1

Introduction to Fundamentals of Cryptography

1.1 Introduction: What is Cryptography?

Cryptography has been used for over 2000 years, but it is only since the mid-1970's that cryptography has come into common use for non-military applications. This is mainly due to factors such as the proliferation of computers, new communications media such as computer networks and the internet, and devices such as cell phones and smart cards. All of these new means of communication have created a need for various forms of security, which has been accompanied by an explosion of research in cryptography and its practical deployment.

In this chapter we review fundamental concepts, goals, and tools used in cryptography. Actually, the more general term *cryptology* is often used to encompass both *cryptography* and *cryptanalysis*. Cryptography refers to the design of schemes and protocols, and cryptanalysis refers to attempts to break them. The eminent cryptologist Gus Simmons has coined the phrase "the science of information integrity" to desribe cryptology.

Cryptography provides algorithms and protocols to achieve the following goals (among others):

confidentiality
: Confidentiality (or *secrecy*) means that data cannot be understood by an unauthorized party.

data integrity
: Data integrity means that data cannot be modified by an unauthorized party.

data origin authentication
: Data origin authentication is achieved when it can be verified that data was transmitted by a particular source.

entity authentication
: Entity authentication (or *identification*) refers to the verification of the identity of a person, computer or other device.

non-repudiation
Non-repudiation occurs when it is impossible for someone to deny having transmitted a message that, in fact, they did transmit.

access control
Access control refers to the restriction of electronic or physical access to authorized parties.

anonymity
Anonymity refers to the anonymous transmission of data, so that the origin cannot be determined.

Several of the above goals may be achieved simultaneously in certain situations if appropriate tools are used. However, it is impossible to achieve both data origin authentication and anonymity, because their specifications are mutually contradictory!

1.1.1 Cryptographic Tools

We now turn to the consideration of the cryptographic tools that are used to realize the objectives listed above. First, we briefly discuss keys. Cryptographic tools often incorporate keys, e.g., secret keys or public/private key pairs. A *secret key* is usually known to two or more parties, and, as its name suggests, it must be kept secret. A *public key* can be made public, and it enables anyone to perform a specified cryptographic operation. A corresponding *private key* is, in general, known only to a single party. Knowledge of the private key enables a certain "complementary" cryptographic operation to be carried out. A scheme that has a public key associated with it is called a *public key scheme*; it will also have a private key, as we have indicated. Sometimes secret key schemes are called *symmetric key schemes* and public key schemes are called *asymmetric key schemes*; this terminology emphasizes whether or not different keys are used for different operations.

A short summary of cryptographic tools and their usage of keys is provided in the following table. An "X" indicates that the given algorithm and key combination is feasible.

scheme	public/private keys?	secret key?	no key?
encryption scheme	X	X	
signature scheme	X		
message authentication code		X	
hash function			X
key agreement scheme	X	X	
identification scheme	X	X	

In more detail, here are the main tools and what they do:

encryption schemes

Encryption schemes are used to achieve confidentiality. Unencrypted data (*plaintext*) is encrypted so that it is untelligible to an unauthorized party. However, an authorized party, who possesses an appropriate key, is able to decrypt the encrypted data (*ciphertext*) and thus reconstruct the plaintext.

There are secret key encryption schemes as well as public key encryption schemes. Popular secret key encryption schemes include the *Data Encryption Standard* and the newer *Advanced Encryption Standard*; in these schemes, the same key is used to encrypt and decrypt. Examples of public key encryption schemes are *RSA*, *ElGamal* and *NTRU*. These use a public key to encrypt and a private key to decrypt.

signature schemes

Signature schemes are used to "sign" data. A signature helps to ensure data integrity and data origin authentication, and it can also provide non-repudiation.

A signature scheme is a public key scheme, in which the private key is used to create a signature on a message and the public key is used to verify the signature. Verification of a signature establishes that the signature was created by the presumed signer and that the data "originated" with the signer. It also means that the signer cannot deny having created the signature, because a valid signature cannot be created without access to the private key.

message authentication codes

A message authentication code provides data integrity.

A message authentication code is in some sense a secret key analog of a signature scheme. In a message authentication code, a secret key is used to create a "signature" on a message, and the same key is used to verify the validity of the signature. This secret key must be known to the signer and the verifier, who usually are two different people. Therefore, a message authentication code provides data integrity, but it does not permit public verification of the signature. Also, it usually does not provide data origin authentication or non-repudiation (becuase more than one person could have created the signature).

cryptographic hash functions

A hash function is used to provide random, unpredictable redundancy in data.

A cryptographic hash function has no key (actually, a message authentication code can be regarded as a keyed hash function, but we have treated these as a different type of crytpogrpahic tool). Therefore anyone can evaluate a publically known hash function, say h, on any input data, say x. The function h maps a "long" input x to a "short" output y in an unpredictable way. The *message digest* y incorporates redundancy when it is specified along with x. This redundancy is useful in constructing efficient signature schemes, for example.

key agreement protocols

A key agreement protocol is used to establish a common secret key known to two or more specified parties. Usually this key is to be subsequently used for another cryptographic purpose such as symmetric-key encryption or message authentication.

Key agreement schemes can be realized both in the secret and public key settings. In the secret key setting, two parties may already have a common long-term secret key, and they might want to derive a new, temporary *sesion key*. In the public key setting, a common key can be derived by two parties "from scratch".

identification schemes

An identification scheme is used to provide entity authentication.

Identification schemes are used in both the secret key and public key setting. In a secret key scenario, two parties have some common secret information, which makes it relatively easy to set up secure methods to authenticate identities. The problem of designing secure identification schemes is more difficult in the public key setting (indeed, it is a common situation that public key cryptography imposes greated challenges than secret key cryptography).

pseudorandom number generators

Pseudorandom number generators expand a small, truly random, seed into a long string of bits that cannot be distinguished from random bits. Pseudorandom number generators are used in many cryptographic contexts, for example, in the generation of keys.

1.1.2 A Practical Protocol: Secure Socket Layer

Practical protocols used in real-life applications may incorporate many cryptographic tools. We use SSL (secure socket layer) as an illustration. An SSL session is used, for example, to facilitiate online purchases from a company's web page. Suppose a *client* (Alice) wants to purchase something from a *server* (Bob, Inc.). The main steps in the resulting SSL session are summarized in Figure 1.1.

In more detail, here is what takes place in an SSL session: First, Alice and Bob, Inc. introduce themselves. This is called a "hello", and no cryptographic tools are used in this step.

Next, Bob Inc. authenticates himself to Alice; he sends her a copy of his public key, PK, signed by a trusted certification authority (CA). Alice verifies the CA's signature on PK using the CA's public verification key (which would have been bundled with the web browser software running on Alice's computer).

Now Alice and Bob, Inc. proceed to determine two common secret keys. Alice generates a random master secret, MS, using an appropriate pseudorandom number generator. She encrypts MS using Bob, Inc.'s public key and sends the resulting ciphertext to Bob., Inc. Bob Inc. decrypts the ciphertext, obtaining MS.

FIGURE 1.1
Setting up an SSL Session

client		server
	$\xrightarrow{\text{I'm Alice}}$	
	$\xleftarrow{\text{I'm Bob, Inc.}}$	
	$\xleftarrow{PK, \mathrm{sig}_{CA}(PK)}$	
verify PK		
generate MS	$\xrightarrow{y = e_{PK}(MS)}$	
		$MS = d_{PK}(y)$
$K_1, K_2 = h(MS)$		$K_1, K_2 = h(MS)$

Now Alice and Bob Inc. independently generate the same two keys K_1 and K_2 from MS (this step probably involves the use of a hash function, denoted h).

Finally, now that Alice and Bob, Inc. have both derived the same two secret keys, they use these keys to authenticate and encrypt the messages they send to each other. The key K_1 would be used to authenticate data using a message authentication code, and the key K_2 would be used to encrypt and decrypt data using a secret key cryptosystem.

1.2 Cryptosystems

We begin with a formal definition of a cryptosystem.

Definition 1.1: A *cryptosystem* is a five-tuple $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$, where the following conditions are satisfied:

1. $\mathcal{P}$ is a finite set of possible *plaintexts*
2. $\mathcal{C}$ is a finite set of possible *ciphertexts*
3. $\mathcal{K}$, the *keyspace*, is a finite set of possible *keys*
4. For each $K \in \mathcal{K}$, there is an *encryption rule* $e_K \in \mathcal{E}$ and a corresponding *decryption rule* $d_K \in \mathcal{D}$. Each $e_K : \mathcal{P} \to \mathcal{C}$ and $d_K : \mathcal{C} \to \mathcal{P}$ are functions such that $d_K(e_K(x)) = x$ for every plaintext element $x \in \mathcal{P}$.

The main property is property 4. It says that if a plaintext x is encrypted using e_K, and the resulting ciphertext is subsequently decrypted using d_K, then the original plaintext x results.

As mentioned earlier, a secret-key cryptosystem is one in which the key K, must be kept secret to everyone except Alice and Bob. A *block cipher* is a secret-key cryptosystem in which $\mathcal{P} = \mathcal{C} = (\mathbb{Z}_2)^n$ and $\mathcal{K} = (\mathbb{Z}_2)^m$, for specified integers n and m. We may regard plaintexts, ciphertexts and keys as boolean vectors or bitstrings of the prescribed lengths n or m.

The requirement of a *public-key cryptosystem* is that it should be computationally infeasible to determine d_K, given e_K. Then the encryption rule e_K can function as a public key.

The main difference between a secret- and public-key cryptosystem is illustrated in the following diagram:

Alice		Bob
	secret-key cryptosystem	
K		K
$y = e_K(x)$	$\xrightarrow{\quad y \quad}$	$x = d_K(y)$
	public-key cryptosystem	
e_k		K
$y = e_K(x)$	$\xrightarrow{\quad y \quad}$	$x = d_K(y)$

In the secret-key cryptosystem, the secret key K is known to Alice and Bob. Either party can encrypt or decrypt messages, using e_K or d_K, respectively. In the public-key cryptosystem, the key K is known only to Bob. Alice (or anyone else) can encrypt messages using the public encryption function, e_K, but only Bob knows the decryption function d_K derived from the key, K.

1.2.1 Iterated Ciphers and SPNs

A commonly used design for block ciphers is that of an *iterated cipher*. Here is a description of a typical iterated cipher: The cipher requires the specification of a *round function* and a *key schedule*, and the encryption of a plaintext will proceed through Nr similar *rounds*.

Let K be a random binary key of some specified length. K is used to construct Nr *round keys* (also called *subkeys*), which are denoted $K^1, \ldots, K^{\mathrm{Nr}}$. The list of round keys, $(K^1, \ldots, K^{\mathrm{Nr}})$, is the key schedule. The key schedule is constructed from K using a fixed, public algorithm.

The round function, say g, takes two inputs: a round key (K^r) and a current *state* (which we denote w^{r-1}). The next state is defined as $w^r = g(w^{r-1}, K^r)$. The initial state, w^0, is defined to be the plaintext, x. The ciphertext, y, is defined to be the state after all Nr rounds have been performed. Therefore, the encryption operation is carried out as follows:

$$\begin{aligned}
w^0 &\leftarrow x \\
w^1 &\leftarrow g(w^0, K^1) \\
w^2 &\leftarrow g(w^1, K^2) \\
\vdots \; &\; \vdots \; \vdots \\
w^{\mathrm{Nr}-1} &\leftarrow g(w^{\mathrm{Nr}-2}, K^{\mathrm{Nr}-1}) \\
w^{\mathrm{Nr}} &\leftarrow g(w^{\mathrm{Nr}-1}, K^{\mathrm{Nr}}) \\
y &\leftarrow w^{\mathrm{Nr}}.
\end{aligned}$$

In order for decryption to be possible, the function g must have the property that it is injective (i.e., one-to-one) if its second argument is fixed. This is equivalent to saying that there exists a function g^{-1} with the property that

$$g^{-1}(g(w, y), y) = w$$

for all w and y. Then decryption can be accomplished as follows:

$$\begin{aligned}
w^{\mathrm{Nr}} &\leftarrow y \\
w^{\mathrm{Nr}-1} &\leftarrow g^{-1}(w^{\mathrm{Nr}}, K^{\mathrm{Nr}}) \\
\vdots \; &\; \vdots \; \vdots \\
w^1 &\leftarrow g^{-1}(w^2, K^2) \\
w^0 &\leftarrow g^{-1}(w^1, K^1) \\
x &\leftarrow w^0.
\end{aligned}$$

These concepts are illustrated by a *substitution-permutation network*, or *SPN*, which is a special type of iterated cipher. Suppose that ℓ and m are positive integers. A plaintext and ciphertext will both be binary vectors of length ℓm (i.e.,

ℓm is the *block length* of the cipher). An SPN is built from two components, which are denoted π_S and π_P.

$$\pi_S : \{0,1\}^\ell \to \{0,1\}^\ell$$

is a permutation, and

$$\pi_P : \{1,\ldots,\ell m\} \to \{1,\ldots,\ell m\}$$

is also a permutation. The permutation π_S is called an *S-box* (the letter "S" denotes "substitution"). It is used to replace ℓ bits with a different set of ℓ bits. π_P, on the other hand, is used to permute ℓm bits.

Given an ℓm-bit binary string, say $x = (x_1, \ldots, x_{\ell m})$, we can regard x as the concatenation of m ℓ-bit substrings, which we denote $x_{(1)}, \ldots, x_{(m)}$. Thus

$$x = x_{(1)} \,||\, \cdots \,||\, x_{(m)}$$

and for $1 \leq i \leq m$, we have that

$$x_{(i)} = \left(x_{(i-1)\ell+1}, \ldots, x_{i\ell}\right).$$

The SPN will consist of Nr rounds. In each round (except for the last round, which is slightly different), we will perform m substitutions using π_S, followed by a permutation using π_P. Prior to each substitution operation, we will incorporate round key bits via a simple exclusive-or operation. We now present an SPN, based on π_S and π_P, as Cryptosystem 1.1.

Cryptosystem 1.1: *Substitution-Permutation Network*

Let ℓ, m and Nr be positive integers, let $\pi_S : \{0,1\}^\ell \to \{0,1\}^\ell$ be a permutation, and let $\pi_P : \{1,\ldots,\ell m\} \to \{1,\ldots,\ell m\}$ be a permutation. Let $\mathcal{P} = \mathcal{C} = \{0,1\}^{\ell m}$, and let $\mathcal{K} \subseteq (\{0,1\}^{\ell m})^{\mathrm{Nr}+1}$ consist of all possible key schedules that could be derived from an initial key K using the key scheduling algorithm. For a key schedule $(K^1, \ldots, K^{\mathrm{Nr}+1})$, we encrypt the plaintext x using Algorithm 1.1.

Algorithm 1.1: $\mathrm{SPN}(x, \pi_S, \pi_P, (K^1, \ldots, K^{\mathrm{Nr}+1}))$

$w^0 \leftarrow x$
for $r \leftarrow 1$ **to** $\mathrm{Nr} - 1$
do $\begin{cases} u^r \leftarrow w^{r-1} \oplus K^r \\ \textbf{for } i \leftarrow 1 \textbf{ to } m \\ \quad \textbf{do } v^r_{(i)} \leftarrow \pi_S(u^r_{(i)}) \\ w^r \leftarrow (v^r_{\pi_P(1)}, \ldots, v^r_{\pi_P(\ell m)}) \end{cases}$
$u^{\mathrm{Nr}} \leftarrow w^{\mathrm{Nr}-1} \oplus K^{\mathrm{Nr}}$
for $i \leftarrow 1$ **to** m
  do $v^{\mathrm{Nr}}_{(i)} \leftarrow \pi_S(u^{\mathrm{Nr}}_{(i)})$
$y \leftarrow v^{\mathrm{Nr}} \oplus K^{\mathrm{Nr}+1}$
output (y)

In Algorithm 1.1, u^r is the input to the S-boxes in round r, and v^r is the output of the S-boxes in round r. w^r is obtained from v^r by applying the permutation π_P, and then u^{r+1} is constructed from v^r by x-or-ing with the round key K^{r+1} (this is called *round key mixing*). In the last round, the permutation π_P is not applied. The very first and last operations performed are x-ors with subkeys. This is called *whitening*, and is regarded as a useful way to prevent an attacker from even beginning to carry out an encryption or decryption operation if the key is not known.

Because the S-boxes are bijective, all operations in the SPN can be reversed to decrpyt the ciphertext.

We illustrate the above general description with a particular SPN; see Figure 1.2. This SPN has block length 16, and each round key consists of 16 bits. Every S-box in the SPN maps four bits to four bits.

1.2.2 The Advanced Encryption Standard

The *Advanced Encryption Standard*, or *AES*, was adopted as a standard on November 26, 2001, and it was published as FIPS 197 in the Federal Register on December 4, 2001. It was required that the *AES* have a block length of 128 bits, and support key lengths of 128, 192 and 256 bits. *AES* is an iterated cipher; the number of rounds, which we denote by Nr, depends on the key length. $\mathrm{Nr} = 10$ if the key length is 128 bits; $\mathrm{Nr} = 12$ if the key length is 192 bits; and $\mathrm{Nr} = 14$ if the key length is 256 bits.

We first give a high-level description of *AES*. The algorithm proceeds as follows:

1. Given a plaintext x, initialize State to be x and perform an operation ADDROUNDKEY, which x-ors the RoundKey with State.
2. For each of the first $\mathrm{Nr} - 1$ rounds, perform a substitution operation called

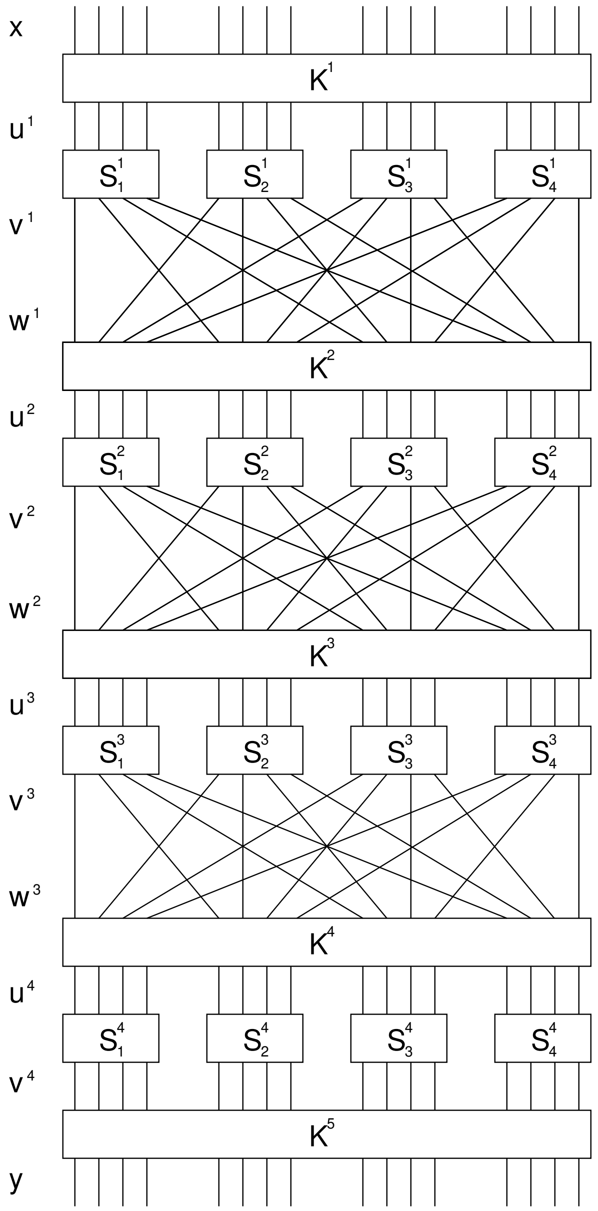

FIGURE 1.2
A substitution-permutation network

SUBBYTES on State using an S-box; perform a permutation SHIFTROWS on State; perform an operation MIXCOLUMNS on State; and perform ADDROUNDKEY.

3. Perform SUBBYTES; perform SHIFTROWS; and perform ADDROUNDKEY.
4. Define the ciphertext y to be State.

From this high-level description, we can see that the structure of the *AES* is very similar in many respects to the SPN discussed earlier. In every round of both these cryptosystems, we have subkey mixing, a substitution step and a permutation step. Both ciphers also include whitening. *AES* is "larger," and it also includes an additional linear transformation (MIXCOLUMNS) in each round.

We now describe in a bit more detail the operations used in the *AES*. All operations in *AES* are byte-oriented operations, and all variables used are considered to be formed from an appropriate number of bytes. The plaintext x consists of 16 bytes, denoted $x_0, \ldots, x_{15}$. State is represented as a four by four array of bytes, as follows:

$s_{0,0}$	$s_{0,1}$	$s_{0,2}$	$s_{0,3}$
$s_{1,0}$	$s_{1,1}$	$s_{1,2}$	$s_{1,3}$
$s_{2,0}$	$s_{2,1}$	$s_{2,2}$	$s_{2,3}$
$s_{3,0}$	$s_{3,1}$	$s_{3,2}$	$s_{3,3}$

Initially, State is defined to consist of the 16 bytes of the plaintext x, as follows:

$$\begin{array}{|c|c|c|c|}\hline s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ \hline s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ \hline s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ \hline s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \\ \hline \end{array} \leftarrow \begin{array}{|c|c|c|c|}\hline x_0 & x_4 & x_8 & x_{12} \\ \hline x_1 & x_5 & x_9 & x_{13} \\ \hline x_2 & x_6 & x_{10} & x_{14} \\ \hline x_3 & x_7 & x_{11} & x_{15} \\ \hline \end{array}$$

The operation SUBBYTES performs a substitution on each byte of State independently, using an S-box, say π_S, which is a permutation of $\{0,1\}^8$. In contrast to the S-boxes in *DES*, which are apparently "random" substitutions, the *AES* S-box can be defined algebraically. The algebraic formulation of the *AES* S-box involves operations in a finite field. We include the following description for the benefit of readers who are familiar with finite fields (other readers may want to skip this description).

The permutation π_S incorporates operations in the finite field

$$\mathbb{F}_{2^8} = \mathbb{Z}_2[x]/(x^8 + x^4 + x^3 + x + 1).$$

Let FIELDINV denote the multiplicative inverse of a field element; let BINARYTOFIELD convert a byte to a field element; and let FIELDTOBINARY perform the inverse conversion. This conversion is done in the obvious way: the field element

$$\sum_{i=0}^{7} a_i x^i$$

corresponds to the byte

$$a_7a_6a_5a_4a_3a_2a_1a_0,$$

where $a_i \in \mathbb{Z}_2$ for $0 \le i \le 7$. Then the permutation π_S is defined according to Algorithm 1.2. In this algorithm, the eight input bits $a_7a_6a_5a_4a_3a_2a_1a_0$ are replaced by the eight output bits $b_7b_6b_5b_4b_3b_2b_1b_0$.

Algorithm 1.2: SUBBYTES($a_7a_6a_5a_4a_3a_2a_1a_0$)

external FIELDINV, BINARYTOFIELD, FIELDTOBINARY
$z \leftarrow$ BINARYTOFIELD$(a_7a_6a_5a_4a_3a_2a_1a_0)$
if $z \neq 0$
  then $z \leftarrow$ FIELDINV(z)
$(a_7a_6a_5a_4a_3a_2a_1a_0) \leftarrow$ FIELDTOBINARY(z)
$(c_7c_6c_5c_4c_3c_2c_1c_0) \leftarrow (01100011)$
comment: In the following loop, all subscripts are to be reduced modulo 8
for $i \leftarrow 0$ **to** 7
  do $b_i \leftarrow (a_i + a_{i+4} + a_{i+5} + a_{i+6} + a_{i+7} + c_i) \bmod 2$
return $(b_7b_6b_5b_4b_3b_2b_1b_0)$

The operation SHIFTROWS acts on State as shown in the following diagram:

$$\begin{array}{|c|c|c|c|}\hline s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ \hline s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ \hline s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ \hline s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \\ \hline \end{array} \leftarrow \begin{array}{|c|c|c|c|}\hline s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ \hline s_{1,1} & s_{1,2} & s_{1,3} & s_{1,0} \\ \hline s_{2,2} & s_{2,3} & s_{2,0} & s_{2,1} \\ \hline s_{3,3} & s_{3,0} & s_{3,1} & s_{3,2} \\ \hline \end{array}$$

The operation MIXCOLUMNS is carried out on each of the four columns of State; it is presented as Algorithm 1.3. Each column of State is replaced by a new column which is formed by multiplying that column by a certain matrix of elements of the field $\mathbb{F}_{2^8}$. (Here "multiplication" means multiplication in the field $\mathbb{F}_{2^8}$. We assume that the external procedure FIELDMULT takes as input two field elements, and computes their product in the field.) Field addition is just componentwise addition modulo 2 (i.e., the x-or of the corresponding bitstrings). This operation is denoted by "$\oplus$" in Algorithm 1.3.

Algorithm 1.3: MIXCOLUMN(c)

external FIELDMULT, BINARYTOFIELD, FIELDTOBINARY
for $i \leftarrow 0$ **to** 3
  do $t_i \leftarrow$ BINARYTOFIELD$(s_{i,c})$
$u_0 \leftarrow$ FIELDMULT$(x, t_0) \oplus$ FIELDMULT$(x+1, t_1) \oplus t_2 \oplus t_3$
$u_1 \leftarrow$ FIELDMULT$(x, t_1) \oplus$ FIELDMULT$(x+1, t_2) \oplus t_3 \oplus t_0$
$u_2 \leftarrow$ FIELDMULT$(x, t_2) \oplus$ FIELDMULT$(x+1, t_3) \oplus t_0 \oplus t_1$
$u_3 \leftarrow$ FIELDMULT$(x, t_3) \oplus$ FIELDMULT$(x+1, t_0) \oplus t_1 \oplus t_2$
for $i \leftarrow 0$ **to** 3
  do $s_{i,c} \leftarrow$ FIELDTOBINARY(u_i)

The key schedule for the 10-round version of *AES* uses which uses a 128-bit key. We need 11 round keys, each of which consists of 16 bytes. The key scheduling algorithm is word-oriented (a *word* consists of 4 bytes, or, equivalently, 32 bits). Therefore each round key is comprised of four words. The concatenation of the round keys is called the *expanded key*, which consists of 44 words. It is denoted $w[0], \ldots, w[43]$, where each $w[i]$ is a word. The expanded key is constructed using an operation called KEYEXPANSION.

We have now described the operations used for encryption in the *AES*. In order to decrypt, it is necessary to perform all operations in the reverse order, and use the key schedule in reverse order. Further the operations SHIFTROWS, SUBBYTES and MIXCOLUMNS must be replaced by their inverse operations (the operation ADDROUNDKEY is its own inverse).

1.2.3 Modes of Operation

A *mode of operation* specifies how a block cipher is to be used to encrypt a long string of plaintext. There are several modes of operation; two of the most commonly used are ECB (*electronic code book*) and CBC (*cipher block chaining*) modes.

ECB mode corresponds to the naive use of a block cipher: given a sequence $x_1x_2\cdots$ of plaintext blocks (each consisting of 128 bits, in the case of the *AES*), each x_i is encrypted with the same key K, producing a string of ciphertext blocks, $y_1y_2\cdots$.

One obvious weakness of ECB mode is that the encryption of identical plaintext blocks yields identical ciphertext blocks. This is a serious weakness if the underlying message blocks are chosen from a "low entropy" plaintext space. For example, if a plaintext block always consists of 64 0's or 64 1's, then ECB encryption is essentially useless.

In CBC mode, each ciphertext block y_i is x-ored with the next plaintext block, x_{i+1}, before being encrypted with the key K. More formally, we start with an *initialization vector*, denoted by IV, and define $y_0 = \mathrm{IV}$. Then we construct $y_1, y_2, \ldots$, using the rule

$$y_i = e_K(y_{i-1} \oplus x_i),$$

$i \geq 1$. Encryption and decryption using CBC mode are depicted in Figure 1.3.

In ECB mode, changing one plaintext block, x_i, causes the corresponding ciphertext block, y_i, to be altered, but other ciphertext blocks are not affected. In some situations this might be a desirable property, for example, if the communication channel is not very reliable.

On the other hand, if a plaintext block x_i is changed in CBC mode, then y_i and all subsequent ciphertext blocks will be affected. This property means that CBC mode are useful for purposes of authentication (see Section 1.4).

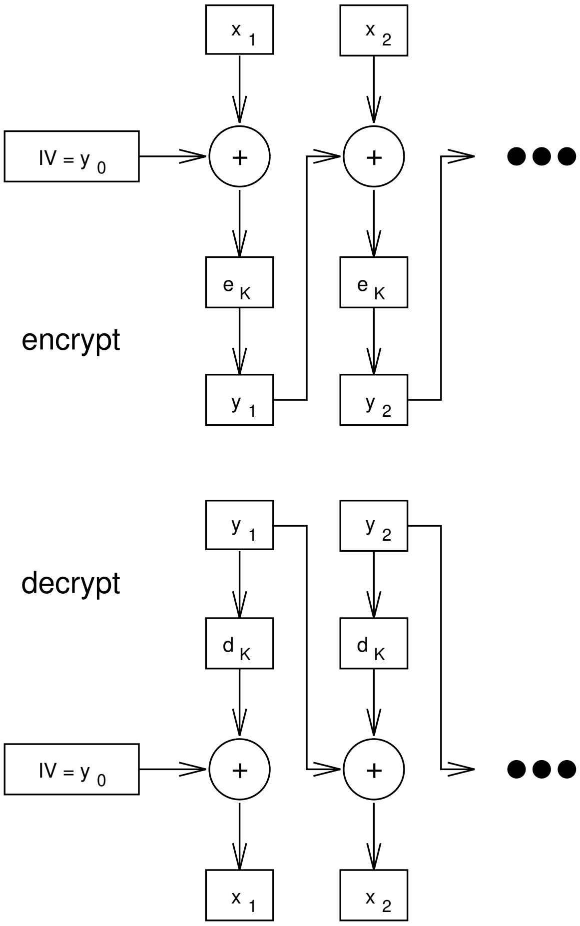

FIGURE 1.3
CBC mode

1.2.4 The RSA and Rabin Public-key Cryptosystems

The first, and most popular, public key cryptosystem is the *RSA Cryptosystem*. This cryptosystem uses computations in $\mathbb{Z}_n$, where n is the product of two distinct odd primes p and q. For such an integer n, denote $\phi(n) = (p-1)(q-1)$. The formal description of the cryptosystem is as follows.

Cryptosystem 1.2: *RSA Cryptosystem*

Let $n = pq$, where p and q are primes. Let $\mathcal{P} = \mathcal{C} = \mathbb{Z}_n$, and define

$$\mathcal{K} = \{(n, p, q, a, b) : ab \equiv 1 \pmod{\phi(n)}\}.$$

For $K = (n, p, q, a, b)$, define

$$e_K(x) = x^b \bmod n$$

and

$$d_K(y) = y^a \bmod n$$

$(x, y \in \mathbb{Z}_n)$. The values n and b comprise the public key, and the values p, q and a form the private key.

Using some elementary properties of modular arithmetic, it is not difficult to verify that encryption and decryption are inverse operations.

Here is a small (insecure) example of the *RSA Cryptosystem*.

Example 1.1 Suppose Bob chooses $p = 101$ and $q = 113$. Then $n = 11413$ and $\phi(n) = 100 \times 112 = 11200$. Suppose Bob chooses $b = 3533$. Then

$$b^{-1} \bmod 11200 = 6597.$$

Hence, Bob's secret decryption exponent is $a = 6597$.

Bob publishes $n = 11413$ and $b = 3533$ in a directory. Now, suppose Alice wants to encrypt the plaintext 9726 to send to Bob. She will compute

$$9726^{3533} \bmod 11413 = 5761$$

and send the ciphertext 5761 over the channel. When Bob receives the ciphertext 5761, he uses his secret decryption exponent to compute

$$5761^{6597} \bmod 11413 = 9726.$$

$\square$

Let us mention a few important algorithmic issues that are omitted from this brief description. These issues arise in any consideration of setting up the cryptosystem (i.e., *parameter generation*), as well as implementing the encryption and decryption operations efficiently. We do not provide details of the algorithms, but the interested reader can consult any introductory textbook on cryptography for more information.

Parameter generation requires finding two large primes, p and q. Current secure implementations of *RSA* will use primes p and q each of size on the order of 2^{512} (in other words, p and q each are 512 bits in length when represented in binary). p and q will be obtained by generating large random numbers and them applying a fast primality test. This is the hard part of parameter generation; the rest is easy: First n is formed as the product $p \times q$ and then $\phi(M)$ is computed to be $\phi(n) = (p-1)(q-1)$. The encryption exponent b is chosen, and, finally, a is computed as the inverse of b modulo $\phi(n)$ (this can be done efficiently using the EXTENDED EUCLIDEAN ALGORITHM).

Encryption and decryption operations are nothing more than modular exponentiations. However, the numbers involved are very large, so a naive computation of $x^b \bmod n$ by multiplying x by iteslf n times would be prohibitively slow. Fortunately, there is a much faster method, known as SQUARE-AND-MULTIPLY that reduces the number of multiplications to no more than $2\log_2 b$.

Next, we describe the *Rabin Cryptosystem*, which is also based on computations in ${\mathbb{Z}_n}^*$, where n is the product of two primes.

Cryptosystem 1.3: *Rabin Cryptosystem*

Let $n = pq$, where p and q are primes. Let $\mathcal{P} = \mathcal{C} = {\mathbb{Z}_n}^*$, and define

$$\mathcal{K} = \{(n, p, q)\}.$$

For $K = (n, p, q)$, define

$$e_K(x) = x^2 \bmod n$$

and

$$d_K(y) = \sqrt{y} \bmod n.$$

The value n is the public key, while p and q are the private key.

There are some subtleties associated with decryption in the *Rabin Cryptosystem*. Suppose that y is a valid ciphertext; this means that $y = x^2 \bmod n$ for some $x \in {\mathbb{Z}_n}^*$. Using the fact that n is the product of two odd primes, it is possible to show that there are exactly four possible plaintexts $z \in {\mathbb{Z}_n}^*$ such that $e_K(z) = y$. Two of these are x and $-x \bmod n$; as well, there are two other possible plaintexts, which are also negatives of each other modulo n.

It is possible for someone who knows the factorization of n to efficiently compute these four possible plaintexts corresponding to a ciphertext y, but these is no way to determine which of the four possible plaintexts is the “real” plaintext unless some redundancy is incorporated into the plaintexts for this specific purpose.

Example 1.2 Let's illustrate the encryption and decryption procedures for the *Rabin Cryptosystem* with a toy example. Suppose $n = 77 = 7 \times 11$. Then the encryption function is

$$e_K(x) = x^2 \bmod 77$$

and the decryption function is

$$d_K(y) = \sqrt{y} \bmod 77.$$

Suppose Bob wants to decrypt the ciphertext $y = 23$. The four square roots of 23 modulo 77 are $\pm 10, \pm 32 \bmod 77$. Therefore, the four possible plaintexts are $x = 10, 32, 45$ and 67. (It can be verified that each of these plaintexts yields the value 23 when squared and reduced modulo 77.) □

1.2.5 Comparison of Public- and Secret-key Cryptosystems

We list, and discuss briefly, a few of the main differences between public- and secret-key cryptosystems. These are observations relating to current practice, rather than requirements, however.

1. Secure secret-key cryptosystems are much faster than secure public-key cryptosystems (roughly 100 to 1000 times faster). Consequently, public-key cryptosystems are not generally used to encrypt "long" messages. They are more often employed to encrypt keys, which are then used in secret-key cryptosystems. SSL, which was described in Section 1.1.2, provides an example to illustrate this.
2. Public-key cryptosystems do not require that a secret key be known to two parties before the system is used, of course. Public keys can be listed in a directory or posted on a web page, for example. However, such methods of communicating public keys must also be accompanied by appropriate mechanisms to allow the public keys to be authenticated. This is done by using certificates, containing signed public keys, in the context of a secure public-key infrastructure. On the other hand, the authenticity of secret keys is, in general, guaranteed by the protocol used to generate or distribute them.
3. Public-key cryptosystems, in general, are based on simple-to-describe mathematical functions, and their security depends on certain computational problems that are infeasible, or believed to be infeasible. Secret-key cryptosystems tend to be based on perfroming a sequence of very simple operations (substitutions and permutations) in such a way that the system has a very complex algebraic description. Substitutions, in particular, are often random-looking functions. However, it should be noted that the S-boxes in the *AES* have a very simple descryption in terms of finite field arithmetic. The specific S-box used in the *AES* has desirable security properties against certain types of attacks (linear and differential cryptanalysis). On the other

hand, it remains to be seen if the algebraic description of *AES* is complex enough to provide long-term security.