Chapter 12
Block Cipher Modes of Operation from a Hardware Implementation Perspective

Debrup Chakraborty and Francisco Rodríguez-Henríquez

12.1 Introduction

Block ciphers are one of the most important primitives in cryptology. They are based on well-understood mathematical and cryptographic principles. Due to their inherent efficiency, these ciphers are used in many kinds of applications which require bulk encryption at high speed.

Generally speaking, a block cipher consists of at least two closely related algorithms: block encryption and block decryption. Block encryption takes as an input a fixed-length block (known as the *plaintext*) and transforms it into another block of the same length (known as the *ciphertext*) under the action of a fixed secret key that may or may not have the same length of the plaintext. A block cipher must be invertible in the sense that by using the block decryption algorithm it should be always possible to recover the original plaintext from the ciphertext and the secret key. Figure 12.1 shows schematically the situation just described. We stress that once the plaintext has been encrypted using a given key, then a successful decryption can only be performed by knowing that key. Due to this feature, block ciphers are classified as a secret or symmetric key primitives.

Formally a block cipher is considered to be secure if it behaves like a strong pseudorandom permutation, i.e., a block cipher is secure if an adversary cannot distinguish its output from a randomly chosen permutation. This definition of security for block ciphers is very strong, it implies that for any possible input, a secure block cipher should produce random outputs. It is unfortunate that there exists no formal security model for assessing whether a block cipher is or not secure or rather, how secure a cipher is. Hence, we rely on a block cipher by the fact that no one has been able to find an attack on it.

Assuming that an adversary has managed to obtain a plaintext and the corresponding cipher text, then she can always try to obtain the n-bit secret key of a

CINVESTAV IPN, Mexico
e-mail: debrup,francisco@cs.cinvestav.mx

Ç.K. Koç (ed.) *Cryptographic Engineering*, DOI 10.1007/978-0-387-71817-0_12,

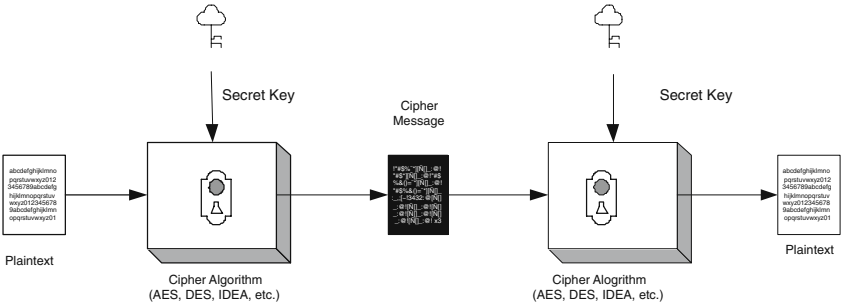

Fig. 12.1 Encrypting/decrypting with block ciphers.

given symmetric block cipher by trying all possible keys, a procedure traditionally termed *brute force attack*. We say that a block cipher has a security strength of n bits if the best known attack against it is not computationally cheaper than the brute force attack. Modern block ciphers typically use a block and key length ranging from 64 bits up to 256 bits. At the present state of the technology, block/key sizes of 128 bits are generally considered adequate in terms of both security and efficiency.

Block ciphers have been around for civilian/commercial use since 1971, when a team leaded by H. Feistel and his colleagues at IBM designed a family of ciphers known as Lucifer [2, 53]. Early versions of Lucifer operated on 24-bit long plaintext blocks. The strongest variant which was released in 1973, operated on 128-bit blocks and 128-bit secret keys. A revised version of that Lucifer variant, known as the data encryption standard (DES), was adopted as a US FIPS standard in 1974 [16, 42]. Across the years, DES became the most influential block cipher ever inspiring many new designs and attacks. Some other examples of famous block ciphers include IDEA, AES, RC6, etc.

Most block ciphers have an iterative design which implies that the block being encrypted/decrypted is processed by repeatedly applying a simpler function called *round*. Typically, the number of rounds in modern ciphers ranges from 10 up to 32. It is also customary to use a different sub-key per round, which is sometimes called *round key*. Round keys are usually derived from the user secret key mentioned before, through a process called *key schedule*. Hence, a contemporary block cipher specification usually comprises three different algorithms, namely, encryption, decryption and key schedule algorithms.

As we have seen, block ciphers can only process plaintexts/ciphertexts with a bit length smaller than blocklength of the block cipher, which is typically less than 256 bits. But this is an unacceptable restriction since applications demand encryption/decryption of arbitrary long messages. In order to overcome this difficulty, it is necessary to introduce the concept of a *mode of operation*, which we will define next.

A block cipher can be viewed as a function $E : \mathscr{K} \times \mathscr{M} \rightarrow \mathscr{C}$, where $\mathscr{K} \in \{0,1\}^k$ and $\mathscr{M}, \mathscr{C} \in \{0,1\}^n$. Then, a mode of operation can be defined as a procedure that takes as input a key $K \in \{0,1\}^k$, a message $P \in \{0,1\}^*$ of arbitrary length and

sometimes an initialization vector or *nonce* $IV \in \{0,1\}^v$ and produces a ciphertext $C \in \{0,1\}^*$ as its output. During the encryption process, some modes also produce a tag $\tau \in \{0,1\}^l$ with $0 < l \leq n$ that can be considered as a checksum or hash value of the plaintext message.[1] The notion of a tag value is useful for offering the security service of data integrity/authentication.

More informally, a mode of operation is a specific way to use a block cipher to enable it to encrypt arbitrary long messages and (optionally) to provide other security services, such as data confidentiality/privacy, authentication or a combination of both.

Let us now assume for a moment that we have a secure block cipher which produces outputs that are indistinguishable from random strings. Unfortunately, even if we manage to obtain such a strong cipher, it is not guaranteed that we can use it securely to encrypt arbitrary long messages. To illustrate this point, let us introduce next the most naive (and arguably the most insecure) mode of operation: electronic code book (ECB).

Let us consider an arbitrary plaintext message P of bitlength l. Then, we can partition the plaintext P into $b = \lceil l/n \rceil$ plaintext blocks $P_1, P_2, \ldots, P_b$ of length n, where n is the block length handled by the cipher. It is noticed that if the message length l is not a multiple of n, then the last plaintext block would be incomplete, but for the sake of simplicity, let us assume that l is a multiple of n. Then, the l-bit ciphertext C can be produced by invoking the block cipher a total of b times, thus producing b cipher blocks C_i given as $C_i = E(K, P_i)$ for $i = 1, 2, \ldots, b$. The procedure just outlined is known as the electronic code book (ECB) mode of operation.

ECB is highly insecure when dealing with plaintexts that exhibit high symmetry at the block level. For instance, Figure. 12.2a shows a 256×256 byte chess board in grayscale. If we use a block cipher with block length of 128 bits (such as AES), then the corresponding ECB-encrypted image will keep the same symmetry of the plaintext (see Figure 12.2b). This shows that designing a scheme able to encrypt arbitrary long messages using a given block cipher is not trivial.

The earliest modes of operation reported in the open literature were described back in 1981, in the standard FIPS Pub. 81 [17].[2] In that document four modes of operation were specified, namely, the electronic code book (ECB), cipher block chaining (CBC), cipher feedback (CFB) and output feedback (OFB) modes, where the data encryption standard (DES) was the underlying block cipher.

Likewise, FIPS Pub. 46-3 [16, 42] approved the seven modes specified in ANSI X9.52 [1]. Four of those modes were equivalent to the ECB, CBC, CFB and OFB modes with the triple DES algorithm (TDEA) as the underlying block cipher, whereas the other three modes in ANSI X9.52 were variants of the CBC, CFB and OFB modes. In [46], the counter mode of operation was added to the list of approved modes of operation.

[1] These modes of operation termed "*authenticated encryption modes*" are discussed in detail in Section 12.6.3.

[2] In fact, counter mode encryption ("CTR mode") was already introduced by Diffie and Hellman in 1979 [14, 32].

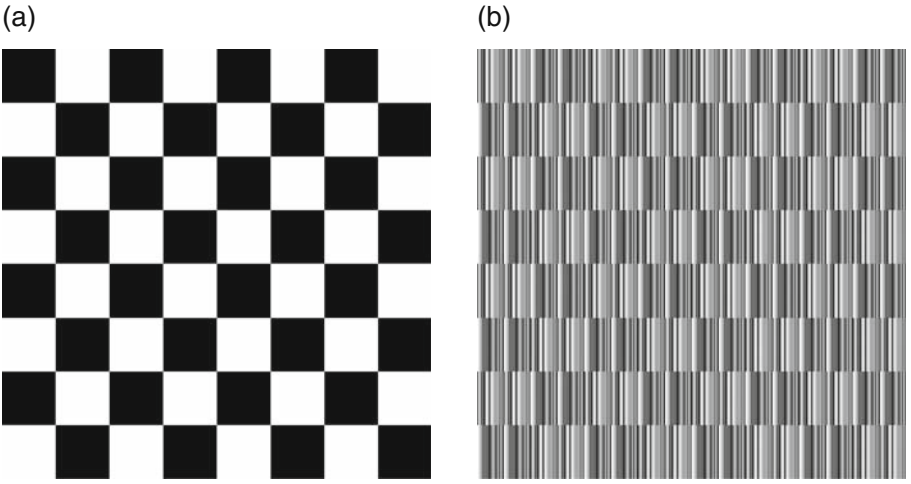

Fig. 12.2 (**a**) A 256×256 byte chess board. (**b**) The ECB-encrypted image using AES-128.

By the end of the last century, several papers pointed toward new directions on modes of operation research. The definition of the type of security provided by block ciphers was fundamental toward developing modes of operations. As stated earlier, a block cipher is considered as a pseudorandom permutation, thus the basic problem to be addressed was to find domain extensions for pseudorandom permutations. The work by Luby and Rackoff [35] was fundamental in this direction. Later, in [4] various security notions for security in the symmetric setting was presented, it also provided formal analysis of the traditional modes of operations proving security of some of them and giving the security bounds. In 2000, formal security notions of authenticity for symmetric encryption was presented in [5]. These led to numerous proposals for authenticated encryption schemes. In another development, Naor and Reingold [44] proposed a scheme to construct a strong pseudorandom permutation using a block cipher. This work along with the one reported in [33] was one of the first proposals for a class of constructions later called as tweakable enciphering schemes or disk encryption schemes.

The second generation of modes of operation, therefore, was designed to offer other security services according to different application goals. Some of the most important classes of modes of operation are those modes which guarantee confidentiality, modes for authenticated encryption, modes for authenticated encryption with associated data and modes for disk encryption.

Another characteristic that distinguishes the second generation of modes of operation is the fact that they are mostly designed for operating with several or even arbitrary selections of block ciphers. The idea that a mode of operation is a research problem largely independent of the specific block cipher being used may seem quite natural nowadays. Nevertheless, 25 years ago, when the first generation modes of operation were being specified, they were usually associated to a specific block cipher (typically DES or triple DES) [16, 17, 42].

A remarkable feature of modes of operation is the fact that, in contrast with what we have for block ciphers, a formal model for assessing their security is available.[3] Applications require various kinds/levels of securities and once a strict security model can be established for a given application, one can have a construction of a mode of operation secure under that model. Hence, a modern mode of operation is always proposed with a security bound proof valid within the model of analysis and the given security definition that provides bounds in terms of adversarial resources.

Block ciphers in different modes of operation have been implemented on all kinds of hardware and software platforms. For example, AES software implementations [7, 19] have a throughput that ranges from 300 to 800 Mbps depending on the specific architecture and platform selected by the developers. Some efficient encryptor/decryptor core VLSI implementations have also been reported in [27, 36, 51]. Performance of VLSI implementations ranges from 2 to 7.5 Gbps for the AES block cipher. Similarly, various reconfigurable hardware implementations have been reported in [8, 18, 21, 26, 34]. Those are one round (*iterative*) or n rounds (*pipeline*) FPGA implementations optimized for encryption or encryption/decryption processes. Reported performance results are broadly variable ranging from 300 Mbps to up to 25 Gbps.

Various design strategies are used for the hardware implementation of a typical block cipher and the corresponding modes of operation. An iterative looping design (IL) implements only one round and n iterations of the algorithm are carried out by feeding back previous round results. It utilizes less area (in terms of hardware resources) but consumes more clock cycles, causing a relatively low-speed encryption. In a loop unrolling or pipeline design (PP), rounds are replicated and registers are provided between the rounds to control the dataflow. The design offers high speed but area requirements are also high.

In fact, the specific selection of the mode of operation to be utilized will have a significant impact in the design of an architectural design for a block cipher. In the vast majority of block cipher hardware implementations, the electronic code book (ECB) mode of operation has been targeted. Arguably, this is because ECB is the simplest of all modes, which allows independent block encryption. Then, several blocks can be processed in parallel or pipeline strategies can be applied to increase performance.

Unfortunately, we are aware of just a handful of hardware designs of the other four traditional modes of operation, namely, CBC, CFB, OFB and the counter mode. Hardware designs implementing those modes of operation can be found in [3, 13, 18, 34, 37, 38].

The situation is even worse for the second generation of modes of operation, since many of them have never been implemented either in hardware or in software platforms. This rather deplorable situation may be caused for at least three factors. First, designing a new mode of operation has become quite a theoretical challenge in the sense that a formal proof of the alleged security of the new mode must be

[3] Usually under the assumption that the underlying block cipher is a strong pseudorandom permutation.

included in the proposal. More often than not, researchers focus all their energies on the arduous search of security bounds for their modes of operation, while the actual implementation of their proposals (including test vector generation) is neglected, ignored or, in the best case, delayed.

A second factor that may explain the lack of actual implementations for modern modes is due to efficiency reasons. Commonly, a hardware designer is only interested on producing the fastest or the most compact possible designs. Fast and ultra fast designs can only be obtained by utilizing (sub)pipeline architectures, which frequently prevent the usage of more sophisticated modern modes. On the other hand, compact designs are very often not compatible with modern modes because they typically include in their specification a number of costly building blocks (such as hash functions, field multipliers, etc.).

A third factor may be due to the fact the many of the second generation modes have been patented. This fact discourages both academicians and IT engineers to devote time to work on the hardware implementation of those modes.

The aim of this chapter is to give a brief overview of some of the most important modes of operation that have been proposed in the last few years. We first describe the traditional modes of operation, followed by a discussion of the security requirements and the adversary model used in modern modes. We describe in detail one authenticated encryption and one disk encryption mode. In order to illustrate our discussion we also provide as a case of study the design and hardware implementation description of a two pass authenticated encryption mode: AES-CCM, which also includes a brief description of the AES block cipher.

12.2 Block Ciphers

A block cipher can be viewed as a function $E : \mathscr{K} \times \mathscr{M} \rightarrow \mathscr{C}$, where $\mathscr{K} = \{0,1\}^k$ and $\mathscr{M} = \mathscr{C} = \{0,1\}^n$. Thus $\mathscr{K}$, $\mathscr{M}$ and $\mathscr{C}$ are finite nonempty sets of bit strings which are called the key space, the message space and the cipher space, respectively. The parameters n and k are called the block length and key lengths, respectively. These parameters can be different for different block ciphers. As evident from the definition, a block cipher takes as input a n-bit message (also called plaintext) and a k-bit key and produces a n-bit ciphertext. For any fixed $K \in \mathscr{K}$ we shall denote $E(K,P)$ as $E_K(P)$. For any key $K \in \mathscr{K}$ it is required that E_K is a permutation, i.e., the function $E_K : \{0,1\}^n \rightarrow \{0,1\}^n$ is a bijection. In other words for every ciphertext $C \in \{0,1\}^n$ there exist one and only one message $P \in \{0,1\}^n$ such that $E_K(P) = C$. So E_K being a permutation, it ensures that for every K a given E_K will have a inverse function which is generally called as $E_K^{-1}()$ or $D_K()$. Thus, D_K also maps $\{0,1\}^n$ to $\{0,1\}^n$ and $D_K(E_K(P)) = P$ and $E_K(D_K(C)) = C$ for all P and C in $\{0,1\}^n$.

The function $E_K()$ and $D_K()$ must be such that they can be easily computed and these functions should be normally publicly available. To use a block cipher, a key is randomly selected from the key space and agreed upon by the sender and receiver. This key should be kept secret.

Modern day block ciphers are usually composed of several identical transforms, denoted as rounds. In each round the plaintext or the semi-transformed plaintext gets transformed with the action of a round key, which is derived from the secret key by some specific transform steps. A concrete instantiation of a block cipher which is widely used is called the Advanced Encryption Standard (AES). To continue the modes of operation discussion we shall not depend on any specific block cipher, as a mode of operation is generally designed irrespective of the block cipher and any secure block cipher can be plugged into it. For the sake of completeness, we give a detailed description of the AES algorithm in the following section. Furthermore, a brief background information on binary extension fields is also given after that; this information will be useful when we discuss the offset codebook mode and its implementation aspects of the OCB and the AES rounds.

12.3 Introduction to AES

Rijndael block cipher algorithm was chosen in October 2000 by NIST as the new Advanced Encryption Standard (AES) [28]. In the rest of this section we shall give a brief summary of the AES encryption process.

The basic structure of AES consists of a message input (128 bits), a secret user key (128 bits) and a cipher message (128 bits) as the output. The AES cipher treats the input 128-bit block as a group of 16 bytes organized in a 4×4 matrix called the *state* matrix.

As is shown in Figure 12.3, the AES encryption algorithm consists of an initial transformation, followed by a main loop where nine iterations called *rounds* are executed. Each *round transformation* is composed of a sequence of four transformations, namely byte substitution (BS), ShiftRows (SR), MixColumns (MC) and AddRoundKey (ARK). For each round of the main loop, a round key is derived from the original key through a process called *Key Scheduling*. Finally, a last round consisting of three transformations BS, SR and ARK are executed.

The AES decryption algorithm operates similarly by applying the inverse of all the transformations described above in reverse order. In the rest of this section we briefly describe the AES round transformations, whereas the round-key derivation process will be explained in Section 12.3.5.

Rounds 1–9 consist of the application of the four basic steps to the state matrix. The order of the AES steps for these rounds are BS, SR, MC and ARK.

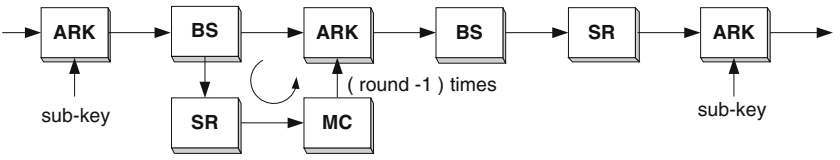

Fig. 12.3 AES encryption process.

12.3.1 Byte Substitution (BS) Step

This is the first step for rounds from 1 to 10 and it is the main non-linear transformation of the encryption process. In BS step, each input byte of the state matrix is independently replaced by another byte from a look-up table called S-box. The S-box of the AES algorithm consists of 256 entries each of one byte, where each byte is represented in the binary extension field $GF(2^8)$ constructed using the irreducible pentanomial $P(x) = x^8 + x^4 + x^3 + x + 1$. An AES S-box is composed of two transformations: First, each input byte is replaced with its multiplicative inverse in $GF(2^8)$ with the element 00 being mapped onto itself; then, an affine transformation over GF(2) is applied. The affine transformation consists of a matrix multiplication by a constant matrix followed by the addition of the hexadecimal value "63". For decryption, the inverse S-box is applied by obtaining the inverse affine transformation followed by multiplicative inversion in $GF(2^8)$.

12.3.2 Shift Rows (SR) Step

It is the second step in the round transformation, consisting of a cyclic shift operation where each row in the state matrix is rotated cyclically to the left using $0-, 1-, 2-$ and $3-$ byte offset. In decryption, the rotation is applied to the right.

12.3.3 Mix Columns (MC) Step

In MC step, each column of the state matrix, considered as a polynomial over $GF(2^8)$, is multiplied by a fixed polynomial $c(x)$ modulo $x^4 + 1$. The polynomial $c(x)$ is given by $c(x) = 03 \cdot x^3 + 01 \cdot x^2 + 01 \cdot x + 02$.

Let $b(x) = c(x)a(x) \bmod (x^4 + 1)$, then the modular multiplication with a fixed polynomial can be written as:

$$\begin{bmatrix} b_{0,0} \\ b_{0,1} \\ b_{0,2} \\ b_{0,3} \end{bmatrix} = \begin{bmatrix} 02\ 03\ 01\ 01 \\ 01\ 02\ 03\ 01 \\ 01\ 01\ 02\ 03 \\ 03\ 01\ 01\ 02 \end{bmatrix} \begin{bmatrix} a_{0,0} \\ a_{0,1} \\ a_{0,2} \\ a_{0,3} \end{bmatrix} \tag{12.1}$$

For the decryption process, we compute Inverse MixColumns using the constant polynomial $w(x) = w_3x^3 + w_2x^2 + w_1x + w_0$, with coefficients $w_0(x) = x^3 + x^2 + x$, $w_1(x) = x^3 + 1$, $w^2(x) = x^3 + x^2 + 1$, $w_3(x) = x^3 + x + 1$, and reduced modulo $M(x) = x^4 + 1$, which is multiplied by each column of a block. The equivalent matrix representation is

$$\begin{bmatrix} b_{0,0} \\ b_{0,1} \\ b_{0,2} \\ b_{0,3} \end{bmatrix} = \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} a_{0,0} \\ a_{0,1} \\ a_{0,2} \\ a_{0,3} \end{bmatrix} \tag{12.2}$$

12.3.4 Add Round Key (ARK) Step

The output of MC is XORed with the round key derived from the user key. The ARK step is symmetric for encryption and decryption. The only difference is that the sub-key rounds are applied in reverse order for decryption.[4] Each one of the above described transformations BS, SR, MC and ARK are invertible [28]. Let us call them IBS, ISR, IMC and IARK, respectively. Then the AES encryption/decryption procedures can be described as follows:

1. ARK, using the 0th round key.
2. Nine rounds of BS, SR, MC, ARK, using round keys 1–9.
3. A final round: BS, SR, ARK, using the 10th round key.

Similarly, the decryption is computed as follows:

1. ARK, using the 10th round key.
2. Nine rounds of IBS, ISR, IMC, IARK, using round keys 9–1.
3. A final round: IBS, ISR, ARK, using the 0th round key.

12.3.5 Key Scheduling Algorithm

The round keys are obtained through the expansion of secret user key by attaching recursively the 4-byte word $k_i = (k_{0,i}, k_{1,i}, k_{2,i}, k_{3,i})$ to the user key. The original user key consists of 128 bits arranged as a 4×4 matrix of bytes. Let $w[0], w[1], w[2]$ and $w[3]$ be the four columns of the original user key. Then, those four columns are recursively expanded to obtain 40 more columns as follows:

$$w[i] = \begin{cases} w[i-4] \oplus w[i-1] & \text{if } i \bmod 4 \neq 0 \\ w[i-4] \oplus T(w[i-1]) & \text{otherwise} \end{cases} \tag{12.3}$$

where $T(w[i-1])$ is a non-linear transformation of $w[i-1]$ computed as follows. Let w, x, y and z be the elements of the column $w[i-1]$, then

1. Shift cyclically the elements to obtain x, y, z and w.
2. Replace each byte by a byte using the S-Box as $S(x), S(y), S(z), S(w)$.
3. Compute the round constant *rcon*, defined as $r(i) = 02^{(i-4)/4}$ over $GF(2^8)$.

[4] However, efficient implementations of AES encryptor/decryptor cores require to append the IMC step to the generation of round keys for decryption.

Then $T(w[i-1])$ is the column vector, $(S(x) \oplus r(i), S(y), S(z), S(w))$. In this way, columns from $w[4]$ to $w[43]$ are generated from the first four columns. Hence, the ith round key consists of the columns

$$(w(4i), w(4i+i), w(4i+2), w(4i+3)) \tag{12.4}$$

In [47] several optimizations based on redundant computation for parallelizing the Key Scheduling process were implemented. As a result, above four steps can be reduced to only two steps [47].

Step 1	Step 2	(12.5)
$k'_0 = k_0 \oplus SBox(k_{13}) \oplus rcon;$	$k'_4 = k_4 \oplus k'_0;$	
	$k'_8 = k_8 \oplus k_4 \oplus k'_0;$	
	$k'_{12} = k_{12} \oplus k_8 \oplus k_4 \oplus k'_0;$	
$k'_1 = k_1 \oplus SBox(k_{14});$	$k'_5 = k_5 \oplus k'_1;$	
	$k'_9 = k_9 \oplus k_5 \oplus k'_1;$	
	$k'_{13} = k_{13} \oplus k_9 \oplus k_5 \oplus k'_1;$	
$k'_2 = k_2 \oplus SBox(k_{15});$	$k'_6 = k_6 \oplus k'_2;$	
	$k'_{10} = k_{10} \oplus k_6 \oplus k'_2;$	
	$k'_{14} = k_{14} \oplus k_{10} \oplus k_6 \oplus k'_2;$	
$k'_3 = k_3 \oplus SBox(k_{12});$	$k'_7 = k_7 \oplus k'_3;$	
	$k'_{11} = k_{11} \oplus k_7 \oplus k'_3;$	
	$k'_{15} = k_{15} \oplus k_{11} \oplus k_7 \oplus k'_3;$	

12.4 A Background in Binary Extension Finite Fields

12.4.1 Rings

A ring R is a set whose objects can be added and multiplied, satisfying the following conditions:

- Under addition, R is an additive (Abelian) group.
- For all $x, y, z \in R$ we have,

$$\begin{aligned} x(y+z) &= xy+xz \\ (y+z)x &= yx+zx \end{aligned}$$

- For all $x, y \in R$, we have $(xy)z = x(yz)$.
- There exists an element $e \in R$ such that $ex = xe = x$ for all $x \in R$.

The integer numbers, the rational numbers, the real numbers and the complex numbers are all rings.

An element x of a ring is said to be invertible if x has a multiplicative inverse in R, that is, if there is a unique $u \in R$ such that $xu = ux = 1$. 1 is called the *unit element* of the ring.

12.4.2 Fields

A field is a ring in which the multiplication is commutative and every element except 0 has a multiplicative inverse. We can define the field F with respect to the addition and the multiplication if

- F is a commutative group with respect to the addition.
- $F \setminus \{0\}$ is a commutative group with respect to the multiplication.
- The distributive laws mentioned for rings hold.

12.4.3 Finite Fields

A finite field or *Galois field* denoted by $GF(q = p^m)$ is a field with characteristic p and a number q of elements. Such a finite field exists for every prime p and positive integer m and contains a subfield having p elements. This subfield is called *ground field* of the original field. For every non-zero element $\alpha \in GF(q)$, the identity $\alpha^{q-1} = 1$ holds. Furthermore, an element $\alpha \in GF(q^m)$ lies in $GF(q)$ itself if and only if $\alpha^q = \alpha$.

In the following we will only consider binary extension fields, where $q = 2^m$, also known as finite fields of characteristic two or simply binary fields.

12.4.4 Binary Finite Field Arithmetic

In the following we will use the *polynomial basis* representation of the binary finite fields elements. We represent each element as a binary string $(a_{m-1} \ldots a_2 a_1 a_0)$, which is equivalently considered a polynomial of degree less than n:

$$a_{m-1}x^{m-1} + \cdots + a_2x^2 + a_1x + a_0. \tag{12.6}$$

Addition is by far the less-costly field operation, whereas multiplication is arguably the most important arithmetic operation.

The addition of two elements $a, b \in F$ is simply the addition of two polynomials, where the coefficients are added in $GF(2)$, or equivalently, using the bit-wise XOR operation on the vectors a and b. The multiplication of two field elements can be accomplished as follows.

Let $A(x), B(x)$ be elements of $GF(2^m)$ and let $P(x)$ be the degree m irreducible polynomial generating $GF(2^m)$. Then, the field product $C'(x) \in GF(2^m)$ can be obtained by first computing the polynomial product $C(x)$ as

$$C(x) = A(x)B(x) = \left(\sum_{i=0}^{m-1} a_i x^i\right)\left(\sum_{i=0}^{m-1} b_i x^i\right) \tag{12.7}$$

Followed by a reduction operation, performed in order to obtain the $(m-1)$-degree polynomial $C'(x)$, which is defined as

$$C'(x) = C(x) \bmod P(x) \tag{12.8}$$

Once the irreducible polynomial $P(x)$ is selected and fixed, the reduction step can be accomplished using only XOR gates.

A particular case of field multiplication is that of multiplying an arbitrary field element $A(x)$ by the field element x, an operation sometimes called *xtimes*, that is frequently used in block cipher modes of operations and forms part of the AES specification. The operation *xtimes* can be accomplished very efficiently by noticing that

$$xtimes(A) = x \cdot A(x) = x \cdot \sum_{i=0}^{m-1} a_i x^i = \sum_{i=0}^{m-1} a_i x^{i+1} \tag{12.9}$$

Therefore, if the most significant bit of A, namely a_{m-1}, is equal to zero, then $xtimes(A)$ can be accomplished by a single left shift of the original element A. On the other hand, if $a_{m-1} = 1$, we can simply add $P(x)$ to $C(x)$, thus reducing the power $a_{m-1}x^m$. As a concrete example consider the case when $P(x)$ is an irreducible pentanomial. Then, the *xtimes* operation can be computed as

$$xtimes(A) = \begin{cases} \sum_{i=0}^{m-2} a_i x^{i+1} & \text{if } a_{m-1} = 0 \\ \sum_{i=0}^{m-2} a_i x^{i+1} + (x^{k_2} + x^{k_1} + x^{k_0} + 1) & \text{if } a_{m-1} = 1 \end{cases} \tag{12.10}$$

The computational cost of the above equation is one left shift followed by possibly an XOR operation.

12.5 Traditional Modes of Operations

As it was already mentioned, a block cipher can only encrypt fixed length strings, but in real life, messages are of arbitrary lengths and are not restricted to the block length of a block cipher. A mode of operation is a specific way to use a block cipher for encrypting arbitrarily long messages. We now discuss some of the traditional modes of operations which have been in use for long. The five modes which we describe next are called the ECB (electronic code book), CBC (cipher block chaining), CFB (cipher feedback), OFB (output feedback) and CTR (counter). For ease of description we shall assume that the length of the plaintexts are multiples of the blocklength of the block cipher. We shall denote by n the blocklength and by m the number of blocks.

12.5.1 Electronic Code Book Mode

This is probably the simplest of all modes. In the electronic code book (ECB) mode the plaintext P is segmented as $P = P_1||P_2||\ldots||P_m$, where each P_i is an n-bit long block. Thereafter, the encryption function E_K is applied separately on each P_i. A schematic diagram of a ECB mode is shown in Figure 12.4.

12.5.2 Cipher Block Chaining Mode

In cipher block chaining (CBC) mode, the output of one block cipher is fed into the other block cipher along with the next block message. The algorithm below describes the mode and a pictorial description is provided in Figure 12.4b.

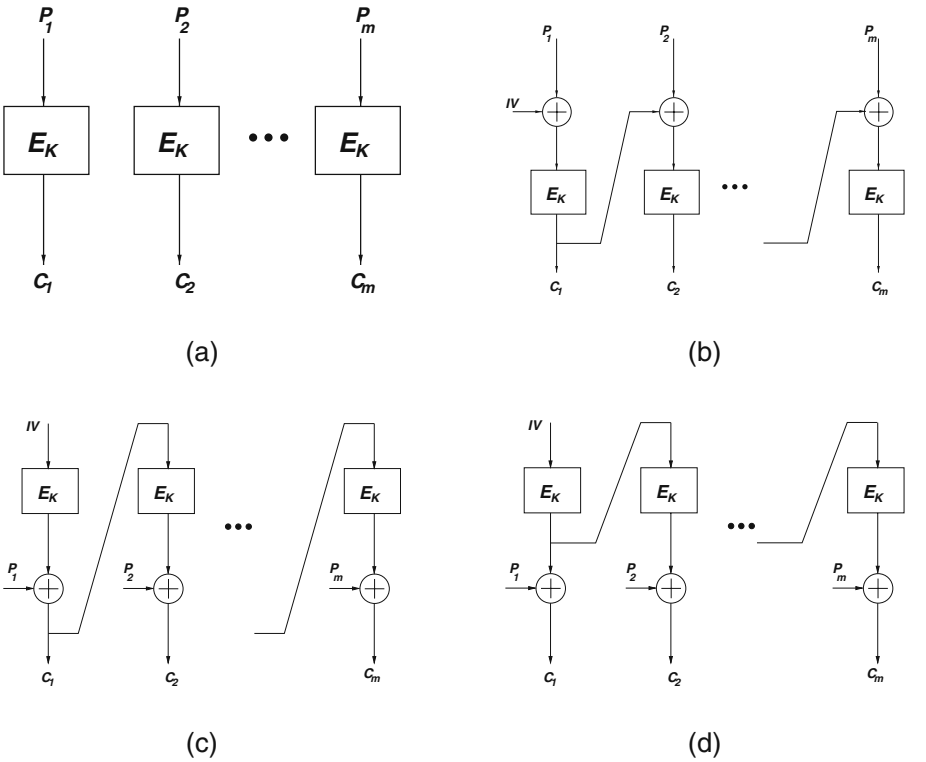

Fig. 12.4 The traditional modes of operation: (**a**) ECB, (**b**) CBC, (**c**) CFB, (d) OFB.

Algorithm CBC.Encrypt$_K^{\text{IV}}(P)$	**Algorithm** CBC.Decrypt$_K^{\text{IV}}(C)$
1. Partition P into $P_1, P_2, \ldots, P_m$	1. Partition C into $C_1, C_2, \ldots, C_m$
2. $C_1 \leftarrow E_K(P_1 \oplus \text{IV})$;	2. $P_1 \leftarrow E_K^{-1}(C_1) \oplus \text{IV}$
3. **for** $i \leftarrow 2$ to m	3. **for** $i \leftarrow 2$ to m
4. $\quad C_i \leftarrow E_K(P_i \oplus C_{i-1})$	4. $\quad P_i \leftarrow E_K^{-1}(C_i) \oplus C_{i-1}$
5. **end for**	5. **end for**
6. **return** $C_1, C_2, \ldots, C_m$	6. **return** $P_1, P_2, \ldots, P_m$

CBC takes as input m message blocks and an initialization vector (IV). During encryption, the output of the ith block depends on the previous $i-1$ blocks. So, CBC encryption is inherently sequential. The output of each block depends on all the previous blocks and thus provides more security than ECB. The sequential design does not allow a fully pipelined implementation for this mode. Note that CBC decryption is not sequential.

12.5.3 Cipher Feedback Mode

The encryption and decryption procedures for the cipher feedback mode (CFB) are described below. A pictorial description of the mode is provided in Figure 12.4c.

Algorithm CFB.Encrypt$_K^{\text{IV}}(P)$	**Algorithm** CFB.Decrypt$_K^{\text{IV}}(C)$
1. Partition P into $P_1, P_2, \ldots, P_m$	1. Partition C into $C_1, C_2, \ldots, C_m$
2. $C_1 \leftarrow E_K(\text{IV}) \oplus P_1$;	2. $P_1 \leftarrow E_K(IV) \oplus C_1$
3. **for** $i \leftarrow 2$ to m	3. **for** $i \leftarrow 2$ to m
4. $\quad C_i \leftarrow E_K(C_{i-1}) \oplus P_i$	4. $\quad P_i \leftarrow E_K(C_{i-1}) \oplus C_i$
5. **end for**	5. **end for**
6. **return** $C_1, C_2, \ldots, C_m$	6. **return** $P_1, P_2, \ldots, P_m$

In CFB mode also the cipher blocks are chained but the output is produced in a manner much different from that of CBC. For each block, the cipher produced is just XORed with the message. Due to such a kind of encryption, the encryption and decryption operations are similar. In case of decryption also the inverse block cipher calls are not required. Like CBC, OFB encryption and decryption are also inherently sequential. But an advantage in terms of implementation is that the block cipher decryption operation is not needed.

12.5.4 Output Feedback Mode

In output feedback mode (Figure 12.4d) unlike CFB, the output of the block cipher is fed back into the next block cipher. The algorithm is as shown below.

Algorithm CFB.Encrypt$_K^{\text{IV}}(P)$	**Algorithm** CFB.Decrypt$_K^{\text{IV}}(C)$
1. Partition P into $P_1, P_2, \ldots, P_m$	1. Partition C into $C_1, C_2, \ldots, C_m$
2. $X \leftarrow \text{IV}$;	2. $X \leftarrow \text{IV}$
3. **for** $i \leftarrow 1$ to m	3. **for** $i \leftarrow 1$ to m
4. $\quad X \leftarrow E_K(X)$;	4. $\quad X \leftarrow E_K(X)$;
5. $\quad C_i \leftarrow X \oplus P_i$	5. $\quad P_i \leftarrow X \oplus C_i$
6. **end for**	6. **end for**
7. **return** $C_1, C_2, \ldots, C_m$	7. **return** $P_1, P_2, \ldots, P_m$

In this mode the *IV* is repeatedly encrypted to get a stream of random bytes. Unlike the other modes described before in OFB no part of the plaintext is ever given as an input to the blockcipher. This makes this mode very similar to a stream cipher, where a stream cipher produces a stream of random bytes and these random strings are XORed with the plaintext to generate the cipher. The specific way in which the *IV* is encrypted in the mode also makes the algorithm sequential. Hence, as in the case of CFB, for both encryption and decryption operations, only a forward call of the block cipher (i.e., its encryption algorithm) is required.

12.5.5 Counter Mode

The counter (CTR) mode is a bit different from the other modes defined above. It takes in an *IV*, and in each iteration the value of the *IV* incremented by one gets encrypted. The ciphertext is produced by XORing the encryption results with the plaintext blocks.

Algorithm CTR.Encrypt$_K^{\text{IV}}(P)$	**Algorithm** CFB.Decrypt$_K^{\text{IV}}(C)$
1. Partition P into $P_1, P_2, \ldots, P_m$	1. Partition C into $C_1, C_2, \ldots, C_m$
2. $C_1 \leftarrow E_K(\text{IV}) \oplus P_1$;	2. $P_1 \leftarrow E_K(IV) \oplus C_1$
3. **for** $i \leftarrow 2$ to m	3. **for** $i \leftarrow 2$ to m
4. $\quad C_i \leftarrow E_K((IV+i) \bmod 2^n) \oplus P_i$	4. $\quad P_i \leftarrow E_K((IV+i) \bmod 2^n) \oplus C_i$
5. **end for**	5. **end for**
6. **return** $C_1, C_2, \ldots, C_m$	6. **return** $P_1, P_2, \ldots, P_m$

In terms of efficiency the CTR mode is better than CBC, OFB or CFB as in CTR the block cipher calls can be done in parallel. No feedback takes place in case of CTR so the input to the *i*th block cipher in no way depends on the output of the previous block ciphers. Also in CTR only the encryption algorithm is needed. Additionally, for performing the decryption operation, the inverse call of the block cipher is not needed.

12.6 Security Requirements for Modes of Operations

In Section 12.5 we discussed some modes which can be used to encrypt messages longer than the block length of the block cipher. Now we shall analyze some of these modes to see whether they are secure. Also we shall try to intuitively formulate security requirements for three important classes of modes.

Let us begin by analyzing the ECB mode. The ECB mode is not suitable for encrypting bulk messages as it can reveal much information about a message. We already illustrated this with the help of Figure 12.2a and its ECB encryption in 12.2b. From Figure 12.2b we see that the encryption reveals much information regarding the image. This is because in ECB, every block is encrypted using the same key and so all equal plaintext blocks gets encrypted into equal ciphertext blocks. So, if we encrypt a four-block message say P_1, P_2, P_3, P_4 where $P_1 = P_2$, then in the ciphertext blocks C_1, C_2, C_3, C_4 also C_1 would be equal to C_2. This is not desirable as the structure of the plaintext blocks gets revealed in the ciphertext blocks. In this particular example, an adversary can readily find out that the first two plaintext blocks were equal just by looking at the ciphertext. This limitation of ECB makes the encryption in Figure 12.2b look so similar with the figure itself. Thus, ECB is insecure.

The important question we would like to address now is when a mode can be considered secure. In order to answer that, we first establish a formal model of the adversary who tries to break the security of a mode.

12.6.1 The Adversary

To define security we need to formalize the goals and resources of an adversary. An adversary can have various goals, the strongest among them, being able to recover the keys that the encryption scheme uses. With knowledge of the key she can always decrypt all encrypted messages that goes through the public channel and can also replace encrypted messages with the encryption of messages of her choice. But the goal of key recovery is a very strong goal and without recovering the key also an adversary can predict some properties of the plaintext. Thus, an intuitive goal of a crypto-system should be that it leaks no information regarding the plaintext through the ciphertext.[5] On the other hand, the weakest goal that an adversary can have is being able to distinguish the ciphertext from random strings. Thus, if a crypto-system is strong enough that an adversary cannot accomplish this weak goal, then to an adversary the ciphertext would not in any way leak any information regarding the plaintext.

An adversary who wants to break the security of a symmetric crypto-system must be given access to some of the inputs or outputs of the system. The type of

[5] Note that we already showed that the electronic code book mode (ECB) leaks some important information regarding the plantext, which is undesirable.

information access the adversary has defines the power of the adversary. The adversary always has access to the ciphertexts as he can eavesdrop the public channel and know the ciphertext. If an adversary only has access to the ciphertext we call the attack mounted by the adversary as a *ciphertext-only attack*. Additionally the adversary may know which messages produce these ciphertexts, such an attack is a *known plaintext attack*. If the adversary can choose the plaintexts whose encryptions he wants, then the attack is a *chosen plaintext attack*. There can be adversaries who can choose the ciphertexts and get the corresponding plaintexts for those ciphertexts. Such a kind of attack is called a *chosen ciphertext attack*. The strongest adversary is the one who can adaptively choose messages (ciphertext) and get their encryptions (decryptions). Such adversaries are called *adaptive chosen plaintext* (respectively ciphertext) adversaries.

To formalize things we shall view the adversary as a polynomial time probabilistic algorithm with certain resources. An adaptive chosen plaintext adversary should be supplied with ciphertexts corresponding to the plaintexts of her choice. To do this we allow the adversary to communicate with the encryption algorithm, i.e., she is given access to the encryption scheme as a black box where she gives some inputs and obtains the corresponding outputs but has no access to the internal workings of the scheme. We call such an access as an oracle access. An adversary may be given access to one or more oracles, as in case of an adaptive chosen plaintext and chosen ciphertext adversary the adversary should be given access to both the encryption and decryption oracles so that he can adaptively obtain encryptions and decryptions of his choice.

12.6.2 Privacy Only Modes

With the above characterization of the adversary we now try to define the security requirements for a class of modes called *privacy only modes*. In a privacy only mode the goal is to create ciphertexts such that the adversary by knowing the ciphertexts can have no knowledge of the plaintext. So, such an adversary has access to the ciphertexts and also we assume that the adversary can have access to the ciphertexts corresponding to plaintexts of her choice. To model security of a privacy only mode we will give the adversary access to two oracles. The first one is the encryption algorithm (i.e., the mode) and the second one is an algorithm which when given an input of a plaintext of length m returns m random bits. Thus, the adversary has two oracles, one of which is the real mode and the other returns random strings. The adversary can query these oracles without repeating any query and his task is to distinguish between these two oracles. If the probability with which any efficient adversary can distinguish between these two oracles is small then the mode can be considered secure in terms of privacy.

With this definition of privacy let us try to analyze the security of the CBC mode of operation shown in Figure 12.4b. Let us recall that in the CBC mode, the algorithm takes as input a key, an IV and the plaintext. The key is secret but in general

we may assume the IV is a public quantity. If we allow an adversary A to freely choose the IVs along with the plaintexts then CBC is also not secure in terms of privacy. An easy distinguishing attack can be mounted by A in the following manner. A provides two encryption queries and gets the corresponding ciphertexts as below:

$$\text{Query 1: } \mathrm{IV}^1; P_1^1, P_2^1 \rightarrow C_1^1, C_2^1$$
$$\text{Query 2: } \mathrm{IV}^2; P_1^2, P_2^2 \rightarrow C_1^2, C_2^2$$

Here P_i^j and C_i^j represent a block (say n bits if the block length of the block cipher is n) of plaintext and ciphertext, respectively. Additionally we assume the following restrictions on the queries:

$$P_1^1 = P_1^2 = \mathrm{IV}^2 = \mathrm{IV}^1$$
$$P_2^2 = C_1^1$$

For such a set of queries C_1^2 will always be equal to C_2^2. This happens because for the first query C_1^1 will be the block cipher output for all zero string and for the second query both C_1^2 and C_2^2 will be the encrypted output of all zero strings. Thus we see that if A is freely allowed to choose IVs then he can easily distinguish a CBC output from random strings, so CBC in this setting is not secure in terms of privacy.

To make CBC secure, we need to put a restriction on the usage of IV. First, a key and IV pair is never to be repeated. Moreover, if in CBC encryption one replaces the IV by the encryption of the *IV* (i.e., $E_K(\mathrm{IV})$) then CBC is secure in terms of privacy. Note that the IV is still a public quantity and we may allow an adversary to obtain encryptions of messages with IVs of his choice, but he is not allowed to obtain encryption of two different messages using the same IV. The IV used in this manner is called a *nonce*.

The formalization of the security requirement of privacy only modes along with the analysis of the security of the traditional modes of operation were first provided in [4] and the security of CBC with the IV as a nonce was proved in [49].

12.6.3 Authenticated Encryption

The security provided by privacy only modes may not be enough in certain scenarios. Recall, for defining privacy we assumed the adversary to be an adaptive chosen plaintext adversary whose task was to distinguish the output of the mode from random strings. Thus, if an adversary sees only ciphertexts from a secure privacy only mode, he cannot determine anything meaningful from the ciphertexts. But, if we assume that the adversary wants to tamper the ciphertexts which goes through the public channel he can always do so. In a privacy only mode the receiver has no way to determine whether she received the ciphertext that was originally sent by the sender. This forms a major limitation of privacy only modes.

To overcome this limitation we need to add some other functionality to a mode so that the receiver of a message can verify whether she had obtained the ciphertext sent by the sender. This is obtained by a tag. A tag can be considered as a checksum of the message that was used to generate the ciphertext. A sender after decrypting the ciphertext can always compute the tag and match the tag which she computed using the decrypted message with the tag that she received. If the tags do not match the receiver can know that a tampering of the ciphertext has taken place during the transit. This functionality in the symmetric setting is called authentication and the modes which provide both privacy and authentication are called authenticated encryption modes.

Thus an authenticated encryption mode can be seen as a pair of algorithms $(\boldsymbol{E},\boldsymbol{D})$ where $\boldsymbol{E}$ produces the ciphertext $\mathsf{C} = (C, \mathsf{tag})$ when given a plaintext P as an input. While the decryption algorithm $\boldsymbol{D}$ on an input C produces the corresponding plaintext P or outputs INVALID if the computed tag does not match tag.

The security requirement of AE schemes are a bit different from privacy only modes. For an AE scheme we do not want an adversary to be able to distinguish ciphertexts produced from plaintexts chosen by her adaptively. So the security requirement for privacy only modes is also a requirement for AE schemes, but additionally we want that the adversary should not be able to construct any ciphertext which will get decrypted. To model this requirement we assume that the adversary is given a number of ciphertext, tag pairs for plaintexts of his/her choice and after observing these ciphertexts the task of the adversary is to forge, i.e., to construct a ciphertext tag pair which on input to the decryption algorithm does not produce INVALID. An AE mode is considered secure in the sense of authenticity if the probability of forging of any efficient adversary is low. A secure AE scheme is needed to be secure both in the sense of privacy and authenticity.

Another class of AE schemes are called authenticated encryption with associated data (AEAD). These schemes can be useful in certain realistic scenarios. Like if we consider network packets, we do not want to encrypt the headers but we want to authenticate the headers so that they cannot be tampered. Such schemes take as input the message and an associated data (the packet header in this case), the message is only encrypted but the tag is produced both with the message and the header. Most AE schemes can be easily converted into AEADs.

12.6.4 Disk Encryption Schemes

Now, let us look at another application. Suppose we want to encrypt all data present in the hard disk of a computer. Whenever there is a disk read then the particular disk sector is decrypted and returned, similarly whenever some given data need to be written to the disk the specific sector is encrypted and then written. The encryption and decryption operations get done by the disk controller, which is a low-level device having no knowledge of the files, directories, etc. maintained by the operating system. Each sector is considered as a message.

In this setting a very important limitation of the encryption algorithm to be used is that the encryption should be length preserving, i.e., the length of the ciphertext and plaintext should be equal. This limitation dictates that AE schemes cannot be used for such applications, as in AE schemes always there is a ciphertext expansion. A privacy only kind of mode is generally length preserving, but the security it provides will not be enough for disk encryption schemes, as we do not want an adversary to tamper the data present in the disk without our knowledge. So, we need schemes in which the ciphertext produced will be indistinguishable from random strings to any adversary who can access ciphertexts corresponding to the plaintexts of his/her choice. Additionally if an adversary chooses ciphertexts and gets plaintexts corresponding to those ciphertexts, she should be unable to distinguish those plaintexts from random strings.

It should be noticed that this adversary is different from the adversary we discussed in case of privacy only and AE modes. In privacy only and the AE schemes the adversary had the freedom to choose plaintexts and get the corresponding ciphertexts and her task was to distinguish the ciphertexts from random strings. Here we are giving the adversary freedom to choose ciphertexts also. This adversary is an adaptive chosen plaintext and chosen ciphertext adversary. So, in this scenario, if the adversary plans to change the original ciphertext in the disk with some ciphertexts of her choice, then the decrypted plaintexts will be indistinguishable from random. Thus, the adversary cannot create any ciphertext which will get decrypted into something meaningful, in other words whatever ciphertext she creates will look like random when it gets decrypted.

To define security of disk encryption (DE) schemes we assumed a more powerful adversary than in case of AE schemes, but the security provided by DE schemes is less than that of AE schemes. Recall, in AE schemes the adversary has two tasks, one of distinguishing and another of forging. If an AE scheme is secure against forgery attacks then the probability with which an adversary can create a valid ciphertext (i.e., the probability with which she can tamper a ciphertext which still gets decrypted) is very low. But in case of DE schemes the adversary can tamper the ciphertext and there is no mechanism in the scheme which can detect the tampering. But the security definition guarantees that if such a tampering takes place then the corresponding plaintext will be random. So, a high-level application which uses the plaintext can detect the tampering. As most applications assume certain structure on the data it uses and a tampering of the data will violate the structure, the probability that an efficient adversary creates a ciphertext which will retain the structure in the plaintext is low.

As we discussed in DE schemes the data in a sector are considered as a plaintext/ciphertext. Thus if two sectors contain the same data they would get encrypted into the same ciphertexts and the adversary can readily get the information that the two sectors contain same data. We would not like the adversary to know such information. To assure that this does not happen the DE schemes take in a quantity called tweak along with the plaintext. The tweak may be considered as a type of associated data. The tweak is not secret and there is no restriction about repetition of the tweak as in nonces. The encryption of a message depends on the tweak used to encrypt it.

In DE schemes the sector address is considered as the tweak. So, if two different sectors contain the same message also they would have different sector addresses and thus different tweaks, so their encryptions would be different.

12.6.5 Security Proofs

We have intuitively discussed some of the security requirements of modes of operations. Note that we have not provided with formal definitions of security which can be done. The basic building block of a mode of operation is a block cipher. So the security of a mode of operation heavily depends on the security of the underlying block cipher. It is a pity that the security of primitives like block ciphers cannot be formally proved. Instead, for block ciphers we just assume security based on the facts that they can resist all known attacks.

A mode of operation is built using a block cipher and the security of a mode is thus reduced in a suitable way to the security of the block cipher. Thus, assuming the block cipher to be secure in a certain way, the security of the mode is derived using the (presumed) security of the block cipher. Such a reduction is called a security proof. We shall not discuss security proofs in this chapter, but any modern mode has a security proof associated with it and it proves an upper bound on advantage of any efficient adversary in breaking the security of that mode [10, 45].

12.7 Some Modern Modes

We already discussed security requirements of three important kinds of modes. Of these three modes, the privacy only modes are of limited interest as they do not provide security against active adversaries who can tamper the ciphertexts. The AE modes and DE modes are of much interest in the current days. There are many modes proposed till date. In Table 12.1 we list some secure AE and DE modes.

AE modes can be classified according to the number of passes over the data it requires. Easiest way to obtain an AE mode is to use two algorithms, one for computing the tag and the other for encrypting. If these two algorithms are used separately they obviously need two passes over the message and additionally two keys will be required. This paradigm is called generic composition and was first formally analyzed in [5].

The most efficient AE modes are the one pass AE modes. As the name suggests they use only one pass over the data. Some single pass AE schemes proposed till date are IACBC [29], IAPM [30], OCB [50], XCBC and XECB [20]. Also a generalization of the OCB construction was provided in [10]. Out of these modes OCB is probably the most efficient and optimized AE mode. We provide a description of OCB in Section 12.7.1

Table 12.1 Some secure modes: AE stands for authenticated encryption and DE for disk encryption.

Mode	Source	Type	Notes
OCB	[50]	AE	One Pass
IAPM	[30]	AE	One pass
IACBC	[29]	AE	One Pass
XCBC	[20]	AE	One Pass
XECB	[20]	AE	One Pass
CCM	[43]	AE	Two Pass
EAX	[6]	AE	Two Pass
CWC	[31]	AE	Two Pass
GCM	[41]	AE	Two Pass
CMC	[24]	DE	Encrypt-Mask-Encrypt
EME	[25]	DE	Encrypt-Mask-Encrypt
EME*	[22]	DE	Encrypt-Mask-Encrypt
PEP	[11]	DE	Hash-ECB-Hash
HCTR	[54]	DE	Hash-CTR-Hash
HCH	[12]	DE	Hash-CTR-Hash
TET	[23]	DE	Hash-ECB-Hash
HEH	[52]	DE	Hash-ECB-Hash

Other than the one pass schemes there exist other AE schemes which require two passes over the data. For such modes, in one pass the ciphertext is computed and in the other pass the tag is computed. Surely such modes are inefficient than the one pass modes. All known one pass schemes except [10] are covered by patent claims.[6] That is why two pass schemes are still of interest though one pass schemes exist. Some of the two pass AE modes are CCM [15, 43], EAX [6], GCM [41] etc. We shall discuss a two pass AE mode called CCM mode in detail including its hardware implementation in Section 12.8.

Till date there are 10 disk encryption modes proposed. They are CMC [24], EME [25] EME* [9, 22], XCB [39], ABL [40], HCTR [54], PEP [11], HCH [12], TET [23] and HEH [52]. The construction of these modes falls under three basic paradigms. The first paradigm is called encrypt-mask-encrypt where two layers of encryption are used with a layer of masking in between. CMC, EME and EME* fall under this category. Another way of construction is to use electronic code book-type encryption in between two hash layers, such constructions are called as hash-ECB-hash constructions. PEP and TET fall under this category of constructions. The other category, hash-counter-hash, uses a counter mode in between two hash layers. HCTR, XCB, ABL and HCH fall under this category of constructions. We provide a description of one DE mode EME in Section 12.7.2 which falls under the encrypt-mask-encrypt category.

[6] There is no known patent granted or pending on [10], but it may be covered by some existing patent claims unknown to the authors.

12.7.1 The Offset Codebook Mode

The offset codebook (OCB) mode was proposed by Rogaway and Black [50]. This is a fully defined efficient mode which provides both privacy and authenticity. The original OCB mode was modified a bit in [48] and called OCB1. OCB1 is not much different from the original OCB. But certain tricks in the construction help to reduce the complexity of the security proof. Also the description of OCB1 is easier than the original OCB proposal.

Figure 12.5 shows the encryption and decryption algorithm using OCB1. The encryption algorithm takes in a m block message (the last block can be an incomplete block, i.e., the last block can have a block length less than the block length of the block cipher), a block cipher key K and a nonce N. It produces a m block ciphertext along with a τ bit tag. If n is the block length of the block cipher E_K, then all n-bit strings in the algorithm are viewed as elements in $GF(2^n)$. So, all n-bit strings in the algorithm can be seen as polynomials of degree less than n whose coefficients are from $GF(2)$ (see Section 12.4 for a detailed discussion). The operation $\oplus$ is addition in the field $GF(2^n)$ and the operations $xE_K(N)$ and $(x+1)E_K$ are multiplications of the polynomials x and $1+x$ with the polynomial $E_K(N)$ modulo a fixed irreducible polynomial in $GF(2^n)$. The algorithm is self-explanatory, but certain points are important to see. The encryption of the last block (i.e., the mth block in this case) is different from the encryption of the other blocks. In step 10 of the algorithm, C_m would be t bits long if P_m is t bit long. The operation $C_m 0^*$ in step 11 means to add $(n-t)$ zeros to C_m to make C_m a full block. These discussions are all valid for the decryption algorithm also.

Algorithm OCB1.Encrypt$_K^N(P)$	**Algorithm** OCB1.Decrypt$_K^N(\mathscr{C})$
1. Partition P into $P_1, P_2, \ldots, P_m$	1. Partition $\mathscr{C}$ into $C_1, C_2, \ldots, C_m, T$
2. $\Delta \leftarrow xE_K(N)$	2. $\Delta \leftarrow xE_K(N)$
3. $\Sigma \leftarrow 0^n$	3. $\Sigma \leftarrow 0^n$
4. **for** $i = 1$ to $m-1$,	4. **for** $i = 1$ to $m-1$,
5. $\quad C_i \leftarrow E_K(P_i \oplus \Delta) \oplus \Delta$	5. $\quad P_i \leftarrow E_K^{-1}(C_i \oplus \Delta) \oplus \Delta$
6. $\quad \Delta \leftarrow x\Delta$	6. $\quad \Delta \leftarrow x\Delta$
7. $\quad \Sigma \leftarrow \Sigma \oplus P_i$	7. $\quad \Sigma \leftarrow \Sigma \oplus P_i$
8. **end for**	8. **end for**
9. Pad $\leftarrow E_K(\mathsf{len}(P_m) \oplus \Delta)$	9. Pad $\leftarrow E_K(\mathsf{len}(C_m) \oplus \Delta)$
10. $C_m \leftarrow P_m \oplus$ Pad	10. $P_m \leftarrow C_m \oplus$ Pad
11. $\Sigma \leftarrow \Sigma \oplus C_m 0^* \oplus$ Pad	11. $\Sigma \leftarrow \Sigma \oplus C_m 0^* \oplus$ Pad
12. $\Delta \leftarrow (1+x)\Delta$	12. $\Delta \leftarrow (1+x)\Delta$
13. Tag $\leftarrow E_K(\Sigma \oplus \Delta)$	13. Tag $\leftarrow E_K(\Sigma \oplus \Delta)$
14. $T \leftarrow$ Tag[first τ bits]	14. $T' \leftarrow$ Tag[first τ bits]
15. **return** $\mathscr{C} \leftarrow C_1 \| C_2 \| \ldots \| C_m \| T$	15. **if** $T = T'$ **return** $P \leftarrow P_1 \| P_2 \| \ldots \| P_m$ **else return** INVALID

Fig. 12.5 Encryption and decryption using OCB1.

OCB1 requires $m+1$ block cipher calls to encrypt a m block message. The other operations it requires have insignificant computational overhead. It requires only one pass over the data and can produce cipher in an online manner. Note that OCB1 requires only the length information of the last block to encrypt or decrypt. Also OCB1 uses only one block cipher key. Assuming a nonce respecting adversary (i.e., an adversary who does not repeat nonces) OCB can be proved to be secure in terms of both privacy and authenticity. These discussions are also valid for the original OCB.

12.7.1.1 Hardware Implementation Aspects of OCB

The parallel nature of the OCB mode of operation allows us to use a pipeline approach when implementing it in hardware. Furthermore, as is discussed in Section 12.4, the operation $xE_K(N)$ of algorithm of Figure 12.5 can be implemented at a negligible computational cost in hardware. In the following, we give a rough estimation of the hardware implementation cost of the OCB mode of operation.

Let us assume that we have an AES block cipher encryption core that uses a pipeline architecture of 10 stages. Then, referring to the algorithm of Figure 12.5, step 2 must be computed in a sequential fashion, implying that 10 clock cycles will be required for calculating Δ. Thereafter, the $m-1$ block cipher calls in steps 4–9 can be accomplished using the benefits of the parallelism associated to the pipeline architecture. So, we can argue that all the C_i for $i = 1, 2, \ldots, m$, can be computed in about $(m-1)+10$ clock cycles. Finally the tag computation of step 13 will require 10 extra clock cycles. Hence, according to the above analysis, we could achieve both authentication and encryption after about $(m-1)+20$ clock cycles when using the OCB mode of operation.

12.7.2 ECB-Mask-ECB Mode

Now we will discuss a disk encryption mode called ECB-mask-ECB (EME) [25]. As the name suggest, the mode consists of two electronic code book layers with a masking layer in between. The encryption and decryption algorithm are given in Figure 12.7. A pictorial description of EME is given in Figure 12.6.

EME takes in a m block message along with a tweak T. Note that the tweak here is different from the nonce N in case of OCB. There is no restriction regarding repetition of the tweak. Here also if the block length of the block cipher used is n then each n-bit string in the algorithm is considered as an element in the field $GF(2^n)$, i.e., they can be treated as polynomials of degree less than n with coefficients from the field $GF(2)$ and the operations x^iL and x^iM denote the multiplication of the polynomial x^i with the polynomials L and M, respectively, modulo a fixed irreducible polynomial.

The algorithm is self-explanatory, but an important feature to note is that in the masking layer, the mask M is dependent on all the plaintext blocks and the mask is

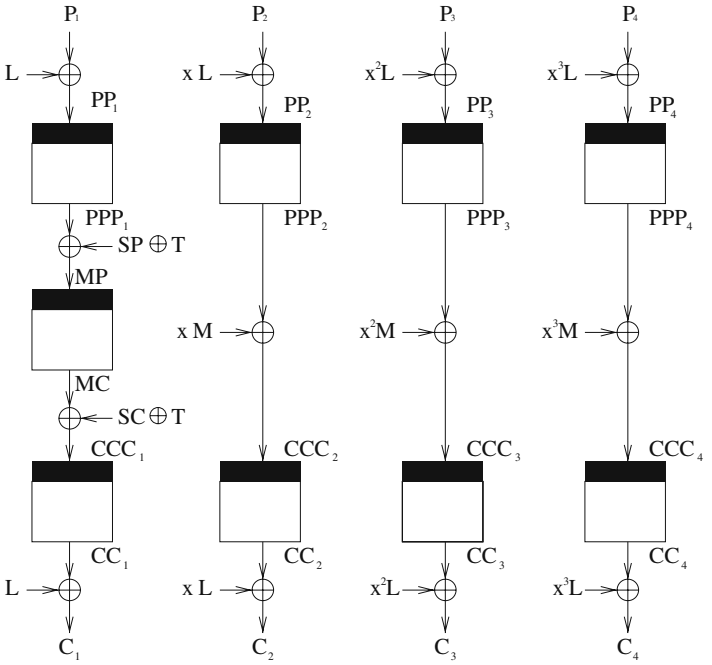

Fig. 12.6 Encryption of four blocks of plaintext using EME. Here, $L = xE_K(0^n)$, $SP = PPP_2 \oplus PPP_3 \oplus PPP_4$, $M = MP \oplus MC$ and $SC = CCC_2 \oplus CCC_3 \oplus CCC_4$.

distributed to all the blocks suitably. This makes each block of ciphertext dependent on all blocks of plaintexts. This is a necessary mechanism for any disk encryption mode.

EME has some message length restrictions. If the block length of the underlying block cipher is n then EME cannot encrypt more than n blocks of messages. Also the message length should always be a multiple of n. This message length restriction may not amount to a serious restriction in case of disk encryption scenarios as generally sector lengths are 512 bytes long.

To encrypt a m block message EME requires $2m+1$ block cipher calls, the other computational overheads are not significant. EME like all other DE modes needs to process the whole plaintext before it can output any ciphertext. Thus it is not an *online* mode of operation. EME is proved to be a secure tweakable enciphering scheme.

12.7.2.1 Hardware Implementation Aspects of EME

The EME mode of operation can be partially implemented in parallel in a hardware implementation. However, we stress that some computations of the algorithm of Figure 12.7 represent a bottleneck from the hardware implementation perspective as is discussed next.

Algorithm EME.Encrypt$_K^T(P)$	**Algorithm** EME.Decrypt$_K^T(C)$
1. Partition P into $P_1, P_2, \ldots, P_m$	1. Partition C into $C_1, C_2, \ldots, C_m$
2. $L \leftarrow xE_K(0^n)$	2. $L \leftarrow xE_K(0^n)$
3. **for** $i \leftarrow 1$ to m **do**	3. **for** $i \leftarrow 1$ to m **do**
4. $PP_i \leftarrow x^{i-1}L \oplus P_i$	4. $CC_i \leftarrow x^{i-1}L \oplus C_i$
5. $PPP_i \leftarrow E_K(PP_i)$	5. $CCC_i \leftarrow E_K^{-1}(CC_i)$
6. **end for**	6. **end for**
7. $SP \leftarrow PPP_2 \oplus PPP_3 \oplus \ldots PPP_m$	7. $SC \leftarrow CCC_2 \oplus CCC_3 \oplus \ldots CCC_m$
8. $MP \leftarrow PPP_1 \oplus SP \oplus T$	8. $MC \leftarrow CCC_1 \oplus SC \oplus T$
9. $MC \leftarrow E_K(MP)$	9. $MP \leftarrow E_K^{-1}(MC)$
10. $M \leftarrow MP \oplus MC$	10. $M \leftarrow MP \oplus MC$
11. **for** $i \leftarrow 2$ to m **do**	11. **for** $i \leftarrow 2$ to m **do**
12. $CCC_i \leftarrow PPP_i \oplus x^{i-1}M$	12. $PPP_i \leftarrow CCC_i \oplus x^{i-1}M$
13. **end for**	13. **end for**
14. $SC \leftarrow CCC_2 \oplus CCC_3 \oplus \ldots CCC_m$	14. $SP \leftarrow PPP_2 \oplus PPP_3 \oplus \ldots PPP_m$
15. $CCC_1 \leftarrow MC \oplus SC \oplus T$	15. $PPP_1 \leftarrow MP \oplus SP \oplus T$
16. **for** $i \leftarrow 1$ to m **do**	16. **for** $i \leftarrow 1$ to m **do**
17. $CC_i \leftarrow E_K(CCC_i)$	17. $PP_i \leftarrow E_K(PPP_i)$
18. $C_i \leftarrow x^{i-1}L \oplus CC_i$	18. $P_i \leftarrow x^{i-1}L \oplus PP_i$
19. **end for**	19. **end for**
20. **return** $C_1, C_2, \ldots, C_m$	20. **return** $P_1, P_2, \ldots, P_m$

Fig. 12.7 Encryption and decryption using EME.

As we did in the analysis of OCB, let us assume that we have an AES block cipher encryption core that uses a pipeline architecture of 10 stages. Then, referring to the algorithm of Figure 12.7, the computation of the parameter L in step 2 must be accomplished in a sequential fashion, implying that at least 10 clock cycles will be required for completing that calculation. Thereafter, the m block cipher calls included in steps 3–6 can be accomplished using the benefits of the parallelism associated to the pipeline approach. So, we can argue that all the PPP_i for $i = 1, 2, \ldots, m$ can be computed in about $(m-1)+10$ clock cycles. On the contrary, the cipher call in step 9 for obtaining MC must be performed in a sequential fashion, which implies 10 extra clock cycles. Similarly, the $m-1$ block cipher calls in steps 11–13 represent a computational effort of about $(m-2)+10$ clock cycles, whereas the last block cipher call in step 17 implies 10 clock cycles more.

In summary, the computational cost of the algorithm in Figure 12.7 can be estimated in about $2m-3+50$ clock cycles. Considering that for a typical EME application, the plaintext will have a length of 32 blocks,[7] then the EME algorithm of Figure 12.7 will encrypt a disk sector in about 111 clock cycles when using a pipeline AES hardware architecture. Some precomputations may save some cost in EME. As L is a quantity only dependent on the key K, L can be easily precomputed thus saving some clock cycles. A more detailed description of the hardware design of EME along with designs of other DE schemes can be found in [37, 38].

[7] Here we are assuming that the size of a disk sector is 512 bytes or thirty-two 128-bit AES blocks.

12.8 The CCM Mode: A Case Study

Here we shall discuss a mode called CCM in detail including its hardware implementation. We shall design CCM with AES as the underlying block cipher. For a summary of the AES algorithm specification we refer the interested reader to Section 12.3.

The rest of this section is organized as follows. In Section 12.8.1 we briefly describe the CCM mode of operation. Then, in Section 12.8.2, we present a reconfigurable hardware implementation of an AES sequential encryptor core. In Section 12.8.3, we give a design description of the AES–CCM mode reconfigurable hardware implementation reported in [34]. Finally, in Section 12.8.4 we compare the design described in this chapter with other architectures already reported in the open literature.

12.8.1 The CCM Mode

CCM stands for counter with CBC–MAC. This means that two different modes are combined into one, namely, the CTR mode and the CBC–MAC. CCM is a generic authenticated encrypt block cipher scheme. It has been specifically designed for being used in combination with a 128-bit block cipher, such as AES. CCM mode can be easily extended to other block sizes, but this would require further definitions not to be addressed here.

CCM mode was proposed by Whiting et al. [15]. Their original paper was sent to NIST for evaluation as a generic new mode. Presently, it has become part of the new 802.11i IEEE standard [43]. CCM is an authenticated encryption scheme which also supports associated data.

The generic CCM mode allows user definition of two main parameters. The first choice is M, the size of the tag or the authentication field. Selecting an adequate value for M involves a trade-off between message expansion and the probability that an attacker can undetectably modify the message. Valid values for M are 4, 6, 8, 10, 12, 14 and 16 bytes. This parameter is encoded as $(M-2)/2$.

In the rest of this section we will use $|P|$ to indicate the length in bytes of the plaintext message P. The parameter L gives the size in bytes of the field that indicates the numerical value of $|P|$. This value involves a trade-off between the maximum message size and the size of the *nonce*, which is an unique integer value associated with each message. The value of L ranges from two to eight.

12.8.1.1 CCM Input Parameters

Before sending a message, a sender must provide the following information:

- A suitable encryption key K for the block cipher to be used.
- A nonce N of $15-L$ bytes. Nonce value must be unique, meaning that the set of nonce values used with any given key shall not contain duplicate values.

- The message P consisting of a string of $|P|$ bytes where $0 \leq |P| < 2^{8L}$.
- Additional authenticated data a, consisting of a string of $|a|$ bytes where $0 \leq |a| < 2^{64}$. This additional data are authenticated but not encrypted and are not included in the output of this mode.

12.8.1.2 CCM Authentication

The first step consists of computing the tag or the authentication field T. This is done using CBC–MAC mode [15, 43]. We first define a sequence of blocks $B_0, B_1, \ldots, B_m$, and thereafter CBC–MAC is applied to those blocks so that the authentication field T can be obtained.

The first block B_0 is formatted as shown in Figure 12.8a, where $l(P)$ is encoded in most-significant-byte first order.

Within the first block B_0, the *Flags* field is formatted as shown in Figure 12.8b The *Reserved bit* field is reserved for future expansions and should always be set to zero. The *Adata* bit is set to zero if $l(a) = 0$ and set to one if $l(a) > 0$.

Authentication data a are formatted by concatenating the string that encodes $l(a)$ with a itself, followed by organizing the resulting string in chunks of 16-byte blocks. If necessary, the last block should be padded with zeros so that its length achieves 16 bytes. The blocks so constructed are appended to the first block B_0.

Message blocks are added right after the (optional) authentication blocks a. Message blocks are formatted by splitting the message P into 16-byte blocks and then padding the last block with zeros if necessary. If the message P consists of the empty string, then no blocks are added in this step. Then a sequence consisting of the concatenation of the blocks $B_0, B_1, \ldots, B_m$ is produced. Finally, the CBC–MAC is computed as

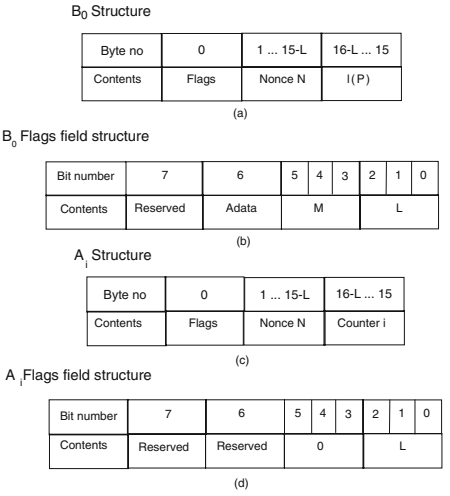

Fig. 12.8 Structure of the B_0 block and its flags.

$$\begin{aligned} X_1 &:= AES_E(K, B_0) \\ X_{i+1} &:= AES_E(K, X_i \oplus B_i) \text{ for } i = 1, \ldots, m \\ T &:= firstMbytes(X_{m+1}) \end{aligned} \tag{12.11}$$

where AES_E is the AES block cipher selected for encryption and T is the MAC value defined above. Note that the last block B_m is XORed with X_m and the result is encrypted with the block cipher. If it is needed, the ciphertext would be truncated in order to obtain T.

Figure 12.9 shows CCM authentication and verification processes dataflow. We stress that because of the CBC feedback nature of the CCM mode, we cannot use a pipeline approach when implementing a hardware CCM architecture.

12.8.1.3 CCM Encryption

CCM encryption is achieved by means of counter (CTR) mode as

$$\begin{aligned} S_i &:= AES_E(K, A_i) \text{ for } i = 0, 1, 2, \ldots, m \\ C_i &:= S_i \oplus P_i \end{aligned} \tag{12.12}$$

Figure 12.8c shows how the values A_i are formatted, where i is encoded in most-significant-byte first order. Within each block A_i, the Flags field is formatted as shown in Figure 12.8d. Once again, *reserved bits* field is reserved for future expansions and must be set to zero.

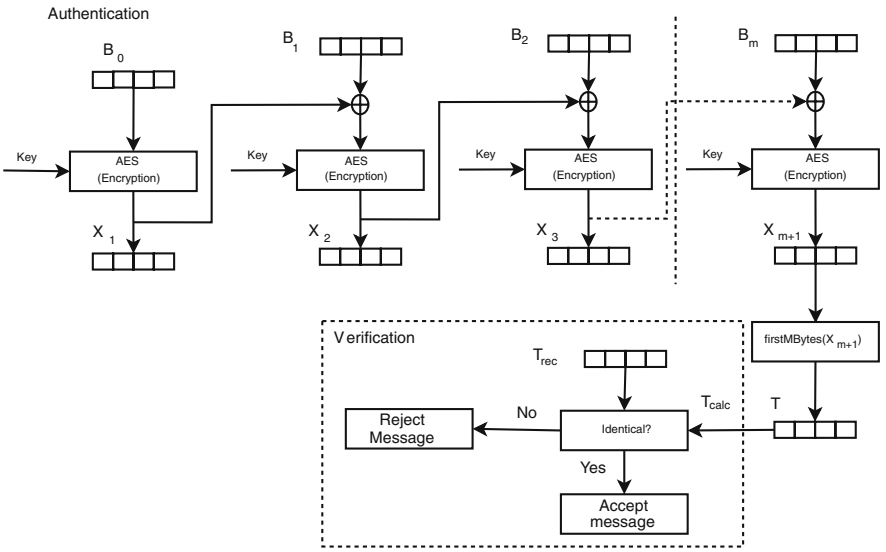

Fig. 12.9 Authentication and verification processes for the CCM mode.

Plaintext P is encrypted by XORing each of its bytes with the first $l(P)$ bytes of the sequence produced by concatenating the cipher blocks $S_1, S_2, S_3, \ldots$ produced by Equation 12.12. Notice that S_0 is not used for message encryption. The authentication value is computed by encrypting T with the key stream block S_0 truncated to the desired length as

$$U := T \oplus firstMbytes(S_0) \tag{12.13}$$

The final result C consists of the encrypted message P, followed by the encrypted authentication value U.

12.8.1.4 Decryption and Verification

To decrypt a message the following informations are required:

- The encryption key K
- The nonce N
- The additional authenticated data a
- The encrypted and authenticated message C

Decryption starts by recomputing the key stream to recover the message P and the MAC value T. Message and additional authentication data are then used to recompute the CBC–MAC value and check T.

If the T value is not correct, the receiver should not reveal the decrypted message, the value T or any other information.

It is important to notice that the AES encryption algorithm is required in both encryption as well as in decryption. Therefore, AES decryption functionality is not necessary in CCM mode, which results in valuable hardware resources saving.

Figure 12.10 shows the CCM encryption/decryption process dataflow.

12.8.2 AES Encryptor Core Implementation

As was mentioned before, in order to implement the CCM scheme, a 128-bit block cipher is needed. In this section we describe the general architecture of an AES sequential encryptor core as shown in Figure 12.11.

12.8.2.1 Implementation of the AES Rounds

Main nine rounds of AES must be implemented in an iterative way. Therefore, only one round is shown in Figure 12.12. That circuit uses a multiplexor to select whether we are going to process the first round or the other eight ones. At the end of the circuit we use a latch block to store the current computed state matrix.

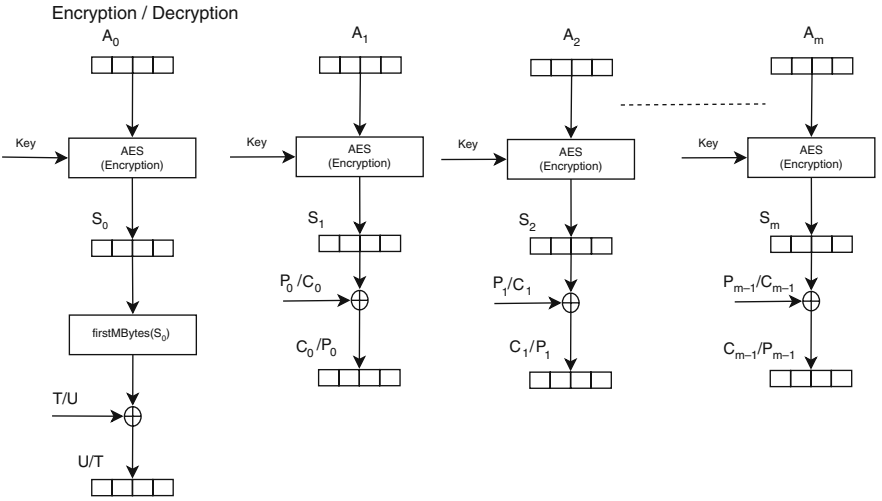

Fig. 12.10 Encryption and decryption processes for the CCM mode.

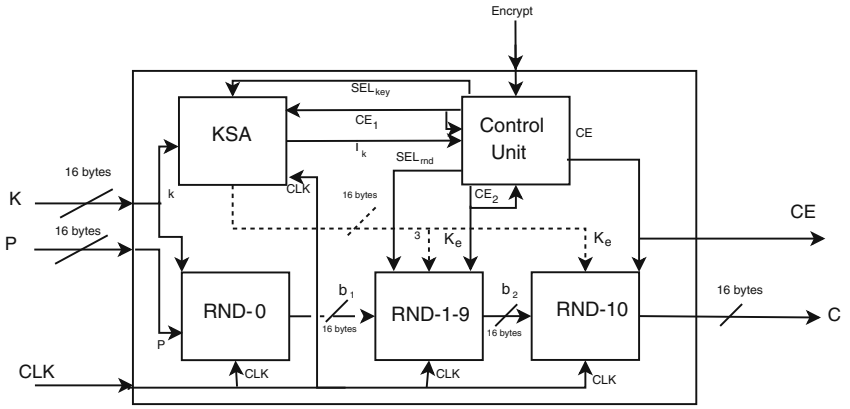

Fig. 12.11 General architecture of an AES encryptor core.

As is shown in Figure 12.12, rounds 1–9 were implemented using two main building blocks. The first one is the BS/SR block that can be instrumented by using the BRAMs (Block RAMs) embedded in the targeted FPGA device. Sixteen 8×256 BRAMs were configured for implementing AES S-box as a look-up table. By doing so, it is possible to compute 16-byte substitutions at the same time. Mix Columns and AddRoundKey Steps can be implemented jointly by doing some minor modifications. For polynomial multiplication the $xtime(v)$ operation described at the end of Section 12.4 was used.

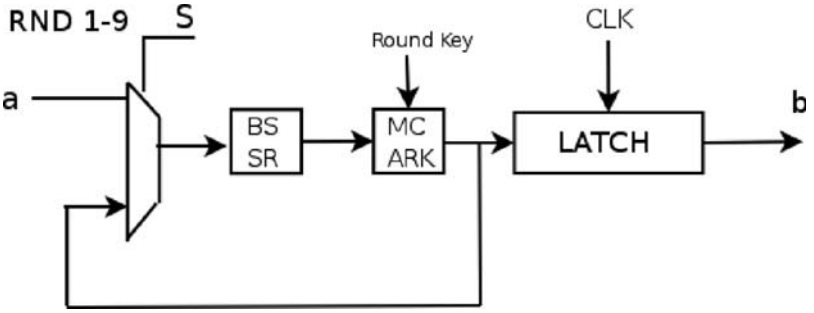

Fig. 12.12 Block diagram of the rounds 1–9.

12.8.2.2 Key Scheduling Implementation

In [47] several optimizations based on redundant computation for parallelizing the Key Scheduling process were implemented. As a result, it takes two steps to compute the round key [47].

Figure 12.13 shows Key Scheduling algorithm block diagram for an Iterative Encryptor Core. That circuit has a multiplexor that selects whether the key to be processed is the original user secret key or the current round key. At the output of the circuit we use a latch that stores the round key so produced. That key will be available until a new round key is generated. The latch is activated in the falling edge of the clock and its CE is activated in high state. The implementation shown can provide a round key every falling edge of the master clock.

12.8.2.3 AES Control Unit

The AES control unit synchronizes the whole process and controls the information flow. In addition, it produces the signals to control the multiplexors and latches that are used in the AES components. These synchronization signals are crucial because each component should select the correct state matrix.

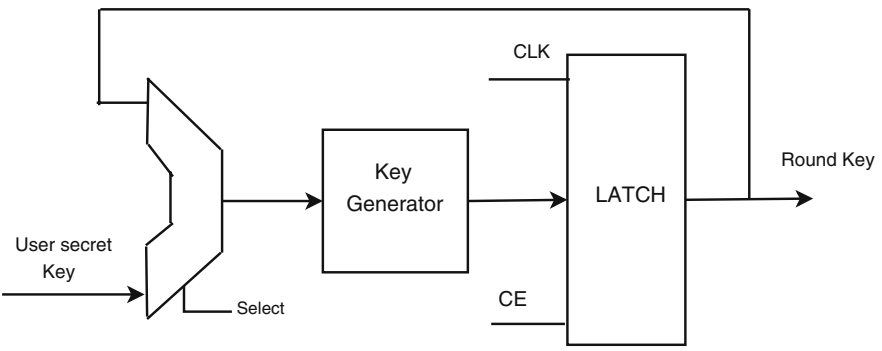

Fig. 12.13 Iterative key scheduling block diagram.

The signal generated to control the Final Round Latch is also used as an indicator that the ciphertext is ready. This is done by a change in the CE output of the AES block. When the plaintext is being processed, the CE output value is "0", but when a ciphertext is ready, this value changes to "1". In addition, the AES block has an extra input called "Encrypt" that indicates to the control unit that a new plaintext is given and that a new process has to begin. This control signal must be high by one single CLK's cycle and after that it must be set to low.

12.8.3 Hardware Implementation of the CCM Mode

In this section we discuss design details utilized for CCM mode and AES encryptor core implementations. It is assumed that the user must provide the additional authentication data *a* as two blocks of 16 bytes each (see Section 12.8.1). This size was selected considering the typical length of a TCP/IP header information. Furthermore, it was assumed that the message *P* to be processed has a maximum length of 1024 bytes.

Figure 12.14 shows the CCM mode general architecture, which comprises three main building modules, namely, authentication module, encryption module and a control unit module. All those three blocks together perform necessary operations for generating a valid cipher text and an encrypted authentication value. Notice that extra hardware is needed for the verification and decryption phases. In the rest of this section we will explain how those three blocks were implemented in [34].

12.8.3.1 CCM Authentication

Figure 12.15a depicts the CCM authentication module architecture. This module consists of an authentication block generator, a CBC–MAC module and a control unit.

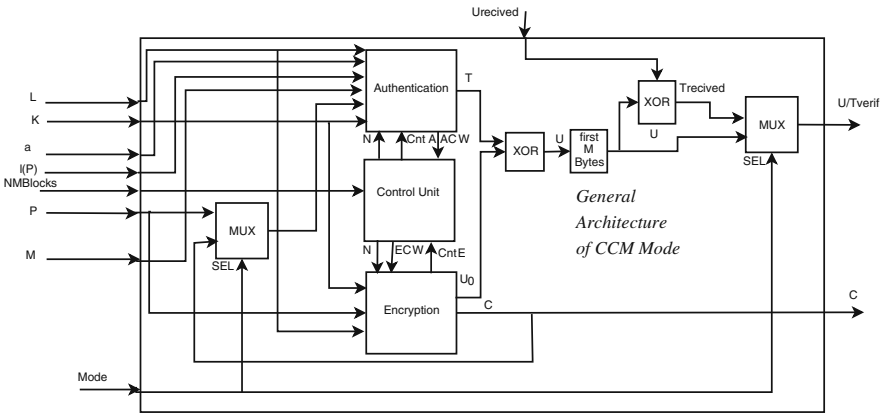

Fig. 12.14 CCM mode general architecture.

Authentication Block Generator: Authentication block generator is the architecture component responsible for generating the B_i blocks (see [15, 43] for details). Those blocks are generated according to the instructions indicated in the control word that the control unit sends in the "CW" line. That control word stipulates which block should be generated, the B_0 block or the blocks that correspond to the additional data a or the ones corresponding to the message P. Each block is generated only when the previous block has already been ciphered by the CBC–MAC module described next.

CBC–MAC: Blocks B_i that were generated by the block generator are the inputs for the CBC–MAC. Any input block B_i (except for the block B_0) is XORed with the X_i that was computed previously. The result of this operation is encrypted using AES, and the resulting cipher text X_{i+1} is fed back to the next block B_{i+1}. Figure 12.15b depicts the CBC–MAC process just outlined.

Authentication Control Unit: Control unit orchestrates the authentication process by receiving control signals from the general control unit. This block generates the appropriate control word for the block generator module and the one that indicates to the CBC–MAC Encryptor that a new B_i block can be processed. A 5-bit control word is utilized, where the LSB is used for controlling the CBC–MAC's latch. The second bit is used to start encryption with the AES block; the third bit controls which input will be selected by the MUX included in the CBC–MAC component and finally, the last two bits indicate which type of block should be generated.

Authentication control unit receives a signal when a B_i block has been processed within the CBC–MAC module. Thereafter, the control unit module produces the appropriate control word to generate the next block B_{i+1} and process it. Control unit runs a counter that indicates which control word should be generated. The process of authentication begins when the general control unit indicates so with the "ACW" word. After receiving this signal, the whole process is controlled by the local control unit. With the aim of parallelizing the authentication and encryption processes, the authentication begins first.

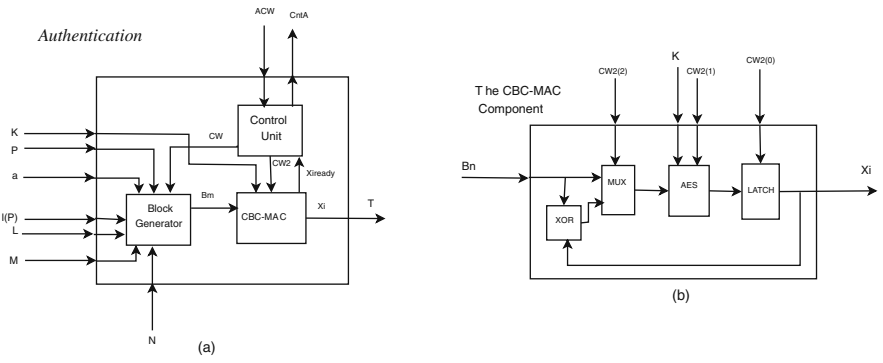

Fig. 12.15 Authentication block.

12.8.3.2 CCM Encryption

Figure 12.16a shows the CCM encryption architecture. This module consists of an encryption block generator, a CTR block and a control unit.

Encryption Block Generator: This module is responsible for generating the A_i blocks, (see Section 12.8.1 for details) generated according to the counter function. In this implementation, the counter begins in 0 and it is incremented one by one. The blocks are formed with the nonce and the counter value. Each block is generated when the CTR module has finished processing the previous one. Let us recall that encryption begins only when the authentication module is processing the second block with the additional data a. Based on this observation, the process of authentication and encryption can be accomplished in parallel by generating the A_1 block first and all subsequent A_i blocks but the first one (A_0). When the CCM authentication module has finished the last block processing, encryption block generator may proceed to generate the A_0 block in order to get S_0.

The CTR Mode: The CTR mode is the last step in the encryption process, it encrypts the A_i blocks with the block cipher (i.e., AES) to generate the S_i stream blocks. When a S_i block is ready, it is XORed with the appropriate message m block. Figure 12.16b shows the internal composition of the CTR mode. This mode uses as cipher block the AES implementation described in Section 12.8.2. The U_0 value shown in Figure 12.16b corresponds to the S_0 block. That is why U_0 is not XORed with the message, when it is the last block (actually the first of the counter function), this value is used to encrypt the authentication value T computed as a part of the authentication process as shown in Figure 12.9.

Encryption Control Unit: The implementation of this module is quite similar to the one for the authentication process. This control block is responsible to keep the counting and to tell the block generator which block is the next to be generated. At the same time, it starts the CTR mode for processing a block in order to get a valid ciphertext. When a ciphertext is ready, it tells to the general control unit that a new ciphertext can be stored. The implementation is based on the counter process that begins when the general control unit indicates that it is time to encrypt the message

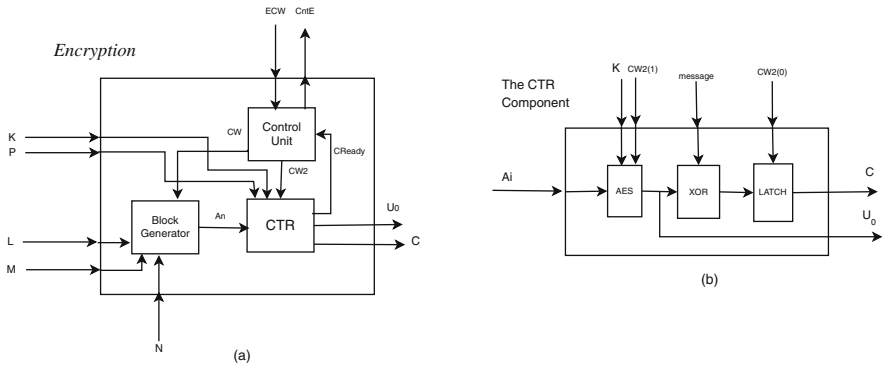

Fig. 12.16 Encryption block.

m, and it keeps counting until the general control unit indicates to stop. The control unit receives a control word that indicates what to do, this control word is 2-bits long, and the four possibilities are "00" or "10" do nothing, remain in initial state, "01" generate the S_0 block and "11" begin and continue counting.

12.8.3.3 General Control Unit

This module is the one that controls the authentication and encryption processes. It synchronizes the information flow in order to parallelize the entire process and to achieve a good performance. The control unit commands when the authentication process must begin the execution. After the authentication process has processed the first block (B_0) and continues with the second of the two additional data a blocks, the encryption process begins.

Notice that the authentication process must authenticate the other a block and all message blocks. The encryption process, on the other hand, must encrypt only the message blocks and since it starts first, one could think that the encryption process would finish first. However, since it is necessary for the extra processing of the block S_0, the architecture discussed here manages to finish both processes at the same time. In this way we can compute the encrypted authentication data U which is done with extra hardware after the authentication and encryption processes. Figure 12.17 shows this process time line. Every unit in the time line represents 12 clock cycles.

The control unit behaves in the same way for all, the authentication and encryption processes and for the decryption and verification processes, which means that the control unit does not know if it is encrypting or verifying.

12.8.3.4 Decryption and Verification

Decryption and verification processes are included in the architecture presented in [34] and they were implemented as extra hardware. The additional hardware is used

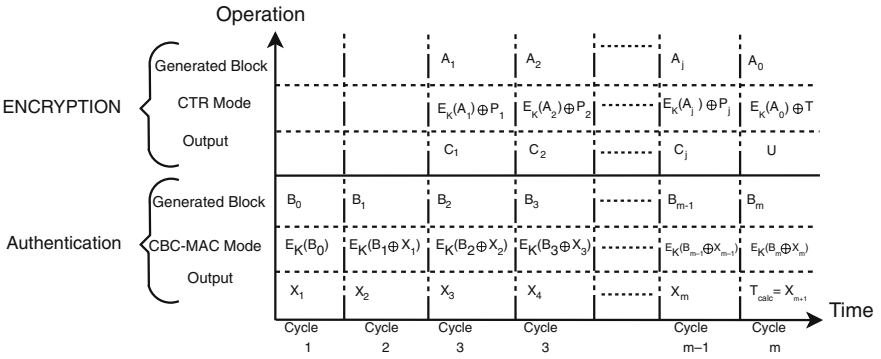

Fig. 12.17 Encryption–authentication time line.

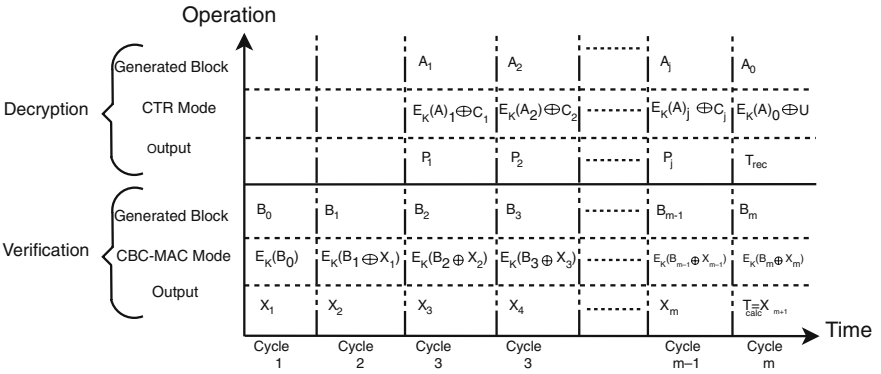

Fig. 12.18 Decryption–verification time line.

to select if the data are going to be authenticated and encrypted or decrypted and verified. In Figure 12.14 that extra hardware as a MUX is shown before the authentication process. This MUX is used to select if the source message is the one provided by the user (in case of authentication) or if it is the message that has been decrypted (when verifying). This is selected by the mode input, when "0" the process is going to authenticate and encrypt. If mode="1", then it will decrypt and verify.

The second MUX is used for selecting the output for the U value; when authenticating, the selection should be the computed value U, this is done with the function *firstMBytes* and the XOR operation between the computed T and the S_0 block. When verifying, computed U value is XORed with the U *received* (sent by the transmitter entity) in order to verify whether the message integrity has been corrupted or not; if the XOR output is equal to zero, then the message is correct, otherwise, it is assumed that the received message is corrupted.

As in the case of the authentication and encryption, the verification process begins first and the decryption process starts after the verification process has processed two blocks. In this way, verification can certify the received message that was just decrypted. Figures 12.17 and 12.18 show the time line of these processes, every unit in time line represents 12 clock cycles.

12.8.4 Experimental Results and Comparison

The design presented in this chapter (including key schedule) occupies 2154 slices, while it makes use of 32 block RAMs. It was implemented on a 3s4000fg900-4 Spartan 3 device using VHDL language and Xilinx's ISE 6.3i development tool along with the ModelSim Xilinx Edition II v5.8c [34]. Table 12.2 summarizes the hardware resources required by the design's main building blocks.

As was mentioned before, it was considered that the additional information a consisted of two 16-byte blocks. The maximum length of the plaintext is 1024 bytes which results in 64 blocks of 16 bytes each. Then, a total of 67 effective blocks must be processed (this estimation excludes the B_0 block, which was considered as

Table 12.2 Hardware resources of the design described in Section 12.8.3.

Block	Slices	BRAMs
Authentication	1031	16
Encryption	713	16
Control unit and extra Hw	410	0
CCM mode	2154	32
Maximum clock frequency	100.08 MHz	

Table 12.3 AES–CCM comparison.

Author	Device	Mode	Slices (BRAMs)	T* (Mbps)
AES–CCM core Helion	Virtex 4	CCM	480 (5)	670
	Virtex 5	CCM	321 (0)	760
Fu et al. [18]	Virtex 2	CTR	2415 (NA)	1490
Charot et al. [13]	Altera APEX	CTR	N/A	512
Bae et al. [3]	Altera Stratix	CCM	5605(LC)	285
This Design	Spartan 3	CBC	2154 (32)	1067

*Throughput

an overhead in the process, so it was omitted from the throughput computation). The processing of a total of 67 blocks can be accomplished by the design described in this section in 804 clock cycles (each block is computed in 12 cycles). Finally, we provide in Table 12.3 a comparison with several CCM–AES designs reported in the open literature.

12.9 Conclusions

Block ciphers are one of the most important symmetric cryptographic primitives. They are widely used for bulk encryptions. Block ciphers are always to be used along with an appropriate mode of operation when one needs to encrypt messages bigger than the block length of the block cipher. Also a mode of operation can provide security services other than privacy/confidentially. Thus, modes of operations are important cryptographic objects. A modern mode needs to be secure in terms of strong security definitions and also needs to be efficient in various respects. These two goals are sometimes contradictory and thus designing an efficient mode which is also secure is a challenging task.

From the beginning of this century many researchers have provided with many secure and efficient designs, though not many of the proposed modes are actually being used in applications. The standardization efforts for modes for different

applications is still going on and we hope that within a few years more modes would be standardized and thus widely used in various applications.

Efficient implementation of modes is another very important aspect. As per hardware design, AES has seen many efficient implementations, but all of these implementations may not be the best for every mode. A mode of operation may contain objects other than the block ciphers like field multipliers, hash functions, etc. Also the data dependencies for different kinds of modes may be quite different. This demands specific implementations of the mode and in particular the block cipher. Not many efficient implementations of different modes have yet been reported in the literature. In fact there are many modes which have not yet been implemented and therefore no test vectors are available for those modes.

Summarizing, in this chapter we provided a brief overview of hardware implementation aspects of modes of operations. We informally defined the various security goals for various modes, provided a partial list of different secure modes and described in detail the hardware implementation aspects of a secure mode called CCM.

Acknowledgments The authors gratefully acknowledge the valuable participation of Emmanuel López-Trejo in the AES–CCM hardware design described in this paper. Authors also acknowledge support from CONACyT through the CONACyT project number 60240-J1.

12.10 Exercises

1. In this chapter we showed that CBC mode is insecure if the IV is repeated. We also showed how to fix the problem. Show that the same is true for counter mode.
2. Assume that a plaintext of m blocks has been encrypted using a mode M. During transmission the $\frac{m}{2}$th block gets corrupted (assume $2|m$). Discuss which of the plaintext blocks will be corrupted after decryption, assuming the mode M to be ECB, CBC, CFB, OFB and CTR.
3. EME has a message length restriction, i.e., if the blocklength of the underlying block cipher is n it can securely encrypt only n blocks of message. Can you figure out why? (Hint: This has something to do with intermediate masking. You can look up the solution in [24].)

12.11 Projects

1. Implement any of the authenticated encryption modes in hardware. Provide test vectors for the mode selected.
2. Implement any of the disk encryption modes in hardware. Provide test vectors for the mode selected.

References

1. American National Standard for Financial Services X9.52-1998. *Triple Data Encryption Algorithm Modes of Operation*. American Bankers Association, Washington, D.C., July 1998.
2. B. Schneier. *Applied Cryptography*. Wiley, Second edition, 1996.
3. D. Bae, G. Kim, J. Kim, S. Park, and O. Song. An Efficient Design of CCMP for Robust Security Network. In *International Conference on Information Security and Cryptology*, vol. 3935, pp. 337–346, Seoul, Korea, Springer-Verlag, December 2005.
4. M. Bellare, A. Desai, E. Jokipii, and P. Rogaway. A concrete security treatment of symmetric encryption. In *38th Annual Symposium on Foundations of Computer Science, FOCS '97*, pp. 394–403, Miami Beach, Florida, 1997.
5. M. Bellare and C. Namprempre. Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm. In *ASIACRYPT '00: Proceedings of the 6th International Conference on the Theory and Application of Cryptology and Information Security*, pp. 531–545, London, UK, Springer-Verlag, 2000.
6. M. Bellare, P. Rogaway, and D. Wagner. The EAX mode of operation. In *FSE*, LNCS vol. 3017, pp. 389–407. Springer, 2004.
7. G. Bertoni, L. Breveglieri, P. Fragneto, M. Macchetti, and S. Marchesin. Efficient software implementation of AES on 32-bits platforms. In *Proceedings of the CHES 2002*, LNCS vol. 2523 pp. 159–171. Springer, 2002.
8. D. Canright. A very compact S-Box for AES. In J. R. Rao and B. Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings*, LNCS, vol. 3659 pp. 441–455. Springer, 2005.
9. A. Canteaut and K. Viswanathan, editors. *Progress in Cryptology – INDOCRYPT 2004, 5th International Conference on Cryptology in India, Chennai, India, December 20–22, 2004, Proceedings*, LNCS, vol. 3348. Springer, 2004.
10. D. Chakraborty and P. Sarkar. A general construction of tweakable block ciphers and different modes of operations. In H. Lipmaa, M. Yung, and D. Lin, editors, *Inscrypt*, LNCS, vol. 4318 pp. 88–102. Springer, 2006.
11. D. Chakraborty and P. Sarkar. A new mode of encryption providing a tweakable strong pseudo-random permutation. In M. J. B. Robshaw, editor, *FSE*, LNCS, vol. 4047, pp. 293–309. Springer, 2006.
12. D. Chakraborty and P. Sarkar. HCH: A new tweakable enciphering scheme using the Hash-Encrypt-Hash approach. In R. Barua and T. Lange, editors, *INDOCRYPT*, LNCS vol. 4329, pp. 287–302. Springer, 2006.
13. F. Charot, E. Yahya, and C. Wagner. Efficient modular-pipelined AES implementation in counter mode on ALTERA FPGA. In P. Y. K. Cheung, G. A. Constantinides, and J. T. de Sousa, editors, *FPL*, LNCS, vol. 2778, pp. 282–291, Springer, 2003.

14. W. Diffie and M. Hellman. Privacy and authentication: An introduction to cryptography. *Proceedings of the IEEE*, 67:397–427, 1979.
15. D. Whiting, R. Housley, and N. Ferguson. Submission to NIST: Counter with CBC-MAC (CCM) AES mode of operation. Computer Security Division, Computer Security Resource Center (NIST), available at: http://csrc.nist.gov/groups/ST/toolkit/BCM/documents/proposedmodes/ccm/ccm.pdf
16. FIPS Publication 46-3. *Data Encryption Standard DES*. US DOC/NIST, October 1999.
17. FIPS Publication 81. *DES Modes of Operation*. US DOC/NIST, December 1980.
18. Y. Fu, L. Hao, and X. Zhang. Design of an extremely high performance counter mode AES reconfigurable processor. In *Proceedings of the Second International Conference on Embedded Software and Systems (ICESS'05)*, pp. 262–268. IEEE Computer Society, 2005.
19. B. Gladman. The AES Algorithm (Rijndael) in C and C++, available at: http://fp.gladman.plus.com/cryptography_technology/rijndael/
20. V. D. Gligor and P. Donescu. Fast encryption and authentication: XCBC encryption and XECB authentication modes. In M. Matsui, editor, *FSE 2001*, LNCS, vol. 2355, pp. 92–108. Springer, 2001.
21. T. Good and M. Benaissa. AES on FPGA from the fastest to the smallest. In J. R. Rao and B. Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 – September 1, 2005, Proceedings*, LNCS, vol. 3659, pp. 427–440. Springer, 2005.
22. S. Halevi. EME*: Extending EME to handle arbitrary-length messages with associated data. In Canteaut and Viswanathan, editors, *Progress in Cryptology – INDOCRYPT 2004, 5th International Conference on Cryptology in India, Chennai, India, December 20–22, 2004, Proceedings*, LNCS, vol. 3348, pp. 315–327, Springer, 2004.
23. S. Halevi. TET: A wide-block tweakable mode based on Naor-Reingold. Cryptology ePrint Archive, Report 2007/014, 2007. http://eprint.iacr.org/
24. S. Halevi and P. Rogaway. A tweakable enciphering mode. In D. Boneh, editor, *CRYPTO*, LNCS vol. 2729, pp. 482–499. Springer, 2003.
25. S. Halevi and P. Rogaway. A parallelizable enciphering mode. In T. Okamoto, editor, *CT-RSA*, LNCS vol. 2964, pp. 292–304. Springer, 2004.
26. S. F. Hsiao and M. C. Chen. Efficient substructure sharing methods for optimising the inner-product operations in Rijndael Advanced Encryption Standard. *IEE Proceedings on Computer and Digital Technology*, 152(5):653–665, September 2005.
27. T. Ichikawa, T. Kasuya, and M. Matsui. Hardware evaluation of the AES finalists. In *The Third AES3 Candidate Conference*, pp. 279–285, New York, April 2000.
28. J. Daemen and V. Rijmen. *The Design of Rijndael: AES The Advanced Encryption Standard*. Springer-Verlag, First edition, 2002.

29. C. S. Jutla. Encryption modes with almost free message integrity. Cryptology ePrint Archive, Report 2000/039, 2000. http://eprint.iacr.org/
30. C. S. Jutla. Encryption modes with almost free message integrity. In B. Pfitzmann, editor, *EUROCRYPT*, LNCS, vol. 2045, pp. 529–544. Springer, 2001.
31. T. Kohno, J. Viega, and D. Whiting. CWC: A high-performance conventional authenticated encryption mode. Cryptology ePrint Archive, Report 2003/106, 2003. http://eprint.iacr.org/
32. H. Lipmaa, P. Rogaway, and D. Wagner. Comments to NIST concerning AES Modes of Operations: CTR-mode encryption, September 2000, available at: http://www.cs.ucdavis.edu/rogaway/papers/ctr.pdf.
33. M. Liskov, R. L. Rivest, and D. Wagner. Tweakable block ciphers. In *CRYPTO '02: Proceedings of the 22nd Annual International Cryptology Conference on Advances in Cryptology*, pp. 31–46, London, UK, Springer-Verlag, 2002.
34. E. López-Trejo, F. R. Henríquez, and A. Díaz-Pérez. An Efficient FPGA Implementation of CCM mode using AES. In *International Conference on Information Security and Cryptology*, LNCS, vol. 3935, pp. 208–215, Seoul, Korea, Springer-Verlag, December 2005.
35. M. Luby and C. Rackoff. How to construct pseudorandom permutations from pseudorandom functions. *SIAM Journal of Computing*, 17(2):373–386, 1988.
36. A. K. Lutz, J. Treichler, F. K. Gurkaynak, H. Kaeslin, G. Basler, A. Erni, S. Reichmuth, P. Rommens, S. Oetiker, and W. Fitchtner. 2 Gbits/s Hardware realization of RIJNDAEL and SERPENT-A comparative analysis. In *Proceedings of the CHES 2002*, LNCS, vol. 2523, pp. 171–184. Springer, 2002.
37. C. Mancillas-López, D. Chakraborty, and F. Rodríguez-Henríquez. Efficient implementations of some tweakable enciphering schemes in reconfigurable hardware. In K. Srinathan, C. P. Rangan, and M. Yung, editors, *INDOCRYPT*, LNCS, vol. 4859, pp. 414–424. Springer, 2007.
38. C. Mancillas-Lopez, D. Chakraborty, and F. Rodriguez-Henriquez. Reconfigurable hardware implementations of tweakable enciphering schemes. Cryptology ePrint Archive, Report 2007/437, 2007. http://eprint.iacr.org/
39. D. A. McGrew and S. R. Fluhrer. The Extended Codebook (XCB) mode of operation. Cryptology ePrint Archive, Report 2004/278, 2004. http://eprint.iacr.org/
40. D. A. McGrew and J. Viega. Arbitrary block length mode, 2004, available at: http://grouper.ieee.org/groups/1619/email/pdf00005.pdf
41. D. A. McGrew and J. Viega. The security and performance of the Galois/Counter Mode (GCM) of operation. In Canteaut and Viswanathan, editors, *Progress in Cryptology – INDOCRYPT 2004, 5th International Conference on Cryptology in India, Chennai, India, December 20–22, 2004, Proceedings*, LNCS, vol. 3348, pp. 343–355. Springer, 2004.
42. M. Dworkin. Recommendation for Block Cipher Modes of operation methods and techniques. National Institute of Standards and Technology (NIST), December 2001 available at: http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf

43. M. Dworkin. Recommendation for Block Cipher Modes of Operation: The CCM Mode for authentication and confidentiality. National Institute of Standards and Technology (NIST), May 2004, available at: http://csrc.nist.gov/publications/nistpubs/800-38C/SP800-38C.pdf
44. M. Naor and O. Reingold. A pseudo-random encryption mode. Manuscript available at: www.wisdom.weizmann.ac.il/naor
45. M. Naor and O. Reingold. On the construction of pseudorandom permutations: Luby-Rackoff revisited. *Journal of Cryptology*, 12(1):29–66, 1999.
46. NIST Special Publication 800-38A 2001 edition. *Recommendation for Block Cipher Modes of Operation*. US NIST, December 2001.
47. F. Rodríguez-Henríquez, N. A. Saqib, and A. Díaz-Pérez. 4.2 Gbit/s single-chip FPGA implementation of AES algorithm. *IEE Electron. Lett.*, 39(15):1115–1116, July 2003.
48. P. Rogaway. Efficient instantiations of tweakable blockciphers and refinements to modes OCB and PMAC. In P. J. Lee, editor, *ASIACRYPT*, LNCS, vol. 3329, pp. 16–31. Springer, 2004.
49. P. Rogaway. Nonce-based symmetric encryption. In B. K. Roy and W. Meier, editors, *FSE*, LNCS vol. 3017, pp. 348–359. Springer, 2004.
50. P. Rogaway, M. Bellare, and J. Black. OCB: A block-cipher mode of operation for efficient authenticated encryption. *ACM Transaction on Information Systems Security*, 6(3):365–403, 2003.
51. A. Rudra, P. K. Dubey, C. S. Julta, V. Kumar, J. R. Rao, and P. Rohatgi. Efficient Rijndael encryption implementation with composite field arithmetic. In *Proceedings of the CHES 2001*, LNCS, vol. 2162, pp. 171–184. Springer, 2001.
52. P. Sarkar. Improving upon the TET mode of operation. In K.-H. Nam and G. Rhee, editors, *ICISC*, LNCS, vol. 4817, pp. 180–192. Springer, 2007.
53. W. Trappe and L. C. Washington. *Introduction to Cryptography with Coding Theory*. Prentice Hall, First edition, 2002.
54. P. Wang, D. Feng, and W. Wu. HCTR: A variable-input-length Enciphering mode. In D. Feng, D. Lin, and M. Yung, editors, *CISC*, LNCS, vol. 3822, pp. 175–188. Springer, 2005.