INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

University of Alberta

A review of cryptography as applied to computer networks

by

Freddy Lap Kei Yeung 

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of Master of Science

in

Applied Mathematics

Department of Mathematical and Statistical Sciences

Edmonton, Alberta

Spring 2005

Abstract

This thesis is a survey about the network security between a sender and a receiver. We will address some of the possible network security problems and their solutions. The security problems can be classified as an internal attack or an external attack. The internal attack only involves the sender and the receiver, while the external attack involves a third party. In the first chapter, we will discuss the four security services and the related algorithms. In the second chapter, we will discuss the next generation of cryptography algorithms which may be implemented in the future.

Acknowledgements

My girlfriend, Kapo Wong, and my mom, Cindy Lam, love me and have made me happy while I was researching and writing the thesis. Without their support, I think I cannot finish my thesis.

I also thank my supervisor, Ioanis Nikolaidis, and my co-supervisor, Andy Liu, for their patience, kindness, and support.

Finally, I would like to thank the Department of Mathematical and Statistical Sciences for the financial support that I received through a Teaching Assistantship.

Contents:

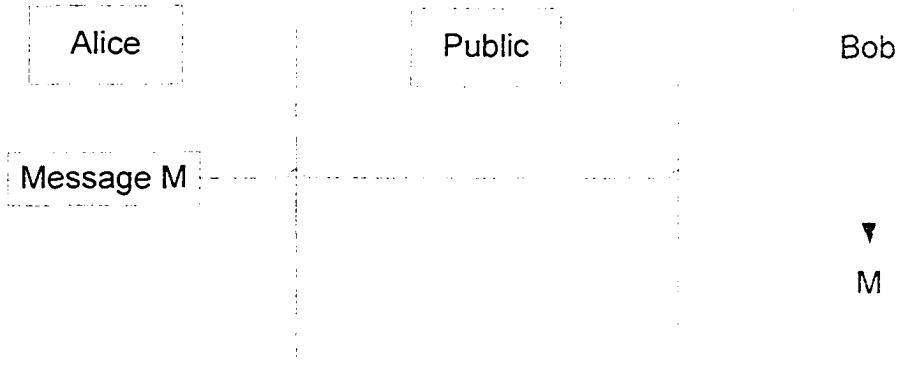
Chapter 0: Network Security and Cryptography Algorithm

0.1 Introduction

We live in the age of communication. Most transactions nowadays are conducted by parties in remote locations, linked by an electronic network. While this mode of operation does provide unrivaled convenience, it also presents a number of problems. For instance, faulty equipments or noisy channels may cause errors in the transmitted messages. In the Appendix, we give an account of how error-correcting codes may address this problem. In this thesis, we are mainly concerned with difficulties arising from dishonesty of human beings.

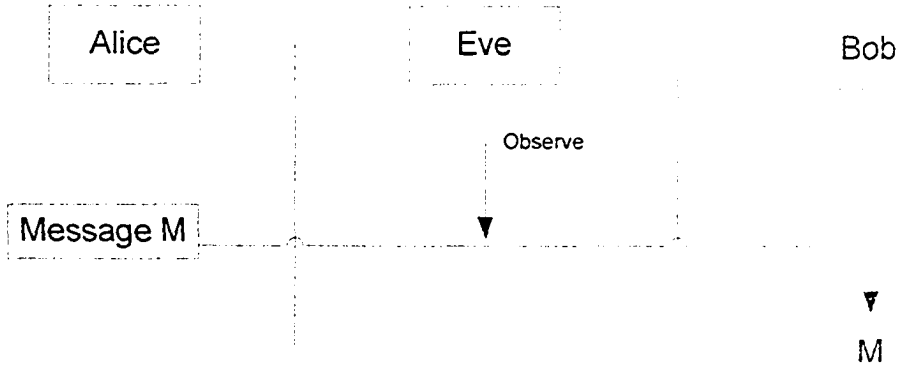
The basic scenario involves two parties who are communicating with each other via network connection. Of course, information flows in both directions. In this thesis, we shall always identify the sender as Alice and the receiver as Bob. For convenience, we may assume that the communication is a sequence of messages where each message has one intended recipient. (See Figure 1)

Figure 1



There are two kinds of security risks, internal and external. In the former case, Alice and Bob may be working against each other. In the latter case, we assume that they work together against a malfeasant third party. In this thesis, we shall identify the potential villain as Eve. (See Figure 2)

Figure 2



Let us consider first external security risks. What do Alice and Bob want to protect against Eve? There are three main areas: [5]

1) Confidentiality.
   Alice and Bob want to ensure that the information in the computer system and transmitted information are accessible only to them.

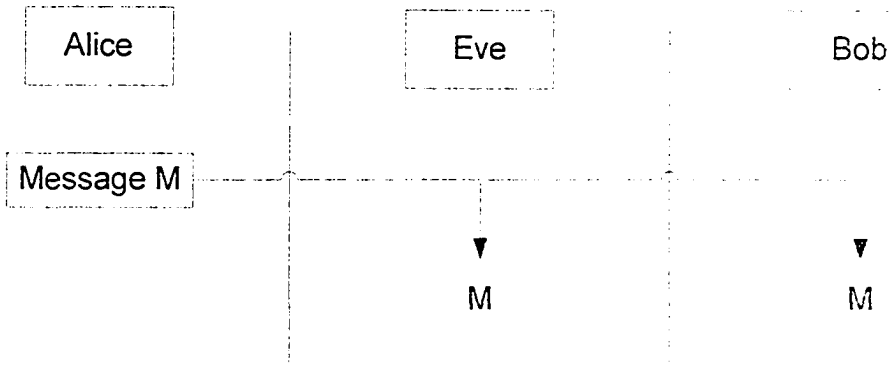
2) Integrity.
   Alice and Bob want to ensure that only they are able to modify the transmitted information.

3) Authentication.
   Bob wants to ensure that the transmitted information does come from Alice.

Accordingly, Eve can mount attacks on these areas: [5, pg 7]

1) Interception.
   This is an attack on confidentiality. Eve simply gains access to the transmitted information. (See Figure 3)

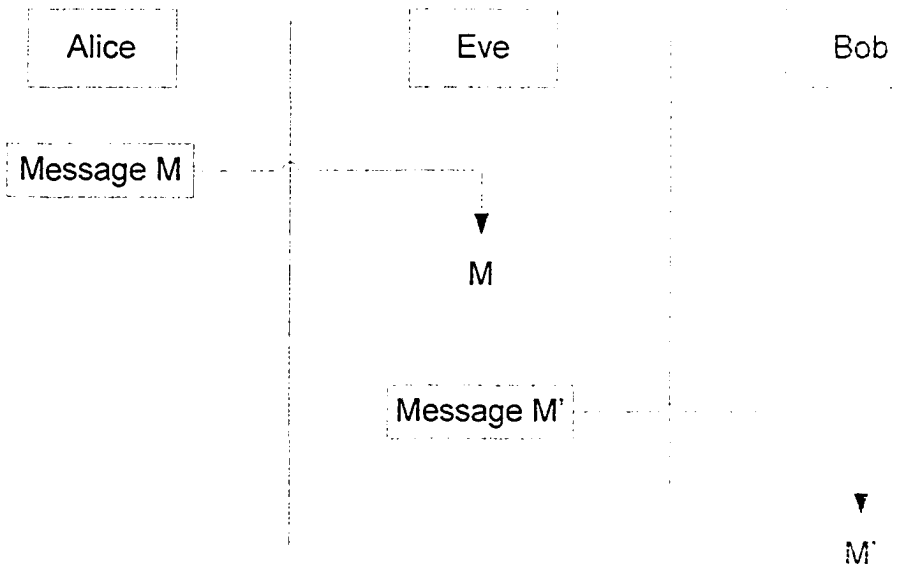
Figure 3



2) Modification.

This is an attack on integrity. Eve does not only gain access to but also tampers with the transmitted information. She can change the values in a data file of a program so that it performs differently, or change the content of the message being transmitted. (See Figure 4)

Figure 4



3) Fabrication.

This is an attack on authenticity. Eve sends Bob a message purported to be from Alice. In some instances, Eve may simply replay earlier valid messages from Alice. (See Figures 5 and 6)

Figure 5

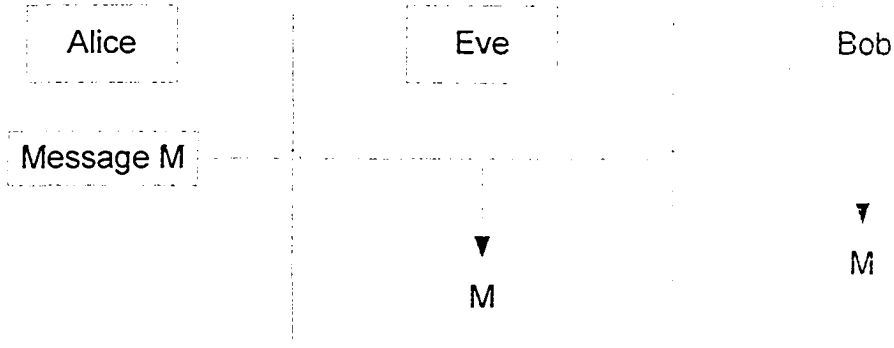
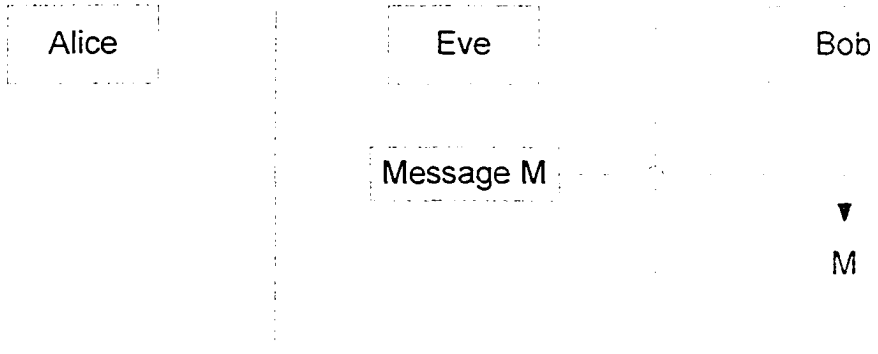


Figure 6



Attacks are difficult for the system to prevent, as it requires physical protection of the whole communication path from one end to the other at all times. However, it is feasible to prevent the success of such attacks. The interception attack does not involve any alteration of the data. Such attacks are called *passive attacks*, and it is difficult for the system to detect them. The focus is on making the intercepted information unintelligible. The modification and fabrication attacks, on the other hand, are *active attacks*. Here, the focus is on detecting them and recovering from

any disruptions or delays caused by them.

The protection against interception is dealt with in chapter 1. The detections of modification and fabrication are dealt with in chapter 2 and 3. In chapter 4, we will turn our attention to internal security risks. When Bob receives a message from Alice (with authentication) asking him to perform a certain task, Bob would like to be able to produce a copy of the message later and prove that Alice did send it. In other words, Bob wants *non-repudiation*.

Chapter 1: Confidentiality

Confidentiality is the property that the information remains unknown to unauthorized parties. Usually, the information is a sequence of bits of length m which needs to be protected for some time period. In this way, only Alice and Bob can read the information and no other unauthorized party (such as Eve) can read the information during that time period.

Note that it may be possible for the information to eventually become known after a substantially long period of time. For example even state secrets become public after several decades, or in other cases the information is of no value in the far future (for example the confidentiality of a banking transaction that took place fifty years ago is of little significance). We are therefore interested in maintaining confidentiality for a period of time as long as possible, at least long enough to not compromise the value of possessing such information (versus not possessing it).

1.1 Encryption/Decryption

The principal mechanism to protect the confidentiality of message m for a time period is *encryption*. One of the first known algorithms transforms plaintext (Original Information) into ciphertext (Unreadable Information) is called Caesar Algorithm [1]. The Caesar Algorithm works as the following:

The encryption function $E(x, y)$ encrypts the plaintext message x with key y to become ciphertext. Assume each plaintext corresponds to its own ciphertext letter. For example:

Plaintext	a	b	c	d	e	f	g	H	I	g	...
Ciphertext	D	E	F	G	H	I	J	K	L	M	...

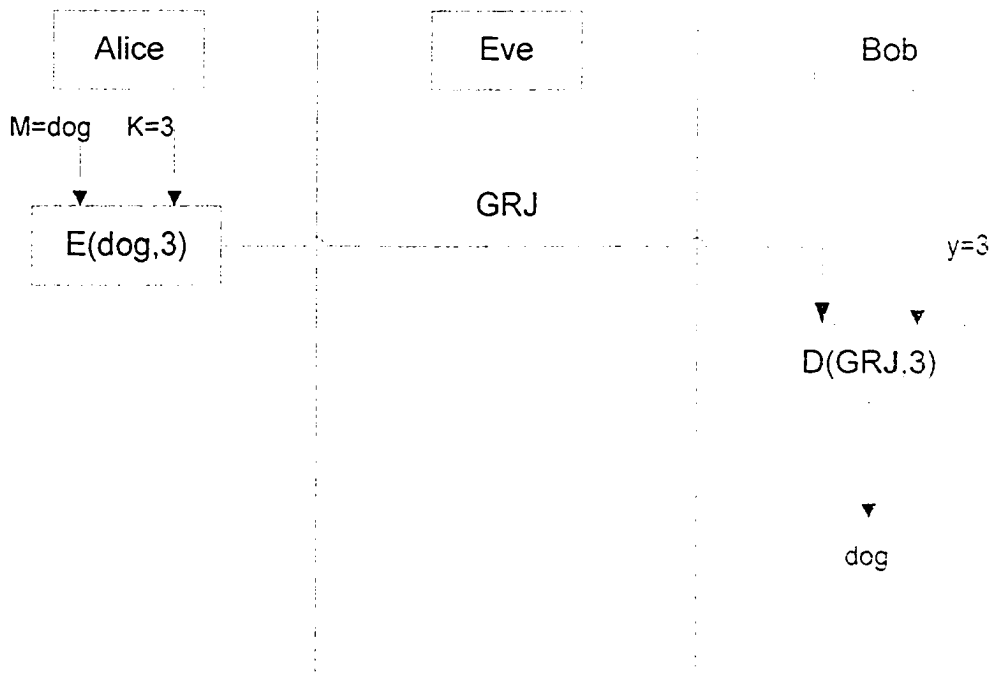
Assign a number to each plaintext letter in sequence. Start with plaintext letter "a" equal to 0 and letter "b" equals to 1 and so on for the rest of the letters. There will be 26 assigned values. From the example, we can see that the plaintext

letter "a" has a corresponding ciphertext letter "D" which is the letter 3 positions down in the alphabet. In this example the key is 3. Likewise "b" maps to "E". That is, the number of positions by which we shift is the key.

The general form of the encryption function for Caesar's Algorithm is $E(x, y) = x + y \pmod{26}$, where x is the plaintext and y is the key. Suppose we wish to encrypt the word "dog" using the key $y = 3$. The ciphertext will become "GRJ". Alice and Bob use the same key in order to encrypt the plaintext into ciphertext in both directions.

The transformation from plaintext to ciphertext is called encryption while the transformation from the ciphertext into plaintext is called decryption. Using Caesar's Algorithm in our example, if Bob receives the ciphertext "GRJ", he needs to use the key with the corresponding decryption function to recover the plaintext. In general, the decryption function is $D(C, y) = x$ where C is the ciphertext and y is the key share with Alice. For Caesar's Algorithm, the decryption function will be $D(C, y) = C - y \pmod{26}$. So if the ciphertext is "GRJ", Bob should be able to derive "dog" by applying $D(C, y) = C - 3$, since the key is $y = 3$. (See Figure 1.1)

Figure 1.1



Where:

M	= A message where Alice wants to send to Bob
K	= A key which share by Alice and Bob share.
$E(x, y)$	= The encryption function. We assume not only Alice and Bob know it, but Eve also knows it.
$D(C, y)$	= The decryption function. We assume not only Alice and Bob know it, but Eve also knows it.

Caesar's algorithm uses the same key for encryption and decryption. This is a major weakness: if any third party retrieves the key, then he or she can perform both the encryption or decryption function.

One of the most important assumptions in modern cryptography is Kerckhoffs's Principle [6, pg 4]: *In assessing the security of a cryptosystem, one should always assume the enemy knows the method being used.* Hence, the strength of the cryptosystem should be based on the key and not on the algorithm. Consequently, we always assume that Eve knows the algorithm that is used to perform encryption.

1.2 Brute force attacks

Once the ciphertext is made public, we should always assume that Eve can get hold of the ciphertext and decrypt it by applying all of the possible decryption methods. The easiest method for Eve to decrypt the ciphertext is to find the key which can decrypt the ciphertext. *Brute force attack* is a form of attack whereby Eve tries to evaluate all of the possible key combinations which can decrypt the ciphertext. Therefore, the size of the key itself and the size of the family of bijections (related key) become important to take into account.

We will consider an example encryption standard and related brute force attacks. The DES (Data Encryption Standard) [5, pg 49-56] algorithm was created by IBM in the early 1970s. The algorithm involves five functions: an initial permutation (IP), a function labeled f_{k1}, the function (SW) which involved permutation and substitution of the two halves of the data, the function f_{k2} again, and finally a permutation function that is the inverse of the initial permutation (IP^{-1}). The

ciphertext is created as follow:

$C = IP^{-1}(f_{k2}(SW(f_{k1}(IP(M)))))$, where M is the plaintext.

The value $k1$ and $k2$ are the sub-key created by the permutation function and the circular left shift operation on the key bits. The decryption function is the inverse of the encryption function by using the same key:

$M = IP^{-1}(f_{k1}(SW(f_{k2}(IP(C)))))$

The algorithm reportedly had no known exploitable cryptographic weaknesses, was using a 56-bit key for encryption at that time when it was announced. In 1977, Diffie and Hellman announced that the key was too short and was searchable in a reasonable time [7]. In 1994, Wiener presented a key search machine which can search the whole space of the DES key in 7 hours [9]. The key search machine cost was one million dollars at that time. In between 1997 and 1998, Electronic Frontier Foundation did actually built a highly parallel DES key search machine out of custom chips for 250K dollars. This new key search machine broke the DES II - 2 challenges in just 56 hours [4, pg 64]. In November 2001, Bound and Clayton [4, pg 64] announced the much cheaper FPGA - based DES - cracking machine that can recover all the DES keys of an IBM 4758 running IBM's own CCA cash machine software. This attack combines a smart DES key search with newly found weaknesses in the CCA software. The machine costs 1000 dollars and can break the cryptosystem in just a couple of days.

From the last example, we can see that even if the algorithm has no weakness, when the technology improves, the cryptosystem will become weaker because larger key spaces can become searchable. The ciphertext should always assume to be viewable after some time period. The brute force attack should always be considered as an upper bound of the searching time when designing the cryptosystem.

1.3 Public Key cryptography

In the security sense, it becomes harder and harder to deliver the key to another party. Usually, the encryption requires a key to encrypt the plaintext before it is transmitted. The decryption requires a key to decrypt the ciphertext into the plaintext. Depending on which algorithm we use, we can classify the

cryptosystems into two categories:

1) Symmetric Algorithm
   In this type of algorithms, both the encryption and the decryption use the same key. If any third party knows the key, they can perform the encryption and decryption like Alice or Bob.

2) Asymmetric Algorithm (Public Key Algorithm)
   In this type of algorithms, the encryption and the decryption use different but related keys. It is easy for the third party to obtain one key but not the other one even with the knowledge of the encryption or decryption function.

Caesar's Algorithm and DES use the same key for encryption and decryption. So these algorithms are classified as symmetric. On the other hand, the RSA is one of the most popular algorithms which use a pair of keys, a public key and a private key, for encryption and decryption. One of the keys is called the public key because this key is announced to the public. The other one is called the private key because only the proper owner knows the key and no other users should know it. This algorithm also indicates that with the knowledge of the public key and the algorithm, it is still not feasible for other users to find the private key. In this thesis, we will focus on asymmetric algorithms rather than symmetric algorithms.

One of the major improvements from symmetric to asymmetric algorithm cryptosystems is that asymmetric algorithms use the public key for encryption while using the private key for decryption. (Therefore, the key distribution for asymmetric cryptosystem is as important as symmetric cryptosystem.) Even if two parties have not communicated before, they still can use the asymmetric cryptosystem to achieve the needs for confidentiality. The public key cryptography was introduced by Diffie and Hellman in 1976 [8]. In the meantime, they have also announced that the algorithm cannot satisfy all the conditions they have described and can be broken by some attacks. However the algorithm is still a key concept for the later development. A couple of years later, Rivest, Shamir, and Adleman (known as RSA) introduced the RSA algorithm [10] which has all the properties which Diffie and Hellman described. The public key cryptography is asymmetric, involving the use of two separate keys.

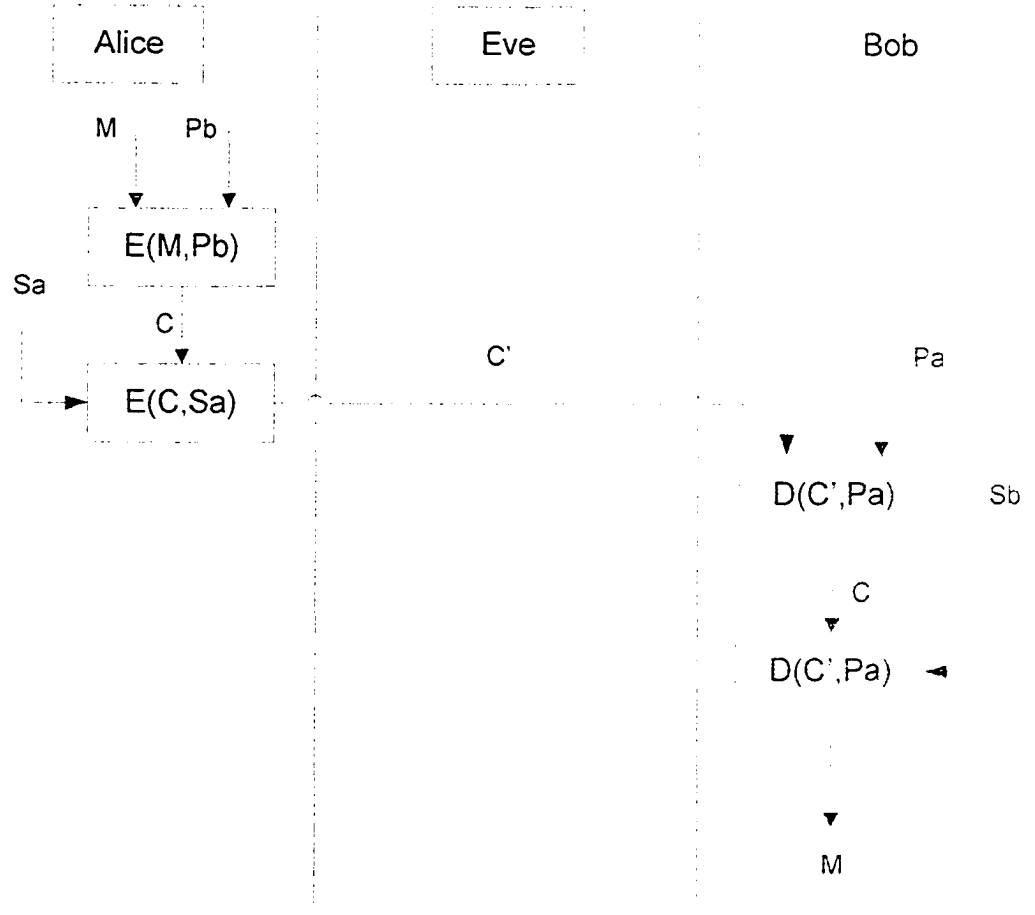
Before we discuss the details of public key cryptography, we should mention several common misconceptions concerning public key encryption. The first misconception is that public key encryption is more secure from cryptanalysis than conventional encryption. In fact, the security of any encryption scheme should depend on the length of the key and the computational work involved in breaking a cipher. There is nothing in principle about either conventional or public key encryption that makes one superior to another from the point of view of resisting cryptanalysis. The second misconception is that public key encryption is a general-purpose technique that has made conventional encryption obsolete. For some situations, because of the computational overhead of current public key encryption schemes, it seems unlikely that conventional encryption will be abandoned. Finally, the key distribution is trivial when using public key encryption compared to conventional encryption. We will demonstrate how a man-in-the-middle attack can occur when two parties communicate. The key distribution for both conventional encryption and public key encryption may involve a central agent, and the procedures involved are neither simpler nor any more efficient in either one of the two approaches.

The concept of public key cryptography evolved from an attempt to attack two of the most difficult problems associated with conventional encryption. The first problem is about key distribution. As we know, conventional encryption requires two communicants who share the same key for encryption and decryption. The second problem is about "digital signatures". If the use of cryptography was to become widespread, for instance, not only in military situations but also for commercial and private purposes, then the electronic messages and documents would need the equivalent of signatures used in paper documents.

Before we introduce any of the public key algorithms, let us look at the overall framework for the public key cryptography. The public key algorithms rely on one key for encryption and a different but related key for decryption. These algorithms have the following important characteristic: it is computationally infeasible to determine the decryption key given only the knowledge of the cryptographic algorithm and the encryption key. In addition, some algorithms also exhibit the following characteristic: either of the two related keys can be used for encryption, while the other one is used for decryption. Figure 1.2 illustrates the public key encryption process which can provide the confidentiality and digital signatures

together.

Figure 1.2



Where,

M	= A message which Alice wants to send to Bob.
Pa	= Alice's public key.
Sa	= Alice's private key.
Pb	= Bob's public key.
Sb	= Bob's private key.

$E(x, y)$ = A encryption function. Given a message x and a key y, the function can generate a ciphertext corresponding to the key.

$D(x, y)$ = A decryption function. Given a ciphertext x and a key y, the function can generate a message corresponding to the key.

1) Each end system in a network generates a pair of keys which is used for encryption and decryption of messages that it will receive. According to the above figure, Alice will have a public key Pa and a private key Sa. Bob also will have a public key Pb and a private key Sb.

2) Each system publishes its public key. Thus Alice's public key Pa and Bob's public key Pb will be placed in a public register. The private keys Sa and Sb are kept secret by Alice and Bob respectively.

3a) If Alice wants to send a message m to Bob, she encrypts the message by using Bob's public key Pb to generate the ciphertext C by the encryption function $E(M, Pb) = C$.

3b) Once the message is encrypted and becomes a ciphertext, the system encrypts the ciphertext C with Alice's private key Sa by the encryption function $E(C, Sa) = C''$.

4a) When Bob receives the message, he decrypts it with Alice's public key Pa to get C' by the decryption function $D(C'', Pa) = C'$. No other sender but Alice could have sent the message, since it can only be decrypted by using Alice's public key Pa and the corresponding private key is held in secret by Alice.

4b) Once Bob gets the ciphertext C', he uses his own private key Sb to decrypt the ciphertext C' and no other recipients can decrypt the ciphertext because only Bob possesses his own private key. Therefore, Bob performs $D(C', Sb) = M$ to recover the original message M.

In this approach, it is assumed that all participants have access to public keys. Private keys are generated locally by each participant to provide the needs for confidentialities and the digital signatures. They must never be distributed. At any time, if a system changes its private key, it needs to publish the companion public key to replace its old public key.

No matter which algorithm we use in the cryptosystem, the algorithms must fulfill the following conditions:

1) It is computationally easy for a party to generate a key pair, a public key Pk and a private key Sk.

2) It is computationally easy for a sender A, who knows the recipient's public key Pk and the message to be encrypted, M, to generate the corresponding ciphertext C:

   $E(M, Pk) = C$, where Pk is the public key and M is the message to send.

3) It is computationally easy for the receiver B to decrypt the resulting ciphertext by using the private key Sk to recover the original message:
   $D(C, Sk) = D(E(M, Pk), Sk) = M$, where Sk is the private key and M is the original message.

4) It is computationally infeasible for an opponent, who knows the public key, to determine the private key.

5) It is computationally infeasible for an opponent, who knows the public key and a ciphertext, C, to recover the original message, M.

1.4 The Diffie-Hellman (-Merkle) Algorithm

The first and the simplest public key cryptographic algorithm to understand is the Diffie-Hellman algorithm [8]. It is named after its inventors Whitfield Diffie, Martin Hellman, and Ralph Merkle, who first announced the results in 1976. The algorithm is based on the simple set of exponentiation rules:

$$(B^x)^y = B^{(x \cdot y)} = B^{(y \cdot x)} = (B^y)^x$$

Definition:

u = A prime number.

B = The generator for the algorithm.
The generator B and the prime number u must satisfy the following condition: For every number n between 1 and $u-1$ inclusive, there is a power K of B such that $n = B^K \bmod u$.

Sa = Alice's secret key

Sb = Bob's secret key

Pa = Alice's public key

Pb = Bob's public key

K = A session key

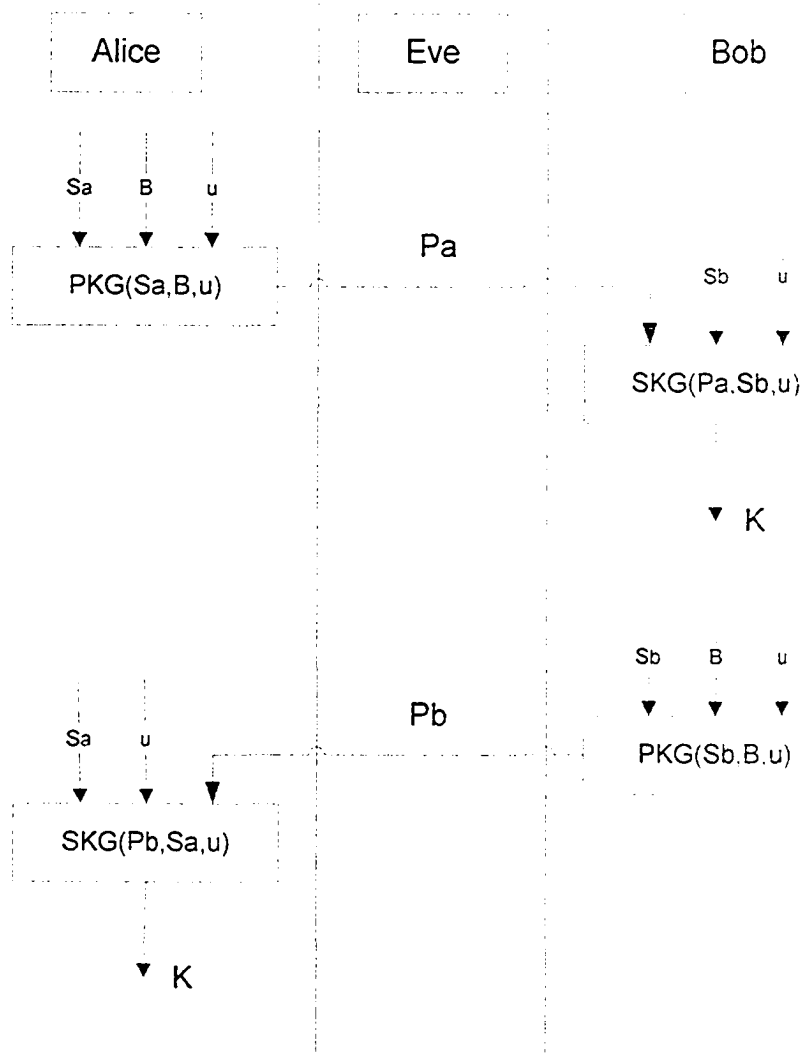
$PKG(x, B, u) = B^x \mod u = Px \mod u$

= The public key generating function. Given the randomly generated secret number x the function $PKG(x, B, u)$ will generate a public key Px.

$SKG(Px, y, u) = Px^y \mod u = B^{x*y} \mod u = K \mod u$

= The session key generating function. Given the public key Px, a secret number y, and the prime number u. The function $SKG(Px, y, u)$ will generate a session key K.

Figure 1.3



In Figure 1.3, Alice and Bob want to communicate with each other confidentially by using DHM algorithm. First, Alice and Bob need to choose a generator B and a prime number u as a modulus which Alice, Bob, and (potentially also) Eve know. The following are the initialization steps:

1) Alice selects a secret key: *Sa*

2) Alice calculates a public key: $Pa = B^{Sa} \bmod u$

3) Bob selects a secret key: *Sb*

4) Bob calculates a public key: $Pb = B^{Sb} \bmod u$

Protocol:

1) Alice and Bob exchange their public keys.

2) Alice now has Bob's public key, Pb, and performs the calculation:

$SKG(Pb, Sa, u) = Pb^{Sa} \bmod u = B^{Sb*Sa} \bmod u = K$

Even though Alice derives K, she does not know the number Sb.

3) Bob also has Alice's public key Pa, and performs the same calculation like Alice:

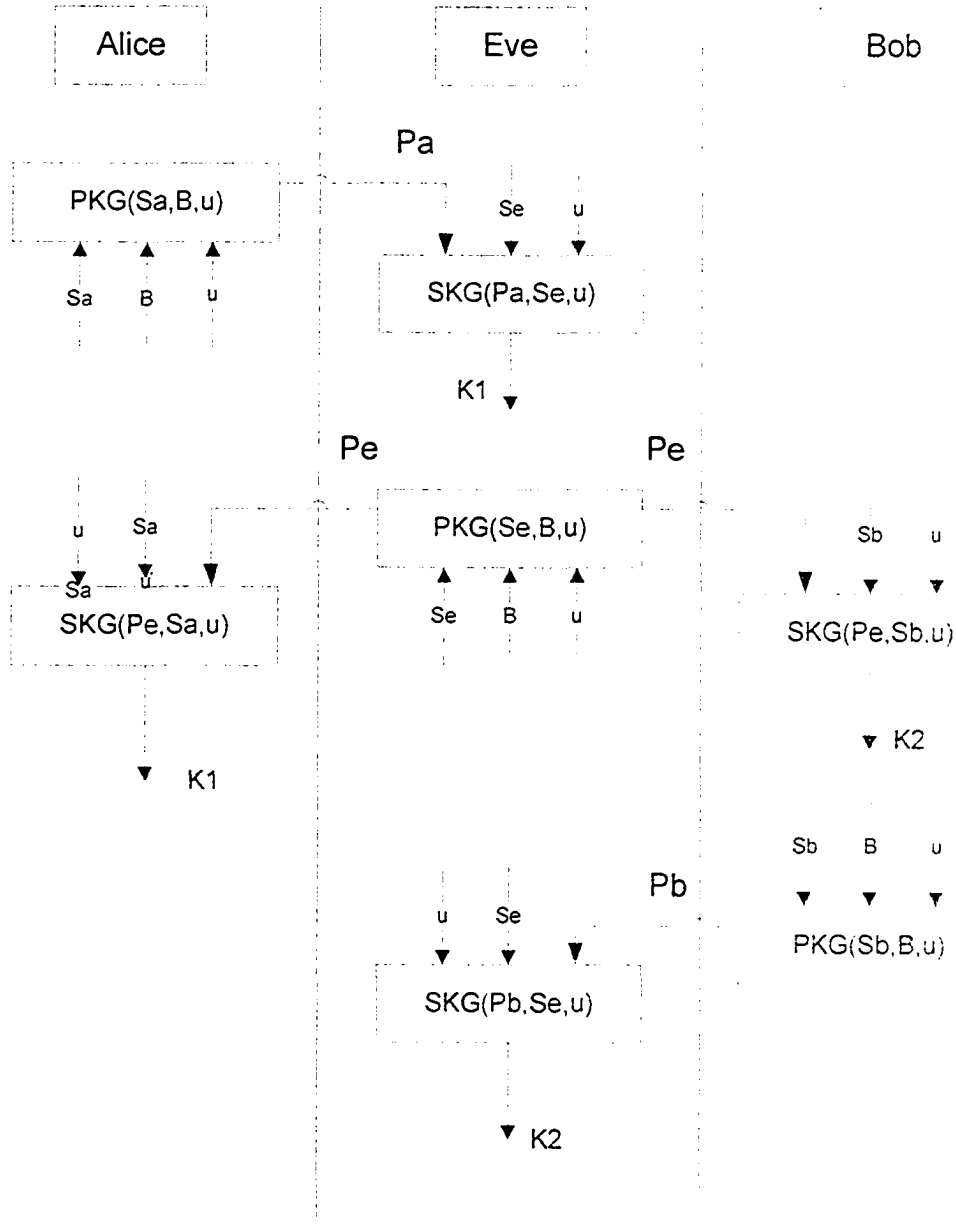
$SKG(Pa, Sb, u) = Pa^{Sb} \bmod u = B^{Sa*Sb} \bmod u = K$

Also, Bob does not know Alice's secret key Sa.

Now Alice and Bob possess the same secret K while Eve does not know anything other than the publicly disclosed public keys. Alice and Bob can use K as their session key with any conventional cryptographic algorithm. This process we just talk about is called a *key agreement protocol*. In a key agreement protocol, both sender and receiver contribute information to establish a common shared secret. Adding the modulo function into the algorithm does not change the core of the algorithm but brings some benefits. First, we can only work with integer numbers. Second, we can limit the size of the numbers to less than the modulus. Without the modulo function, computing the logarithm of an integer is trivial to solve for the modern computer and it is easy for an opponent to break. It is the computation difficulty of calculating the discrete logarithm that the Diffie-Hellman scheme capitalizes on to provide properties (4) and (5) listed earlier.

However, the DHM scheme could be broken by a man-in-the-middle attack. In Figure 1.4, there is the example in which Eve can masquerade as Bob or Alice

Figure 1.4



It is crucial for the attacker to be able to intervene in the channel, and select (or fabricate) what and which messages can be exchanged between parties. In particular, it can inject messages of its own that appear as if they originated by the communicating parties. In this sense, the attack represents a scenario of a security protocol running over an insecure network infrastructure. As it is clear from the example, the security protocol fails to provide the needed confidentiality.

The attack is successful because even though the algorithm itself is intact, the protocol used to implement the algorithm has a crucial component missing: message origin authentication. Unless we encrypt the public key by using a digital signature method, we are unable to prevent the attack.

1.5 The RSA Algorithm

The RSA algorithm [10] was invented by Ronald Rivest, Adi Shamir, and Leonard Adleman in 1977 after DHM algorithm was introduced. The core of the RSA algorithm is based on the mathematical basis that in modulo multiplication, numbers can be inverses of each other. This means that if they are multiplied, they cancel each other out to be congruent to 1:

$m * m^{-1} = 1 \bmod u$, where m^{-1} is the modulus inverse of m

In modulo arithmetic, the m^{-1} of a number is another number within the modulo multiplication tables. The modulo arithmetic exponent can be added, subtracted, and multiplied as in normal arithmetic, even though the modulo arithmetic exponent has a different meaning than in normal arithmetic.

Before we look at the RSA algorithm, let us define some notation:

S = the secret Private Key.

P = the Public Key.

Pick P and S such that $(P * S) = 1 \bmod n$, for some number n

So,

$S = P^{-1} \bmod n$ and $P = S^{-1} \bmod n$

Because

$P * S \bmod n = P * (P^{-1}) \bmod n = 1 \bmod n$

And

$P * S \bmod n = S * (S^{-1}) \bmod n = 1 \bmod n$

We can write $msg^{P} \bmod n = C$ where msg is the plaintext and C is the ciphertext

We can do the following to recover the plaintext from C:

$C^{S} \bmod n = (msg^{P})^{S} \bmod n = msg^{(P*S)} \bmod n = msg$

Before we discuss how the RSA algorithm works, let us define some of the terms:
Definition:

U = A prime number.

V = A prime number with a same length of U.

N = A product of U and V: $N = U * V$.

M = A message. If the length of M greater than N then divide it into sub-messages.

msg = A sub-message length less then N

Given that N is a produce of two large prime numbers U and V, some condition must be satisfied:

1) $msg < N$. (If not, then the algorithm will not work)
2) If msg is 0 or 1, then is not secure to prevent the attack.
   ($0^K = 0$ and $1^K = 1$ for any $K < N$)
3) For every number n between 1 and $P-1$ inclusive, there is a power k of msg such that $n = msg^k \bmod P$.

Pb = Bob's public key.

Sb = Bob's private key.

$Alp(N)$ $= ((U-1)*(V-1))$.

C = A ciphertext.

$E(msg, x, n) = msg^x \bmod n = C \bmod n$

= The encryption function. Given a message msg, the public key x, and the modulo number n. The function will generate the ciphertext C corresponding to the key.

$D(C, y, n) = C^y \bmod n = msg^{x*y} \bmod n = msg \bmod n = msg$

= The decryption function. Given a ciphertext C, the private key y, and the modulo number n. The function will generate the message msg corresponding to the key.

Let see how Alice uses the RSA algorithm to communicate with Bob. Bob needs to follow some initialization steps:

1) Selects two prime numbers U and V. U and V should be the same length.

2) Computes $N = U * V$.

3) Selects Pb as a public key such that Pb has no division in common with $(U-1)*(V-1)$.

4) Computes Sb as a private key such that Sb and Pb are multiplicative inverses mod $(U-1)*(V-1)$:

$Sb * Pb = 1 \bmod((U-1)*(V-1))$

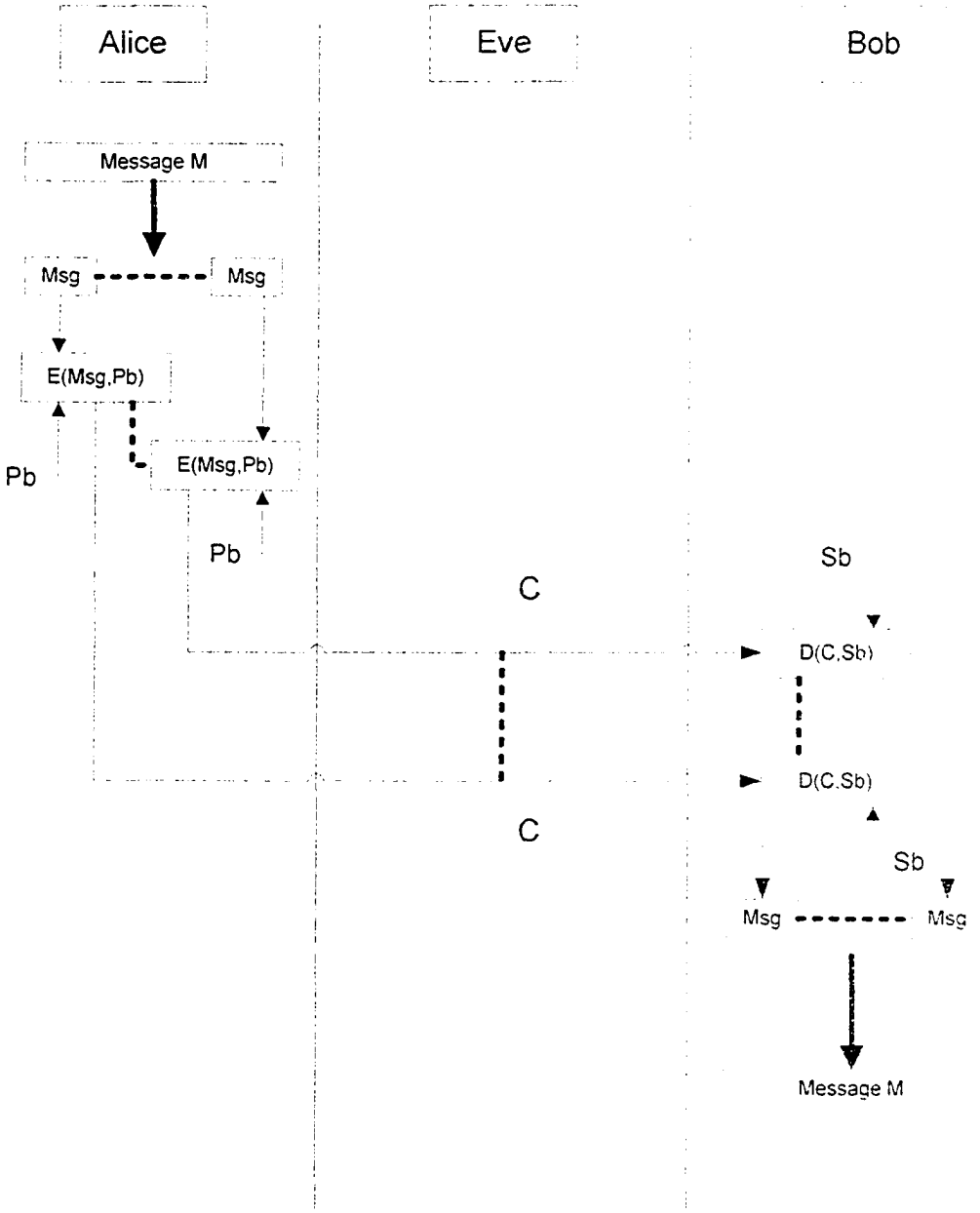
And from now on, define $Alp(N) = ((U-1)*(V-1))$

So the last formula can be rewritten as:

$$Sb * Pb = 1 \bmod Alp(N)$$
$$\Rightarrow Sb = Pb^{-1} \bmod Alp(N)$$
$$\Rightarrow Pb = Sb^{-1} \bmod Alp(N)$$

After that, Bob publishes his public key Pb through a trusted arbiter with the number N. Now let us see how Alice sends a confidential message msg by using RSA algorithm to Bob. Shown in figure 1.5

Figure 1.5



1) Alice computes the ciphertext C from the msg:

$$E(msg, Pb) = C = msg^{Pb} \bmod N$$

2) Alice sends the ciphertext C to Bob

3) Bob receives the ciphertext C and decrypts it to obtain the plaintext message msg:

$$D(C, Sb) = C^{Sb} \bmod N = (msg^{Pb})^{Sb} \bmod N = msg^{(Pb*Sb)} \bmod N = msg$$

The initialization steps 1 and 2 create the TRAPDOOR function that makes the RSA algorithm useful. A trapdoor function is the function that is easy to compute but it is difficult to compute its inverse. Since Bob knows U and V, it is easy for him to compute Pb and Sb. For anyone who does not know U and V, this is almost infeasible to compute Sb given N and Pb. This is the property that makes RSA satisfy the requirements of public key cryptography.

The RSA algorithm does not only provide confidentiality but also authentication using digital signatures. We will discuss how the RSA algorithm can provide digital signature in chapter 5 in more detail.

In most of the time, the RSA algorithm is used to transmit an encrypted session key only because the algorithm requires a lot of computational time and resource for encryption and decryption. The key, once received, is used for further encryption using any kind of shared-key algorithm.

1.6 The Elliptic Curve Algorithm

The elliptic curve was introduced into cryptography by Miller and Koblitz in the mid 1980. Later, Lenstra showed how to use elliptic curves to factor integers. Since that time, elliptic curves have played an important role in many cryptographic situations. One of the advantages is that the algorithm seems to offer a level of security comparable to classical cryptosystems that use much larger key sizes. Using much shorter numbers as a key can represent a considerable savings in hardware implementations. The following provide the details of the algorithm [5, pg 193] [6, pg 272].

To understand the elliptic curve, we have to start from the *addition law*. An elliptic curve E is the graph of an equation:

$$E: y^2 = x^3 + a*x + b$$

where a,b can be rational numbers, complex numbers, or integers mod n as long as it is the appropriate set. Let us work on the real numbers in our example. If a,b are real numbers, the graph E has one of two possible forms, depending on whether the cubic polynomial in x has one real root or three real roots.

Figure 1.6 Example for $y^2 = x(x+2)(x-2)$

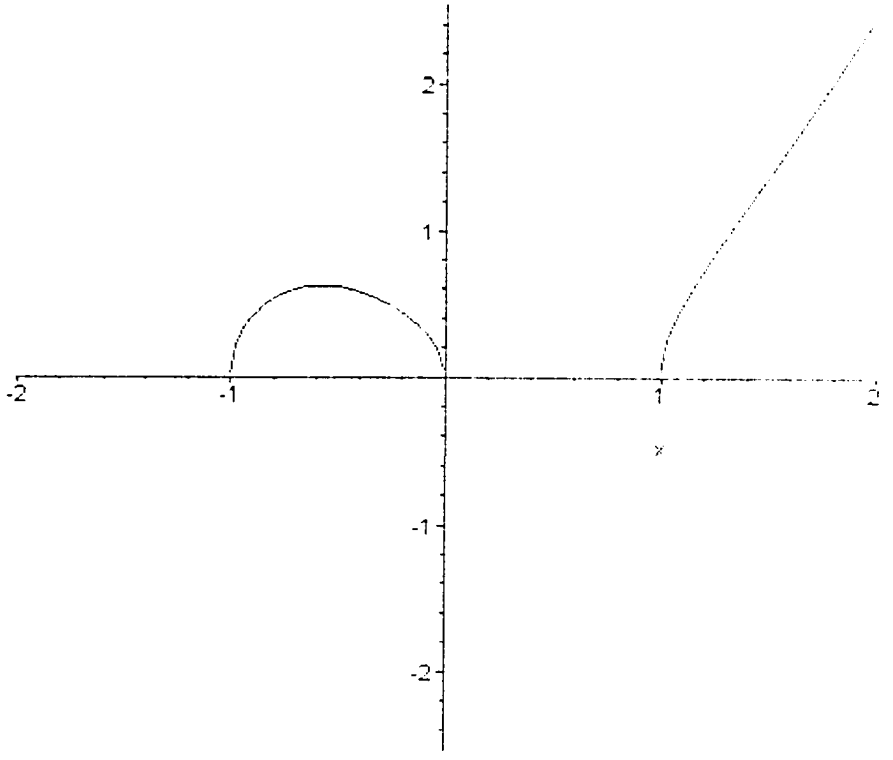
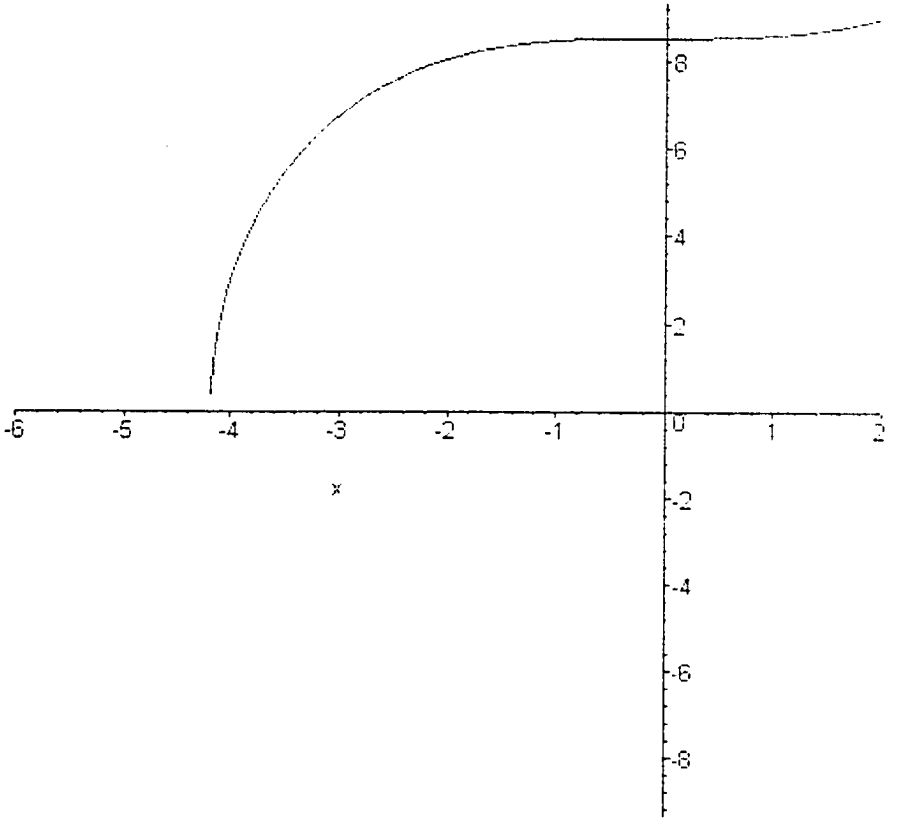
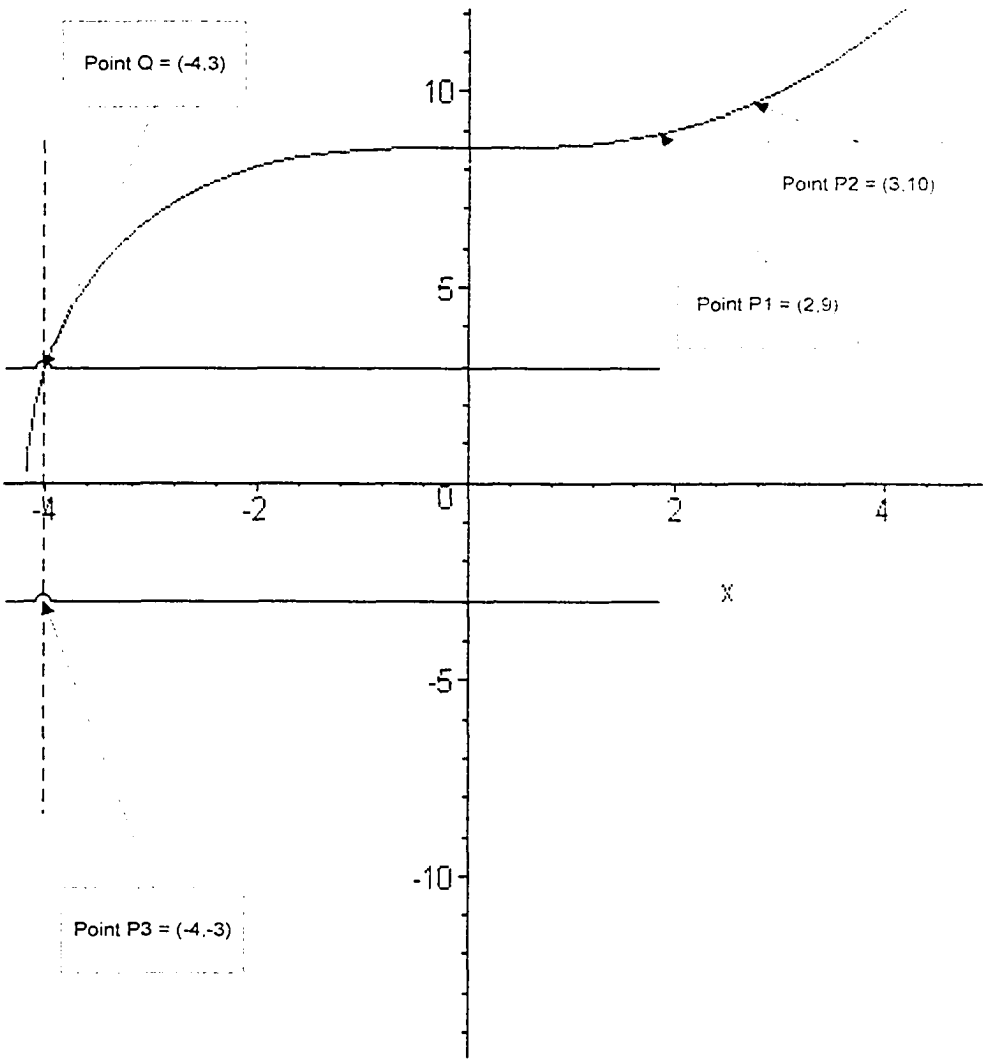


Figure 1.7 Example for $y^2 = x^3 + 17$



We exclude the case that the cubic polynomial $x^3 + a*x + b$ has multiple roots. For example, the graph $y^2 = x^2(x+1)$ will be excluded. Given any two points *P1* and *P2* on *E*, we can obtain the third point *P3* on E as the following:

Figure 1.8



Draw the line *L* through *P1* and *P2* (if *P1* = *P2*, take the tangent line to E at *P1*). The line *L* intersects *E* in a third point *Q*. Reflect Q through the x-axis to get *P3*. This is defined as a law of addition on E by

$P1 + P2 = P3$,where $+$ stands for the above operation, not ordinary addition.

Let us start with the example [6, pg 274]. Suppose E is defined by $y^2 = x^3 + 73$. Let *P1* = (2, 9) and *P2* = (3,10). The line *L* through *P1* and *P2*

with a slope $\frac{Dy}{Dx}=1$ is $y=x+7$. Substituting into the equation for E yields $(x+7)^2=x^3+73$, simplifies to $x^3-x^2-14x+24=0$. Since L intersects E through $P1$ and $P2$, we know the sum of the three roots minus the coefficient of x^2 and therefore equals 1. We already know two roots, $x=2$ and $x=3$. So the third root will be $1-(2+3)=-4$. Since $y=x+7$, therefore substitute $x=-4$ to get $y=(-4)+7=3$, and Q = (-4,3). Therefore, $P3$ = (-4,-3). Now suppose we want to add $P3$ to itself ($P3$ + $P3$). The slope of the tangent line to E at $P3$ is obtained by implicit differentiation of the equation for E:

$$2ydy=3x^2dx. \quad \text{So} \quad \frac{dy}{dx}=(3x^2)(2y)^{-1}$$

Substitute (-4,-3) from $P3$ to get the slope, -8. In this case, line L is

$$y=-8(x+4)-3$$

Substitute into the equation for E to yield $(-8(x+4)-3)^2=x^3+73$. Simplify the equation E to get $x^3-64x^2-560x-1152=0$. The sum of the three roots is 64. Therefore the third root is 64 - (-4) - (-4) = 72. The corresponding value of y is -611. Changing y to $-y$ to get:

$P3$ + $P3$ = (72,-611).

In general, the addition Law is defined as the following:

Let E be given by $y^2=x^3+ax+b$ and let $P1=(x_1,y_1)$ and $P2=(x_2,y_2)$
Then consider the operation $P1$ + $P2=P3=(x_3,y_3)$.

Where

$$x_3=m^2-x_1-x_2$$
$$y_3=m(x_1-x_3)-y_1$$

And

$$m=\frac{(y_2-y_1)}{(x_2-x_1)} \qquad \text{if } P1\neq P2$$
$$m=\frac{(3*(x_1)^2+a)}{(2*(y_1))} \quad \text{if } P1=P2$$

The above operations can be calculated under modulo arithmetic as well. Let's consider the following example:

Given E: $y^2=x^3+2x+3 \pmod 5$

Find (1,4) + (3,1) on the curve E. First we have to find the slope m

$$m = \frac{(1-4)}{(3-1)} = 1(\text{mod}\,5)$$

Therefore

$$x3 = m^2 - x1 - x2 = 1 - 1 - 3 = 2(\text{mod}\,5)$$
$$y3 = m(x1 - x3) - y1 = 1(1 - 20 - 4) = 0(\text{mod}\,5)$$

So the point (1,4)+(3,1) = (2,0) which is on the curve and we can verify it.

$$x = 2 \Rightarrow y^2 = 15 = 0(\text{mod}\,5)$$

Let us define some terminology before we discuss the algorithm.

Definition:

$*$ = stands for repetitive application of the $+$ operator, not ordinary multiplication.

P = The prime number.

m = The original message.

E = The elliptic curve we use for the algorithm. The curve represent by $f(x) = x^3 + ax + b \mod P$. Most of the time we generate the curve by imbedding method using the message m:
Suppose we have a function E and an integer message m such that $0 < m < \frac{P}{(1000-1)}$. (Check that m must be less then $\frac{P}{(1000-1)}$ because we want to append at least three digits. For a larger number n of digits, the upper bound will be $m < \frac{P}{10^n - 1}$.) Appending three digits to m will produce a value Pm such that $1000m < Pm < 1000(m+1) < P$. We append different digits until we find an Pm such that $f(Pm) = (Pm)^3 + a(Pm) + b$.

G = The base point. The point is on the curve E.

Pm = The point represents the message m.

Sb = Bob's private key.

Pb = Bob's public key. The public key is generated by $Sb * G$.

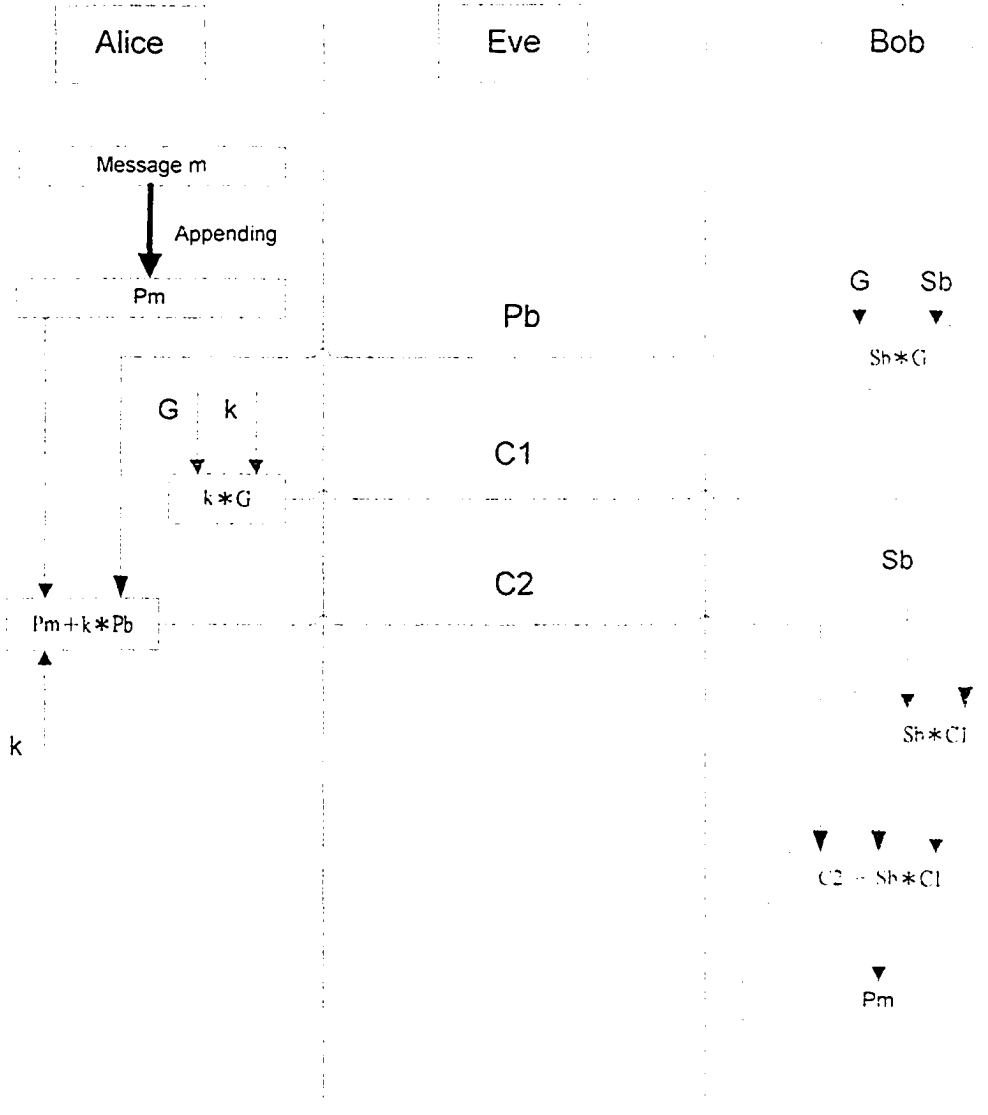
k = A random number.

$C1$ = Part of the ciphertext. $C1$ is calculated by $k * G$.

$C2$ = Part of the ciphertext. $C2$ is calculated by $Pm + k * Pb$.

Now let us see how Alice uses the Elliptic Curve to encrypt a message that she can send. (Shown figure 1.9)

Figure 1.9



In the initialization steps:

1) Pick the point $G = (x_1, y_1)$ as a base point on E, and the curve E:

$f(x) = x^3 + ax + b \mod P$ where P is a prime number. The point G and equation E are published; therefore Eve knows them as well.

2) Bob has chosen a random number Sb as a private key and has published the point $Sb * G = Pb$ as a public key.

Protocol:

1) Alice has a message and uses a point $Pm = (x, y)$ to represent the message.

2) Alice downloads the Pb and chooses a random number k. She sends Bob $k * G = C1$ and $Pm + k * Pb = C2$

3) Once Bob receives it, he first calculates $Sb * C1$ and then subtracts this from $C2$ to get Pm

$$C2 - Sb * k * G = Pm + k * Sb * G - Sb * k * G = Pm$$

Because of the property of the addition law, we can use elliptic curve and DHM to exchange the symmetric key thus using the public key algorithm only occasionally to establish shared secret keys. Let us see how Alice and Bob can exchange the session key confidentially [6, pg 288]: (Shown figure 1.2.11)

Definition:

Sa = Alice's private key.

Pa = Alice's public key. The public key is calculated by $Sa * G$

Sb = Bob's private key.

Pb = Bob's public key. The public key is calculated by $Sb * G$

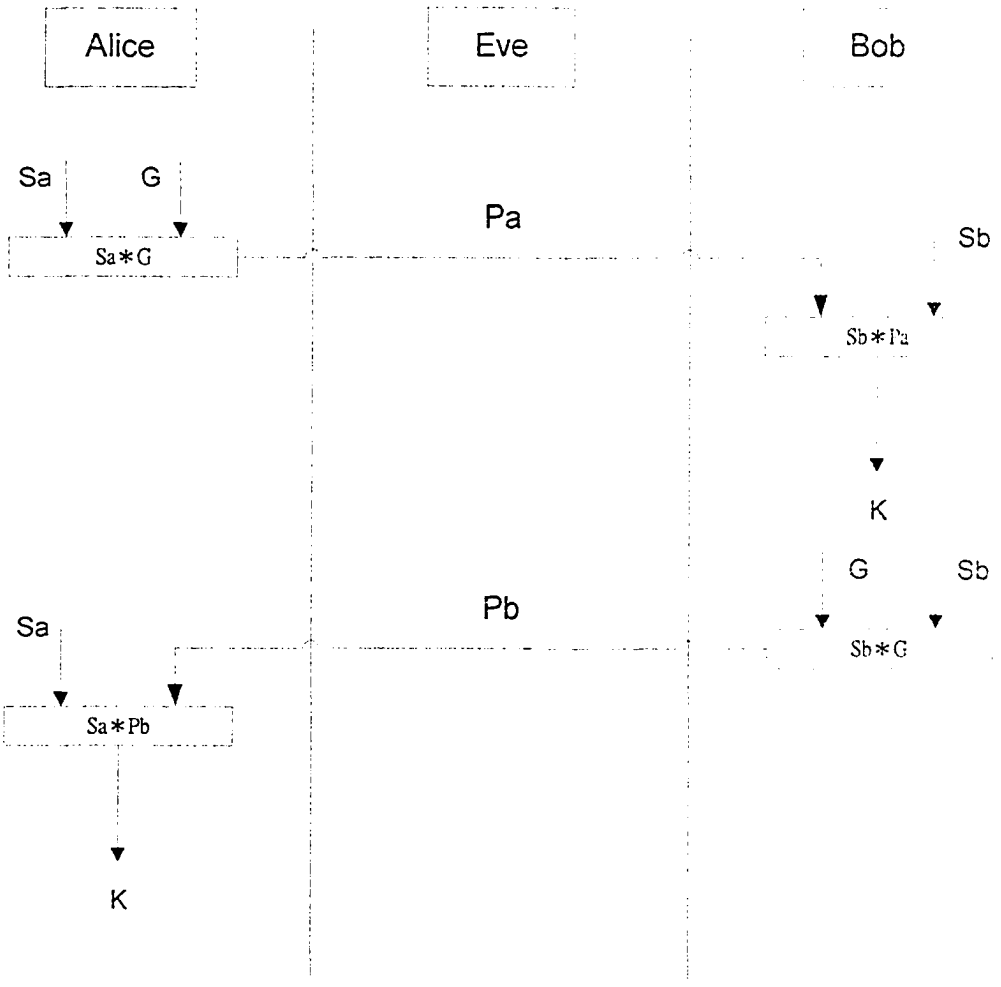
Alice chooses Sa and Bob chooses Sb. The Sa and Sb must be satisfies the following:

For every number n between 1 and $P-1$ inclusive, there is a coefficient Sa and Sb such that $n = Sa * Sb * G \mod P$.

If $Sa * Sb * G > P$,then Alice and Bob may need to chooses again.

K = The session key.

Figure 1.10



Initialization steps:

1) Pick a public point G and elliptic curve E as before with some prime numbers P.

2) Alice chooses Sa as a private key, and Bob also chooses Sb as a private key. They need to keep them private to themselves but publish $Sa * G = Pa$ and $Sb * G = Pb$.

Protocol:

1) Alice now takes Pb and multiplies it by Sa to get the key:
$Sa * Pb = Sa * (Sb * G) = K$

2) Bob now takes Pa and multiplies it by Sb to get the key:
$Sb * Pa = Sb * (Sa * G) = K$

After that, they both have a same session key.

The advantage of this algorithm is that for any given point P on the curve it is easy to find $C * P = M$ where C is a coefficient, but it is difficult to find C given M, P, and the equation of the curve since the addition in this section is not the ordinary addition, the multiplication is different too. Therefore, if Eve wants to retrieve C, then she needs to try all the possible given M, P, and the equation of the curve.

1.7 Hybrid System

Public key algorithms are built in a completely different way from symmetric key cryptosystems. The public key algorithm usually needs more computation operations than a symmetric key algorithm for encryption and decryption of comparable strength. For example, the RSA algorithm involves the exponential operation to encrypt and decrypt the message and the ciphertext. Using the same key size, the DES will encrypt the message faster than the RSA because the RSA needs to operate an exponential function rather than a linear function. Therefore, most of the public key algorithms are used only for key distribution, which transfer a key which is subsequently used for symmetric cryptography. When Alice communicates with Bob, she uses an asymmetric system to transmit the symmetric key to Bob. After that, Alice and Bob can communicate with each other by using the same symmetric key. We know that symmetric systems can be attacked by brute force attack. An improvement is a hybrid system [4, pg 67], where Alice generates a random session key, encrypts it by using Bob's public key and then sends it to Bob. In this case, they can share the same session key for later communication. (See Figure 1.11.)

Definition:

K = A session key.
Pb = Bob's public key.
Sb = Bob's private key.
$C1$ = A ciphertext generated by public key algorithm encryption.

$C2$ = A ciphertext generated by symmetric algorithm encryption.

M = The message which Alice wants to send to Bob.

$PE(K, x) = C1$

= Any public key algorithm's encryption function. The function will take K as an input message and x as a key to generate a ciphertext $C1$.

$PD(C1, y) = K$

= The public key algorithm's decryption function. The function will take a ciphertext $C1$ as an input and decrypt the ciphertext by using the key y to get the message K.

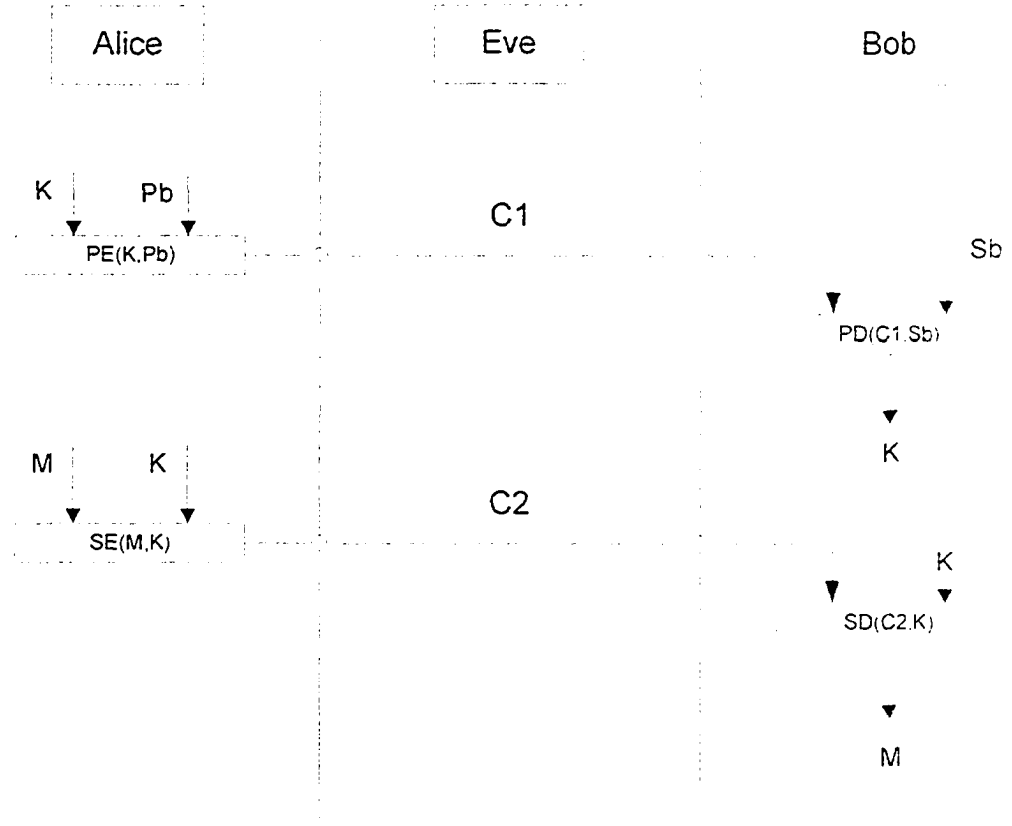
$SE(M, K) = C2$

= Any symmetric key algorithm's encryption function. The function will take a message M as an input and encrypt the message by using the session key K to generate the ciphertext $C2$.

$SD(C2, K) = M$

= The symmetric key algorithm's decryption function. The function will take the ciphertext $C2$ as an input and decrypt the ciphertext by the same session key K to get M.

Figure 1.11



The benefit of the combination for both public key system and symmetric key system is that, Alice and Bob can deliver the session key to each other to decrease the computational time for encryption and decryption without Eve knowing the session key. Also there is another benefit, each time when Alice generates the session key and encrypts the session key; the ciphertext she sends to Bob will be different. This makes Eve harder to obtain the session key from each ciphertext. Even if Eve gets the right session key, the key may only be used once and after that Alice can generate another session key, and so Eve needs to find the session key again. The randomness involved in generating the session key ensures that a third party cannot guess it except by brute force attack, and if it is found it only compromises one session.

Chapter 2: Integrity

Integrity is the property that no other unauthorized parties are able to modify the data when it is transmitted. As we have seen, protecting the confidentiality of a message in transmission on a communication channel between Alice and Bob ensures that no other unauthorized parties can discover the contents of the message. Protecting the integrity of the message under the same circumstances means to ensure that once the message leaves Alice, no other unauthorized parties can alter the contents of the message until it reaches Bob. In practice, it is impossible to prevent an attacker (our old friend Eve) who has control of the channel from altering the message, so what we actually mean is ensuring that no other unauthorized parties can alter the message without Bob noticing it.

2.1 Independence from confidentiality

Most of the people think that confidentiality implies integrity. This means that a ciphertext which has been changed ever so slightly will automatically decrypt into garbage, so that the recipient will notice that integrity was compromised. Actually this is not always true. For example, suppose the ciphertext is obtained by the bit-by-bit exclusive-or of the plaintext and a pseudo-random key stream. If Eve knows the exact format of the message, she will know which bit positions correspond to a specific field. She can create an altered ciphertext and substitute the original ciphertext and resend it into the channel. In this case, Alice and Bob would not detect any error after the transmission. This is an attack called an *attack in depth*. There is another similar attack for public key cryptosystem and we will discuss more in section 1.4 (*Man-in-the-middle attack*). Therefore, in general, neither integrity nor confidentiality implies the other.

2.2 Hash

An error - detecting code with an external interface but suitable for integrity protection is called a *cryptographic hash function*, or sometimes called *one-way hash* to convey the intuitive meaning which is easy to compute the function in the forward direction, but practically impossible to compute its inverse [4, pg 70]. Its

fundamental property is non-invertibility: given any hash output "Sout", it is computationally infeasible to find an input "Sin" such that

$$h(Sin) = Sout$$

The idea here is to produce a representative "fingerprint" of any input message. This way, if the hash output is secure from modifications, it will not be possible for any attacker to modify the message in a way that still matches the hash. Stronger requirement for a hash function is collision resistance: it is computationally infeasible to find two inputs with the same image. If one input could do that, then it would no longer be possible to consider hashes as representative fingerprints of longer strings.

Usually the Hash function is publicized. Not only the sender or receiver but also the attacker will know the hash functions. Therefore if the attacker is able to modify the hash, then it will be trivial for the attacker to calculate a new hash that matches the modified message. There are two other cryptographic primitives, the MAC and the digital signature, capable of withstanding that type of attack.

2.3 MD5 Message Digest Algorithm

The MD5 message – digest algorithm [5, pg 272] [24] was developed by Ron Rivest at MIT. Until the last few years, the MD5 was the most widely used secure hash algorithm. The algorithm takes a message of arbitrary length as an input and produces a 128-bit message digest (checksum) as an output. The input is processed in 512-bit blocks.

When Alice wants to send a message to Bob and produces the message digest by the MD5 hash function, she has to do the followings in order to create the MD5 message digest:

1) Append padding bits
   The message is padded so its length in bits is congruent to 448 modulo 512. The message is extended so that it is just 64 bits shy of being a multiple of 512 bits long. Padding is always added, even if the message is already with the

desired length. For example. if the message is 448 bits long. it is padded by 512 bits to a length of 960 bits.

2) Append length

A 64 bits representation of the length in bits of the original message is appended to the result of step 1. If the original length is greater than 2^{64}, then only the low order 64 bits of the length are used. Therefore. the field contains the length of the original message modulo 2^{64}. At this point the resulting message has a length that is an exact multiple of 512 bits.

3) Divide appended result

The outcome of the first two steps yields a message that is an integer multiple of 512 bits in length. We can divide the appended result into L pieces. Therefore. the appended result is $L*512$ bits length long. We call each piece $Y(q)$ where $0 \le q \le L$.

4) Initialize MD buffer

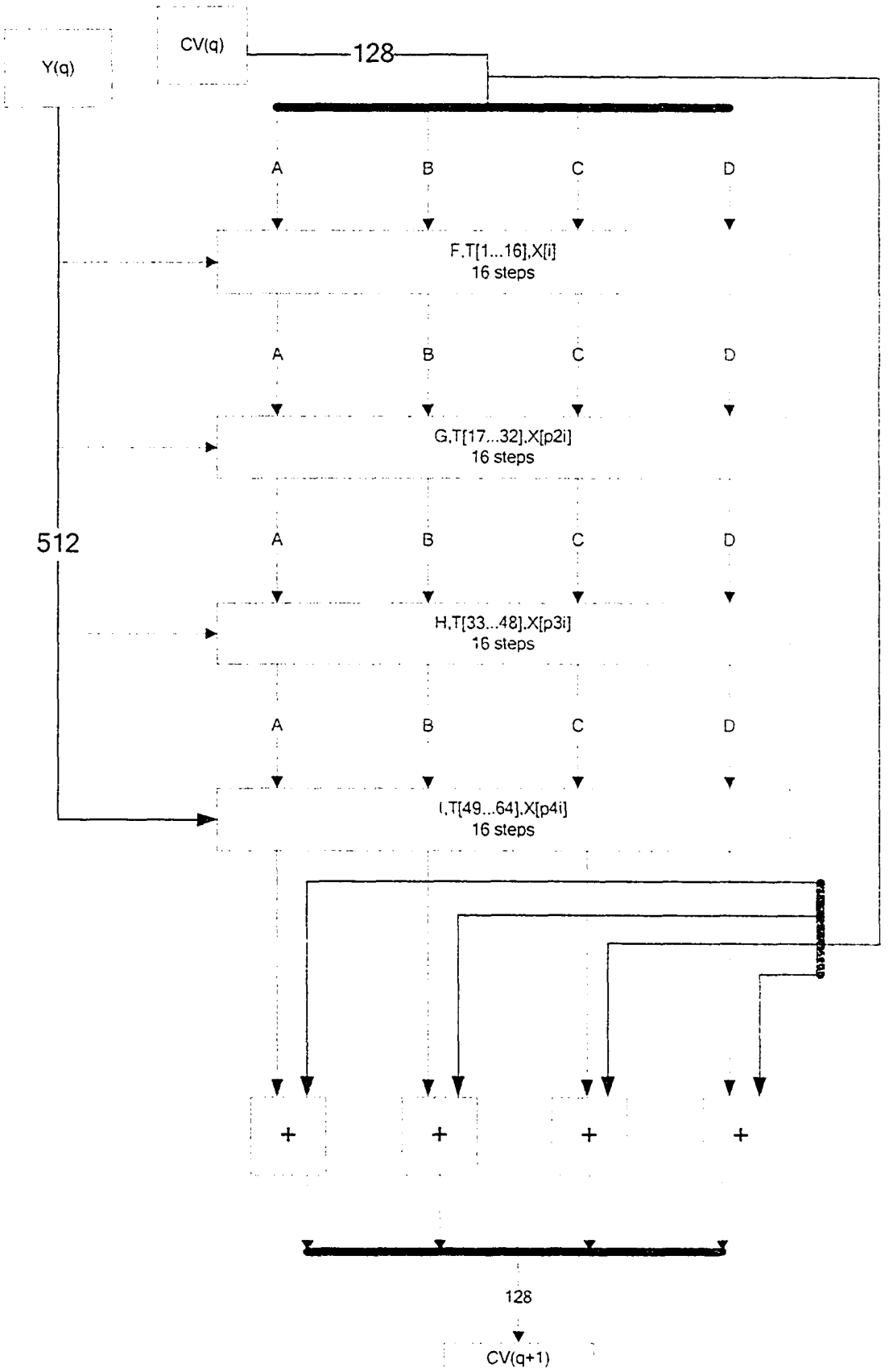
A 128-bit buffer is used to hold intermediate and final results of the hash function. The buffer can be represented as four 32-bit registers (named A. B. C. and D). These registers are initialized to the following 32-bit integers in hexadecimal. bit order least-significant-first:

A = 01 23 45 67
B = 89 AB CD EF
C = FE DC BA 98
D = 76 54 32 10

5) Process message in 512-bit blocks

The main idea of this algorithm is a compression function that consists of four "rounds" of processing. This module is labeled hash(MD5) in the figure 2.3. The logic of this function is illustrated in figure 2.1

Figure 2.1



The four rounds have a similar structure, but each uses a different primitive logical function, referred as F, G, H, and I in the specification. Each round takes the current 512-bit block $Y(q)$ being processed and the 128-bit MD buffer $(A \quad B \quad C \quad D)$ as an input and updates the contents of the buffer. Also, each round will make use of one-fourth of a 64-element table T[1..64] which is constructed from the sin function. The element T[i] has the value equal to the integer part of $2^{32} \times abs(\sin(i))$ where i is in radians. The output of the fourth round is added to the input to the first round $CV(q)$ to produce $CV(q+1)$. The addition is done independently for each of the four words in the buffer with each of the corresponding words in $CV(q)$, using addition modulo 2^{32}.

6) Output

After all L 512-bit blocks have been processed, the output from the L^{th} stage is the 128-bit message digest. We can summarize the process of MD5 into the following:

$$CV(0) = (A \quad B \quad C \quad D)$$
$$CV(q+1) = Sum(CV(q), RF(I)[Y(q), RF(H)[Y(q), RF(G)[Y(q), RF(F)[Y(q), CV(q)]]]])$$
$$MD = CV(L)$$

where $RF(x)$ is a round function using primitive logical function x and MD is the message digest. The function $Sum(\ )$ is an addition function modulo 2^{32} which is performed separately on each word of the pair of inputs.

Let us look at more details at the logic in each of the four rounds of the processing of one 512-bit block. Each round consists of a sequence of 16 steps operating on the buffer $(A \quad B \quad C \quad D)$. Each step performs the following formulas:

$$A' \leftarrow D$$
$$C' \leftarrow B$$
$$D' \leftarrow C$$
$$B' \leftarrow B + ((A + g(B, C, D) + X[k] + T[i]) <<< s)$$

Where

$A \quad B \quad C \quad D$	= the four words of the buffer.
$A' \quad B' \quad C'' \quad D'$	= the four words of the buffer after processing the primitive function.
$g(B, C, D)$	= one of the primitive functions F, G, H, I
$<<< \ s$	= circular left shift (rotation) of the 32-bit argument by s bits
$X[k]$	= the k^{th} 32-bit word in the q^{th} 512-bit block of the message
$T[i]$	= the i^{th} 32-bit word in sin matrix T
$+$	= addition modulo 2^{32}

Each of the primitive functions F, G, H, I has the following logic structure:

$$F(B,C,D) = (B \wedge C) \vee (\overline{B} \wedge D)$$
$$G(B,C,D) = (B \wedge D) \vee (C \wedge \overline{D})$$
$$H(B,C,D) = B \oplus C \oplus D$$
$$I(B,C,D) = C \oplus (B \vee \overline{D})$$

Where $\wedge, \vee, \oplus, \alpha$ are bit-wise operation.

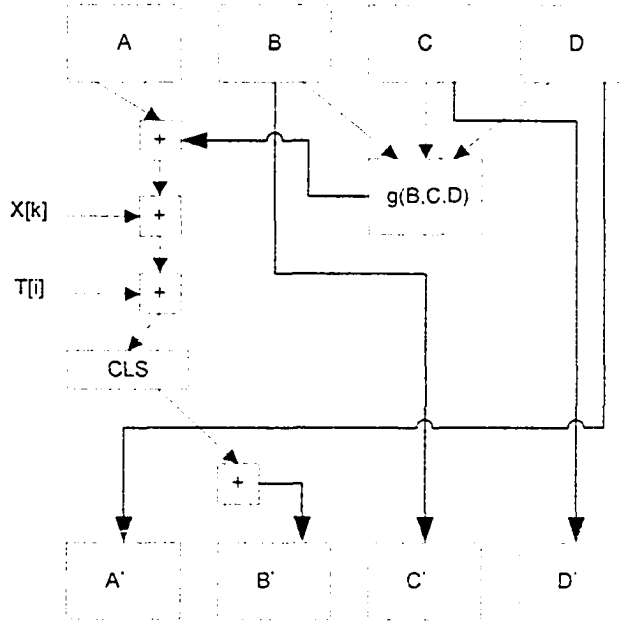
$\wedge$ = "And" operation

$\vee$ = "Or" operation

$\oplus$ = "Excl Or" operation

α = "Complement" operation, where α can be B or D

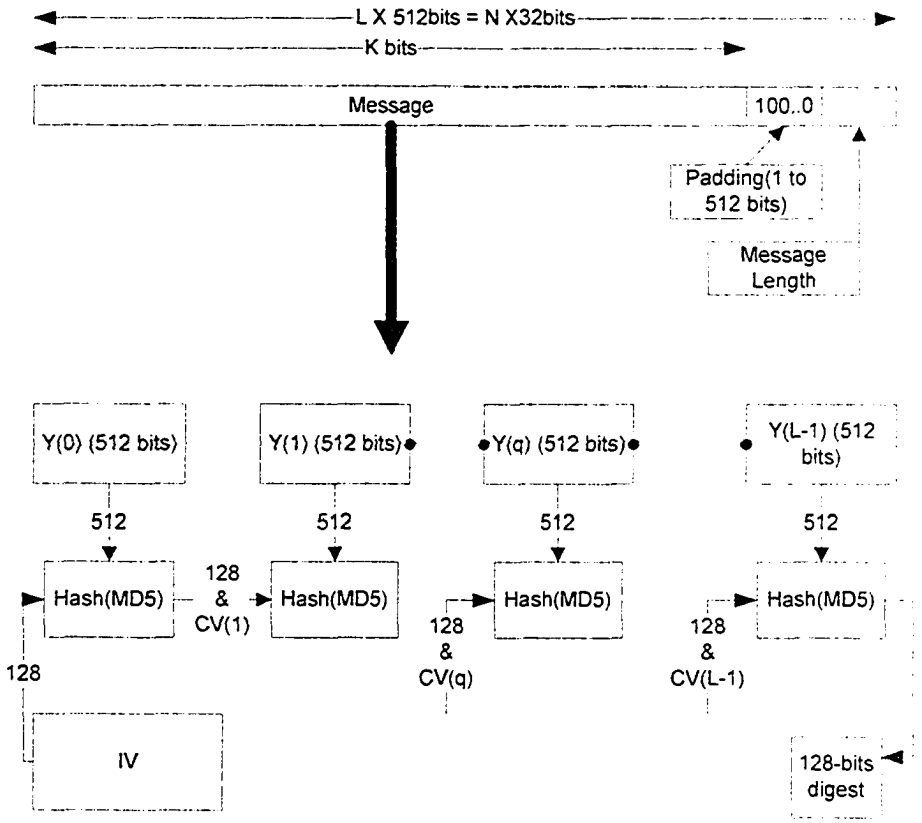
Figure 2.2



The MD5 $RF(x)$ is shown in figure 2.2. It takes the primitive function F in the first round and performs the formula. Once the result is calculated, update the value B with the result, update C with old B value, update D with old C value and update A with the old D value. After the first round, we repeat the same thing except we will use the primitive function G and update the value A, B, C, D again. After we finish the fourth round by using the I primitive function, we will have the final result of A, B, C, D. We can perform the addition function $Sum(x, y)$ to get the next $CV(q+1)$ from $CV(q)$.

Figure 2.3 shows the overall process of a message to produce a message digest.

Figure 2.3



The MD5 algorithm has the property that every bit of the hash code is a function of every bit in the input. The complex repetition of the basic functions F, G, H, I produces results that are well mixed. Therefore, it is unlikely that two messages chosen at random will have the same hash code. Rivest conjectures that MD5 is as strong as possible for a 128-bit hash code, which mean this is difficult of coming up with two messages having the same message digest is on the order of 2^{64} operations. Also, it is difficult to find a message with a given digest is on the order of 2^{128} operations.

2.4 Secure Hash Algorithm

The secure hash algorithm (SHA) [5, pg 281] [15] was developed by the National Institute of Standards and Technology (NIST) in 1993. A revised version was issued in 1995 and is generally referred as SHA-1. The algorithm takes a message with a maximum length of less than 2^{64} bits as an input and produces a 160-bit message digest as an output. The input is processed in 512-bit blocks like MD5.

Same as before, if Alice wants to send a message to Bob with a message digest then she needs to proceed the following steps:

1) Append padding bits

The message is padded so that its length is congruent to 448 modulo 512 like MD5.

2) Append length

A block of 64 bits is appended to the message. Again, this process is same as in MD5.

3) Initialize MD buffer

A 160-bit buffer is used to hold intermediate and final results of the hash function. The buffer can be represented as five 32-bit register (named A, B, C, D, and E). These registers are initialized to the following 32-bit integers in hexadecimal, bit order most-significant-first:

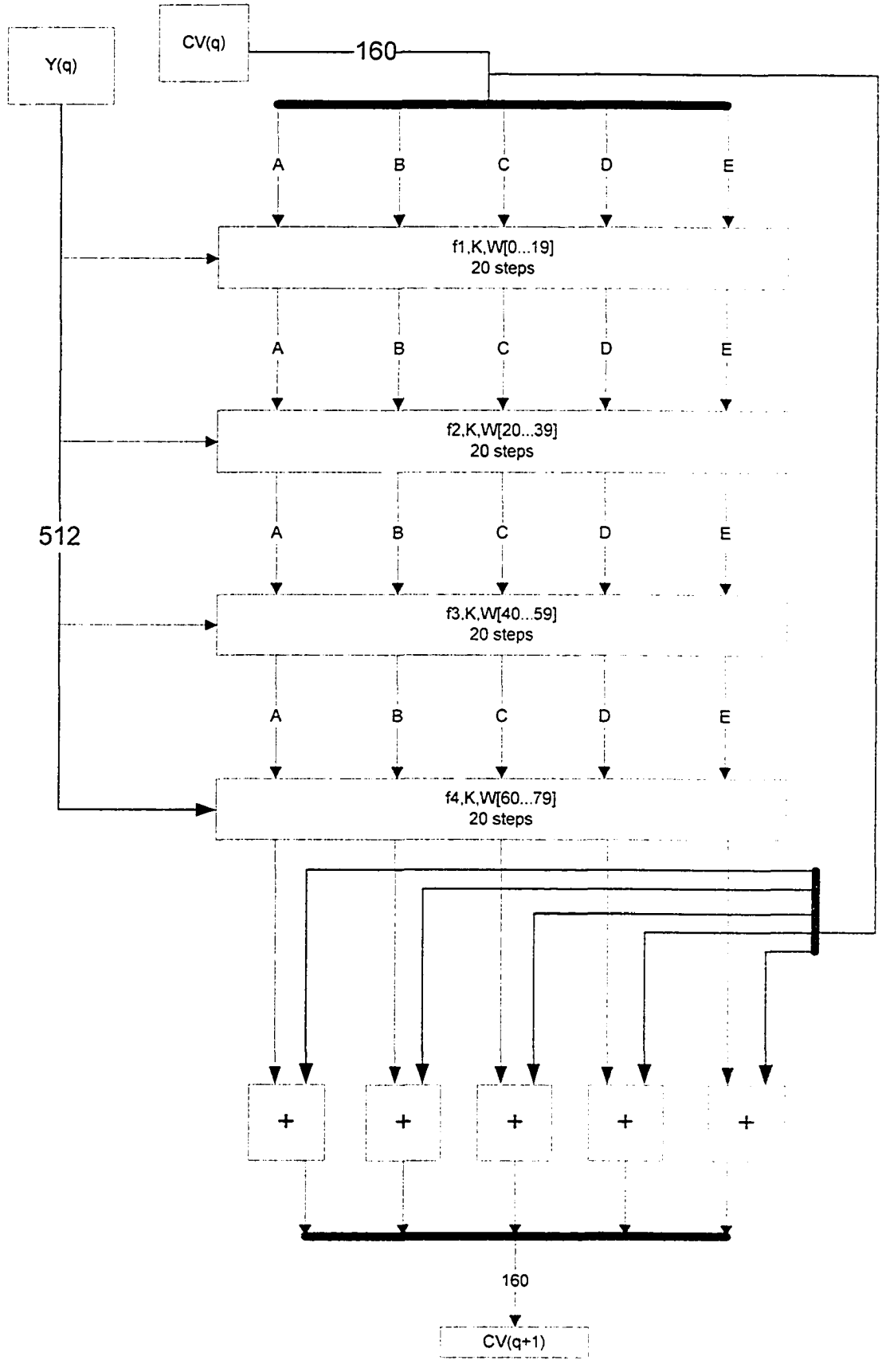
A = 67 45 23 01
B = EF CD AB 89
C = 98 BA DC FE
D = 10 32 54 76
E = $C3$ $D2$ $E1$ $F0$

Note that the first four values are the same as those used in MD5

4) Process the message in 512-bit blocks

The main idea of this algorithm which is different from MD5 is this algorithm is a module that consists of four rounds of processing of 20 steps each. The four rounds have a similar structure which is shown in figure 2.4

Figure 2.4



The difference in the primitive logical function is different from what we have seen in MD5, namely $f1, f2, f3, f4$. Each round takes the current 512-bit block being processed $Y(q)$ like in MD5 and the 160-bit buffer value $(A \quad B \quad C \quad D \quad E)$ as an input and updates the contents of the buffer. Also, each round makes use of an additive constant $K(t)$, where $0 \le t \le 79$ indicates one of the 80 steps across four rounds.

$$
\begin{array}{llllll}
0 \le t \le 19 & K(t) & = & 5A & 82 & 79 \quad 99 \\
20 \le t \le 39 & K(t) & = & 6E & D9 & EB \quad A1 \\
40 \le t \le 59 & K(t) & = & 8F & 1B & BC \quad DC \\
60 \le t \le 79 & K(t) & = & CA & 62 & C1 \quad D6
\end{array}
$$

As before, the output of the fourth round is added to the input to the first round $CV(q)$ to produces $CV(q+1)$. The addition is done independently for each of the five words in the buffer with each of the corresponding words in $CV(q)$, using addition modulo 2^{32}.

5) Output

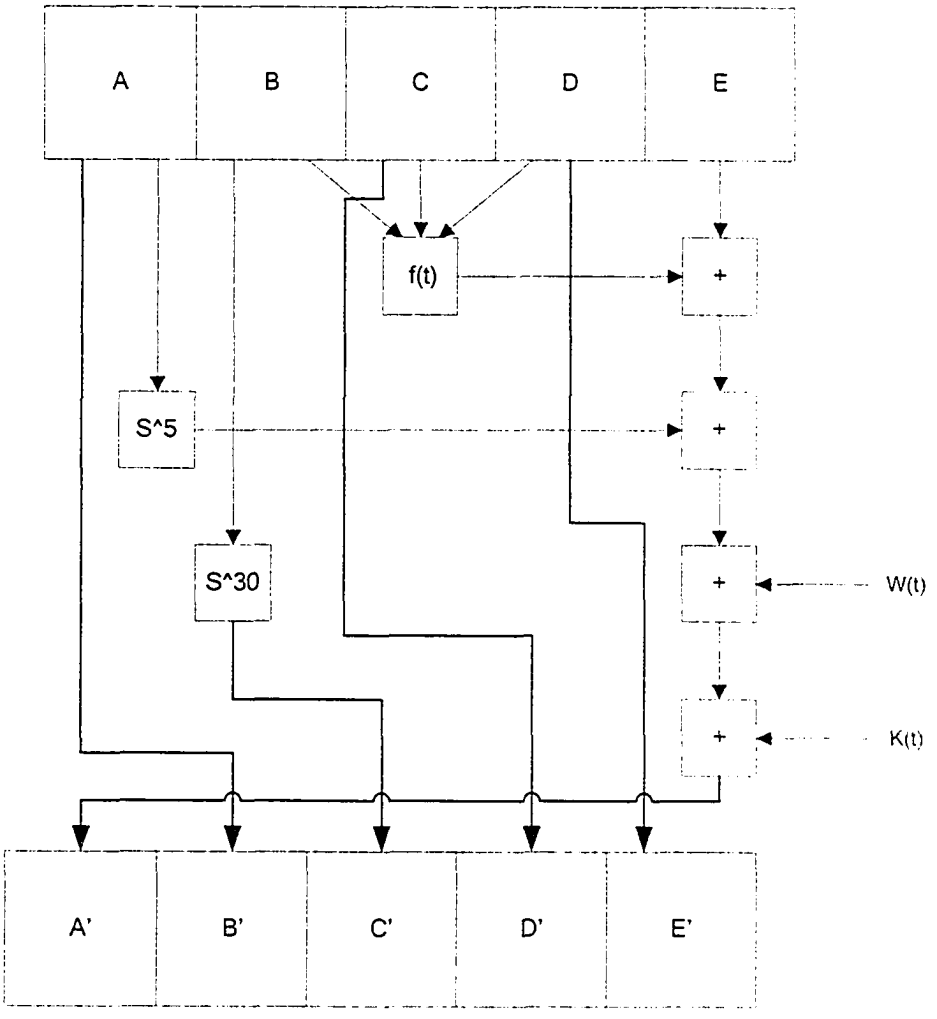
After all L 512-bit blocks have been processed; the output from the L^{th} stage is the 160-bit message digest. Same as MD5, we can summarize the behavior of SHA-1 as the following:

$$CV(0) = (A \quad B \quad C \quad D \quad E)$$

$$CV(q+1) = Sum(CV(q), RF(f4)[Y(q), RF(f3)[Y(q), RF(f2)[Y(q), RF(f1)[Y(q), CV(q)]]]])$$

$$MD = CV(L)$$

Figure 2.5



Let us look in more details about the logic in each of the 80 rounds of the processing of one 512-bit block. Shown in figure 2.5, each round performs the following formulas:

$A' \leftarrow (E + f(t, B, C, D) + S^5(A) + W(t) + K)$

$B' \leftarrow A$

$C' \leftarrow S^{30}(B)$

$D' \leftarrow C$

$E' \leftarrow D$

Where

$A \quad B \quad C \quad D \quad E$ = the five words of the buffer.

$A' \quad B' \quad C' \quad D' \quad E'$ = The five words of the buffer after processing the

	primitive function.
$f(t,B,C,D)$	= primitive logical function for step t
S^k	= circular left shift of the 32-bit argument by k bits
$W(t)$	= a 32-bit word derived from the current 512-bit input block
$K(t)$	= The four distinct values which are defined previously
$+$	= addition modulo 2^{32}

Each of the primitive functions $f1, f2, f3, f4$ has the following logic structures:

For $f1 = f(t,B,C,D)$, Function Value is $(B \wedge C) \vee (\overline{B} \wedge D)$

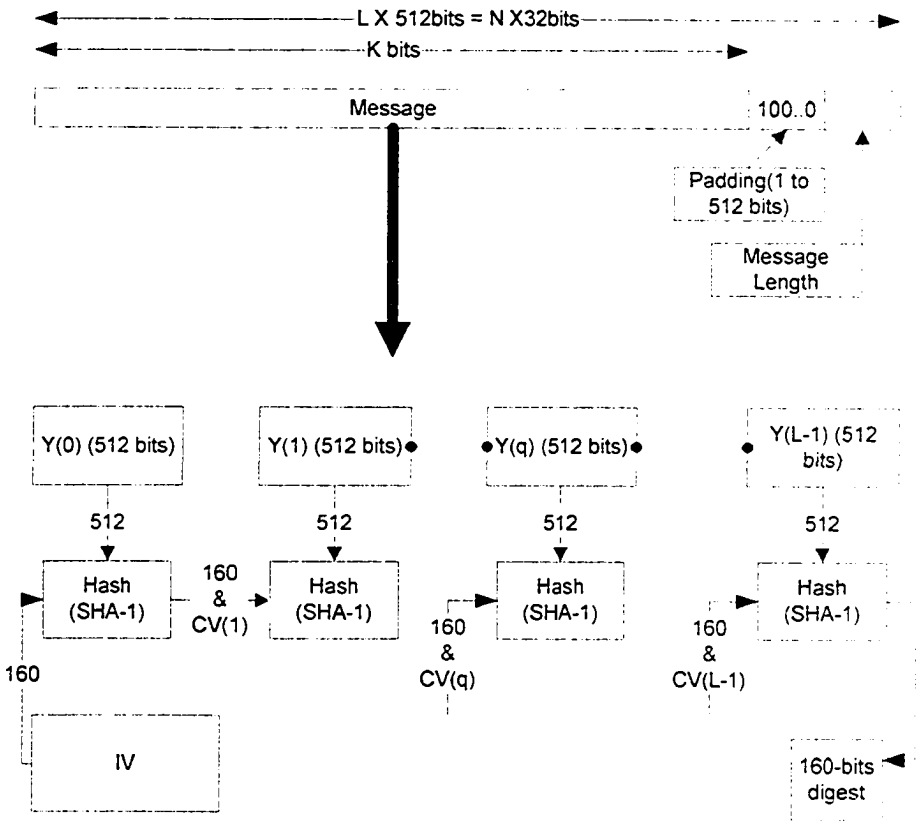
For $f2 = f(t,B,C,D)$, Function Value is $B \oplus C \oplus D$

For $f3 = f(t,B,C,D)$, Function Value is $(B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$

For $f4 = f(t,B,C,D)$, Function Value is $B \oplus C \oplus D$

The process starts from $t = 0$, each round updates the buffer by using the first K value and $f1$ as an input and the primitive logical function. Until $t = 20$, use second K value and $f2$ as a primitive logical function to update the buffer by performing the formula. Repeat the process until $t = 40$, using third K value and $f3$ as the primitive logical function as before. After that, when $t = 60$, use the last K value and $f4$ as the primitive logical function to update the buffer until $t = 79$. Use the result from the 80 steps to perform the function $Sum(\ )$ by taking $CV(q)$ as other input to create $CV(q+1)$ like in MD5. Figure 2.6 shows the overall process to produce a message digest.

Figure 2.6



MD5 and SHA-1 are quite similar to one another. Therefore, their strengths and characteristics should be similar. We can compare the two algorithms using the design goals: Security, Speed, Simplicity and compactness, and choice of the architecture [5, pg 285] [14].

1) Security:

The most obvious and most important difference is the SHA-1 digest is 32 bits longer than the MD5 digest. Using a brute-force technique, the difficulty of producing any message having a given message digest is on the order of 2^{128} operations for MD5 and 2^{160} for SHA-1. Also, using a brute-force technique, the difficulty of producing two messages having the same message digest is on the order of 2^{64} operations for DM5 and 2^{80} for SHA-1. Thus, SHA-1 is considerably stronger against brute-force attacks.

2) Speed:

SHA-1 involves more steps than MD5 (80 versus 64) and must proceed a 160-bit buffer compared to MD5's 128-bit buffer. Therefore, SHA-1 should execute more slowly than MD5 on the same hardware.

3) Simplicity and compactness:

Both algorithms are simple to describe and implement. Nowadays, MD5 and SHA-1 are already embedded into most of the software tools. Thus, the SHA-1 has no advantage over MD5.

4) Little-endian versus big-endian architecture:

MD5 uses a little-endian scheme for interpreting a message as a sequence of 32-bit words, where SHA-1 uses a big-endian scheme. There seems to be no strong advantage over either approach.

2.5 MAC

The MAC [4, pg 71], or message authentication code, is similar to a hash parameterized with a secret key. Each key you apply to the MAC gives you a different hash function. The hash is a surjection while the MAC is a family of surjections. You might view the hash as a MAC whose key has been carried away with a well-know constant. Different than Hash function, in order to use MAC both sender and recipient need the same key. The key is used to calculate the MAC. If the key is different the result for the sequence of bits for the ciphertext from the sender will be different when the recipient tries to verify the ciphertext. So what will Alice do if she wants to send something to Bob? First, she chooses a MAC function she likes to use with a key, a key that Alice and Bob share and use the message as an input. The MAC function will generate a unique output which shares the same property like Hash function that it is computationally infeasible to find the input M from the output C.

$$MAC(message + Key) = C$$

Now an attacker (Eve?) who wants to modify the message can no longer generate a new MAC for the forgery because she does not know the key. For the same reason, she cannot check whether her forgery matches the old MAC as well. MAC provides an improvement over Hash for the purpose of integrity but it now requires a mechanism to allow sender and recipient to establish a shared secret key.

2.6 HMAC

The hash function was not designed for the use as a MAC and cannot be used directly for that purpose because it does not rely on a secret key: instead it relies on a message. There have been a number of proposals for the incorporation of a secret key into an existing hash algorithm. The approach that has received the most support is HMAC [5, pg 293] [21]. HMAC has been chosen as mandatory to implement MAC for IP security and is used in other Internet protocols, such as SSL. The HMAC has the following design objectives [5, pg 294]:

1) To use available hash functions without any modifications.
2) To allow easy replaceability of the embedded hash function in case faster or more secure hash functions are found.
3) To preserve the original performance of the hash function without incurring a significant degradation.
4) To use and handle keys in a simple way.

Those objectives are important for the acceptability of HMAC. There are two benefits. First, an existing implementation of a hash function can be used as a module in implementing HMAC. Second, if any faster hash function module comes up, then we can just drop off the old hash function and implement the new hash function. Moreover, if the security of the embedded hash function is compromised, the security of HMAC could be retained simply by replacing the embedded hash function with a more secure one.

Figure 2.7

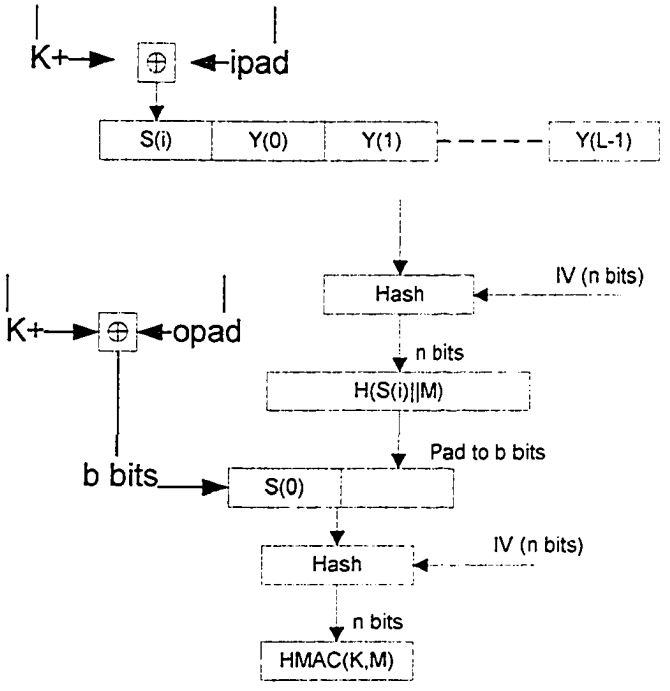


Figure 2.7 shows the overall operation of HMAC.
Definitions:

H	= embedded hash function (e.g. MD5 or SHA-1)
M	= message input to HMAC
$Y(i)$	= i^{th} block of M, $0 \le i \le L-1$
L	= number of blocks in M
b	= number of bits in a block
n	= length of hash code produced by embedded hash function
K	= secret key; if key length is greater than b, the key is input to the hash function to produce an n-bit key. Recommended length is $\ge n$
$K+$	= K padded with zeros on the left so that the result is b-bit string $K+$
ipad	= 00110110 is repeated $\frac{b}{8}$ times
opad	= 01011010 is repeated $\frac{b}{8}$ times

Assume Alice wants to send a message to Bob with message digest which is created by HMAC. She can perform the following:

Initialization steps:

1) Assume Alice and Bob share the same secret key K.
2) Embedded hash function H, such as MD5 or SHA-1.
3) Alice uses message M as an input for HMAC.

Operation steps:

1) Append zeros to the left end of K to create a b-bit string $K+$.
2) XOR $K+$ with *ipad* to produce the b-bit block $S(i)$
3) Append M to $S(i)$
4) Apply H to the stream generated in step 3.
5) XOR $K+$ with *opad* to produce the b-bit block $S(0)$
6) Append the hash result from step 4 to $S(0)$
7) Apply H to the stream generated in step 6 and output the message digest.

Note that, the XOR operation for $K+$ with *ipad* and *opad* can be done before we apply any hash function. This means we can have $S(i)$ and $S(0)$ pre-computed. The security of any MAC function is based on an embedded hash function depending in some ways on the cryptographic strength of the underlying hash function. The HMAC is considered as broken if an attacker, without having a key K, can find some message M' together with its correct HMAC value $HMAC(M', K)$. This is the same that an attacker who can forge the HMAC function can break the underlying hash function (MD5 or SHA) in one of the following ways [21]:

1) The attacker finds collisions in the hash function even when the IV is random and secret.

2) The attacker is able to compute an output of the compression function even with an IV that is random, secret and unknown to the attacker. (IV is the initial buffer in any hash function)

Since the feasibility of any of these attacks would contradict some of our basic assumptions about the cryptographic strength of these hash functions, the HMAC schemes could be secure. Could the secret key K be found after awhile when Alice and Bob repeatedly use it? For example, take a HMAC with output size of

128 bits as the example, the attacker need to acquire 2^{64} correct plain message with the corresponding HMAC value (with the same key) to find out the right HMAC secret key. This is considered as an impossible task in any realistic scenario even with a message block length of 64 bytes, this would take 250.000 years in continuous 1Gbps link and provided there is no change of the secret key during all this time. If we replace a strong HMAC with 160 bits output size then the number of original message and HMAC code required should become 2^{80} which is even harder and impossible to attack.

2.7 Internal/External Error Control

The Hash/Mac can provide not only the message authentication but also can save some time to decrypt the ciphertext. We know that to decrypt the ciphertext which is encrypted by using public key method can take a lot of time and resources. It is therefore not useful to first decrypt a message in order to subsequently perform error detection. We can use internal or external error control [5, pg 243]. Let's assume the Hash/Mac function $F(\ )$ which takes the message as an input and generates a unique output from $F(\ )$.

$$F\left(message \quad M\right) = checksum \quad D$$

The internal error control ensures that when Alice wants to send Bob a message, she can find the message checksum from the original message

$$F\left(Original \quad message \quad M\right) = checksum \quad D$$

and encrypt the message and the checksum D together and send the ciphertext to Bob. So once Bob gets the ciphertext, Bob decrypts it and gets the checksum D and the message M. What Bob can do is run the same Hash/Mac function $F(\ )$ and get another checksum. If both checksums from the ciphertext and the message are the same, then Bob can tell this is the message from Alice. (Shown in figure 2.8)

Definition:

M = A message from Alice.

C = A ciphertext generated by the encryption function.

$F(\ )$ = Hash/Mac function

D = The checksum generated by $F(\ )$.

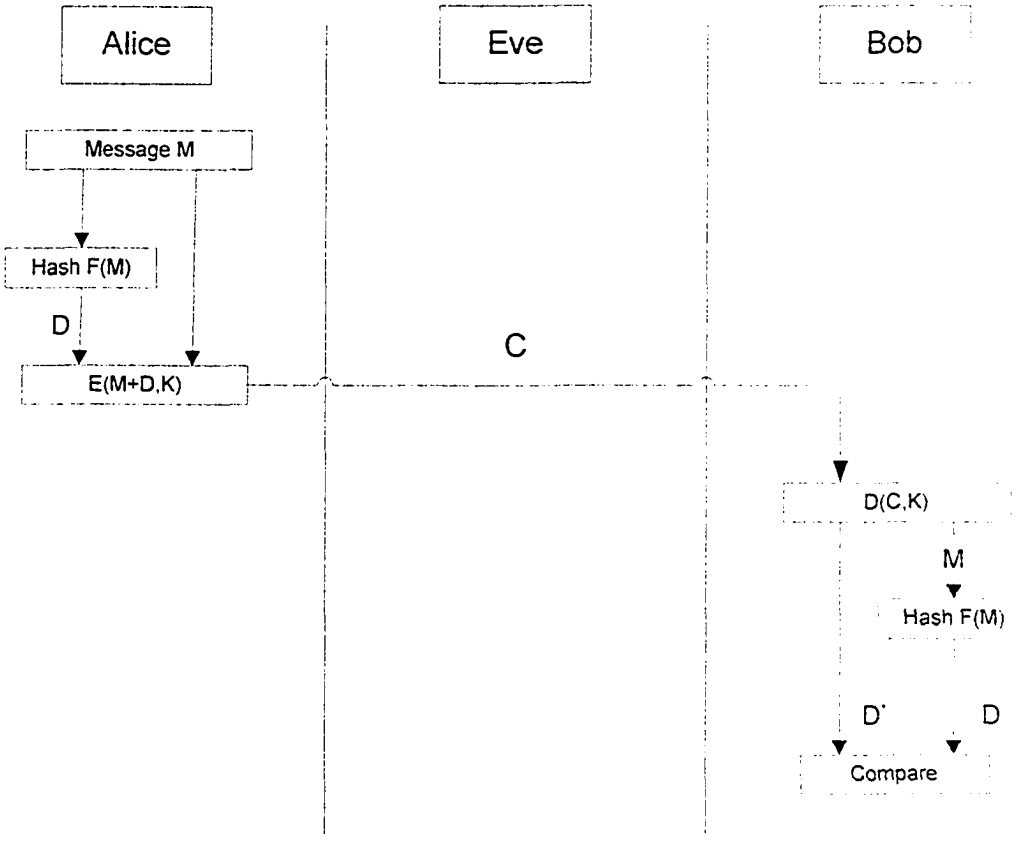
$E(x, K) = C$

= Any encryption function. The function takes an input x and encrypts it by using the key K to generate a ciphertext C

$D(C, K) = x$

= The decryption function. The function takes an input C and decrypts it by using the key K to generate a message x. The key K can be different than the encryption function depending on what encryption function we choose.

Figure 2.8



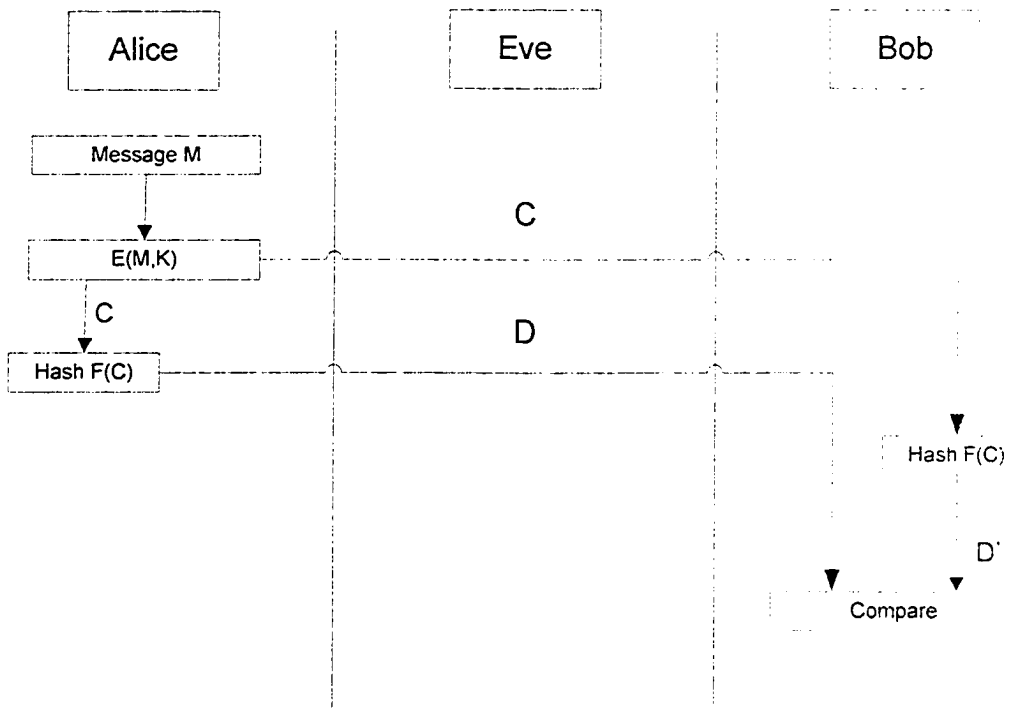
In the case of the external error control. Alice encrypts the message first and

runs the Hash/Mac function after it has been encrypted.

$$F\left(ciphertext \quad C\right) = checksum \quad D$$

She sends both the checksum and ciphertext to Bob. After Bob gets the ciphertext, he can run the Hash/Mac function first before he decrypts it. If both the checksum from the ciphertext and the Hash/Mac function are the same, then Bob can run the decryption on ciphertext to get the message. In this case, if the checksum is not the same then Bob can detect it before he decrypts it. This can save him a lot of time and resources. (Shown in figure 2.9)

Figure 2.9



With internal error control, authentication is provided because an opponent would have difficulty generating ciphertext that, when decrypted, would have valid error control bits. On the other hand, with external error control, authentication may not be provided because of the man-in-the-middle attack. An opponent can get the original ciphertext from Alice and use his own message and Hash/Mac

function to generate other checksum to fault Bob. However, as far as decryption is concerned, external error control provides better run time because Bob does not need to decrypt the ciphertext to run the Hash/Mac function. Instead he can just run the Hash/Mac function by using the ciphertext as an input. Depending on the application, the choice to use internal or external error control will be different.

Chapter 3: Authentication

Authentication is the process of verifying a principal's claimed identity. In the network connection, we do not only need to identify the received message or information but also the receiver or sender. The sender or the receiver can be a computer or program. For example, if Eve knows Alice's or Bob's password (login id, or user name) she can login to the computer to pretend to be Alice or Bob. All of the integrity service which we have mentioned in the last section cannot detect it. Therefore, integrity service can only help us to achieve certain authentication needs with respect to network connections but we need to consider more possibilities to prevent mis-authentication.

3.1 Authentication Requirement

The following attacks can be identified against authentication [5, pg 238]:

1) Traffic analysis: Discovery of the pattern of traffic between parties. In a connection-oriented application, the frequency and duration of connections could be determined. In either connection-oriented or connectionless environment, the number and length of messages between parties could be determined.

2) Masquerade: Insertion of messages into the network from a fraudulent source. This includes the creation of messages by an opponent that are purported to come from an authorized entity. It also includes fraudulent acknowledgments of message receipt or nonreceipt by someone other than the message recipient.

3) Sequence modification: Any modification to a sequence of messages between parties, including insertion, deletion, and reordering.

4) Timing modification: Delay or replay of messages. In a connection-oriented application, an entire session or sequence of messages of some previous valid session could be a replayed, or individual messages in the sequence could be delayed or replayed. In a connectionless application, an individual message

could be delayed or replayed.

In general, authentication requires that with proper protocols and techniques we can identify the origin of the message and also the authenicity of the content of the message.

3.2 Password

Let us assume that Alice wishes to authenticate herself to a computer C: She is known to C by her name "Alice" (or some other login name) and can prove her identity by demonstrating knowledge of a secret password p that she previously agreed with C. The problem here is that, if C keeps a list of the passwords of its users, this password file becomes a valuable target for an attacker. Encrypting the file containing the password would bring little benefit since C itself would need the key to decrypt in order to be able to check any supplied password. So the attacker who manages to break into C could simply steal both the key and the encrypted file. Therefore, to address this problem, C just simply records a hash value of Alice's password instead of the password. This is a technique pioneered by Needham in the 1960s [4, pg 76]. This means that an attacker grabbing the list stored on C does not have access to any passwords. Even with this improvement, the system is still vulnerable. There is one attack which is similar to brute force attack and it is called the *dictionary attack.* Many users will choose passwords that are easy to guess. The attacker generates hashes of such candidate guesses and checks whether the result matches any entry in the list stolen from C. If it does, a password has been found. The guesses may even be pre-computed once and for all and stored in a lookup table indexed by the hash value. Then, the attacker can just simply crack on sight any account that used one of the guesses as its password. The method against this attack can be *salting* or slowing down the hash function in order to increase the attacker's workload. Salting is the practice of adding random bits to the user-supplied password before hashing it. These random bits have to be stored on C in the password list next to the hash value, otherwise the system itself would not be able to regenerate the hash for comparison even if the correct password is supplied. The technique does not prevent the attacker from verifying the validity of a guess, but it frustrates pre-computation and parallel search. Furthermore, two users might choose the same password, without salting, their hash values would be identical, telling an attacker that the penetration of one account also opens the door

to the other, and also indicates that the password is so easy to guess while those two users thought of it independently.

Another suggested method from Lamport is to use a chain of hashes to protect the password. The idea is to generate a series of passwords $p_0, p_1, \ldots, p_n$ linked by the recurrence relation $p_i = h(p_{i-1})$. In this case, even if the attacker got the file containing the list of hashes, it will be limited with respect to using pre-computation and parallel search as well. The disadvantage, of course, is the cost to compute the hash every time when Alice logins. Moreover, if Alice exhibits p or $h(p)$ (hash result) in cleartext across the network, an attacker can record what Alice says and replays it later to impersonate her. Note that the hash offers no protection here. The attacker won't know the actual password, but would not need it to impersonate her. This attack is defined as a *replay attack*. To prevent this attack, it is necessary to ensure that Alice's authenticators cannot be predicated from previous ones. This mean that the p and $h(p)$ have to be always different after it has been transmitted once.

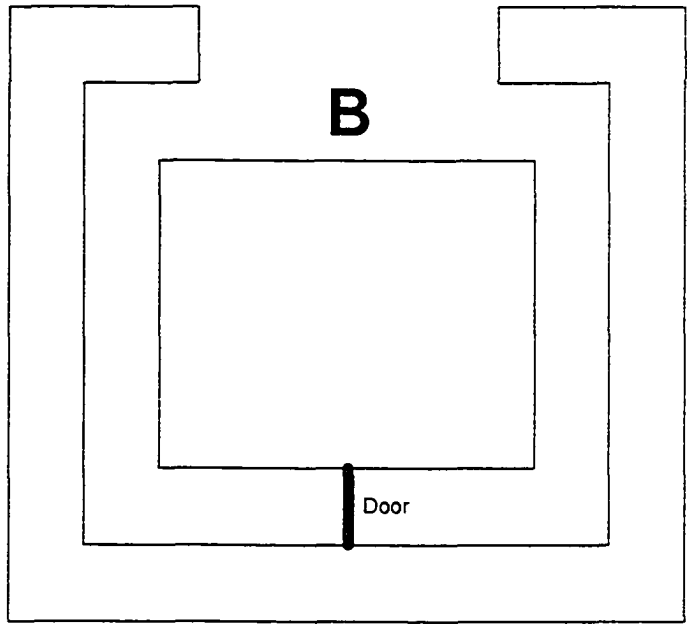
3.3 Zero – Knowledge Proofs

Consider the following scenario: A computer reports to Bob that it received the ciphertext from Alice. The ciphertext requires Bob's private key to decode the ciphertext and asking Bob to input his private key. Can Bob be sure that here is really some ciphertext sent from Alice? Can Bob be sure that after he inputs his private key to the computer he will see the message?

One of the possibilities is that Eve may setup a fake computer that tells Bob the computer received a ciphertext from Alice and asking him to input his private key. If Bob inputs his private key, then the computer may show up an error message to Bob that the key is in error or the ciphertext is in error in order to fool Bob. At the same time, Eve can retrieve Bob's private key for later use. A few years ago, similar scenarios happened on ATM machines. Some thieves set up a fake ATM machine at a shopping mall. When a person inserted a bank card and typed in an ID number, the machine recorded the information but responded with the message that it could not accept the card. The thieves then would make counterfeit bank cards and go to legitimate ATM machine to withdraw cash, using the ID they had obtained.

To avoid that problem, what we need is a way to use the secret number to identify a person or computer without giving any information that can be reused by an eavesdropper. This is where zero-knowledge techniques come in. The basic challenge-response protocol [11] is best illustrated by an example due to Quisquater, Guillou, and Berson. (Shown in figure 3.1)

Figure 3.1



Suppose there is a tunnel with a door like in figure 3.1. Alice wants to prove to Bob that she can go through the door without giving any information to Bob about how she does and which direction she can pass through the door. They proceed as following: Alice enters the tunnel at point B and goes down either the left side or the right side of the tunnel without Bob knowing in which side she goes. There are some physical constraints, so that Bob needs to wait outside for a minute, then comes in at point B can calls Alice out from "Left" or "Right". Alice then comes to point B by the left or right tunnel, as requested. This entire protocol is repeated several times until Bob is satisfied. Of course, in each round, Alice chooses which side she will go down and where Bob chooses which side he will request randomly.

Since Alice must choose to go down the left or right side before she knows what Bob will say, she has only a 50% chance of fooling Bob if she does not know how to go through the door. Therefore, if Alice comes out the correct side for each

of 10 repetitions, there is only one chance in $2^{10} = 1024$ possible that Alice does not know how to go through the door. At this point, Bob is probably convinced.

Suppose Eve is watching the process on a video monitor set up at point B. She will not be able to use information she sees to convince Bob or anyone else that she can go through the door. Moreover, she might not even be convinced that Alice can go through the door. The sequence of rights and lefts which were chosen by Bob is random and ahead of time before Alice goes into the tunnel. By this reasoning, there is no useful information that Eve obtains that can be transmitted to anyone. Note that there is never a proof in a strict mathematical sense that Alice can go through the door. But there is overwhelming evidence which obtained through a series of challenges-responses sequences. This is a feature of zero-knowledge "proofs".

There are several mathematical versions of this procedure and we will concentrate on one of them. The process is called Feige-Fiat-Shamir identification scheme [12]. This is used as the basis of an identification scheme.

Definition:

p = Prime number.

q = Prime number.

$n = pq$ = The product of two primes p and q.

r = The random integer less then n.

k = The length of the number which Bob chooses to send to Alice.

$b = b_1 \ldots\ldots b_k$ = The numbers which Bob chooses to send to Alice.

$s = s_1 \ldots\ldots s_k$ = The secret numbers which Alice possesses.

v = The numbers v is calculated by $v_i \equiv s_i^{-2} \bmod n$.

$f(r) = r^2 \bmod n = x(\bmod n)$

= The function takes r as an input and calculate the result x.

$F(r,s,b) = r s_1^{b_1} s_2^{b_2} \bullet\bullet\bullet s_k^{b_k} \bmod n = y(\bmod n)$

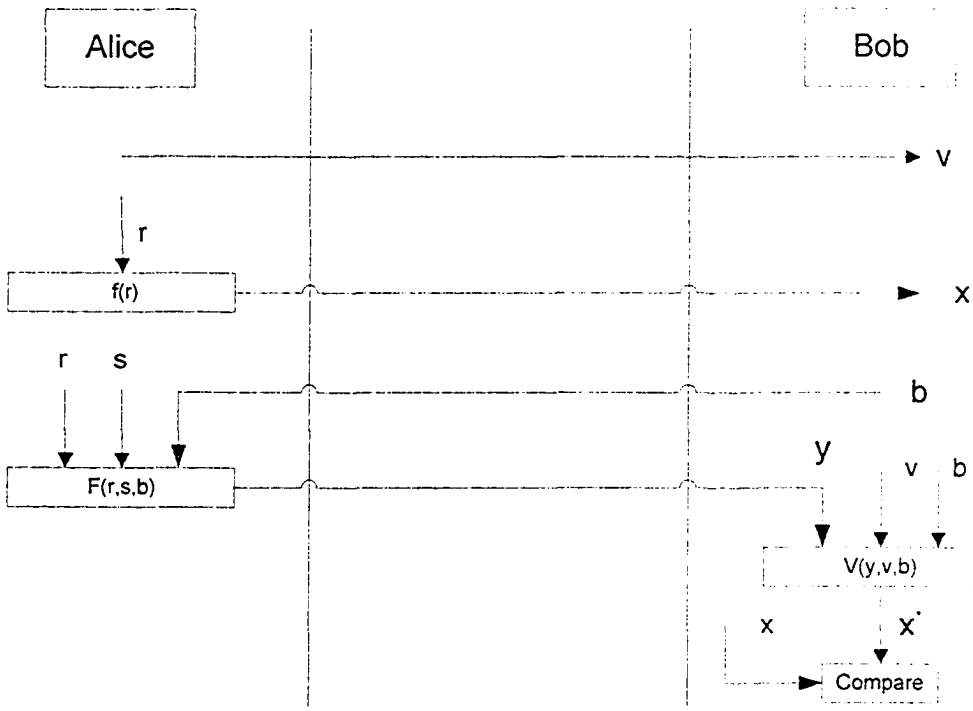
= The function takes r, s, and b as an input and calculate the result y.

$V(y,v,b) = y^2 v_1^{b_1} v_2^{b_2} \bullet\bullet\bullet v_k^{b_k} \bmod n = x'(\bmod n)$

= The function takes y, v, and b as an input and calculate the result x'

Let $n = pq$ be the product of two large primes p and q. Alice has secret number $s_1,.....,s_k$. Let $v_i \equiv s_i^{-2} \bmod n$ where $\gcd(s_i, n) = 1$. The numbers v_i are sent to Bob. Bob will try to verify that Alice knows the number $s_1,.....s_k$. Alice and Bob proceed as follows: (Shown in figure 3.2)

Figure 3.2



1) Alice chooses a random integer r, computes $x \equiv r^2 \bmod n$ and sends x to Bob.

2) Bob chooses number $b_1,.....b_k$ with each $b_i \in \{0,1\}$. He sends these to Alice.

3) Alice computes $y \equiv r s_1^{b_1} s_2^{b_2} \bullet\bullet\bullet s_k^{b_k} \bmod n$ and sends y to Bob.

4) Bob checks that $x' \equiv y^2 v_1^{b_1} v_2^{b_2} \bullet\bullet\bullet v_k^{b_k} \bmod n$.

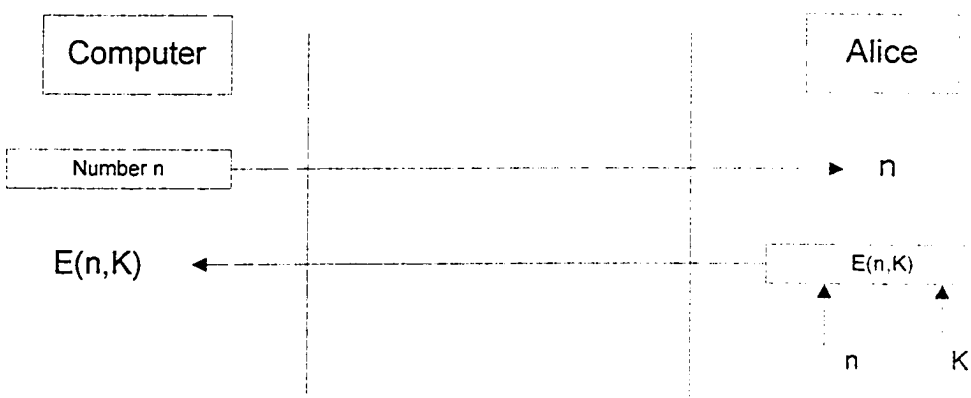
5) Repeat Step 1 through Step 4 until Bob is satisfied.

Consider the case $k=1$. Then Alice is asked for either r or rs_1. These are two random numbers whose quotient is a square root of v_1. Therefore, this method gives Alice a 50% chance of fooling Bob. Now consider the case of larger k. Suppose Bob sends $b_1=1, b_2=0, b_3=1, b_4=1, b_5=0 \ldots\ldots b_k=0$. Then Alice must produce $y \equiv rs_1s_3s_4$, which is a square root of $xv_1v_3v_4$. In each round, Bob is asking for a square root of a number and Alice can supply a square root if she knows $r, s_1 \ldots\ldots s_k$. If she cannot supply it, then she will have a hard time to compute a square root to fool Bob.

3.4 Challenge-response and man-in-the-middle attacks

Authentication can be performed in a stateless manner by using a challenge response strategy. The term "stateless" means that the computer has no information about what the user does previously. Therefore, the computer does not know whether the response from Alice is from the same person who responded earlier. Computer C generates a random number n and asks Alice to return n which is encrypted (Shown in figure 3.3).

Figure 3.3



Note that in this case a MAC, which is not invertible, would serve the same purpose as the encryption, since the computer is in no need to decrypt the result to verify. The computer simply performs the encryption on n and compares the result which what is returned from Alice. This is the model originally used for Air Force IFF(identify friend or foe) systems. The system operates in this way: if the fighters from the same side share a secret key, they can challenge each other via radio

transmission. If they can prove that they have the same secret key, this means they are from the same alliance. This method is safe from passive replay attacks but it remains vulnerable to active attacks, such as "*man-in-the-middle*".

Let us discuss what the "*man-in-the-middle*" attack is. There are two parties communicate through the network like what we have described at the beginning. (Shown in figure 3.4)

Definition:

Sa = Alice's private key.

Pa = Alice's public key.

Sb = Bob's private key.

Pb = Bob's public key.

C = The ciphertext.

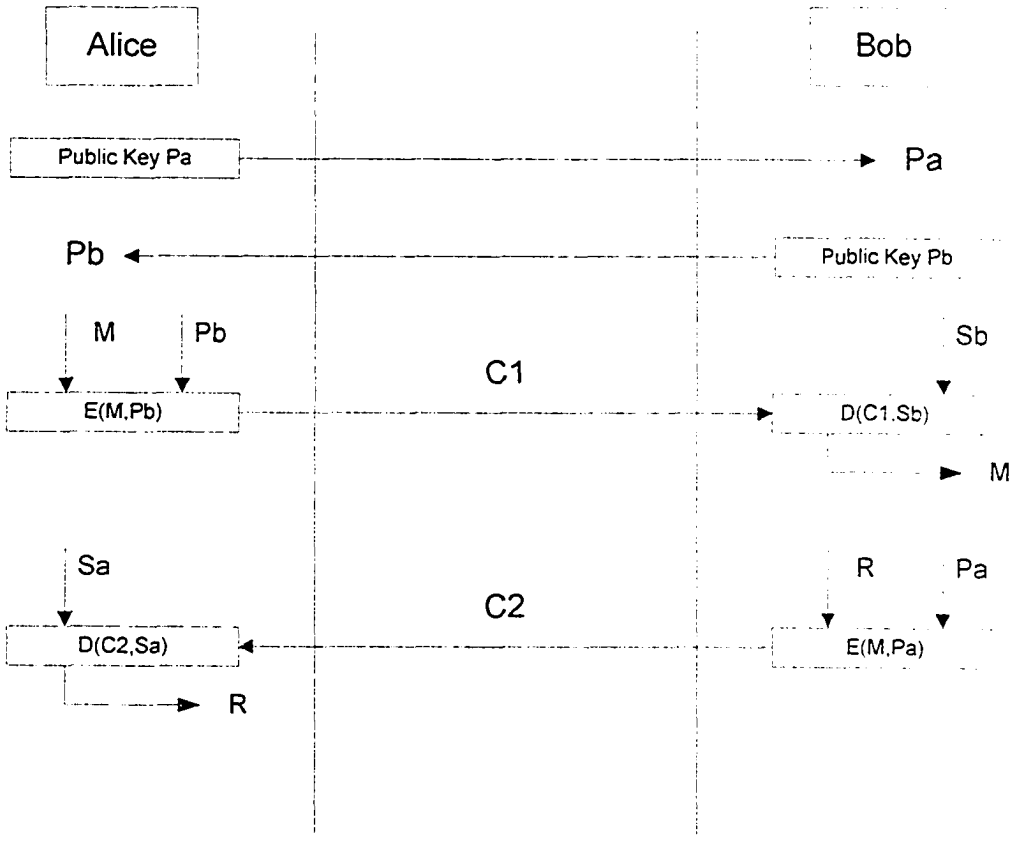
M = Original message from Alice.

R = Reply message from Bob.

$E(x, y)$ = Encryption function.
Takes a key y and encrypts the input x.

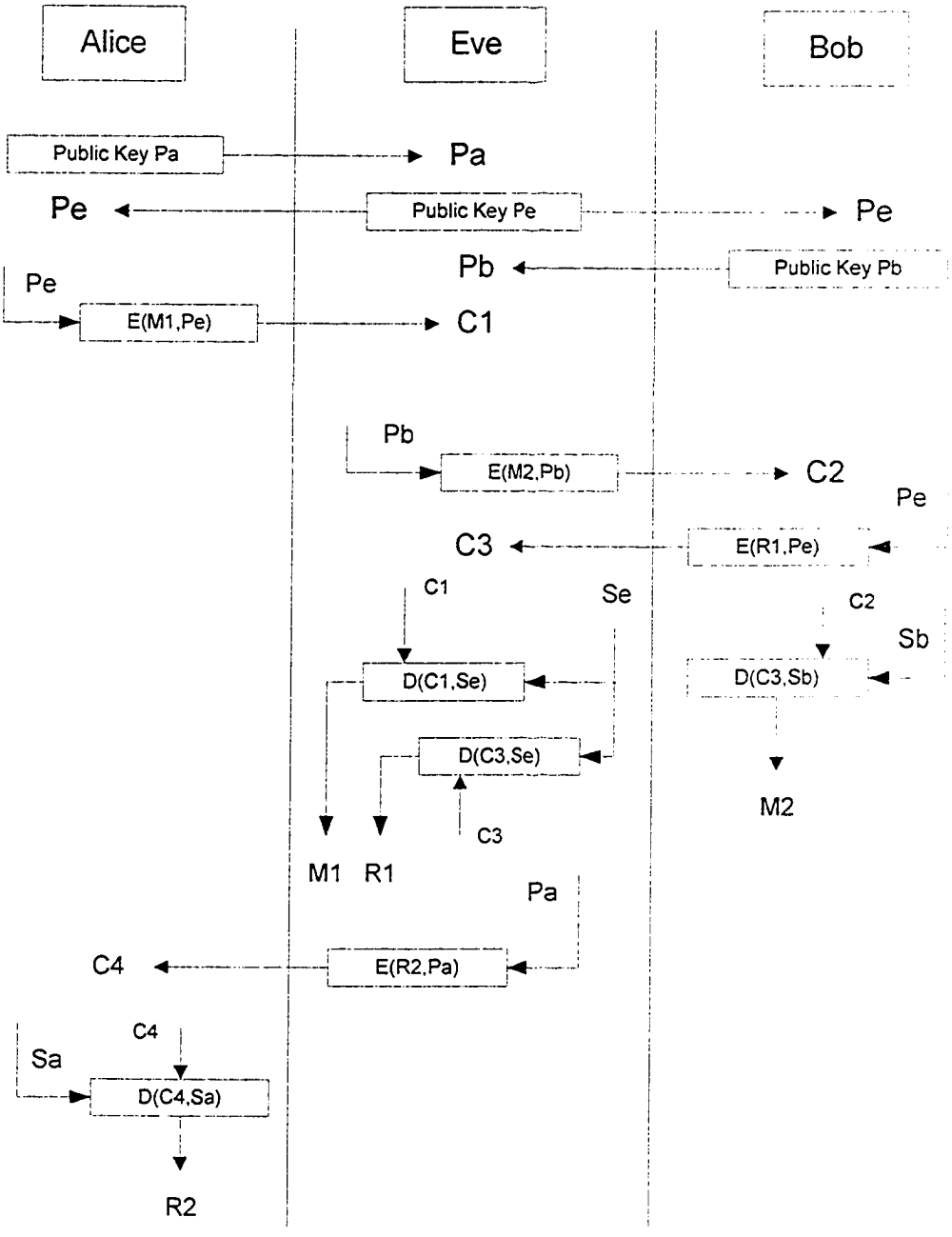
$D(x, y)$ = Decryption function.
Takes a key y and decrypts the ciphertext x.

Figure 3.4



Alice sends her public key to Bob and Bob replies her with his public key. So in this case, Alice can send Bob a ciphertext encrypted by Bob's public key and Bob can also send Alice a ciphertext encrypted by Alice's public key. However, there is vulnerability of this scheme. (Shown in figure 3.5) If there is a third party, says Eve, who interrupts the connection and is able to get the message from Alice. Eve is still unable to find Alice's and Bob's private keys from their public keys but she can read the message from Alice and modify it or redirect it to Bob.

Figure 3.5



Alice wants to send her public key to Bob. At this point Eve interrupts the connection and gets the public key from Alice. Now, Eve sends her own public key to Bob and impersonates Alice. Bob receives the public key and replies to Eve: (Thinking it is Alice) with his public key. Once Eve receives the reply from Bob, she can reply to Alice with another public key. At this point, if Bob wants to send a secret message to Alice, he will encrypt the message using Eve's public key. So Eve can decrypt the ciphertext from Bob and reply to Alice with other message using

Alice's public key. Another possibility is that if Alice wants to send a secret message to Bob, Eve can decrypt it and modify it as well. In this scenario, we can see that this position which is vulnerable to attack involves the key distribution and authentication problem. One possible solution is using digital signature with an arbitrator in between the sender and receiver.

3.5 Public Key Infrastructures (PKI)

Public key cryptography is a powerful tool that allows confidentiality, non-repudiation, and key distribution for symmetric cryptography [6, pg 246]. However, when the public key is published or private key is signed as a digital signature, what assurances do we have so that Alice's public and private key actually belong to Alice? Eve could have substituted her own public key in place of Alice's, as in the *man-in-the-middle attack*. The digital signature will provide a better improvement against the attack, but the public key cryptography is still vulnerable unless the keys are generated in a manner that does not break confidentiality.

In order for public key cryptography to be useful in communication, it is necessary to have an infrastructure that keeps track of public keys. A public key infrastructure (PKI) is a framework consisting of policies which define the rules of operation and procedures of particular cryptographic systems for generating and publishing keys and certificates. All PKIs consist of certification and validation operations. The certification binds a public key to an entity, such as a user. Validation guarantees the certificates are valid.

A certificate is a quantity of information that has been signed by its publisher: usually referred to as the *certification authority* (CA). There are many types of certificates, and they are composed of at least two parts: User's identity and User's public key. Some of the certificates may also include the CA's identity, certificate's identity number, date of issuance, and the time period for which the certificate is valid. Two popular ones are identity certificates and credential certificates. Identity certificates contain an entity's identity information, such as email address, and a list of public keys for the entity. Credential certificates contain information describing access rights. In either case, the data are typically encrypted using the CA's private key. (See Figure 3.6)

Definition:

Pa = Alice's public key

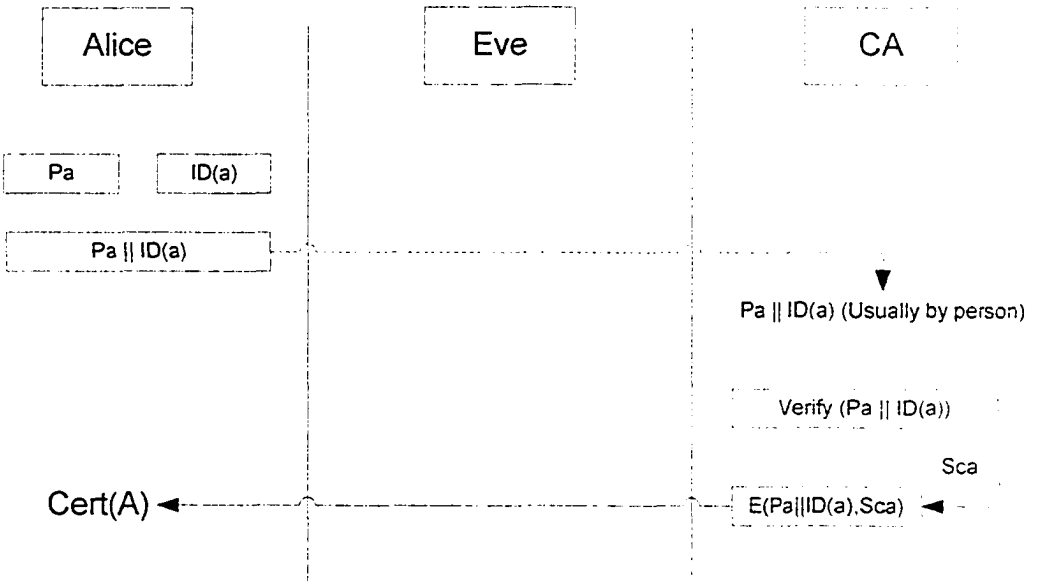
$ID(a)$ = Alice's identification.

Sca = CA's private key.

$Cert(A)$ = Certificate created by the CA.

$E(x, y)$ = Encryption function. Takes a key y and encrypts the input x to generate the ciphertext. If the ciphertext is created by the CA, then the ciphertext may be the certificate $Cert$.

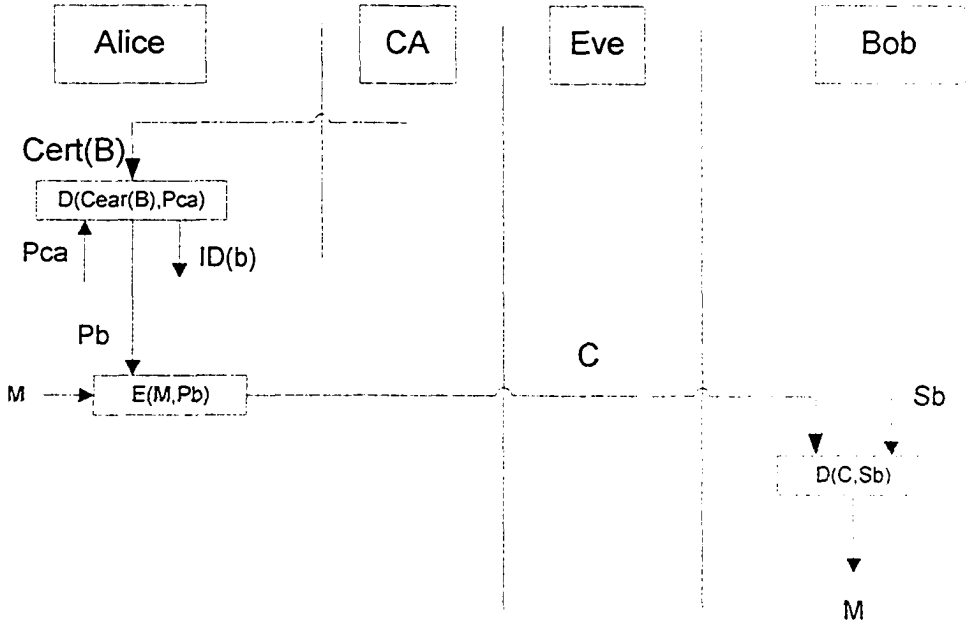
Figure 3.6



Suppose we have a PKI, and the CA publishes identity certificates for Alice and Bob. If Alice and Bob know the CA's public key, then Alice can take the encrypted identity certificate for Bob that has been published and extract Bob's identity information as well as a list of public keys needed to communicate securely with Bob.

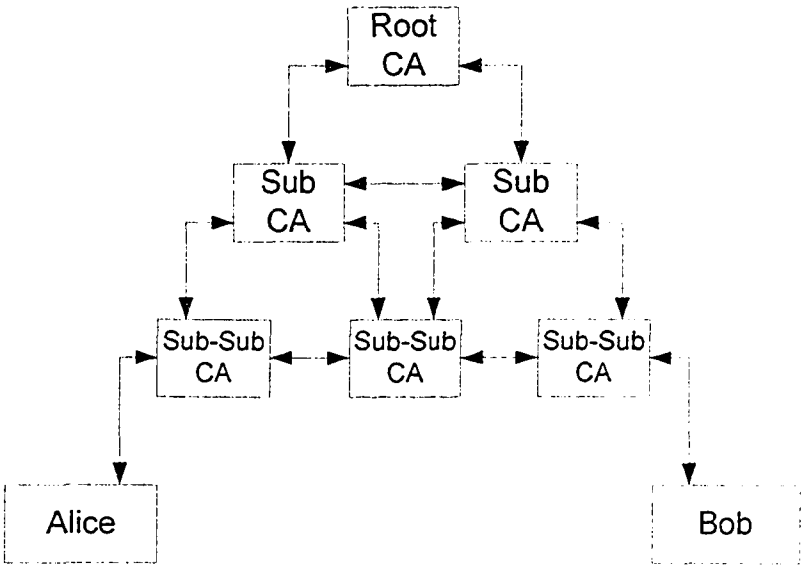
(Shown in figure 3.7)

Figure 3.7



In the last scenario, Alice and Bob need to trust the CA and CA's certificates. The concept of trust is critical to PKI's and perhaps is one of the most important properties. It is unlikely that a single entity could ever keep track of and issue every user's public keys. Instead, PKIs often consist of multiple CAs that are allowed to certify each other and the certificates they issue. Thus, Bob might be associated with a different CA from Alice. When Alice requests Bob's identity certificates, she might only trust it if she trusts Bob's CA. It is necessary for each of the CAs between Alice and Bob to trust each other. In addition, most PKIs have varying levels of trust, allowing some CAs to certify other CAs with varying degrees of trust. It is possible that CAs may only trust other CAs to perform specific tasks. For example, there can be a root CA issue certificates to other CAs, thereby constructing a certificate hierarchy similar to figure 3.8 [1, pg 86]

Figure 3.8



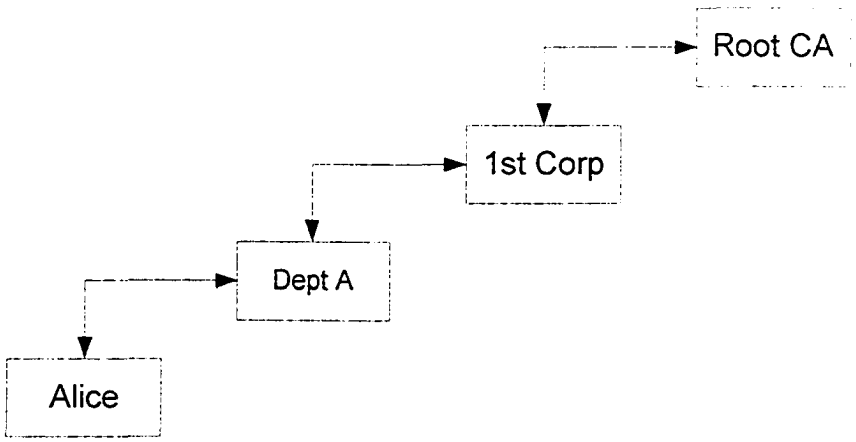
The root CA issues certificates to sub-CAs, which issues certificates to sub-sub CAs and on down to the individual like Alice or Bob. The sequence of certificates forms a chain of trust. As we can see in figure 3.8, the root CA has issued certificates to the First and Second Corporation. Within each Corporation there is a sub-CAs which issues certificates to its own department. Within the first corporation, there is Department A which can have another sub-CAs issuing a certificate for its employee, named Alice. Alice's basic certificate may only contain the following [1, pg 87]:

CERT for Alice = {Alice's public key + Alice's identity}

But for now, within the certificate hierarchy Alice's certificate will reside at the end of a certificate chain like the following:

$CERT_1$ = {CERT for Alice} issue from Dept A.
$CERT_2$ = {CERT for Dept A} issue from 1st Corp
CERT = {$CERT_1$ + $CERT_2$} issue from Root CA

Figure 3.9 Alice's Certificate Chain



If Bob wishes to authenticate a message that Alice has purportedly signed, he can proceed up the certificate chain until he can find a certificate he can trust. In this case, it would be the root CA certificate. Then he can progress back down the certificate chain, authenticate each certificate, in turn, with the trusted public key from the preceding level.

Now we come to a problem that what happens if Alice's certificate becomes invalid prior to its expiry date. For instance, Alice might lose her computer that contains her secret keys. Other possibility is Alice's private key is somehow discovered by other parties. Under these circumstances, Alice's certificate would be invalid because it may no longer represent Alice. In this case, Alice should report the possible compromise of her secret keys to her CA. If this happens, what the CA would do when Alice's secret key may be compromised or when she no longer meets the certification criteria? One of the most frequently talked-about mechanisms for invalidating certificates is called *certificate revocation*. If a CA is given valid information that either Alice's secret keys have been compromised or that Alice no longer meets the certification criteria, then the CA places Alice's certificate on a certification revocation list (CRL). Placing a certificate onto a CRL is a serious business. In the CA's certification practice statement, the CA will have to validate the claim of key compromise or certificate invalidation. The CA must be cautious and not revoke valid certificates without sufficient proof of the authenticity of the revocation evidence. If the evidence is fraudulent, the CA may deny their ability to conduct business of the valid certificate users. Also, CA must post invalid certificates on the CRL as quickly as possible to minimize the possible

fraudulent use of invalid certificates. The CA passes out the CRL is controversial. There are two possible methods the CA can do to put the certificates onto the CRL. The first method we called it "The Push model", in which the CA updates the CRL and then sends it out to recipients on a scheduled basis. This method has the advantage of automatically giving the updated CRL but the CA must use significant bandwidth in the periodic broadcast of the CRL. The second method we called it "The Pull model", in which the recipient must query the CA each time when the recipient has a question about the validity of a certificate. This method has the advantage that the CA does not need to send the CRL periodically to each recipient but someone could interfere with the CRL queries and deny the recipients access to the information they require. Another disadvantage for the "The Pull model" is that there are so many recipients but less CA. If every recipient requests verification from the CA, then the CA server may be tied up and cannot handle all of the requests from many recipients at once.

Incidentally, certain programs (for example web browsers) have built-in into them a few public keys of certification authorities that they trust. If a user browses the Internet and requires a certification for a website (For example, Bank webpage) which is not built-in the web browser, then the web browser downloads it from the web server and saves it for later use. In principle, one could modify the executable program to alter the authorities being trusted. Hence it is crucial to guarantee the integrity of downloaded programs that have built-in trust relationships to certification authorities.

Chapter 4: Nonrepudiation

Nonrepudiation is the property that neither the sender nor the receiver of a message is able to deny the transmission. Up to the previous section, all the methods we provide can only protect two parties who exchange messages from any third party. However, it does not protect the two parties against each other. The most important development from the public key cryptography is the digital signature. The digital signature provides a set of security capabilities that would be difficult to implement in any other way. With this level of difficulty, we can achieve the purpose of nonrepudiation.

4.1 Digital Signatures Requirement

Before we introduce any algorithm for digital signature, let us start with some requirements for digital signatures. To understand that, we have to know what can happen after the transmission. We assume Alice sends an authenticated message to Bob using any public key algorithm to protect it. After the transmission, Bob may forge a different message and claim that it came from Alice. Bob would simply have to create a message and append a checksum from the Hash/Mac which Alice and Bob share. Another possibility is that, Alice can deny sending the message. Since it is possible for Bob to forge a message, there is no way to prove that Alice in fact sent the message. In situations where there is no complete trust between sender and receiver, something more than authentication is needed. The most attractive solution to this problem is the digital signature. The digital signature is analogous to the handwritten signature. It must have the following properties [5, pg 300]:

1) It must be able to verify the author and the date and time of the signature.
2) It must be able to authenticate the contents at the time of the signature.
3) The signature must be verifiable by *third parties*, to resolve disputes.

Thus, the digital signature function includes the authentication function.

On the basis of these properties, we can formulate the following requirement

for a digital signature [5. pg 300]:

1) The signature must be a bit pattern that depends on the message being signed.
2) The signature must use some information unique to the sender, to prevent both forgery and denial.
3) It must be relatively easy to produce the digital signature.
4) It must be relatively easy to recognize and verify the digital signature.
5) It must be computationally infeasible to forge a digital signature or to construct a fraudulent digital signature for a given message.
6) It must be practical to retain a copy of the digital signature in storage.

A variety of approaches has been proposed for the digital signature function. These approaches fall into two categories: direct and arbitrated.

4.2 Direct Digital Signature

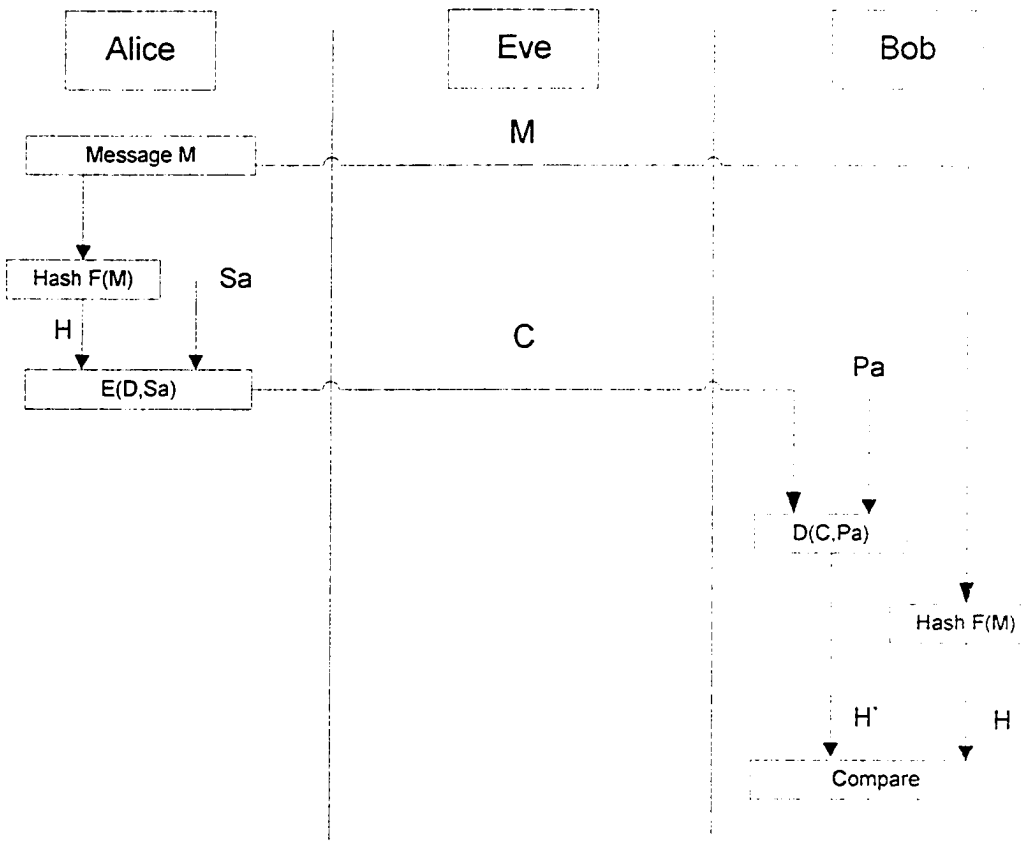
The direct digital signature involves only the communicating parties, sender and receiver. It is assumed that the receiver knows the public key of the sender. A digital signature may be formed by encrypting the entire message with the sender's private key or by encrypting a hash code of the message with the sender's private key. Confidentiality can be provided by encrypting the entire message and the signature with the receiver's public key. It is important to perform the signature function first then an outer confidentiality function. In case of dispute, some third parties must view the message and its signature. If the signature is calculated on an encrypted message, then the third party also needs access to the decryption key to read the original message. However, if the signature is the inner operation, then the recipient can store the plaintext message and its signature for later use in dispute resolution.

All direct schemes discussed so far have a common weakness, which depends on the security of the sender's private key. If the sender wishes to deny sending a particular message later, the sender can claim that the private key is lost or stolen and someone else has forged his or her signature. Another possibility is that some private keys might actually be stolen from X at time T. The opponent can then send a message signed with X's signature and stamped with a time before or equal to T.

4.3 RSA for Digital Signature

The RSA algorithm not only uses for message encryption but also can use to perform the need of digital signature [1, pg 130] [5, pg 312]. The basic concept is shown as follow: (Figure 4.1)

Figure 4.1



Instead of using the public key to encrypt the message, the sender can use his own private key to encrypt the message. The sender performs a hash function using the message as an input and encrypts the digest by his private key. Once the receiver receives the message and the ciphertext, he can perform the hash function by using the message he received. The he can decrypt the ciphertext by using the sender's public key. Since only the proper sender knows his own private key and no other people know it, if the receiver is able to decrypt the ciphertext then he can get the digest from the ciphertext. Comparing the digest created by the receiver

and the digest from the ciphertext. if both digest are the same. then the receiver can assume the message created by the proper sender.

4.4 Digital Signature Standard

The digital signature standard (DSS) [5, pg 312] was introduced by the National Institute of Standards and Technology (NIST) in 1991. known as "Federal Information Processing Standard FIPS PUB 186". The DSS makes use of the Secure Hash Algorithm (SHA) and the digital signature algorithm (DSA). The DSS was originally proposed in 1991 and revised in 1993 in response to public feedback. A further minor revision occurred in 1996. The DSS uses an algorithm that is designed only to provide the digital signature function. Unlike RSA. the DSS cannot be used for encryption or key exchange.

Figure 4.2

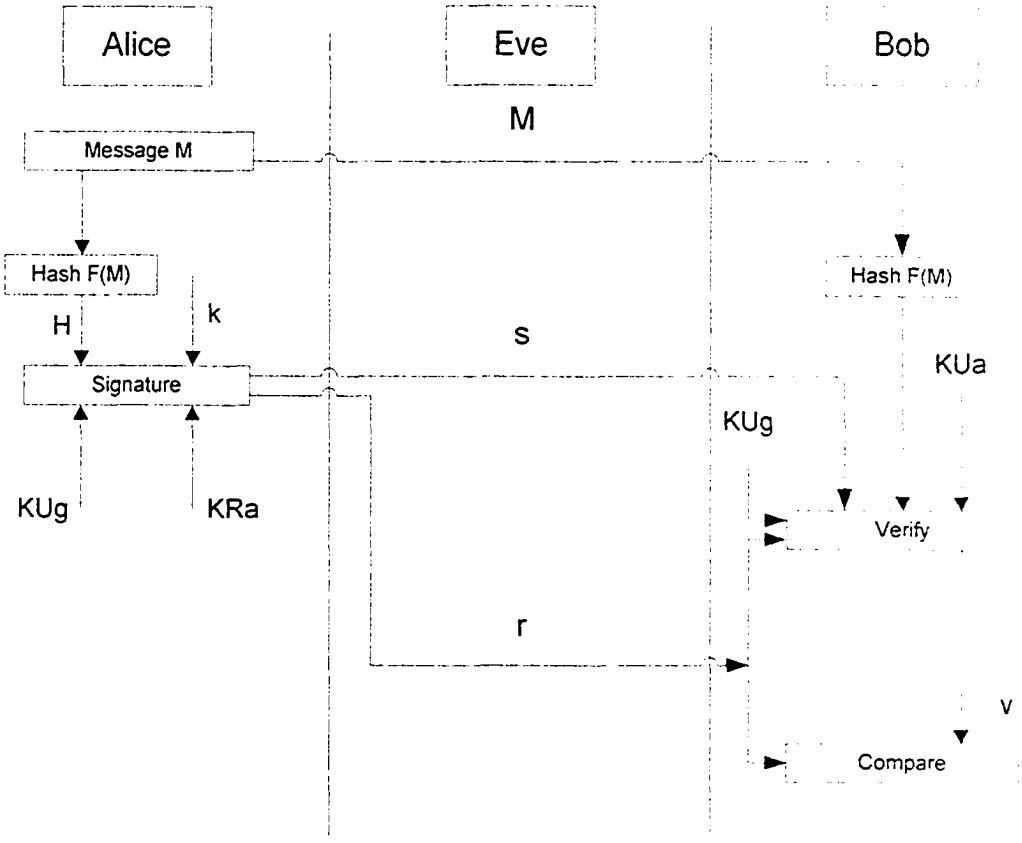


Figure 4.2 shows the DSS approach and figure 4.1 shows the RSA approach for

generating digital signature. In the RSA approach, the message to be signed is an input to a hash function that produces a secure hash code of fixed length. The hash code is then encrypted with Alice's private key to form the digital signature. Once Alice has the message and the digital signature, she will transmit both message and the digital signature to Bob. Bob takes the message and produces a hash code. Bob also decrypts the digital signature by using Alice's public key. If the calculated hash code matches the decrypted signature, the signature is accepted as valid. Since only Alice knows her private key, Bob can verify that only Alice could have produced a valid signature.

The DSS approach also makes use of a hash function. The hash code is provided as an input to a signature function along with a random number k generated for this particular signature. The signature function also depends on the sender's private key (*KRa*) and a set of parameters known to a group of communicating principals. We can consider this set as a global public key (*KUg*). The result is a signature consisting of two components, named as r and s.

At the receiving end, the hash code of the incoming message is generated. The hash code plus the signature are input to a verification function. The verification function also depends on the global public key as well as the sender's public key (*KUa*). The output of the verification function is a value v that is equal to the signature component r if the signature is valid. Same as RSA, only the sender with knowledge of the private key could have produced the valid signature.

4.5 Digital Signature Algorithm

The digital signature algorithm [5, pg 313-315] is based on the difficulty of computing discrete logarithms. There are three parameters that are public and can be common to a group of users, which is KUg. Two prime numbers p and q, such that q divides $p-1$, and g is chosen to be of the form

$$g = h^{\frac{p-1}{q}} \bmod p,$$

Where h is an integer between 1 and $p-1$. The length of these numbers is somehow restricted. The prime number q must be 160-bit long and p is selected with a length between 512 and 1024 bits. Also, number g must be

greater than 1.

Once we have these numbers set up, Alice can select a private key and generates a public key. The private key must be a number from 1 to $q-1$ and should be chosen randomly. The public key is calculated from the private key as $y = g^x \bmod p$. As before, calculating y given x is easy but is computationally infeasible to determine x from y.

To create a signature, Alice calculates two quantities, r and s, which are the functions of the public key components $(p \quad q \quad g)$, Alice's private key x, the hash code of the message, $H(M)$, and an additional integer k that should be generated randomly and be unique for each signing. The integer k and the public key components should also satisfy the following condition: For every number n between 1 and $p-1$ inclusive, there is a power k of g such that $n = g^k \bmod p$.
The quantity r is calculated by the following operation:

$$r = (g^k \bmod p) \bmod q$$

And the quantity s is calculated by the following operation:

$$s = [k^{-1}(H(M) + xr)] \bmod q$$

Once Alice creates the signature, she can send the message M and the signature $(r \quad s)$ to Bob.

At the receiving end, verification is performed using the following formula in the following order. Assume Bob gets the signature $(r' \quad s')$ and message M':

First, find $w = (s')^{-1} \bmod q$
$u1 = [H(M')w] \bmod q$
$u2 = ((r')w) \bmod q$
$v = [(g^{u1})(y^{u2}) \bmod p] \bmod q$
Check $v = r'$

Bob generates a quantity v that is a function of the public key components.

the sender's public key, and the hash code of the incoming message. If the quantity v matches the r component of the signature, then the signature is validated.

The structure of the algorithm is quite interesting and different from RSA. Note that the test at the end is on the value r, which does not depend on the message at all. Instead, r is a function of k and the three global key components. The multiplicative inverse of $k \bmod p$ is passed to a function that also has the message hash code and the user's private key as the inputs. The signature created by RSA may only depend on the public key of the user and the hash code. The structure of the DSA is that the receiver can recover r using the incoming message, signature, the public key of the user, and the global public key which involved more parameter than RSA. The DSA has one more advantage which is the signature r does not depend on the message to be signed. It can be computed ahead of time and be used to sign documents later.

4.6 Arbitrated Digital Signature

The solution for the direct digital signatures can be used as an arbiter. Like direct signature schemes, there is a variety of arbitrated signature schemes. In general terms, they all operate as follows: Every signed message from Alice to Bob should go to a trusted arbiter A first. An arbiter subjects the message and its signature to a number of tests to check its origin and content. The message then is dated and is sent to Bob with an indication that it has been verified to the satisfaction of the arbiter. The presence of an arbiter solves the problem which we have indicated in the last section. The arbiter plays a sensitive and crucial role in this scheme and all parties must have a great deal of trust that the arbitration mechanism is working properly.

For the public key encryption scheme, the arbiter does not need to see the message to check the digital signature. What Alice should do is to send the identifier of Alice, and the ciphertext CN which is encrypted by Alice's private key, to the arbiter. The ciphertext CN contains the identifier of Alice and the double encrypted ciphertext M, which is encrypted by Alice's private key and Bob's public key. After the arbiter gets the message from Alice, an arbiter can decrypt the ciphertext CN to make sure the public or private key is still valid. An arbiter can compare the identifier of Alice is the same as the one with the message to confirm

the message is made from Alice. An arbiter confirms the sender is Alice, and sends the identifier of Alice, timestamp, and the double encryption ciphertext M to Bob encrypted with arbiter's private key [5, pg 301]. (Shown in figure 4.3)

Definition:

Pa = Alice's public key.

Sa = Alice's private key.

Pb = Bob's public key.

Sb = Bob's private key.

PA = Arbiter's public key.

SA = Arbiter's private key.

M = The original message from Alice.

T = Timestamp created by Arbiter.

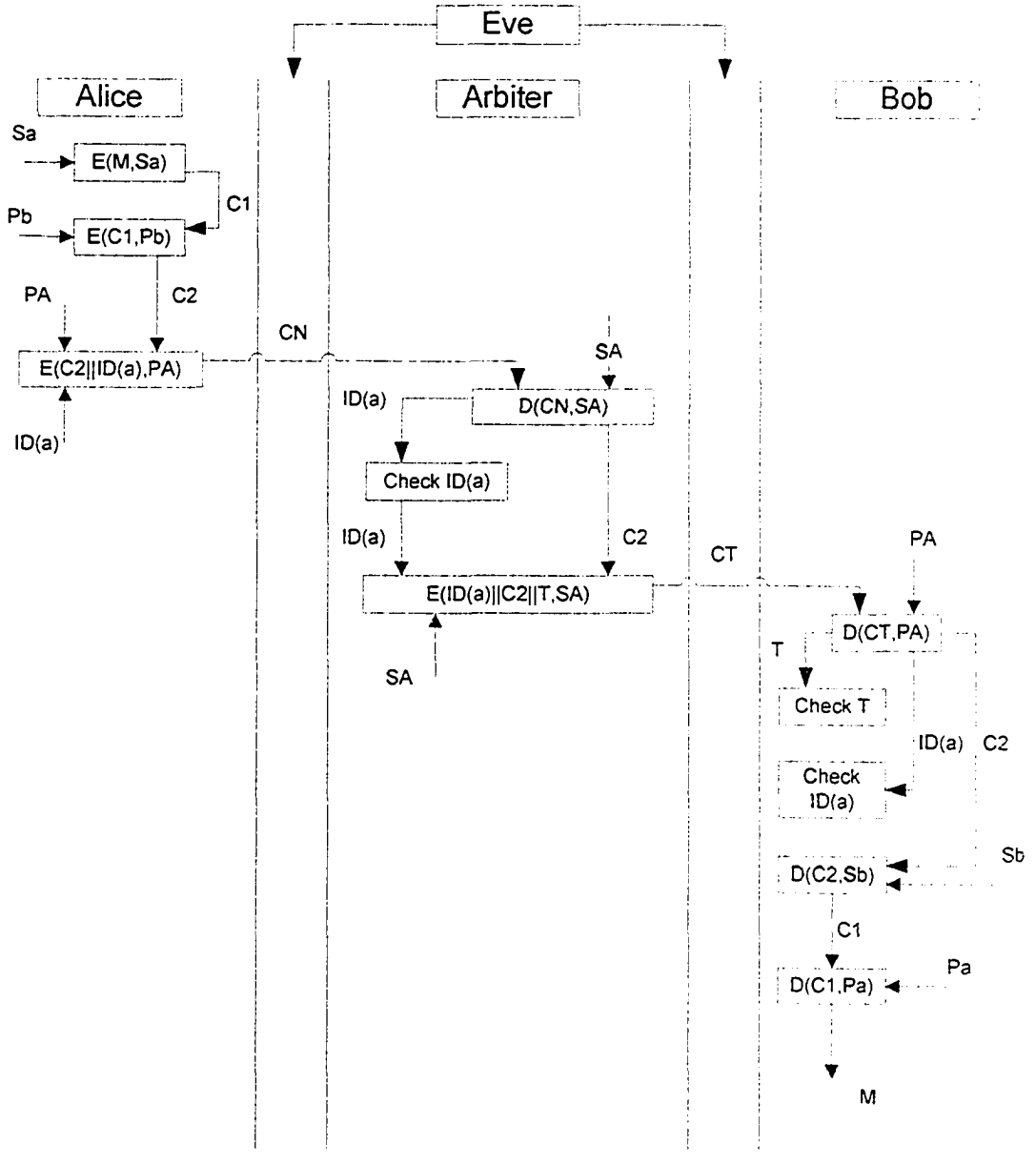
CN = The ciphertext with sender's identification.

CT = The ciphertext with arbiter's timestamp and the sender's identification.

$E(x, y)$ = The encryption function which has the same properties like in the previous chapter.

$D(x, y)$ = The decryption function which has the same properties like in the previous chapter.

Figure 4.3



This scheme has a number of advantages over the direct digital signature. First, the information is shared by the sender and the receiver parties like before. Second, no incorrect dated message can be sent, even if Alice's private key is compromised, assuming that arbiter's private key is not compromised.

Chapter 5: The Next Generation of Algorithm

5.1 Introduction

In the last chapter we have introduced the modern cryptography algorithms and cryptosystems. One of the most common problems in the modern cryptography algorithm is that the algorithms were developed in the early 90's. This mean they are more than 10 years old. If we are still using the same algorithms to protect the network from security problems, soon or later, the algorithms will not meet our requirement as they will be prone to other attacks like brute-force attack. The researchers and developers try to develop some new algorithms by either using the existing hardware or new technologies which may become available in next few years. In order to increase security, there are two options which may be tried. The first method is to improve the existing algorithms by using modern technology. The second method is to use some new technologies in conjunction with new algorithms. Both methods are workable and there are some examples showing that both attempts have enjoyed some degree of success. In this chapter, we will examine these examples and the cryptography algorithms for both attempts.

First, we will introduce the DNA computing and the cryptography algorithms which are built by current computer technology. Second, we will introduce the cryptography algorithms for the quantum computer, a next generation computer which has not yet come into existence.

5.2 DNA computing

DNA computing is a new approach for the massive paralliel computation based on groundbreaking work by Leonard Adleman [22]. Leonard Adleman is the one who developed RSA cryptography in early 90 and he used DNA to solve a seven-node Hamiltonian path problem. The seven-node Hamiltonian path problem is a special case of an NP-complete problem that attempts to visit every node in a graph exactly once. This special case is trivial to solve with a conventional computer or even by hand, but for more nodes in the graph, it will not be easy for

conventional computer. We will discuss more about it later in section 5.3 and see the example about how the DNA computing can solve this special NP problem. Later in section 5.4, we will see what the DNA computing can do for encryption and decryption. The idea come from Ashish Gehani, Thomas LaBean, and John Reif and is the first example of how DNA computing can perform encryption and decryption [16]. The research on DNA computing is ongoing. Hopefully, efficient designs of possible DNA computers will exist in the future.

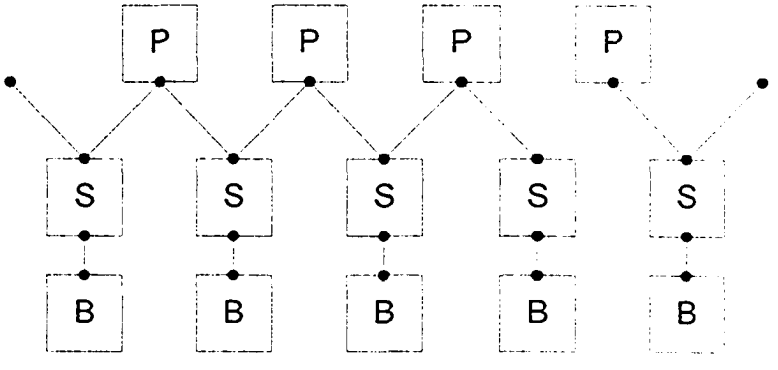
5.2.1 Introduction

To understand how DNA computing works for cryptography, we have to know what is DNA. DNA (Deoxyribonucleic acid) is the molecular basis of genetics. For our purposes, the following features are important [2, pg 241] [20] [23]:

1) A DNA molecule is made of two intertwined, parallel strands which usually called double helix

2) Each strand has the following structure (figure 5.1):

Figure 5.1

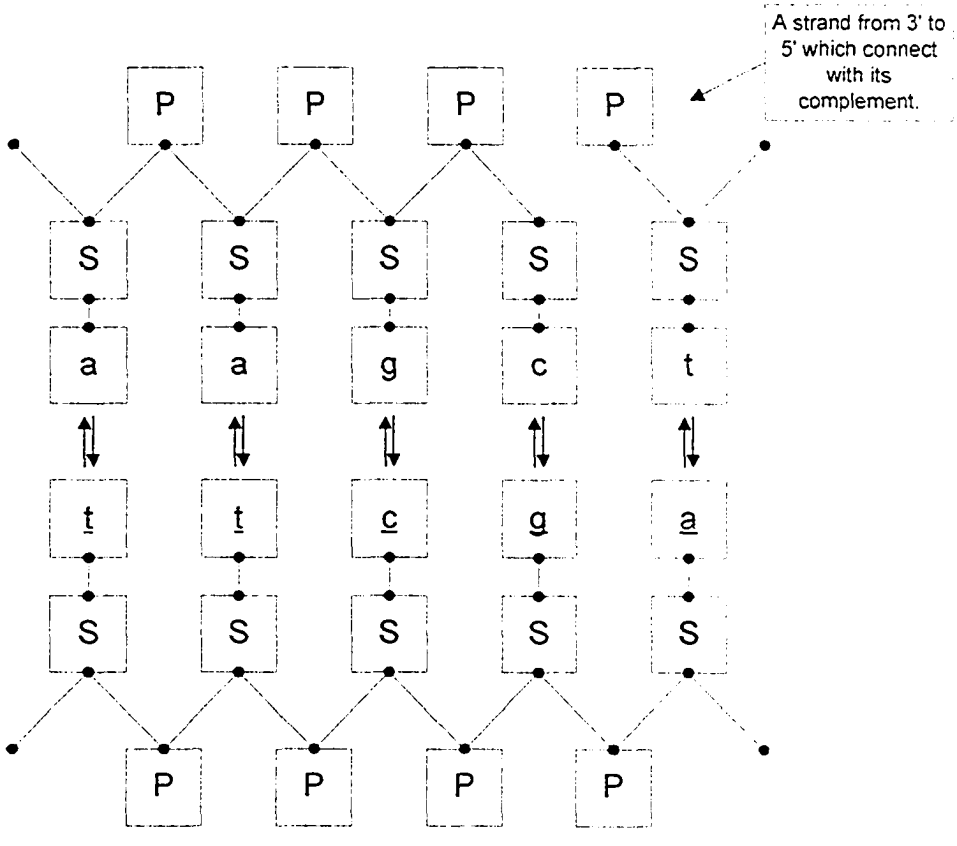


Where p = phosphate, s = sugar, and b = base. Each b can be any one of the four bases: a = adenine, t = thymine, c = cytosine, g = guanine.

3) Two consecutive s-p bonds must occur at distinct places on the s molecule: one at 3' (three-prime) position and one at the 5' (five-prime) position. So each strand has a 3'-end and a 5'-end, and so they are systematically oriented. We will write a, t, c, g when the strand is being read from the 3'-end to the 5'-end, and a, t, c, g when the strand is being read the opposite way, that is, from 5'-end to the 3'-end.

4) The two intertwined strands in a DNA molecule have opposite orientations, and complementary base sequences. a is always complements with t and vice versa. Also, c is always complements with g and vice versa. A typical stretch of the DNA molecule looks like (figure 5.2):

Figure 5.2

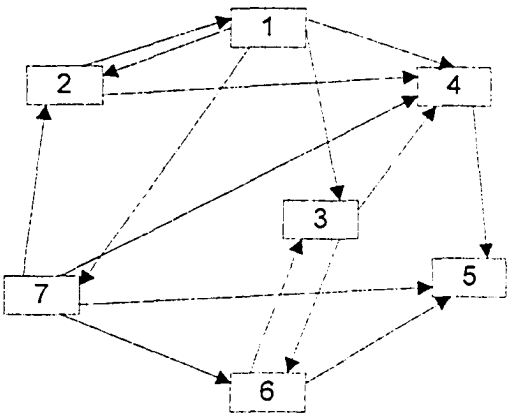


This schematic representation of a segment of a DNA molecule will be abbreviated to:

We name the template strand on the top as: a, a, g, c, t and the complement strand as: t, t, c, g, a.

We now show how DNA can solve the seven-node Hamiltonian Path Problem. The seven-node Hamiltonian Path Problem involves an oriented graph, a collection of vertices and oriented path from one vertex to another vertex. The following graph is one of the examples of seven-node Hamiltonian Path problem (figure 5.3):

Figure 5.3



The problem is this: We designate a Start vertex and a Finish vertex. In these terms, the Hamiltonian path problem for this graph is to find a sequence starting from the Start vertex and ending at the Finish vertex and to visit each vertex in the graph exactly once. In usual graph terminology, such a sequence is called a Hamiltonian Path. For seven-nodes, it may be easy to solve by hand. However, when the number of vertex increase, the amount of computation to find the Hamiltonian Path by using the present efficient algorithm will also increase exponentially. Also, to check if there exists a Hamiltonian Path in the graph it means we have to check if the Path contains all the vertices in the graph. If the numbers of vertices are increasing, the time to check the path will grow at a polynomial rate. The combination of the exponential growth of the list to be searched and polynomial growth of the length of the solution verification algorithm makes this problem an NP problem.

Leonard Adleman is the first researcher who used DNA strands to solve this NP problem [22].

1) Each vertex and the link connecting each vertex will assign a single DNA strand 20 bases long. For example:

Vertex 2: TATCGGATCGGTATATCCGA
Vertex 3: GCTATTCGAGCTTAAAGCTA
Vertex 4: GGCTAGGTACCAGCATGCTT

Link 2→3: GTATATCCGAGCTATTCGAG
Link 3→4: CTTAAAGCTAGGCTAGGTAC
Etc..

Note that Link 2→3 is made of the last half of vertex 2 plus the first half of vertex 3, same for the Link 3→4. Also, Link 4→3 must be different from Link 3→4. So the strand which represents Link 4→3 is totally different from the strand which represents Link 3→4. Instead of using 20 bases, we use 8 bases in our example.

Vertex 2: TATCCCGA
Vertex 3: GCTAAGCT
Vertex 4: GGCTCGTT
Link 2→3: CCGAGCTA
Link 3→4: AGCTGGCT
Etc..

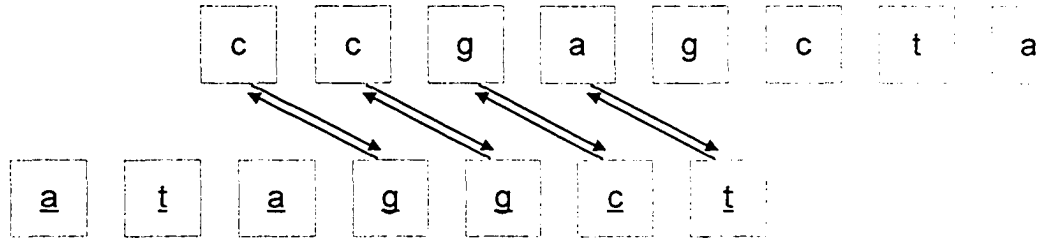
Once we have all the strands representing each vertex and the link connecting each vertex, we need to mix the strands representing the link of connection with its complementary strands representing the vertex in the test-tube. So, for our example, the test-tube holds:

Link 2→3: CCGAGCTA
Link 3→4: AGCTGGCT
Complement of Vertex 2: ATAGGGCT
Complement of Vertex 3: CGATTCGA
Complement of Vertex 4: CCGAGCAA
Etc..

When the contents are mixed, the complementary sections of the strand will bond, yielding products of reaction like (figure 5.4):

C, C, G, A, G, C, T, A
/ / / /
A, T, A, G, G, C, T

Figure 5.4



And

C. C. G. A. G. C. T. A
/ / / / / / / /
A. T. A. G. G. G. C. T. C. G. A. T. T. C. G. A

Etc..

The products of reaction represent all the possible of path that can exist in the graph. By using recombinant DNA technology we can easily eliminate the path which

1) Does not start with the START vertex and does not end with the END vertex
2) Does not contain exactly 7 vertices
3) Contains repeated vertices

If there is anything left, it must be the molecules encoding a path that goes from the START vertex and end with the END vertex and each vertex exists in the path once. This mean there is a Hamiltonian path in the graph.

5.3 DNA cryptography using random one-time-pad

This is the first example of using DNA computing to encrypt and decrypt data. The method is suggested by Ashish Gehani, Thomas LaBean, and John Reif in 2000 [16]. They are using the DNA characteristics to form a one-time-pad codebook as

a key to encrypt and decrypt data. Here is the protocol:
Initial steps:

Alice and Bob want to communicate with each other by using DNA cryptosystem to encrypt and decrypt the data. What they need is a long strand as a one-time codebook that can map a plaintext word to a codeword. The map must be a one-to-one map from each plaintext word to a codeword. For example we have a strand A as our codebook in which each of the four bases represents a plaintext word

Strand A: TTGCAAGTCCGTTTAA

Therefore we will have a unique complement of strand A:

Complement of strand A: AACGTTCAGGCAAATT

Each of the four bases in the complement of strand A represents a codeword. Alice and Bob share this one-time codebook which Eve does not know.

Steps:
1) Alice uses the codebook to map each plaintext word to the DNA strand. Once she finishes and has all the DNA strands, she can form the complement of those strands as a codeword and sends it to Bob.

2) Once Bob gets the codeword, he can use the codebook to form the DNA strands from the codeword and find the complement of those strands to get the plaintext words.

Since Eve does not know the codebook, even she gets the DNA strands and finds the complement of those strands, without the codebook, she cannot get the plaintext from Alice. The reason we only use the codeword once is because Eve can trace the codeword by the brute-force attack or by counting the number of existence for each DNA strands. Also, the codebook must be shared by Alice and Bob secretly in order to make this method work.

5.4 Quantum computing and Quantum cryptography

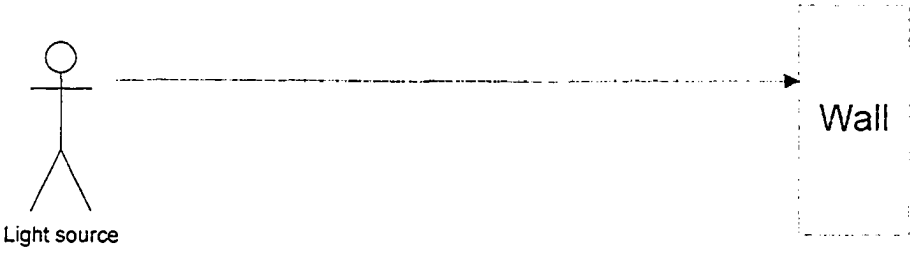
Quantum computing [18] is a new area of research that has only recently started

to blossom. Quantum computing and quantum cryptography were born out of the study of how quantum mechanical principles might be used in performing computations. Quantum computing is totally different from classical computing. In classical computing, a bit has a discrete angle and can represent either a 0 state or a 1 state. In quantum computing, the computer uses a *qubit* instead of a bit where a qubit can be in a linear superposition of the two states. In 1982, the Nobel Laureate Richard Feynamn observed that certain quantum mechanical phenomena could not be simulated efficiently on a classical computer. He suggested that the situation could perhaps be reversed by using quantum mechanics to do computations that are almost impossible on classical computers. At that time, Feynman did not present any examples of such devices.

5.4.1 Introduction

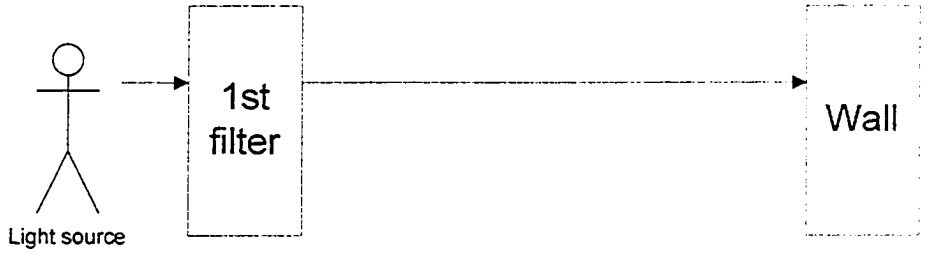
To understand more about quantum computing, we will start with a simple example [6, pg 353] [18]. We know photons are the particles that make up light and are therefore observable. We start with some light source with three Polaroid filters which they have the following polarization: horizontal, 45° , and vertical. Start a light source on a wall as the following figure:

Figure 5.5



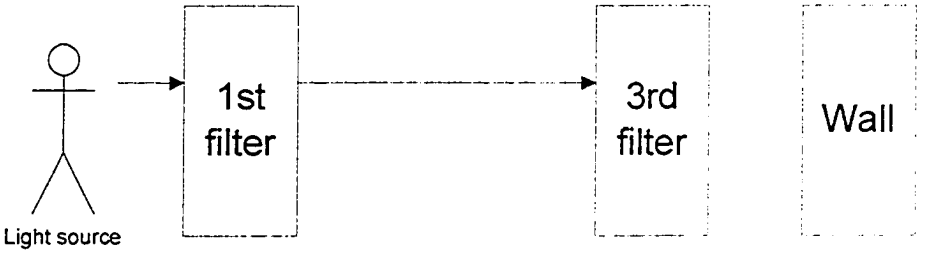
We insert the filter with a horizontal polarization between the light source and the wall as the following figure:

Figure 5.6



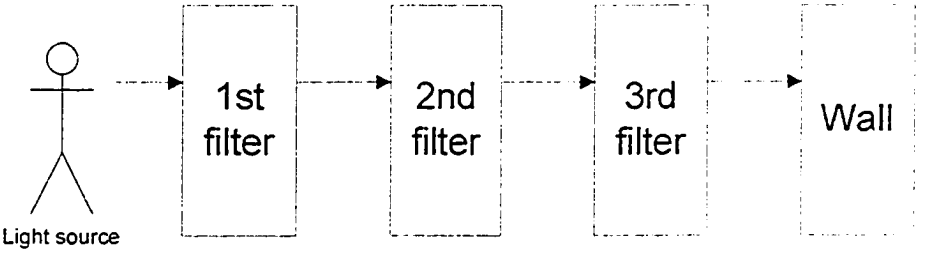
After that all the photons coming out from the filter will have horizontal polarization. Now insert the filter with the vertical polarization as the following figure:

Figure 5.7



Since the last filter has vertical polarization, it filters out all the horizontal polarized photons from the previous filter. Therefore, no photon will past the last filter and no light on the wall. The last step we do is to insert the filter with 45° polarization in between two filters as the following figure:

Figure 5.8



After you insert the last filter, we will notice that there is a light on the wall. What happened in the last example? In order to explain the last example, we will represent a photon's polarization with a unit vector in the two dimensional complex vector space. This vector space has a dot product given by $(a,b) \cdot (c,d) = ac + bd$.

where c,d denotes the complex conjugates of c and d. Choose a basis for this vector space, which we can denote as↑ and →. Therefore, an arbitrary polarization may be represented as a ↑ + b → . We could just have well chosen a different orthogonal basis, for example, one corresponding to a 45° rotation: ↖ and ↗. Similarly, we measure a vertical aligned photon with respect to a 45° filter. Since

$$\uparrow = 2^{-2}\nwarrow + 2^{-2}\nearrow.$$

the probability that the photon ↑ passes through the 45 polarization filter is $(2^{-2})^2 = \frac{1}{2}$. Similarly, the probability that it does not pass through the filter is also $\frac{1}{2}$. When the original light is emitted with random polarization a ↑ + b → , only half of the photons being emitted will pass through the → filter, and all of these photons will have their state changed to →. We place the ↑ filter after the → filter, the photons which in → state will be stopped. When we insert the 45 polarization filter in the middle of ↑ filter and → filter, it corresponds to the measure with respect to ↗. and hence those photons with → polarity will come out as ↗ polarity with probability of $\frac{1}{2}$. Therefore, there has been a 4:1 reduction in the amount of photons passing through up to the ↗ filter. Now the ↗ photons pass through the ↑ filter with probability of $\frac{1}{2}$, so the total intensity of light arriving at the wall after the last filter will be 8:1 reduction.

5.5 Quantum Key Distribution

Once we know the ideas behind quantum mechanics, we can use them to exchange information between two parties through a quantum channel with some qubits. As we know previously, if a photon's polarization is read in the same basis twice, the polarization will be read correctly and will remain unchanged. If it is read in different basis, only $\frac{1}{2}$ chance the polarization will be read correctly. Combine the number of chances to choose a different basis, and it will become $\frac{1}{4}$ chance the original polarization will be read correctly because in different basis there are 2 choices. The following protocol can be used by Alice and Bob to exchange secret keys [6, pg 357] [19]:

Initial steps:

1) Alice sends Bob a stream of photons, each with a random polarization, in a random basis. She records the polarizations. For example, she sends
The bits : 0,1,1,1,0,0,1,0
The basis 1, B1: 0 = ↑, 1 = →
The basis 2, B2: 0 = ↖, 1 = ↗
So the stream will become: ↑↗→→↖↖→↖ with the basis she chooses randomly: B1, B2, B1, B1, B2, B2, B1, and B2.

2) Bob measures each photon in a randomly chosen basis and records the results. In our example, he measures it with the basis: B2,B2,B2,B1,B2,B1,B1,B2

3) Bob announces over an authenticated channel, the basis he uses for each photon. So now Alice should know which basis Bob has chosen.

4) Alice tells him which choices of bases are correct. So in this example, both Alice and Bob will share the 2nd, 4th, 5th, 7th, and 8th bases because the basis made from Bob matches her choices.
The bits: 1, 1, 0, 1, 0 with bases B2, B1, B2, B1, B2.
The qubits: ↗→↖→↖

Protocol:

1) Alice sends a message to Bob encrypted by the key 1, 1, 0, 1, 0

2) Since Bob has the same key, he should be able to decrypt the ciphertext after he receives it.

If the key is long enough, Eve can read the ciphertext only if she knows the key. The security behind quantum key distribution is based upon the laws of quantum mechanics. Assume Eve is eavesdropping between Alice and Bob when they exchange the key. Eve may measure the photon with different basis; therefore Bob will have a 1/4 chance to get the correct polarization from the photon after he measures from Eve. If the steam is long enough, Alice and Bob should find out

they are having different values for their secret keys, then they can detect the presence of eavesdropping, and start the protocol over.

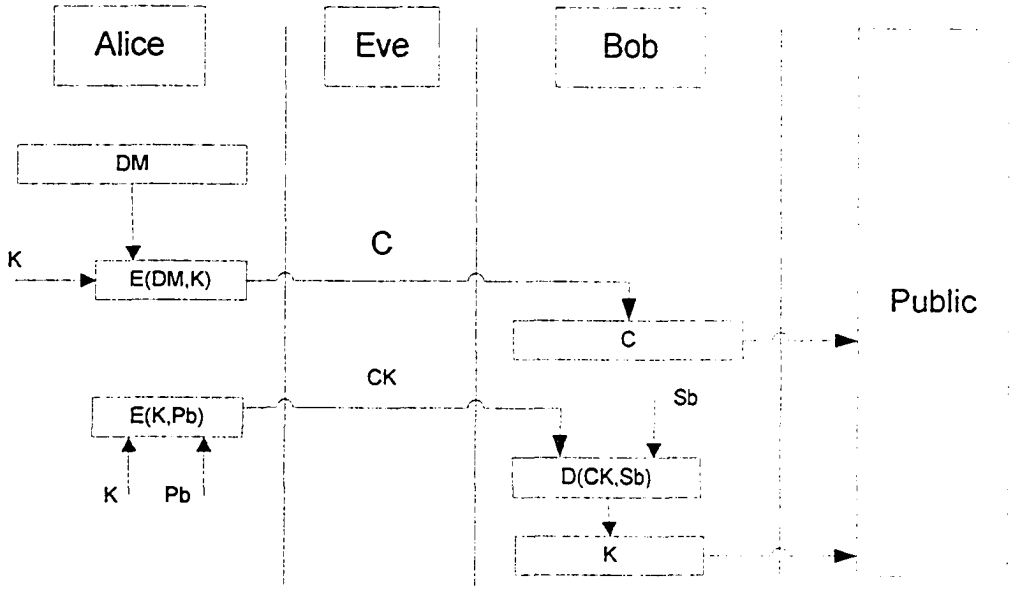
Chapter 6: Digital Rights Management (DRM)

After year 2000, most of the documents, pictures, or music is produced as digital files. It is not easy to prevent copying or sharing on the digital file because once the user owns the digital file either from the store (music from a CD, image from a DVD, or documents from a textbook) or downloads it from the internet (Napster), it is hard to control how users share it with other users or copy it to resell. From the number of users using the Napster, we can tell that many people on the internet try to share and copy the same digital file. We need to have a management of rights for the digital file. The rights indicates which group of users has the rights to operate some actions on the digital file for some certain time while which group of user has no rights to access it all the time. We will first introduce some of the requirements for DRM in the next section, and then we will provide an example to understand how the DRM works.

6.1 Introduction

The digital rights management is developed to prevent the user to copy or share the digital file against the copyrights. DRM involves the description, identification, trading, protection, monitoring and tracking of all forms of rights usage over both tangible and intangible assets (physical and digital form) including management of right holder relationship [26, pg 2]. Usually DRM consists only of the identification of intellectual property and the enforcement of usage restrictions. The identification consists of the attribution of an identifier (such as the ISBN numbers for books) and the marking of the property with a sign (such as a watermark). The enforcement works via encryption, which can ensure the digital content is only used for the purposes agreed by the rights holder. Encryption system cannot prevent the copying and sharing. Consider the following scenario: Assume Alice is the author of the digital file, and Bob is the one who requests the digital file. If Alice only uses the encryption system to encrypt the digital file and sends the ciphertext to Bob along with the key, then Bob is not only able to decrypt the ciphertext to get the digital file, but also can share the ciphertext and key with other users. Shown in figure 6.1

Figure 6.1



In the last scenario, we can see the encryption system cannot prevent the copying and sharing, but the encryption system can assist the DRM. Basically, the DRM works as following:

1) Alice encrypts the digital file and sends it to the trusted third party T (Trusted Authority).

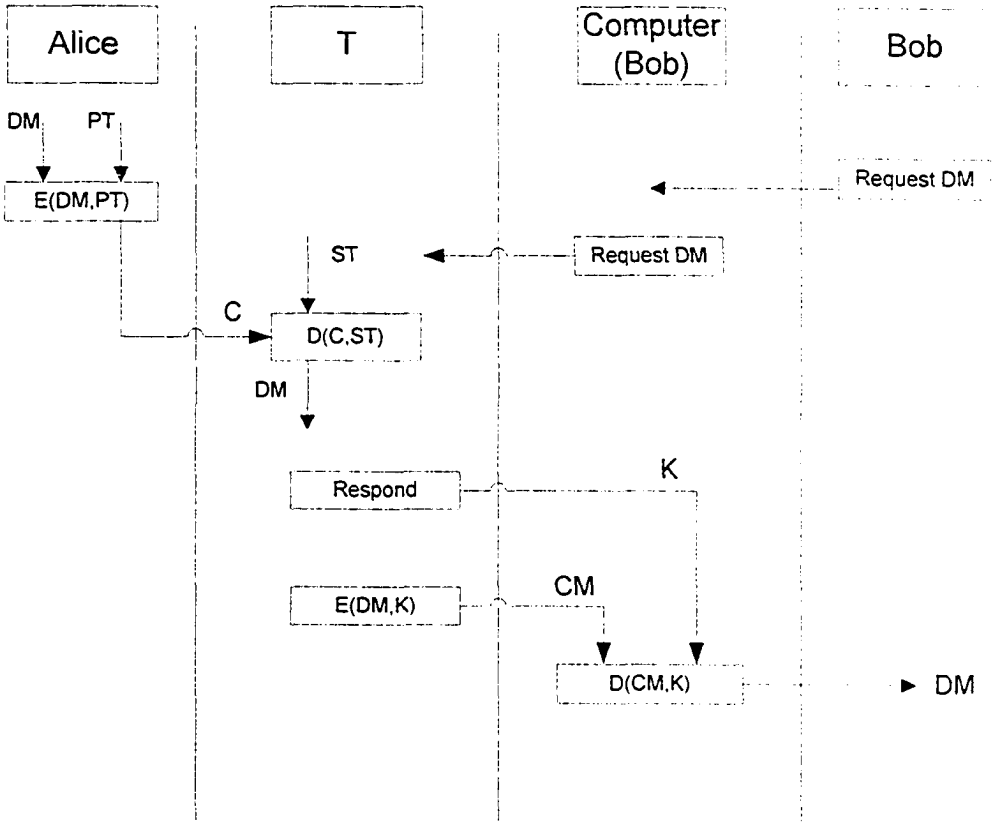
2) Bob requests the digital file from Alice. Bob needs to download some software from T which may contain the unique key that Bob cannot view the key or share with other users.

3) After Bob installs the software into his computer, the third party T will send the ciphertext to Bob's computer.

4) The software in Bob's computer will decrypt the ciphertext and view the digital file to Bob. Since Bob cannot access the key, he cannot share the digital file with others.

(Shown in figure 6.2)

Figure 6.2



The DRM has some disadvantages, because [93]:

1) The encryption/decryption system remains vulnerable.
2) The software may not be able to install into the client computer
3) The verification from the client to the server may require the internet connection, which limits the number of users.

There are so many different approaches from different companies (Microsoft, Adobe, etc) to achieve the purpose of DRM. In the next section, we will provide the example to achieve the purpose of DRM by using elliptic curve algorithm.

6.2 Example of DRM

As before, assume Alice is the rights holder of the digital file and Bob is the one who requests that digital file from Alice. The following is the protocol which

Alice can send the digital file to Bob and Bob does not copy or share with other users (Shown in figure 6.3):

Definition:

P = A prime number.

E = An elliptic curve.

DM = The digital file from Alice.

Pa = Alice's public key.

Sa = Alice's private key.

Pb = Bob's public key.

Sb = Bob's private key.

PT = Arbiter T's public key.

ST = Arbiter T's private key.

S = The software to install to Bob's computer. The software contains few keys to unlock the ciphertext. Bob cannot view the key and share the key with other user.

Na = Key to unlock the software S.

Nb = Key contained in software S.

P = Payment.

$E(m,x) = C$

= Any public key encryption function.

$D(C,y) = m$

= The related public key decryption function.

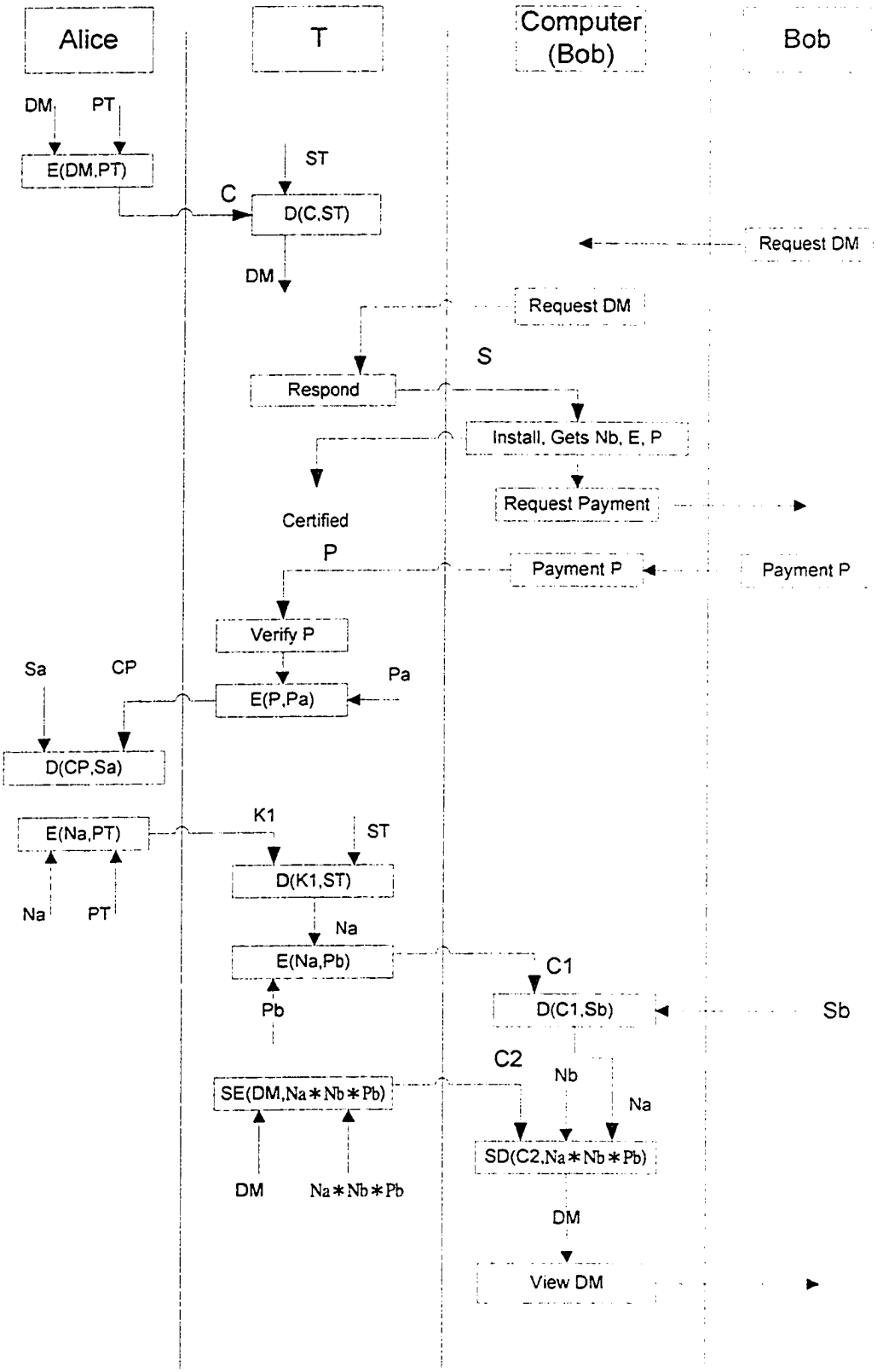
$SE(DM,x) = Cs$

= The symmetric key encryption function. The ciphertext can be created by the arbiter T only. Also, the ciphertext can decrypted by the software S.

$SD(Cs,x) = DM$

= The symmetric key decryption function which is embedded in the software S.

Figure 6.3



Initialization steps:

1) Trusted authority T is between Alice and Bob, which contains the elliptic curve E with some prime numbers P.

2) T may also contain the public key PT which Alice and Bob can use to encrypt their data. Of course, T will have the private key ST which is related to PT for decryption. No one knows the ST except T.

As before, Alice has her own public key Pa, and private key Sa. Bob also has his own public key Pb, and private key Sb.

Protocols:

1) Alice encrypts the digital file DM with trusted authority T's public key PT and sends it to T: $E(DM, PT) = C$.

2) Bob sends the request to T. T needs Bob to download some software S which contain the unique key Nb. At this point, we assume Bob cannot copy, access, view, or share Nb with other users. Bob needs to install the software S into his PC to become a certified client.

After the installation, the software S connects to T to indicate that Bob is able to install the software properly.

3) The software S requests a payment from Bob.

4) Bob makes a payment P and sends it to T. T verifies P and sends it to Alice: $E(P, Pa) = CP$

5) Once Alice accepts the payment from Bob, she sends the key Na to T: $E(Na, PT) = K1$

6) T gets the key Na from Alice and encrypts the digital file by using combination key: $SE(DM, Na * Nb * Pb) = C1$
Also, T encrypts the key Na by Pb: $E(Na, Pb) = C2$

7) T sends $C1$ and $C2$ to Bob's computer. The software S asks Bob to input his own private key Sb in order to verify himself.

8) If Bob inputs the correct private key Sb, then the software will be able to decrypt $C2$ to get Na. The software S does not need to show the key Na to Bob, instead, S combines the Na with Nb and Pb to make the key $Na * Nb * Pb$ to decrypt $C1$ to get DM:

$SD(C1, Na * Nb * Pb) = DM$

If Bob wants to copy or share DM to other users, then Bob needs to operate the software to get DM. Depending on the software or the rights to access, it may not be easy to make a copy of DM. If Bob copies or shares the ciphertext $C1$ or $C2$ with other users, then the next user needs to know Sb in order to decrypt $C1$ to get DM. Since Nb is uniquely installed in Bob's PC, Bob cannot access the software to get Nb. This is really hard for other users to crack $C1$ to get DM. Assume Eve tries to eavesdrop and get DM without Alice or Bob noticing it. The first attempt from Eve is to make a copy C and get DM from C. We know this is almost impossible to crack C if C is encrypted by some modern encryption systems likes RSA. The second attempt from Eve is to make a copy $C1$ and get DM from $C1$. Again, this is almost impossible for her because if she needs to crack the $C1$, then she needs to know Na, Nb, and Pb. She can get the Pb easily because Pb is a public key. However, it is not easy to get Nb because Nb is installed in Bob's computer. If Eve wants to get Nb, then she needs to hack to Bob's computer and crack the software to get it. Also, if Eve wants to get Na, she needs to crack $C2$ by using Sb. Again, that is almost impossible for her to get Na as well.

6.3 Watermarking of DRM

Watermarking is the insertion of "hidden" data such as copyright information into visible data such as an image or a video. There are various kinds of watermarks, the usage or the watermarks are various. The following considerations apply on choosing different watermarks [17]:

1) The purpose of the embedded data.
2) Whether it is the same for each instance of a given content item.
3) Whether one or both of the signals are analog vs. digital.
4) How subtly the data is embedded.
5) How perceptible the data is.

6) Whether the watermark is intended to survive manipulation of the marked file.

It is important to understand that a watermark is not encryption. A watermark modifies data but cannot prevent or enable playback of the data by itself, except in the special case where playback is restricted to proprietary closed boxes which insist on seeing the watermark.

Watermark schemes fall into three classes [17]:

1) Forensic watermark: Watermarks do not actually stop anyone from copying or manipulating content, but they can establish where the content came from originally.

2) Denial watermark: A watermarks aimed to prevent content from being accessed fraudulently.

3) Mulit-phase watermarking: A watermarking schemes usually involve a change of states in the content. In the initial state, the content is in a sample from which may or may not be easily usable. Then a consumer legitimately acquires the content and it is transformed into a form which is more usable.

If the content has a forensic watermark, then it is only intend to track copies of the data but not able to prevent the copying. Mostly, the forensic watermark has be found in the image files such as JPEG file. The denial watermarking is a mix of encryption and the information of copyright. This watermarking has been found in a physical audio formats music CD or video disk such as VCD or DVD. The main scheme to make this watermarking work is that the closed boxes able to play the music CD, VCD, or DVD need to match the watermark on the disk. If the watermark is not matched, then the closed boxes will terminate the operation. The multi-phase watermarking schemes are just emerging as of spring 2003 and are still under development. The main idea of mulit-phase watermarking is that the content exists in one from as originally distributed, and transformed to the second form once legitimately licensed. The mulit-phase watermark is hard to find for now in the digital data because these schemes are really complex. If the content is cracked at the first phase, the features of the second phase never come into play. That is the main reason why this scheme is still under development unless there is a way to ensure the security of the first phase.

Chapter 7: Conclusion

The development of cryptography helps protect the communication between the sender and the receiver. The protocols and the algorithms we have studied in this thesis are only for one-to-one communication. We have studied the aspect of the network security, possible attacks which can occur during the communication, and the algorithms which solve the corresponding problems. Mostly, network security involved four important services to prevent internal and external attack. The four services are confidentiality, integrity, authentication, and non-repudiation. In chapter 1, we have studied the aspect of confidentiality and the algorithms which can meet the needs of confidentiality. In chapter 2, we have studied aspect of the integrity and the hash/MAC functions. In chapter 3, we have studied the aspect of authentication and the requirement to achieve it. In chapter 4, we have studied the aspect of non-repudiation and the solutions to associated problems. In chapter 5, we have studied the new approaches for the next generation of cryptograph algorithms, which include DNA computing and Quantum computing. Although these algorithms may never be implemented as they are, they do advance the concept of key-based cryptography. Some of the algorithms are extensions of existing algorithms, such as DNA computing, while others are developed from new approaches, such as quantum computing. In chapter 6, we have studied the digital right management, which is designed to manage digital files. This idea is developed to address many users sharing digital files over the Internet. Therefore, this research area is relatively new and still under development. Nowadays, the attempted solutions are still not sufficient for preventing all possible attacks. Also, many restrictions apply in trying to achieve the requirements, such as Internet connection between the client and the authority.

In the future, I believe that having an authority between the sender and the receiver is still the best solution for protecting both sides. The authority can be a person or a computer. The one-to-one communication will not only include the sender and the receiver. Instead, we are more concerned with the communication between the sender and the authority or the authority and the receiver. From the DRM example, we can see the trusted authority party is a must-have requirement in order to protect both sides and the digital file. The communication between the

sender and the receiver will become more complex once we involve an authority. However, this should be able to provide a better solution to network security.

Another possible improvement in the future is that the public key or private key may be embedded into the computer hardware or software. The DRM example shows that the software may include the key for decoding but not to the user. The next generation of cryptography algorithm should be able to operate faster than before even with keys of the same size, against any attacks. They should also be easy to integrate into the software or hardware.

Finally, the computer hardware or software may embed multiple versions of encryption and decryption functions. Again, as the DRM example shows, the use of one encryption and decryption function cannot protect both sides. Therefore, we need two different algorithm's encryption and decryption functions in order to satisfy all the possible requirements for the DRM. Under different constraints, we may need different algorithms to prevent the problems and achieve the requirements.

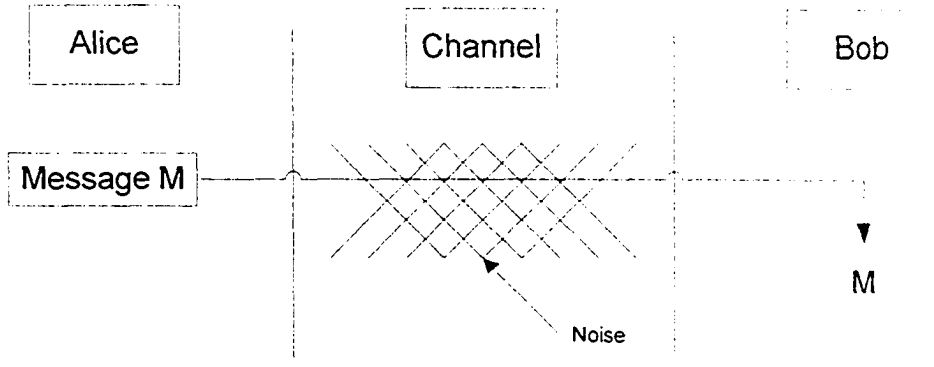
Appendix I: The McEliece Algorithm

A.1 Introduction

In a normal communication cannel, changing one bit in the ciphertext can change enough bits in the corresponding plaintext to make it unreadable or unmatchable. Therefore, we need a way to detect or correct errors that could occur when the ciphertext is transmitted. Therefore, we need a method which provides an ability to correct the error when the error occurs in the channel.

Before we start to explain any of the error correcting methods, let us see what the problem between Alice and Bob is. Assume Alice wants to send a message to Bob through a *noisy* channel. Alice will "encode" the message in order to obtain the *codewords* consisting of sequences of *symbols*. After the transmission, Bob may get the sequences of symbols which are not the same as what Alice sends. In this case, Bob needs to "decode" the sequences of symbols in order to get the correct codewords. "Decode" means to correct the error and get the same codeword as before it is sent. Shown in figure A.1

Figure A.1



For now, the codeword is represented by a sequence of binary numbers {0,1} *symbols*, which is called a *binary code*. If the code uses sequences of 3 symbols,

we often call it a *ternary* code. In general, a code that uses an alphabet F_q consisting of q distinct symbols is called a $q-ary$ code. Also, we denote $(F_q)^n$ as a nonempty set of $n-tuples$ elements $a = a_1 a_2 .... a_n$ where each $a_i \in F_q$. For example, $(F_2)^3$ contains a vector: (000,001,010,011,100,101,110,111). One of the important definitions is the Hamming distance $d(c)$ because the hamming distance of a code c can tell us how many errors can be corrected for code c.

Definition:
Let x and y be any vector in c. The Hamming distance $d(c)$ is the number of symbols that disagree in x and y.

A code c can correct up to t errors if $d(c) \geq 2*t+1$. We call any code c a (n,M,d) code, meaning the code c has a length n, M codewords, and with minimum distance between codewords of $d = d(c)$. Sometime, we will say code c is perfect code if every vector in $(F_q)^n$ is at distance $\leqq t$ from exactly one codeword. For example, the binary repetition code:

$$\begin{cases} 000.....0 \\ 111.....1 \end{cases}$$

Of length n, where n is odd, is a perfect $(n,2,n)-code$.

A.2 Linear Code

A linear code [3, pg 55-78] [6, pg 311] is one of the methods which can be used to correct errors on codewords. If the linear code has a dimension k and length n over a field F, we call it a $[n,k]$ code. Usually, the linear code is generated by a generating matrix $G = [I(k), P]$, where $I(k)$ is identity matrix with k rows and columns, and P is a $k \times (n-k)$ matrix. The rows of G are the basis for a k-dimensional subspace of the space of all vectors of length n. It is named linear code because any codeword v in C is a linear combination of vectors v from

the generating matrix G. Suppose we have $G=[I(k).P]$ as the generating matrix for a code C. Let $H=[-P^T,I(n-k)]$ be the matrix that is used to correct errors, where $-P^T$ is the transpose of P. In general, the matrix H is called a parity check matrix for G if H has the property that a vector $v\in F^n$ is in G if and only if $v*H^T=0$. Assume there are some errors that occur after the transmission. The codeword $r=v+e$ where v is the codeword in c and e is the vector which tells us which position the error is occurred. So

$$r*H^T=(v+e)*H^T=v*H^T+e*H^T=e*H^T$$

We say e is the *coset leader* while $e*H^T$ is called the *syndrome*. One of the helpful lemmas is that two vectors u and v belong to the same coset if and only if they have the same syndrome. Therefore, we can correct the error by subtracting the coset to get the correct codeword.

Let us start with some examples. Assume Alice wants to send a message to Bob through a noisy channel.

Initialization steps:

1) Alice and Bob need to agree on the same generating matrix G.

Let c be the binary linear [4,2] code with a generator G

$$G=\begin{pmatrix}1&0&1&1\\0&1&1&0\end{pmatrix}$$

2) Alice will calculate the codeword from G and use it to represent the message
In this case, the codeword is $(0\ 0\ 0\ 0),(1\ 0\ 1\ 1),(0\ 1\ 1\ 0),(1\ 1\ 0\ 1)$

3) Alice can calculate the syndrome to see how many errors the codeword can correct by picking the coset leader.

If the coset leader is $(0\ 0\ 0\ 0)$, then the syndrome is $(0,0)$
If the coset leader is $(1\ 0\ 0\ 0)$, then the syndrome is $(1,1)$
If the coset leader is $(0\ 1\ 0\ 0)$, then the syndrome is $(1,0)$
If the coset leader is $(0\ 0\ 0\ 1)$, then the syndrome is $(0,1)$

Protocol:

1) Alice sends the codeword through the channel to Bob.

Assume Alice sends a codeword $(1 \quad 0 \quad 1 \quad 1)$ to Bob.

2) Bob gets the codeword but he is not sure if it is the codeword which Alice sends. He can try to decode the codeword and get the correct codeword.

Assume Bob gets the codeword $(0 \quad 0 \quad 1 \quad 1)$. He can first use the matrix H to calculate the syndrome.

$$(0 \quad 0 \quad 1 \quad 1)\bullet\begin{pmatrix}1 & 1\\ 1 & 0\\ 1 & 0\\ 0 & 1\end{pmatrix}=(1,1)$$

This is the syndrome for the second row. Subtract the coset leader

$$(0 \quad 0 \quad 1 \quad 1)-(1 \quad 0 \quad 0 \quad 0)=(0 \quad 0 \quad 1 \quad 1)+(1 \quad 0 \quad 0 \quad 0)=(1 \quad 0 \quad 1 \quad 1)$$

Because we work on the binary field: $1+1=2=0(\mathrm{mod}\,2)$. Therefore, $-1=1(\mathrm{mod}\,2)$ for the binary field.

The example shows some disadvantages. The first one is that it cannot correct two errors. The second one is that if the error occurs at the third position, then the syndrome cannot correct the error. For a linear code, in order to correct more errors of course we need the larger hamming distance $d(c)$. Also, large $d(c)$ can help us to have more syndromes to represent more coset leaders then we can correct the errors accurately as well.

Appendix II: McEliece Algorithm

There are so many cryptographic systems that are based on number theoretic principles and other algorithms that are based on other complex problems. The McEliece algorithm [13] is based on the difficulty of finding the nearest codeword for a linear binary code. The idea of this algorithm is that with a binary string of length 1024 bits, and 50 bits is error, there are ${}_{1024}C_{50} = 3\times10^{85}$ possible locations for these errors; therefore to search these locations by trying all possible combinations is nearly infeasible. The algorithm itself is not vulnerable but

because the ciphertext and the public key contain so many 1's and 0's. Therefore, the algorithm is not effective against brute force attacks. The reason we introduce this algorithm is to compare other cryptography algorithm, like RSA.

Suppose you have an efficient decoding algorithm that is unknown to anyone else. Only you can correct these errors and find the corrected string.

Definition:

N = Modulo number.

G = Generating matrix size (n,k).

H = Parity check matrix H for G

S = A $k \times k$ matrix that is invertible mod N.

$S^{\wedge}(-1)$ = An inverse matrix of S.

P = An $n \times n$ permutation matrix.

$P^{\wedge}(-1)$ = An inverse matrix of P.

M = The message from Alice. Always, the message is represented in vector form and all elements must be less than N.

x = The sub-message. If the message vector's length is longer then n, then divide it into sub-message.

C = The ciphertext.

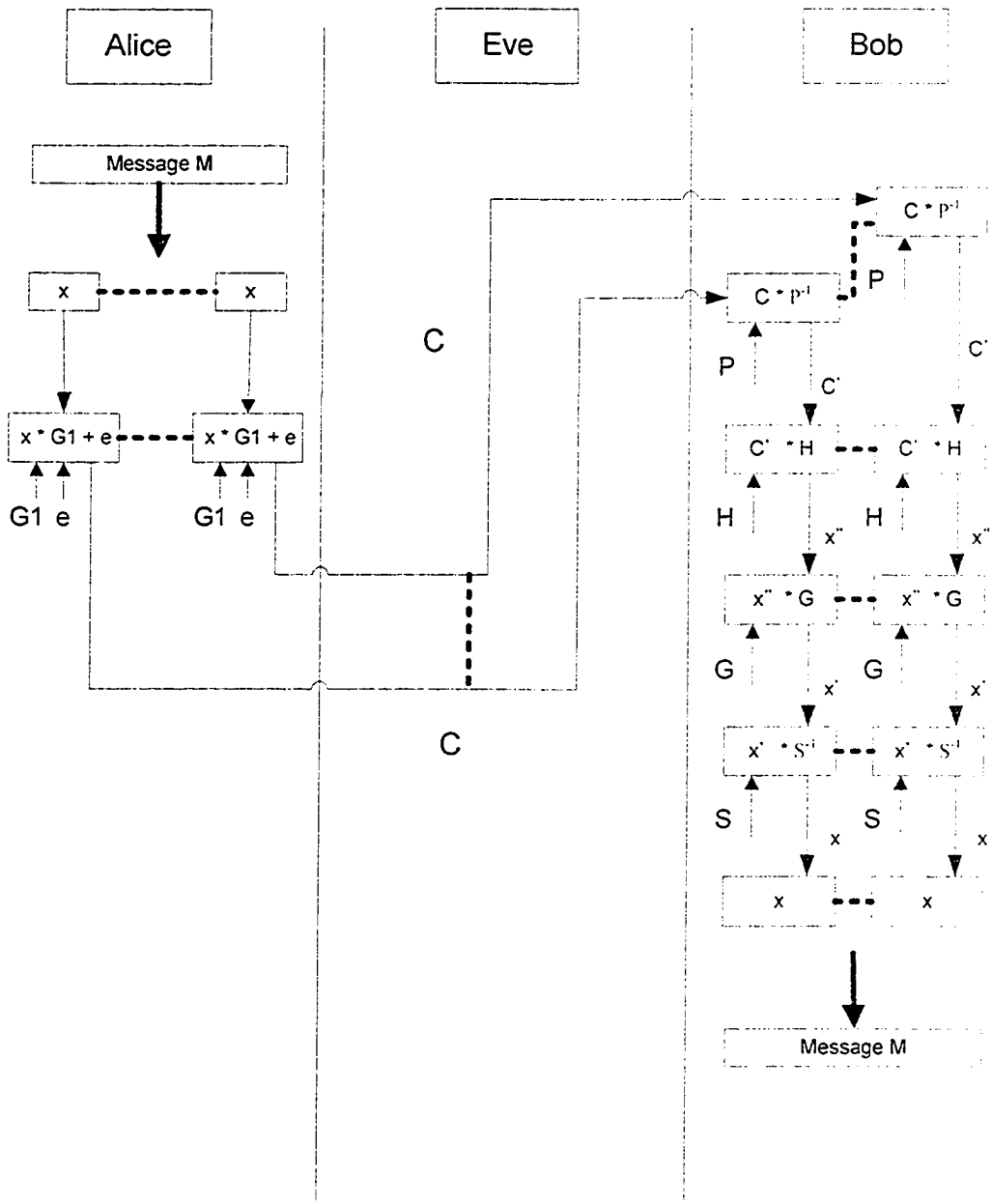
e = The error occuring in the ciphertext.

$\bullet$ = The $\bullet$ operator is ordinary matrix multiplication.

$G1 = S \bullet G \bullet P$

= $G1$ is the Bob's public key. The public key is calculate by S, G, and P. The value S, G, and P is Bob's private key.

Figure A.2



Let start with some examples with $N = 2$ (Shown in figure A.2):

Initialization steps:

Bob chooses G to be the generating matrix for a (n,k) linear error correcting code c with $d(c) = d$.

Let us use G for $[7,4]$ Hamming code as an example

He chooses S to be a $k \times k$ matrix that is invertible mod 2 and let's P be an $n \times n$ permutation matrix. This means P has exactly one "1" in every row and in

every column, with all the other entries being 0.

Let us use S:

$$S = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

$$P = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

He needs to calculate $G1 = S \bullet G \bullet P$, publish $G1$ as his public key and keep S, G, and P secret.

The matrix $G1 = \begin{pmatrix} 0 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \end{pmatrix}$

Protocol:

Alice wants to send Bob a message x. She generates a random binary string e of length n that has weight t.

She chooses $x = (1,0,1,1)$ and $e = (0,1,0,0,0,0,0)$ with weight 1.

She forms the ciphertext by computing $C = xG1 + e(\mathrm{mod}\,2)$. So now Alice will have $C = xG1 + e(\mathrm{mod}\,2) = (0,0,0,1,1,0,0)$.

Now Bob receives the ciphertext. He needs to decrypt it in order to get the message x. First, he calculates the $C' = C * P^{-1}$.

So $C' = C * P^{-1} = (0,0,1,0,0,0,1)$.

Then he calculates the syndrome of C' by applying the parity check matrix H

for G and changing the corresponding position bit to get x''

Assume j is the position that needs to be changed, so $j = C'' * H^t = (0,1,0)$. Change the 2^{nd} position bit in C' to get $x'' = (0,0,1,0,0,1,1)$.

He next forms a vector x' such that $x' * G = x''$.

In this case, $x' * G = x'' \Rightarrow x' = (0,0,1,0)$.

He decrypts by calculating $x = x' * S^{-1}$.

Now $x = x' * S^{-1} = (0,0,1,0) \bullet S^{-1} = (1,0,1,1)$.

This algorithm seems reasonably secure but compared to other algorithm, the size of the public key $G1$ is rather larger. If the size of the public key is large, then the operation we need in order to encrypt a message to a ciphertext will be increased as well. Therefore if the attacker, Eve, tries to use the brute force attacks to find the private key, then the time she needs is same as trying all ${}_{1024}C_{50} = 3 \times 10^{85}$ possible.

Compared with the RSA algorithm, given the same amount of time for Eve to try all the possible combination by brute force attacks, the public key can be smaller. Therefore, the McEliece algorithm is not really effective compare to other existing algorithms. Also, the size of the encrypted message can be an issue.

Bibliography

Literature

[1] Graff, J.C. "Cryptography and E-Commerce". Toronto: John Wiley & Sons Inc. 2001.

[2] Griffiths, A. Miller, J. Suzuki, D. Lewontin, R. Gelbart, W. "An Introduction to Genetic Analysis". 7th Edition. New York: W. H. Freeman and Company. 2000. PP 241 – 262.

[3] Hill, R. "A First Course in Coding Theory". Oxford: Clarendon press. 21. 2002. PP 47-80.

[4] Stajano, F. "Security for Ubiquitous Computing". Toronto: John Wiley & Sons Inc. 2002.

[5] Stallings, W. "Cryptography and Network Security: Principles and Practice". 2nd Edition. New Jersey: Prentice Hall. 1999.

[6] Trappe, W. Washington, L.C. "Introduction to Cryptography with Coding Theory". New Jersey: Prentice Hall. 2002.

[7] Diffie, W. Hellman, M. E. "Exhaustive Cryptanalysis of the NBS data encryption standard". Computer. 10(6): 74 – 84. June 1977.

[8] Diffie, W. Hellman, M. E. "New Directions in Cryptography". IEEE Transactions on Information Theory. Vol. IT – 22. No. 6. PP 644 – 654.

[9] Wiener, M. J. "Efficient DES Key Search". Technical Report 244. School of Computer Science. Carleton University. May 1994.

[10] Rivest, R. Shamir, A. Adleman, L. "A Method for obtaining Digital Signatures and Public-key Cryptosystems Communications of the ACM". Vol. 21 (2). 1978. PP 120 – 126.

[11] Quisquater, J-J. Quisquater, M. Quisquater, M. Quisquater M. Guillou, L.
Guillou, M. A. G. Guillou, G. Guillou, A. Guillou, G. Guillou, S.
Berson, T. "How to explain zero-knowledge protocols to your children".
In G. Brassard, Editor, Advances in Cryptology, CRYPTO '89,
Vol. 435 of Lecture Notes in Computer Science, PP 628 – 631.
Springer – Verlag, 1990. 20 – 24 August 1989.

[12] Feige, U. Fiat, A. Shamir, A. "Zero-knowledge proofs of identity". Journal of Cryptology. Vol. 1, Issue 2. August 1988. PP 77 – 94.

[13] McEliece, R. J. "A public-key cryptosystem based on algebraic coding theory". DSN Progress Report 42 – 44. Jet Propulsion Laboratory. Pasadena. 1978. PP 114 – 116.

Online Resource

[14] Brown, L. "Cryptographyand Network Security: Ch 12". Istanbul Technical University. Lecture Notes in Computer Science. 2003.
www.cs.itu.edu.tr/~etaner/courses/BESec02_03/ch12.ppt. (July, 2004)

[15] Eastlake, D. Jones, P. "RFC 3174 – US Secure Hash Algorithm 1 (SHA1)". September 2001. www.faqs.org/rfcs/rfc3174.html. (August, 2004)

[16] Gehani, A. LaBean, T. Reif, J. "DNA -- Cryptography". Department of Computer Science. Duke University. September, 2003.
www.cs.duke.edu/~reif/paper/DNAcrypt/dna.pdf. (July 2004)

[17] Larose, G.. "DRM Technologies: The Foundations of Usable Content Control".
www.info-mech.com/drm_technology.html. (August, 2004)

[18] RSA Laboratories. "RSA Laboratories' Frequently Asked Questions about

Today's Cryptography". Version 4.1. Section 7.17. 2000. www.rsasecurity.com/rsalabs/node.asp?id=2353. (July, 2004)

[19] RSA Laboratories. "RSA Laboratories' Frequently Asked Questions about Today's Cryptography". Version 4.1. Section 7.18. 2000. www.rsasecurity.com/rsalabs/node.asp?id=2354. (July, 2004)

[20] RSA Laboratories. "RSA Laboratories' Frequently Asked Questions about Today's Cryptography". Version 4.1. Section 7.19. 2000. www.rsasecurity.com/rsalabs/node.asp?id=2355. (July, 2004)

[21] Krawczyk, H. Bellare, M. Canetti, R. "RFC 2104 – HMAC: Keyed-Hashing for Message Authentication". February 1997. www.faqs.org/rfcs/rfc2104.html. (August, 2004)

[22] Adleman, L. "Molecular Computation of Solutions to Combinatorial Problems".
Science, 266:1021-1024, November 1994.
www.usc.edu/dept/molecular-science/fp-sci94.pdf. (August, 2004)

[23] Phillips, T. "The ABC of DNA computing". American Mathematics Society. February, 2000. www.ams.org/new-in-math/cover/dna-abc1.html. (August, 2004)

[24] Rivest, R. "RFC 1321 – The MD5 Message-Digest Algorithm". April 1992. www.faqs.org/rfcs/rfc1321.html. (August, 2004)