## Stream Ciphers and RC4:

### *Stream Cipher Structure*

A typical stream cipher encrypts plaintext one byte at a time, although a stream cipher may be designed to operate on one bit at a time or on units larger than a byte at a time. Figure 10.1 is a representative diagram of stream cipher structure. In this structure a key is input to a pseudorandom bit generator that produces a stream of 8-bit numbers that are apparently random. That a pseudorandom stream is one that is unpredictable without knowledge of the input key. The output of the generator, called a **keystream**, is combined one byte at a time with the plaintext stream using the bitwise exclusive-OR (XOR) operation.

For example,

```
  11001100   plaintext
⊕ 01101100   key stream
  10100000   ciphertext
```

Decryption requires the use of the same pseudorandom sequence:

```
  10100000   ciphertext
⊕ 01101100   key stream
  11001100   plaintext
```

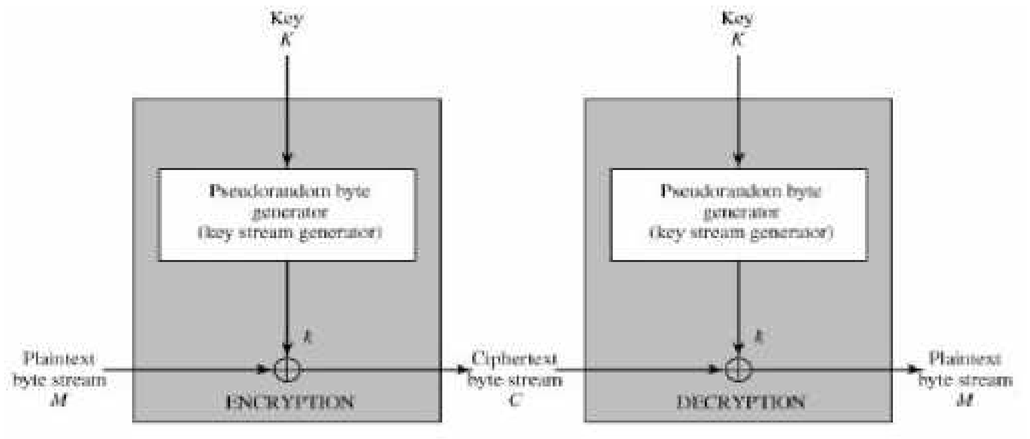

Key
K
Pseudorandom byte generator (key stream generator)
k
Plaintext byte stream M
ENCRYPTION
Ciphertext byte stream C
Key
K
Pseudorandom byte generator (key stream generator)
k
DECRYPTION
Plaintext byte stream M

**Figure (10.1) Stream Cipher Diagram**

The stream cipher is similar to the one-time pad discussed in lecture 2. The difference is that a onetime pad uses a genuine random number stream, whereas a stream cipher uses a pseudorandom number stream.

## The RC4 Algorithm

RC4 is a stream cipher. It is a variable key-size stream cipher with byte-oriented operations. The algorithm is based on the use of a random permutation.

Analysis shows that the period of the cipher is overwhelmingly likely to be greater than $10^{100}$.

The RC4 algorithm is remarkably simply and quite easy to explain. A variable-length key of from 1 to 256 bytes (8 to 2048 bits) is used to initialize a 256-byte state vector S, with elements S[0], S[1],..., S [255]. At all times, S contains a permutation of all 8-bit numbers from 0 through 255. For encryption and decryption, a byte *k* (see Figure 10.1) is generated from S by selecting one of the 255 entries in a systematic fashion. As each value of *k* is generated, the entries in S are once again permuted.

### *Initialization of S*

To begin, the entries of S are set equal to the values from 0 through 255 in ascending order; that is; S[0] = 0, S[1] = 1,..., S[255] = 255. A temporary vector, T, is also created. If the length of the key K is 256 bytes, then K is transferred to T. Otherwise, for a key of length *keylen* bytes, the first *keylen* elements of T are copied from K and then K is repeated as many times as necessary to fill out T.

```
/* Initialization */
for  i = 0 to 255 do
S[i] = i;
T[i] = K[i mod keylen];
```

Next we use T to produce the initial permutation of S. This involves starting with S[0] and going through to S[255], and, for each S[i], swapping S[i] with another byte in S according to a scheme dictated by T [i].

```
/* Initial Permutation of S */
j = 0;
for  i = 0 to 255 do
   j = (j + S[i] + T[i]) mod 256;
   Swap (S[i], S[j]);
```

Because the only operation on S is a swap, the only effect is a permutation. S still contains all the numbers from 0 through 255.

***Stream Generation***

Once the S vector is initialized, the input key is no longer used. Stream generation involves cycling through all the elements of S[i], and, for each S[i], swapping S[i] with another byte in S according to a scheme dictated by the current configuration of S. After S[255] is reached, the process continues, starting over again at S[0].

```
/* Stream Generation */
i, j = 0;
while (true)
   i = (i + 1) mod 256;
   j = (j + S[i]) mod 256;
   Swap (S[i], S[j]);
   t = (S[i] + S[j]) mod 256;
   k = S[t];
```

To encrypt, XOR the value *k* with the next byte of plaintext. To decrypt, XOR the value *k* with the next byte of ciphertext. Figure 10.2 illustrates the RC4 logic.

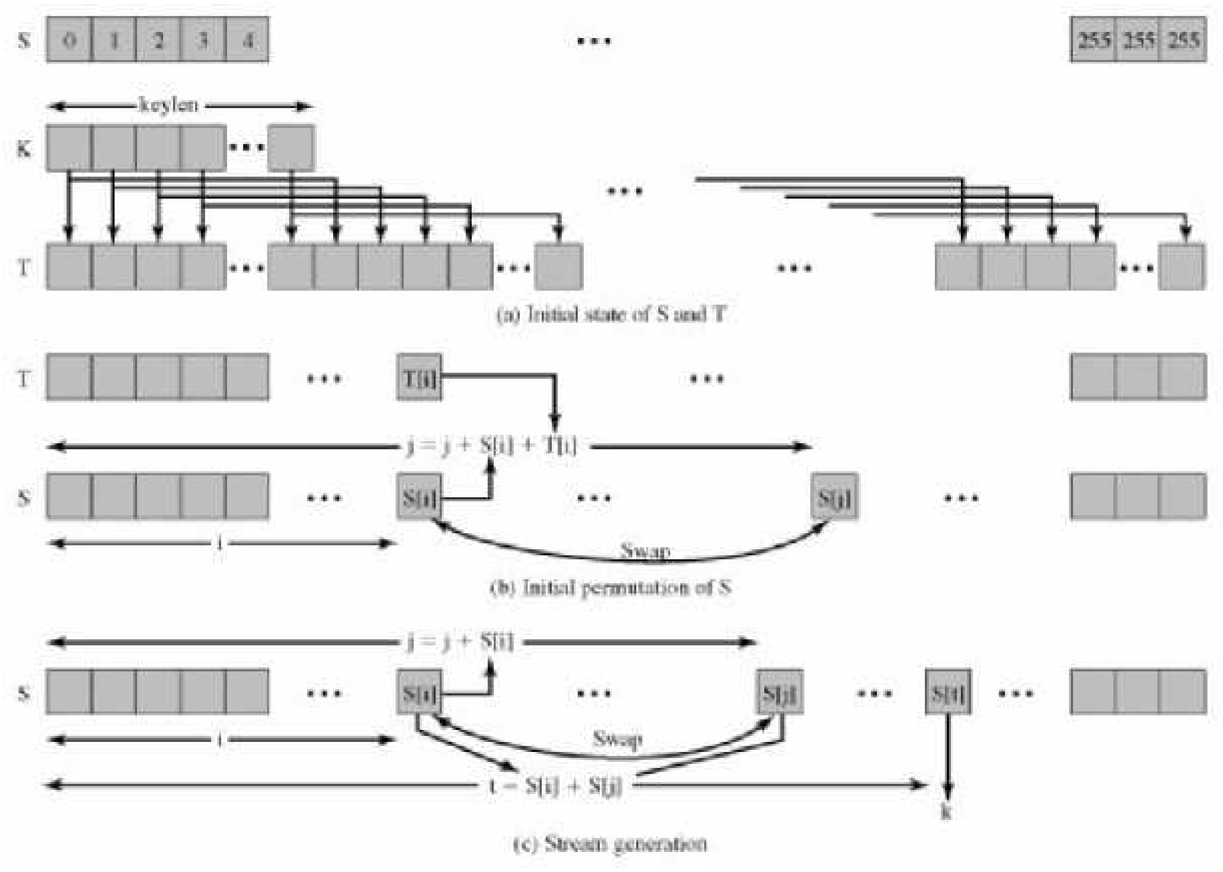


**Figure (10.2) RC4**

## Public key cryptography:

Asymmetric encryption is a form of cryptosystem in which encryption and decryption are performed using the different keys one a public key and one a private key. It is also known as public-key encryption.

Asymmetric encryption transforms plaintext into ciphertext using a one of two keys and an encryption algorithm. Using the paired key and a decryption algorithm, the plaintext is recovered from the ciphertext.

Asymmetric encryption can be used for confidentiality, authentication, or both. The most widely used public-key cryptosystem is RSA. The difficulty of attacking RSA is based on the difficulty of finding the prime factors of a composite number.

Public key algorithms are based on mathematical functions rather than on substitution and permutation. More important, public-key cryptography is asymmetric, involving the use of two separate keys, in contrast to symmetric encryption, which uses only one key.

The security of any encryption scheme depends on the length of the key and the computational work involved in breaking a cipher. There is nothing in principle about either symmetric or public-key encryption that makes one superior to another from the point of view of resisting cryptanalysis.

## Public-Key Cryptosystems

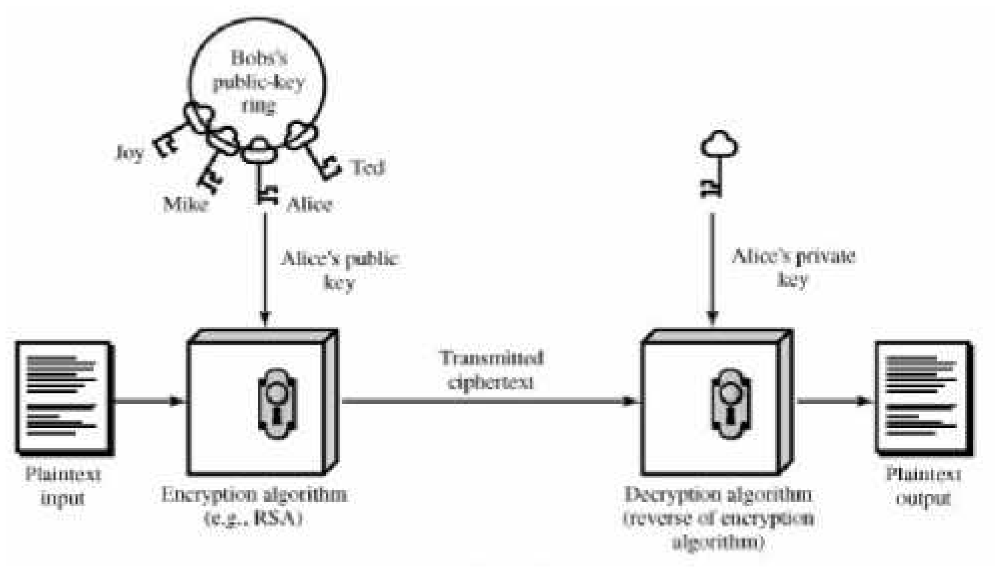
Asymmetric algorithms rely on one key for encryption and a different but related key for decryption.

These algorithms have the following important characteristic:

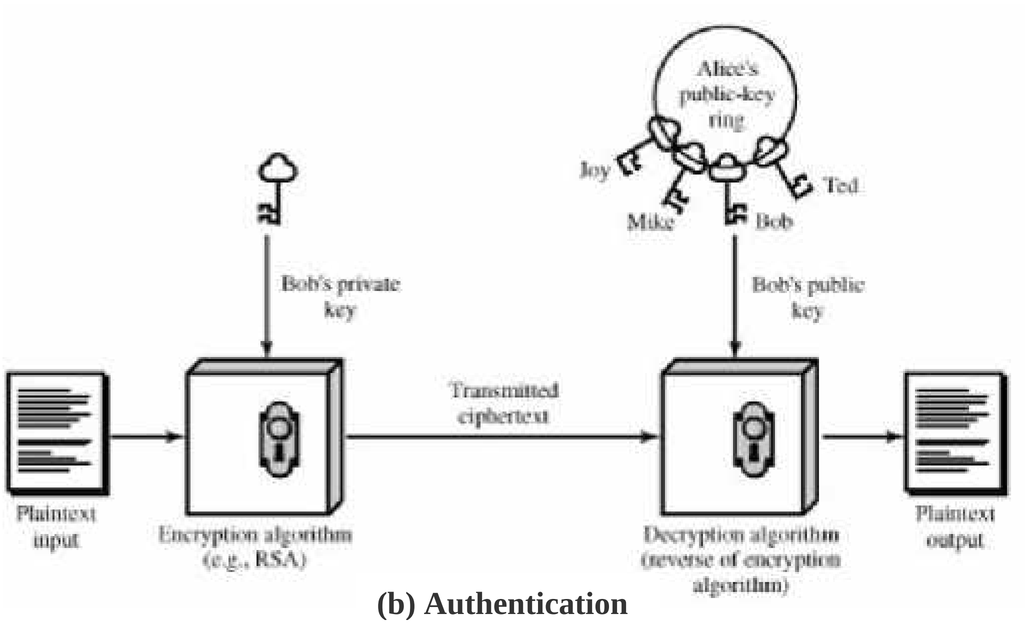
● It is computationally infeasible to determine the decryption key given only knowledge of the cryptographic algorithm and the encryption key.

In addition, some algorithms, such as RSA, also exhibit the following characteristic:

● Either of the two related keys can be used for encryption, with the other used for decryption. A public-key encryption scheme has six ingredients as shown on figure (10.3 a).



**(a) Encryption**



**(b) Authentication**

**Figure (10-3) Public key Encryption**

These categories are:

- **Plaintext:** This is the readable message or data that is fed into the algorithm as input.
- **Encryption algorithm:** The encryption algorithm performs various transformations on the plaintext.
- **Public and private keys:** 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 algorithm depend on the public or private key that is provided as input.
- **Ciphertext:** This is the scrambled message produced as output. It depends on the plaintext and the key. For a given message, two different keys will produce two different ciphertexts.
- **Decryption algorithm:** This algorithm accepts the ciphertext and the matching key and produces the original plaintext.

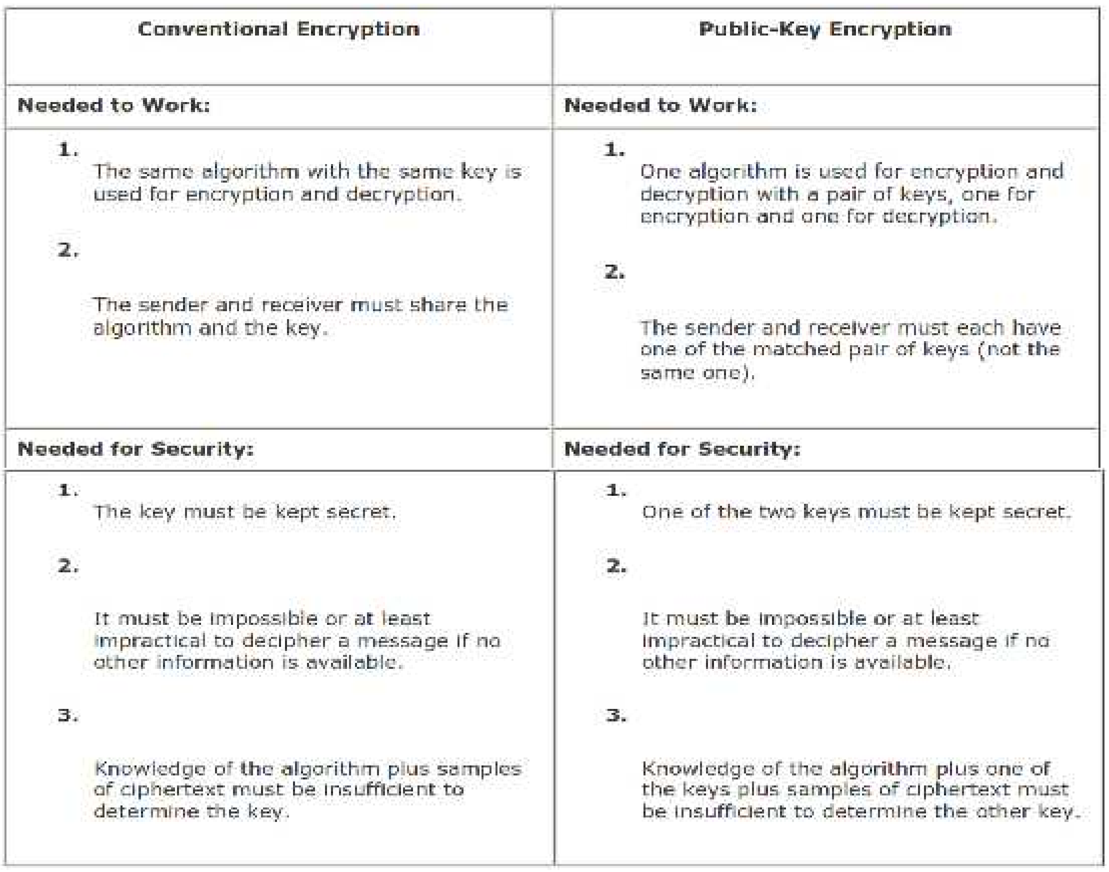
The essential steps are the following:

1. Each user generates a pair of keys to be used for the encryption and decryption of messages.
2. Each user places one of the two keys in a public register or other accessible file. This is the public key. The companion key is kept private. As Figure 10.3 suggests, each user maintains a collection of public keys obtained from others.
3. If Bob wishes to send a confidential message to Alice, Bob encrypts the message using Alice's public key.
4. When Alice receives the message, she decrypts it using her private key. No other recipient can decrypt the message because only Alice knows Alice's private key.

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 user's private key remains protected and secret, incoming communication is secure. At any time, a system can change its private key and publish the companion public key to replace its old public key.

Table 10.1 summarizes some of the important aspects of symmetric and public-key encryption. To discriminate between the two, we refer to the key used in symmetric encryption as a **secret key**. The two keys used for asymmetric encryption are referred to as the **public key** and the **private key**. Invariably, the private key is kept secret, but it is referred to as a private key rather than a secret key to avoid confusion with symmetric encryption.

**Table 10.1 Conventional and Public-Key Encryption**



| Conventional Encryption | Public-Key Encryption |
|---|---|
| **Needed to Work:** | **Needed to Work:** |
| 1. The same algorithm with the same key is used for encryption and decryption.<br>2. The sender and receiver must share the algorithm and the key. | 1. One algorithm is used for encryption and decryption with a pair of keys, one for encryption and one for decryption.<br>2. The sender and receiver must each have one of the matched pair of keys (not the same one). |
| **Needed for Security:** | **Needed for Security:** |
| 1. The key must be kept secret.<br>2. It must be impossible or at least impractical to decipher a message if no other information is available.<br>3. Knowledge of the algorithm plus samples of ciphertext must be insufficient to determine the key. | 1. One of the two keys must be kept secret.<br>2. It must be impossible or at least impractical to decipher a message if no other information is available.<br>3. Knowledge of the algorithm plus one of the keys plus samples of ciphertext must be insufficient to determine the other key. |

Let us take a closer look at the essential elements of a public-key encryption scheme, using Figure 10.4.

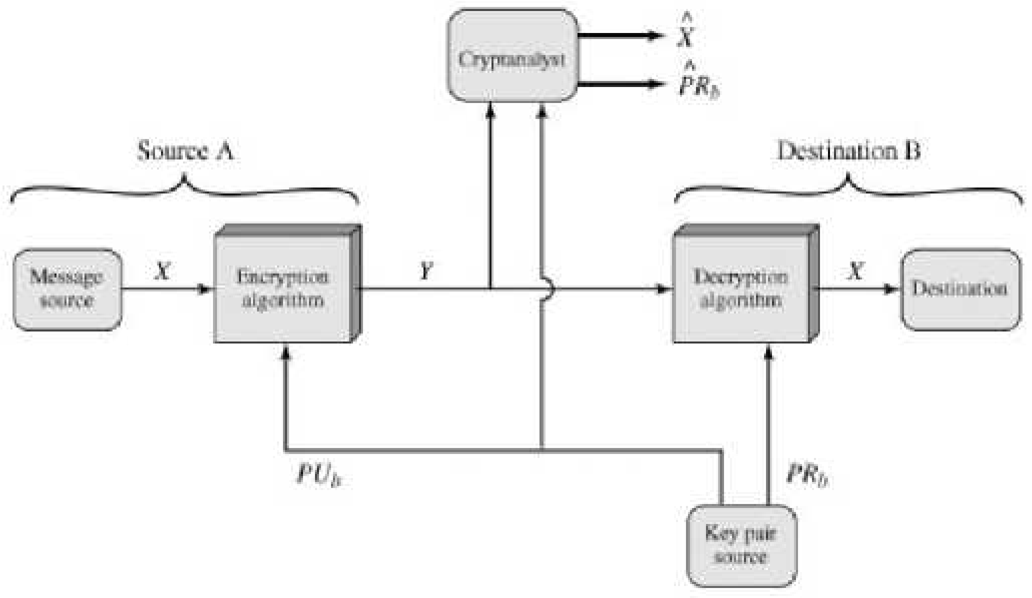


**Figure 10.4. Public key Cryptosystem:Secrecy**

There is some source A that produces a message in plaintext, $X = [X_1, X_2, \ldots, X_M,]$. The $M$ elements of $X$ are letters in some finite alphabet. The message is intended for destination B. B generates a related pair of keys: a public key, $PU_b$ , and a private key, $PR_b$ . $PR_b$ is known only to B, whereas $PU_b$ is publicly available and therefore accessible by A.

With the message $X$ and the encryption key $PU_b$ as input, A forms the ciphertext $Y = [Y_1, Y_2, \ldots, Y_N]$:

$$Y = E(PU_b, X)$$

The intended receiver, in possession of the matching private key, is able to invert the transformation:

$$X = D(PR_b, Y)$$

An adversary, observing $Y$ and having access to $PU_b$ but not having access to $PR_b$ or $X$, must attempt to recover $X$ and/or $PR_b$ .It is assumed that the adversary does have knowledge of the encryption (E) and decryption (D) algorithms. If the adversary is interested only in this particular message, then the focus of effort is to recover *X*, by generating a plaintext estimate X .

Often, however, the adversary is interested in being able to read future messages as well, in which case an attempt is made to recover *PRb* by generating an estimate PRb.

We mentioned earlier that either of the two related keys can be used for encryption, with the other being used for decryption. This enables a rather different cryptographic scheme to be implemented. Whereas the scheme illustrated in Figure 10.4 provides confidentiality, Figures 10.3b and 10.5 show the use of public-key encryption to provide authentication:

$$Y = E(PRa, X)$$

$$Y = D(PUa, X)$$

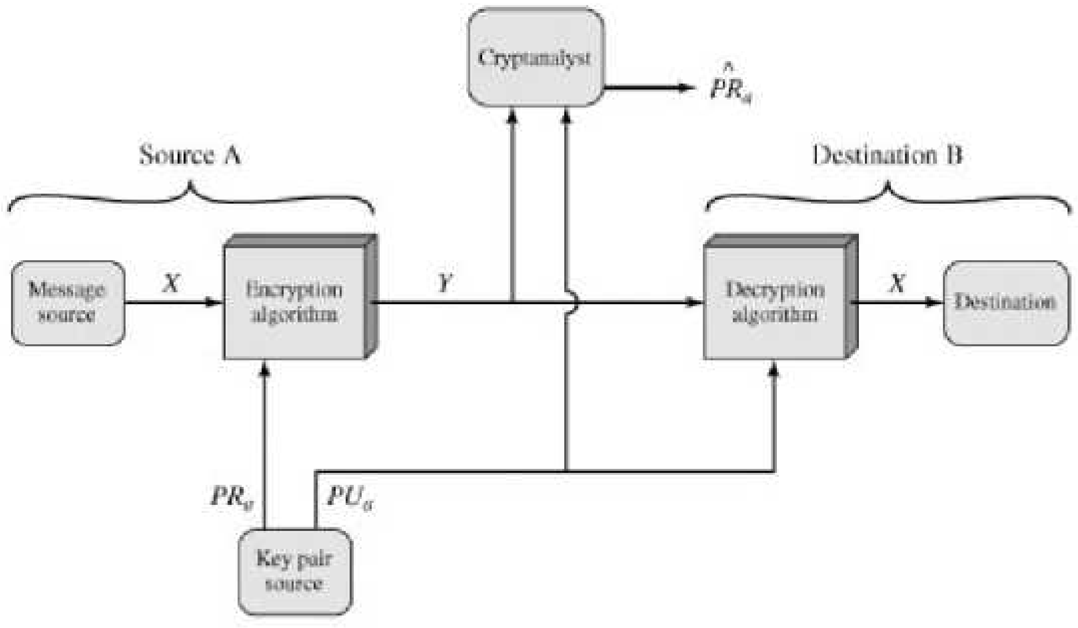


**Figure 10.4 Public-Key Cryptosystem: Authentication**

In this case, A prepares a message to B and encrypts it using A's private key before transmitting it. B can decrypt the message using A's public key. Because the message was encrypted using A's private key, only A could have prepared the message.

It is, however, possible to provide both the authentication function and confidentiality by a double use of the public-key scheme (Figure 10.6):

$$Z = \mathrm{E}(PUb, \mathrm{E}(PRa, X))$$

$$X = \mathrm{D}(PUa, \mathrm{D}(PRb, Z))$$

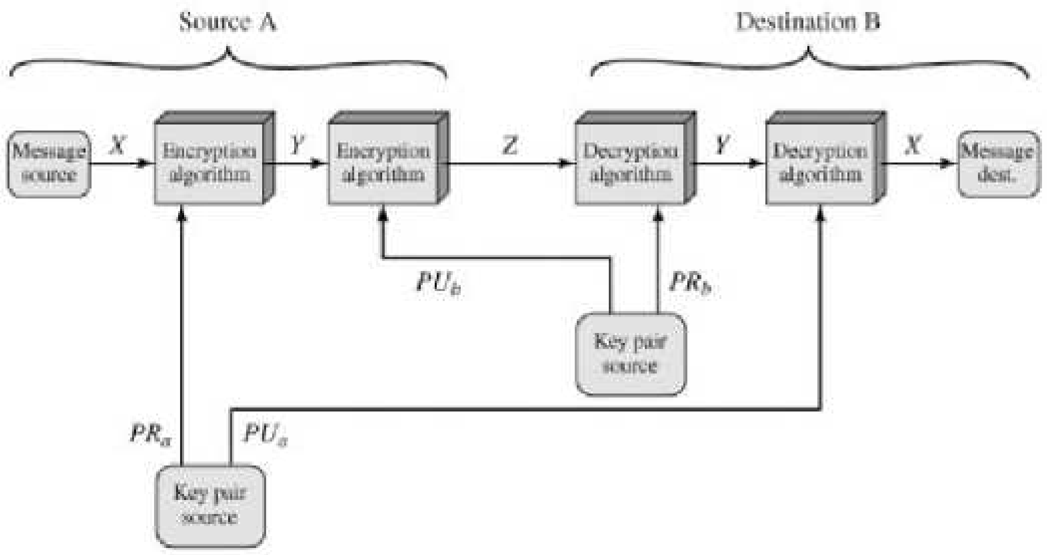


**Figure 9.4. Public-Key Cryptosystem: Authentication and Secrecy**

In this case, we begin as before by encrypting a message, using the sender's private key. This provides the digital signature. Next, we encrypt again, using the receiver's public key. The final ciphertext can be decrypted only by the intended receiver, who alone has the matching private key. Thus, confidentiality is provided. The disadvantage of this approach is that the public-key algorithm, which is complex, must be exercised four times rather than two in each communication.