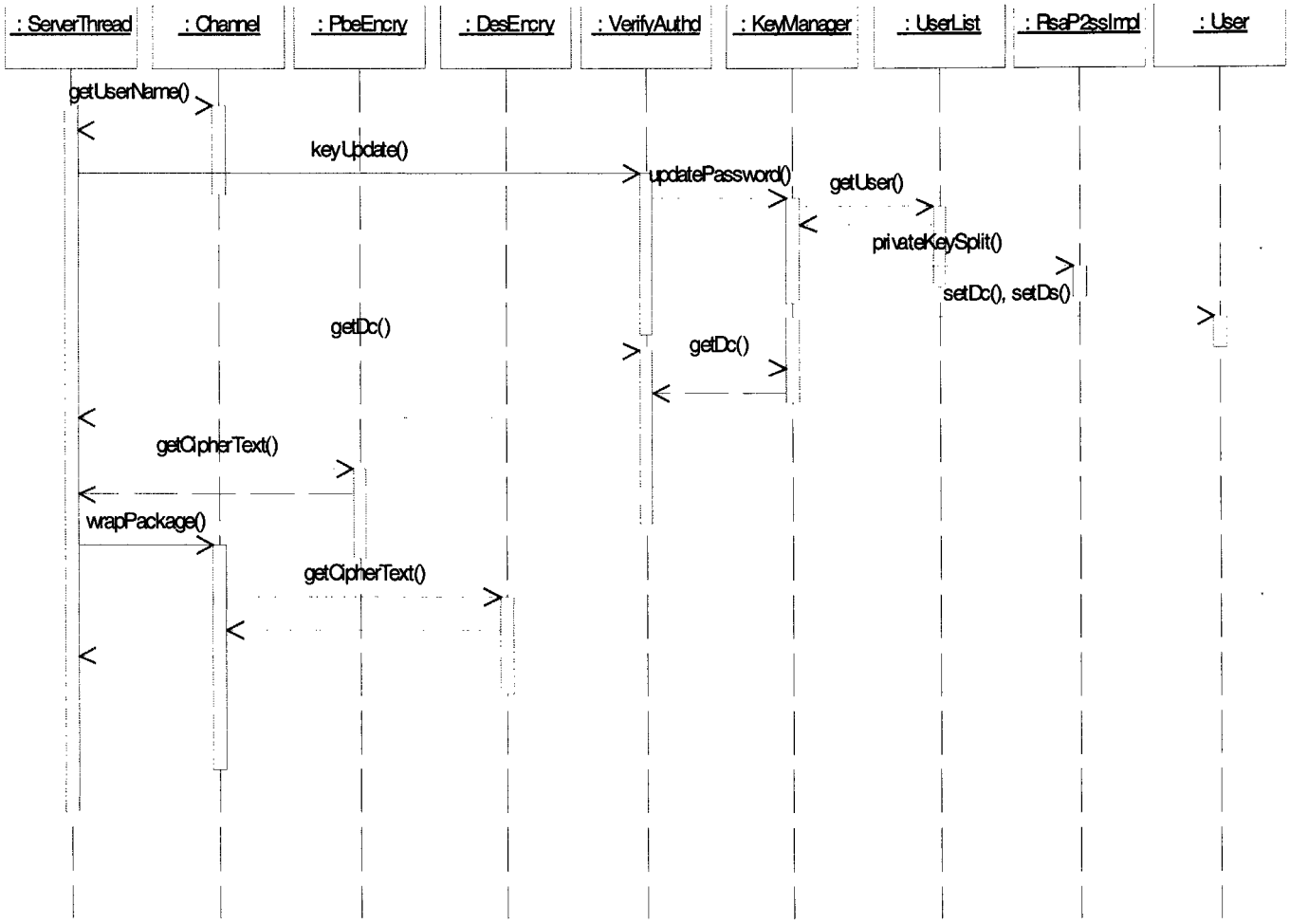


Figure 11: Sequence Diagram of Key Update from Server Side

Figure 11 shows the process of key update from the server side. The key update means the update of the private key share: d_c and d_s. The ServerThread class assigns the key update task to the VerifyAuth class by calling the keyUpdate method. The refreshed d_c is retrieved by the VerifyAuth from the KeyManager class, and eventually passed to the ServerThread class. The refreshed d_c is sent back to the client side. The whole private key d and public key e do not change. The event that the user changes the password triggers the key update on the default Realm Server: The default Realm Server chooses new d'_s randomly from $Z^*_{\phi(n)}$, then the server sets $d_c' \equiv d\ d_s'^{-1} (\mathrm{mod}\ \phi(n))$. After encrypting it with the new password, the refreshed private key half is sent to the user's machine. After putting it into the repository in the default Realm Server, the encrypted refreshed private key half and the new password are distributed to other Realm Servers in different realms.

4.2.6 Secure Channel

Every message between the client and the server, and between two servers, should be sent via a Secure Channel. The sender uses $M'=E_{Kt}[Pad(M)]$ to encrypt the package. The receiver uses $M=D_{Kt}(M')$ to decrypt the package. All the senders and receivers are holding the same Kt.

4.2.7 Package Format

TYPE 1: SIGNIN REQUEST

T	User Name Length	User Name	User Name Cipher Length	User Name Cipher	Sign In Random

TYPE 2: SIGN IN GRANT

T	N Cipher Length	N Cipher	d_c Cipher Length	d_c Cipher	Credential ID

TYPE 3: SIGN IN DENY

T

TYPE 4: MESSAGE OF CLIENT

T	Message Digest Length	Message Digest y	User Signature Length	User Signature x_c

TYPE 5: MESSAGE VERIFIED

T

TYPE 6: MESSAGE UNVERIFIED

T

TYPE 7: PASSWORD CHANGE REQUEST

T	New Password Cipher Length	New Password Cipher

TYPE 8: PASSWORD CHANGE GRANT

T	New d_c Cipher Length	New d_c Cipher

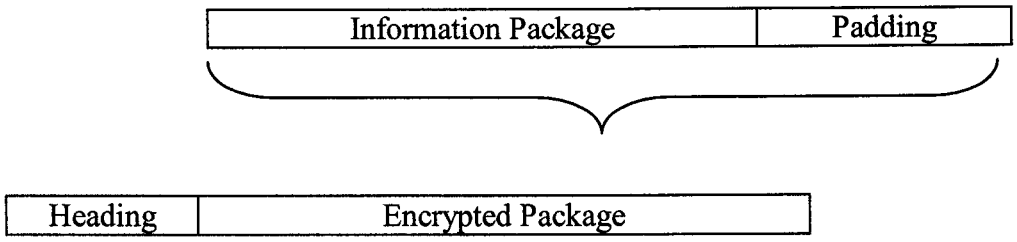
TYPE 9: PASSWORD CHANGE DENY

T

Notation:

- T---- type number, in 1 byte.
- Length---- in 2 bytes.
- Credential ID----in 1 byte.
- Sign In Random----in 8 bytes.
- Cipher---- Encrypted with PBE (Password-Based Encryption).

4.2.8 Package Wrapping



- An information package is padded with 0 to 7 bytes in order to make the number of bytes dividable by 8. It is the preparation for the DES encryption.
- The padded package is inserted with the 2-byte heading, which tells the length of the encrypted package.

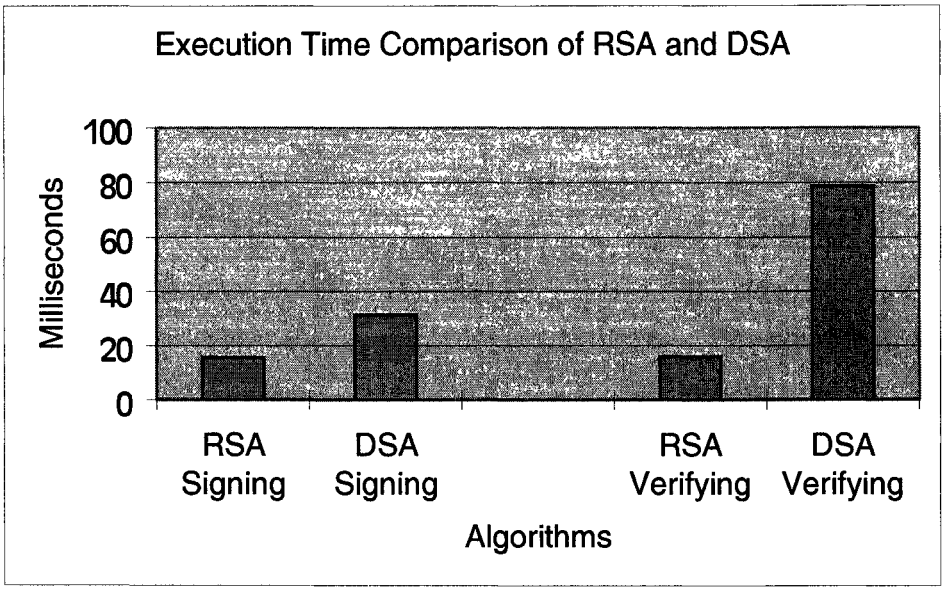
Chapter 5 Experiment and Performance Evaluation

This Chapter evaluates the performance of the 2-part RSA by comparing it with DSA, a variant of Schnorr algorithm. Our major concern is response times of signing the message and verifying the signature, respectively. Although generating the keys, splitting the keys and verifying the user login consume some system overhead, these jobs can be done before the users first sign in. Therefore there is very limited impact on the system response time.

In order to achieve the accuracy of the execution times of the algorithms, the SMS was run on a single PC to avoid the effect of network latency. The machine has 687 MHzs CPU, 384MB memory with Windows XP. The experiments were conducted applying 2-part RSA algorithm and DSA algorithm, respectively. Both algorithms used 1024-bit keys. Each time the execution time of signing the user message in the client side was measured. After the server side received the message and the signature, the execution time was gained by running the verifying component.

The data collected in the experiments is shown as follows:

Item	Message	Execution Time (Milliseconds)			
		2-part RSA Signing	DSA Signing	2-part RSA Verifying	DS
1	"hello world"	16	31	16	
2	"abc"	15	31	16	
3	"can you send me important message"	15	32	15	
Average		15.3	31.3	15.7	
2-part RSA/DSA		49%		20%	



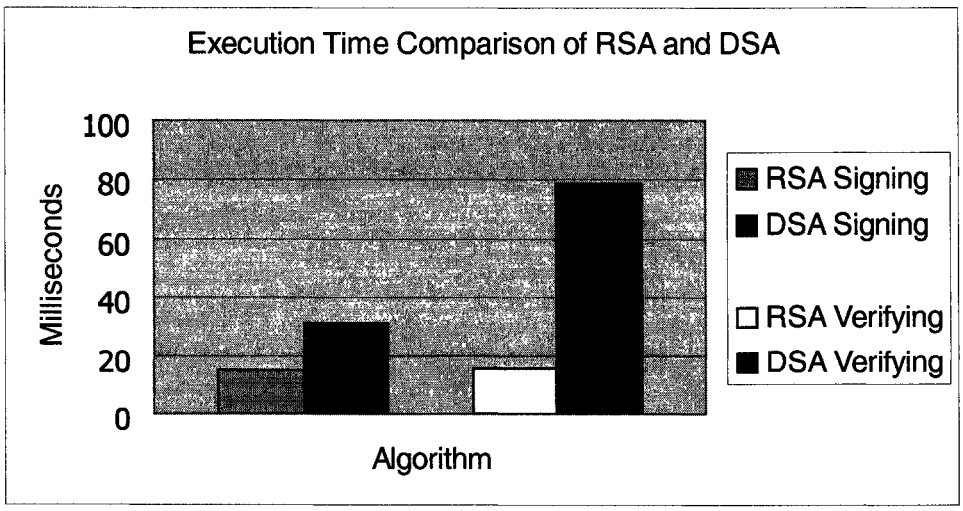


Figure 12: Execution Time Comparisons of 2-part RSA and DSA

From the data, we can draw the following conclusions:

- The computation of RSA signing and verifying is much faster than DSA. While RSA took less than half time in signing a message than DSA, DSA consumed five times of computing overhead as RSA in verifying the signature.
- There were no explicit relations between the length of messages and the execution time, both of signing and verifying. This can be explained by taking into account the hashing before the message was signed.

In addition, to improve the system responsiveness, the key generation and key splitting can be conducted before the first user sign in. Then when the user sends the request for first-time sign in, his/her own key-pair can be achieved immediately without the delay of key manipulation.

The DSA is more expensive than RSA in terms of execution overhead. Along with the consideration of the security , we concluded that compared with DSA algorithm, the 2-part RSA version is more efficient in terms of system responsiveness and security.

Chapter 6 Conclusion

Proactive security is the major requirement for today's distributed system, which have to face continuous attacks by hackers. We used P2SS to obtain PUDE, an effective tool to address the challenge of User Authentication in different realms.

In a PUDE system, refreshing the shared key would lessen the threat and the system would be able to recover if found to be corrupted. Even though after each update period, the sharing is independent of previous sharing, the shared function is always the same (For example: PUDE system that computes signatures, will use the same shared overall private key, and the same corresponding public key). If password is compromised, it can be changed immediately on the primary server only, which blocks the attacker immediately.

The Secure Multi-realm System, designed as a part of the thesis, uses decentralized control and a secure channel to transfer the information for authentication from client to server

An important characteristic of the threshold crypto-systems, that is preserved in our proactive setting, is that of being transparent to the user, who is receiving the public-key service.

My work shows that the two-party proactive signature, based on 2-part RSA algorithm, can be implemented securely in a multi-realm distributed system.

From the experimental results, we found that the DSA is more expensive than PUDE in terms of execution overhead. We, therefore, conclude that compared with DSA algorithm, the algorithm used in PUDE is more efficient in terms of system response time. Moreover it provides a greater security as discussed in the thesis.

References

[Bellovin 1989] S.M.Bellovin, Security problems in the TCP/IP protocol suite, *Computer communication review*, 19(2):32-48, April 1989.

[Bellovin 1995] S.M.Bellovin, Using the Domain Name System for System Break-ins. *Proceeding of the 5th Usenix UNIX security symposium*, June 1995

[Ben-Or 1988] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness Theorems for Noncryptographic Fault-Tolerant Distributed Computations. *In Proc.20th Annual Symposium. on the Theory of Computing*, pages 1-10, Chicago, IL USA, May 2--4, 1988. ACM,1988.

[Blakley 1979] G. R. Blakley. Safeguarding cryptographic keys. I*n Proc. AFIPS 1979 National Computer Conference*, pages 313-317. AFIPS, 1979.

[Boyd 1989] C. Boyd. Digital Multisignatures. *In IMA Conference on Cryptography and Coding*, pages 241–246. Oxford University Press, 1989.

[Brassard 1988] G. Brassard, D. Chaum and C. Crepeau, Minimum Disclosure Proofs of Knowledge, *Journal of Computing and System Sciences*, Vol. 37, No. 2, 1988, pp. 156-189.

[Bellare 2001] M. Bellare and R. Sandhu. The security of practical two-party RSA signature schemes. Available as IACR eprint archive Report 2001/060, July 2001.

[Canetti1997] R. Canetti, S. Halevi, and A. Herzberg. Maintaining Authenticated Communication in the Presence of Break-ins. *In Proc. 16th ACM Symposium. on Principles of Distributed computation.* ACM, 1997.

[Canetti 1994] R. Canetti and A. Herzberg. Maintaining Security in the Presence of Transient Faults. In *Advances in Cryptology*, pages 425{438, 1994. Springer-Verlag. Lecture Notes in Computer Science No. 839.

[Carrel 1997] D. Carrel and D. Harkins, The Resolution of ISAKMP with Oakley, Internet Draft draft-ietf-ipsec-isakmp-oakley-03.txt, March 1997.

[Chaum 1988] D. Chaum, C. Crepeau, and I. Damgard. Multiparty Unconditionally Secure Protocols. *In Proc. 20th Annual Symp. on the Theory of Computing*, pages 11-19. ACM, 1988.

[Denning 1982] D.E. Denning. Cryptography and Data Security. Addison-Wesley, 1982.

[Deering 1995] S. Deering and R. Hinden. Internet protocol version 6(Ipv6) specification. Request for comments (RFC) 1883, Internet Engineering Task Force (IETF) 1995.

[Desmedt 1989] Y. Desmedt and Y. Frankel. Threshold cryptosystems. In G. Brassard, editor, *Advances in Cryptology '89*, pages 307-315, 1989. Springer-Verlag. Lecture Notes in Computer Science No. 435.

[Dodis 2002] Y. Dodis, J. Katz, S. Xu, and M. Yung. Strong key-insulated signature schemes. Unpublished Manuscript, 2002.

[DSS 2002] Digital Signature Standards, NIST, January 2002, http://www.itl.nist.gov/fibspubs/fib186.htm.

[EFA 2002] Cryptographic Terminology, EFA, January 2002, http://www.efa.org.au/Issues/Crypto/crypto5.html.

[FIPS 186] FIPS 186. Digital Signature Standard. U.S. Department of commerce/N.I.S.T., National Technical Information Service, Springfield, VA, 1994.

[Francis 2002] L. Francis. The Mathematical Guts of RSA Encryption. January 2002. http://world.std.com/~franl/crypto/rsa-guts.html.

[Frankel 1997] Y. Frankel, P. Gemmell, P. Mackenzie, and M. Yung. Optimal resilience proactive public-key cryptosystems. *In Proc. 38th Annual Symp. on Foundations of Computer Science*. IEEE, 1997.

[Frankel 1997a] Y. Frankel, P. Gemmell, P. Mackenzie, and M. Yung. Proactive RSA. In B. Kaliski, editor, *Advances in Cryptology '97*, 1997.

[Gemmell 1997] P. Gemmell. An introduction to threshold cryptography. *Cryptobytes*, Winter 97, pages 7-12, 1997.

[Gennaro 1996] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Robust threshold DSS signatures. In Ueli Maurer, editor, *Advances in Cryptology*, Eurocrypt '96, pages354-371, 1996. Springer- Verlag. Lecture Notes in Computer Science No. 1070.

[Goldreich 1987] O. Goldreich, S. Micali, and A. Wigderson. How to Play Any Mental Game. *In Proc. 19th Annual Symp. on the Theory of Computing*, pages 218-229. ACM, 1987.

[Ganesan 1995] R. Ganesan. Yaksha: Augmenting kerberos with public-key cryptography. *In Proc. of the ISOC Network and Distributed Systems Security Symposium*, pages 132–143, 1995.

[Herzberg 1995] A. Herzberg, S. Jarecki, H. Krawczyk, and M. Yung. Proactive Secret Sharing. In D. Coppersmith, editor, *Advances in Cryptology'95*, pages 339{352, 1995. Springer-Verlag Lecture.

[Herzberg1997] A. Herzberg, M. Jacobsson, S. Jarecki, H. Krawczyk, and M. Yung. Proactive public key and signature schemes. *In 4th ACM Conference on computer and Communication Security*, pages 100–110. ACM, 1997.

[Howard 1988] J. H. Howard, M. L. Kazar, S. G. Menees, D. A. Nichols, M. Satyanarayanan, R. N. Sidebotham, and M. J. West. Scale and performance in a distributed file system. *ACM Transactions on Computer Systems*, 6(1):51–81, February 1988.

[Kent 1998] S. Kent and R. Atkinson. Security architecture for the Internet Protocol, RFC-2401. Internet Engineering Task Force(IETF) 1998.

[Mazieres 2000] D. Mazieres. Self – Certifying File System (*Ph. D Thesis*), Massachusetts Institute of Technology 2000.

[MacKenzie 2001] P. MacKenzie and M. Reiter. Networked cryptographic devices resilient to capture. *In Proc.of the 2001 IEEE Symposium on Security and Privacy*, 2001.

[MacKenzie 2001a] P. MacKenzie and M. Reiter. Two-party generation of DSA Signature. *In Advances in Cryptology—Crypto'01*, volume 2139 of Lecture Notes in Computer Science, pages 137–154, Berlin, 2001. Springer-Verlag.

[Menezes 1997] A. Menezes, P. Ooschot. HandBook of Applied Cryptography. CRC Press LLC, 1997.

[Morris 1979] Robert Morris and Ken Thompson. Password security: A case history. Communications of the ACM, 22(11):594-597, November 1979.

[Murdoch 2002] M.T. Murdoch, Introduction to cryptography, Part 2: Symmetric Cryptography, IBM, January 2002. http://www-106.ibm.com/developworks/security/library/s-crypt02.html.

[Nicolosi 2003] A. Nicolosi, M. Krohn, Y. Dodis, D. Mazieres. Proactive two party Signatures for User Authentication. *The 10th Annual Network and Distributed System Security Symposium.* 2003.

[Ostrovsky 1991] R. Ostrovsky, M. Yung. How to withstand mobile virus attacks. *In Proceedings of the 10th Annual ACM Symposium on Principles of Distributed Computing*, pages 51–59, 1991.

[PKCS 2000] PKCS #5: Password-Based Cryptography Specification Version 2.0, RFC 2898, September 2000.

[Postel 1981] J. Postel. Internet Protocol. Request for Comments (RFC) 791, Internet Engineering Task Force, September 1981. ftp://ds.internic.net/rfc/rfc791.txt.

[RSA 1978] R. L. Rivest, A. Shamir, L. Adleman Public key cryptography, CACM 21, 120-126, 1978.

[RSA 1999] RSA Code-Breaking Contest Again Won by Distributed.Net and Electronic Frontier Foundation (EFF). January 1999. http://www.rsasecurity.com/company/news/releases/pr.asp?doc_id=462

[Schneier 1996] B. Schneier, *Applied Cryptography,* Wiley, 1996.

[Shamir 1979] A. Shamir. How to Share a Secret. *Communications of the ACM*, 22:612-613, 1979.

[Steffen 2002] D.Steffen. Asymmetric Cryptography. January, 2002. http://www.maths.mq.edu.au/~steffen/old/PCry/report/node8.html.

[Steffen 2002a] D. Steffen. Symmetric Cryptography. January, 2002. http://www.maths.mq.edu.au/~steffen/old/PCry/report/node7.html.

[Steffen 2002b] D. Steffen. Hash Functions, January 2002. http://www.maths.mq.edu.au/~steffen/old/Pcry/report/node9.html.

[Steiner 1988]J. G. Steiner, B. C. Neuman, and J. I. Schiller. Kerberos: An authentication service for open network systems. In Proceedings of the Winter 1988 USENIX, pages 191–202, Dallas, TX, February 1988. USENIX.

[Stallings 1999] W. Stallings. Cryptography and Network Security, 2nd Edition, Prentice-Hall, Inc,1999.

[Woo 1992] T. Y. C. Woo, S.L. Simon. Authentication for distributed system, *Computer*, 25(1):39–52, January 1992. IEEE Computer Society Press.

[Yao 1982] A. Yao, Protocols for Secure Computation, *In Proc. 23rd Annual Symp. on Foundations of Computer Science*, pages 160-164. IEEE, 1982.

[Yao 1986] A. Yao. How to generate and exchange secrets. *In Proc. of the 27th IEEE Symposium on Foundations of Computer Science*, pages 162–167, 1986.

[Ylonen 1996] T. Ylonen. SSH – secure login connections over the Internet. *In Proc. of the 6th USENIX Security Symposium*, pages 37–42, San Jose, CA, July 1996.

Appendix A Introduction to Password Based Encryption (PBE)

Public Key Cryptography Standards # 5 (PKCS #5) defines one of the applications of public-key cryptography, Password Based Encryption (PBE). In PBE a secret key is generated based on a password and a random number. This randomly generated number is called the salt.. And this secret key is used to encrypt and decrypt the message data.

The salt in PBE is used to produce a large set of keys corresponding to a given password. Therefore the salt can be viewed as an index into the keys table. This approach makes it difficult for opponents to compute the secret even if the password is compromised. Another purpose of using salt is to avoid the duplicate secret key, while using the identical password twice because the collision between keys can be significantly reduced by applying salt.

In PBE, an iteration count is also used to increase the difficulty of attack by increasing the cost of searching the password without giving noticeable impost on the computation of the secret key. PKCS #5 recommends 1000 iterations as the minimum. [PKCS 2000]

Appendix B Notation used in SMS

p,q	Both are prime numbers selected by the server.
n	A large number where n=pq. All computations are performed modulo n.
$\phi(n)$	The Euler totient function, which is the number of positive integers less than n and relatively prime to n. It has been proved that $\phi(n)=\phi(pq)=(p-1)(q-1)$.
$Z_{\phi(n)}$	The integers modulo n. It is the set of integer $\{0,1,2,\ldots, (\phi(n)-1)\}$.
$Z^*_{\phi(n)}$	The multiplicative group of $Z_{\phi(n)}$.
P	A user's password.
H ( )	A One-way hash function.
$E_k()$	Encryption with secret key based on DES.
$D_k()$	Decryption with secret key based on DES.
PE()	Password-Based Encryption with password, salt, and iteration.
PD()	Password-Based Decryption with password, salt, and iteration.
Pad()	Message padding, to make the length of message, before encryption, divisible by 8.
gcd(a,b)	The greatest common divider of a and b.
M	A user's message.
d	An RSA encryption exponent of the user's private key.
d_c	The private key half used by the user.
d_s	The private key half used by the server.
e	An RSA decryption exponent of the user's public key.
K_t	A transmission key, shared by both client and server, used to encrypt and decrypt the messages transferred in the Security Channel.

Glossary

DES: The Data Encrytion Standard is known as the Data Encryption Algorithm(DEA) by ANSI. DES is a block cipher; it encrypts data in 64-bits block. DES is a symmetric algorithm. [Stallings 1999]

Session Key: A temporary encryption key used between two principals. [Stallings 1999]

SSL: Secure Sockets Layer. The primary goal of the SSL Protocol is to provide privacy and reliability between two communicating applications. [Stallings 1999]

TLS: Transport Layer Security. The primary goal of the TLS Protocol is to provide privacy and data integrity between two communicating applications. [Stallings 1999]

Vita Auctoris

Arshad Ahmed Shaikh was born in Pakistan. He graduated from Mehran University Of Engineering and Technology in 1994 with a Bachelor of Engineering degree in Computer Engineering. In September 2000, Arshad joined the Master of Science program in Computer Science at the University of Windsor, Windsor, Ontario, Canada.