ADDIS ABABA UNIVERSITY
SCHOOL OF GRADUATE STUDIES
FACULTY OF TECHNOLOGY

Department of
Electrical and Computer Engineering

Analysis of the Key Exchange method of SSH using Elliptic Curve Cryptography and a Public Key Infrastructure

A Thesis Submitted to School of Graduate Studies of
Addis Ababa University in Partial Fulfillment of the
requirement of The Degree of
Master of Science In Computer Engineering.

By
Banchi Hailu
Advisor
Prof. Dr. DP.Roy

February 2008
Addis Ababa, Ethiopia.

ADDIS ABABA UNIVERSITY

SCHOOL OF GRADUATE STUDIES

Analysis of the Key Exchange method of SSH
using Elliptic Curve Cryptography and a Public Key Infrastructure

by
Banchi Hailu

FACULTY OF TECHNOLOGY

APPROVAL OF BOARD OF EXAMINERS

_____	_____
Chairman Department of Graduate Committee	**signature**
_____	_____
Advisor	**signature**
_____	_____
Examiner	**signature**
_____	_____
External examiner	**signature**

Acknowledgement

I would like to thank my advisor Prof. Dr. DP.Roy for the suggestions, ideas, and advices during the development of this thesis. Especially, I thank him for his consistent guidance and encouragement throughout my work. The achievement of this thesis work would not have been possible with out his help.

I am very grateful to Dr. Mengesha, Head-ECE Department for his all cooperations to complete my thesis. I would also like to thank to the staff of our department office.

Finally I like to acknowledge the support and constant cooperation received from my family and friends.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Acronyms

AD	Authenticated Dictionaries
CA	Certification Authority
CRL	Certificate Revocation List
CP	Certificate Policy
DES	Data Encryption Standard
DH	Diffie Hellman
DSA	Digital Signature Algorithm
DSS	Digital Signature Standard
ECC	Elliptic Curve Cryptography
ECDH	Elliptic Curve Diffie-Hellman
ECDSA	Elliptic Curve Digital Signature Algorithm
HMAC	Hash-based Message Authentication Code
KDC	Key Distribution Center
KEX	Key Exchange
LDAP	Light Weight Directory Access Protocol
OCSP	Online Certificate Status Protocol
PKI	Public-Key Infrastructure
RA	Registration Authority
RSA	Rivest Shamir Adleman
SHA-1	Secure Hash Algorithm (revised)
SSH	Secured Shell

Abstract

SSH, Secure Shell, is a protocol that allows user to log into another computer, to execute commands in a remote machine, and to move files from one machine to another securely over an insecure network. It provides cryptographic authentication, encryption and data integrity to secure network communications. Negotiation of the security parameters and authentication of the peers require using public key cryptosystems. Public key operations are generally slow. In order to improve the performance of the protocol and make it applicable in both powerful and resource constrained environments Elliptic Curve Cryptography is used.

In addition, since SSH uses plain public keys to authenticate a remote server, always the first time authentication is vulnerable to the Man-in-the-Middle attack. Using a public key certificate as a host key will eliminate the above vulnerability. And it requires a PKI, Public Key Infrastructure to support the certificate approach. PKI may potentially impact the performance of the security protocol. And PKI path validation techniques (certificate revocation status checking) need more storage capacity, more communication cost and more processing time. This seems to have a problem to scale with large communicating nodes.

In this thesis, SSH's key exchange handshake is implemented using java and bouncy castle cryptographic api.

Performance with RSA (Rivest-Shamir-Adleman) and ECDH_ECDSA (Elliptic Curve Diffie-Hellman Elliptic Curve Digital Signature Algorithm) key exchange suites have been compared for both PKI and non-PKI authentication. Client waiting time (key exchange latency), server key exchange throughput, and revocation status message size have been measured for each key exchange suite.

Simulation results show that ECC has better processing time performance and better throughput than RSA. Response time and revocation status message size are minimum when Authenticated Directories are used as a certificate status responder.

Keywords used: SSH, PKI, Elliptic Curve Cryptography, ECDH, ECDSA, certificate, certificate path validation, certificate revocation status checking, key exchange handshake, authentication, Authenticated Dictionaries and RSA.

Chapter 1
Introduction

1.1 SSH Overview

As Internet is becoming increasingly relatively inexpensive and available, it has become a viable replacement for telephone, and fax, as well as remote dial-up access to a company's internal computer resources.

Much of the data that we send travels on the Internet or local networks is transmitted as plain text. The email we send, the files we transmit between computers, even the passwords we type may be readable by others. Sensitive data transmitted will be left open to different types of attacks.

And one of the biggest concerns about using the electronic world is internet security.
The purpose of information and network security is to provide availability, integrity, and confidentiality.

Secure Shell is a protocol that provides authentication, encryption and data integrity to secure network communications. Implementations of Secure Shell offer the following capabilities: a secure command-shell, secure file transfer, and remote access to a variety of TCP/IP applications via a secure tunnel (secure remote login).
There are two incompatible SSH protocols, the SSH1 protocol and the SSH2 protocol.
Generally SSH2 is considered to be more secure as it avoids the known vulnerabilities of SSH1.
SSH2 employs a public-key cryptosystem to derive symmetric-keys and then use fast symmetric-key algorithms to ensure confidentiality, integrity and source authentication of bulk data.
SSH2 uses public key authentication (DSA and RSA) and deffie-hellman-kex-sha1-group1 to authenticate and create a shared secret key.

SSH2 permits the use of digital certificates for host authentication. Digital certificates are issued by trusted Certification Authorities (CA). They contain identity information of the host together with the public key to be used during the authentication. Authentication of the host requires verification of its certificate using the public key retrieved from the CA certificate.

1.2 Motivation

Today, there is an explosive growth in the amount of sensitive data exchange over the internet. And more significant number of internet hosts are mobile and wireless (resource constrained).
Given the above two things, there is a clear need for efficient, scalable security mechanisms and protocols that operate well in all environments.

SSH2 uses a public key cryptosystems to derive symmetric keys and then use faster symmetric-key algorithms to ensure confidentiality, integrity and source authentication of bulk-data. It employs RSA and DSA public-key cryptosystems for that. Because of plain public-key usage for server authentication the protocol is vulnerable to the Man-in-the-Middle Attack. So certificate authentication is needed to avoid that problem.

And security of the protocol is as good as that of its weakest component; thus the work factor needed to break the symmetric key must match that needed to break the public-key cryptosystem used for key-establishment. Due to expected advances in cryptanalysts and increase in computing power available to an adversary, both symmetric and public-key sizes must grow over time to offer acceptable security for a fixed protection life span. The processing time significantly increases as the larger key sizes are used.
Table 1 [18] shows this expected key-size growth for various symmetric and public-key cryptosystems.
Table 1: Computationally equivalent key sizes

Symmetric	ECC	RSA/DH/DSA
80	163	1024
128	283	3072
192	409	7680
256	571	15360

As shown in Table 1, the Elliptic Curve Cryptosystem (ECC) offers the highest strength per bit of any known public-key cryptosystem today. ECC not only uses smaller keys for equivalent strength compared to traditional public-key cryptosystems like RSA, the key size disparity grows as security needs increase. This makes it especially attractive for constrained wireless devices because smaller keys result in power, bandwidth and computational savings.

Considering the low processing power of most internet hosts like handheld devices, it may be reasonable to restrict the key sizes. Therefore, evaluating the performance of the protocol for different key exchange cipher suites and proposing a better solution is valuable.

This thesis tries to integrate ECC in to the Secure Shell Protocol and evaluate the performance impact. In addition thesis also tries to integrate certificate authentication and analyze the different PKI revocation mechanisms and use a scalable revocation scheme.

1.3 Objective

1.3.1 General objective

- The main objective of this thesis is to study the key exchange method of SSH and to improve its security level with better scalability to be applicable on both powerful and resource constrained internet hosts using ECC and PKI.

1.3.2 Specific objective

1. To study the existing public key cryptosystems employed by SSH2 and to evaluate their performance when used during SSH key exchange phase

2. To study the usage of PKI on SSH so that to avoid the Man-in-the-Middle Attack.
3. To study and analyze the different PKI revocation mechanisms. And add Authenticated Dictionaries as an online certificate status responders and see the performance impact.
4. To integrate ECC based key exchange mechanism in to the Secure Shell protocol and investigate its performance impact

1.4 Related work

1.4.1 A third party archive that stores hostnames and their corresponding public key hashes

Yasir Ali[14] proposed a third party archive that stores hostnames and their corresponding public key hashes (public key hashes)

Here a database is employed to store server hostnames and their corresponding public key hashes. A keyed MAC—a "poor man's digital signature" also known as a keyed hash algorithm is used to generate the hashes. HMAC provides both data integrity and data origin authentication for files sent between two users. HMAC is like the other hash algorithms but it also includes a shared secret key i.e. the client's password.
The server prior to any connection generates its public key hash taking as input the client's password and its public key and populates the archive. A client wishing to make connection with a particular server first will retrieve the server's public key hash using its password as input and will latter on compare this hash with the hash of the public key that the server will send to identify itself and if the hash value generated is the same as the one the client already retrieved, he/she can be certain that the sever key is the correct one. A keyed MAC algorithm takes a message M and a secret key k, and produces a mac value Mac(M; k) with the property that it is believed infeasible to find another M0; k0 pair that generate the same keyed MAC. Thus, if Bob knows secret key k, and retrieves a message M accompanied by a MAC value h which he confirms as Mac(M; k), then Bob can conclude that M was produced by a party that knew k and has not been altered in transit.

A keyed-hash generation tool is required on both the client and server machines.

Disadvantages

- The server needs to generate separate public key hashes for each client. This might make the database hard to maintain.
- When the server's public key is compromised the connection will be open for (spoofing) attack till the server realizes and change the hash, the client would be vulnerable to a replay attack.
- Plain public keys lack enough information to properly identify a particular server.
- Revocation is not possible and no point of trust (risk managment).

1.4.2 ECC in SSL

Gupta et. al. present an estimation of the performance improvements that can be expected in SSL protocol by adding ECC support in [19]. They modify the OpenSSL[21] cryptographic library OpenSSL0.9.6.b to support ECDH, ECDSA and X.509 certificates with ECC parameters. The analytical model especially considers the handshake crypto latency and server crypto throughput however they are also aware of the extra delays due to message parsing, hashing and network latency. RSA (1024-2048bit) and ECDH_ECDSA (163-193 bit) handshakes are compared in three cases. One of these three cases is a client talking to an Ultra 80 server simulating a wireless web scenario. They measure the performance of public key algorithms for RSA encrypt, verify, decrypt and sign, ECDSA verify and sign operations and use these values at their analytical model. The results show that 1024 bit RSA performs better than 163 bit ECC curve while 193 bit ECC curve is faster then 2048 bit RSA for server authenticated SSL handshake. ECDH key agreement is faster for both key sizes when mutual authenticated SSL handshake is performed.

1.5 Methodology

1. Study of literature on the topics
2. Analyzing the existing public key authentication and key-exchange algorithms
3. Integrating certificate authentication

4. examining the existing key-exchange scheme for alternative scalable cryptographic operations
5. developing a sample application program to analyze the performance of the different methods employed

1.6 Thesis organization

The outline of the thesis is as follows.

Chapter 2 describes the theoretical background and operational concepts of SSH.

Chapter 3 deals with public key cryptography and PKI in details.

Chapter 4 deals with the thesis work, modification of the key-exchange method of SSH2 by integrating ECC in to it, using X.509Certificates for authentication of SSH servers, and also using another method of certificate revocation status checking.

Chapter 5 deals with the simulation and analysis of the results of the simulation.

Chapter 6 concludes the thesis work and includes the recommendation for future work.

Chapter 2
Background of SSH2

2.1 The SSH protocol

SSH is a protocol for secure remote login and other secure network services over an insecure network [1].

SSH is intended to run over a reliable transport protocol, such as TCP. There are two versions of SSH, imaginatively called SSH1 and SSH2. Use of SSH1 is deprecated because of some security problems.

SSH2 has been separated into modules and consists of three protocols working together:

- SSH Transport Layer Protocol (SSH-TRANS)
- SSH Authentication Protocol (SSH-AUTH)
- SSH Connection Protocol (SSH-CONN)

SSH is designed to be modular and extensible. All of the core protocols define abstract services they provide and requirements they must meet, but allow multiple mechanisms for doing so, as well as a way of easily adding new mechanisms. All the critical parameters of an SSH connection are negotiable, including the methods and algorithms used in:

- Session key exchange
- Server authentication
- Data privacy and integrity
- User authentication
- Data compression

SSH-TRANS is the fundamental building block, providing the initial connection, record protocol, server authentication, and basic encryption and integrity services. After establishing an SSH-TRANS connection, the client has a single, secure, full-duplex byte stream to an authenticated peer.

Next, the client can use SSH-AUTH over the SSH-TRANS connection to authenticate itself to the server. SSH-AUTH defines a framework within which multiple authentication mechanisms may be used, fixing such things as the format and order of authentication requests, conditions for success or failure, and how a client learns the available methods.
The user authentication can be preformed in two ways: The first is using public key authentication, and the second is by password authentication.

After authentication, SSH clients invoke the SSH-CONN protocol, which provides a variety of richer services over the single pipe provided by SSH-TRANS. This includes everything needed to support multiple interactive and non-interactive sessions: multiplexing several streams (or *channels*) over the underlying connection; managing X, and TCP forwarding.

The SSH2 protocol allows two hosts to construct a secure channel for data communication using public key authentication and a diffie-hellman-group1-sha1 exchange. The Diffie-Hellman key exchange provides a shared secret key that cannot be determined by either party alone. The shared secret key established is used as a session key. Once an encrypted tunnel is created using this key, the context for negotiated compression algorithm, and encryption algorithm are initialized. These algorithms may use independent keys in each direction. The first session key established is randomly unique for every session.

There are three main parts of the SSH protocol [1]:

- Algorithm Negotiation
- Authentication
- Data Encryption

Algorithm Negotiation is mainly responsible for determining the encryption algorithms, compression algorithms and the authentication methods supported and to be used between the client and the server.
Algorithm Negotiation is followed by Authentication. Authentication is further broken down in two pieces:

- Key exchange (transport layer) [9 ,10]
- User authentication (user authentication layer) [11]

The purpose of the key exchange is dual. Firstly, it attempts to authenticate the server to the client. Secondly, a shared key is established which is used as a session key to encrypt all the data being transferred between the two machines. The session key encrypts the payload and a hash generated for integrity checking of the payload using the private key of the server. The client verifies the server's public key, verifies the server signature received and then continues with user authentication.

Once Authentication is successful, one of the negotiated encryption algorithms is used to encrypt the data transferred between the two machines.

Following is a diagram [10] showing the key exchange mechanism that comes after the "algorithm negotiation" stage. The key exchange produces two values: a shared secret **K**, and an exchange hash **H**. Given the following variables,

n is a number of bits the client requested,

p is a large safe prime,

g is a generator for a subgroup of **GF(p)**, usually set to be equal to 2,

q is the order of the subgroup;

V_S is Server's version string;

V_C is Client's version string;

K_S is Server's public host key;

I_C is Client's **KEXINIT** message and

I_S is Server's **KEXINIT** message (which has been exchanged before the key exchange begins).

The client generates a random number **x** where **(1 < x < q)** and the server generates a random number **y** where **(0 < y < q)** and initiate the protocol as shown below [9,11]:

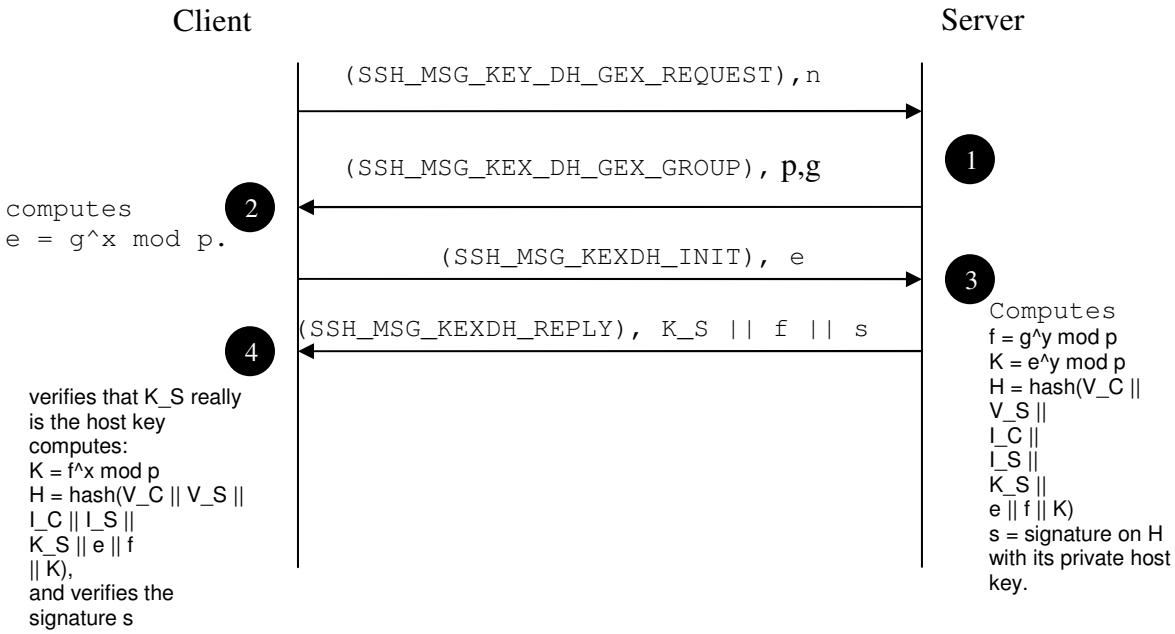

Fig 2.1 Transport layer key exchange

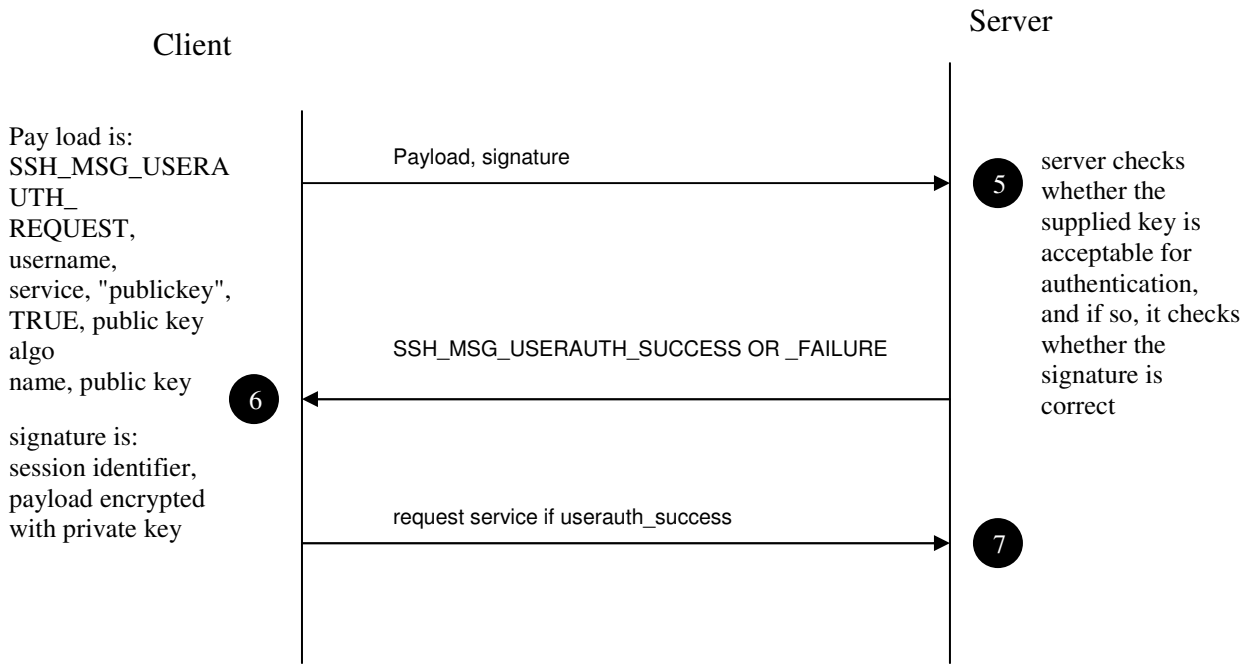

Fig 2.2 User Authentication Layer (using public key)

2.2 Trust

The server host key is used during key exchange to verify that the client is really talking to the correct server. For this to be possible, the client must have a priori knowledge of the server's public host key.

Two different trust models can be used:

- The client has a local database that associates each host name (as typed by the user) with the corresponding public host key. This method requires no centrally administered infrastructure, and no third-party coordination. The downside is that the database of name-to-key associations may become burdensome to maintain.

The protocol provides the option that the server name - host key association is not checked when connecting to the host for the first time. This allows communication without prior communication of host keys or certification. The connection still provides protection against passive listening; however, it becomes vulnerable to active man-in-the-middle attacks.

2.3 The Man in the middle Attack

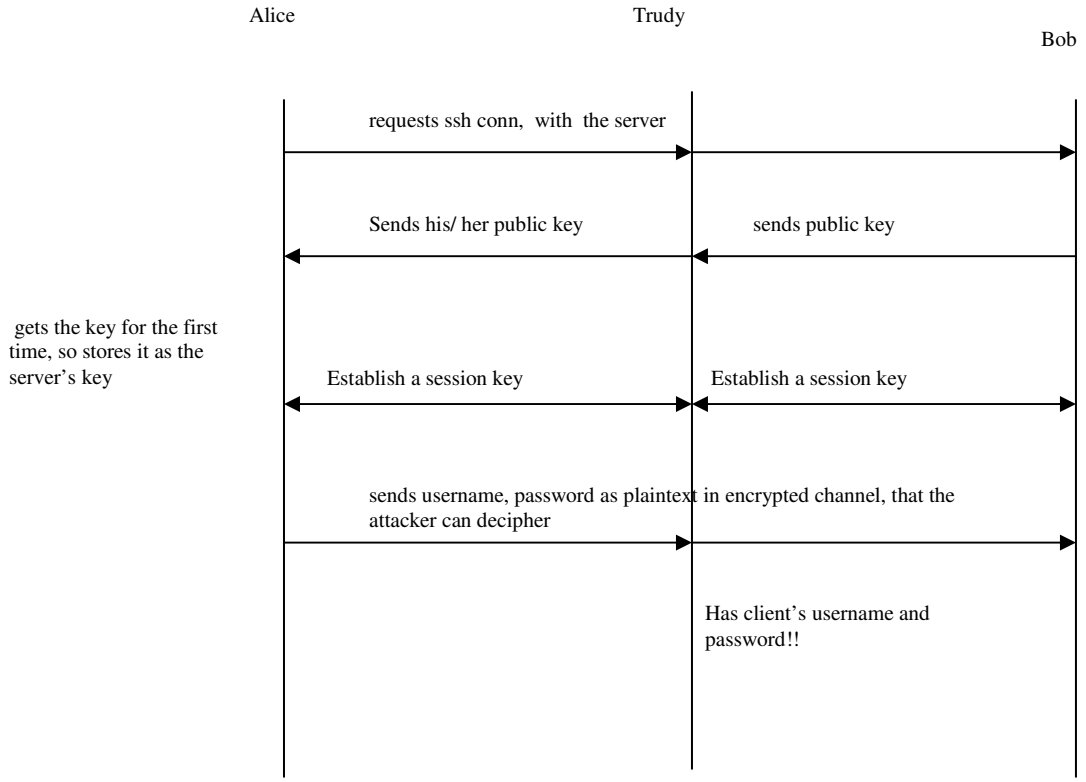

Fig 2.3 Overview of Man-in-the-Middle Attack

Suppose Alice (user) wants to talk to Bob (server) and a malicious user Trudy wants to play the man-in-the-middle attack.

- **Alice** initiates a connection with **Bob**.
- **Bob** sends his public key to **Alice**, which **Trudy** intercepts;

- **Trudy** then sends her own public key to **Alice**.
- **Alice** accepts the new public key and stores in its database. If it was the "first time authentication", it would blindly add the public key provided by **Trudy** thinking that it is **Bob**'s public key.
- **Alice** then sends her username and password to **Bob** that is again intercepted by **Trudy.**
- **Trudy** decrypts Alice's username and password using the session key and her private key,
- **Trudy** then encrypts Alice's credentials using the public key provided by **Bob** and forwards the new packet to him.
- **Bob** authenticates **Trudy** thinking that it authenticated **Alice**.

Trudy can now send commands to Bob and manipulate Alice's user account. Trudy can also just accumulate user id and password pairs. She might just pass Alice on to Bob, so Alice's session works as normal, except Trudy can log in later.

2.6 Public key cryptography in SSH

The public-key cryptographic operations performed by a client and server in different modes of the public key algorithm are summarized as follows.

Table 2.1 public key cryptographic operations

	DSA	**RSA**
Client	**DSAverify+DHoperation**	**RSAverify+DHoperation**
Server	**DSAsign+DHoperation**	**RSAsign+DHoperation**

Chapter 3
Cryptography

3.2 Introduction

Cryptography has been an important tool in securing information transactions for thousands of years. It was originally intended to disguise messages so that adversaries could not acquire or alter sensitive information. Cryptography involves encryption, which is the conversion of plaintext to an unreadable format known as ciphertext. If the encryption is secure, only an entity with the secret key can decrypt the ciphertext and convert it back to plaintext. Since its origination, cryptography has been divided to include two types of cryptosystems: symmetric and asymmetric. These terms define whether the cryptosystem uses a single key or a pair of keys for encryption and decryption. Cryptography has also been expanded to provide the following information security requirements:

- **Confidentiality** is a service used to keep the content of information accessible to only those authorized to have it. This service includes both protection of all user data transmitted between two points over a period of time as well as protection of traffic flow from analysis.
- **Integrity** is a service that requires that computer system assets and transmitted information be capable of modification only by authorized users. Modification includes writing, changing, changing the status, deleting, creating, and the delaying or replaying of transmitted messages. It is important to point out that integrity relates to active attacks and therefore, it is concerned with detection rather than prevention. Moreover, integrity can be provided with or without recovery, the first option being the more attractive alternative.
- **Authentication** is a service that is concerned with assuring that the origin of a message is correctly identified. That is, information delivered over a channel should be authenticated as to the origin, date of origin, data content, time sent, etc. For these reasons this service is subdivided into two major classes: entity

authentication and data origin authentication. Notice that the second class of authentication implicitly provides data integrity.

- **Non-repudiation** is a service which prevents both the sender and the receiver of a transmission from denying previous commitments or actions.

3.1.2 Symmetric Cryptosystems

Symmetric algorithms share the same key for encryption and decryption. Keeping this single key private is essential to the security of the algorithms. These traditional cryptosystems suffer from many disadvantages including key exchange and key management (when large groups must manage many pairs of keys). Despite this security risk, they do offer the advantages of having a small key length and consuming a low amount of computing power. Examples of well-known symmetric algorithms include the Data Encryption Algorithm (DEA) defined by the Data Encryption Standard (DES), and Triple-DES.

3.1.4 Asymmetric Cryptosystems

Asymmetric cryptosystems are better suited for real-world environments where the secure transfer of the secret key is not essential to the security of the system. These algorithms use a pair of keys – one private key typically used for decryption and one public key typically used for encryption. This allows for anyone who knows an entity's public key to send them an encrypted message with assurance that only that entity can decrypt it with their secret private key. The security and utility of public key cryptosystems is aided by their asymmetric design. Well-known asymmetric algorithms include DSA, RSA, and ELGAMAL.

3.1.5 Hash functions

Hash functions are (also known as **message digests)** are **one-way functions**. It is considered as a function because it takes an input message and produces an output.

A cryptographic hash function is a mathematical transformation that takes a message of arbitrary length (transformed in to a string of bits) and computes from it a fixed-length (short) number.

The hash of a message m, represented as h(m) has the following properties:

- For any message m, it is relatively easy to compute h(m). This means that in order to be practical it can't take a lot of processing time to compute the hash.
- Given h(m), there is no way to find an m that hashes to h(m) in a way that is substantially easier than going through all possible values of m and computing h(m) for each one.
- Even though it is obvious that many different values of m will be transformed to the same value h(m) (because there are many more possible values of m), it is computationally infeasible to find two values that hash to the same thing.

Hash algorithms can be used to improve the efficiency of public key algorithms. The best-known public key algorithms are sufficiently processor-intensive that it is desirable to compute a message digest of the message and sign that, rather than to sign the message directly. The message digest algorithms are much less processor-intensive, and the message digest is much shorter than the message.

Among others, MD5 and SHA-1 are two of the important hash functions that are usually used in the design of digital signatures. Even though, MD5 and SHA-1 are quite similar to each other, SHA-1 is considered to be more secure. SHA-1 can have a better resistance to brute-force attack as the message digest is 160 bits long; i.e. the digest is 32 bits longer than MD5 digest, which is 128 bits long. Moreover, MD5 is more vulnerable to cryptanalysis attack than SHA-1. However, it should be noticed that SHA-1 executes more slowly than MD5 as it involves more steps (80 versus 64) [26], and must process a 160 bit long buffer.

3.2 Public key cryptography

Earlier cryptographic applications have been constructed from substitution and transposition algorithms. With the advent of communication technology it was inevitably

important to develop more complex and effective cryptosystems to avoid sophisticated attacks that can easily break existing substitution based algorithms. Accordingly, various symmetric key block ciphers have been developed. Although, the newly developed complex symmetric cryptosystems has got a prominent role in insuring the security of modern communication networks, they are not without drawbacks. For one thing, they still rely on substitution and permutation process. For the other thing, key distribution and digital signature is not simple in symmetric cryptosystems.

The concept of public-key cryptography evolved from an attempt to solve two of the aforementioned problems associated with symmetric encryption. Key distribution under symmetry encryption requires either that two communicants already share a key; or the use of a key distribution center [5]. Diffie and Hellman, [25], argue that in symmetric encryption system, it is necessary for the communicating parties to share a key which is known to no one else. This is done by sending the key in advance over some secure channel such as a private courier or registered mail. A private conversation between two people with no prior acquaintance is a common occurrence in business, however, it is unrealistic to expect initial business contacts to be postponed long enough for keys to be transmitted by some physical means. Therefore, according to their argument, the cost and delay imposed by this key distribution problem is a major barrier to the transfer of business communications to large teleprocessing networks.

One additional problem that can be observed in symmetric key cryptosystem is that one party should generate and share as many keys as the number of contacts the party has. In case, if one key is to be shared among the contacts the party established, then the secrecy of the key is virtually eliminated. To alleviate such problems, public key cryptosystem offers a different approach to eliminate the need for a secure key distribution channel [25].

As long as there are various business and private contacts, documents will need equivalent signatures as that of hand written signatures on paper documents. This is because a message may be sent but later repudiated by either the sender or the receiver.

Or, it may be alleged by either party that a message was sent when in fact none was [25]. This is the reason why the counter-part of hand written signatures – the digital signature - is required in digital form to insure non-repudiation.

It is worth mentioning that there are misconceptions related to symmetric and asymmetric encryption. There is a misconception that public-key encryption is more secure from cryptanalysis than is the symmetric key [5]. In fact, the security of any encryption scheme depends on the length of the key and the computational work involved in breaking the cipher. There is nothing in principle about either symmetric or public-key encryption that makes one superior to the other from the point of view of resisting cryptanalysis [5]. Another, misconception is that symmetric encryption may become obsolete because of public-key encryption. However, because of the computational overhead of current public-key encryption schemes, there seems no foreseeable likelihood that symmetric encryption will be abandoned.

3.2.2 Principles of Public-key Cryptosystems

Public-key cryptosystem was first introduced by Whitfield Diffie and Martin E. Hellman in 1976. They introduced their findings in [25]. In general, public-key cryptography algorithm has two different but related keys. One is for encryption and the other is for decryption. The two characteristics of these algorithms are explained in [3] and it is generalized as given below:

i. It is computationally infeasible to determine the decryption key given only knowledge of the cryptographic algorithm and the encryption key. Algorithms like RSA also exhibit the following characteristics.
ii. Either of the two related keys can be used for encryption, with the other used for decryption.

3.2.2 The Mathematical Hard Problems

Public key cryptosystems are developed based on the fundamental notion of number theory and abstract algebra.
All cryptosystems are secure only if the difficulty of the mathematical problem that they are based on is determined to be hard. If the fastest algorithm known currently to solve a mathematical problem with the current technology and resources takes a long time, i.e. exponential time to the input size, then the problem is considered to be a difficult one.

It is apparently understood that if the input size of a particular algorithm for solving a mathematical problem is small, then solving the problem will be simple (Note: size is the number of bits needed to represent the input). However, when the input size of the algorithm is becoming large the hardness of the problem will increase accordingly. The time that may take to solve the problem will also grow. Thus, we can arrive at a certain level of confidence that the problem will become totally intractable for a certain minimum input size of the algorithm.

Public key cryptosystems are based on the intractability of one of three problems [4, 6]. These problems and the cryptosystems based on them are:

1. The Integer Factorization Problem; RSA
2. The Discrete Logarithm Problem; DSA, Diffie-Hellman, ElGamal
3. The Elliptic Curve Discrete Logarithm Problem; ECDSA, ECDH

3.2.2.4 The Integer Factorization Problem

Given N an odd composite integer with at least two distinct prime factors p and q, such that $N = p \times q$, it is believed that finding p and q is intractable. It is assumed that IFP is difficult only under properly chosen parameters.

The RSA problem (RSAP) is given in [4] as follows: Given a positive integer n that is a product of two distinct odd primes p and q, a positive integer e such that $gcd(e, (p-1)(q-1)) = 1$, and an integer c, find an integer m such that $m^e \equiv c \pmod{n}$.

It is obvious that an algorithm which solves the IFP will solve the RSA problem, since the receiver decrypt RSA cipher text exactly by first computing $d \equiv e^{-1} \pmod{(p-1)(q-1)}$, i.e. from the knowledge of factorization of N.

3.2.2.5 Discrete Logarithm Problem

In 1985, Taher ElGalmal first proposed the use of the DLP for public-key cryptosystems and digital signature schemes. The discrete logarithm problem, like the factoring problem, is said to be difficult in addition to being the hard direction of a one-way function [9]. It involves the mathematical structure of groups, of which the simplest definition is a nonempty set S together with a binary operation *. Let g be an element of group G. If the order n of element g is also the order of the group G, then g is referred to as a generator of G. This means that repeated exponentiation of g ($g*g*g$) will yield all elements of G. Cryptography commonly uses the group Z_p*, a finite and multiplicative group of integers modulo a prime number p. The discrete logarithm problem is characterized by these elements under the following condition:

1. Given a generator g of the multiplicative group Z_p*, a prime integer p, and another element $h \in Z_p*$, find the unique integer **j** in the interval $[0, p-1]$ such that $h \equiv g^j \pmod{p}$.

The inverse operation of exponentiation on $g^j \bmod p$ can be computed efficiently, but it is not practical to calculate $g^j \bmod p$ for large values. This is why the DLP is a one-way function and considered a hard problem.

3.2.2.6 Elliptic Curve Discrete Logarithm Problem

Based on the infeasibility of computing discrete logs on elliptic curves over finite fields, the Elliptic Curve Discrete Logarithm Problem is the security behind elliptic curve cryptography. Cryptosystems based on the computational complexity of this mathematical problem include ECDSA, ECElGamal, and Elliptic Curve Diffie-Hellman (ECDH). ECDLP is similar to the aforementioned Discrete Logarithm Problem and can be thought of as an analogue to DLP. In the ECDLP, however, the subgroup Z_p^* is replaced by the group of points on an elliptic curve over a finite field. In addition, unlike the DLP and the integer factorization problem, no subexponential-time algorithm is known for the ECDLP. ECDLP is considered to be significantly harder than the DLP, thus giving elliptic curve cryptosystems a greater strength-per-key-bit than their non-analogue discrete logarithm counterparts.

3.2.3 Applications for public key cryptosystems

In public key cryptography each user or the device taking part in the communication have a pair of keys, a public key and a private key, and a set of operations associated with the keys to do the cryptographic operations. Only the particular user/device knows the private key whereas the public key is distributed to all users/devices taking part in the communication. Since the knowledge of public key does not compromise the security of the algorithms, it can be easily exchanged online.

A shared secret can be established between two communicating parties online by exchanging only public keys and public constants if any. Any third party, who has access only to the exchanged public information, will not be able to calculate the shared secret unless it has access to the private key of any of the communicating parties. This is key agreement and is defined in the next section.

Apart from Key Agreement the other important applications of public key cryptography are Data Encryption and Digital Signature [5].

3.2.3.1 Key Agreement

Key agreement is a method in which the device communicating in the network establishes a shared secret between them without exchanging any secret data. In this method the parties that need to establish shared secret between them exchange their public keys.

Both the parties on receiving the other party's public key performs key generation operation using its private key to obtain the shared secret.

As we see in the previous section the public keys are generated using private key and other shared constants. Let P be the private key of an entity and U(P, C) be the public key. Since public key is generated using private key, the representation U(P, C) shows that the public key contain the components of private key P and some constants C where C is known by all the entity taking part in the communication.

Consider two entities A and B. Let PA and UA(PA, C) be the private key and public key of device A, and PB and UB(PB, C) be the private key and public key of device B respectively.

Both party exchanges their public keys. A, having got the public key of B, uses its private key to calculate shared secret KA=Generate_Key(PA, UB(PB, C))

B, having got the public key of A, uses its private key to calculate the shared secret KB=Generate_Key(PB, UA(PA, C))

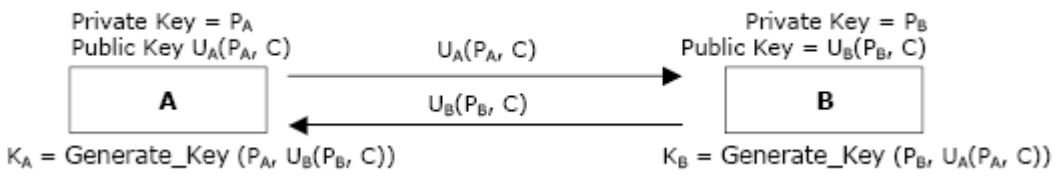

Fig 3.1 key agreement between two devices A and B

The key generation algorithm 'Generate_Key' will be such that the generated keys at the device A and B will be the same, that is shared secret KA=KB=K(PA, PB, C).

Since it is practically impossible to obtain private key from the public key any middleman, having access only to the public keys UA(PA, C) and UB(PB, C), will never be able to obtain the shared secret K.

Examples of key agreement algorithms are DH, RSA and ECDH.

During the key exchange process the public keys may pass through different intermediate points. Any middleman can thus tamper or change the public keys to its public key.

Therefore for establishing shared secret it is important that device A receives the correct public key from device B and vice versa. Digital Certificate helps to deliver the public key in authenticated method.

3.2.3.2 Digital Signature

The wide use of the Internet in our day to day activities triggers a new kind of requirement. Everyone wants to be sure with whom he/she is communicating through the on-line communication channel. In a hand written message transfer, it is possible to identify the identity of the sender based on the sender's hand-written signature. In the case of computer communication, communicating parties need a way by which they can identify whom they are communicating with and a way by which possible disputes can be resolved. The new requirement is, in fact, for having digital signature which is the counter part of the hand-written signature.

Digital signing techniques are employed to provide sender authentication, message integrity and sender non-repudiation, provided that private keys are kept secret and the integrity of public keys is preserved. Provision of these services is furnished with the proper association between the users and their public/private key pairs.

If A wishes to sign a document, he or she must use the private key available only to him or her. When B receives a digitally signed message from A, B must verify its signature. B needs A's public key for this verification. A should have high confidence in the integrity of that key.

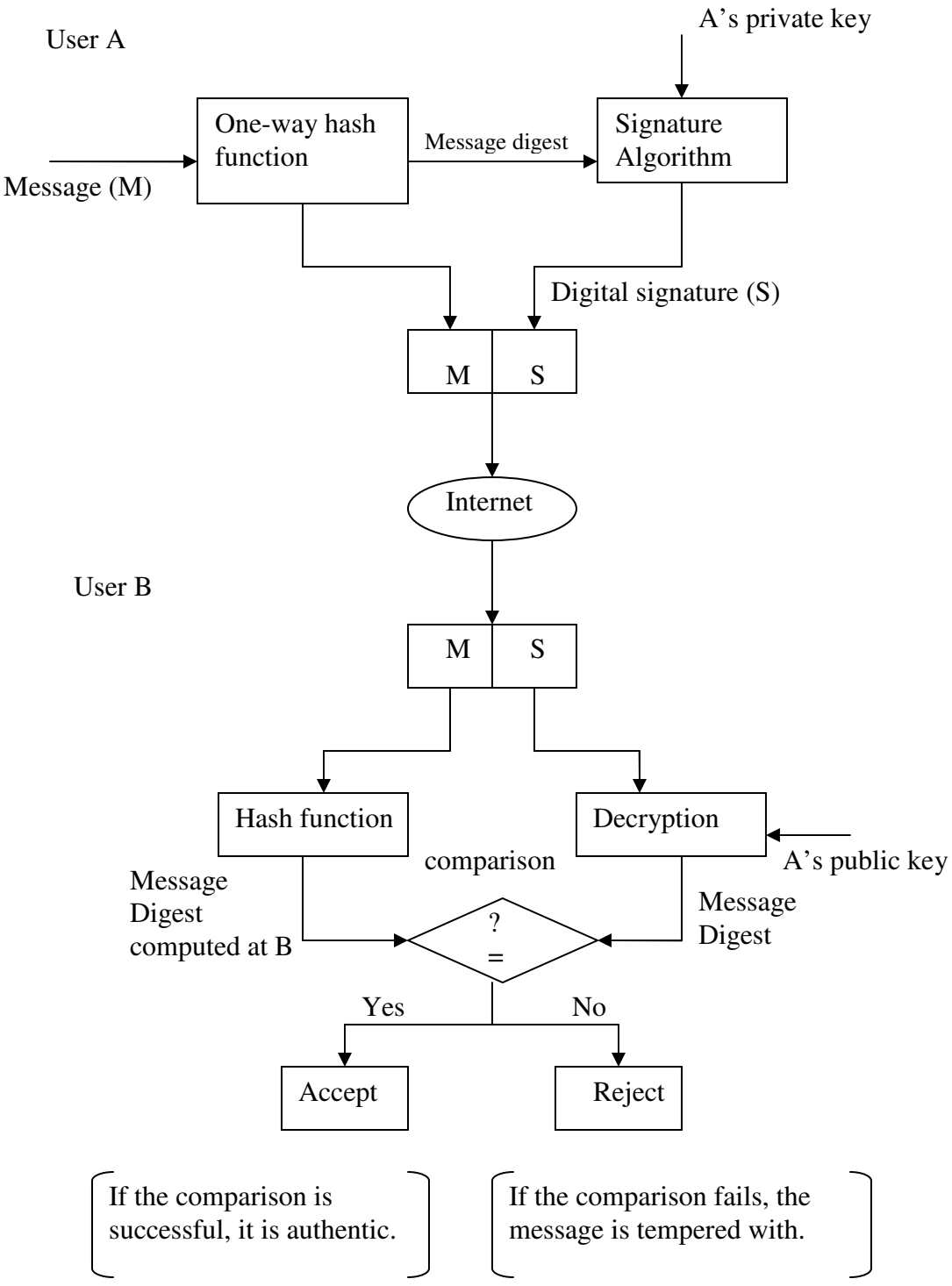

Fig 3.2 Overview of a typical signature scheme

3.2.3.3 Encryption

Encryption is a process in which the sender encrypts/scrambles the message in such a way that only the recipient will be able to decrypt/ descramble the message.

Consider a device B whose private key and public key are PB and UB respectively. Since UB is public key all devices will be able to get it. For any device that needs to send the message 'Msg' in a secured way to device B, it will encrypt the data using B's public key to obtain the cipher text 'Ctx'. The encrypted message, cipher text, can only be decrypted using B's private key. On receiving the message the B decrypts it using its private key PB.

Since only B knows its private key PB none other including A can decrypt the message.

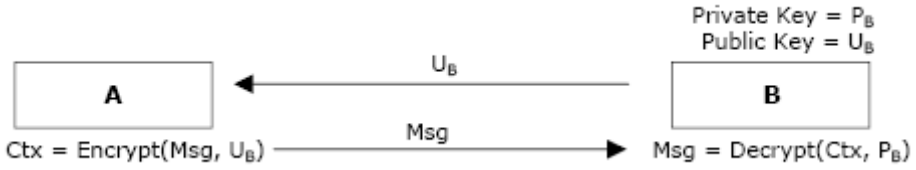

Fig 3.3 encryption

It is important that device A receives the correct public key from device B, i.e. no middleman must tamper or change the public key to its public key.

One of the popular public key encryption algorithms is RSA.

3.2.5 Diffie–Hellman Exponential Key Exchange

The first practical public key scheme that can help to solve key exchange problem was proposed by Diffie and Hellman and is generally referred to as Diffie-Hellman key exchange.

In 1976, Diffie and Hellman proposed a scheme using the exponentiation modulo q (a prime) as a public key exchange algorithm [25]. Exponential key exchange takes advantage of easy computation of exponentials in a finite field GF(q) with a prime q

compared with the difficulty of computing logarithms over GF(q) with q elements {1, 2, . . . , q − 1}. Let q be a prime number and α a primitive element of the prime number q. Then the powers of α generate all the distinct integers from 1 to q − 1 in some order. For any integer Y and a primitive element α of prime number q, a unique exponent X is found such that

$Y \equiv \alpha^X \pmod{q}$, $1 \leq X \leq q - 1$

Then X is referred to as the discrete logarithm of Y to the base α over GF(q):

$X = \log_\alpha Y$ over GF(q), $1 \leq Y \leq q - 1$

Calculation of Y from X is comparatively easy, using repeated squaring, but computation of X from Y is typically far more difficult.

Suppose the user i chooses a random integer Xi and the user j a random integer Xj .

Then the user i picks a random number Xi from the integer set {1, 2, . . . , q − 1}. The user i keeps Xi secret, but sends

$Yi \equiv \alpha^{Xi} \pmod{q}$ to the user j.

Similarly, the user j chooses a random integer Xj and sends

$Yj \equiv \alpha^{Xj} \pmod{q}$ to the user i.

Both users i and j can now compute:

$Kij \equiv \alpha^{XiXj} \pmod{q}$ and use Kij as their common key.

The user i computes Kij by raising Yj to the power Xi:

$Kij \equiv Y^{Xi}_j \pmod{q}$

$\equiv (\alpha^{Xj})^{Xi} \pmod{q}$

$\equiv \alpha^{XiXj} \pmod{q}$

and the user j computes Kij in a similar fashion:

$Kij \equiv Y^{Xj}i \pmod{q}$

$\equiv (\alpha^{Xi})^{Xj} \equiv \alpha^{XiXj} \pmod{q}$

Thus, both users i and j have exchanged a secret key. Since Xi and Xj are private, the only available factors are the public values q, α, Yi and Yj. Therefore the opponent is forced to compute a discrete logarithm which is considered to be unrealistic, particularly for large primes.

When utilising finite field GF(q), where q is either a prime or $q = 2^k$, it is necessary to ensure the $q - 1$ factor has a large prime, otherwise it is easy to find discrete logarithms in GF(q).

3.2.5 RSA Cryptosystems

The Rivest-Shamir-Adleman public-key algorithm (RSA) is the most widely used asymmetric cipher. RSA was proposed by Rivest, Shamir, and Adleman in 1977[3, 5]. It derives its security from the difficulty of factoring large integers that are the product of two large primes of roughly equal size. Factoring is widely believed to be intractable (i.e., infeasible, admitting no efficient, polynomial-time solution).

In other words, given two large integers p and q, it is easy to compute their product $n = pxq$. However, given a large integer n (where n has over 300 digits) it is extremely difficult, given today's resources, to factor n into two non-trivial factors p and q. For this reason, integer factorization is called a **one-way function**: it is easy to multiply integers, but hard to invert the operation (i.e. factor integers).

RSA Key Generation

A RSA public and private key pair can be generated using the algorithm below [26]:

Choose two random prime numbers p and q such that the bit length of p is approximately equal to the bit length of q.

Compute n such that $n = p * q$.

Compute $\varphi(n)$ such that $\varphi(n) = (p - 1)*(q - 1)$.

Choose a random integer e such that $e < \varphi(n)$ and $\gcd(e, \varphi(n)) = 1$, then compute the integer d such that: $e*d \equiv 1 \bmod \varphi(n)$.

(n, e) is the public key, and d is the private key.

RSA Signature Generation

Signature of a message m is a straightforward modular exponentiation using the hash of the message and the private key. The signature s can be obtained by:

$s = \text{hash}(m)^d \pmod n$

A common hash algorithm used is SHA-1.

RSA Signature Verification

To verify a signature s for message m, the signature must first be decrypted using the author's public key (n, e). The hash h is thus obtained by

$h = s^e \pmod{n}$

If h matches hash(m), then the signature is valid – the message was signed by the author, and the message has not been modified since signing.

3.2.6 Digital Signature Algorithm (DSA)

In August 1991, The National Institute of Standards and Technology (NIST) proposed the Digital Signature Algorithm (DSA) for use in their Digital Signature Standard (DSS) [5, 26].

The proposed DSS uses a public key to verify to a recipient the integrity of data and identity of the sender of the data. The DSS can also be used by a third party to ascertain the authenticity of a signature and the data associated with it.

The technique selected provides for efficient implementation of the signature operations in smart card applications. In these applications the signing operations are performed in the computationally modest environment of the smart card while the verification process is implemented in a more computationally rich environment such as a personal computer, a hardware cryptographic module, or a mainframe computer.

The DSA is based on the difficulty of computing discrete logarithms [26],

The public key consists of three parameters, p, q and g, and is common to a group of users. Choose q of a 160-bit prime number and select a prime number p with $512 < p < 1024$ bits such that q is a prime factor of $p - 1$. Next, choose $g > 1$ to be of the form $h'^{(p-1)/q} \pmod{p}$ such that h' is an integer between 1 and $p - 1$.

With these three numbers, each user chooses a private key x in the range $1 < x < q - 1$ and the public key y is computed from x as $y \equiv g^x \pmod{p}$. Recall that determining x is computationally impossible because the discrete logarithm of y to the base g (mod p) is difficult to calculate.

To sign a message m, the sender computes two parameters, r and s, which are functions of (p, q, g and x), the message digest $H(m)$, and a random number $k < q$. At the receiver, verification is performed as shown in Table 5.10. The receiver generates a quantity v that is a function of parameters (x, y, r, $s-1$ and $H(m)$).

When a one-way hash function H operates on a message m of any length, a fixed-length message digest (hash code) h can be produced such that $h = H(m)$. The message digest h to the DSA input computes the signature for the message m. Signing the message digest rather than the message itself often improves the efficiency of the signature process, because the message digest h is usually much smaller than the message m.

Key pair generation:

p: a prime number between 512 to 1024 bits long

q: a prime factor of $p - 1$, 160 bits long

$g \equiv h'^{(p-1)/q} \pmod{p} > 1$, and $h' < p - 1$

(p, q and g): public parameters

$x < q$: the private key, 160 bits long

$y \equiv g^x \pmod{p}$: the public key, 160 bits long

Signing process (sender):

$k < q$: a random number

$r \equiv (gk \bmod p) \pmod{q}$

$s \equiv k^{-1} (h + xr) \pmod{q}$, $h = H(m)$ is a one-way hash function of the message m.

(r, s): signature

Verifying signature (receiver):

$w \equiv s-1 \pmod{q}$

$u1 \equiv h \times w \pmod{q}$

$u2 \equiv r \times w \pmod{q}$

$v \equiv (gu1yu2 \pmod{p}) \pmod{q}$

If $v = r$, then the signature is verified.

3.2.7 Elliptic Curve Cryptography

Although elliptic curves have been studied for over 150 years, they weren't applied to cryptography until very recently. In 1985, Neal Koblitz and Victor Miller independently proposed the use of elliptic curve points over a finite field for cryptosystems [26]. Since then, several standards have accepted the use of elliptic curve cryptosystems. Interest in elliptic curve cryptography is due to the determination that is based on a harder mathematical problem than other cryptosystems. The elliptic curve discrete logarithm problem appears to be substantially more difficult than the existing discrete logarithm problem. Considering they have equal levels of security, ECC uses smaller parameters than the conventional discrete logarithm systems. Because of this, ECC can offer certain advantages over non-elliptic curve schemes. For effective use in cryptosystems, elliptic curves are defined over finite fields that require unique arithmetic operations.

3.2.7.3 Utilization of Elliptic Curve Cryptography

Elliptic curve cryptography's strengths make it most suitable for resource-constrained systems. ECC provides greater security for a given key size and can be efficiently and compactly implemented. These attributes make it well suited for systems with constraints on processor speed, security, heat production, power consumption, bandwidth, and memory. Cell phones, PDAs, wireless devices, laptops, and smartcards are applications that benefit from elliptic curve cryptosystems.

3.2.7.4 Elliptic Curves Over Finite Fields

Elliptic curve cryptography uses elliptic curves to find points on an ellipse. These points are used for keys within a discrete range. This means that all numbers must be finite and positive integers from 0 to a specified number used. The field constraints of elliptic curves used in cryptosystems prevent problems such as infinitely repeating fractions and round-off errors.

Of the different fields that elliptic curves reside in, GF(p) prime fields and GF(2^k) binary polynomial fields are most effective for cryptographic implementations[6,26]. Of the two, GF(2^k) is more popular due to available space and time-efficient implementations of elliptic curve arithmetic in GF(2^k).

3.2.7.2.1 Elliptic curves over F_2^m

3.2.7.2.1.1 The finite field F_2^m

There are many ways to represent the elements of a finite field with 2^m elements. The particular method used in this challenge is called a *polynomial basis representation.*

Let $f(x) = x^m + f_{m-1} x^{m-1} + \ldots + f_2 x^2 + f_1 x + f_0$ (where $f_i \in \{0, 1\}$ for $i = 0, 1, \ldots, m - 1$) be an irreducible polynomial of degree m over F2. That is, $f(x)$ cannot be factored as a product of two polynomials over F_2, each of degree less than m. The polynomial $f(x)$ *is* called the *reduction polynomial.*

The finite field F2m is comprised of all polynomials over F2 of degree less than m:

$F_2^m = \{a_{m-1} x^{m-1} + a_{m-2} x^{m-2} + \ldots + a_1 x + a_0 : a_i \in \{0, 1\}\}$.

The field element $a_{m-1} x^{m-1} + a_{m-2} x^{m-2} + \ldots + a_1 x + a_0$ is usually denoted by the binary string (am-1 am-2… a1a0) of length m, so that

$F_2^m = \{(a_{m-1}\ a_{m-2} \ldots a_1 a_0) : a_i \in \{0, 1\}\}$.

Thus the elements of F_2^m can be represented by the set of all binary strings of length m. The multiplicative identity element (1) is represented by the bit string (00. . .01), while the zero element (additive identity) is represented by the bit string of all 0's.

The following arithmetic operations are defined on the elements of F_2^m:

- *Addition*: If $a = (a_{m-1}\ a_{m-2} \ldots a_1 a_0)$ and $b = (b_{m-1}\ b_{m-2} \ldots b_1 b_0)$ are elements of F_2^m, then $a + b = c = (c_{m-1}\ c_{m-2} \ldots c_1 c_0)$, where $c_i = (a_i + b_i) \bmod 2$. That is, field addition is performed bitwise.
- *Multiplication*: If $a = (a_{m-1}\ a_{m-2} \ldots a_1 a_0)$ and $b = (b_{m-1}\ b_{m-2} \ldots b_1 b_0)$ are elements of F_2^m, then $a \bullet b = r = (r_{m-1}\ r_{m-2} \ldots r_1 r_0)$, where the polynomial $r_{m-1} x^{m-1} + r_{m-2} x^{m-2} + \ldots + r_1 x + r_0$ is the remainder when the polynomial $(a_{m-1} x^{m-1} + a_{m-2} x^{m-2} + \ldots + a_1 x + a_0) \bullet (b_{m-1} x^{m-1} + b_{m-2} x^{m-2} + \ldots + b_1 x + b_0)$ is divided by *f(x)* over F2.
- *Inversion*: If *a* is a non-zero element in F_2^m, the *inverse* of *a,* denoted a^{-1}, is the unique element $c \in F_2^m$ for which $a \bullet c = 1$.

3.2.7.2.1.2 Elliptic curves over F2*m*

A (non-supersingular) *elliptic curve E*(F2*m*) over F2*m* defined by the parameters *a, b* $\in F_2^m$, $b \neq 0$, is the set of all solutions (x, y), $x, y \in$ F2*m*, to the equation

$$y2 + xy = x3 + ax2 + b, \qquad (3.1)$$

together with an extra point O, the *point at infinity.*

The set of points *E*(F2*m*) forms a group with the following addition rules:

1. O + O = O
2. $(x, y) + O = O + (x, y) = (x, y)$ for all $(x, y) \in E(F_2^m)$.
3. $(x, y) + (x, x + y) = O$ for all $(x, y) \in E(F_2^m)$ (i.e., the negative of the point (x, y) is $-(x, y) = (x, x+ y)$).
4. (Rule for adding two distinct points that are not inverses of each other)

Let $P = (x1, y1) \in E(F_2^m)$ and $Q = (x2, y2) \in E(F_2^m)$ be two points such that $x1 \neq x2$. Then $P + Q = (x3, y3)$, where

$$X_3 = \lambda^2 + \lambda + x_1 + x_2 + a,$$

$$Y_3 = \lambda(x_1 + x_3) + x_3 + y_1, \text{ and}$$

$$\lambda = (y_2 + y_1)/(x_2 + x_1)$$

5. (Rule for doubling a point)

Let $P = (x1, y1) \in E(F2m)$ be a point with $x1 \neq 0$. (If $x1 = 0$ then $P = -P$, and so $2P = O$.)

Then $2P = (x3, y3)$, where

$x_3=\lambda^2+\lambda+a,$

$y3=x_1^2+(\lambda+1)x_3,$and

$\lambda=x_1+y_1/x_1.$

3.2.7.2.1.2 Elliptic curves over F*p*

3.2.7.2.1.2.1 The finite field F*p*

Let *p* be a prime number. The finite field F*p* is comprised of the set of integers

$$\{0, 1, 2, \ldots, p-1\}$$

with the following arithmetic operations

- *Addition*: If *a, b* ϵF*p,* then $a + b = r,$ where *r* is the remainder when $a + b$ is divided by *p* and $0 \leq r \leq p - 1$. This is known as addition modulo *p*.
- *Multiplication:* If *a, b*ϵF*p,* then $a \bullet b = s,$ where *s* is the remainder when $a \bullet b$ is divided by *p* and $0 \leq s \leq p - 1$. This is known as multiplication modulo *p*.

Inversion: If *a* is a non-zero element in F*p*, the *inverse* of *a* modulo *p*, denoted $a^{-1},$ is the unique integer *c*ϵF*p* for which $a \bullet c = 1$.

3.2.7.2.1.2.2 Elliptic curves over F*p*

Let $p > 3$ be a prime number. Let *a, b*ϵF*p* be such that $4a3 + 27b2 \neq 0$ in F*p*. An *elliptic curve E*(F*p*) over F*p* defined by the parameters *a* and *b* is the set of all solutions (*x*, *y*), *x*,*y* ϵF*p*, to the equation

$$y2 = x3 + ax + b, \tag{3.2}$$

together with an extra point O, the *point at infinity.*

The set of points *E*(F*p*) forms a group with the following addition rules:

1. O + O = O
2. (*x*, *y*) + O = O + (*x*, *y*) = (*x*, *y*) for all (*x*, *y*) ϵ*E*(F*p*).
3. (*x*, *y*) + (*x*, – *y*) = O for all (*x*, *y*) ϵ*E*(F*p*) (i.e., the negative of the point (*x*, *y*) is – (*x*, *y*) = (*x*, –*y*)).
4. (Rule for adding two distinct points that are not inverses of each other)

Let $P = (x1, y1) \in E(Fp)$ and $Q = (x2, y2) \in E(Fp)$ be two points such that $x1 \neq x2$. Then $P + Q = (x3, y3)$, where

$x3=\lambda 2-x1-x2,$

$y3=\lambda(x1-x3)-y1,$and

$\lambda=(y2-y1)/(x2-x1)$

5. (Rule for doubling a point)

Let $P = (x1, y1) \in E(Fp)$ be a point with $y1 \neq 0$. (If $y1 = 0$ then $P = -P$, and so $2P = O$.)

Then $2P = (x3, y3)$, where

$x_3=\lambda^2-2x_1,$

$y_3=\lambda(x_1-x_3)-y_1,$and

$\lambda=(3x_1^2+a)/2y_1$

3.2.7.2.1.3 Point multiplication

Point multiplication or scalar multiplication is the central operation in ECC and dominates the execution time of elliptic curve cryptographic schemes [6]. In point multiplication a point P on the elliptic curve is multiplied with a scalar k using elliptic curve equation to obtain another point Q on the same elliptic curve.

i.e. k*P=Q

Point multiplication is achieved by two basic elliptic curve operations

• **Point addition**, adding two points J and K using elliptic curve equation to obtain another point L i.e., L = J + K.

• **Point doubling**, adding a point J to itself using elliptic curve equation to obtain another point L i.e. L = 2J.

Here is a simple example of point multiplication.

Let P be a point on an elliptic curve. Let k be a scalar that is multiplied with the point P to obtain another point Q on the curve. i.e. to find Q = k*P.

If k = 23 then k*P = 23*P = 2(2(2(2P) + P) + P) + P.

In the ECC explanations given below upper case letter indicates a point in the elliptic curve and the lower case letter indicates a scalar

Right-to-left binary method for point multiplication [6]
INPUT: $k = (kt-1, \ldots, k1, k0)2$, $P \in E(\mathrm{F}q)$.
OUTPUT: kP.
1. $Q \leftarrow \infty$.
2. For i from 0 to $t-1$ do
2.1 If $ki = 1$ then $Q \leftarrow Q + P$.
2.2 $P \leftarrow 2P$.
3. Return(Q).

3.2.7.2.1.4 Domain parameters

There are certain public constants that are shared between parties involved in secured and trusted ECC communication. This includes curve parameter **a**, **b**, a generator point **G** in the chosen curve, the modulus **p**, order of the curve **n** and the cofactor **h**. The parameters should be chosen so that the ECDLP is resistant to all known attacks.

3.2.7.2.1.5 Elliptic Curve Key Pairs

All the public-key cryptographic schemes described in this document use key pairs known as elliptic curve key pairs.
Given some elliptic curve domain parameters $T=(p,a,b,G,n,h)$ or (m,f(x),a,b,G,n,h), an elliptic curve key pair $(d;Q)$ associated with T consists of an elliptic curve secret key d which is an integer in the interval[1,n-1], and an elliptic curve public key $Q= (xQ;yQ)$ which is the point $Q = dG$.

3.2.7.2.1.5.1 Elliptic Curve Key Pair Generation Primitive

Elliptic curve key pairs should be generated as follows:

Input: Valid elliptic curve domain parameters $T = (p;a;b;G;n;h)$ or $(m; f(x);a;b;G;n;h)$.

Output: An elliptic curve key pair $(d;Q)$ associated with T.

Actions: Generate an elliptic curve key pair as follows:

1. Randomly or pseudorandomly select an integer d in the interval $[1;n-1]$.
2. Calculate $Q = dG$.
3. Output $(d;Q)$.

3.2.7.2.1.6 ECDH – Elliptic curve Diffie Hellman

ECDH, a variant of DH, is a key agreement algorithm. For generating a shared secret between A and B using ECDH, both have to agree up on Elliptic Curve domain parameters. An overview of ECDH is given below.

Key Agreement Algorithm

For establishing shared secret between two parties A and B

Let dA and dB be the private key of A and B respectively, Private keys are random number less than n, where n is a domain parameter.

Let QA = dA*G and QB = dB*G be the public key of device A and B respectively, G is a domain parameter

A and B exchanged their public keys

The end A computes K = (xK, yK) = dA*QB

The end B computes L = (xL, yL) = dB*QA

. Since K=L, shared secret is chosen as xK

ECDH - Mathematical Explanation

To prove the agreed shared secret K and L at both A and B are the same

From the above equations

K = dA*QB = dA*(dB*G) = (dB*dA)*G = dB*(dA*G) = dB*QA = L

Hence K = L, therefore xK = xL

Since it is practically impossible to find the private key dA or dB from the public key QA or QB, it is not possible to obtain the shared secret for a third party.

3.2.7.2.1.7 ECDSA - Elliptic curve Digital Signature Algorithm

ECDSA is a variant of the Digital Signature Algorithm (DSA). For sending a signed message from A to B, both have to agree up on Elliptic Curve domain parameters. Sender A have a key pair consisting of a private key dA (a randomly selected integer less than n, where n is the order of the curve, an elliptic curve domain parameter) and a public key QA = dA*G (G is the generator point, an elliptic curve domain parameter). An overview of ECDSA process is defined below.

Signing

Consider the device A that signs the data M that it sends to B.

Let dA be A's private key

Calculate m = HASH (M), where HASH is a hash function, such as SHA-1

Select a random integer k such that 0<k<n

Calculate r = x1 mod n, where (x1, y1) = k*G

Calculate s = k − 1(m + dA*r) mod n

The signature is the pair (r, s)

Verification

Let M be the message and (r, s) be the signature received from A

Let QA be A's public key. Since QA is public, B has access to it.

Calculate m = HASH (M)

Calculate w = s^{-1} mod n

Calculate u1 = m*w mod n and u2 = r*w mod n

Calculate (x1, y1) = u1*G + u2*QA

The signature is valid if x1 = r mod n, invalid otherwise

3.2.8 Computational Overhead of public-key cryptosystems

In any public-key system, the private key is used for signing and decrypting and the public key is used for verifying and encrypting.

The RSA crypto algorithm really only uses one type of calculation **exponentiation**. RSA takes a message to the power of an exponent. The nature of the exponent determines the speed of the operation. For signing and decryption, the exponent (private key) is large, so

the calculation is quite slow. For verification and encryption, however, the exponent (public key) can be quite small. Some implementers use a public key as small as 3 for very fast verification. Therefore, any analysis of RSA speed consists of slow times for signing and decrypting and much faster speeds for verification and encryption.

In ElGamal systems including DSA and Elliptic Curve DSA (ECDSA), completely different mathematical operations are used for signatures and encryption/decryption, so signing and decryption times do not equal each other, nor do those of verifying and encrypting. And the basic ratio is reversed: Signing is faster than verification; decryption is faster than encryption. More significant is that the difference between public and private key operation times is trivial compared with this difference in RSA. Even though ECC public key operations are a bit slower than those in RSA, the sum of the public and private key times in ECC is much smaller than in RSA.

In both types of systems, considerable computational savings are achievable [4]. In RSA, as previously described, a short public exponent can be employed (although this does incur some security risks) to speed up signature verification and encryption. In both DSA and ECC, a large portion of the signature generation and encrypting transformations can be pre-computed. Various special bases for the finite field $F2m$ can be used to accelerate the arithmetic involved in ECC operation.

3.3 Public-Key Infrastructure

public-key infrastructure (PKI) is the set of hardware, software, people, policies, and procedures needed to create, manage, store, distribute, and revoke digital certificates based on asymmetric cryptography[5]. The primary function of a PKI is to allow the distribution and use of public keys and certificates with security and integrity. A PKI is a foundation on which other applications and network security components are built.

PKI Architectural Model

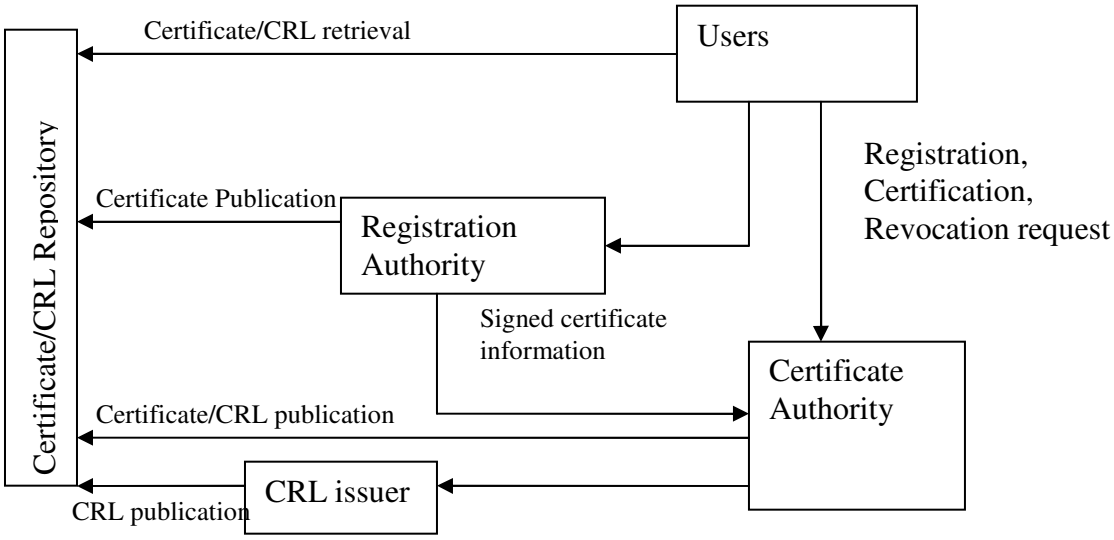

Fig 3.4 PKI Architecture

3.3.1 Components of PKI

PKI provides security services such as: Authentication, Integrity, confidentiality and non-repudiation. In order to support the above services, the infrastructure relies on a few central components and concepts to be discussed in the next sections [27].

3.3.1.1 X509 Public Key Certificates

Authentication in a PKI based system is realized through the use of *public key certificates.*

A public key certificate demonstrates or attests to the binding of an end entity's identity and its public key. That is, it contains enough information for some other relying entity to validate and verify the identity of the owner of the certificate. The basic constructs of a certificate should include the name of an entity, identifying information about the entity, expiration period for the certificate, and the entity's public key. Normally one would expect a certificate used in the internet world to include additional information and be based upon an accepted format to allow interoperability.

Many different formats for certificates exist, but the most widely deployed is *X509 version 3 public key certificates*. The X509 certificate structure is outlined in Table 2.1.

In a certificate, a binding between a public key and a Distinguished Name (DN) is created. A DN is a string on the following format [CN, OU, O, L, ST, C] that is meant to uniquely identify the owner:

- Common Name (CN): The full given name of the owner.
- Organizational Unit (OU): The unit the owner belongs to within her organization.
- Organization (O): The organization the owner is associated with.
- Locality (L): The location of the owner, i.e. city.
- State (ST): The state or province the owner resides in.
- Country (C): Two-letter country code for the owner.

Table 3.1 X509 Version 3 public key certificates

Field	**Description**
Version	X509 public key certificate version number. Three versions exist: 1, 2, and 3.
Serial Number	Unique identifier for the certificate within the CA.
Signature	Identifies the algorithm that was used to sign the certificate.
Issuer	The DN of the issuing CA
validity	Specifies two dates between which the certificate is considered valid, unless it has been otherwise canceled
subject	The DN of the owner of the certificate.
Subject Public Key Info	The public key and algorithm identifier for the owner.
Issuer Unique ID	Optional field assigning a unique ID to the CA issuing the certificate.
Subject Unique ID	Optional field assigning a unique ID to the owner of the certificate.
Extentions	Includes a set of optional extension fields. Examples include *Certificate Policies*, and *Subject Alternative Name*.
Digital signature	A signature on all the above fields by the issuing CA.

3.3.1.2 Certificate Revocation List (CRL)

The validity period of a certificate is determined by the field Validity presented in Table 3.1. In certain situations it's desirable to cancel the certificate before the end of that time window, e.g., in the case of theft or suspected compromise of the private key. The victim would then need to notify all other members of the PKI about the revocation. In practice, the user informs the CA who in turn will inform the rest of the user community. This can be accomplished by using a CRL, which is a listing of serial numbers belonging to certificates that have been canceled. X509 public key certificates contain an extension named 'CRL Distribution Point', which points to the location where revocation information for the given certificate can be obtained. Other techniques for revocation exist, such as on-line revocation solutions and schemes relying on reduced data structures [27].

Table 3.2 Certificate Revocation List (CRL)

field	Description
Version (optional)	X.509 CRL format This field is present only if extensions are used
Signature	For CRL issuer's signature, signature algorithm (RSA or DSA) and hash function (MD5 or SHA-1)
Issuer Name	CRL issuer Name
This Update	Issue the data of CRL (date/time)
Next Update	Issue the CRLs with a next update time equal to or later than all previous CRLs (date/Time)

Revoked certificates	A list of certificates that have been revoked: Identify uniquely by certificate serial number Date on the revocation occurrence is specified Optional CRL entry extensions: - Give the reason for revoked certificate - State the data for invalidity - State the name of CA issuing the revoked certificate
CRL Extensions	Authority key identifier Issuer alternative name CRL number, delta CRL indicator, issuing distribution point
CRL Entry Extensions	Reason code Hold instruction code Invalidity date Certificate issuer
CRL issuer's digital signature	

3.3.1.3 Certification Authority (CA)

A PKI enables the establishment of a trust hierarchy that provides risk management controls. The concept of trust, relative to a PKI, can be explained by the role of the CA. In the Internet environment, entities unknown to each other do not have sufficient trust established between them to perform business, contractual, legal, or other types of transactions. The implementation of a PKI using a CA provides this trust.

In short, a CA functions as follows. Entities that are unknown to one another, each individually establish a trust relationship with a CA. The CA performs some level of entity authentication, according to its established rules as noted in its Certificate Practices Statement or CPS, and then issues each individual a digital certificate. That certificate is signed by the CA and thus vouches for the identity of the individuals.
Unknown individuals can now use their certificates to establish trust between them because they trust the CA to have performed an appropriate entity authentication, and the CA's signing of the certificates attests to this fact. The identification procedure varies with the intended use of the certificate. a certificate that is to be in critical operations identification would probably involve meeting with the CA in person, presenting a passport, birth certificate, and possibly other credentials. Getting a certificate intended for signing of e-mails is typically less cumbersome.

CA functions as a trusted third party and provides various key management services. A CA essentially certifies the identity of an end entity. This is accomplished by an entity providing sufficient proof of their identity to the CA, and then having the CA generate a message containing the entity's identity and public key. This message is called a certificate and is cryptographically signed by the CA.
The level of trust that a CA has depends on the level of acceptance that other entities have in that CA. This level of acceptance depends on the policies and procedures the CA has established to ascertain the user's identity.
A CA's public keys must be distributed to all entities that trust the CA's certificates.
If a CA is a Root CA, that is, at the top of the trust hierarchy and has no superior CA to vouch for it, then the CA must distribute its public keys as self-signed certificates with an acceptable key certificate format and distribution protocol. The CA must also make its cleartext public keys available, so that relying entities can resolve the self-signed certificates. The key management-related functions performed by a CA are:

- Certificate generation
- Certificate revocation

A CA works within the context of an overall business policy known as a Certificate

Policy (CP) and functions operationally according to a Certificate Practices Statement (CPS).

Certificate Generation

To generate a certificate, the CA performs the following steps. Note that although an RA may be used, it is excluded from the process here for brevity.

- Acquire a public key from the end entity.
- Verify the identity of the end entity.
- Determine the attributes needed for this certificate, if any.
- Format the certificate.
- Digitally sign the certificate data.

Certificate Revocation

Certificate revocation must be initiated by the CA or their delegate such as an RA, which originated the end entity's certificate. The predominant vehicle for certificate revocation is known as a Certificate Revocation List or CRL. A CRL is a list generated by the CA that contains unique information about the revoked certificates which enables relying entities to determine if a certificate is valid or not. A CRL entry should be serialized, time-stamped, and signed by the CA. It is normally the relying entity's responsibility to retrieve the CRL.

A CRL must be published in a publicly available repository or directory. LDAP is the industry's current directory of choice and has been developed as an X.500 compliant protocol for the Internet. LDAP defines the directory query, data storage, and management protocols.

The timely publication of the CRL is critical in Internet and e-commerce environments. This timely publication is particularly critical during the latency period between when a CA revokes a certificate and its subsequent publication where there is a reliance upon the

certificate for making critical decisions. If for example, a high-value transaction is being negotiated and a seller is relying on the CRL so they can make the decision to sell or not, a timely CRL publication cycle may be critical to them.

There are other types of CRLs such as, delta-CRLs, and CRL-like mechanisms—for example, the Online Certificate Status Protocol (OCSP). OCSP is the most interesting because it can address the latency issue associated with standard CRL publication. OCSP is a mechanism that actually requests, in real time, a status check for a particular certificate from the originating CA. This has the benefit of not only being timely but in fact, reduces or eliminates the necessity for a CA to publish a CRL. The CRL simply waits for status check requests and responds to them as necessary.

3.3.1.4 Certificate Policy

The CP statement provides the overall guiding principles that an organization endorses regarding who may do what and how to systems and data. A CP also specifies how controls are managed. In addition, a CP names a set of rules that indicates the applicability of a public key certificate to a particular community or class of applications with common security requirements. For example, a particular CP might indicate applicability of a type of public key certificate to the authentication of electronic data interchange transactions for the trading of goods for monetary value.

3.3.1.5 Registration Authorities

A Registration Authority (RA) is an optional but common component of a PKI. An RA is used to perform some of the administrative tasks that a CA would normally undertake. Most importantly, an RA is delegated, with the CA's explicit permission, the authority to perform tasks on behalf of the CA. The primary purpose of an RA is to verify an end entity's identity and determine if an end entity is entitled to have a public key certificate issued. The RA must enforce all policies and procedures defined in the CA's CP. A typical function of an RA is to interrogate an end entity's certificate request by

examining the name, validity dates, applicable constraints, public key, certificate extensions, and related information.

3.3.1.6 Certificate Repositories

A certificate repository, sometimes referred to as a certificate directory, is an optional but common component of a PKI. Certificate distribution can be accomplished by simply publishing certificates in a directory controlled by a CA or RA. When the directory is controlled by the CA or RA, the certificate distribution process is greatly simplified. Rather than trying to distribute every certificate to a unique point, the CA simply updates the directory. A critical factor is that only the CA must have the authority to update or modify the directory, but the directory must be publicly readable.

LDAP is a good example of a simple and efficient standards based directory format and protocol that can be used for certificate distribution. In particular, LDAP is optimized for easy read access by thin clients that are a necessity in many PKI implementations.

3.3.1.7 Certificate paths

If the user does not have a trusted copy of the public key of the CA that signed the subject's public key certificate, then another public key certificate vouching for the signing CA is required. This can be applied recursively, until a chain of certificates (or **a certification path**) is discovered from a trust anchor or a most-trusted CA to the target subject (commonly referred to as the **end-entity**).

In order to accept a certificate signed by a CA, a relying party must check the signature using the public key known to belong to the signer, and the relying party must also trust the signer to act as a CA. In a certification hierarchy, the public key of a subordinate CA, and the trust in that CA, can be conveyed by a certificate signed by a higher-level CA, delegating CA authority. If the public key and authority of the higher-level CA is not already known, that can be conveyed by another certificate, and so on. The sequence of certificates called a certification path, and in this report we refer to it also as a *certificate*

chain. A certificate chain must stop at some previously known CA, referred to as the *anchor* of the chain.

3.3.2 Certification Path Validation

The certification path validation procedure for the Internet PKI describes the verification process for the binding between the subject distinguished name and/or subject alternative name and subject public key. The binding is limited by constraints that are specified in the certificates which comprise the path.

For basic path validation, all valid paths begin with certificates issued by a single most-trusted CA. The algorithm requires the public key of the CA, the CA's name, the validity period of the public key, and any constraints upon the set of paths which may be validated using this key. Depending on policy, the most-trusted CA could be a root CA, the CA that issued the verifier's own certificate, or any other CA in a network PKI. The path validation procedure is the same regardless of the choice of the most-trusted CA.
The goal of path validation is to verify the binding between a subject distinguished name or subject alternative name and subject key, as represented in the end-entity certificate, based on the public key of the most-trusted CA. This requires obtaining a sequence of certificates which support that binding.

A certification path is a sequence of n certificates where, for all x in $\{1, (n - 1)\}$, the subject of certificate x is the issuer of certificate $x + 1$. Certificate $x = 1$ is the self-signed certificate, and certificate $x = n$ is the end-entity certificate

The actions performed by the path processing software for each certificate $x = 1$ to n are described below [26]. The self-signed certificate is $x = 1$ and the end-entity certificate is $x = n$.

- Verify the basic certificate information:

- The certificate was signed using the subject public key from certificate $x - 1$. For the special case $x = 1$, this step is omitted.
- The certificate validity period includes time *T*.
- The certificate had not been revoked at time *T* and is not currently on hold, a status that commenced before time *T*.
- The subject and issuer names chain correctly; that is, the issuer of this certificate was the subject of the previous certificate.

- Recognize and process any other critical extension present in the certificate.
- Verify that the certificate is a CA certificate as specified in a basic constraints extension or as verified out of band.
- If a key usage extension is marked as critical, ensure the KeyCertSign bit is set.

If any one of the above checks fails, the procedure terminates, returning a failure indication and an appropriate reason; if none of the above checks fail on the end-entity certificate, the procedure terminates, returning a success indication together with the set of all policy qualifier values encountered in the set of certificates.

3.3.3 Certificate revocation mechanisms

Certificate revocation can be implemented in two ways [2]. One method is to use periodic publication mechanisms such as Certificate Revocation Lists (CRLs), which can be instantiated in a number of different forms or variations. The other method is on-line query mechanisms such as the Online Certificate Status Protocol (OCSP) [15].

3.3.3.1 Periodic Publication Mechanisms

3.3.3.1.1 Complete CRLs

It is possible to create complete CRLs so that all revocation information associated with a particular CA domain is posted on a single CRL. Complete CRL postings may be appropriate for some CA domains, particularly those in which the number of end entities is relatively small.

However, there are two primary criticisms levied against the use of complete CRLs:

1. The issue of scalability Given that revocation information must survive throughout the life of an issued certificate, it is conceivable that full CRL postings can become quite voluminous in some domains. Although this is not a concern for relatively modest-sized communities, this can become an issue in the larger domains.
2. The timeliness of the posted certificate revocation information As the CRL size grows, it is reasonable to expect that the CRL validity period would be fairly generous, because continual downloading of new, voluminous CRLs every time a certificate is validated would represent unacceptable performance degradation with respect to network resources.

3.3.3.1.2 CRL Distribution Points

CRL Distribution Points (sometimes referred to as Partitioned CRLs) allow revocation information within a single CA domain to be posted in multiple CRLs. CRL Distribution Points have two significant benefits over complete CRLs:

1. The revocation information can be subdivided or partitioned into more manageable pieces to avoid the proliferation of voluminous CRLs.
2. The certificates can point to the location of the CRL Distribution Point, so the relying party doesn't need to have prior knowledge of where the revocation information for a particular certificate might reside.

In summary, CRL Distribution Points offer a much more scalable alternative as compared with complete CRL postings. They can also be used to alleviate the performance issue when combined with proper partitioning and caching. However, one criticism levied against the use of CRL Distribution Points is that the CRL partitions are fixed or static. The notion of a more dynamic partitioning scheme is discussed next.

3.3.3.1.3 Delta CRLs

The idea behind Delta CRLs is to allow incremental postings of Certificate Revocation information. The revocation information can be relative to a base CRL or it can be

relative to a particular point in time. If the Delta CRL references a base CRL, the combination of the base CRL and the Delta CRL constitutes all known revocation information, within the indicated scope, at the time the Delta CRL was issued. In this particular case, the Delta CRL Indicator extension is used to point to the base CRL Number. Alternatively, the Base Revocation Information component within the CRL Scope extension can be used to indicate that this is a Delta CRL. In this case, the Base Revocation Information references a particular point in time from which this Delta CRL provides updates.

Delta-CRLs do not handle large-size revocations efficiently. Large size revocations may be unavoidable when a large scale organizational changes occur or when a Certificate Authority (CA) is compromised. The requirement of timely disseminating such a large amount of revocation information would degenerate the scalability of the delta-CRLs scheme close to that of the original CRL approach.

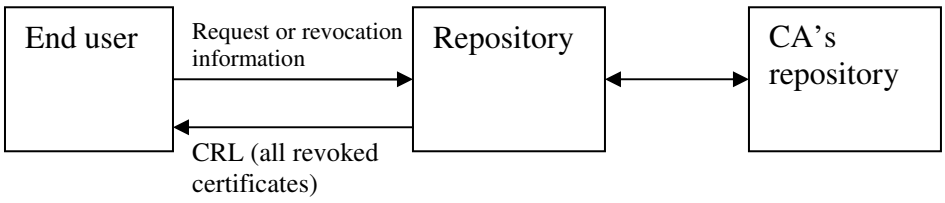

Fig 3.2 certificate revocation checking using CRL repository

Drawbacks

Normally, CRLs (Certificate Revocation Lists) are used to check the revocation status of certificates. A CRL is a list of the serial numbers of all unexpired certificates that a CA has revoked. The CA updates the CRL periodically, and each relying party retrieves the

CRL from the repository of the CA or other distribution points. The relying party can then validate the status of individual certificates from its own copy of the CRL.
CRL distribution gives rise to some problems. The CRL has a validity period indicating how long the relying party may use its CRL before it is required to obtain the updated version from the CA. If the validity period is long, the relying parties can use the cached CRL for a long time. However, it can take a long time before the serial number of a revoked certificate is listed on the relying party's CRL. In order to reduce the difference between the current revoked certificate information and the contents of the relying party CRL, the validity period of the CRL should be short. In this case, the relying parties have to access the repository of the CA or some other validation authority frequently. This may cause a heavy processing load for the repository. Another problem is the size of CRLs. If a CA has revoked many certificates, the size of the CRL for that CA may become large, and transmission of it to relying parties consumes significant bandwidth on the network. Moreover, each relying party may use only a part of the information, and the burden of sending the rest of the large CRL may be wasted.

3.3.3.2 **Online mechanisms**

3.3.3.2.1 **Online Certificate Status Protocol (OCSP)**

Version 1 of the OCSP is documented in the Request for Comments (RFC) entitled "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol—OCSP" [15]. OCSP is a relatively simple request-response protocol that offers a vehicle for obtaining on-line revocation information from a trusted entity referred to as an OCSP responder.

An OCSP request consists of the protocol version number (currently only Version 1 is defined), the service request type, and one or more certificate identifiers. The certificate identifier consists of the hash of the certificate issuer's DN, the hash of the issuer's public key, and the certificate serial number. Additional optional extensions may also be present.

Responses are also fairly straightforward, consisting of the certificate identifier, the certificate status (that is, "good," "revoked," or "unknown"), and the validity interval of the response associated with each certificate identifier specified within the original request. If the status of a given certificate is "revoked," the time that the revocation occurred is indicated; optionally, the reason for revocation may also be included.

The validity interval consists of This Update and, optionally, Next Update. However, whether the OCSP response can be cached locally will ultimately be a policy decision dictated by the governing domain. Their intended use is consistent with the same fields in a CRL as described earlier in the chapter. Like the request, the response may also contain optional extensions. OCSP also defines a small set of error codes that can be returned in the event that processing errors are encountered.

The OCSP responses must be digitally signed to provide assurance that the response is originating with a trusted entity and that it is not altered in transit. The signing key may belong to the same CA that issued the subject certificate, a trusted third party, or an entity that has been approved (through delegation) by the CA that signed the subject certificate. In any case, the relying party must be able to trust the response, which inherently implies that the relying party must trust the signer of the response. The relying party must therefore obtain a copy of the OCSP responder's public-key certificate, and that certificate must be signed by a trusted source. Requests may also be signed (useful, for example, if OCSP is operated as a "for-fee" service), but this is an optional feature within the protocol.

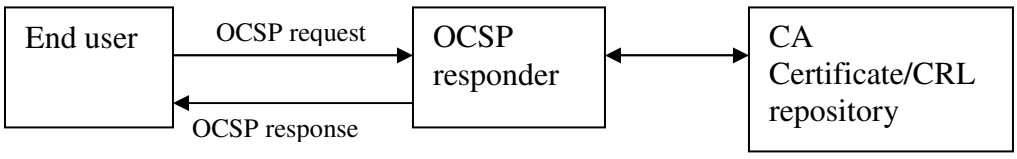

Fig 3.3 OCSP revocation checking mechanism

OCSP Messages

The *OCSP request* message is composed by a user, and sent to an OCSP Responder.

Table 3.3: OCSP request message

Version	Version number of the request syntax.
Certificate ID	A list of certificate information that should be validated.
Signature	Signature of certificate for a requesting user. (Optional)
Extensions	Optional extensions which may be processed by the OCSP Responder.

The *OCSP response* message is composed by the OCSP Responder, and sent to the requesting user. Table 3.3 shows the contents of this message.

Table 3.4: OCSP response message

Version	Version number of the response syntax.
Responder ID	Name of the responder and its key hash.
Results Certificate ID Status Validity Extensions	 Identifier of target certificate. Status of target certificate (good, revoked, unknown). Validity period of this status information. Optional extensions used for this status information.
Extensions	Optional extensions for this response message.
Signature	Signature algorithm information and signature computed across hash of this response message.

Drawbacks

On-line certificate status checking protocols have been proposed to enable real-time certificate validation to provide information on a given certificate in a timely manner for those end entities which are expected to be always on-line and to provide certificate validation service for those end entities having constrained storage, computing power, and /or network bandwidth. However, the pre-existing protocols for on-line certificate validation are considered to be short of being complete in terms of security and scalability. Unlike CRLs, they require an additional trusted entity such as a protocol responder while CRL requires only one trusted entity, the CA. Moreover, while CRL is a pre-signed entity which does not require on-the-fly signing upon responding to end entities, the protocol responses of the pre-existing on-line certificate validation protocols needs be digitally signed every time they are returned in order to make the response unforgeable. This signing requirement can significantly degrade the scalability of such on-line certificate validation responders. To remedy this scalability limitation, it became a practice to cache pre-signed certificate validation responses or to return unsigned responses especially in large scale deployments, sacrificing some level of security over scalability.

3.3.3.2.2 Authenticated Search Data Structures

Naor and Nissim [16] propose yet another data structure for revoked certificates.
They propose *authenticated search data structures* as a representation structure for lists of revoked certificates. An authenticated search data structure is a data structure that allows one to efficiently verify the status of certificates and answer a validation request with a proof that confirms the membership or non-membership of a certificate in a list of revoked certificates. Moreover, such a data structure allows one to efficiently update (insert and delete) revoked certificates. An authenticated search data structure is a special kind of authenticated dictionary (AD), that is, a dictionary that can reply to a validation

request with authenticated answer. Therefore, we refer to authenticated search data structures also as AD data structures.

An AD data structure is constructed from a CRL supplied by a CA. The AD data structure is built by a CA or a trusted validation authority, and it may be distributed to un-trusted directories that can respond to queries with authenticated replies.
The AD data structure is a perfectly balanced 2-3 Merkle hash tree with serial numbers of revoked certificates, in order, as leaf nodes. That means each interior node has 2 or 3 children and the paths from the root to the leaves have all the same length. The serial numbers at the leaves are sorted.

In order to prove that a certificate is revoked, the existence of a leaf in the AD data structure that corresponds to that certificate has to be proved with a hash chain i.e. the minimum set of hash values required to calculate the hash value at the root are used along with the signature of the root hash value signed by the CA are used as the proof attesting that the queried certificate is revoked. In order to prove that a certificate is not revoked, the existence of two certificates corresponding to two neighboring leaves in the tree with serial numbers surrounding the queried number is proved. This requires two hash chains from adjacent leaves and thus doubles the reply cost.

Figure 3.4 illustrates an AD data structure for the revoked certificates 2;5;7;8and 9. The hash chain proof that certificate 6is not revoked is established by the following values: H2;5and 7;H8;H9;in the order given, and the signature on H0. The signature check by the user implies the adjacency of 5and 7in the CRL.

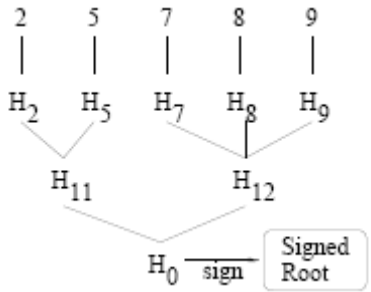

Figure 3.4: Sample AD data structure

The communication cost required to transmit the proof from the CRL repository to the end entities when an online certificate validation response is sent to the end entities is logarithmic to the number of CRL entries in a CRL. When the cardinality of the hash tree is C, $(C-1)\log_C N+1$ hash values and one signed root hash value need be transmitted as the proof of authenticity and integrity of the response.

Advantage over the other revocation mechanisms

Scalability of the responder will be significantly improved since the proposed method does not mandate the response to be signed to provide a proof of authenticity and integrity of the response. Instead, they are provided by a hierarchical authenticated data structure which can attest the presence of a certificate in a CRL by providing the value of the relevant CRL entry in the leaf level and the values of the internal nodes that together can reconstruct the signature at the root.

Response time of the on-line certificate validation process will be improved by much because of the elimination of the signing step from the response generation path.

Reduced communication cost can be achieved.

Chapter Four

ECC and PKI based key exchange

4.1 Possible solutions to avoid Man-in-the-Middle Attack

4.1.1 Key distribution center (KDC)

One way to check the binding of the server's public key to identity is to use a trusted node known as a Key Distribution Center [28]. The KDC knows keys for all the nodes. If a new node is installed in the network, only that new node and the KDC need to be configured with a key for that node. The KDC database essentially consists of a list of principals and their keys. For user principals, the key is derived from a password. Principals may also correspond to software services, such as an SSH server, etc.; their keys are randomly generated and stored in protected files where the services can access them.

When principal A wants to communicate with another—say, B—principal A first tells the KDC that it wants to talk to B. Principal A needs to do two things: prove its identity to B, and establish a shared secret with B for secure communication, called a *session key*. The KDC provides these things in a message called a *ticket*, which it sends back to A. The ticket is sealed with A's secret key, known only to the KDC and A—hence A trusts that it is genuine, and it is protected from network snooping. Unsealing the ticket, A finds the needed session key—and yet another ticket! This one, however, is sealed with B's secret key (known only to the KDC and B). A can't read this at all, but that doesn't matter; all A needs to do is send this ticket as-is to B. When B unseals its ticket, it finds A's name and another copy of the session key. Just as before, since B's ticket is sealed with B's key, B trusts that the ticket is genuine.

The meaning of each ticket is that the KDC has shared the session key with A and B. The two principals then execute a protocol which proves to each that the other does in fact hold that key—at which point, mutual authentication is accomplished. Further, the session key can be used for subsequent security functions, such as encrypting a conversation between them.

There are some disadvantages to KDCs, though:

- The KDC has enough information to impersonate anyone to anyone. If it is compromised, all the network resources are vulnerable.
- The KDC is a single point of failure. If it goes down, nobody can use anything on the network (or rather, nobody can start using something on the network-keys previously distributed can continue to be used). It is possible to have multiple KDCs which share the same database of keys, but that means added complexity and cost for extra machines and replication protocols, and added vulnerability, since there are now more targets that need to be protected.
- The KDC might be a performance bottleneck, since everyone will need to frequently communicate with it. Having multiple KDCs can alleviate this problem.

4.1.2 Distributed Key Management

Distributed key management, solves the key management problem with **introducers**. Introducers are other users of the system who sign their friends' public keys. For example, when Bob generates his public key, he gives copies to his friends: Carol and Dave. They know Bob, so they each sign Bob's key and give Bob a copy of the signature. Now, when Bob presents his key to a stranger, Alice, he presents it with the signatures of these two introducers. If Alice also knows and trusts Carol, she has reason to believe that Bob's key is valid. If she knows and trusts Carol and Dave a little, she has reason to believe that Bob's key is valid. If she doesn't know either Carol or Dave, she has no reason to trust Bob's key.

Over time, Bob will collect many more introducers. If Alice and Bob travel in similar circles, the odds are good that Alice will know one of Bob's introducers. To prevent against Mallory's substituting one key for another, an introducer must be sure that Bob's key belongs to Bob before he signs it. Perhaps the introducer should require the key be given face-to-face or verified over the telephone.

The down side is that when Alice receives Bob's public key, she has no guarantee that she will know any of the introducers and therefore no guarantee that she will trust the validity of the key.

4.3 Proposed method of authentication and key agreement

Proposed solution:

In the previous sections we have stated different alternatives but none of them can be employed because of their own limitations. Hence this thesis proposes to use a Public Key Infrastructure to make the SSH authentication strong.

4.2.1 PKI

Public key cryptography supports security mechanisms such as confidentiality, integrity, authentication, and non-repudiation. However, to successfully implement these security mechanisms, you must carefully plan an infrastructure to manage them. A public key infrastructure (PKI) is a foundation on which other applications, system, and network security components are built. A PKI is an essential component of an overall security strategy that must work in concert with other security mechanisms, business practices, and risk management efforts.

PKI supports the various business and technical needs by allowing for the identification of entities.

The distribution and management of public keys and associated certificates normally occur through the use of Certification Authorities (CAs), Registration Authorities (RAs), and directory services, which can be used to establish a hierarchy or chain of trust. CA, RA, and directory services allow for the implementation of digital certificates that can be used to identify different entities.

One of the primary principles of a PKI is the establishment of a trust hierarchy. In Internet-based communications like e-commerce, formal trust mechanisms must exist to provide risk management controls. The concept of trust, relative to a PKI, can be explained by the role of the CA. In the Internet environment, entities unknown to each

other do not have sufficient trust established between them to perform business, contractual, legal, or other types of transactions. The implementation of a PKI using a CA provides this trust. In other words, the implementation of PKI provides mechanisms to ensure trusted relationships between entities are established and maintained.

In short, a CA functions as follows. Entities that are unknown to one another, each individually establish a trust relationship with a CA. The CA performs some level of entity authentication, according to its established rules as noted in its Certificate Practices Statement or CPS, and then issues each individual a digital certificate. That certificate is signed by the CA and thus vouches for the identity of the individuals. Unknown individuals can now use their certificates to establish trust between them because they trust the CA to have performed an appropriate entity authentication, and the CA's signing of the certificates attests to this fact.

4.3.1.1 SSH authentication using x509certificate

Here an X.509 certificate is used as a host key. The server's fully qualified domain name is assigned to be the certificate's subjectAltName extension's dNSName entries. If the certificate does not contain dNSName subjectAltName extensions, the (most specific) Common Name field in the certificate Subject must be assigned by the server's host name. And the certificate's subjectAltName extension's IPaddress entries should contain the IPaddress of the host.

During the key-exchange phase of the SSH protocol, the client receives the certificate along with the server's hostkey; Instead of looking up the hostkey in a list, the client:
1. Compares the subject name and the subjectaltName in the certificate to hostname server and ipaddress of the replying server respectively. And verifies that they match
2. Validates the certificate provided using the standard PKIX path validation algorithm
3 Verifies the server's signature on the key-exchange transaction, proving it actually holds the corresponding private key

If the key passes all these tests, then the client considers the key valid, and server authentication succeeds. Here the clients do still need to get the CA key in a trusted manner. But it's much easier to distribute or update a single key that changes very infrequently, than to manage a constantly changing known-hosts list.

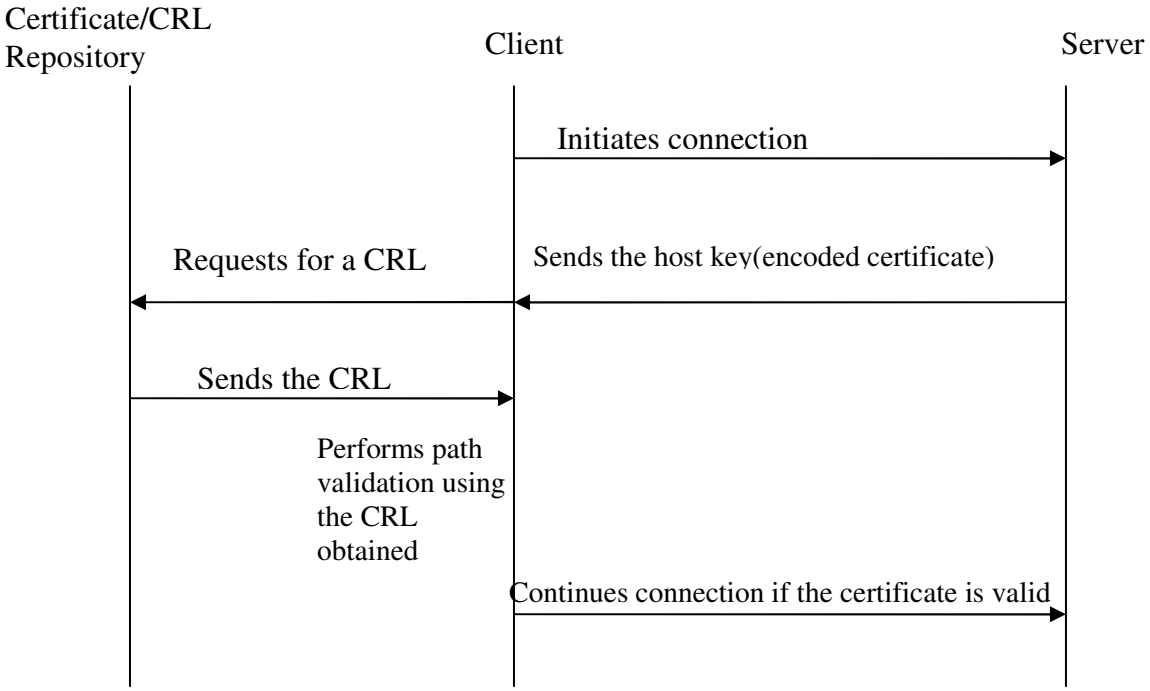

Fig 4.2 Authentication using public-key certificates

3.2.1.2 Public key cryptographic operations

The public-key cryptographic operations performed by a client and server in different modes of the public key algorithm including the online OCSP responder public key operation are summarized as follows.

Table 4.1 public-key operations

	OCSP	**CRL**
Client	Server_Certverify+OCSP resp.verify + DSAverify/RSAverify +DHoperation	Server_Certverify+CRLverify + DSA/RSAverify+DHoperation
Server	2(DSAsign/RSAsign) +DHoperation	RSA/DSAsign+DHoperation
OCSP responder	DSAverify/RSAverify +OCSPresp.sign	

SSH allows both client- and server-side authentication. However, due to the difficulty of managing user certificates across multiple client devices, the former is rarely used.
User authentication, in such cases, happens at the application layer, *e.g.* through passwords sent over an SSH protected channel. Authentication of the SSH client is not discussed further in this paper.

How is it secure?

A certificate is uniquely identified by either its Subject Name, (which is a composite of Common Name, Organization Unit, Organization, Location, State, Country, Email) or one of the X509v3 extensions that can be added to any server certificate issued. X509v3 extensions may include a server name, or a DNS address. In our case, the server name's would be used as a unique identifier and would be entered as a Common Name. All the fields in a certificate, including the public key, are signed by the corresponding private

key component for CA certificate. The self-signed root certificate can then also be appended to the server certificate. This forms a 2-node certificate chain.

1. Server sends the certificate to the client machine. An attacker who has a different server certificate issued by the same certificate authority can intercept the server certificate. Suppose he wants to pose the real server. He replaces the server certificate, with his own certificate. Certificate verification would fail in such a scenario because when the ssh client looks at the uniquely identifying field of the certificate, it would not be the one it expected. The IP address x509v3 extension would not match. If the attacker forges it to be the same, he would not be able to modify his signature to match the forged IP extension, as the CA generated the signature, and therefore the certificate verification would fail again. The user would verify the signature of the attacker using the CA's public key and match it with the credentials provided which would not be the same. This establishes the fact that the attacker cannot forge a certificate, even if he has a certificate issued by the same certificate authority.
2. User accesses the certification path from repository (ldap server). An attacker can intercept and pass a different modified certification path. However, when the user validates that path it will not pass the validation test. Because each certificate in the path should be signed by their corresponding CA's private key and finally the final certificate in the path should be signed by the most trusted CA. so an attacker can't forge a certification path unless he/she has access to the private key of the CA which is supposed to be very secure.
3. Suppose the server certificate changes. In such a scenario, an attacker could use the old invalid server certificate from a previous session and successfully pose as the server. The user would receive the old certificate, and he would retrieve the CA certification path, from the ldap server (repository). As the CA's public key has not been modified, the user would validate the invalid server certificate, and send his username and plaintext password to the attacker. But here the client will also check the revocation status of the server certificate using Certificate Revocation Lists or OCSP. And if it has been revoked, he would not continue authentication with an invalid certificate.

4.2.2 Proposed key-exchange scheme

4.2.2.1 ECC Basics

At the foundation of every public key cryptosystem is a hard mathematical problem that is computationally infeasible to solve. For instance, RSA and Diffie-Hellman rely on the hardness of integer factorization and the discrete logarithm problem respectively. Unlike these cryptosystems which operate over integer fields, the Elliptic Curve Cryptosystems (ECC) operates over points on an elliptic curve.

The fundamental mathematical operation in RSA and Diffie-Hellman is modular integer exponentiation. However, the core of elliptic curve arithmetic is an operation called *scalar point multiplication*, which computes $Q = kP$ (a point P multiplied k times resulting in another point Q on the curve). Scalar multiplication is performed through a combination of point-additions (which add two distinct points together) and point doublings (which add two copies of a point together). For example, $11P$ can be expressed as

$11P = (2 * ((2* (2* P)) + P)) + P$.

The security of ECC relies on the hardness of solving the Elliptic Curve Discrete Logarithm Problem (ECDLP), which states that given P and $Q = kP$, it is hard to find k. While a brute-force approach is to compute all multiples of P until Q is found, k would be so large in a real crypto-graphic application that it would be infeasible to determine k in this way.

Besides the curve equation, an important elliptic curve parameter is the *base point*, G, which is fixed for each curve. In the Elliptic Curve Cryptosystem, the large random integer k is kept private and forms the secret key, while the result Q of multiplying the private key k with the curve's base point G serves as the corresponding public key.

Not every elliptic curve offers strong security properties and for some curves the ECDLP may be solved efficiently.

Since a poor choice of the curve can compromise security, standards organizations like NIST and SECG have published a set of recommended curves with well understood security properties.

Elliptic Curve Diffie Hellman (ECDH) and EllipticCurve Digital Signature Algorithm (ECDSA) are the Elliptic Curve counterparts of the Diffie-Hellman key exchange and Digital Signature Algorithm, respectively. In ECDH key agreement, two communicating parties A and B agree to use the same curve parameters. They generate their private keys, *kA* and *kB* and corresponding public keys *QA = kA:G* and *QB = kB:G*. The parties exchange their public keys. Finally each multiplies its private key and the other's public key to arrive at a common shared secret
kA:QB = kB:QA = kA:KB:G.

4.2.2.2 The modified key-exchange method

In this study, our goal is to maintain strong scalable security (protocol) scheme that is applicable for all types of internet hosts or scheme that ease practical deployment on the Internet.
SSH uses public key to identify an entity by using digital signatures and deffie-hellman key exchange algorithm to establish a shared secret key. In here, we can see that two public-key cryptographic operations will be performed for every key exchange. But if the communicating parties can agree on a common p and g, and server's deffie-hellman public key component is certified by a trusted party, the public key algorithm used for server authentication can be removed. This will lead to a reduction in computational intensive signature generation and verification operations.
And to accomplish a required level of security with relatively small key size we propose the use of ECC. So the proposed scheme uses server certificates attesting the ECDH public component.

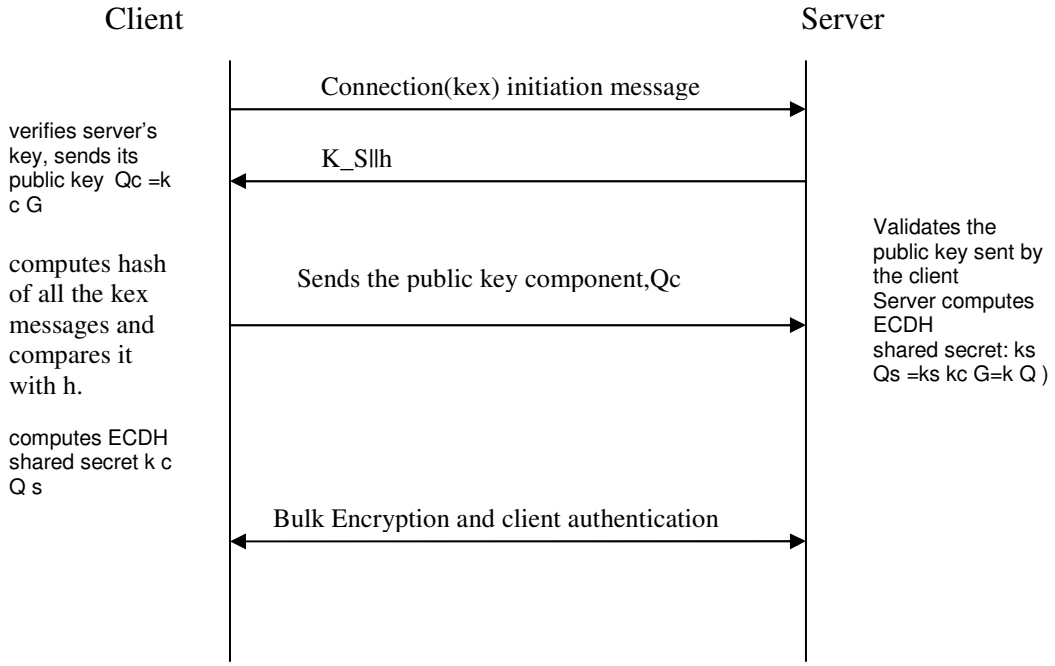

Fig 4.3 The proposed key exchange handshake

The Server Certificate contains the server's ECDH public key signed by a certificate authority using ECDSA. After validating the ECDSA signature, the client conveys its ECDH public key to the server in the Client Key Exchange message. Next, each entity uses its own ECDH private key and the other's public key to perform an ECDH operation and arrive at a shared secret key. The derivation of the session symmetric keys is unchanged.

4.2.2.3 PKI proposed revocation method

Certificates are valid until they expire, unless they are revoked beforehand. Many things can happen that require the revocation of a certificate. For example, the secret key may be lost or irreversibly destroyed, the certificate holder may cease operation, the certificate holder's identifier may need to be updated due to a name change, one of the (other)

attributes in the certificate may have become invalid, or the secret key may have been compromised.

While revocation is an exceptional circumstance, the task of verifiers to check the revocation status of unexpired certificates unfortunately is not. They must either have the certificate status validated online (at the time of the communication or transaction) or regularly download a digitally signed update of a blacklist called the Certificate Revocation List (CRL). In both cases the status of certificates must be maintained by the CA (or by a special Revocation Authority). Note that the certificate revocation or validation data must itself be authenticated.

And revocation mechanism is based on one of the following: full CRLs, delta-CRLs, CRL Distribution Points and OCSP.

Online certificate validation avoids the need for verifiers to manage their own versions of a CRL and to deal with certificates they are not interested in. One of the primary problems is scalability to large communities, not in the least because responses to queries must be authenticated by the trusted central database.

Distribution of CRLs (or their updates), is more attractive in many respects, but creates a lag between the time a certificate becomes invalid and when it appears on the next CRL update. If validity periods are long, CRLs will grow and additional computing resources are needed for searching and storing them.

Either way, certificate revocation seriously reduces the finality of secure communications and transactions.

4.2.2.3.2 A PKI authentication mechanism with Authenticated Dictionaries

By using Authenticated Dictionaries as revocation status responder we can eliminate the response delays caused by OCSP responders and long sized CRLs given by CRL repositories.

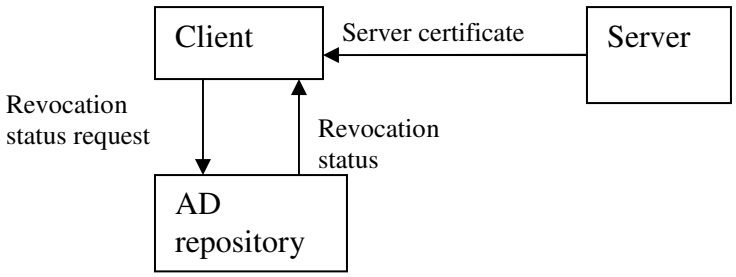

Fig 4.4 Overview of authentication using AD

4.2.2.3.2 Public-key cryptography in the modified scheme

Table 4.2 Proposed KEX with PKI

	OCSP	CRL
Client	ECDSAverify+OCSPverify +ECDHoperation	2*ECDSAverify +ECDHoperation
Server	OCSPreq.sign+ECDHoperation	ECDHoperation
OCSP server	OCSPreq.verify+OCSPresp.sign	

The client performs an ECDSA verification to verify the server's certificate and RSA verification to verify the root hash value of the revocation response and then an ECDH operation using its private ECDH key and the server's public ECDH key to compute the shared secret key. All the server needs to do is perform an ECDH operation to arrive at the same secret.

Table 4.3 Proposed KEX with proposed PKI

	AD
Client	2*ECDSAverify +ECDHoperation
Server	ECDHoperation

Chapter 5
Simulation, Results and Result Analysis

5.1 Performance Metrics

While Table 2 identifies the individual cryptographic operations for the two sides in an SSH connection, the relevant performance metrics for the client and server may be quite different. An appropriate choice of such metrics is key to evaluate “perceived" performance accurately. In the case of a server that aggregates thousands of SSH requests, connection throughput (number of connections handled per second) is important. On the other hand, for a typical client which sequentially establishes one SSH connection at a time, connection latency is more important. In addition, the client storage capacity to process the received message might be limited especially in the case of CRL download. So the size of the message received from the CRL repository is another issue for the client.

Handshake KEX Latency This is the total time spent on performing cryptographic operations on the client and server. For instance, in the case of an RSA handshake without client authentication, this is the sum of times spent by the client doing two RSA public key operations and the server doing one RSA private key operation.

Server KEX Throughput This is the rate at which the server can perform the cryptographic operations needed in the handshake. The client's performance is not a factor because the server can interleave multiple connections from different clients. The Server Crypto Throughput measured in connections per second is 1000/(*server-kex-time* in milliseconds).

Besides the public-key cryptographic operations, a full key-exchange handshake also involves other delays like network latency. Therefore, the handshake key exchange crypto latency serves as a lower bound on the total key-exchange latency. Similarly, the

Server Crypto Throughput figure only provides an upper bound on the actual SSH connection rate.

Repository status proof size This is the size of the response data from the repository in return to the revocation status request by the client or the server.

Communication cost (size of response message)

	OCSP	CRL	AD
size	1	n	$(C-1)\log_C N+1+1$

5.2 Experiments performed

The experiment is performed using

Signature algorithms: SHA1withRSA, SHA1withECDSA

Key exchange: DH and ECDH

For each cipher suite, we studied three different security levels — 1024, 2048 and 3072 bits for RSA and 160, 224 and 521 bits for ECC. For ECC, we chose three elliptic curves (secp160r1, secp224r1, and secp521r1 [23,24]) defined over prime integer fields recommended by NIST and/or SECG.

5.3 Platform

java 1.5 , openssl and Legion of Bouncy Castle cryptographic api to simulate the operations performed during the key exchange phase of the SSH security protocol.

The sample implemented program is run on a PC with Intel Pentium IV processor having a speed of 1.7 GHZ, and 384MB RAM.

5.4 Results

5.4.1 Crypto latency

RSA based key-exchange

Table 5.1 average crypto latency in milliseconds using RSA based key-exchange

512 DH

	pk	crl	ocsp	ad
1024	289	722	1226	676
2048	671	980	1722	820
3072	1820	2172	4937	2140

For 1024 DH

	pk	crl	ocsp	ad
1024	1004	1593	2515	1457
2048	1446	4659	3211	
3072	1968	2914	3891	

ECC-based key-exchange

	crl	ocsp	ad
163	1081	1778	1022
224	1140	1906	1023
521	1550	2472	1332

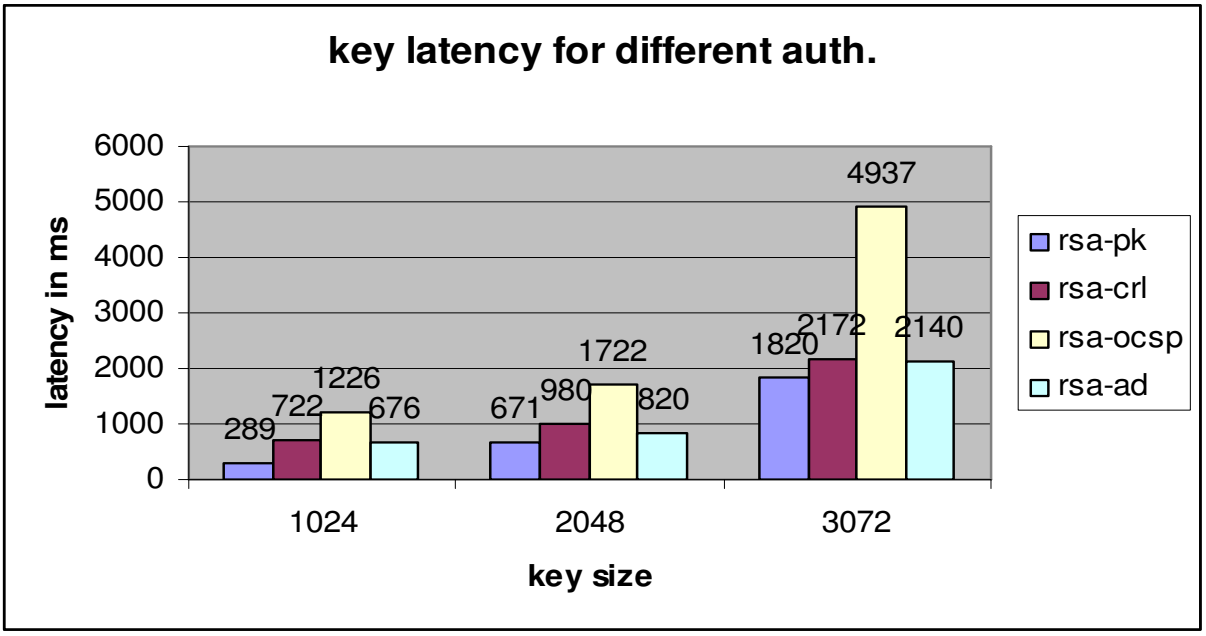

Fig 5.1 kex latency of RSA based method for different authentication methods

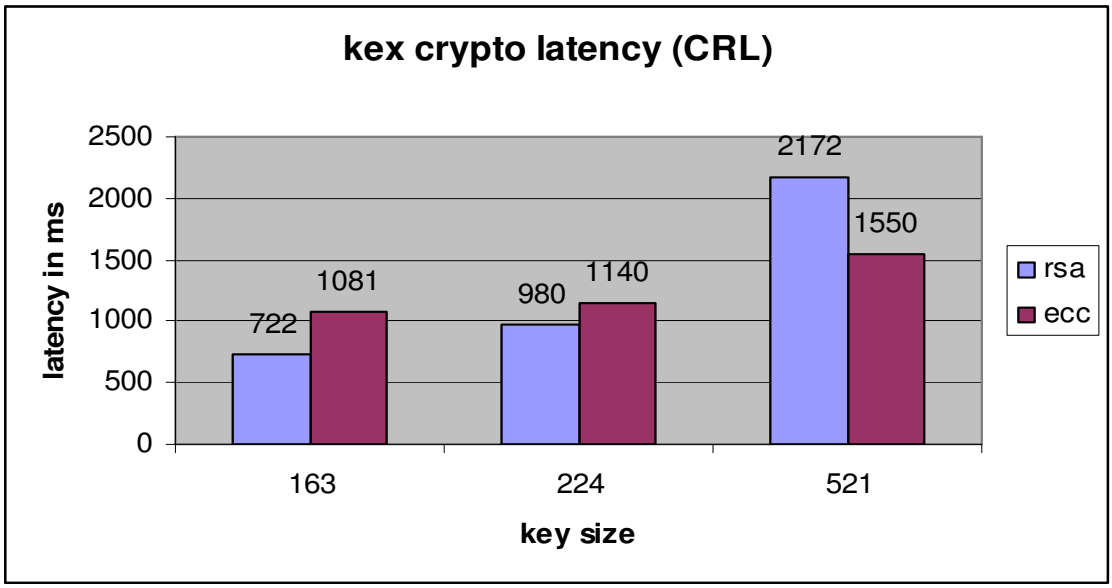

Fig 5.2 kex time of ECC based and RSA based with total CRL revocation mechanism

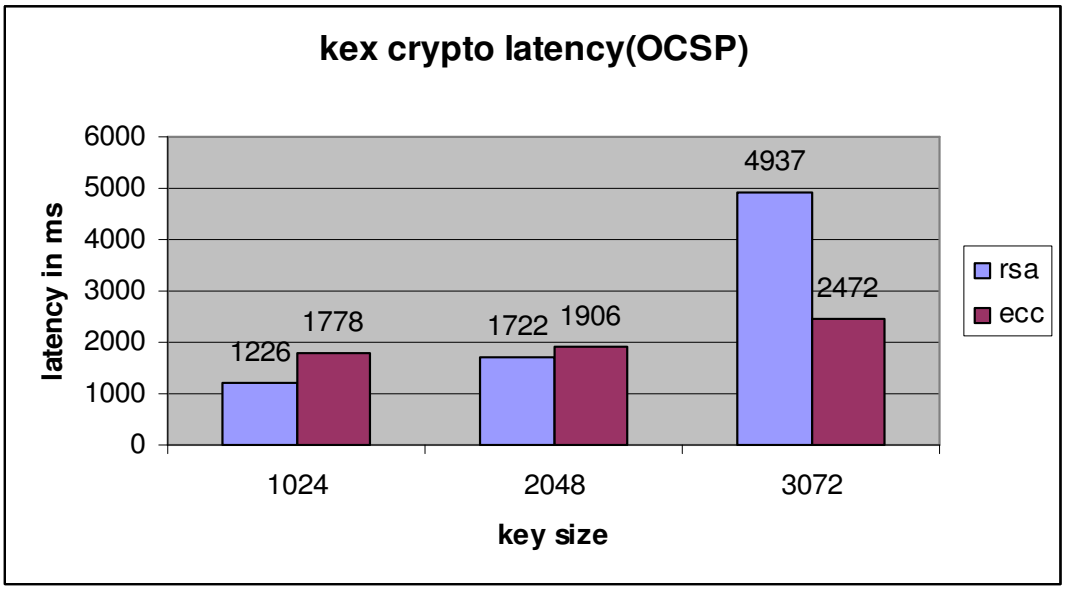

Fig 5.3 kex time of ECC based and RSA based with OCSP revocation mechanism

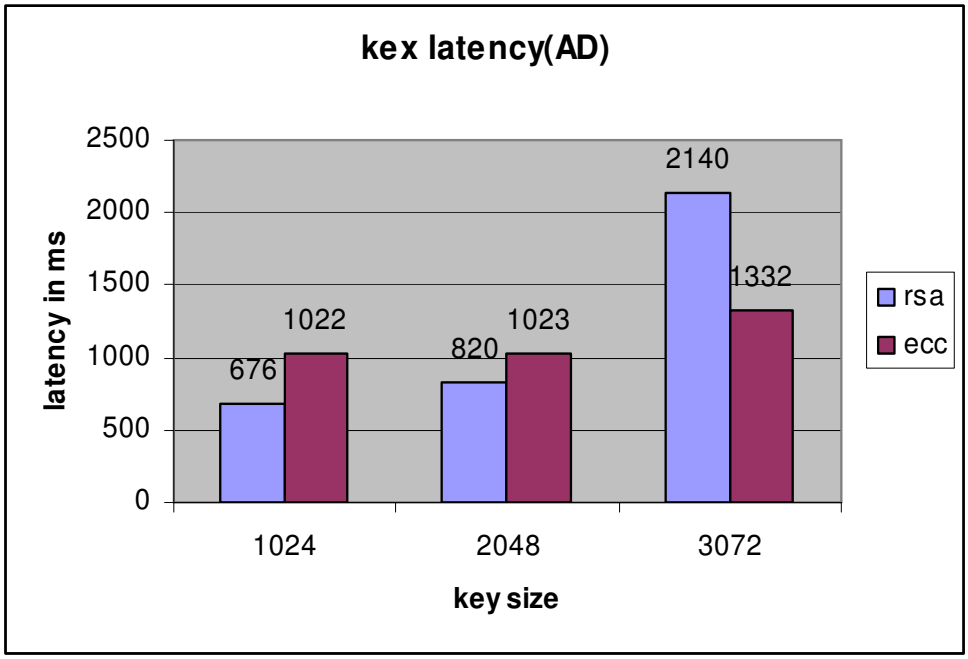

Fig 5.4 kex time of ECC based and RSA based with AD revocation mechanism

5.4.2 Server throughput

Server key-exchange time

RSA

Table 5.3 average time consumed by the server during the RSA kex in milliseconds

	pk	crl	ocsp
1024	164	177	672
2048	493	523	1344
3072	1851	1831	3422

server latency			
	pk	crl	ocsp
1024	398	476	1228
2048	950	934	2406
3072	1537	1937	3195

ECC

Table 5.4 average time consumed by the server during the ECC kex in milliseconds

	crl	ad	ocsp
163	159	165	828
224	181	187	856
521	441	335	974

Server Throughput

RSA

Table 5.5 average RSA based kex throughput of the server

DH 512

	pk	crl	ocsp
1024	6.097561	5.649718	1.488095
2048	2.028398	1.912046	0.744048
3072	0.540249	0.54615	0.292227

DH 1024

	pk	crl	ocsp
1024	2.512563	2.10084	0.814332
2048	1.052632	1.070664	0.415628
3072	0.650618	0.516262	0.312989

ECC

Table 5.6 average ECC based kex throughput of the server

	crl	ad	ocsp
163	6.289308	6.060606	1.207729
224	5.524862	5.347594	1.168224
521	2.267574	2.985075	1.026694

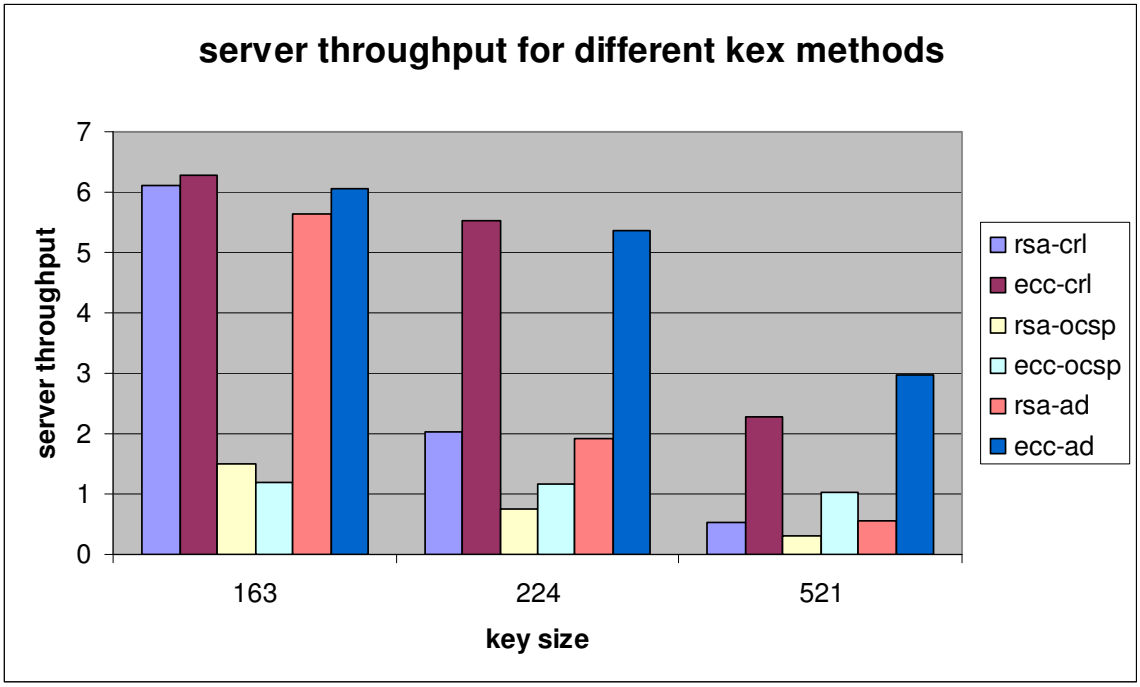

Fig 5.5 server crypto-throughput measured for different kex methods

4.4.3 Message size of the revocation status

Table 5.7 response size of the different revocation checking methods

no. of revoked certs	CRL	OCSP	AD
10	10	1	4
20	20	1	5
30	30	1	6
50	50	1	6
100	100	1	7
150	150	1	8
200	200	1	7
300	300	1	9
400	400	1	9
1000	1000	1	8
2000	2000	1	11
5000	5000	1	12
6000	6000	1	13
10000	10000	1	11

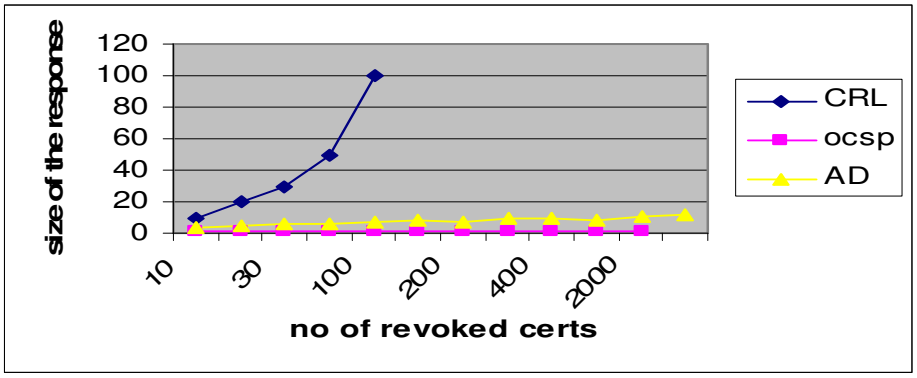

Fig 5.6 certificate status response message size

5.5 Analysis of results

The results shown above show the performance advantage of ECC over RSA for different security levels. Note that the performance advantage of ECC gets even better than its key-size advantage as security needs increase. As table 5.1 shows introducing PKI has cause an increase in crypto-latency. But in the modified scheme even if there is a PKI authentication its performance is better than the previous kex using public key authentication. And among the other revocation methods AD responders reply with smaller transferable message sizes even in case of very big CRL sizes. But the response time is the almost the same with that of the total CRL method. As expected kex using OCSP has shown the worst performance in crypto-latency and server throughput, because of its computational overhead for additional signature generation and verification. So ECC with AD repositories outperforms the others and has a very good security level.

Chapter 6

Conclusions and future work

6.1 Conclusions

It is discussed that (in theory) ECC has higher security per bit. But the main objective of this thesis is to see its performance when used with SSH key-exchange phase and to check that whether it is applicable or not.

The results of the thesis work suggest that the use of ECC based key exchange offers significant performance benefits to SSH clients and servers especially as security need increases.

In addition, the thesis has tried to show that how certificate authentication resolves the key ownership problem of public key authentication and in turn eliminates the Man-in-the-Middle Attack but validating a server certificate, especially with revocation status checking, has public-key (computational computational-intensive) operations like signature generation and verification. So the proper PKI should be selected so that SSH2 can perform well even with larger group of communication entities. Among the revocation mechanisms discussed Authenticated Dictionaries shows good performance.

6.2 Future work

This thesis has used certificates for server authentication and assumed clients prefer password-username method of authentication. So the performance analysis should also be done when clients use public keys and certificates for authentication, this can be treated as a future work.

References:

[1] Daniel J. Barrett & Richard E. Silverman," SSH The Secure Shell, The Definitive Guide", O'REILLY & Associates, 2005

[2] Carlisle Adams, Steve Lloyd,"Understanding PKI: Concepts, Standards, and Deployment Considerations", November 2002.

[3] Bruce Schneier, "Applied Cryptography: Protocols, Algorithms, and Source Code in C", John Wiley & Sons, 1996.

[4] A. Menezes, P. Van Oorschot, S. Vanstone," Handbook of Applied Cryptography", CRC Press, 1997.

[5] William Stallings, "Cryptography and Network Security: Principles and Practice", Prentice Hall, 2003.

[6] D. Hankerson, A. Menezes, S. Vanstone, "Guide to Elliptic Curve Cryptography", Springer, 2004.

[7] Markus Friedl, et al., "Diffie-Hellman Group Exchange for the SSH Transport Layer Protocol", Internet Draft, January 2002.

[8] Saarenmaa & Galbraith "X.509 authentication in SSH ", Internet Draft, February 28, 2006

[9] T. Ylonen, T. Kivinen, M. Saarinen, T. Rinne, and S. Lehtinen, "SSH Protocol Architecture", Network Working group, Internet Draft, January 31, 2002.

[10] T. Ylonen, T. Kivinen, M. Saarinen, T. Rinne, and S. Lehtinen, "SSH Transport Layer Protocol", Network Working group, Internet Draft, January 31, 2002.

[11] T. Ylonen, T. Kivinen, M. Saarinen, T. Rinne, and S. Lehtinen, "Authentication Protocol", Network Working group, Internet Draft, February 28, 2002.

[12] T. Ylonen, T. Kivinen, M. Saarinen, T. Rinne, and S. Lehtinen, "SSH Connection Protocol", Network Working group, Internet Draft, January 31, 2002.

[13] W. Ford ,R. Housley , W. Polk , D. Solo ,"Internet X.509 Public Key Infrastructure certificate and Certificate Revocation List (CRL) Profile",RFC 3280, April 2002

[14] Yasir Ali and Sean W. Smith, “Flexible and Scalable Public Key Security for SSH ”, submitted for the Second Annual PKI Workshop, Feb 2003

[15] M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. “X.509 Internet Public Key Infrastructure: Onlice Certificate Status Protocol – OCSP” Technical Report RFC2560, IETF, June 1999.

[16] M. Naor and K. Nissim. “Certificate Revocation and Certificate Update” *IEEE Journal on Selected Areas in Communications*, 18(4):561–570, 2000.

[17] Sean Mullan “Java™ Certification Path API Programmer's Guide”, *February 2003*

[18] S. Blake-Wilson and T. Dierks, “ECC Cipher Suites for TLS", Internet draft <draft- ietf-tls-ecc-01.txt>, work in progress, Mar. 2001.

[19] Vipul Gupta, Douglas Stebila, Stephen Fung, Sheueling Chang Shantz, Nils Gura, Hans Eberle,” Speeding up Secure Web Transactions Using Elliptic Curve Cryptography”

[20] R. Rivest, A. Shamir, L. Adleman, “A Method for Obtaining Digital Signatures and Public-key Cryptosystems”, http://theory.lcs.mit.edu/~rivest/rsapaper.pdf, 1977.

[21] OpenSSL Project, see http://www.openssl.org/.

[22] Robert Zuccherato, “Elliptic Curve Cryptography Support in Entrust”, May, 2000.

[23] Certicom Research, “SEC 2: Recommended Elliptic Curve Domain Parameters”, Standards for Efficient Cryptography, Version 1.0, Sep. 2000.

[24] NIST, “Special Publication 800-57: Recommendation for Key Management. Part 1: General Guideline”, Draft Jan. 2003

[25] W. Diffie, M. Hellman, “New Directions in Cryptography”, http:// crypto.csail.mit.edu/classes/6.857/papers/diffie-hellman.pdf, 1976.

[26] Man Young Rhee ,“**Internet Security** Cryptographic Principles, Algorithms and Protocols”, 2003.

[27] Joel Weise, “Public Key Infrastructure Overview”, August 2001.

[28] C. kaufman, R Perlman and M. Speciner, “ Network Security- Private communication in a public world”,2006.