Symmetric Key cryptosystem

Symmetric encryption, also referred to as conventional encryption or single key encryption was the only type of encryption in use prior to the development of public-key encryption in 1976.

The symmetric encryption scheme has five ingredients (see Figure 1):

1. **Plaintext:** This is the original intelligible message or data that is fed to the algorithm as input.
2. **Encryption algorithm:** The encryption algorithm performs various substitutions and permutations on the plaintext (see the examples of the substitution and permutation ciphers in Lecture 8).
3. **Secret Key:** The secret key is also input to the encryption algorithm. The exact substitutions and permutations performed depend on the key used, and the algorithm will produce a different output depending on the specific key being used at the time.
4. **Ciphertext:** This is the scrambled message produced as output. It depends on the plaintext and the key. The ciphertext is an apparently random stream of data, as it stands, is unintelligible.
5. **Decryption Algorithm:** This is essentially the encryption algorithm run in reverse. It takes the ciphertext and the secret key and produces the original plaintext.

There are two requirements for a symmetric key cryptosystem

1. We assume it is impractical to decrypt a message on the basis of the ciphertext plus knowledge of the encryption/decryption algorithm. In other words, we do not need to keep the algorithm secret; we need to keep only the key secret.
2. Sender and the receiver must have obtained copies of the secret key in a secure fashion and must keep the key secure. If someone can discover the key and knows the algorithm, all communications using this key is readable. We will describe how we can use a public-key cryptosystem for a secure key exchange later in this lecture.

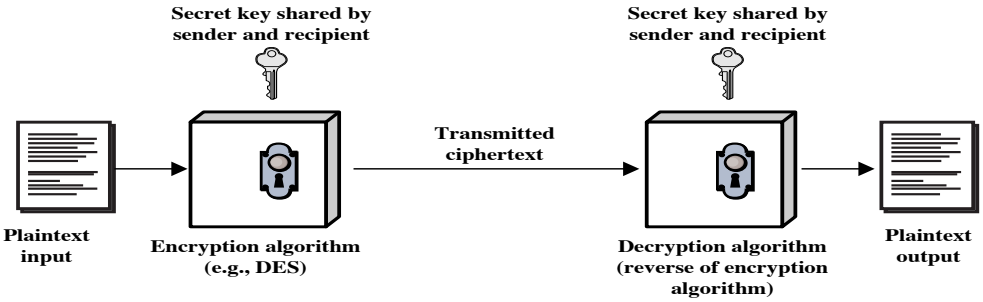

Figure 2.1 Simplified Model of Symmetric Encryption

Figure 1: Model for Symmetric key encryption

We saw examples of some very simple symmetric key cryptosystems in Lecture 8: these included the shift, substitution, Vignere and permutation ciphers. The shift, substitution, and Vignere ciphers are also called *stream ciphers*, since the ciphers act more or less on each plaintext character as it comes along. On the other hand, transposition ciphers are called *block ciphers* since they act on blocks of plaintext instead. We have seen that some of the substitution ciphers may be safe from a brute force attack (since the keyspace is large), but many of them are not resistant to a statistical analysis of the ciphertext.

In the late 1940s Claude Shannon laid out some good design criteria for symmetric ciphers. First, he repeated Kerckhoff's principle (see Lecture 8) that the only secret should be the key, not the larger mechanism (encryption/decryption algorithms) of the cipher. Second, Shannon emphasized that a good cipher should incorporate both *confusion* and *diffusion*. By *confusion* we mean that a cipher should hide *local* patterns in language from an attacker. By *diffusion* we mean that the cipher should mix around different parts of the plaintext, so that nothing is left in its original position. For example, a monoalphabetic cipher (see Lecture 8) such as a substitution cipher fails at *confusion* since local features such as single letter frequencies are very revealing. A polyalphabetic substitution cipher such as the Vignere cipher is more effective at *confusion* because it does not encrypt the same character the same way every time. But Vignere fails at diffusion, because it does not move anything. This makes it prone to an attack known as Friedman's attack (I have an URL on the course webpage that discusses Friedman's attack). The transposition ciphers on the other hand excel at *diffusion*, because exactly what they do is move things around. Thus, a good cipher should combine both substitution and transposition to achieve Shannon's goals of *confusion* and *diffusion*. Most contemporary symmetric ciphers repeat cycles of one or the other: a substitution, then a transposition, then a substitution, and so on. One such example of a symmetric cipher is the Data Encryption Standard (DES).

DES was developed at IBM, as a modification of an earlier system known as *Lucifer*. It was developed as a standard for applications in 1977. It has since be replaced by the *Advanced Encryption Standard*.

Simplified Data Encryption Standard (DES)

We will present a simplified version of DES in this lecture. A good account of this also appears in Section 3.1 of Stallings [1]. Figure 2 illustrates the overall structure of the simplifies DES, which we will refer to as S-DES. The S-DES encryption algorithm takes an 8-bit block of plaintext and a 10-bit key as input and produces an 8 bit block of ciphertext as output. The S-DES decryption algorithm takes an 8-bit block of ciphertext and the same 10-bit key used to produce the ciphertext as input and produces the original 8-bit block of plaintext.

The encryption algorithm involves five functions: an initial permutation (IP); a complex function labelled f_K, which involves both permutation and substitution operations and depends on a key input; a simple permutation function (SW) that switches the two halves of the data; the function f_K again; and finally a permutation function that is the inverse of the initial permutation (IP^{-1}).

The function f_K takes as input not only the data passing through the en-

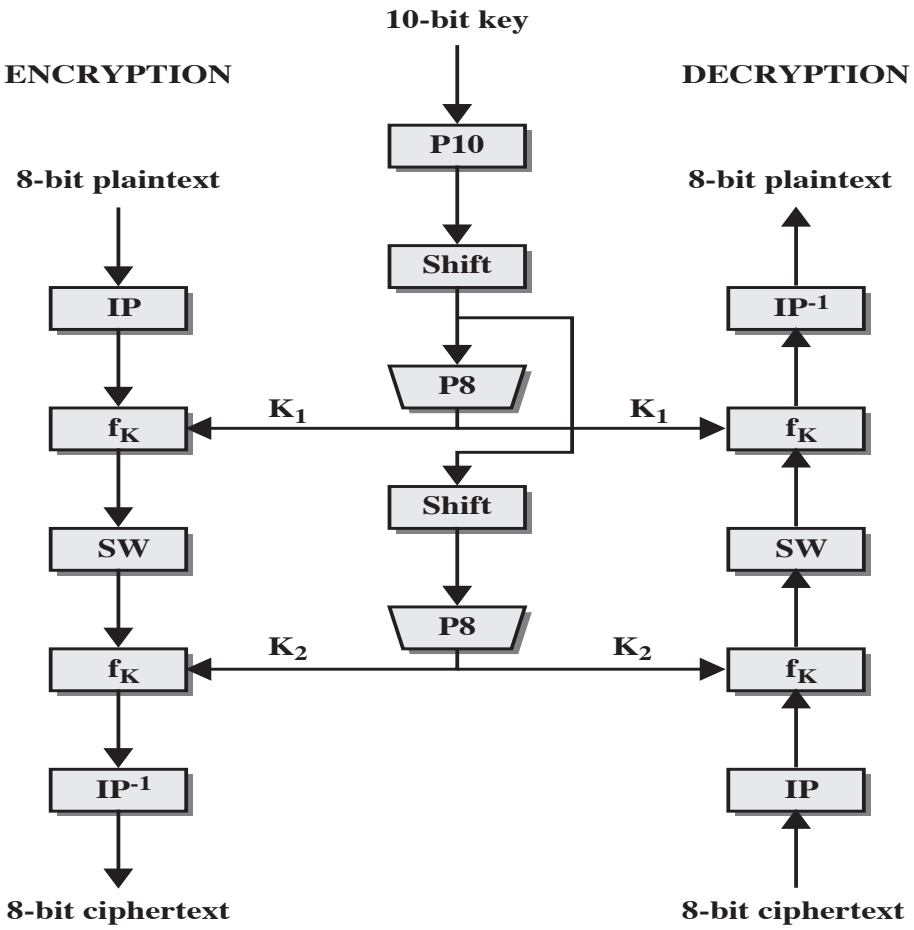

Figure 3.1 Simplified DES Scheme

Figure 2: Simplified DES Scheme

cryption algorithm, but also an 8-bit key. The idea is to generate two 8-bit subkeys from the original 10-bit key as shown in figure 2. In this case, the key is first subjected to a permutation (P10). Then a shift operation is performed. The output of the shift operation (left shift by 1 bit on the two halves of the input) then passes through a permutation function that produces an 8-bit output (P8) for the first subkey (K_1). The output of the shift operation also feeds into another shift (left shift by 2 bits on the two halves of the data) and another instance of (P8) to produce the second subkey (K_2).

We can concisely express the encryption algorithm as

$$\text{ciphertext} = IP^{-1} \circ (f_{K2}(SW(f_{K1}(IP(\text{plaintext})))))$$

where

$$\begin{aligned} K_1 &= \text{P8(Shift(P10(key)))} \\ K_2 &= \text{P8(Shift(Shift(P10(key))))} \end{aligned}$$

Decryption is also shown in figure 2 and is essentially done in the same fashion as encryption (with the order of the keys reversed), i.e.,

$$\text{plaintext} = IP^{-1}(f_{K1}(SW(f_{K2}(IP(\text{ciphertext})))))$$

It is shown below that the functions f_{K1}, f_{K2} and SW are their own inverses, i.e., applying them twice on an input gives back this input (using this information verify that decrypting the ciphertext gives the original plaintext). We now examine the elements of S-DES in more detail.

S-DES Key Generation:

S-DES depends on the use of a 10-bit key shared between sender and receiver. From this key, two 8-bit subkeys are produced for use in particular stages of the encryption and decryption algorithm. Figure 3 depicts the stages followed to produce the subkeys.

The first operation P10 is simply a permutation of the 10 bits. As an example, let P10 be the following: For example, the key (1010000010) is permuted

P10

3	5	2	7	4	10	1	9	8	6

to (1000001100) (can you see how this is done?). Next, perform a circular left shift (LS-1), or rotation, separately on the first five bits and the second five bits. In our example, the result is (00001 11000). Next, we apply P8, which picks out and permutes 8 of the 10 bits according to the following rule: The result is

P8

6	3	7	4	8	5	10	9

subkey 1 (K_1). In our example, this yields (10100100). We then go back to the pair of 5-bit strings produced by the two LS-1 functions and perform a circular left shift of 2 bit positions on each string. In our example, the value (00001 11000) becomes (00100 00011). Finally, P8 is applied again to produce K_2. In our example, the result is (01000011).

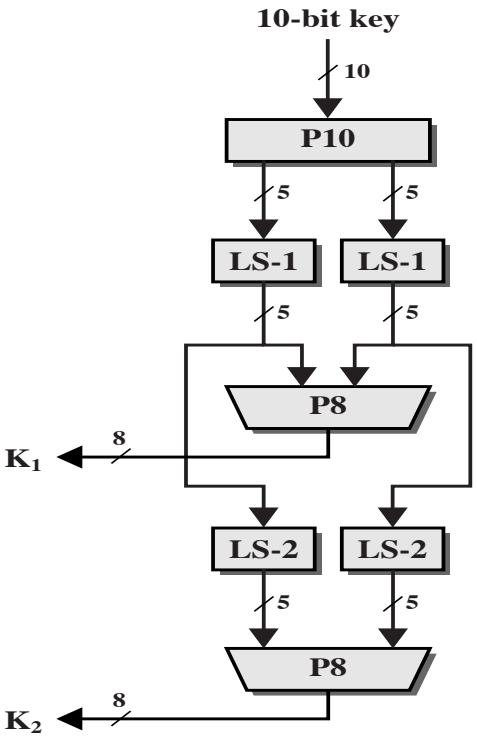

Figure 3.2 Key Generation for Simplified DES

Figure 3: Key Generation for Simplified DES

S-DES Encryption:

The S-DES encryption algorithm is shown in detail in figure 4. As we mentioned before, encryption involves the sequential application of five functions. We examine each of these in detail below:

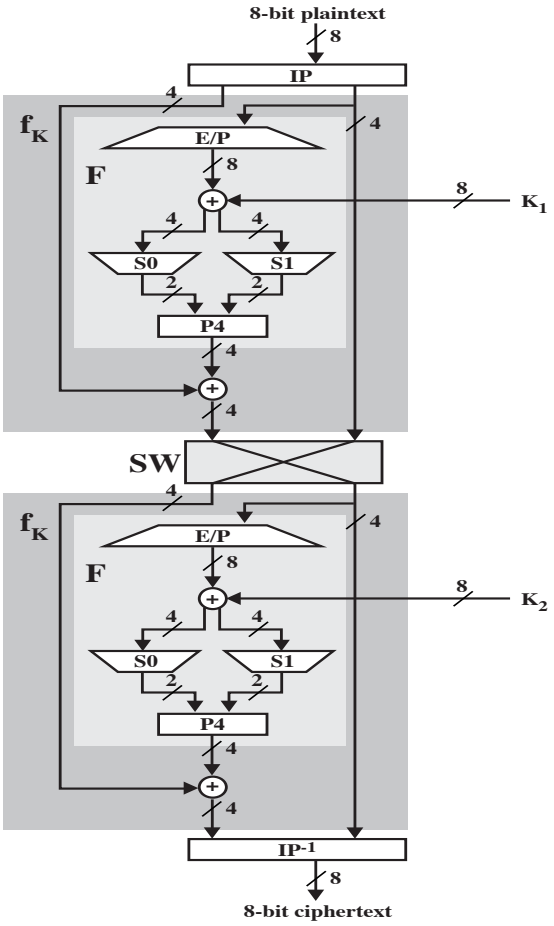

Figure 3.3 Simplified DES Encryption Detail

Figure 4: Simplified DES Encryption Detail

Initial and Final Permutations:

The input to the algorithm is an 8-bit block of plaintext, which we first permute using the IP function. For example, IP could be This retains all 8 bits of the plaintext but mixes them up. At the end of the algorithm, the following inverse

IP

2	6	3	1	4	8	5	7

permutation is used: It is easy to show by an example (try this out!) that the

IP^{-1}

4	1	3	5	7	2	8	6

second permutation is indeed the inverse of the first; that is, $IP^{-1}(IP(X)) = X$.

The function f_K

The most complex component of the S-DES is the function f_K, which consists of a combination of permutation and substitution functions. The functions can be expressed as follows: let L and R be the leftmost 4 bits and rightmost 4 bits of the 8 bit input to f_K, and let F be a mapping from 4-bit strings to 4-bit strings. Then we let

$$f_K(L, R) \quad = \quad (L \oplus F(R, SK), R)$$

where SK is a subkey (which is K_1 in stage 1 and K_2 in stage 2) and $\oplus$ is the bit-by-bit exclusive OR function (I hope everyone is familiar with exclusive OR function; in simple terms the output is 1 if the two bits being exclusive ORed are dissimilar and 0 if these are the same).

Using the definition of the exclusive OR operation, we find that is we apply the function f_K again to the input $f_K(L, R)$, we have

$$\begin{array}{rcc} f_K(f_K(L, R)) & = & f_K(L \oplus F(R, SK), R) \\ & = & (L \oplus F(R, SK) \oplus F(R, SK), R) \\ & = & (L, R) \end{array}$$

I urge you all to verify the last equality via an example; thus the function f_K is its own inverse (note that the subkey SK employed must be the same).

We now describe the mapping F. Let us say that the input is a 4-bit number $(n_1n_2n_3n_4)$. The first operation is an expansion/permutation (EP) operation. As an instance, let us take EP as For the above E/P, the 8 bit result

EP

4	1	2	3	2	3	4	1

is $(n_4n_1n_2n_3n_2n_3n_4n_1)$. The 8-bit subkey $K_1 = (k_{11}, \ldots, k_{18})$ is added to this value using exclusive-OR. Let the result be $(p_{0,0}, p_{0,1}, p_{0,2}, p_{0,3}, p_{1,0}, p_{1,1}, p_{1,2}, p_{1,3})$. The first 4 bits are fed into the S-box S0 to produce a 2-bit output, and the remaining 4 bits are fed into S1 to produce another 2-bit output. These two boxes are defined as follows:

$$S0 \quad = \quad \begin{bmatrix} 1 & 0 & 3 & 2 \\ 3 & 2 & 1 & 0 \\ 0 & 2 & 1 & 3 \\ 3 & 1 & 3 & 2 \end{bmatrix} \qquad S1 \quad = \quad \begin{bmatrix} 0 & 1 & 2 & 3 \\ 2 & 0 & 1 & 3 \\ 3 & 0 & 1 & 0 \\ 2 & 1 & 0 & 3 \end{bmatrix}$$

Let us denote the four rows and columns of these S-matrices are $0, \ldots, 3$. The S-boxes operate as follows: The first and fourth input bits are treated as a 2-bit number that specifies a row of the S-box, and the second and third input specify a column of the S-box. The entry in that row and column, in base 2, is the 2-bit output. For example, if $(p_{0,0}, p_{0,3}) = (00)$ and $(p_{0,1}, p_{0,2}) = (10)$, then the output is from row 0, column 2 of S0, which is 3, or (11) in binary. Similarly, $(p_{1,0}, p_{1,3})$ and $(p_{1,1}, p_{1,2})$ are used to index into a row and column of S1 to produce an additional 2 bits of output. Next, the 4 bits produced by S0 and S1 undergo a further permutation (P4); a particular instance is given below: The output of P4 is the output of the function F.

P4

2	4	3	1

The Switch function

The function f_K only alters the leftmost 4 bits of the input. The switch function (SW) interchanges the left and right 4 bits so that the second instance of f_K operates on a different 4 bits. In the 2nd instance, the EP,S0,S1, and P4 functions are the same. The key input is K_2.

In the actual DES, the input was broken into blocks of 64 bits, there are 16 rounds of encryption, and a 56-bit key is used from which sixteen 48-bit subkeys are calculated.

Public key cryptography:

There are two problems with symmetric key cryptography:

1. The first is the distribution of the symmetric key to be shared by Alice and Bob. Moreover, if there are n people communicating with each other, we would need to distribute $\frac{n(n-1)}{2}$ symmetric keys between them. Is there some way to reduce the number of keys that need to be shared?.

2. The 2nd is the concept of a digital signature: for instance, how can Bob prove that a message indeed came from Alice?. This is important if Alice and Bob do not really trust one another, and Alice later disclaims that she sent Bob any message.

Diffie and Hellman at Stanford University achieved an astounding breakthrough in 1976 by coming up with a method that addressed both problems and that was radically different from all previous approaches to cryptography, going back over four millennia. Before we describe how these shortcomings are overcome, let us describe the various components of a public-key cryptosystem. There are six ingredients in all; see figure 5 (compare with the five ingredients of a symmetric key cryptosystem in figure 1).

1. **Plaintext:** This is the readable message or data that is fed into the algorithm as input.

2. **Encryption algorithm:** The encryption algorithm performs very transformations on the plaintext.

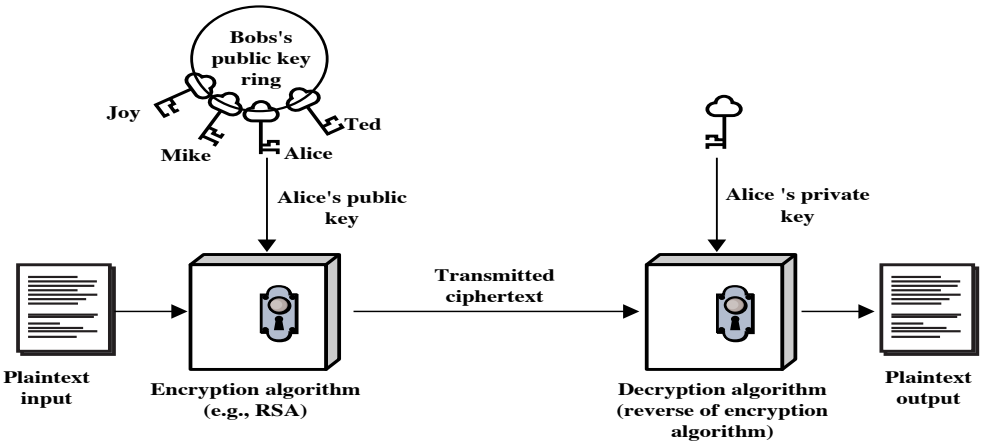

Figure 9.1 Public-Key Cryptography

Figure 5: Public-key cryptography

3. **Public and private key:** This is a pair of keys that have been selected so that if one is used for encryption, the other is used for decryption. The exact transformations performed by the encryption algorithm depends on the public or private keys that is provided as input.

4. **Ciphertext:** The scrambled message produced as output after encryption.

5. **Decryption algorithm:** This algorithm accepts the ciphertext and the matching key and produces the original plaintext.

The essential steps followed by Alice and Bob in a public-key cryptosystem are the following:

1. Alice generates a pair of keys to be used for encryption and decryption of messages. We shall assume that Bob does so independently too.

2. Alice then places her public key in a public register in a key distribution center, or her own personal webpage etc. The companion key is kept private. Bob does so too.

3. If Alice wants to send a confidential message to Bob, Alice encrypts the message using the encryption algorithm using Bob's public key.

4. When Bob receives the message, he decrypts it using her private key (to which he alone has access).

With this approach, all participants have access to public keys, and private keys are generated locally by each participant and therefore need never be distributed. As long as a system controls its private key, its incoming communication is secure. At any time, a system can change it private key and publish the companion public key to replace its old public key. Each participant communicates with any other participant through the use of four keys (two private/two public) alone. Moreover, each participant uses the same set of public/private keys while communicating with n other participants, i.e., one no longer needs $\frac{n(n-1)}{2}$ keys that need top be exchanged.

The following analogy (courtesy Kartik) best illustrates the public key cryptosystem. Imagine a postal system which is insecure, i.e., the postal employees open all boxes. Since Alice wants to send Bob a confidential message, she encloses it in a box secured with a press lock maintained by Bob (Bob's public key). By a press lock, I mean one that can be locked by just pressing it; no key is required for locking purposes; only one for unlocking. Since the box is secured with a press lock no one (including Alice) can open it once it is shut. The box makes it to Bob via the postal system (you can imagine the disgruntled employees who just can't open it:), and Bob opens (unlocks) the box via the key to the press lock (Bob's private key), which he alone maintains.

For the transaction between Alice and Bob just mentioned, let KU_a, KR_a (KU_b, KR_b) be Alice's(Bob's) public and private keys respectively. If Alice has to send Bob an m bit message $X = x_1x_2 \ldots x_m$, then she encrypts it using Bob's public key to form the ciphertext $Y = KU_b(X)$. When Bob receives the ciphertext, he decrypts it using his private key, i.e., $X = KR_b(Y) = KR_b(KU_b(X)) = X$. We also illustrate this in figure 6.

Now suppose Alice wants to sign a message X to be sent to Bob (let's not worry about the security of the message for the time being), she encrypts it

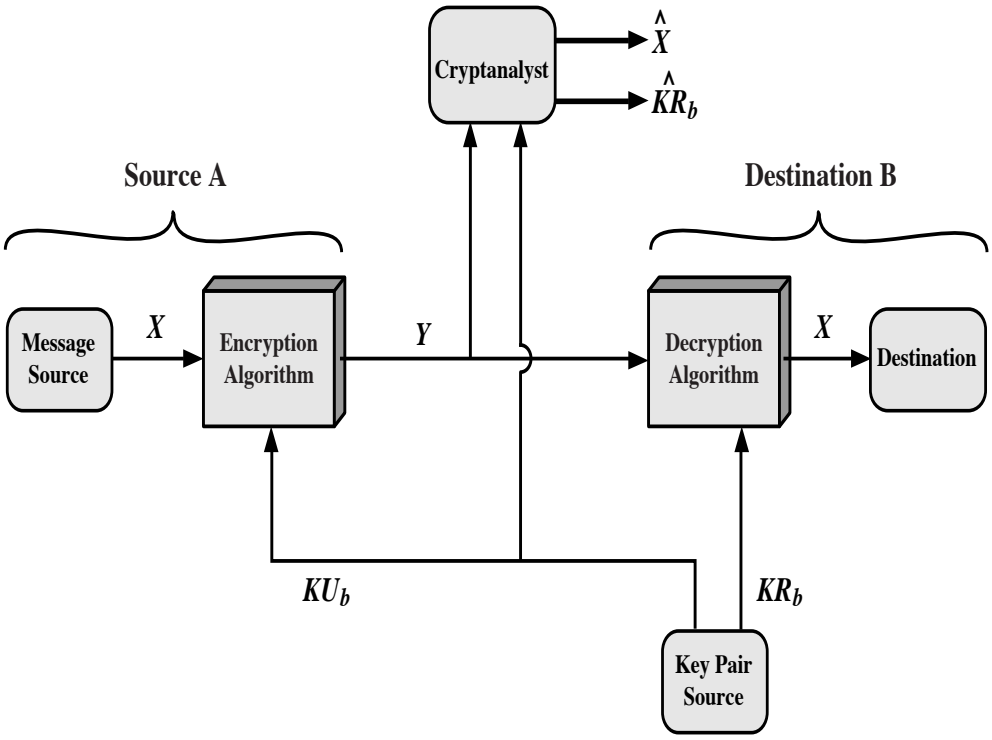

Figure 9.2 Public-Key Cryptosystem: Secrecy

Figure 6: Public-key cryptosystem: Secrecy

using her private key, i.e., $Y = KR_a(X)$. When Bob gets this message from Alice, he can ensure that it came from her by decrypting it using Alice's public key (to which he has access) to recover the plaintext X, i.e. $X = KU_a(Y)$. Note that since Alice alone has access to her private key, she alone could have sent the message. We illustrate this in figure 7.

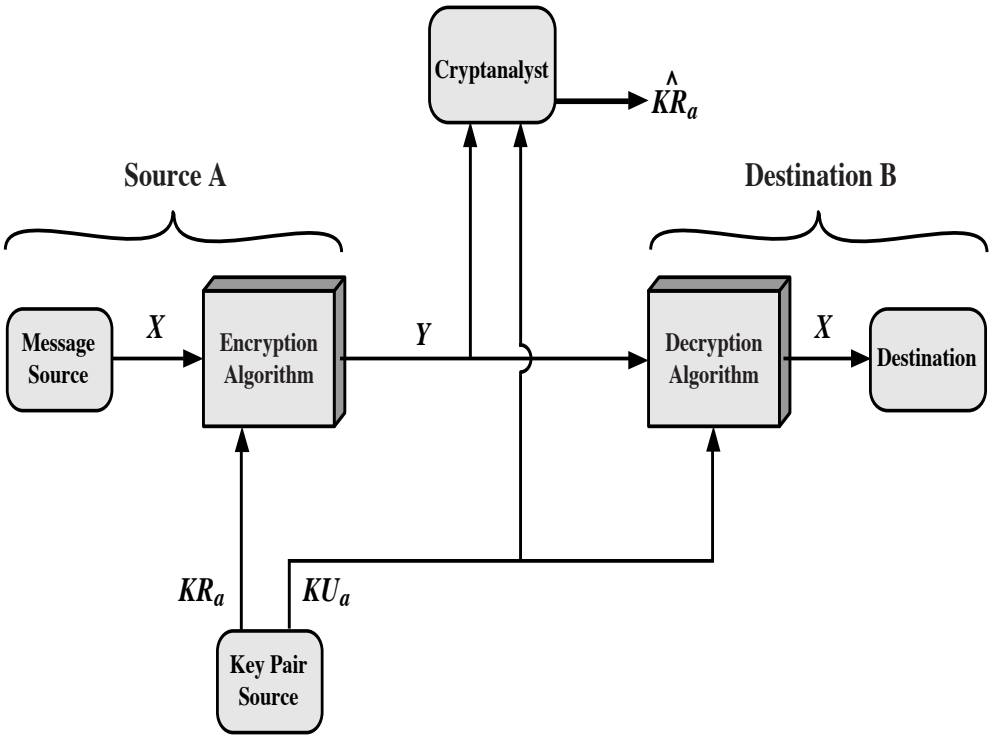

Figure 9.3 Public-Key Cryptosystem: Authentication

Figure 7: Public-key cryptosystem: Digital signatures

If one needs confidentiality as well as authentication, then Alice first signs the plaintext X using her private key to obtain $X' = KR_a(X)$; she then encrypts it using Bob's public key to obtain the ciphertext $Y = Ku_b(X') = KU_b(KR_a(X))$ (note the order of the two operations), and sends Y to Bob. Bob first decrypts it using his private key to obtain X', i.e. $X' = KR_b(Y)$; he then verifies that it was indeed sent by Alice by decrypting it using Alice's public key to obtain

$X = KU_a(X') = KU_a(KR_b(Y))$. Thus, both confidentiality and authentication are guaranteed in the above transaction. We illustrate in figure 8. Finally, Alice

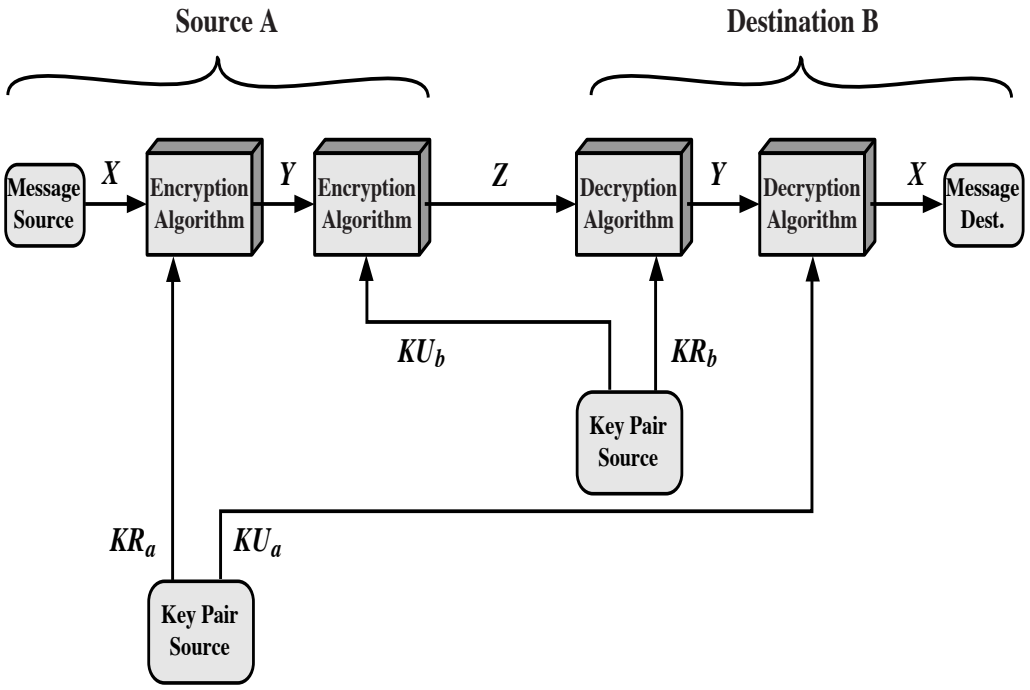

Figure 9.4 Public-Key Cryptosystem: Secrecy and Authentication

Figure 8: Public-key cryptosystem: Secrecy and Digital Signatures

and Bob can use the public key cryptosystem just mentioned to exchange a session key to be used for symmetric key cryptography. In this case the plaintext exchanged is just the session key. It must be emphasized that the encryption and decryption operations in a public key cryptosystem are more expensive than those in a symmetric key cryptosystem like DES. Thus, one can use a public key cryptosystem simply to exchange the session key in a symmetric key cryptosystem, and then make use of the fast encryption/deryption prevalent in such a symmetric key cryptosystem.

The requirements for public-key cryptography are the following:

1. It should be computationally easy for Bob to generate his private and public keys KR_b and KU_b respectively.

2. It should be computationally easy for Alice, knowing Bob's public key and the message M to be encrypted, to generate the corresponding ciphertext.

3. It should be computationally easy for Bob to decrypt the resulting ciphertext using his private key to produce the corresponding plaintext.

4. It is computationally infeasible (difficult) for an opponent (Trudy/Oscar), knowing Bob's public key KU_b and the ciphertext, to recover the original plaintext or to guess Bob's private key KR_b.

Both encryption and decryption in public key cryptography involve calculating functions. In the above text, *easy* is defined to mean a problem that can be solved in polynomial time as a function of input length. Thus, if the length of the input is n bits, then the time needed to compute the function is proportional to n^a, where a is a fixed constant. In complexity terminology, such functions/algorithms are said to belong to the class **P**. The term *difficult* implies that the time required to compute the function is proportional to 2^n.

We now turn to the definition of a *trapdoor one way function*. Such functions are easy to calculate in one direction, but their inverses are difficult to compute, unless certain additional information is known. With additional information, the inverse of the function can be computed efficiently, i.e., in polynomial time. Thus, a trapdoor one-way function is a family of invertible functions f_K, such that

$$\begin{array}{lcll} Y & = & f_K(X), & \text{easy if k and X are known} \\ X & = & f_K^{-1}(Y), & \text{easy if k and Y are known} \\ X & = & f_K^{-1}(Y), & \text{infeasible if only Y (but not k) is known} \end{array}$$

Thus, the development of a practical public-key scheme depends on discovery of an appropriate trapdoor one-way function.

The RSA algorithm:

Diffie and Hellman came up with the concept of public-key cryptography. One of the first cryptographic algorithms that met the requirements for public-key systems was the RSA algorithm developed by Ron Rivest, Adi Shamir, and Len Adelman then at MIT; they were the recipients of the 2003 Turing award (the Noble prize in Computer Science) for this algorithm. The actual algorithm is shown in figure 9. The RSA scheme is a block cipher in which the plaintext and ciphertext are integers between 0 and $n-1$. A typical size of n is 1024 bits, or 309 decimal digits. Thus, the plaintext is encrypted in blocks, with each block having a binary value less than some number n. That is, the block size must be less than or equal to $\log_2(n)$; in practice, the block size is k bits, where $2^k < n < 2^{k+1}$. Encryption and decryption are of the following form, for some plaintext block M and ciphertext block C:

$$\begin{array}{lcl} C & = & M^e \bmod n \\ M & = & C^d \bmod n \end{array}$$

Both sender (Bob) and receiver (Alice) must know the value of n. The sender knows the value of e, and only the receiver knows the value of d. Thus, this

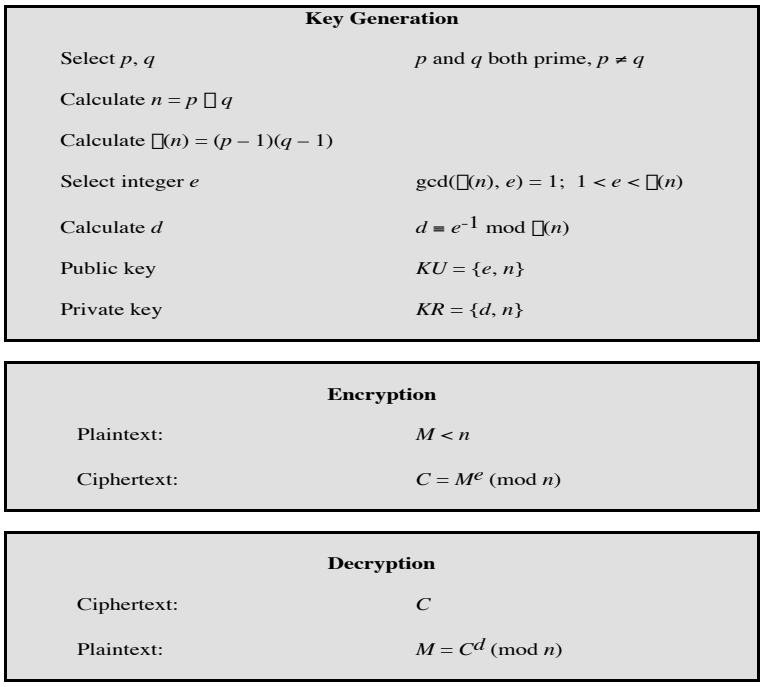

Figure 9.5 The RSA Algorithm

Figure 9: The RSA Algorithm

is a public-key encryption algorithm with a public key of $KU_a = \{e, n\}$ and a private key $KR_a = \{d, n\}$. The ingredients e, d, n are chosen as follows:

1. First one generates two prime numbers p and q. These are privately chosen by Bob (say).
2. $n = pq$. The product n is then part of the public key (KU_b) of Bob. One also computes $\phi(n) = (p-1)(q-1)$; this is the number of integers less than n and relatively prime to n.
3. One then computes e, such that $1 < e < \phi(n)$, with $\gcd(\phi(n), e) = 1$, i.e. e is relatively prime with $\phi(n)$. This is also part of Bob's public key (KU_b).
4. One then computes $d = e^{-1} \bmod \phi(n)$. This can be computed using extended Euclid's algorithm. The value of d is part of Bob's private key (KR_b) and keeps this information to himself.

The security of the RSA cryptosystem relies on the fact that, given n, it is believed to be infeasible to factor n into its two prime factors p and q. Although, the factoring problem appears to be very difficult (there is no present algorithm that can factor numbers efficiently), no one has shown this as yet. Thus, the security of the RSA algorithm lies on the unproven claim that factoring is indeed a difficult problem. If one could factor n into its factors p and q, then one knows the value of $\phi(n) = (p-1)(q-1)$ and one can determine $d = e^{-1} \bmod \phi(n)$ (e is already known as it is part of the public key).

We close this lecture with an example of the RSA algorithm. For this example, for Bob the keys were generated as follows:

1. Select two prime numbers, $p = 17$ and $q = 11$.
2. Calculate $n = pq = 17 \times 11 = 187$ and $\phi(n) = (p-1)(q-1) = 16 \times 10 = 160$.
3. Select e such that e is relatively prime to $\phi(n) = 160$ and less than $\phi(n)$; we choose $e = 7$.
4. Determine $d = e^{-1} \bmod 160$, i.e., $de = 1 \bmod 160$ and $d < 160$. The unique value of d that fits this bill is $d = 23$, since $23 \times 7 = 161 = 1 \times 160 + 1$; d can be calculated using extended Euclid's algorithm.

The resulting keys are public key $KU_b = \{7, 187\}$ and $KR_b = \{23, 187\}$. Now suppose Alice wants to send Bob the number (message) $M = 88$. For encryption, we would need to calculate $C = 88^7 \bmod 187$. Exploiting the properties of modular arithmetic, we can compute this as follows (this is needed for the efficient implementation of the encryption algorithm):

$$\begin{aligned} 88^7 \bmod 187 &= \left[(88^4 \bmod 187) \times (88^2 \bmod 187) \times (88^1 \bmod 187)\right] \bmod 187 \\ &= (88 \times 77 \times 132) \bmod 187 \\ &= 11 \end{aligned}$$

For decryption, Bob would need to calculate $M = 11^{23} \bmod 187$. This is computed as follows:

$$\begin{aligned} 11^{23} \bmod 187 &= \left[(11^1 \bmod 187) \times (11^2 \bmod 187) \times (11^4 \bmod 187) \times (11^{16} \bmod 187)\right] \bmod 187 \\ &= 88 \end{aligned}$$

The results are shown in figure 10.

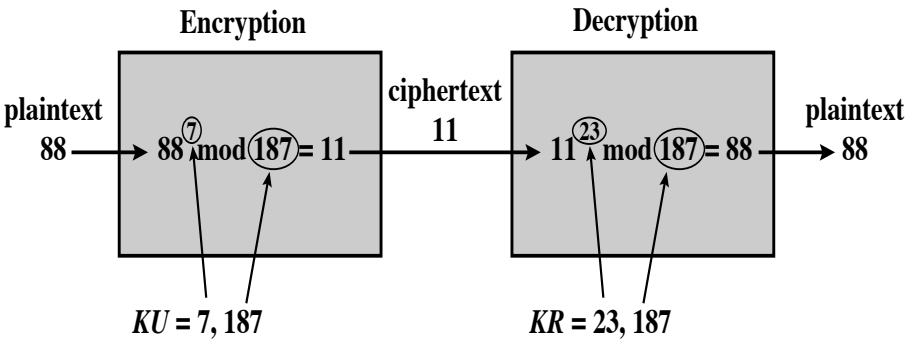

Figure 9.6 Example of RSA Algorithm

Figure 10: Sample example for the RSA Algorithm

Recommended Reading

1. Chapters 2 and 3 of Stallings [1] for a discussion of symmetric key cryptosystems. In particular, Section 3.1 of Stallings presents the simplified DES cryptosystem discussed in this lecture.

2. Chapter 9 of Stallings has a nice discussion on public-key cryptography, the Diffie-Hellman scheme, and the RSA algorithm.

3. All the figures in this lecture were taken from Chapters 2,3, and 9 of Stallings.

References

[1] W. Stallings, *Cryptography and Network Security: Principles and Practices*, 3rd edition, Prentice Hall, NJ, 2003.