Chapter 4. Encryption and Randomness©

Encryption is apparently the earliest cryptographic primitive. The goal of encryption is to hide some secret text, which we call *plaintext*, by encoding it into another text, which we call *ciphertext*. The ciphertext supposedly does not provide any information about the plaintext, except to the intended recipients, which know some secret. We refer to the schemes (algorithms) used for encryption and decryption as *cryptosystems*[1]. In very early cryptosystems, the secret was the algorithm used for decryption, i.e. reversing the encoding and retrieving the plaintext from the ciphertext. However, following Kerckhoffs' principle, we only consider cryptosystems where the adversary knows the encryption and decryption algorithms, and the intended recipients use *secret decryption key* (or just secret key) to decipher the ciphertext back to plaintext. The encoding of the plaintext into ciphertext also uses a key, the encryption key.

The encryption key may be identical to the decryption key, in which case it is a *shared secret key* (or just shared or secret key) used for both encryption and decryption. We call such cryptosystems *ciphers* or *shared key cryptosystems*[2] and illustrate them in Figure 4.1. Alternatively, encryption may use a *public encryption key* (or just public key), from which it is infeasible for the adversary to compute the secret decryption key. We call these cryptosystems *public key cryptosystems*[3] and illustrate them in Figure 4.2.

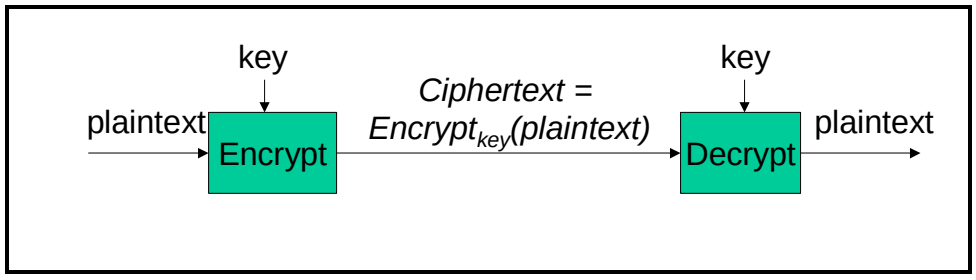

Figure 4.1 Cipher (Shared Key Cryptosystem)

[1] Some authors use the terms `cipher` and `encryption scheme` instead of `cryptosystem`.
[2] Some authors use the term `symmetric cryptosystem`.
[3] Some authors use the term `asymmetric cryptosystem`.

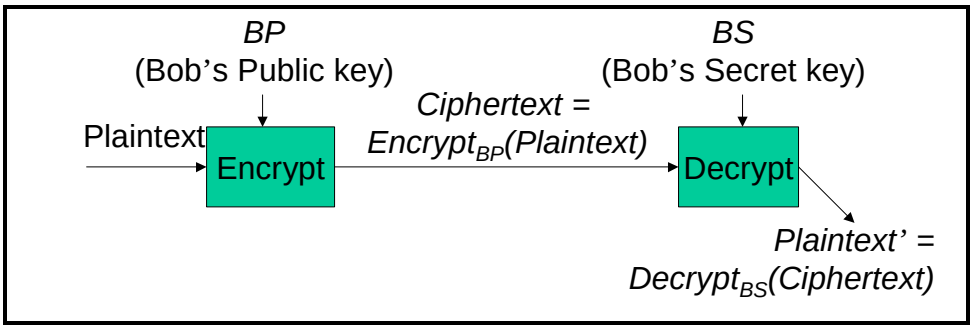

Figure 4.2: Public Key Cryptosystem

4.1. Defining security – attack models and the indistinguishability test

An intuitive, fuzzy definition of security for encryption schemes may be `*without access to the secret key, it is infeasible for an attacker to learn anything meaningful about the plaintext given the ciphertext*`. This fuzzy definition leaves several issues vague; in particular, what constitutes `meaningful information` that the attacker should not learn? What is the attack model - what information and abilities are available to the attacker?

The definition of `meaningful information` depends, of course, on the specific application. However, since we are unaware of the specific application and its requirements, and following the prudence principle, we prefer to use the most conservative definition – except if this cause considerable overhead. The most conservative definition seems to be the distinguishability test, which we illustrate in Figure 4.3.

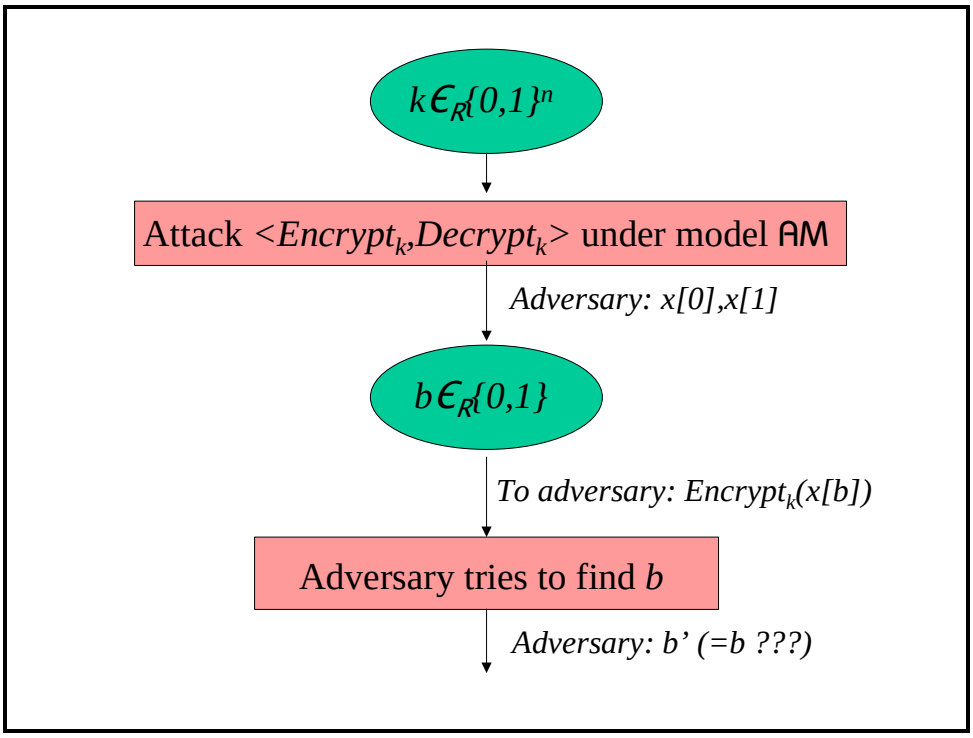

Figure 4.3: Distinguishability Test

Definition 1 Define the *distinguishability test* for an arbitrary attack model AM, as follows. First, the attacker attacks under AM. Then, the attacker chooses two different plaintext messages, *x[0]* and *x[1]*. The attacker is given the encryption of either *x[0]* or *x[1]*, each with probability half. The attacker has to output *b* if it received encryption of *x[b]*. The encryption scheme *ensures indistinguishability under attack model AM*, if the success probability is bounded by half, plus a negligible value (a function which diminishes faster than any polynomial in the length of the key).

We now discuss the attack model, i.e. the abilities we assume that attacker (adversary) may have. Following Kerckhoffs' principle, we always assume that the attacker knows the algorithms in use and the probability distribution of the inputs (plaintext); we may also make some assumptions on this probability distribution. Beyond that, we consider several possible attack models. The most commonly used attack models are:

- *Known plaintext distribution attack* (also called *ciphertext-only*): Adversary knows the encryptions (ciphertext) corresponding to a tiny, randomly chosen fraction of the plaintext messages. For example, when encrypting text in English letter-by-letter, the attacker may have statistics on the probability of different letters and their combinations.
- *Known plaintext attack:* Same as above, but the adversary is also given the randomly chosen plaintext values. A variant here is whether the adversary is allowed to choose the same values for *x* or *y* in the indistinguishability test.
- *Chosen plaintext attack:* Same as above, but the adversary chooses the plaintext values. Security against chosen plaintext implies security against known plaintext.

Also, notice that when the attacker has the (public) encryption key, as we normally assume for public key cryptosystems, this attack is always possible.

- *Known / Chosen plaintext attack, distinguishing unused messages:* Same as the corresponding attack above, but the adversary is not allowed to choose *x,y* (to distinguish) which were one of the plaintext messages.
- *Chosen ciphertext attack:* Same as one of the attacks above, but in addition the adversary can choose a tiny fraction of all possible ciphertext messages and receive the respective plaintext (decryption). Intuitively, for an attack in this model to be successful, the attacker must then decrypt a different ciphertext message. The distinguishability test ensures this by requiring the adversary to complete the chosen ciphertext attack before selecting the plaintext to be distinguished, *x* and *y*.
- *Related key attacks:* Same as one of the attacks above, but in addition the adversary knows the encryption of (the same or related) plaintext messages under two or more secret keys which are unknown, but some relationship between them is known. This is really a family of attacks, depending on the relationship between the keys. Such attacks are mostly relevant where the attacker may gain physical access to a tamper-resistant device containing the secret key, and may be able to cause a change in the key, e.g. to cause a random error which is likely to impact a single bit.

Prudent cryptographic design attempts to design encryption schemes resilient to the strongest attack models, while designing the protocols and system to limit the attacks to the weakest models possible, e.g. prevent chosen plaintext or ciphertext attacks. For example, by checking for incorrectly formatted decryptions, or (better) encrypting messages with an integrity code (e.g. CRC) and checking it in decryption, we can detect attempts to provide chosen ciphertext and raise alert, making this attack unattractive.

The definition of security further depends on the type of cryptosystem. We next discuss, in separate subsections, shared-key cryptosystem (ciphers) and public-key cryptosystems.

4.2. Shared Key Cryptosystem (Ciphers)

We consider three different types of shared key cryptosystems:

- *Deterministic block ciphers*, where the encryption and decryption algorithms are stateless, and their only inputs are the key and the plaintext (respectively ciphertext) messages.
- *Randomized block ciphers*, where encryption algorithm also has random input.
- *Stream ciphers*, where encryption and decryption are stateful processes, applied sequentially to (often short) blocks of plaintext (respectively, ciphertext).

4.2.1. Deterministic block cipher

A *deterministic block cipher* is a shared key cryptosystem that has bounded sets of plaintext and ciphertext messages, and two keyed functions Encrypt and Decrypt, such that for every plaintext message *p* holds:

$$p = Decrypt_k(Encrypt_k(p))$$

Equation 4.1

We can assume that the plaintext and ciphertext sets are both the set of all binary strings of some fixed length *l*. This makes Encrypt and Decrypt, for any given key, into permutations over this set.

All deterministic block ciphers are vulnerable even under the weakest attack model, known plaintext distribution, for scenarios where the probability distribution of the plaintext is significantly biased, or where the number of possible plaintexts is limited. Therefore, we define the security of a deterministic block cipher relative to the probability distribution of the plaintext.

In particular, we assume that the length *l* of the plaintext is sufficiently large, to prevent the attacker from seeing a significant fraction of all possible ciphertexts or identifying repetitions. For example, consider using short length e.g. *l=5*, allowing us to encode 32 symbols. This allows us to encrypt messages in lowercase Latin alphabet, by replacing each letter with its permutation. However, the security provided by this cipher is very weak. Specifically, it is sufficient for the attacker to know that the plaintext is text in English, in order to use well-known statistics and find out the permutation used.

Therefore, we must use longer blocks, to prevent the attacker from using statistics or collections of plaintext-ciphertext pairs. In practice, most systems, e.g. the Data Encryption Standard [DES], have block length of at least 64 bits. For simplicity, we assume henceforth that the block length of deterministic block ciphers is the same as the security parameter, i.e. *l=n*.

4.2.2. Simple substitution cipher (random permutation)

Denote by $\mathcal{N}$ the set of all permutations of binary strings of length *n*. The *simple substitution cipher* of length *n* is defined by picking as key *k* a random element (permutation) in $\mathcal{N}$. Encryption is simply application of the key (permutation) *k* to the plaintext, and decryption is simply the application of the inverse permutation. Clearly:

$$p = k^{-1}(k(p))$$

Equation 4.2

A simple substitution cipher is clearly secure if used to encrypt a single plaintext block (message of length *n*). In fact, in this case it is unconditionally secure.

If the block length is sufficient (as we assumed), and the plaintext distribution is uniform, then it is easy to see that the simple substitution cipher is (computationally) secure even for repeated encryptions.

Claim 1 A simple substitution cipher ensures indistinguishability for unused plaintext, under chosen plaintext attack, for large plaintext space.

Argument: Let the attacker pick f plaintext messages P_A and receive their encryptions (ciphertext), C_A. Next, the attacker picks $x,y \notin P_A$ and receives the encryption $c \notin C_A$ of either x or y (with probability half for each). There are exactly $(2^n\text{-}f\text{-}1)!$ permutations which map x to c and P_A to C_A, and exactly the same mapping y to c. By symmetry, the attacker cannot distinguish between these. ♦

The simple substitution cipher remains secure even if the plaintext distribution is only `fairly` uniform, since, as the attacker is computationally-limited, we assume that $f << 2^n$. It is also secure for chosen ciphertext attacks.

The disadvantage of the simple substitution cipher is its key length, which is the index of the randomly chosen permutation in N. The number of keys (permutations) is $2^n!$, which grows very fast (in n); even the minimal key length, $lg_2(2^n!)$, grows very fast. Even for very short (and therefore insecure) block length e.g. $l=5$, the number of keys is huge, $2^5!=32! \approx 2.63*10^{35}$, and requires keys of roughly 35 digits or 118 bits.

Therefore, we rarely use simple substitution ciphers in practice. Instead, `practical` substitution ciphers often attempt to be indistinguishable from simple substitution cipher. If we prove (or assume) that a cipher is indistinguishable from simple substitution ciphers, then its security follows immediately from Claim 1 above. We next discuss such ciphers.

4.2.3. Pseudo-random permutations and functions

Bellare, Kilian and Rogaway [BKR94] recognized that practical block ciphers are good candidates for a very powerful cryptographic functions, namely pseudo-random permutations.

Definition 2 A *pseudo-random permutation* is a collection of pairs of a permutation and its inverse: $\{<E_k,D_k>\}$, with domain and range $\{0,1\}^n$ such that:

- *Efficient evaluation and inversion:* For a given key[4] k, the pseudo-random permutation can be computed efficiently, i.e. given any message m it is easy to compute both $E_k(m)$ and $D_k(m)$.
- *Pseudo-randomness:* a computationally-bounded adversary cannot distinguish between E_k and a random permutation on $\{0,1\}^n$, for randomly chosen key k

Namely, if k is unknown to the adversary, then pseudo-random permutations are indistinguishable from random permutations, even if the attacker has `black box` access to E_k (chosen plaintext attack). Hence,

Claim 2 A pseudo-random permutation ensures indistinguishability under chosen plaintext attack, for uniformly distributed plaintext. ♦

[4] . The index k is the *key* (or the *seed*) of the permutation (more precisely, of the pair of permutation and its inverse).

A *super pseudo-random permutation* is a pseudo-random permutation which remains indistinguishable from a random permutation, even when the attacker is allowed `black box` access to D_k as well as to E_k .

Claim 3 A super pseudo-random permutation ensures indistinguishability under chosen ciphertext attack, for uniformly distributed plaintext. ♦

Following [BKR94], we believe that many designers of well-known practical deterministic block ciphers tried, in fact, to make them (super) pseudo-random permutation. Similarly, we believe that cryptanalysts considered any proof that a specific block cipher is not a (super) pseudo-random permutation, as success in breaking the block cipher (at least after [BKR94]!). In particular, notice that the precise definition and construction of a pseudo-random permutation by Luby and Rackoff in [LR88], as well as the follow-up work by Naor and Reingold [NR99], is based on the Feistel cipher, used extensively by [DES] and other block ciphers, as described in the next section. Both constructions create a pseudo-random permutation from a pseudo-random function, which Goldreich, Goldwasser and Micali defined and implemented from the very weak cryptographic primitive of one-way functions [GGM86].

Pseudo-random functions and permutations have several applications, in addition to their use for block ciphers. In particular, a very important use for pseudo-random functions and permutations is as a mechanism to derive multiple secret values (e.g. keys) $v_1,v_2,\ldots$ from a single secret value v. The derived values are result of pseudo-random function or permutation with key v computed over different values, e.g. $v_i=f_v(i)$. Whenever you need secret, pseudo-random values (e.g. shared keys) for more than one purpose, e.g. for encryption and for authentication, use the above mechanism (or similar) to derive independent keys from a single shared secret value.

Claim 4 Assume a system is secure when some parties share secret values $v_1,v_2,\ldots$. Then this system is secure, against computationally bounded adversary, also if the parties share only one secret value v and use $v_i=f_v(i)$, where f is a pseudo-random permutation or function.

4.2.4. Practical Deterministic Block Ciphers – Feistel, DES and AES

Deterministic block ciphers are the most widely used encryption technique in practice. We now briefly describe popular designs, in particular, the Data Encryption Standard [DES] and the Advanced Encryption Standard [AES].

Most practical block ciphers use iterated, multiple-round design. Such designs involve the sequential application of an internal *round function*, plus possibly some special additional processing of the input to the first round and the output from the last round. The round function operates on the output from previous round, and a *subkey* – a function over the key to the cipher and over the round number. Figure 4.4 illustrates this structure.

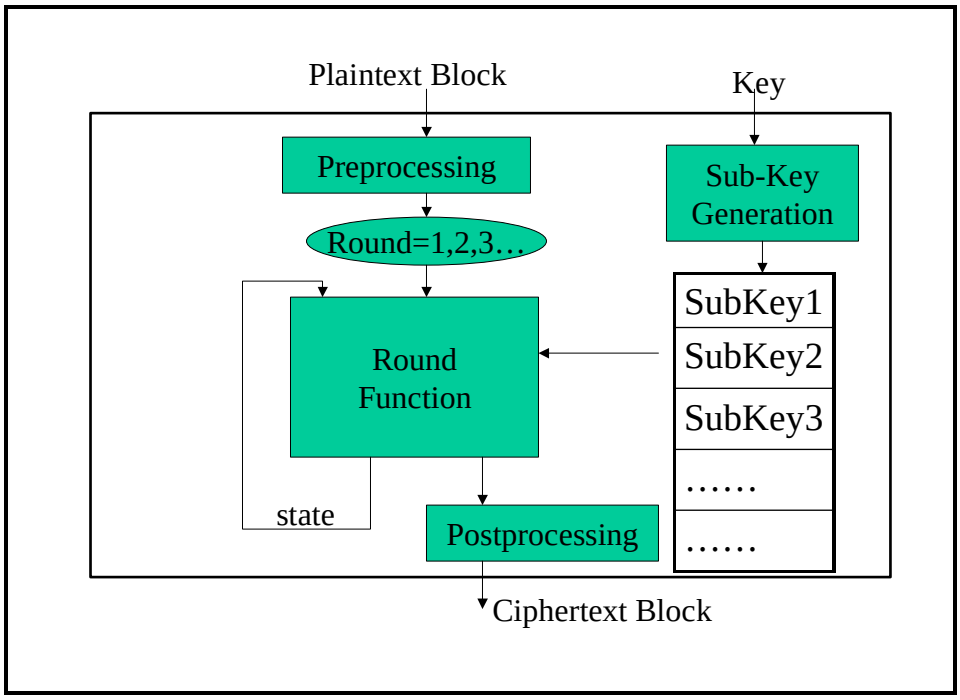

Figure 4.4: Iterative block cipher

A particularly common design for a round function is the *Feistel cipher* (see Figure 4.5). In the Feistel cipher [F73], the input to each round, say round t, is broken into two parts, L_t and R_t. The outputs are defined as $L_{t+1}=R_t$ and $R_{t+1}=L_t+f(R_t,K_t)$, where f is a `single round cipher function` and K_t is the subkey of round t. As mentioned above, the Feistel construction is useful also to construct pseudo-random permutations (from pseudo-random functions).

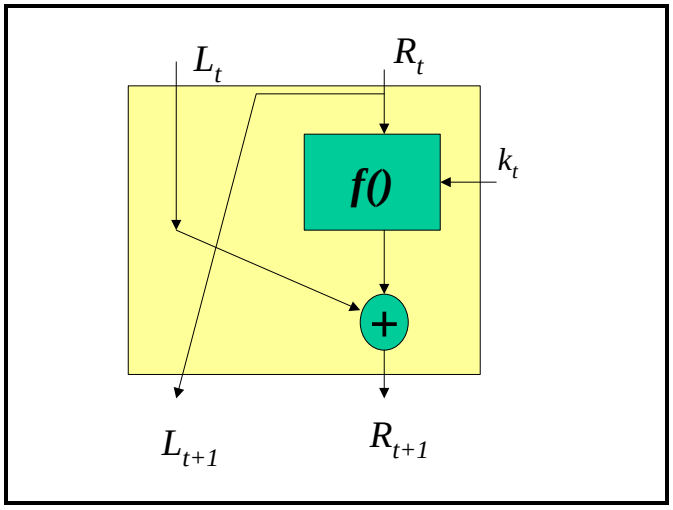

Figure 4.5: Feistel Round Function Design

DES is a 16-round Feistel cipher whose input and output block sizes are 64 bits, and a key of 56 bits (or 64 bits out of which 8 bits are used for parity check). DES was designed in

the late 1970's, by a USA government standards body, NIST, and was often criticized as potentially containing an intentional weakness (e.g. the use of only 56 bit key) to allow government eavesdropping; one of the reasons for this criticism was the fact that the design considerations were not published. In spite of these criticisms, many systems used DES. Substantial cryptoanalysis effort revealed no trapdoors or substantial shortcut weaknesses. Indeed, DES appears relatively well protected even against differential and linear cryptoanalysis methods, published only in the 90's; some of the designers claimed that they were aware of these attacks (justifying the decision not to publish the design considerations). However, the limited key size eventually made DES vulnerable to exhaustive, `brute-force` key guessing attacks.

Some systems still use a variant of DES called triple-DES, also defined in [DES], which involves three applications of DES with two keys. For compatibility with `regular` DES, the triple-DES design uses DES encryption for the first and third application and DES decryption for the second; this makes triple-DES with two identical keys equivalent to DES with the same key.

With triple-DES, the key length is 112bits, which seems sufficient to protect against exhaustive key search for many applications. However, notice that since the block size remains 64 bits, the pseudo-randomness property may not be secure, and indeed for some applications triple-DES is not secure enough.

A more secure solution, for new systems, is to use the new Advanced Encryption Standard [AES], defined recently by NIST; this is the official name of the Rijndael cipher [DR01], which was selected among 18 proposals submitted to a lengthy, open evaluation process led by NIST. The goals of AES are to improve security and efficiency compared to DES; DES is quite efficient in hardware, but not very efficient in software.

The Rijndael cipher has a variable block length and key length. The specifications cover keys with a length of 128, 192, or 256 bits, and blocks with length of 128, 192 or 256 bits (for AES, the block size is always 128 bits). Rijndael allows efficient implementations in both software and hardware.

4.2.5. Modes of Operation

Deterministic block ciphers have two significant limitations:

- Being completely deterministic, the encryption of fixed blocks of plaintext results in corresponding fixed blocks of ciphertext. The adversary may exploit this, e.g. when there are some frequently repeating blocks of plaintext.
- They operate over fixed-size, *n*-bit plaintext blocks.

We deal with these limitations by using *modes of operation* defining the use of the ciphers for longer messages and/or introducing randomness. The most commonly used modes of operations, defined in [DES], for producing the sequence of ciphertext blocks *c[1],c[2],…* from the plaintext blocks *p[1],p[2],…* are as follows:

- *Electronic code book (ECB)* mode: encrypt each plaintext block separately, i.e. for every *j* holds $c[j]=E_k(p[j])$. This mode is trivial – in fact, it is the same as the block cipher, and does not add any security. It is called a `mode of operation` just so that every way of using the cipher is called `mode of operation`.
- *Cipher Block Chaining (CBC) mode:* use an additional input of *n*-bit called *Initialization Vector (IV)*, and let $c[0]=IV$. The ciphertext is produced by computing $c[j]=E_k(c[j-1] \oplus p[j])$. The IV should be changed with every message (or at least frequently). See Figure 4.6.
- *Cipher Feedback (CFB) mode:* again use IV and let $c[0]=IV$. The ciphertext is produced by computing $c[j]=E_k(c[j-1]) \oplus p[j]$.
- *Output Feedback (OFB) mode:* again use IV, but use also an auxiliary sequence of blocks $o[0],o[1],o[2]\ldots$with $o[0]=IV$. The ciphertext is produced by computing $o[j]=E_k(o[j-1])$ and $c[j]=o[j] \oplus p[j]$. See Figure 4.7.

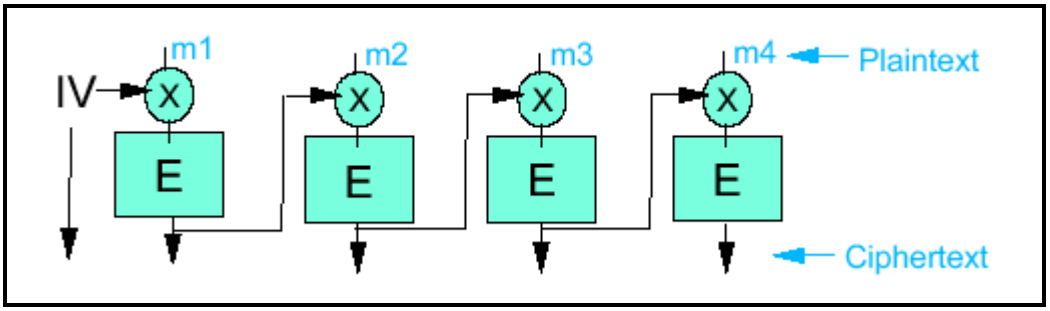

Figure 4.6: Cipher Block Chaining (CBC) mode

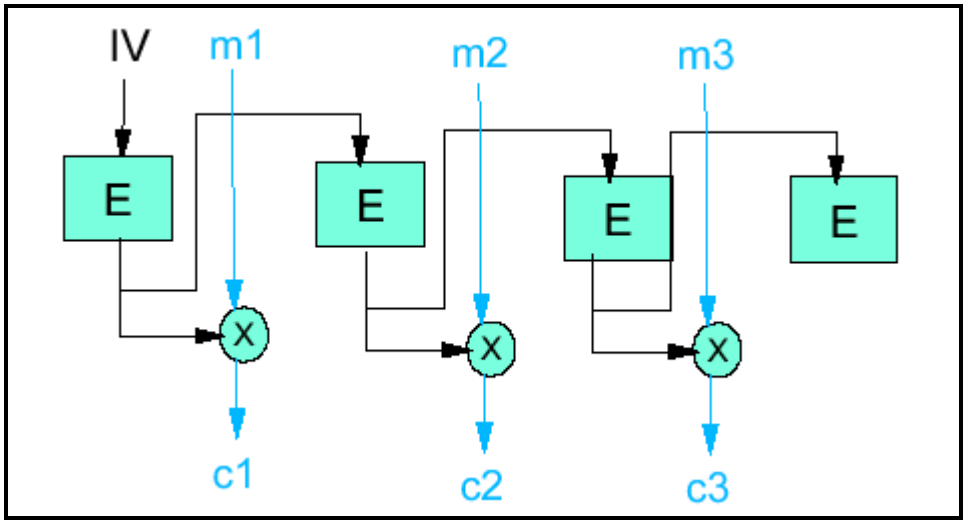

Figure 4.7: Output Feedback (OFB) mode

4.2.6. Randomized Block Ciphers

Deterministic block ciphers can only be secure for specific distributions of the plaintext. In many cases, the designer cannot know the probability distribution of the plaintext in advance, or the probability distribution is such that a deterministic block cipher is not secure. For example, if there are only two plaintext messages, say "buy" and "sell", then the attacker can quickly learn to identify the deterministic encryption and identify the plaintext. It is desirable to ensure security regardless of the probability distribution of the plaintext, and in particular maintain secrecy even if there are only two plaintext messages. We can achieve this by adding randomization as part of the encryption process, using a *randomized cryptosystem*. In this section, we discuss *randomized block ciphers*, and in the following sections, we explain how to apply randomization for stream ciphers and public key cryptosystems.

Randomized block ciphers have three inputs: the plaintext, the key, and uniformly distributed random string (coin tosses). The random input is combined with the plaintext during the encryption process, therefore the resulting ciphertext is longer than the plaintext; this overhead is acceptable in most applications. By combining the plaintext with the random inputs, randomized block ciphers attempt to ensure indistinguishability (secrecy), regardless of the plaintext distribution.

The `modes of operation` defined in [DES] provide simple and secure constructions for randomized block ciphers, when used with random Initial Vector (IV). In particular, given a deterministic block cipher $<E_k,D_k>$ with block length n, for plaintext of length i, to encrypt a message $x \in \{0,1\}^n$ with key k we randomly select a random string of the same block length, $r \in_R \{0,1\}^n$, and the ciphertext is the pair:

- Using the *Cipher Block Chaining (CBC) Mode:* $<r,E_k(x \oplus r)>$
- Using the *Output Feedback (OFB) Mode:* $<r,x \oplus E_k(r)>$

The symbol $\oplus$ denotes bit-wise exclusive-or operation. Decryption simply reverses this process (using D_k for CBC, or again E_k for OFB).

Claim 5 If $<E_k,D_k>$ is a pseudo-random permutation, then $<r,E_k(x \oplus r)>$ ensures indistinguishability under chosen plaintext attack.

Claim 6 If E_k is a pseudo-random permutation (or function), then $<r,x \oplus E_k(r)>$ ensures indistinguishability under chosen plaintext attack.

However, these constructions (and modes) are insecure against chosen ciphertext attack; see Exercise 7. Since these are modes are widely used in practice, we conclude that it is very important to design protocols and applications so that chosen ciphertext attacks will not be feasible.

4.2.7. Stream Ciphers and Pseudo-Random Generators

Randomized ciphers avoid assumptions about the probability distribution of their inputs, by `randomizing` their inputs. However, in the constructions above, the block size must

remain large, to make it almost certain that the encryption will use each input to the pseudo-random function or permutation only once. This is since the attacker is aware of the randomization.

We could consider the randomization bits as part of the secret key. This can allow us to use very short blocks. Taking this to the most extreme, we can encrypt the plaintext bit by bit, by exclusive-or with a secret, uniformly-random bit sequence serving as a key. Each bit of the key must be used with only one bit of the secret, hence we often refer to this scheme, illustrated in Figure 4.8, as *one time pad*.

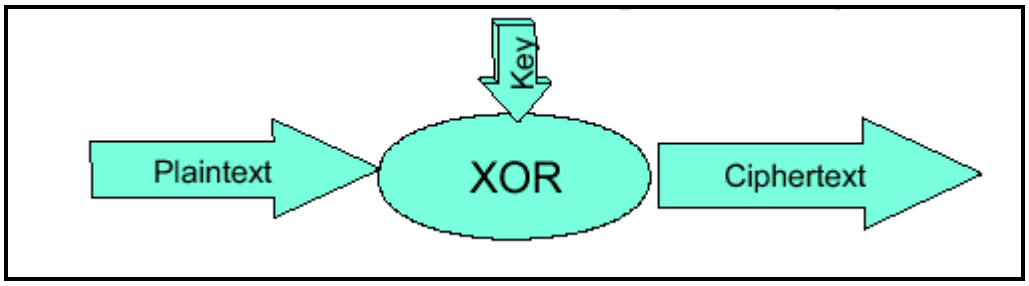

Figure 4.8: One Time Pad

Shannon [S49] showed that one time pad ensures unconditional security (secrecy). Namely, for any probability distribution of the plaintext (known to the attacker), the probability of any plaintext bit to be '1' does not change by knowing that the corresponding ciphertext bit is '1'. (See Chapter 2.) Therefore, even an adversary with unlimited computational abilities, cannot learn anything from the ciphertext.

Unfortunately, one time pad requires the parties to share a secret key of the same length as the plaintext. For most practical applications, this is unfeasible. One solution is to use a pseudo-random bit sequence as key, i.e. a bit sequence that is indistinguishable from uniformly distributed random sequence.

A *pseudo-random generator* is an algorithm with input $seed \in \{0,1\}^n$ and whose output is an infinite bit sequence, which is indistinguishable from a uniformly-distributed random bit sequence (and hence called *pseudo-random bit sequence*).

We use the term *pseudo-random generator based stream cipher* for such cryptosystems, i.e. one-time-pad like cryptosystems with a pseudorandom sequence instead of truly random key (see Figure 4.9). The Output Feedback (OFB) mode of operation presented in section 4.2.5 above is an example of a pseudo-random generator based stream cipher (and of one possible construction of a pseudo-random generator, using block cipher).

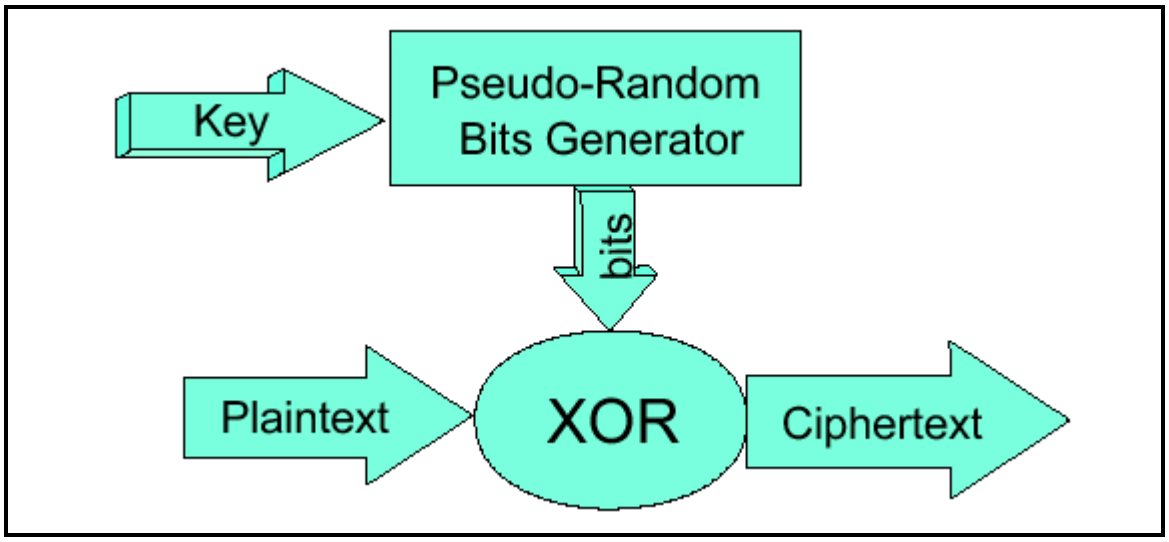

Figure 4.9: Pseudo-random generator based stream cipher

The security of a pseudo-random generator based stream cipher depends on the indistinguishability of the pseudo-random sequence. Therefore, such ciphers are only computationally secure, i.e. not secure against computationally unlimited adversary. We caution the reader that quite often, intentionally or not, this point is blurred when vendors present pseudo-random generator based stream ciphers.

In general, a *stream cipher* is a cryptosystem that operates on unbounded length plaintext, and produces the ciphertext block-by-block (often, with one bit blocks, hence bit-by-bit). Each ciphertext block is a function of the corresponding plaintext block as well as of a state variable, as illustrated in Figure 4.10

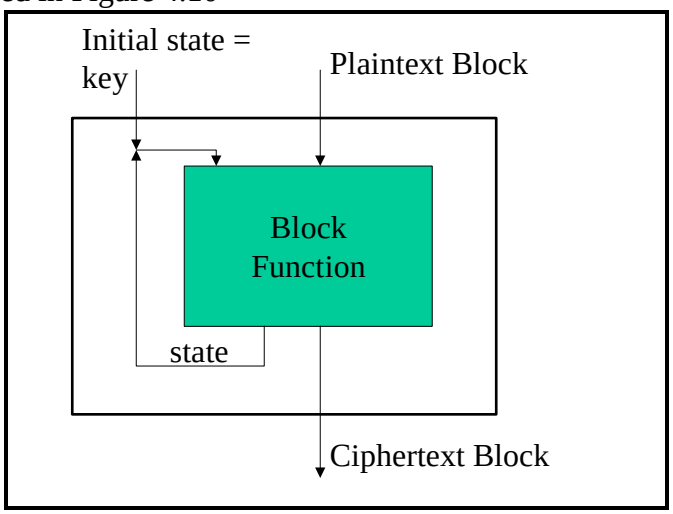

Figure 4.10: Stream Cipher

Designers often use stream ciphers for hardware implementations, where many stream cipher constructions can be more efficient, simpler and less expensive to produce than

comparable block ciphers. Unfortunately, there is no widely adopted or standardized stream cipher, comparable to the DES and AES block cipher standards. Most `modes of operation` of block ciphers, e.g. CBC and OFB, are actually stream ciphers constructions based on the block cipher. The modes of operations repeatedly apply the block cipher to plaintext blocks, combined with some state from the previous block. Of course, these constructions cannot be more efficient or simpler than the block cipher.

4.3. Extracting Randomness (to be completed)

Randomness is required for many cryptographic mechanisms, e.g. to pick random keys, or for randomizing the encryption process. Most of these applications require a uniformly distributed random string. However, computer systems are quite deterministic by design, and it is difficult to find a source of truly random bits, certainly with uniform distribution. In particular, the `random` function calls, available in most programming languages, do not provide truly random bits.

The two main techniques to collect somewhat random bits are:

- Physical randomness: to sample some source of physical randomness. This usually requires the addition of a special hardware device such as a noisy diode.
- Sampling of highly unpredictable inputs: this is a software only solution. We often use the unpredictability of the timing of human actions, such as keystrokes or mouse movements (mouse movements also have unpredictability of exact locations). Other randomized inputs that software can sample are memory contents, disk delays, log files, etc..

Unfortunately, none of these sources is completely random and unpredictable, and certainly not uniform. Therefore, we collect such randomized inputs from one or more sources, and then apply some process to produce a string that is indistinguishable from a uniformly random string – namely, a *pseudo-random string*. We can then use this (possibly short) uniformly random string as a key to a pseudo-random function, to generate multiple pseudo-random strings for different applications. Alternatively, we can use the short uniformly random string as input to a *pseudo-random number generator*, to generate a long (unbounded length) pseudo-random string.

There have been considerable amount of theoretical works on extracting randomness from weakly random sources.

To be completed: survey works, explain impossibility, demonstrate the Van Neuman and Vazirani technique for independent random variables, and complete the claim below.

If we can assume that the adversary has no control over the generation of the weakly random bits, then there is a simple and practical mechanism to extract randomness, which uses pseudo-random functions. The function uses a (fixed) uniformly distributed key, embedded in the software. This key is not necessarily secret. However, the result of applying the function with this key to input that is slightly random, results in pseudo-random output.

[[Elaborate here and also show why this may fail if the adversary can select some input to the PRF, discuss in general the importance of not letting adversary know the key to PRF and control its inputs]]

Notice that many practical systems use a different construction to extract randomness, based on hash functions. We will learn hash functions in the next chapter, and discuss the construction there.

4.4. Encryption and Compression

Messages sent often contain a large amount of redundancy, namely their probability distribution is far from uniform. Many communication systems use compression to remove some of that redundancy and reduce the size of the communicated message.

A good encryption algorithm produces output (ciphertext) that is indistinguishable from uniformly random string. Therefore, in particular, there should be no gain (no reduced length) by compressing the ciphertext. On the other hand, compressing the plaintext (before encryption) removes redundancies in it, possibly making cryptoanalysis harder. The correct order is therefore to first compress, then encrypt.

Note, however, that encryption does not hide the length of its input, which in fact is assumed to be constant. (for block ciphers) or infinite (for stream ciphers). Therefore, when we encrypt after compression, the design should prevent information leakage from the length of the (compressed) plaintext.

4.5. Exercises

1. Demonstrate that all deterministic block ciphers are vulnerable to known plaintext distribution attack.
2. Show that the simple substitution cipher is unconditionally secure, when applied to a single block of plaintext.
3. Show that the simple substitution cipher is secure against chosen ciphertext attack, for sufficiently large block size and normally distributed plaintext.
4. Alice wants to encrypt communication with Bob and Charlie, but has storage for only one key. She decides to use the same key for both of them, but a different cryptosystem for each. Criticize this design, preferably by presenting a counterexample – two cryptosystems which are secure if used alone, but easily broken if used with the same key. Suggest a simple and secure alternative solution.
5. Show how to generate a pseudo-random sequence, given a pseudo-random permutation.
6. Let $<E,D>$ be a pseudo-random permutation on $\{0,1\}^n$. Consider the function $E'_k(x)= E_k(x[1..n]) \mid E_k(E_k(x[1..n]) \oplus x[(n+1)..2n])$, namely the CBC mode applied to input of length $\{0,1\}^{2n}$. Is this E' a permutation? A pseudo-random permutation?
7. Show that CBC and OFB mode, when used with random IV, are not secure against chosen ciphertext attack, even if the underlying block cipher is secure against chosen ciphertext attack. See section 4.2.6.