Chapter 8

Public Key Cryptography

Public-key cryptography is a radical departure from all that has gone before. Right up to modern times all cryptographic systems have been based on the elementary tools of substitution and permutation. However, public-key algorithms are based on mathematical functions and are asymmetric in nature, involving the use of two keys, as opposed to conventional single key encryption. Several misconceptions are held about p-k:

1. That p-k encryption is more secure from cryptanalysis than conventional encryption. In fact the security of any system depends on key length and the computational work involved in breaking the cipher.

2. That p-k encryption has superseded single key encryption. This is unlikely due to the increased processing power required.

3. That key management is trivial with public key cryptography, this is not correct.

8.1 Principles of Public-Key Cryptosystems

The concept of P-K evolved from an attempt to solve two problems, *key distribution* and the development of *digital signatures*. In 1976 Whitfield Diffie and Martin Hellman achieved great success in developing the conceptual framework. For conventional encryption the same key is used for encryption and decryption. This is not a necessary condition. Instead it is possible to develop a cryptographic system that relies on one key for encryption and a different but related key for decryption. Furthermore these algorithms have the following important characteristic:

> It is *computationally infeasible* to determine the decryption key given only knowledge of the algorithm and the encryption key.

In addition, some algorithms such as RSA, also exhibits the following characteristics:

> Either of the two related keys can be used for encryption, with the other used for decryption.

Figure 8.1 illustrates the P-K process. The steps are:

1. Each system generates a pair of keys.
2. Each system publishes its encryption key (public key) keeping its companion key private.
3. If A wishes to send a message to B it encrypts the message using B's public key.
4. When B receives the message, it decrypts the message using its private key. No one else can decrypt the message because only B knows its private key.

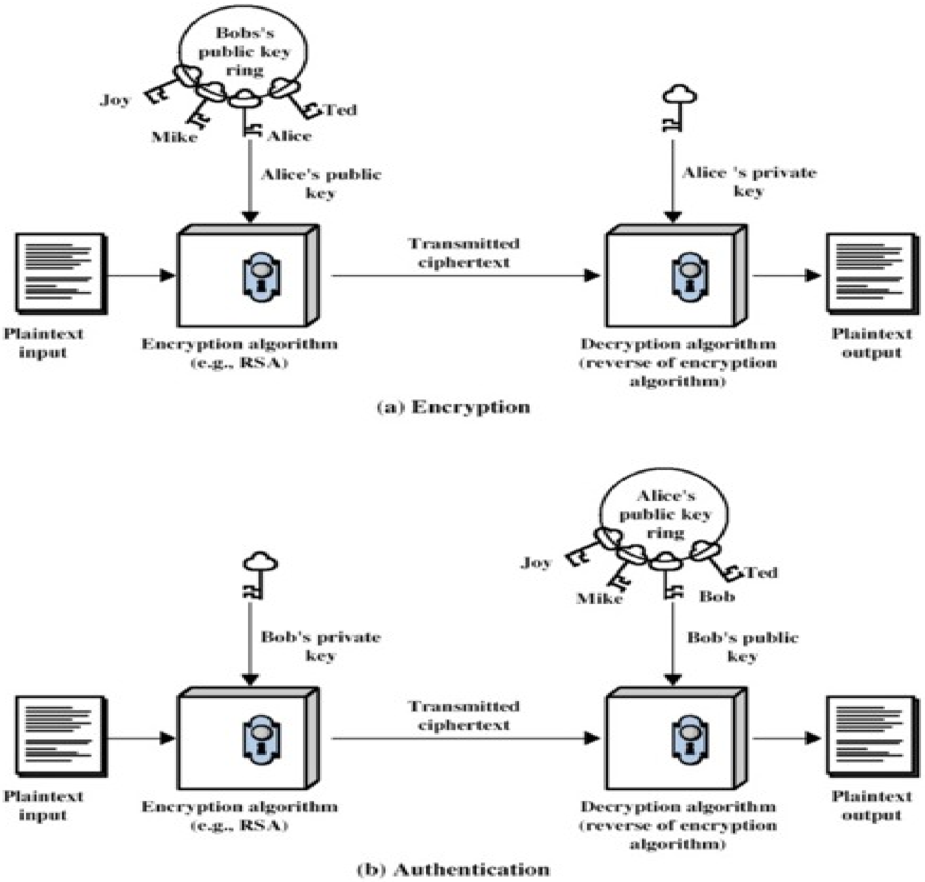


Figure 8.1: Public Key Cryptography.

Considering P-K in more detail we have a source A that produces plaintext X destined for B (figure 8.2). B generates a pair of keys KU$_b$ (a public key) and KR$_b$ (a private key). With X and KU$_b$ as inputs, A forms the ciphertext Y:

$$Y = E_{KU_b}(X)$$

The intended receiver B is able to invert the transformation with his private key:

$$X = D_{KR_b}(Y).$$

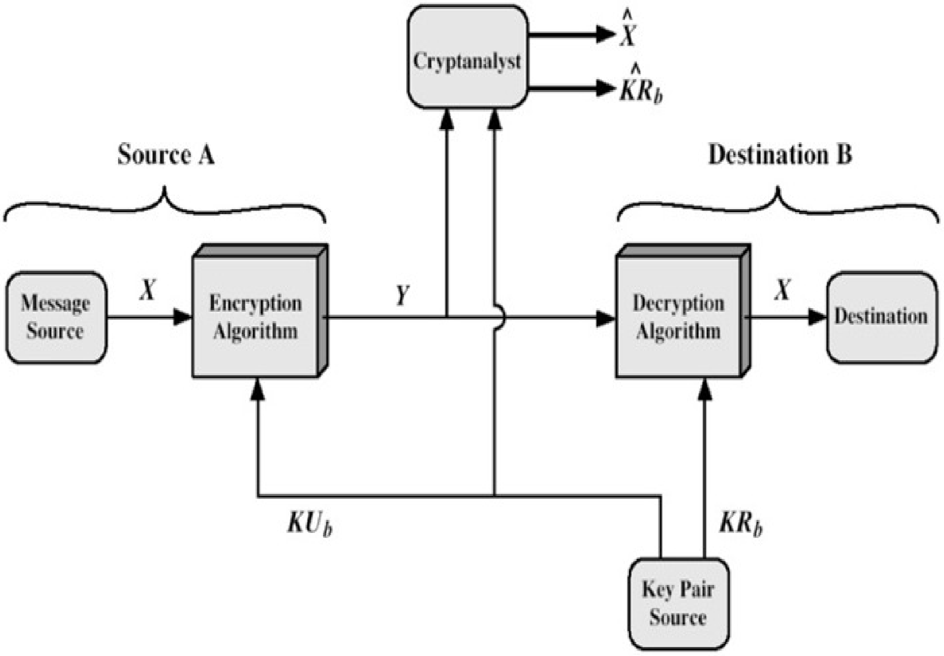


Figure 8.2: Public Key Cryptography: Secrecy.

8.1.1 Authentication

As previously mentioned, either key may be used for encryption with the other used for subsequent decryption. This facilitates a different form of scheme as shown in figure 8.3. In this case A prepares a message to B using his *private key* to encrypt and B can decrypt it using A's *public key* .

$$Y = E_{KR_a}(X)$$
$$X = D_{KU_a}(Y).$$

As the message was prepared using A's private key it could only have come from A therefore the entire message serves as a **digital signature**.

It should be noted that this scheme does not provide confidentiality because everyone has access to A's public key. Also the scheme is not efficient because B must maintain/store both the ciphertext (as proof of authenticity) and the decoded plaintext (for practical use of the document). A more efficient way of achieving the same result is to encrypt a small block of bits that are a function of the document. This block, called an **authenticator**, must have the property that it is infeasible to change the document without changing the authenticator. If the authenticator is encrypted using the senders

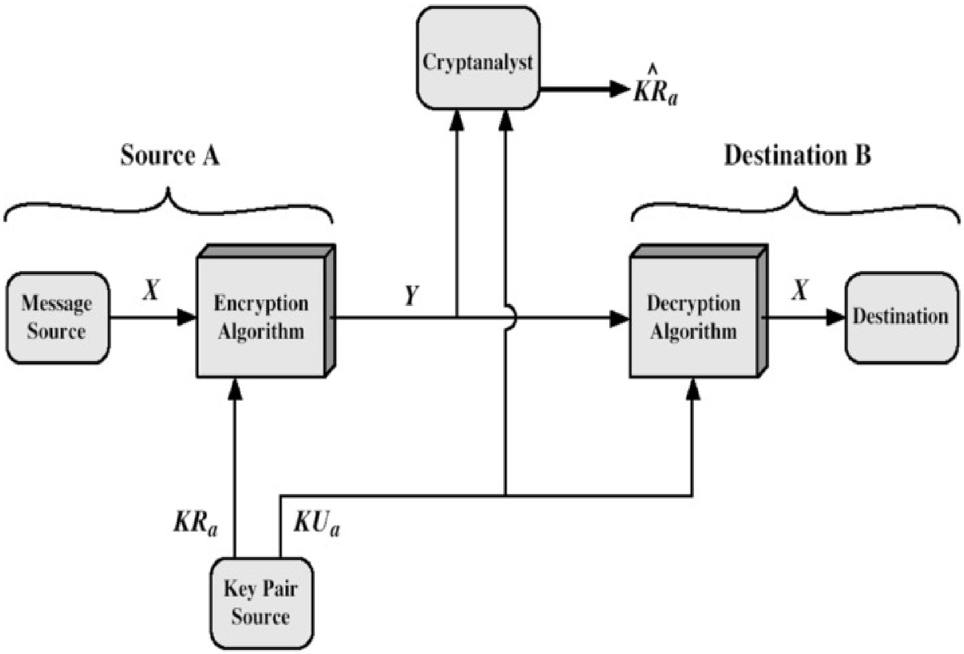


Figure 8.3: Public Key Cryptography: Authentication.

private key then it serves as a signature that verifies the origin, content and sequencing of the document.

8.1.2 Confidentiality and Authentication

If both are required, the double use of the public key scheme (figure 8.4) facilitates this. Here

$$\begin{aligned} Z &= E_{KU_b}[E_{KR_a}(X)] \\ X &= D_{KU_a}[D_{KR_b}(Z)] \end{aligned} \tag{8.1}$$

In this case the message is first encrypted using the senders private key, providing the digital signature. Then a second encryption is performed using the receivers public key, which delivers confidentiality. The disadvantage with this scheme is that the public-key algorithm which is complex must be used four times.

8.1.3 Applications for P-K cryptosystems

In broad terms, we can classify the use of public-key cryptosystems into three categories:

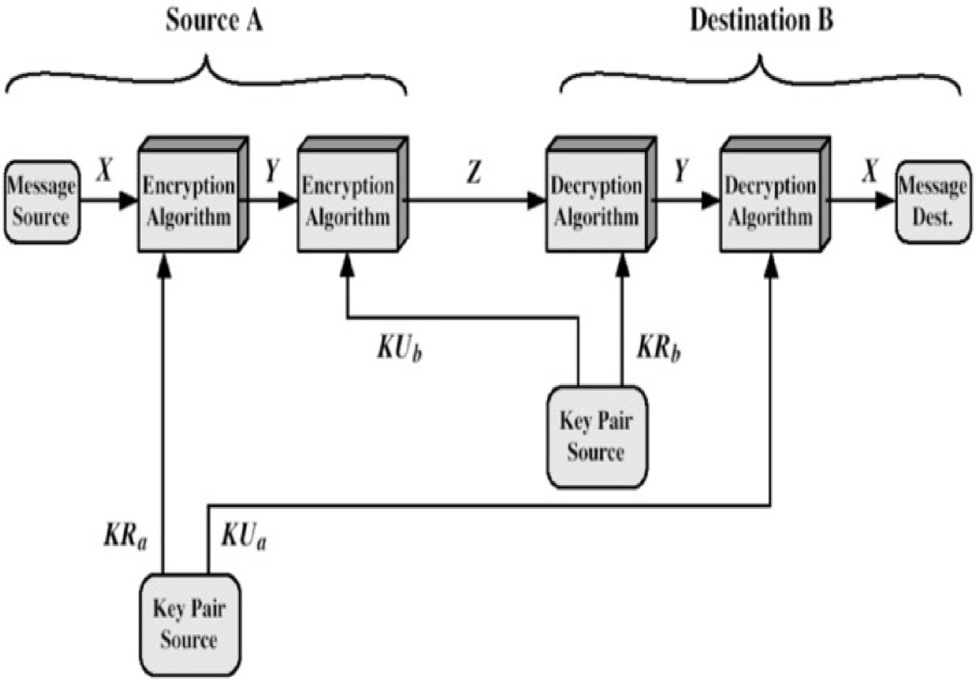


Figure 8.4: Public Key Cryptography: Secrecy and Authentication.

1. Encryption/decryption: where the sender encrypts the message with the receivers public key.
2. Digital signature: where the sender “signs” a message with his private key.
3. Key exchange: several approaches later.

However, not all algorithms are suitable for all three applications. Some can only be used for (say) digital signatures. RSA however can be used for all three as will be seen.

8.1.4 Requirements of the algorithm

The requirements of any P-K system were laid out by Diffie and Hellman:

1. It is computationally easy for party B to generate a key pair (public (KU) and private (KR)).
2. It is computationally easy for sender A knowing KU$_b$ and the message to be encrypted to generate the corresponding ciphertext $C = E_{KU_b}(M)$.
3. It is computationally easy for the receiver B to decrypt the resulting ciphertext using his private key (KR$_b$) to recover the original message. $M = D_{KR_b}(C) = D_{KR_b}[E_{KU_b}(M)]$.

4. It is computationally infeasible for an opponent, knowing the public key KU$_b$, to determine the private key KR$_b$.
5. It is computationally infeasible for an opponent, knowing KU$_b$ and C to recover the plaintext message M.
6. A sixth requirement that, although useful, is not necessary for all public-key applications - the encryption and decryption can be applied in either order: $M = E_{KU_b}[D_{KR_b}(M)] = D_{KU_b}[E_{KR_b}(M)]$.

These are formidable requirements as is evidenced by the fact that only one algorithm (RSA) has received widespread acceptance in over 20 years. The requirements boil down to the need for a **trapdoor one-way function**.

A **one-way function** is a function that maps a domain into a range such that every function value has a unique inverse, with the condition that the calculation of the function is easy whereas the calculation of the inverse is infeasible:

$$\begin{array}{lll} Y & = f(X) & \text{easy} \\ X & = f^{-1}(Y) & \text{infeasible} \end{array}$$

"Easy" is defined to mean a problem that can be solved in *polynomial time* as a function of input length (n). For example, the time to compute is proportional to n^a where a is a fixed constant. "Infeasible" is not as well defined however. Generally we can say that if the effort to solve is greater than polynomial time the problem is infeasible, e.g. if time to compute is proportional to 2^n.

Trapdoor one-way functions are a family of invertible functions f_k such that $Y = f_k(X)$ is easy if k and X known, $X = f_k(Y)$ is easy if k and Y are known, and $X = f_k^{-1}(Y)$ is infeasible if Y is known but k is not known. The development of a practical public-key scheme depends on the discovery of a suitable trapdoor one-way function.

8.1.5 The Knapsack Algorithm

Many algorithms have been proposed for P-K, and have subsequently been broken. The most famous of these was proposed by Ralph Merkle as follows. The problem deals with determining which of a set of objects are in a container, say a knapsack. Of the list of say six objects of different weights shown below, which subset is in the knapsack if it weighs S?

Object 1	455 g
Object 2	341 g
Object 3	284 g
Object 4	132 g
Object 5	82 g
Object 6	56 g

Given that the weight of the knapsack is $S = 821$ grams, the problem is to determine which of the items are in the knapsack. The problem shown here is simple but when the number of items is increased (> 100) it becomes *computationally infeasible*.

So what we have is six different objects with six different weights. The knapsack weighs nothing itself but with a selected number of objects in it weighs (say) 821 grams. Which objects does it contain?

Merkle's contribution was to show how to turn the knapsack problem into a scheme for encryption and decryption. In other words how to incorporate "trapdoor" information which enabled the easy solution of the knapsack problem. Suppose we wish to send messages in blocks of n bits. We define the following:

- Cargo vector: $a = (a_1, a_2, \ldots, a_n)$, where a_i is an integer.
- Plaintext message block $x = (x_1, x_2, \ldots, x_n)$, where x_i is a binary digit.
- Corresponding ciphertext S:

$$S = \mathbf{a} \cdot \mathbf{x} = \sum_{i=1}^{n} (a_i x_i).$$

The vector **a** is considered to be a list of potential elements to be put into the knapsack with each vector element equal to each weight of the element. The message block **x** is considered to be a selection of elements of the cargo vector in the knapsack. Each element is set equal to 1 if the corresponding element is in the knapsack and 0 if it is not. The product S is simply the sum of the selected item's weights (i.e. the weight of the contents of the knapsack).

As an example lets take a cargo vector as follows:

$$\begin{aligned} &\mathbf{a} = (455, 341, 284, 132, 82, 56) \\ &\mathbf{x} = (x_1, x_2, x_3, x_4, x_5, x_6) \qquad \text{a six bit binary number} \\ &S = 821 \end{aligned}$$

For encryption **a** is used as the public key. The person sending the message **x** performs $S = \mathbf{a} \cdot \mathbf{x}$ and sends S as the ciphertext. The receiving party must recover **x** from S and **a**. Two requirements are as follows:

1. That there be a unique inverse for each value of S. For example if $S = 3$ and $\mathbf{a} = (1, 3, 2, 5)$ then the problem would have two solutions, $\mathbf{x} = (1, 0, 1, 0)$ and $\mathbf{x} = (0, 1, 0, 0)$. The value of **a** must be chosen so that each combination of elements yields a unique value of S.
2. That decryption is hard in general but easy if special knowledge is available.

For large values of n the knapsack problem is hard in general. If however we impose the condition that each element of **a** is larger than the sum of the preceding elements we have:

$$a_i > \sum_{j=1}^{i-1} a_j \qquad 1 < i < n$$

This is known as a **superincreasing vector** and in this case the solution is easy. For example, consider the vector $\mathbf{a}' = (171, 197, 459, 1191, 2410)$ which satisfies the condition. Suppose we have $S' = \mathbf{a}' \cdot \mathbf{x}' = 3798$. Because $3798 > 2410$, a_5 must be included ($x_5 = 1$) because without a_5 all the other elements cannot contribute enough to add up to 3798 (or 2410). Now consider $3798 - 2410 = 1388$. The number 1388 is bigger than 1191 so a_4 must be included ($x_4 = 1$). Continuing[1] in this fashion we find that $x_3 = 0$, $x_2 = 1$ and $x_1 = 0$.

What Merkle did was to tie an easy superincreasing knapsack problem to a hard general knapsack problem. Suppose we choose an easy knapsack vector $\mathbf{a}'$ with n elements. Also select two integers m and ω such that m is greater than the sum of the elements, and ω is relatively prime to m, that is:

$$m > \sum_{i=1}^{n} a_i \qquad gcd(w, m) = 1$$

Now, we construct a hard knapsack vector, **a**, by multiplying an easy vector a' by $\omega \pmod{m}$:

$$\mathbf{a} = \omega \mathbf{a}' \pmod{m}$$

The vector **a** will in general not be superincreasing and therefore can be used to construct hard knapsack problems. However, knowledge of ω and m enables the conversion of this hard knapsack problem to an easy one. To see this, first observe that since ω and m are relatively prime, there exists a unique multiplicative inverse ω^{-1}, modulo m. Therefore:

$$\omega^{-1}\mathbf{a} = \mathbf{a}' \pmod{m}.$$

We can now state the knapsack scheme. The ingredients are as follows:

1. $\mathbf{a}'$, a superincreasing vector (private, chosen).
2. m, an integer larger than the sum of all a_j's (private, chosen).
3. ω, an integer relatively prime to m (private, chosen).
4. ω^{-1}, the inverse of ω, modulo m (private, calculated).
5. **a**, equal to $\omega\mathbf{a}' \pmod{m}$ (public, calculated).

[1] If there exists a solution the value will eventually end up being zero. If no solution exists then the value will not go to zero.

The private key consists of the triple ($\omega^{-1}, m, \mathbf{a}'$) and the public key consists of the value of $\mathbf{a}$.

Suppose user A has published his public key $\mathbf{a}$ and that user B wishes to send a message $\mathbf{x}$ to A. B calculates the sum $S = \mathbf{a} \cdot \mathbf{x}$. The determination of $\mathbf{x}$ given S and $\mathbf{a}$ is difficult so this is a secure transmission. However, on receipt, user A is able to decrypt easily. Defining $S' = \omega^{-1}S \pmod{m}$ we have the following:

$$\begin{aligned} S &= \mathbf{a} \cdot \mathbf{x} = \omega \mathbf{a}' \cdot \mathbf{x} \\ S' &= \omega^{-1}S \pmod{m} \\ &= \omega^{-1}\omega \mathbf{a}' \cdot \mathbf{x} \pmod{m} \\ &= \mathbf{a}' \cdot \mathbf{x} \end{aligned}$$

Thus we have converted the hard problem of finding $\mathbf{x}$ given S into the easy problem of finding $\mathbf{x}$ given S' and $\mathbf{a}'$.

For example, given the plaintext message $\mathbf{x} = (0,1,0,0,1,0,1,1)$, user B computes $\mathbf{a} \cdot \mathbf{x} = 818$. User A first computes $S' = \omega^{-1}S \pmod{m} = 415$, and then solves the easy knapsack problem to recover $\mathbf{x} = (0,1,0,0,1,0,1,1)$.

Merkle put up $\$100$ for anyone who broke the algorithm. Within 4 years Adi Shamir (of RSA) collected the money.

8.2 RSA

The RSA algorithm was developed by Ron **R**ivest, Adi **S**hamir and Len **A**dleman at MIT in 1978. Since this time it has reigned supreme as the most widely accepted and implemented general-purpose approach to public-key encryption.

The RSA scheme is a *block cipher* in which the plaintext and ciphertext are integers between 0 and $n-1$ for some n. The scheme makes use of an expression with exponentials. Plaintext is encrypted in blocks having a binary value less than some number n. For some plaintext block M and ciphertext block C we have:

$$\begin{aligned} C &= M^e \pmod{n} \\ M &= C^d \pmod{n} = (M^e)^d \pmod{n} \\ M &= M^{ed} \pmod{n} \end{aligned} \tag{8.2}$$

Both sender and receiver know n. The sender knows the value of e and only the receiver knows the value of d. To restate:

$$\begin{aligned} KU &= \{e, n\} \\ KR &= \{d, n\} \end{aligned} \tag{8.3}$$

For this algorithm to be satisfactory for public-key encryption, the following requirements must be met:

1. It is possible to find values of e, d and n such that $M^{ed} = M \pmod{n}$ for all $M < n$.

2. It is relatively easy to calculate M^e and C^d for all values of $M < n$.

3. It is infeasible to determine d given e and n.

Focusing initially on the first question we need to find a relationship of the form: $M^{ed} = M \pmod{n}$.

If we recall that Euler's theorem states that

$$a^{\phi(m)} \equiv 1 \pmod{m} \qquad \text{gcd}(a, m) = 1 \tag{8.4}$$

There is a corollary to this theorem that can be used to produce the required relationship. Given two prime numbers p and q and integers $n = pq$ and m, with $0 < m < n$, the following relationship holds:

$$m^{\phi(n)+1} \equiv m^{(p-1)(q-1)+1} \equiv m \pmod{n} \tag{8.5}$$

If gcd$(m, n) = 1$ then this holds by virtue of Euler's theorem. Suppose however that gcd$(m, n) \neq 1$. What does this mean? Well, because $n = pq$, the equality gcd$(m, n) = 1$ is equivalent to the logical expression (m is not a multiple of p) **AND** (m is not a multiple of q). If m is a multiple of p then n and m share the prime factor p and are not relatively prime (the same can be said for q). Therefore, the expression gcd$(m, n) \neq 1$ must be equivalent to the negation of the foregoing logical expression. Therefore, gcd$(m, n) \neq 1$ is equivalent to the logical expression (m is a multiple of p) **OR** (m is a multiple of q).

Looking at the case in which m is a multiple of p, so that the realtionship $m = cp$ holds for some positive integer c. In this case we must have gcd$(m, q) = 1$. Otherwise, we have m a multiple of p and m a multiple of q and yet $m < pq$. If gcd$(m, q) = 1$ then Euler's theorem holds and

$$m^{\phi(q)} \equiv 1 \pmod{q}$$

But then, by the rules of modular arithmetic,

$$\begin{aligned} [m^{\phi(q)}]^{\phi(p)} &\equiv 1 \pmod{q} \\ m^{\phi(n)} &\equiv 1 \pmod{q} \end{aligned}$$

Therefore, there is some integer k such that

$$m^{\phi(n)} = 1 + kq$$

Multiplying each side by $m = cp$,

$$\begin{aligned} m^{\phi(n)+1} &= m + kcpq = m + kcn \\ m^{\phi(n)+1} &\equiv m \pmod{n} \end{aligned}$$

A similar line of reasoning is used for the case in which m is a multiple of q. Thus, equation 8.5 is proven. An alternative form of this corollary is directly relevant to RSA:

$$\begin{aligned} m^{k\phi(n)+1} &\equiv [(m^{\phi(n)})^k \times m)] \pmod{n} \\ &\equiv [(1)^k \times m)] \pmod{n} \text{ by Euler's theorem} \\ &\equiv m \pmod{n} \end{aligned} \tag{8.6}$$

We can now state the RSA scheme. The ingredients are the following:

$p, q,$ two primes	(private, chosen)
$n = pq$	(public, calculated)
e, with gcd$(\phi(n), e) = 1$; $1 < e < \phi(n)$	(public, chosen)
$d \equiv e^{-1} \pmod{\phi(n)}$	(private, calculated)

The private key consists of $\{d, n\}$ and public key is $\{e, n\}$. Suppose that user A has published his public key and that user B wishes to send the message M to A. B calculates $C = M^e \pmod{n}$ and transmits C. On receipt of the ciphertext C user A decrypts by calculating the following: $M = C^d \pmod{n}$. Figure 8.5 summarises the algorithm.

Example:

a) Select p=7 ,q=17

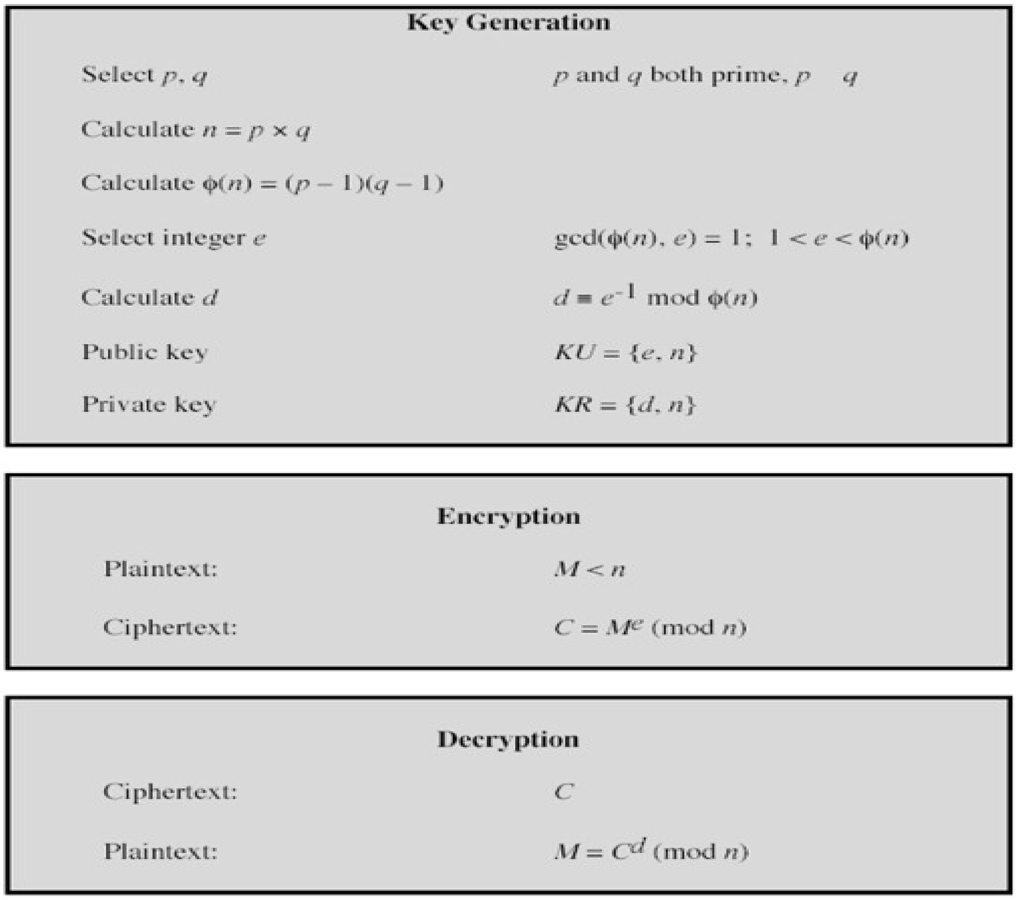
b) Calculate $n = pq = 7 \times 17 = 11$

c) Calculate $\phi(n) = (p - 1)(q - 1) = 96$.

d) Select e, relatively prime to and less than $\phi(n)$, say $e = 5$.

e) Determine d such that $de = 1 \pmod{96}$ and $d < 96$.

f) The correct value for d is 77 because $77 \times 5 = 385 = 4 \times 96 + 1$ (can be calculated using the extended version of Euclid's algorithm).



Key Generation	
Select p, q	p and q both prime, p q
Calculate $n = p \times q$	
Calculate $\phi(n) = (p-1)(q-1)$	
Select integer e	$\gcd(\phi(n), e) = 1;\ 1 < e < \phi(n)$
Calculate d	$d \equiv e^{-1} \bmod \phi(n)$
Public key	$KU = \{e, n\}$
Private key	$KR = \{d, n\}$

Encryption	
Plaintext:	$M < n$
Ciphertext:	$C = M^e \pmod{n}$

Decryption	
Ciphertext:	C
Plaintext:	$M = C^d \pmod{n}$

Figure 8.5: The RSA algorithm

g) The resulting public key is KU= $\{5, 119\}$ and private key is KR$\{77, 119\}$. Say the plaintext is $M = 19$. For encryption 194 is raised to the 5th power, yielding $2, 476, 099$. Upon division by 119, the remainder is 66, hence ciphertext sent is 66. For decryption it is determined using KR that $6677 \equiv 19 \pmod{119}$ so the recovered plaintext is 19.

8.3 Computational Aspects

The complexity of the computation required boils down to two aspects, the actual encryption/decryption process and the key generation process.

1. Encryption and Decryption: Both involve raising a (large) integer to a (large) integer power modulo n. If the exponentiation was done over the integers and then reduced modulo n, the intermediate values would be gigantic. Fortunately

we can make use of a property of modular arithmetic:

$$[(a \bmod n).(b \bmod n)] \bmod n = (a.b) \bmod n \tag{8.7}$$

Thus, intermediate results may be reduced modulo n. This makes the calculation more practical. Another consideration is the efficiency of exponentiation, since with RSA we are dealing with large exponents.

To see how efficiency might be improved consider calculating x^{16}. A straightforward approach is to perform 15 multiplications, $x^{16} = x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x \cdot x$. However we can receive the same result with just four multiplications if we repeatedly take the square of each partial result successively forming x^2, x^4, x^8 and x^{16}. Note that even utilising shortcuts etc. there is a requirement for arithmetic operations with arbitrarily large integers and most computers are restricted in this capability. More generally, suppose we wish to find the value a^m, with a, m positive integers. If we express m as a binary number $b_k b_{k-1} \ldots b_0$, then we have the following:

$$m = \sum_{b_i \neq 0} 2^i$$

therefore,

$$a^m = a^{(\sum_{b_i \neq 0} 2^i)} = \prod_{b_i \neq 0} a^{(2^i)}$$

$$a^m \bmod n = \left[\prod_{b_i \neq 0} a^{(2^i)}\right] \bmod n = \left(\prod_{b_i \neq 0} [a^{(2^i)} \bmod n]\right) \bmod n$$

which can be done using a square an multiply algorithm.

Generally the values of a, m and n are of the order 1024 bits and larger and are therefore implemented on specially designed crypto ASICs utilising algorithms such as the **Montgomery exponentiation scheme**.

2. Key Generation: Before two parties can use a public key system, each must generate a pair of keys. This involves the following tasks:

   - Determining two prime numbers p, q.
   - Selecting either e or d and calculating the other.

   Firstly, considering selection of p and q. Because the value $n = pq$ will be known to any opponent, to prevent the discovery of p, q through exhaustive methods, these primes must be chosen from a sufficiently large set (must be large numbers). On the other hand the method used for finding large primes must be reasonably efficient. At present there are no useful techniques that yield arbitrarily large primes. The procedure is to pick at random an odd number of the desired magnitude and test that it is prime. If not, repeat until a prime is found.

A variety of tests for primality have been developed, all of which are statistical in nature. The tests however may be run in such a way as to attain a probability, of as near 1 as is desired, that a particular number is prime. One of the more efficient algorithms is the **Miller-Rabin scheme**, which performs calculations on n, the candidate prime and a randomly chosen integer a. This procedure may be repeated as required.

In summary the procedure for picking a prime is as follows:

(a) Pick an odd integer n at random (e.g., using a pseudorandom number generator).

(b) Pick an integer $a < n$ at random.

(c) Perform the probabilistic primality test, (such as Miller-Rabin). If n fails the test then go to step a.

(d) If n passes a sufficient number of tests then accept it, otherwise go to step b.

8.4 The Security of RSA

RSA gets its security from the difficulty of factoring large numbers. The public and private keys are functions of a pair of large (100 to 200 digits) prime numbers. Recovering the plaintext from one key and the ciphertext is equivalent to factoring the product of two primes

Taking a first look at cryptographic considerations. Three possible approaches include:

1. Brute Force: Try all possible keys. Standard defense is a large key space. The larger e and d are the better, so we have the following:

	5 years ago	Today
Casual use	384 bits	768 bits
Commercial use	512 bits	1024
Military Spec.	1024 bits	4096 bits

where the military specification is only an estimate due to this information being classified. For comparison, 512 bits is about 150 decimal digits.

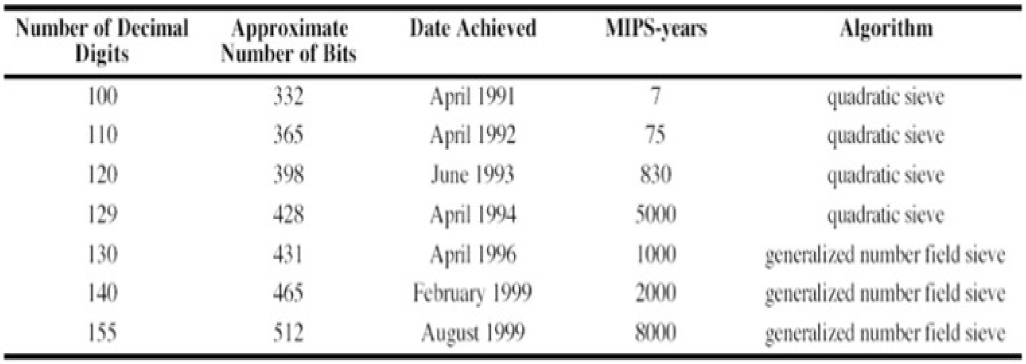
2. Mathematical attacks:

- Factor n into its 2 primes thus enabling calculation of $\phi(n)$ and the private key $e \equiv d^{-1} \pmod{\phi(n)}$. The best known algorithm used in factoring an integer n is time proportional to:

$$O(n) = e^{\sqrt{ln(n).ln(ln(n))}} \tag{8.8}$$

as discussed in the section on complexity theory. For a 200 digit number this would take about 1000 years on a large machine. However, there has been a lot of progress made in factorisation over the last number of years. This can be seen in figure 8.6.

- Determining $\phi(n)$, given n or determining d given n and e. These are at least as time consuming as factoring n so the factorising performance of algorithms is used as the benchmark to evaluate the security of RSA. In addition to specifying n of order $150 - 200$ decimal digits, some other recommendations are that p and q should differ in length by only a few digits. Other constraints are also specified to ensure the difficulty of factorising is maintained.



Number of Decimal Digits	Approximate Number of Bits	Date Achieved	MIPS-years	Algorithm
100	332	April 1991	7	quadratic sieve
110	365	April 1992	75	quadratic sieve
120	398	June 1993	830	quadratic sieve
129	428	April 1994	5000	quadratic sieve
130	431	April 1996	1000	generalized number field sieve
140	465	February 1999	2000	generalized number field sieve
155	512	August 1999	8000	generalized number field sieve

Figure 8.6: Progress in factorisation

3. Timing attacks: These are an implementation attack that depends on the running time of an algorithm. We will look at them in more detail when we study attacks on cryptosystems.

Another public key algorithm was defined by Diffie and Hellman. This algorithm is limited to key exchange only however. We will look at this next.

8.5 Diffie Hellman Key Exchange

The first published P-K algorithm appeared in the paper by Diffie and Hellman that defined public key cryptography however it is limited to the secure exchange of a secret key and not of a message. The security of the scheme depends on the difficulty of computing discrete logarithms which were discussed earlier in the course.

The Diffie-Hellman key exchange consists of two publicly known numbers: a prime number p and an integer α that is a primitive root of q. Suppose the users A and

B wish to exchange a key. User A selects a random integer $X_A < q$ and computes $Y_A = \alpha^{X_A} \bmod q$. Similarly user B independently selects a random integer $X_B < q$ and computes $Y_B = \alpha^{X_B} \bmod q$. Each side keeps the X values private and makes the Y value available publicly to the other side. User A computes the key as $K = (Y_B)^{X_A} \bmod q$ and user B computes the key as $K = (Y_A)^{X_B} \bmod q$. These two calculations produce identical results and the result is that the two sides have exchanged a secret key. This can be seen because:

$$\begin{aligned} K &= (Y_B)^{X_A} \bmod q \\ &= (\alpha^{X_B} \bmod q)^{X_A} \bmod q \\ &= (\alpha^{X_B})^{X_A} \bmod q \\ &= (\alpha^{X_A})^{X_B} \bmod q \\ &= (\alpha^{X_A} \bmod q)^{X_B} \bmod q \\ &= (Y_A)^{X_B} \bmod q \end{aligned}$$

Furthermore because X_A and X_B are private, an opponent is forced to take a discrete logarithm to determine the key. For example, attacking the secret key of user B the opponent must compute:

$$X_B = ind_{\alpha,q}(Y_B)$$

where $ind_{\alpha,q}(Y_B)$ is the discrete logarithm, or index, of Y_B for the base $\alpha \bmod q$. The scheme can be summarised as shown in figure 8.7

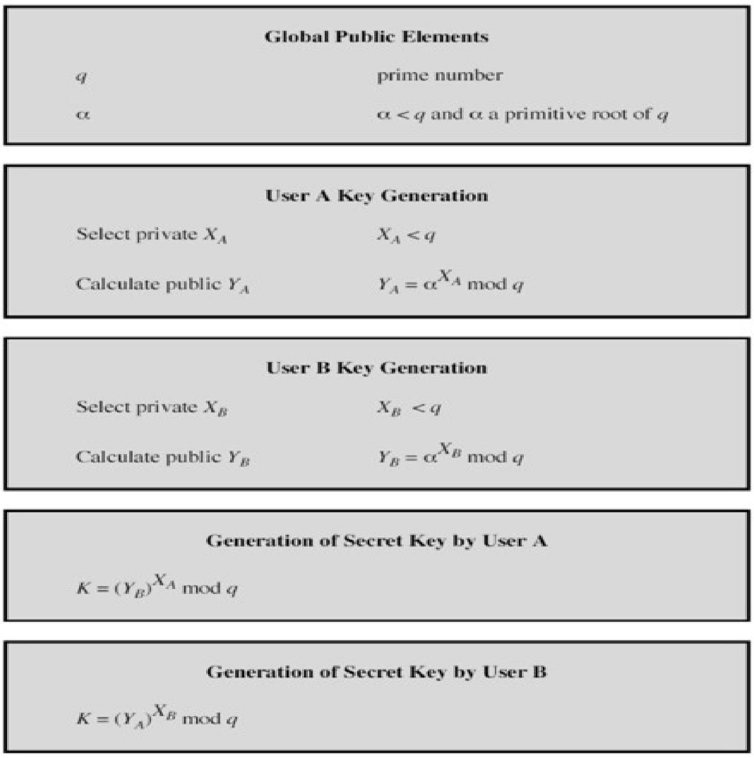
For example lets say we have the values $q = 353$ and a primitive root $\alpha = 3$. We can see that $\alpha = 3$ is a primitive root of $q = 353$ due to the following reasoning. If α is a primitive root of a prime q then the set of numbers $\{\alpha, \alpha^2, \ldots, \alpha^{\phi(q)}\}$ are distinct modulo q and hence form the set $\{1, 2, \ldots (q-1)\}$ in some order. In this case $\alpha = 3$ and it can be seen to be a primitive root of $q = 353$ as $\{3 \bmod 353, 3^2 \bmod 353, \ldots, 3^{353} \bmod 353, \}$ which contains all the elements of $\{1, 2, \ldots 352\}$.

Suppose A and B select the private keys $X_A = 97$ and $X_B = 233$ respectively. To calculate the secret key K user A calculates:

$$\begin{aligned} Y_A &= \alpha^{X_A} \bmod q \\ &= 3^{97} \bmod 353 \\ &= 40 \end{aligned}$$

Similarly user B calculates

$$\begin{aligned} Y_B &= \alpha^{X_B} \bmod q \\ &= 3^{233} \bmod 353 \\ &= 248 \end{aligned}$$



Global Public Elements

q prime number

α $\alpha < q$ and α a primitive root of q

User A Key Generation

Select private X_A $X_A < q$

Calculate public Y_A $Y_A = \alpha^{X_A} \bmod q$

User B Key Generation

Select private X_B $X_B < q$

Calculate public Y_B $Y_B = \alpha^{X_B} \bmod q$

Generation of Secret Key by User A

$K = (Y_B)^{X_A} \bmod q$

Generation of Secret Key by User B

$K = (Y_A)^{X_B} \bmod q$

Figure 8.7: The Diffie Hellman Key Exchange Algorithm.

Then we have $K = 248^{97} \bmod 353 = 40^{233} \bmod 353 = 160$.

We assume the attacker would have q, α, Y_A, Y_B which for this example might be enough using a brute force approach. However with large numbers this becomes impractical.