Computer Science Department
Computer & Cyber Security Branch

Advanced Cryptography

2023-2024

Dr. Ayad Hazim

Groups, Rings, and Fields

Groups, rings, and fields are the fundamental elements of a branch of mathematics known as abstract algebra, or modern algebra. In abstract algebra, we are concerned with sets on whose elements we can operate algebraically; that is, we can combine two elements of the set, perhaps in several ways, to obtain a third element of the set. These operations are subject to specific rules, which define the nature of the set. By convention, the notation for the two principal classes of operations on set elements is usually the same as the notation for addition and multiplication on ordinary numbers. However, it is important to note that, in abstract algebra, we are

not limited to ordinary arithmetical operations. All this should become clear as we proceed.

Groups

A group G, sometimes denoted by {G, ·} is a set of elements with a binary operation, denoted by ·, that associates to each ordered pair (a, b) of elements in G an element (a · b) in G, such that the following axioms are obeyed:

The operator · is generic and can refer to addition, multiplication, or some other mathematical operation.

(A1) Closure:	If a and b belong to G, then a · b is also in G.
(A2) Associative:	a · (b · c) = (a · b) · c for all a, b, c in G.
(A3) Identity element:	There is an element e in G such that a · e = e · a = a for all a in G.
(A4) Inverse element:	For each a in G there is an element a' in G such that a · a' = a' · a = e.

Let N_n denote a set of n distinct symbols that, for convenience, we represent as {1,2,...,n}. A permutation of n distinct symbols is a one-to-one mapping from N_n to N_n. Define S_n to be the set of all permutations of n distinct symbols. Each element of S_n is represented by a permutation of the integers in {1,2,...,n}. It is easy to demonstrate that S_n is a group:

A1: If $\pi, \rho \in S_n$, then the composite mapping $\pi \cdot \rho$ is formed by permuting the elements of ρ according to the permutation π. For example, {3,2,1} · {1,3,2} = {2,3,1}. Clearly, $\pi \cdot \rho \in S_n$.

A2: The composition of mappings is also easily seen to be associative.

A3: The identity mapping is the permutation that does not alter the order of

the n elements. For S_n, the identity element is {1,2,...,n}.

A4: For any $\pi \in S_n$, the mapping that undoes the permutation defined by π is the inverse element for π .There will always be such an inverse. For example {2,3,1} · {3,1,2} = {1,2,3}

If a group has a finite number of elements, it is referred to as a finite group, and the order of the group is equal to the number of elements in the group. Otherwise, the group is an infinite group.

A group is said to be abelian if it satisfies the following additional condition:

(A5) Commutative: $a \cdot b = b \cdot a$ for all a, b in G.

The set of integers (positive, negative, and 0) under addition is an abelian group. The set of nonzero real numbers under multiplication is an abelian group. The set S_n from the preceding example is a group but not an abelian group for $n > 2$.

When the group operation is addition, the identity element is 0; the inverse element of a is a; and subtraction is defined with the following rule: a b = a + (b).

Cyclic Group

We define exponentiation within a group as repeated application of the group operator, so that $a^3 = a \cdot a \cdot a$. Further, we define $a^0 = e$, the identity element; and $a^{-n} = (a')^n$. A group G is cyclic if every element of G is a power

a^k (k is an integer) of a fixed element a G. The element a is said to generate the group G, or to be a generator of G. A cyclic group is always abelian, and may be finite or infinite.

The additive group of integers is an infinite cyclic group generated by the element 1. In this case, powers are interpreted additively, so that n is the nth power of 1.

Rings

A ring R, sometimes denoted by {R, +, x}, is a set of elements with two binary operations, called addition and multiplication, such that for all a, b, c in R the following axioms are obeyed:

Generally, we do not use the multiplication symbol, x, but denote multiplication by the concatenation of two elements.

(A1-A5) R is an abelian group with respect to addition; that is, R satisfies axioms A1 through A5. For the case of an additive group, we denote the identity element as 0 and the inverse of a as a.

(M1) Closure under multiplication:	If a and b belong to R, then ab is also in R.
(M2) Associativity of multiplication:	a(bc) = (ab)c for all a, b, c in R.
(M3) Distributive laws:	a(b + c) = ab + ac for all a, b, c in R. (a + b)c = ac + bc for all a, b, c in R.

In essence, a ring is a set in which we can do addition, subtraction [a b = a + (-b)], and multiplication without leaving the set.

With respect to addition and multiplication, the set of all n-square matrices over the real numbers is a ring.

A ring is said to be commutative if it satisfies the following additional condition:

(M4) Commutativity of multiplication: ab = ba for all a, b in R.

Let S be the set of even integers (positive, negative, and 0) under the usual operations of addition and multiplication. S is a commutative ring. The set of all n-square matrices defined in the preceding example is not a commutative ring.

Next, we define an integral domain, which is a commutative ring that obeys the following axioms:

(M5)	Multiplicative identity:	There is an element 1 in R such that a1 = 1a = a for all a in R.
(M6)	No zero divisors:	If a, b in R and ab = 0, then either a = 0 or b = 0.

Let S be the set of integers, positive, negative, and 0, under the usual operations of addition and multiplication. S is an integral domain.

Fields

A field F, sometimes denoted by {F, +, x}, is a set of elements with two binary operations, called addition and multiplication, such that for all a, b, c in F the following axioms are obeyed:

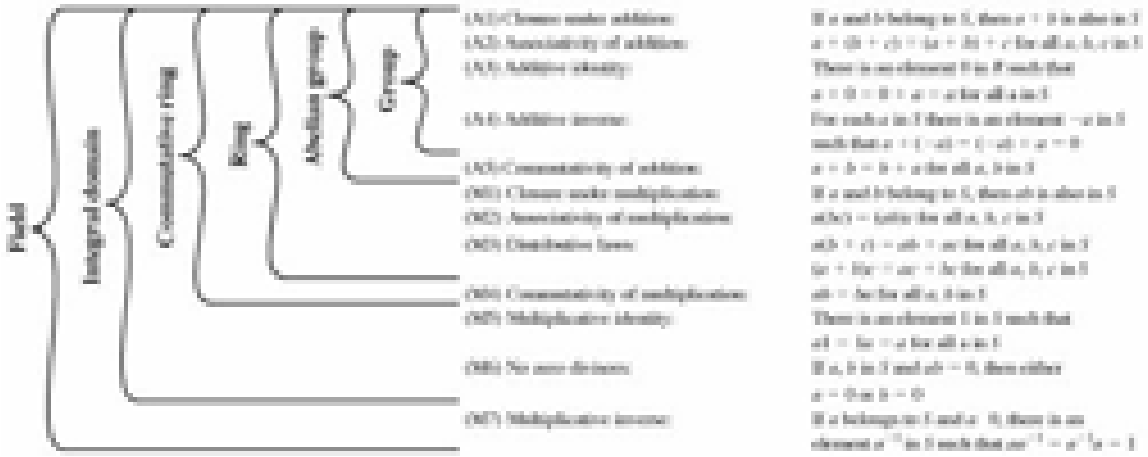
(A1M6)		F is an integral domain; that is, F satisfies axioms A1 through A5 and M1 through M6.
(M7)	Multiplicative inverse:	For each a in F, except 0, there is an element a^{-1} in F such that $aa^{-1} = (a^{-1})a = 1$.

In essence, a field is a set in which we can do addition, subtraction, multiplication, and division without leaving the set. Division is defined with the following rule: $a/b = a(b^{-1})$.

Familiar examples of fields are the rational numbers, the real numbers, and the complex numbers. Note that the set of all integers is not a field, because not every element of the set has a multiplicative inverse; in fact, only the elements 1 and -1 have multiplicative inverses in the integers.

The next figure summarizes the axioms that define groups, rings, and fields.

Figure Group, Ring, and Field



The Euclidean Algorithm

One of the basic techniques of number theory is the Euclidean algorithm, which is a simple procedure for determining the greatest common divisor of two positive integers.

Greatest Common Divisor

Recall that nonzero b is defined to be a divisor of a if a = mb for some m, where a, b, and m are integers. We will use the notation gcd(a, b) to mean the greatest common divisor of a and b. The positive integer c is said to be the greatest common divisor of a and b if

1. c is a divisor of a and of b;
2. any divisor of a and b is a divisor of c.

An equivalent definition is the following:

gcd(a, b) = max[k, such that k|a and k|b]

Because we require that the greatest common divisor be positive, gcd(a, b) = gcd(a, b) = gcd(a, b) = gcd(a, b). In general, gcd(a, b) = gcd(|a|, |b|).

gcd(60, 24) = gcd(60, 24) = 12

Also, because all nonzero integers divide 0, we have gcd(a, 0) = |a|.

We stated that two integers a and b are relatively prime if their only common positive integer factor is 1. This is equivalent to saying that a and b are relatively prime if gcd(a, b) = 1.

8 and 15 are relatively prime because the positive divisors of 8 are 1, 2, 4, and 8, and the positive divisors of 15 are 1, 3, 5, and 15, so 1 is the only integer on both lists.

Finding the Greatest Common Divisor

The Euclidean algorithm is based on the following theorem: For any nonnegative integer a and any positive integer b,

Equation 4-4

$$\gcd(a, b) = \gcd(b, a \bmod b)$$

gcd(55, 22) = gcd(22, 55 mod 22) = gcd(22, 11) = 11

To see that Equation (4.4) works, let d = gcd(a, b). Then, by the definition of gcd, d|a and d|b. For any positive integer b, a can be expressed in the form

a = kb + r≡≡ r (mod b)

a mod b = r

with k, r integers. Therefore, (a mod b) = a kb for some integer k. But because d|b, it also divides kb. We also have d|a. Therefore, d|(a mod b). This shows that d is a common divisor of b and (a mod b). Conversely, if d is a common divisor of b and (a mod b), then d|kb and thus d|[kb + (a mod b)], which is equivalent to d|a. Thus, the set of common divisors of a and b is equal to the set of common divisors of b and (a mod b). Therefore, the gcd of one pair is the same as the gcd of the other pair, proving the theorem.

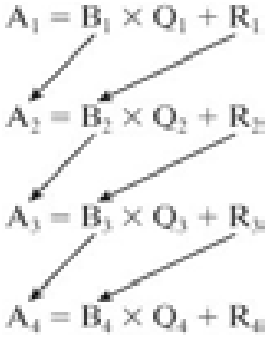
Equation (4.4) can be used repetitively to determine the greatest common divisor.

gcd(18, 12) = gcd(12, 6) = gcd(6, 0) = 6
gcd(11, 10) = gcd(10, 1) = gcd(1, 0) = 1

The Euclidean algorithm makes repeated use of Equation (4.4) to determine the greatest common divisor, as follows. The algorithm assumes a > b > 0. It is acceptable to restrict the algorithm to positive integers because gcd(a, b) = gcd(|a|, |b|).

EUCLID(a, b)
1. A ← a; B ← b
2. if B = 0 return A = gcd(a, b)
3. R = A mod B
4. A ← B
5. B ← R
6. goto 2

The algorithm has the following progression:



To find gcd(1970, 1066)		
1970	= 1 x 1066 + 904	gcd(1066, 904)
1066	= 1 x 904 + 162	gcd(904, 162)
904	= 5 x 162 + 94	gcd(162, 94)
162	= 1 x 94 + 68	gcd(94, 68)
94	= 1 x 68 + 26	gcd(68, 26)
68	= 2 x 26 + 16	gcd(26, 16)
26	= 1 x 16 + 10	gcd(16, 10)
16	= 1 x 10 + 6	gcd(10, 6)
10	= 1 x 6 + 4	gcd(6, 4)
6	= 1 x 4 + 2	gcd(4, 2)
4	= 2 x 2 + 0	gcd(2, 0)
Therefore, gcd(1970, 1066) = 2		

Finite Fields Of the Form GF(2^n)

Earlier in this chapter, we mentioned that the order of a finite field must be of the form p^n where p is a prime and n is a positive integer. we looked at the special case of finite fields with order p. We found that, using modular arithmetic in Z_p, all of the axioms for a field (Figure 4.1) are satisfied. For polynomials over p^n, with n > 1, operations modulo p^n do not produce a field. In this section, we show what structure satisfies the axioms for a field in a set with p^n elements, and concentrate on GF (2^n).

Motivation

Virtually all encryption algorithms, both symmetric and public key, involve arithmetic operations on integers. If one of the operations that is used in the algorithm is division, then we need to work in arithmetic defined over a field. For convenience and for implementation efficiency, we would also like to work with integers that fit exactly into a given number of bits, with no wasted bit patterns. That is, we wish to work with integers in the range 0 through 2^n 1, which fit into an n-bit word.

Suppose we wish to define a conventional encryption algorithm that operates on data 8 bits at a time and we wish to perform division. With 8 bits, we can represent integers in the range 0 through 255. However, 256 is not a prime number, so that if arithmetic is performed in Z_{256} (arithmetic modulo 256), this set of integers will not be a field. The closest prime number less than 256 is 251. Thus, the set Z_{251}, using arithmetic modulo 251, is a field. However, in this case the 8-bit patterns representing the integers 251 through 255 would not be used, resulting in inefficient use of storage.

As the preceding example points out, if all arithmetic operations are to be used, and we wish to represent a full range of integers in n bits, then arithmetic modulo will not work; equivalently, the set of integers modulo 2^n, for n > 1, is not a field. Furthermore, even if the encryption algorithm uses only addition and multiplication, but not division, the use of the set Z_{2^n} is questionable, as the following example illustrates.

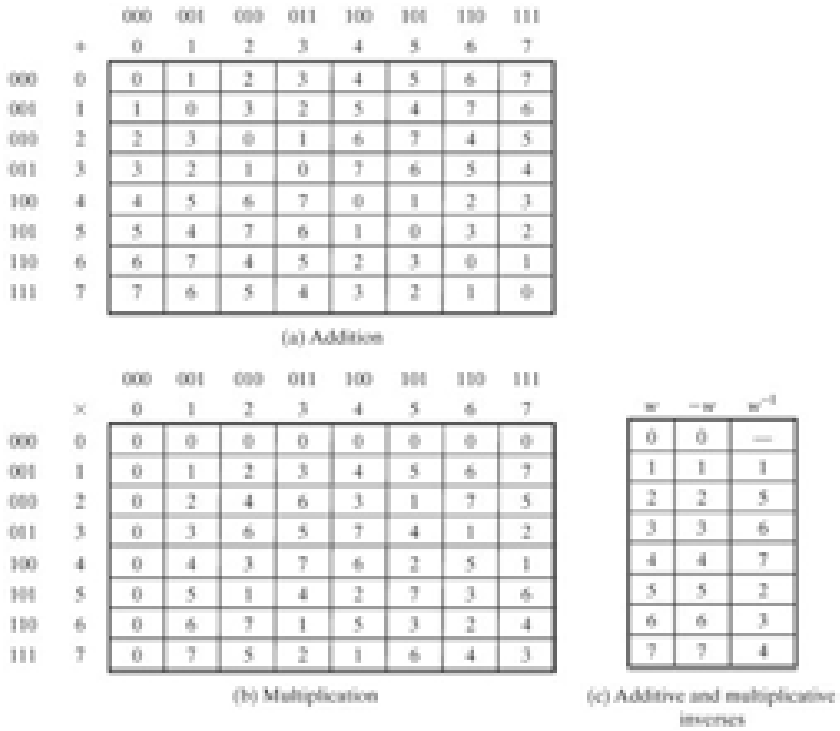
Suppose we wish to use 3-bit blocks in our encryption algorithm, and use only the operations of addition and multiplication. Then arithmetic modulo 8 is well defined, as shown in Table 4.1. However, note that in the multiplication table, the nonzero integers do not appear an equal number of

times. For example, there are only four occurrences of 3, but twelve occurrences of 4. On the other hand, as was mentioned, there are finite fields of the form GF (2^n) so there is in particular a finite field of order $2^3 = 8$. Arithmetic for this field is shown in Table 4.5. In this case, the number of occurrences of the nonzero integers is uniform for multiplication. To summarize,

Integer	1	2	3	4	5	6	7
Occurrences in Z_8	4	8	4	12	4	8	4
Occurrences in GF(2^3)	7	7	7	7	7	7	7

Table 4.5. Arithmetic in GF(2^3)

(This item is displayed on page 121 in the print version)



(a) Addition

×	000 (0)	001 (1)	010 (2)	011 (3)	100 (4)	101 (5)	110 (6)	111 (7)
000 (0)	0	0	0	0	0	0	0	0
001 (1)	0	1	2	3	4	5	6	7
010 (2)	0	2	4	6	3	1	7	5
011 (3)	0	3	6	5	7	4	1	2
100 (4)	0	4	3	7	6	2	5	1
101 (5)	0	5	1	4	2	7	3	6
110 (6)	0	6	7	1	5	3	2	4
111 (7)	0	7	5	2	1	6	4	3

(b) Multiplication

w	−w	w^{-1}
0	0	—
1	1	1
2	2	5
3	3	6
4	4	7
5	5	2
6	6	3
7	7	4

(c) Additive and multiplicative inverses

For the moment, let us set aside the question of how the matrices of Table 4.5 were constructed and instead make some observations.

1. The addition and multiplication tables are symmetric about the main diagonal, in conformance to the commutative property of addition and multiplication. This property is also exhibited in Table 4.1, which uses mod 8 arithmetic.
2. All the nonzero elements defined by Table 4.5 have a multiplicative inverse, unlike the case with Table 4.1.
3. The scheme defined by Table 4.5 satisfies all the requirements for a finite field. Thus, we can refer to this scheme as GF(2^3).

For convenience, we show the 3-bit assignment used for each of the elements of GF (2^3).

Intuitively, it would seem that an algorithm that maps the integers unevenly onto themselves might be cryptographically weaker than one that provides a uniform mapping. Thus, the finite fields of the form GF(2^n) are attractive for cryptographic algorithms.

To summarize, we are looking for a set consisting of 2^n elements, together with a definition of addition and multiplication over the set that define a field. We can assign a unique integer in the range 0 through 2^n 1 to each element of the set. Keep in mind that we will not use modular arithmetic, as we have seen that this does not result in a field. Instead, we will show how polynomial arithmetic provides a means for constructing the desired field.

Modular Polynomial Arithmetic

Consider the set S of all polynomials of degree n 1 or less over the field Z_p. Thus, each polynomial has the form

$$f(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1x + a_0 = \sum_{i=0}^{n-1} a_i x^i$$

where each a_i takes on a value in the set {0, 1,..., p 1}. There are a total of p^n different polynomials in S.

For p = 3 and n = 2, the $3^2 = 9$ polynomials in the set are		
0	x	2x
1	x + 1	2x + 1
2	x + 2	2x + 2
For p = 2 and n = 3, the $2^3 = 8$ the polynomials in the set are		
0	x + 1	$x^2 + x$
1	x^2	$x^2 + x + 1$
x	$x^2 + 1$	

With the appropriate definition of arithmetic operations, each such set S is a finite field. The definition consists of the following elements:

1. Arithmetic follows the ordinary rules of polynomial arithmetic using the basic rules of algebra, with the following two refinements.

   Arithmetic on the coefficients is performed modulo p. That is, we use the rules of arithmetic for the finite field Z_p.

2. If multiplication results in a polynomial of degree greater than n 1, then the polynomial is reduced modulo some irreducible polynomial m(x) of degree n. That is, we divide by m(x) and keep the remainder. For a polynomial f(x), the remainder is expressed as r(x) = f(x) mod m(x).

The Advanced Encryption Standard (AES) uses arithmetic in the finite field GF (2^8), with the irreducible polynomial $m(x) = x^8 + x^4 x^3 + x + 1$. Consider the two polynomials $f(x) = x^6 + x^4 + x^2 + x + 1$ and $g(x) = x^7 + x + 1$. Then

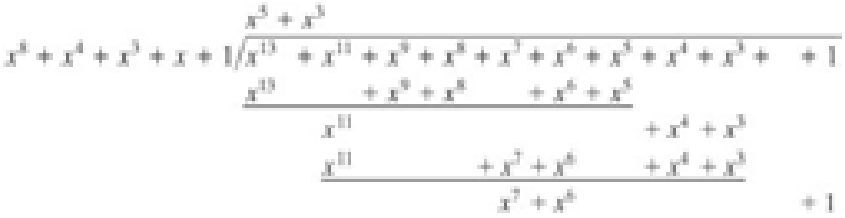
$f(x) + g(x) = x^6 + x^4 x^2 + x + 1 + x^7 + x + 1$

$f(x) \times g(x) = x^{13} + x^{11} + x^9 + x^8 + x^7 +$

$x^7 + x^5 + x^3 + x^2 + x +$

$x^6 + x^4 + x^2 + x + 1$

$= x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$



Therefore, f(x) x g(x) mod m(x) = $x^7 + x^6 + 1$

As with ordinary modular arithmetic, we have the notion of a set of residues in modular polynomial arithmetic. The set of residues modulo m(x), an nth-degree polynomial, consists of p^n elements. Each of these elements is represented by one of the p^n polynomials of degree m < n.

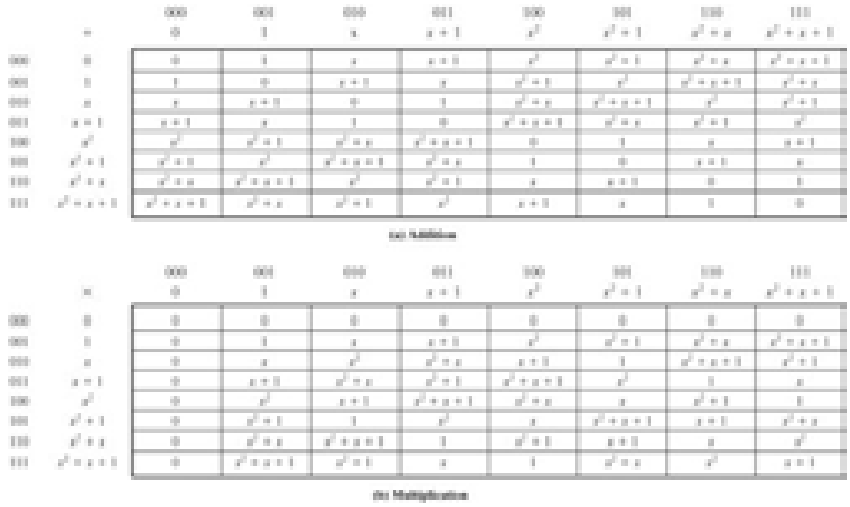
The residue class [x + 1], modulo m(x), consists of all polynomials a(x) such that a(x) (x + 1) (mod m(x)). Equivalently, the residue class [x + 1] consists of all polynomials a(x) that satisfy the equality a(x) mod m(x) = x + 1.

It can be shown that the set of all polynomials modulo an irreducible nth-degree polynomial m(x) satisfies the axioms in Figure 4.1, and thus forms a finite field. Furthermore, all finite fields of a given order are isomorphic; that is, any two finite-field structures of a given order have the same structure, but the representation, or labels, of the elements may be different.

To construct the finite field GF (2^3), we need to choose an irreducible polynomial of degree 3. There are only two such polynomials: ($x^3 + x^2 + 1$) and ($x^3 + x + 1$). Using the latter, Table 4.6 shows the addition and multiplication tables for GF (2^3). Note that this set of tables has the identical structure to those of Table 4.5. Thus, we have succeeded in finding a way to define a field of order 2^3.

Table 4.6. Polynomial Arithmetic Modulo ($x^3 + x + 1$)

(This item is displayed on page 124 in the print version)



Finding the Multiplicative Inverse

Just as the Euclidean algorithm can be adapted to find the greatest common divisor of two polynomials, the extended Euclidean algorithm can be adapted to find the multiplicative inverse of a polynomial. Specifically, the algorithm will find the multiplicative inverse of b(x) modulo m(x) if the degree of b(x) is less than the degree of m(x) and gcd[m(x), b(x)] = 1. If m(x) is an irreducible polynomial, then it has no factor other than itself or 1, so that gcd[m(x), b(x)] = 1. The algorithm is as follows:

EXTENDED EUCLID [m(x), b(x)]
1. [A1(x), A2(x), A3(x)] ←[1, 0, m(x)]; [B1(x), B2(x), B3(x)] ← [0, 1, b(x)]
2. if B3(x) = 0 return A3(x) = gcd[m(x), b(x)]; no Inverse
3. if B3(x) = 1 return B3(x) = gcd[m(x), b(x)]; B2(x) = $b(x)^{1}$ mod m(x)
4. Q(x) = quotient of A3(x)/B3(x)
5. [T1(x), T2(x), T3(x)] ← [A1(x) Q(x)B1(x), A2(x) Q(x)B2(x), A3(x) QB3(x)]
6. [A1(x), A2(x), A3(x)] ← [B1(x), B2(x), B3(x)]

7. [B1(x), B2(x), B3(x)] ← [T1(x), T2(x), T3(x)]
8. goto 2

Table 4.7 shows the calculation of the multiplicative inverse of $(x^7 + x + 1)$ mod $(x^8 + x^4 + x^3 + x + 1)$. The result is that $(x^7 + x + 1)^1 = (x^7)$. That is, $(x^7 + x + 1)(x^7) \equiv 1 \pmod{(x^8 + x^4 + x^3 + x + 1)}$.

Table 4.7. Extended Euclid $[(x^8 + x^4 + x^3 + x + 1), (x^7 + x + 1)]$

(This item is displayed on page 125 in the print version)

Initialization	A1(x) = 1; A2(x) = 0; A3(x) = $x^8 + x^4 + x^3 + x + 1$ B1(x) = 0; B2(x) = 1; B3(x) = $x^7 + x + 1$
Iteration 1	Q(x) = x A1(x) = 0; A2(x) = 1; A3(x) = $x^7 + x + 1$ B1(x) = 1; B2(x) = x; B3(x) = $x^4 + x^3 + x^2 + 1$
Iteration 2	Q(x) = $x^3 + x^2 + 1$ A1(x) = 1; A2(x) = x; A3(x) = $x^4 + x^3 + x^2 + 1$ B1(x) = $x^3 + x^2 + 1$; B2(x) = $x^4 + x^3 + x + 1$; B3(x) = x
Iteration 3	Q(x) = $x^3 + x^2 + x$ A1(x) = $x^3 + x^2 + 1$; A2(x) = $x^4 + x^3 + x + 1$; A3(x) = x B1(x) = $x^6 + x^2 + x + 1$; B2(x) = x^7; B3(x) = 1
Iteration 4	B3(x) = gcd[$(x^7 + x + 1)$, $(x^8 + x^4 + x^3 + x + 1)$] = 1 B2(x) = $(x^7 + x + 1)^1 \bmod (x^8 + x^4 + x^3 + x + 1) = x^7$

Computational Considerations

A polynomial f(x) in $GF(2^n)$

$$f(x) = a_{n-1}x^{n-1} + a_{n-2}x^{n-2} + \cdots + a_1 x + a_0 = \sum_{i=0}^{n-1} a_i x^i$$

can be uniquely represented by its n binary coefficients $(a_{n1}a_{n2}...a_0)$. Thus, every polynomial in $GF(2^n)$ can be represented by an n-bit number.

Tables 4.5 and 4.6 show the addition and multiplication tables for $GF(2^3)$ modulo $m(x) = (x^3 + x + 1)$. Table 4.5 uses the binary representation, and Table 4.6 uses the polynomial representation.

Addition

We have seen that addition of polynomials is performed by adding corresponding coefficients and, in the case of polynomials over Z_2 addition is just the XOR operation. So, addition of two polynomials in $GF(2^n)$ corresponds to a bitwise XOR operation.

Consider the two polynomials in $GF(2^8)$ from our earlier example: $f(x) = x^6 + x^4 + x^2 + x + 1$ and $g(x) = x^7 + x + 1$.

$(x^6 + x^4 + x^2 + x + 1) + (x^7 + x + 1)$	$= x^7 + x^6 + x^6 + x^4 + x^2$	(polynomial notation)
$(01010111) \oplus (10000011)$	$= (11010100)$	(binary notation)
$\{57\} \oplus \{83\}$	$= \{D4\}$	(hexadecimal notation)[7]

[7] A basic refresher on number systems (decimal, binary, hexadecimal) can be found at the Computer Science Student Resource Site at WilliamStallings.com/StudentSupport.html. Here each of two groups of 4 bits in a byte is denoted by a single hexadecimal character, the two characters enclosed in brackets.

Multiplication

There is no simple XOR operation that will accomplish multiplication in $GF(2^n)$ However, a reasonably straightforward, easily implemented technique is available. We will discuss the technique with reference to $GF(2^8)$ using $m(x) = x^8 + x^4 + x^3 + x + 1$, which is the finite field used in AES. The technique readily generalizes to GF (2^n).

The technique is based on the observation that

Equation 4-8

$$x^8 \bmod m(x) = [m(x) - x^8] = (x^4 + x^3 + x + 1)$$

A moment's thought should convince you that Equation (4.8) is true; if not, divide it out. In general, in GF (2^n) with an nth-degree polynomial p(x), we have $x^n \bmod p(x) = [p(x)\ x^n]$.

Now, consider a polynomial in GF (2^8), which has the form $f(x) = b_7x^7 + b_6x^6 + b_5x^5 + b_4x^4 + b_3x^3 + b_2x^2 + b_1x + b_0$. If we multiply by x, we have

Equation 4-9

$$x \times f(x) = (b_7x^8 + b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 + b_1x^2 + b_0x) \bmod m(x)$$

If $b_7 = 0$, then the result is a polynomial of degree less than 8, which is already in reduced form, and no further computation is necessary. If $b_7 = 1$, then reduction modulo m(x) is achieved using Equation (4.8):

$x \times f(x) = (b_6x^7 + b_5x^6 + b_4x^5 + b_3x^4 + b_2x^3 +$

$b_1x^2 + b_0x) + (x^4 + x^3 + x + 1)$

It follows that multiplication by x (i.e., 00000010) can be implemented as a 1-bit left shift followed by a conditional bitwise XOR with (00011011), which represents $(x^4 + x^3 + x + 1)$. To summarize,

Equation 4-10

$$x \times f(x) = \begin{cases} (b_6b_5b_4b_3b_2b_1b_00) & \text{if } b_7 = 0 \\ (b_6b_5b_4b_3b_2b_1b_00) \oplus (00011011) & \text{if } b_7 = 1 \end{cases}$$

Multiplication by a higher power of x can be achieved by repeated application of Equation (4.10). By adding intermediate results, multiplication by any constant in GF(2^8) can be achieved.

In an earlier example, we showed that for $f(x) = x^6 + x^4 + x^2 + x + 1$, $g(x) = x^7 + x + 1$, and $m(x) = x^8 + x^4 + x^3 + x + 1$, $f(x) \times g(x) \bmod m(x) = x^7 + x^6 + 1$. Redoing this in binary arithmetic, we need to compute (01010111) x (10000011). First, we determine the results of multiplication by powers of x:

(01010111) x (00000001) = (10101110)

(01010111) x (00000100) = (01011100) ⊕ (00011011) = (01000111)

(01010111) x (00001000) = (10001110)

(01010111) x (00010000) = (00011100) ⊕ (00011011) = (00000111)

(01010111) x (00100000) = (00001110)

(01010111) x (01000000) = (00011100)

(01010111) x (10000000) = (00111000)

So,

(01010111) x (10000011) = (01010111) x [(00000001) x (00000010) x (10000000)]

= (01010111) ⊕ (10101110) ⊕ (00111000) = (11000001)

which is equivalent to $x^7 + x^6 + 1$.

Using a Generator

An equivalent technique for defining a finite field of the form GF(2^n) using the same irreducible polynomial, is sometimes more convenient. To begin, we need two definitions: A generator g of a finite field F of order q (contains q elements) is an element whose first q 1 powers generate all the nonzero elements of F. That is, the elements of F consist of 0, g^0, g^1,..., g^{q2}. Consider a field F defined by a polynomial f(x). An element b contained in F is called a root of the polynomial if f(b) = 0. Finally, it can be shown that a root g of

an irreducible polynomial is a generator of the finite field defined on that polynomial.

Let us consider the finite field GF (2^3), defined over the irreducible polynomial $x^3 + x + 1$, discussed previously. Thus, the generator g must satisfy $f(x) = g^3 + g + 1 = 0$. Keep in mind, as discussed previously, that we need not find a numerical solution to this equality. Rather, we deal with polynomial arithmetic in which arithmetic on the coefficients is performed modulo 2. Therefore, the solution to the preceding equality is $g^3 = g\ 1 = g +$
1. We now show that g in fact generates all of the polynomials of degree less than 3. We have the following:

$$g^4 = g(g^3) = g(g + 1) = g^2 + g$$

$$g^5 = g(g^4) = g(g^2 + g) = g^3 + g^2 = g^2 + g + 1$$

$$g^6 = g(g^5) = g(g^2 + g + 1) = g^3 + g^2 + g = g^2 + g + g + 1 = g^2 + 1$$

$$g^7 = g(g^6) = g(g^2 + 1) = g^3 + g = g + g + 1 = 1 = g^0$$

We see that the powers of g generate all the nonzero polynomials in GF(2^3). Also, it should be clear that $g^k = g^{k \bmod 7}$ for any integer k. Table 4.8 shows the power representation, as well as the polynomial and binary representations.

Table 4.8. Generator for GF(2^3) using $x^3 + x + 1$

Power Representation	Polynomial Representation	Binary Representation	Decimal (Hex) Representation
0	0	000	0
g^0 (= g^7)	1	001	1
g^1	g	010	2
g^2	g^2	100	4
g^3	g + 1	011	3
g^4	g^2 + g	110	6
g^5	g^2 + g + 1	111	7
g^6	g^2 + 1	101	5

This power representation makes multiplication easy. To multiply in the power notation, add exponents modulo 7. For example, $g^4 \times g^6 = g^{(10 \bmod 7)} = g^3 = g + 1$. The same result is achieved using polynomial arithmetic, as follows: we have $g^4 = g^2 + g$ and $g^6 = g^2 + 1$. Then, $(g^2 + g) \times (g^2 + 1) = g^4 + g^3 + g^2 + 1$. Next, we need to determine $(g^4 + g^3 + g^2 + 1) \bmod (g^3 + g + 1)$ by division:

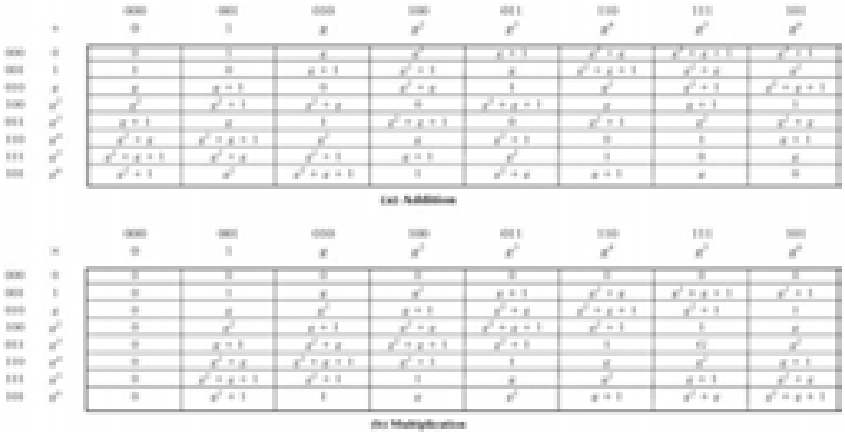
$$
\begin{array}{r}
g + 1 \\
g^3 + g + 1 \overline{)\, g^4 + g^3 + g^2 + g} \\
\underline{g^4 \quad + g^2 + g} \\
g^3 \\
\underline{g^3 \quad + g + 1} \\
g + 1
\end{array}
$$

We get a result of $g + 1$, which agrees with the result obtained using the power representation.

Table 4.9 shows the addition and multiplication tables for GF(2^3) using the power represenation. Note that this yields the identical results to the polynomial representation (Table 4.6) with some of the rows and columns interchanged.

Table 4.9. GF(2^3) Arithmetic Using Generator for the Polynomial ($x^3 + x + 1$)

(This item is displayed on page 128 in the print version)



In general, for GF(2^n) with irreducible polynomial f(x), determine g^n = f(x) g^n. Then calculate all of the powers of g from g^{n+1} through g^{2n2}. The elements of the field correspond to the powers of g from through g^{2n2}, plus the value 0. For multiplication of two elements in the field, use the equality $g^k = g^{k \bmod (2n1)}$ for any integer k.

Finite Fields of the Form GF (p)

we defined a field as a set that obeys all of the axioms of Figure 4.1 and gave some examples of infinite fields. Infinite fields are not of particular interest in the context of cryptography. However, finite fields play a crucial role in many cryptographic algorithms. It can be shown that the order of a finite field (number of elements in the field) must be a power of a prime p^n, where n is a positive integer. a prime number is an integer whose only positive integer factors are itself and 1. That is, the only positive integers that are divisors of p are p and 1.

The finite field of order p^n is generally written GF(p^n); stands for Galois field, in honor of the mathematician who first studied finite fields. Two special cases are of interest for our purposes. For n = 1, we have the finite field GF(p); this finite field has a different structure than that for finite fields with n > 1 and is studied in this section. .

Finite Fields of Order p

For a given prime, p, the finite field of order p, GF(p) is defined as the set Z_p of integers {0, 1,..., p 1}, together with the arithmetic operations modulo p.

that the set Z_n of integers {0,1,...,n 1}, together with the arithmetic operations modulo n, is a commutative ring (Table 4.2). We further observed that any integer in Z_n has a multiplicative inverse if and only if that integer is relatively prime to n

If n is prime, then all of the nonzero integers in Z_n are relatively prime to n, and therefore there exists a multiplicative inverse for all of the nonzero integers in Z_n. Thus, we can add the following properties to those listed in Table 4.2 for Z_p:

[4] As stated in the discussion of Equation (4.3), two integers are relatively prime if their only common positive integer factor is 1.

Multiplicative inverse (w^{-1})	For each $w \in Z_p$, $w \neq 0$, there exists a $z \in Z_p$ such that $w \times z \equiv 1 \pmod{p}$

Because w is relatively prime to p, if we multiply all the elements of Z_p by w, the resulting residues are all of the elements of Z_p permuted. Thus, exactly one of the residues has the value 1. Therefore, there is some integer Z_p in that, when multiplied by w, yields the residue 1. That integer is the multiplicative inverse of w, designated w^{-1}. Therefore, Z_p is in fact a finite field. Further, Equation (4.3) is consistent with the existence of a multiplicative inverse and can be rewritten without the condition:

Equation 4-5

$$\textbf{if } (a \times b) \equiv (a \times c) \pmod{p} \textbf{ then } b \equiv c \pmod{p}$$

Multiplying both sides of Equation (4.5) by the multiplicative inverse of a, we have:

$((a^{-1}) \times a \times b) \equiv ((a^{-1}) \times a \times c) \pmod{p}$

$b \equiv c \pmod{p}$

The simplest finite field is GF(2). Its arithmetic operations are easily summarized:

Addition

+	0	1
0	0	1
1	1	0

Multiplication

×	0	1
0	0	0
1	0	1

Inverses

w	$-w$	w^{-1}
0	0	—
1	1	1

In this case, addition is equivalent to the exclusive-OR (XOR) operation, and multiplication is equivalent to the logical AND operation.

Table 4.3 shows GF (7). This is a field of order 7 using modular arithmetic modulo 7. As can be seen, it satisfies all of the properties required of a field (Figure 4.1). Compare this table with Table 4.1. In the latter case, we see that the set Z_8 using modular arithmetic modulo 8, is not a field. Later in this chapter, we show how to define addition and multiplication operations on Z_8 in such a way as to form a finite field.

Table 4.3. Arithmetic in GF (7)

(This item is displayed on page 111 in the print version)

+	0	1	2	3	4	5	6
0	0	1	2	3	4	5	6
1	1	2	3	4	5	6	0
2	2	3	4	5	6	0	1
3	3	4	5	6	0	1	2
4	4	5	6	0	1	2	3
5	5	6	0	1	2	3	4
6	6	0	1	2	3	4	5

(a) Addition modulo 7

×	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6
2	0	2	4	6	1	3	5
3	0	3	6	2	5	1	4
4	0	4	1	5	2	6	3
5	0	5	3	1	6	4	2
6	0	6	5	4	3	2	1

(b) Multiplication modulo 7

w	$-w$	w^{-1}
0	0	—
1	6	1
2	5	4
3	4	5
4	3	2
5	2	3
6	1	6

(c) Additive and multiplicative inverses modulo 7

Finding the Multiplicative Inverse in GF (p)

It is easy to find the multiplicative inverse of an element in GF(p) for small values of p. You simply construct a multiplication table, such as shown in

Table 4.3b, and the desired result can be read directly. However, for large values of p, this approach is not practical.

If gcd(m, b) = 1, then b has a multiplicative inverse modulo m. That is, for positive integer b < m, there exists a b^1 < m such that bb^1 = 1 mod m. The Euclidean algorithm can be extended so that, in addition to finding gcd(m, b), if the gcd is 1, the algorithm returns the multiplicative inverse of b.

EXTENDED EUCLID (m, b)
1. (A1, A2, A3) ← (1, 0, m); (B1, B2, B3) ← (0, 1, b)
2. if B3 = 0 return A3 = gcd(m, b); no inverse
3. if B3 = 1 return B3 = gcd(m, b); B2 = b^1 mod m

4. $Q = \left\lfloor \frac{A3}{B3} \right\rfloor$

5. (T1, T2, T3) ← (A1 QB1, A2 QB2, A3 QB3)
6. (A1, A2, A3) ← (B1, B2, B3)
7. (B1, B2, B3) ← (T1, T2, T3)
8. goto 2

Throughout the computation, the following relationships hold:

$$mT1 + bT2 = T3 \quad mA1 + bA2 = A3 \quad mB1 + bB2 = B3$$

To see that this algorithm correctly returns gcd(m, b), note that if we equate A and B in the Euclidean algorithm with A3 and B3 in the extended Euclidean algorithm, then the treatment of the two variables is identical. At each iteration of the Euclidean algorithm, A is set equal to the previous value of B and B is set equal to the previous value of A mod B. Similarly, at each step of the extended Euclidean algorithm, A3 is set equal to the previous value of B3, and B3 is set equal to the previous value of A3 minus the integer quotient of A3 multiplied by B3. This latter value is simply the remainder of A3 divided by B3, which is A3 mod B3.

Note also that if gcd(m, b) = 1, then on the final step we would have B3 = 0 and A3 = 1. Therefore, on the preceding step, B3 = 1. But if B3 = 1, then we can say the following:

$mB1 + bB2 = B3$

$mB1 + bB2 = 1$

$bB2 = 1\ mB1$

$bB2 \equiv 1 \pmod{m}$

And B2 is the multiplicative inverse of b, modulo m.

Table 4.4 is an example of the execution of the algorithm. It shows that gcd(1759, 550) = 1 and that the multiplicative inverse of 550 is 355; that is, 550 x 335 ≡ 1 (mod 1759).

Table 4.4. Finding the Multiplicative Inverse of 550 in GF(1759)

Q	A1	A2	A3	B1	B2	B3
	1	0	1759	0	1	550
3	0	1	550	1	3	109
5	1	3	109	5	16	5
21	5	16	5	106	339	4
1	106	339	4	111	355	1

Modular Arithmetic

Given any positive integer n and any nonnegative integer a, if we divide a by n, we get an integer quotient q and an integer remainder r that obey the following relationship:

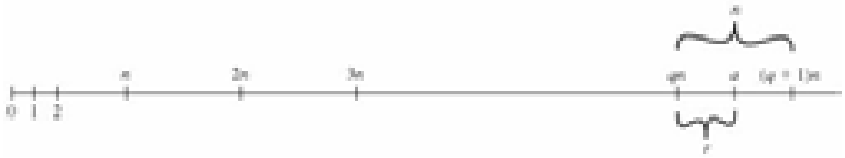
Equation 4-1

$$a = qn + r \qquad 0 \leq r < n; q = \lfloor a/n \rfloor$$

where ⌊x⌋is the largest integer less than or equal to x.

Figure 4.2 demonstrates that, given a and positive n, it is always possible to find q and r that satisfy the preceding relationship. Represent the integers on the number line; a will fall somewhere on that line (positive a is shown, a similar demonstration can be made for negative a). Starting at 0, proceed to n, 2n, up to qn such that qn ≤a and (q + 1)n > a. The distance from qn to a is r, and we have found the unique values of q and r. The remainder r is often referred to as a residue.

Figure 4.2. The Relationship a = qn + r, 0 ≤ r < n



a = 11;	n = 7;	11 = 1 x 7 + 4;	r = 4	q = 1
a = -11;	n = 7;	-11 = (-2) x 7 + 3;	r = 3	q = -2

If a is an integer and n is a positive integer, we define a mod n to be the remainder when a is divided by n. The integer n is called the modulus. Thus, for any integer a, we can always write:

$a = \lfloor a/n \rfloor x\ n + (a \bmod n)$

11 mod 7 = 4;	-11 mod 7 = 3

Two integers a and b are said to be congruent modulo n, if (a mod n) = (b mod n). This is written as $a \equiv b \pmod{n}$.

We have just used the operator mod in two different ways: first as a binary operator that produces a remainder, as in the expression a mod b; second as a congruence relation that shows the equivalence of two integers, as in the expression To distinguish the two uses, the mod term is enclosed in parentheses for a congruence relation; this is common but not universal in the literature. See Appendix D for a further discussion.

$73 \equiv 4 \pmod{23}$;	$21 \equiv -9 \pmod{10}$

Divisors

We say that a nonzero b divides a if a = mb for some m, where a, b, and m are integers. That is, b divides a if there is no remainder on division. The notation is commonly used to mean b divides a. Also, if b|a, we say that b is a divisor of a.

The positive divisors of 24 are 1, 2, 3, 4, 6, 8, 12, and 24.

The following relations hold:

- If a|1, then a = ±1.
- If a|b and b|a, then a = ±b.
- Any b ≠ 0 divides 0.
- If b|g and b|h, then b|(mg + nh) for arbitrary integers m and n.

To see this last point, note that

If b|g, then g is of the form g = b x g_1 for some integers g_1.

If b|h, then h is of the form h = b x h_1 for some integers h_1.

So

mg + nh = mbg_1 + nbh_1 = b x (mg_1 + nh_1)

and therefore b divides mg + nh.

b = 7; g = 14; h = 63; m = 3; n = 2.

7|14 and 7|63. To show: 7|(3 x 14 + 2 x 63)

We have (3 x 14 + 2 x 63) = 7(3 x 2 + 2 x 9)

And it is obvious that 7|(7(3 x 2 + 2 x 9))

Note that if a ≡ 0 (mod n), then n|a.

Properties of Congruences

Congruences have the following properties:

1. a ≡ b (mod n) if n|(a b).
2. a ≡ b (mod n) implies b ≡ a (mod n)..
3. a ≡ b (mod n) and b ≡ c (mod n) imply a ≡ c (mod n).

To demonstrate the first point, if n|(a b), then (a b) = kn for some k. So we can write a = b + kn. Therefore, (a mod n) = (reminder when b + kn is divided by n) = (reminder when b is divided by n) = (b mod n)

23 ≡ 8 (mod 5)	because	23 8 = 15 = 5 3
11 ≡ 5 (mod 8)	because	11 5 = 16 = 8 x (2)
81 ≡ 0 (mod 27)	because	81 0 = 81 = 27 x 3

The remaining points are as easily proved.

Modular Arithmetic Operations

Note that, by definition (Figure 4.2), the (mod n) operator maps all integers into the set of integers {0, 1,... (n 1)}. This suggests the question: Can we perform arithmetic operations within the confines of this set? It turns out that we can; this technique is known as modular arithmetic.

Modular arithmetic exhibits the following properties:

1. [(a mod n) + (b mod n)] mod n = (a + b) mod n
2. [(a mod n) (b mod n)] mod n = (a b) mod n
3. [(a mod n) x (b mod n)] mod n = (a x b) mod n

We demonstrate the first property. Define (a mod n) = r_a and (b mod n) = r_b. Then we can write a = r_a + jn for some integer j and b = r_b + kn for some integer k. Then

(a + b) mod n = (r_a + jn + r_b +kn) mod n

= (r_a + r_b (k + j)n) mod n

= (r_a + r_b) mod n

= [(a mod n] + (b mod n)] mod n

The remaining properties are as easily proved. Here are examples of the three properties:

11 mod 8 = 3; 15 mod 8 = 7
[(11 mod 8) + (15 mod 8)] mod 8 = 10 mod 8 = 2 (11 + 15) mod 8 = 26 mod 8 = 2
[(11 mod 8) (15 mod 8)] mod 8 = 4 mod 8 = 4 (11 15) mod 8 = 4 mod 8 = 4
[(11 mod 8) x (15 mod 8)] mod 8 = 21 mod 8 = 5 (11 x 15) mod 8 = 165 mod 8 = 5

Exponentiation is performed by repeated multiplication, as in ordinary arithmetic.

To find 11^7 mod 13, we can proceed as follows:
$11^2 = 121 \equiv 4 \pmod{13}$
$11^4 = (11^2)^2 \equiv 4^2 \equiv 3 \pmod{13}$
$11^7 \equiv 11 \times 4 \times 3 \equiv 132 \equiv 2 \pmod{13}$

Thus, the rules for ordinary arithmetic involving addition, subtraction, and multiplication carry over into modular arithmetic.

Table 4.1 provides an illustration of modular addition and multiplication modulo 8. Looking at addition, the results are straightforward and there is a regular pattern to the matrix. Both matrices are symmetric about the main diagonal, in conformance to the commutative property of addition and multiplication. As in ordinary addition, there is an additive inverse, or negative, to each integer in modular arithmetic. In this case, the negative of an integer x is the integer y such that (x + y) mod 8 = 0. To find the additive inverse of an integer in the left-hand column, scan across the corresponding row of the matrix to find the value 0; the integer at the top of that column is the additive inverse; thus (2 + 6) mod 8 = 0. Similarly, the entries in the multiplication table are straightforward. In ordinary arithmetic, there is a multiplicative inverse, or reciprocal, to each integer. In modular arithmetic mod 8, the multiplicative inverse of x is the integer y such that (x x y) mod 8
= 1 mod 8. Now, to find the multiplicative inverse of an integer from the multiplication table, scan across the matrix in the row for that integer to find the value 1; the integer at the top of that column is the multiplicative inverse;

thus (3 x 3) mod 8 = 1. Note that not all integers mod 8 have a multiplicative inverse; more about that later.

Table 4.1. Arithmetic Modulo 8

+	0	1	2	3	4	5	6	7
1	1	2	3	4	5	6	7	0
2	2	3	4	5	6	7	0	1
3	3	4	5	6	7	0	1	2
4	4	5	6	7	0	1	2	3
5	5	6	7	0	1	2	3	4
6	6	7	0	1	2	3	4	5
7	7	0	1	2	3	4	5	6

(a) Addition modulo 8

×	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7
2	0	2	4	6	0	2	4	6
3	0	3	6	1	4	7	2	5
4	0	4	0	4	0	4	0	4
5	0	5	2	7	4	1	6	3
6	0	6	4	2	0	6	4	2
7	0	7	6	5	4	3	2	1

(b) Multiplication modulo 8

w	$-w$	w^{-1}
0	0	—
1	7	1
2	6	—
3	5	3
4	4	—
5	3	5
6	2	—
7	1	7

(c) Additive and multiplicative inverses modulo 8

Properties of Modular Arithmetic

Define the set Z_n as the set of nonnegative integers less than n:

$Z_n = \{0, 1,...,(n\ 1)\}$

This is referred to as the set of residues, or residue classes modulo n. To be more precise, each integer in Z_n represents a residue class. We can label the residue classes modulo n as [0], [1], [2],...,[n 1], where

$[r] = \{a: a \text{ is an integer}, a \equiv r \pmod{n}\}$

The residue classes modulo 4 are

The residue classes modulo 4 are	
	[0] = { ..., 16, 12, 8, 4, 0, 4, 8, 12, 16,... }
	[1] = { ..., 15, 11, 7, 3, 1, 5, 9, 13, 17,... }
	[2] = { ..., 14, 10, 6, 2, 2, 6, 10, 14, 18,... }
	[3] = { ..., 13, 9, 5, 1, 3, 7, 11, 15, 19,... }

Of all the integers in a residue class, the smallest nonnegative integer is the one usually used to represent the residue class. Finding the smallest nonnegative integer to which k is congruent modulo n is called reducing k modulo n.

If we perform modular arithmetic within Z_n, the properties shown in Table 4.2 hold for integers in Z_n. Thus, Z_n is a commutative ring with a multiplicative identity element (Figure 4.1).

Table 4.2. Properties of Modular Arithmetic for Integers in Z_n

Property	Expression
Commutative laws	$(w + x) \bmod n = (x + w) \bmod n$ $(w \times x) \bmod n = (x \times w) \bmod n$
Associative laws	$[(w + x) + y] \bmod n = [w + (x + y)] \bmod n$ $[(w \times x) \times y] \bmod n = [w \times (x \times y)] \bmod n$
Distributive laws	$[w + (x + y)] \bmod n = [(w \times x) + (w \times y)] \bmod n$ $[w + (x \times y)] \bmod n = [(w + x) \times (w + y)] \bmod n$
Identities	$(0 + w) \bmod n = w \bmod n$ $(1 + w) \bmod n = w \bmod n$
Additive inverse (-w)	For each $w \equiv Z_n$, there exists a z such that $w + z \quad 0 \bmod n$

There is one peculiarity of modular arithmetic that sets it apart from ordinary arithmetic. First, observe that, as in ordinary arithmetic, we can write the following:

Equation 4-2

$$\text{if } (a + b) \equiv (a + c) \pmod{n} \quad \text{then} \quad b \equiv c \pmod{n}$$

(5 + 23) ≡ (5 + 7)(mod 8}; 23 ≡ 7 (mod 8)

Equation (4.2) is consistent with the existence of an additive inverse. Adding the additive inverse of a to both sides of Equation (4.2), we have:

((a) + a + b) ≡ ((a) + a + c)(mod n)

b ≡ c (mod n)

However, the following statement is true only with the attached condition:

Equation 4-3

$$\text{if } (a \times b) \equiv (a \times c) \pmod{n} \text{ then } b \equiv c \pmod{n} \quad \text{if } a \text{ is relatively prime to } n$$

where the term relatively prime is defined as follows: two integers are relatively prime if their only common positive integer factor is 1. Similar to the case of Equation (4.2), we can say that Equation (4.3) is consistent with the existence of a multiplicative inverse. Applying the multiplicative inverse of a to both sides of Equation (4.2), we have:

((a^1)ab) ≡ ((a^1)ac)(mod n)

b ≡ c (mod n)

To see this, consider an example in which the condition of Equation (4.3) does not hold. The integers 6 and 8 are not relatively prime, since they have the common factor 2. We have the following:

6 x 3 = 18 ≡ 2 (mod 8)

6 x 7 = 42 ≡ 2 (mod 8)

Yet 3 ≠7 (mod 8).

The reason for this strange result is that for any general modulus n, a multiplier a that is applied in turn to the integers 0 through (n 1) will fail to produce a complete set of residues if a and n have any factors in common.

With a = 6 and n = 8,

Z_8	0	1	2	3	4	5	6	7
Multiply by 6	0	6	12	18	24	30	36	42
Residues	0	6	4	2	0	6	4	2

Because we do not have a complete set of residues when multiplying by 6, more than one integer in Z_8 maps into the same residue. Specifically, 6 x 0 mod 8 = 6 x 4 mod 8; 6 x 1 mod 8 = 6 x 5 mod 8; and so on. Because this is a many-to-one mapping, there is not a unique inverse to the multiply operation.

However, if we take a = 5 and n = 8, whose only common factor is 1,

Z_8	0	1	2	3	4	5	6	7
Multiply by 6	0	5	10	15	20	25	30	35
Residues	0	5	2	7	4	1	6	3

The line of residues contains all the integers in Z_8, in a different order.

In general, an integer has a multiplicative inverse in Z_n if that integer is relatively prime to n. Table 4.1c shows that the integers 1, 3, 5, and 7 have a multiplicative inverse in Z_8, but 2, 4, and 6 do not.

Polynomial Arithmetic

Before pursuing our discussion of finite fields, we need to introduce the interesting subject of polynomial arithmetic. We are concerned with polynomials in a single variable x, and we can distinguish three classes of polynomial arithmetic:

- Ordinary polynomial arithmetic, using the basic rules of algebra
- Polynomial arithmetic in which the arithmetic on the coefficients is performed modulo p; that is, the coefficients are in GF(p)
- Polynomial arithmetic in which the coefficients are in GF(p), and the polynomials are defined modulo a polynomial m(x) whose highest power is some integer n

This section examines the first two classes, and the next section covers the last class.

Ordinary Polynomial Arithmetic

A **polynomial** of degree n (integer $n \geq 0$) is an expression of the form

$$f(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 = \sum_{i=0}^{n} a_i x^i$$

where the a_i are elements of some designated set of numbers S, called the **coefficient set**, and $a_n \neq 0$. We say that such polynomials are defined over the coefficient set S.

A zeroth-degree polynomial is called a constant polynomial and is simply an element of the set of coefficients. An nth-degree polynomial is said to be a **monic polynomial** if $a_n = 1$.

In the context of abstract algebra, we are usually not interested in evaluating a polynomial for a particular value of x [e.g., f(7)]. To emphasize this point, the variable x is sometimes referred to as the indeterminate.

Polynomial arithmetic includes the operations of addition, subtraction, and multiplication. These operations are defined in a natural way as though the variable x was an element of S. Division is similarly defined, but requires that S be a field. Examples of fields include the real numbers, rational numbers, and Z_p for p prime. Note that the set of all integers is not a field and does not support polynomial division.

Addition and subtraction are performed by adding or subtracting corresponding coefficients. Thus, if

$$f(x) = \sum_{i=0}^{n} a_i x^i; \quad g(x) = \sum_{i=0}^{m} b_i x^i; \quad n \geq m$$

Then addition is defined as

$$f(x) + g(x) = \sum_{i=0}^{m} (a_i + b_i)x^i + \sum_{i=m+1}^{n} a_i x^i$$

and multiplication is defined as

$$f(x) \times g(x) = \sum_{i=0}^{n+m} c_i x^i$$

where

$c_k = a_0 b_{k1} + a_1 b_{k1} + ... + a_{k1} b_1 + a_k b_0$

In the last formula, we treat a_i as zero for i > n and b_i as zero for i > m. Note that the degree of the product is equal to the sum of the degrees of the two polynomials.

As an example, let $f(x) = x^3 + x^2 + 2$ and $g(x) = x^2$ x + 1, where S is the set of integers. Then

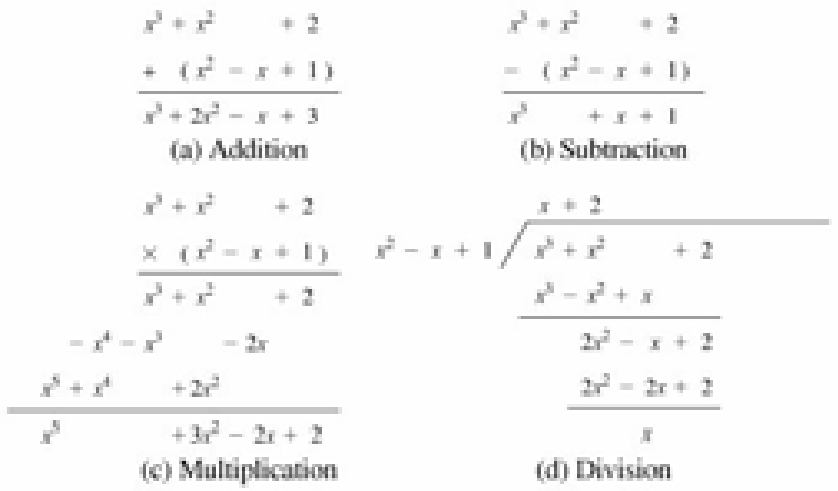
$f(x) + g(x) = x^3 + 2x^2$ x + 3

f(x) g(x) = x^3 + x + 1

f(x) x g(x) = $x^5 + 3x^2$ 2x + 2

Figures 4.3a through 4.3c show the manual calculations. We comment on division subsequently.

Figure 4.3. Examples of Polynomial Arithmetic



Polynomial Arithmetic with Coefficients in Zp

Let us now consider polynomials in which the coefficients are elements of some field F. We refer to this as a polynomial over the field F. In that case, it is easy to show that the set of such polynomials is a ring, referred to as a **polynomial ring**. That is, if we consider each distinct polynomial to be an element of the set, then that set is a ring.

In fact, the set of polynomials whose coefficients are elements of a commutative ring forms a polynomial ring, but that is of no interest in the present context.

When polynomial arithmetic is performed on polynomials over a field, then division is possible. Note that this does not mean that exact division is possible. Let us clarify this distinction. Within a field, given two elements a and b, the quotient a/b is also an element of the field. However, given a ring R that is not a field, in general division will result in both a quotient and a remainder; this is not exact division.

Consider the division 5/3 within a set S. If S is the set of rational numbers, which is a field, then the result is simply expressed as 5/3 and is an element of S. Now suppose that S is the field Z_7. In this case, we calculate (using Table 4.3c):

$5/3 = (5 \times 3^1) \bmod 7 = (5 \times 5) \bmod 7 = 4$

which is an exact solution. Finally, suppose that S is the set of integers, which is a ring but not a field. Then 5/3 produces a quotient of 1 and a remainder of 2:

$5/3 = 1 + 2/3$

$5 = 1 \times 3 + 2$

Thus, division is not exact over the set of integers.

Now, if we attempt to perform polynomial division over a coefficient set that is not a field, we find that division is not always defined.

If the coefficient set is the integers, then $(5x^2)/(3x)$ does not have a solution, because it would require a coefficient with a value of 5/3, which is not in the coefficient set. Suppose that we perform the same polynomial division over Z_7. Then we have $(5x^2)/(3x) = 4x$ which is a valid polynomial over Z_7.

However, as we demonstrate presently, even if the coefficient set is a field, polynomial division is not necessarily exact. In general, division will produce a quotient and a remainder:

Equation 4-6

$$\frac{f(x)}{g(x)} = q(x) + \frac{r(x)}{g(x)}$$
$$f(x) = q(x)g(x) + r(x)$$

If the degree of f(x) is n and the degree of g(x) is m, ($m \geq n$), then the degree of the quotient q(x) m n is and the degree of the remainder is at most m - 1.

With the understanding that remainders are allowed, we can say that polynomial division is possible if the coefficient set is a field.

In an analogy to integer arithmetic, we can write f(x) mod g(x) for the remainder r(x) in Equation (4.6). That is, r(x) = f(x) mod g(x). If there is no remainder [i.e., r(x) = 0], then we can say g(x) divides f(x), written as g(x)|f(x); equivalently, we can say that g(x) is a factor of f(x) or g(x) is a divisor of f(x).

For the preceding example and [$f(x) = x^3 + x^2 + 2$ and $g(x) = x^2 x + 1$], f(x)/g(x) produces a quotient of q(x) = x + 2 and a remainder r(x) = x as shown in Figure 4.3d. This is easily verified by noting that

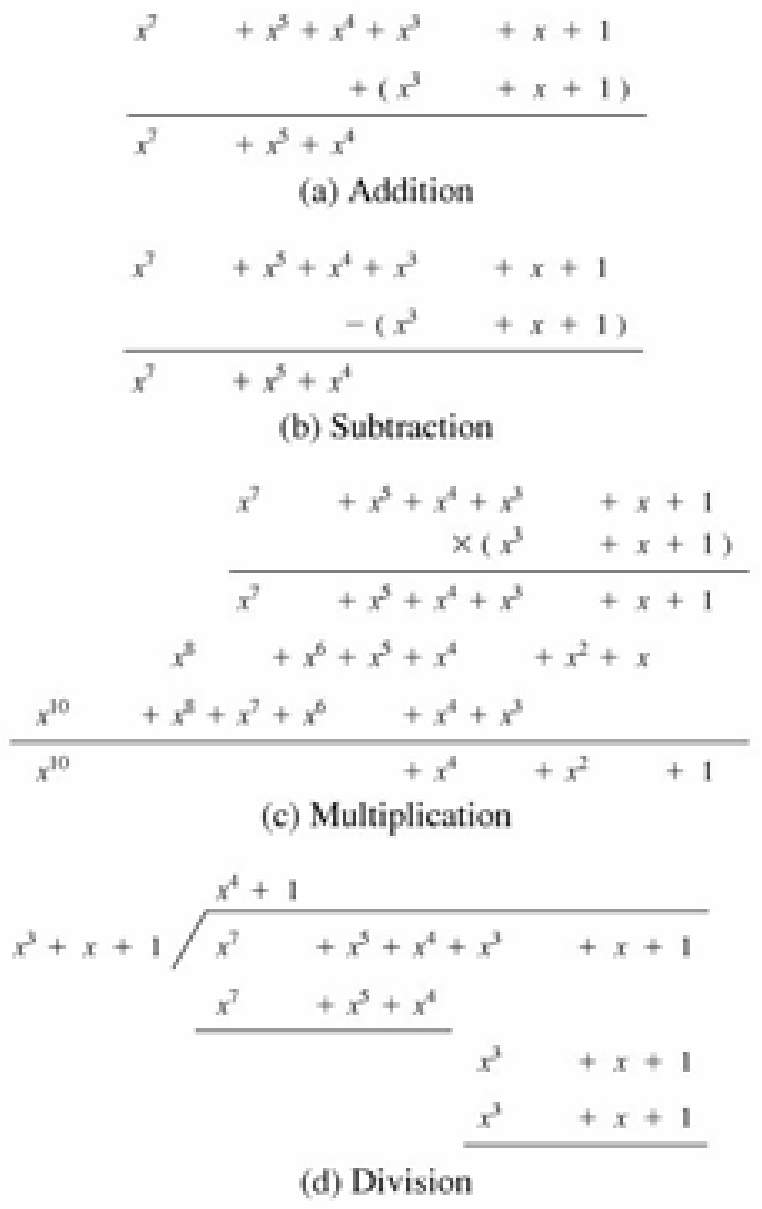
$q(x)g(x) + r(x) = (x + 2)(x^2 x + 1) + x = (x^3 + x^2 x + 2) + x$

$= x^3 + x^2 + 2 = f(x)$

For our purposes, polynomials over GF (2) are of most interest. That in GF(2), addition is equivalent to the XOR operation, and multiplication is equivalent to the logical AND operation. Further, addition and subtraction are equivalent mod 2: 1 + 1 = 1 1 = 0; 1 + 0 = 1 0 = 1; 0 + 1 = 0 1 = 1.

Figure 4.4 shows an example of polynomial arithmetic over GF(2). For $f(x) = (x^7 + x^5 + x^4 + x^3 + x + 1)$ and $g(x) = (x^3 + x + 1)$, the figure shows f(x) + g(x); f(x) g(x); f(x) x g(x); and f(x)/g(x). Note that g(x)|f(x)

Figure 4.4. Examples of Polynomial Arithmetic over GF(2)



A polynomial f(x) over a field F is called irreducible if and only if f(x) cannot be expressed as a product of two polynomials, both over F, and both

of degree lower than that of f(x). By analogy to integers, an irreducible polynomial is also called a **prime polynomial**.

The polynomial $f(x) = x^4 + 1$ over GF(2) is reducible, because $x^4 + 1 = (x + 1)(x^3 + x^2 + x + 1)$

Consider the polynomial $f(x) = x^3 + x + 1$. It is clear by inspection that x is not a factor of f(x). We easily show that x + 1 is not a factor of f(x):

$$
\begin{array}{r}
x^2 + x \\
x + 1 \overline{)\, x^3 \quad + x + 1} \\
\underline{x^3 + x^2} \\
x^2 + x \\
\underline{x^2 + x} \\
1
\end{array}
$$

Thus f(x) has no factors of degree 1. But it is clear by inspection that if f(x) is reducible, it must have one factor of degree 2 and one factor of degree 1. Therefore, f(x) is irreducible.

Finding the Greatest Common Divisor

We can extend the analogy between polynomial arithmetic over a field and integer arithmetic by defining the greatest common divisor as follows. The polynomial c(x) is said to be the greatest common divisor of a(x) and b(x) if

1. c(x) divides both a(x) and b(x);
2. any divisor of a(x) and b(x) is a divisor of c(x).

An equivalent definition is the following: gcd[a(x), b(x)] is the polynomial of maximum degree that divides both a(x) and b(x).

We can adapt the Euclidean algorithm to compute the greatest common divisor of two polynomials. The equality in Equation (4.4) can be rewritten as the following theorem:

Equation 4-7

$$\gcd[a(x), b(x)] = \gcd[b(x), a(x) \bmod b(x)]$$

The Euclidean algorithm for polynomials can be stated as follows. The algorithm assumes that the degree of a(x) is greater than the degree of b(x). Then, to find gcd[a(x), b(x)],

EUCLID[a(x), b(x)]
1. A(x)← a(x); B(x) ← b(x)
2. if B(x) = 0 return A(x) = gcd[a(x), b(x)]
3. R(x) = A(x) mod B(x)
4. A(x) ← B(x)
5. B(x) ← R(x)
6. goto 2

Find gcd[a(x), b(x)] for $a(x) = x^6 + x^5 + x^4 + x^3 + x^2 + x + 1$ and $b(x) = x^4 + x^2 + x + 1$.

$$
\begin{array}{r}
x^2 + x \\
x^4 + x^2 + x + 1 \overline{)\, x^6 + x^5 + x^4 + x^3 + x^2 + x + 1} \\
\underline{x^6 \qquad + x^4 + x^3 + x^2} \\
x^5 \qquad\qquad + x + 1 \\
\underline{x^5 \qquad + x^3 + x^2 + x} \\
x^3 + x^2 \qquad + 1
\end{array}
$$

A(x) = a(x); B(x) = b(x)

R(x) = A(x) mod B(x) = $x^3 + x^2 + 1$

A(x) = $x^4 + x^2 + x + 1$; B(x) = $x^3 + x^2 + 1$

Summary

We began this section with a discussion of arithmetic with ordinary polynomials. In ordinary polynomial arithmetic, the variable is not evaluated; that is, we do not plug a value in for the variable of the polynomials. Instead, arithmetic operations are performed on polynomials (addition, subtraction, multiplication, division) using the ordinary rules of algebra. Polynomial division is not allowed unless the coefficients are elements of a field.

Next, we discussed polynomial arithmetic in which the coefficients are elements of GF(p). In this case, polynomial addition, subtraction, multiplication, and division are allowed. However, division is not exact; that is, in general division results in a quotient and a remainder.

Finally, we showed that the Euclidean algorithm can be extended to find the greatest common divisor of two polynomials whose coefficients are elements of a field.

All of the material in this section provides a foundation for the following section, in which polynomials are used to define finite fields of order p^n.

Polynomials with Coefficients in GF (2^8)

we discussed polynomial arithmetic in which the coefficients are in Z_p and the polynomials are defined modulo a polynomial M(x) whose highest power is some integer n. In this case, addition and multiplication of coefficients occurred within the field Z_p; that is, addition and multiplication were performed modulo p.

The AES document defines polynomial arithmetic for polynomials of degree 3 or less with coefficients in GF(2^8). The following rules apply:

1. Addition is performed by adding corresponding coefficients in GF(2^8). if we treat the elements of GF(2^8) as 8-bit strings, then addition is equivalent to the XOR operation. So, if we have

   Equation 5-8

   $$a(x) = a_3x^3 + a_2x^2 + a_1x + a_0$$

   Equation 5-9

   $$b(x) = b_3x^3 + b_2x^2 + b_1x + b_0$$

   then

$$a(x) + b(x) = (a_3 \oplus b_3)x^3 + (a^2 \oplus b_2)x^2 + (a_1 \oplus b_1)x + (a_0 \oplus b_0)$$

2. Multiplication is performed as in ordinary polynomial multiplication, with two refinements:
    a. Coefficients are multiplied in $GF(2^8)$.
    b. The resulting polynomial is reduced mod $(x^4 + 1)$.

We need to keep straight which polynomial we are talking about. that each element of $GF(2^8)$ is a polynomial of degree 7 or less with binary coefficients, and multiplication is carried out modulo a polynomial of degree 8. Equivalently, each element of $GF(2^8)$ can be viewed as an 8-bit byte whose bit values correspond to the binary coefficients of the corresponding polynomial. For the sets defined in this section, we are defining a polynomial ring in which each element of this ring is a polynomial of degree 3 or less with coefficients in $GF(2^8)$, and multiplication is carried out modulo a polynomial of degree 4. Equivalently, each element of this ring can be viewed as a 4-byte word whose byte values are elements of $GF(2^8)$ that correspond to the 8-bit coefficients of the corresponding polynomial.

We denote the modular product of a(x) and b(x) by $a(x) \oplus b(x)$. To compute $d(x) = a(x) \oplus b(x)$, the first step is to perform a multiplication without the modulo operation and to collect coefficients of like powers. Let us express this as $c(x) = a(x) \times b(x)$ Then

Equation 5-10

$$c(x) = c_6x^6 + c_5x^5 + c_4x^4 + c_3x^3 + c_2x^2 + c_1x + c_0$$

where

$c_0 = a_0 \cdot b_0$

$c_1 = (a1 \cdot b_0) \oplus (a0 \cdot b_1)$

$c_2 = (a2 \cdot b_0) \oplus (a_1 \cdot b_1) \oplus (a_0 \cdot b_2)$

$c_3 = (a3 \cdot b_0) \oplus (a_2 \cdot b_1) \oplus (a_1 \cdot b_2) \qquad (a_0 \cdot b_3)$

$c_4 = (a_3 \cdot b_1) \oplus (a_2 \cdot b_2) \oplus (a_1 \cdot b_3)$

$c_5 = (a_3 \cdot b_2) \oplus (a_2 \cdot b_3)$

$c_6 = (a_3 \cdot b_3)$

The final step is to perform the modulo operation:

$d(x) = c(x) \bmod (x^4 + 1)$

That is, d(x) must satisfy the equation

$c(x) = [(x^4 + 1) \times q(x)] \oplus d(x)$

such that the degree of d(x) is 3 or less.

A practical technique for performing multiplication over this polynomial ring is based on the observation that

Equation 5-11

$$x^i \bmod (x^4 + 1) = x^{i \bmod 4}$$

If we now combine Equations (5.10) and (5.11), we end up with

$d(x) = c(x) \bmod (x^4 + 1) = [c_6x^6 + c_5x^5 + c_4x^4 + c_3x^3 + c_2x^2 + c_1x + c_0] \bmod (x^4 + 1)$

$= c_3x^3 + (c_2 \oplus c_6)x^2 + (c_1 \oplus c_5)x + (c_0 \oplus c_4)$

Expanding the c_i coefficients, we have the following equations for the coefficients of d(x):

$d_0 = (a_0 \cdot b_0) \oplus (a_3 \cdot b_1) \oplus (a) \cdot b_2) \oplus (a_1 \cdot b_3)$

$d_1 = (a_1 \cdot b_0) \oplus (a_0 \cdot b_1) \oplus (a_3 \cdot b_2) \oplus (a) \cdot b_3)$

$d_2 = (a_2) \cdot b_0) \oplus (a_1 \cdot b_1) \oplus (a_0 \cdot b_2) \oplus (a_3 \cdot b_3)$

$d_3 = (a_3 \cdot b_0) \oplus (a) \cdot b_1) \oplus (a_1 \cdot b_2) \oplus (a_0 \cdot b_3)$

This can be written in matrix form:

Equation 5-12

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

MixColumns Transformation

In the discussion of MixColumns, it was stated that there were two equivalent ways of defining the transformation. The first is the matrix multiplication shown in Equation (5.3), repeated here:

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

The second method is to treat each column of State as a four-term polynomial with coefficients in GF(2^8). Each column is multiplied modulo ($x^4 + 1$) by the fixed polynomial a(x), given by

$a(x = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$

From Equation (5.8), we have $a_3 = \{03\}$; $a_2 = \{01\}$; $a_0 = \{02\}$. For the jth column of State, we have the polynomial $col_j(x) = s_{3,j}x^3 + s_{2,j}x^2 + s_{1,j}x + s_{0,j}$. Substituting into Equation (5.12), we can express $d(x) = a(x) \times col_j(x)$ as

$$\begin{bmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{bmatrix} = \begin{bmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{bmatrix} \begin{bmatrix} s_{0,j} \\ s_{1,j} \\ s_{2,j} \\ s_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,j} \\ s_{1,j} \\ s_{2,j} \\ s_{3,j} \end{bmatrix}$$

which is equivalent to Equation (5.3).

Multiplication by x

Consider the multiplication of a polynomial in the ring by x: $c(x) = x \oplus b(x)$. We have

$$c(x) = x \oplus b(x) = [x \times (b_3x^3) + b_2x^2 + b_1x + b_0)] \bmod (x^4 + 1)$$

$$= (b_3x^4 + b_2x^3 + b_1x^2 + b_0x) \bmod (x^4 + 1)$$

$$= b_2x^3 + b_1x^2 + b_0x + b_3$$

Thus, multiplication by x corresponds to a 1-byte circular left shift of the 4 bytes in the word representing the polynomial. If we represent the polynomial as a 4-byte column vector, then we have

$$\begin{bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{bmatrix} = \begin{bmatrix} 00 & 00 & 00 & 01 \\ 01 & 00 & 00 & 00 \\ 00 & 01 & 00 & 00 \\ 00 & 00 & 01 & 00 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Evaluation Criteria For AES

The Origins of AES

in 1999, NIST issued a new version of its DES standard (FIPS PUB 46-3) that indicated that DES should only be used for legacy systems and that triple DES (3DES) be used. 3DES has two attractions that assure its widespread use over the next few years. First, with its 168-bit key length, it overcomes the vulnerability to brute-force attack of DES. Second, the underlying encryption algorithm in 3DES is the same as in DES. This algorithm has been subjected to more scrutiny than any other encryption algorithm over a longer period of time, and no effective cryptanalytic attack based on the algorithm rather than brute force has been found. Accordingly, there is a high level of confidence that 3DES is very resistant to cryptanalysis. If security were the only consideration, then 3DES would be an appropriate choice for a standardized encryption algorithm for decades to come.

The principal drawback of 3DES is that the algorithm is relatively sluggish in software. The original DES was designed for mid-1970s hardware implementation and does not produce efficient software code. 3DES, which has three times as many rounds as DES, is correspondingly slower. A secondary drawback is that both DES and 3DES use a 64-bit block size. For reasons of both efficiency and security, a larger block size is desirable.

Because of these drawbacks, 3DES is not a reasonable candidate for long- term use. As a replacement, NIST in 1997 issued a call for proposals for a new Advanced Encryption Standard (AES), which should have a security strength equal to or better than 3DES and significantly improved efficiency. In addition to these general requirements, NIST specified that AES must be a symmetric block cipher with a block length of 128 bits and support for key lengths of 128, 192, and 256 bits.

In a first round of evaluation, 15 proposed algorithms were accepted. A second round narrowed the field to 5 algorithms. NIST completed its evaluation process and published a final standard (FIPS PUB 197) in November of 2001. NIST selected Rijndael as the proposed AES algorithm. The two researchers who developed and submitted Rijndael for the AES are both cryptographers from Belgium: Dr. Joan Daemen and Dr. Vincent Rijmen.

Ultimately, AES is intended to replace 3DES, but this process will take a number of years. NIST anticipates that 3DES will remain an approved algorithm (for U.S. government use) for the foreseeable future.

AES Evaluation

It is worth examining the criteria used by NIST to evaluate potential candidates. These criteria span the range of concerns for the practical application of modern symmetric block ciphers. In fact, two set of criteria evolved. When NIST issued its original request for candidate algorithm nominations in 1997 , the request stated that candidate algorithms would be compared based on the factors shown in Table 5.1 (ranked in descending order of relative importance). The three categories of criteria were as follows:

- Security: This refers to the effort required to cryptanalyze an algorithm. The emphasis in the evaluation was on the practicality of the attack. Because the minimum key size for AES is 128 bits, brute-force attacks with current and projected technology were considered impractical. Therefore, the emphasis, with respect to this point, is cryptanalysis other than a brute-force attack.

  Cost: NIST intends AES to be practical in a wide range of applications. Accordingly, AES must have high computational

efficiency, so as to be usable in high-speed applications, such as broadband links.

- Algorithm and implementation characteristics: This category includes a variety of considerations, including flexibility; suitability for a variety of hardware and software implementations; and simplicity, which will make an analysis of security more straightforward.

Table 5.1. NIST Evaluation Criteria for AES (September 12, 1997)

SECURITY
• Actual security: compared to other submitted algorithms (at the same key and block size). • Randomness: the extent to which the algorithm output is indistinguishable from a random permutation on the input block. • Soundness: of the mathematical basis for the algorithm's security. • Other security factors: raised by the public during the evaluation process, including any attacks which demonstrate that the actual security of the algorithm is less than the strength claimed by the submitter.
COST
• Licensing requirements: NIST intends that when the AES is issued, the algorithm(s) specified in the AES shall be available on a worldwide, non-exclusive, royalty-free basis. • Computational efficiency: The evaluation of computational efficiency will be applicable to both hardware and software implementations. Round 1 analysis by NIST will focus primarily on software implementations and specifically on one key-block size combination (128-128); more attention will be paid to hardware implementations and other supported key-block size combinations during Round 2 analysis. Computational efficiency essentially refers to the speed of the algorithm. Public comments on each algorithm's efficiency (particularly for various platforms and applications) will also be taken into consideration by NIST. • Memory requirements: The memory required to implement a candidate algorithmfor both hardware and software implementations of the algorithmwill also be considered during the evaluation process. Round 1 analysis by NIST will focus primarily on software implementations; more attention will be paid to hardware implementations during Round 2. Memory requirements will include such factors as gate counts for hardware implementations, and code size and RAM requirements for software implementations.
ALGORITHM AND IMPLEMENTATION CHARACTERISTICS
• Flexibility: Candidate algorithms with greater flexibility will meet the needs of

Table 5.1. NIST Evaluation Criteria for AES (September 12, 1997)

SECURITY
more users than less flexible ones, and therefore, inter alia, are preferable. However, some extremes of functionality are of little practical application (e.g., extremely short key lengths); for those cases, preference will not be given. Some examples of flexibility may include (but are not limited to) the following: a. The algorithm can accommodate additional key- and block-sizes (e.g., 64-bit block sizes, key sizes other than those specified in the Minimum Acceptability Requirements section, [e.g., keys between 128 and 256 that are multiples of 32 bits, etc.]) b. The algorithm can be implemented securely and efficiently in a wide variety of platforms and applications (e.g., 8-bit processors, ATM networks, voice & satellite communications, HDTV, B-ISDN, etc.). c. The algorithm can be implemented as a stream cipher, message authentication code (MAC) generator, pseudorandom number generator, hashing algorithm, etc. • Hardware and software suitability: A candidate algorithm shall not be restrictive in the sense that it can only be implemented in hardware. If one can also implement the algorithm efficiently in firmware, then this will be an advantage in the area of flexibility. • Simplicity: A candidate algorithm shall be judged according to relative simplicity of design.

Using these criteria, the initial field of 21 candidate algorithms was reduced first to 15 candidates and then to 5 candidates. By the time that a final evaluation had been done the evaluation criteria, as described in , had evolved. The following criteria were used in the final evaluation:

- General security: To assess general security, NIST relied on the public security analysis conducted by the cryptographic community. During the course of the three-year evaluation process, a number of cryptographers published their analyses of the strengths and weaknesses of the various candidates. There was particular emphasis on analyzing the candidates with respect to known attacks, such as differential and linear cryptanalysis. However, compared to the analysisof DES, the amount of time and the number of cryptographers devoted to analyzing Rijndael are quite limited. Now that a single AES cipher has been chosen, we can expect to see a more extensive security analysis by the cryptographic community.

- Software implementations: The principal concerns in this category are execution speed, performance across a variety of platforms, and variation of speed with key size.
- Restricted-space environments: In some applications, such as smart cards, relatively small amounts of random-access memory (RAM) and/or read-only memory (ROM) are available for such purposes as code storage (generally in ROM); representation of data objects such as S-boxes (which could be stored in ROM or RAM, depending on whether pre-computation or Boolean representation is used); and subkey storage (in RAM).
- Hardware implementations: Like software, hardware implementations can be optimized for speed or for size. However, in the case of hardware, size translates much more directly into cost than is usually the case for software implementations. Doubling the size of an encryption program may make little difference on a general-purpose computer with a large memory, but doubling the area used in a hardware device typically more than doubles the cost of the device.
- Attacks on implementations: The criterion of general security, discussed in the first bullet, is concerned with cryptanalytic attacks that exploit mathematical properties of the algorithms. There is another class of attacks that use physical measurements conducted during algorithm execution to gather information about quantities such as keys. Such attacks exploit a combination of intrinsic algorithm characteristics and implementation-dependent features. Examples of such attacks are timing attacks and power analysis. The basic idea behind power analysis is the observation that the power consumed by a smart card at any particular time during the cryptographic operation is related to the instruction being executed and to the data being processed. For example, multiplication consumes more power than addition, and writing 1s consumes more power than writing 0s.

  Encryption versus decryption: This criterion deals with several issues related to considerations of both encryption and decryption. If the encryption and decryption algorithms differ, then extra space is needed for the decryption. Also, whether the two algorithms are the same or not, there may be timing differences between encryption and decryption.

- Key agility: Key agility refers to the ability to change keys quickly and with a minimum of resources. This includes both subkey

computation and the ability to switch between different ongoing security associations when subkeys may already be available.

- Other versatility and flexibility: indicates two areas that fall into this category. Parameter flexibility includes ease of support for other key and block sizes and ease of increasing the number of rounds in order to cope with newly discovered attacks. Implementation flexibility refers to the possibility of optimizing cipher elements for particular environments.
- Potential for instruction-level parallelism: This criterion refers to the ability to exploit ILP features in current and future processors.

Table 5.2 shows the assessment that NIST provided for Rijndael based on these criteria.

Table 5.2. Final NIST Evaluation of Rijndael (October 2, 2000)

General Security
Rijndael has no known security attacks. Rijndael uses S-boxes as nonlinear components. Rijndael appears to have an adequate security margin, but has received some criticism suggesting that its mathematical structure may lead to attacks. On the other hand, the simple structure may have facilitated its security analysis during the timeframe of the AES development process.
Software Implementations
Rijndael performs encryption and decryption very well across a variety of platforms, including 8-bit and 64-bit platforms, and DSPs. However, there is a decrease in performance with the higher key sizes because of the increased number of rounds that are performed. Rijndael's high inherent parallelism facilitates the efficient use of processor resources, resulting in very good software performance even when implemented in a mode not capable of interleaving. Rijndael's key setup time is fast.
Restricted-Space Environments
In general, Rijndael is very well suited for restricted-space environments where either encryption or decryption is implemented (but not both). It has very low RAM and ROM requirements. A drawback is that ROM requirements will increase if both encryption and decryption are implemented simultaneously, although it appears to remain suitable for these environments. The key schedule for decryption is separate from encryption.
Hardware Implementations
Rijndael has the highest throughput of any of the finalists for feedback modes and second

Table 5.2. Final NIST Evaluation of Rijndael (October 2, 2000)
General Security
highest for non-feedback modes. For the 192 and 256-bit key sizes, throughput falls in standard and unrolled implementations because of the additional number of rounds. For fully pipelined implementations, the area requirement increases, but the throughput is unaffected.
Attacks on Implementations
The operations used by Rijndael are among the easiest to defend against power and timing attacks. The use of masking techniques to provide Rijndael with some defense against these attacks does not cause significant performance degradation relative to the other finalists, and its RAM requirement remains reasonable. Rijndael appears to gain a major speed advantage over its competitors when such protections are considered.
Encryption vs. Decryption
The encryption and decryption functions in Rijndael differ. One FPGA study reports that the implementation of both encryption and decryption takes about 60% more space than the implementation of encryption alone. Rijndael's speed does not vary significantly between encryption and decryption, although the key setup performance is slower for decryption than for encryption.
Key Agility
Rijndael supports on-the-fly subkey computation for encryption. Rijndael requires a one-time execution of the key schedule to generate all subkeys prior to the first decryption with a specific key. This places a slight resource burden on the key agility of Rijndael.
Other Versatility and Flexibility
Rijndael fully supports block sizes and key sizes of 128 bits, 192 bits and 256 bits, in any combination. In principle, the Rijndael structure can accommodate any block sizes and key sizes that are multiples of 32, as well as changes in the number of rounds that are specified.
Potential for Instruction-Level Parallelism
Rijndael has an excellent potential for parallelism for a single block encryption.

Simplified AES

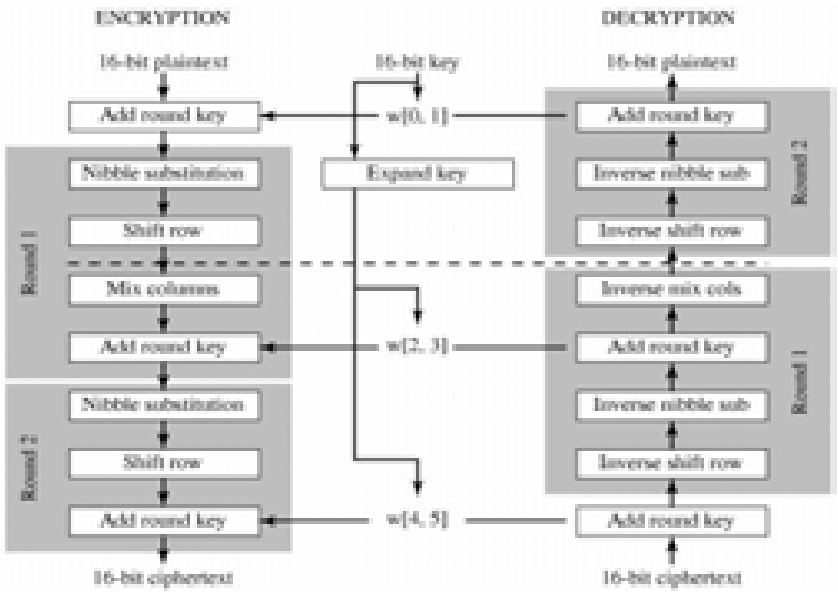
Simplified AES (S-AES) was developed by Professor Edward Schaefer of Santa Clara University and several of his students . It is an educational rather than a secure encryption algorithm. It has similar properties and structure to AES with much smaller parameters. The reader might find it useful to work through an example by hand while following the discussion in this appendix. A good grasp of S-AES will make it easier for the student to appreciate the structure and workings of AES.

Overview

Figure 5.8 illustrates the overall structure of S-AES. The encryption algorithm takes a 16-bit block of plaintext as input and a 16-bit key and produces a 16-bit block of ciphertext as output. The S-AES decryption algorithm takes an 16-bit block of ciphertext and the same 16-bit key used to produce that ciphertext as input and produces the original 16-bit block of plaintext as output.

Figure 5.8. S-AES Encryption and Decryption

(This item is displayed on page 165 in the print version)



The encryption algorithm involves the use of four different functions, or transformations: add key (A_K) nibble substitution (NS), shift row (SR), and mix column (MC), whose operation is explained subsequently.

We can concisely express the encryption algorithm as a composition of functions:

Definition: If f and g are two functions, then the function F with the equation y= F(x) = g[f(x)] is called the composition of f and g and is denoted as F = g º f.

A_{K2} º SR º NS º A_{K1} º MC º SR º NS º A_{K0}

so that A_{K0} is applied first.

The encryption algorithm is organized into three rounds. Round 0 is simply an add key round; round 1 is a full round of four functions; and round 2 contains only 3 functions. Each round includes the add key function, which makes use of 16 bits of key. The initial 16-bit key is expanded to 48 bits, so that each round uses a distinct 16-bit round key.

Each function operates on a 16-bit state, treated as a 2 x 2 matrix of nibbles, where one nibble equals 4 bits. The initial value of the state matrix is the 16-bit plaintext; the state matrix is modified by each subsequent function in the encryption process, producing after the last function the 16-bit ciphertext. As Figure 5.9a shows, the ordering of nibbles within the matrix is by column. So, for example, the first eight bits of a 16-bit plaintext input to the encryption cipher occupy the first column of the matrix, and the second eight bits occupy the second column. The 16-bit key is similarly organized, but it is somewhat more convenient to view the key as two bytes rather than four nibbles (Figure 5.9b). The expanded key of 48 bits is treated as three round keys, whose bits are labeled as follows: $K_0 = k_0...k_{15}$; $K_1 = k_{16}...k_{31}$; $K_2 = k_{32}...k_{47}$.

Figure 5.9. S-AES Data Structures

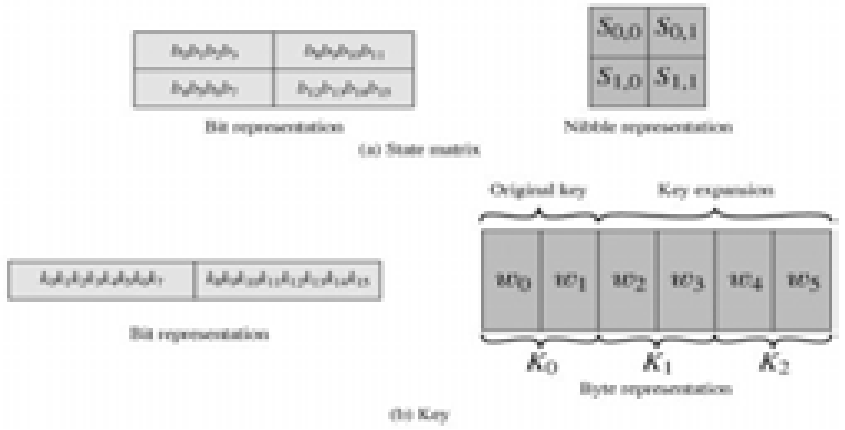
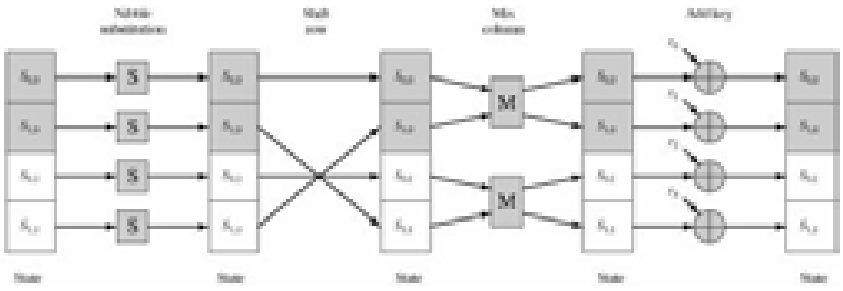


Figure 5.10 shows the essential elements of a full round of S-AES.

Figure 5.10. S-AES Encryption Round

(This item is displayed on page 167 in the print version)



Decryption is also shown in Figure 5.8 and is essentially the reverse of encryption:

$$A_{K0} \circ INS \circ ISR \circ IMC \circ A_{K1} \circ INS \circ ISR \circ A_{K2}$$

in which three of the functions have a corresponding inverse function: inverse nibble substitution (INS), inverse shift row (ISR), and inverse mix column (IMC).

S-AES Encryption and Decryption

We now look at the individual functions that are part of the encryption algorithm.

Add Key

The add key function consists of the bitwise XOR of the 16-bit state matrix and the 16-bit round key. Figure 5.11 depicts this as a columnwise operation, but it can also be viewed as a nibble-wise or bitwise operation. The following is an example:

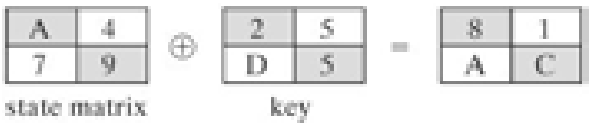
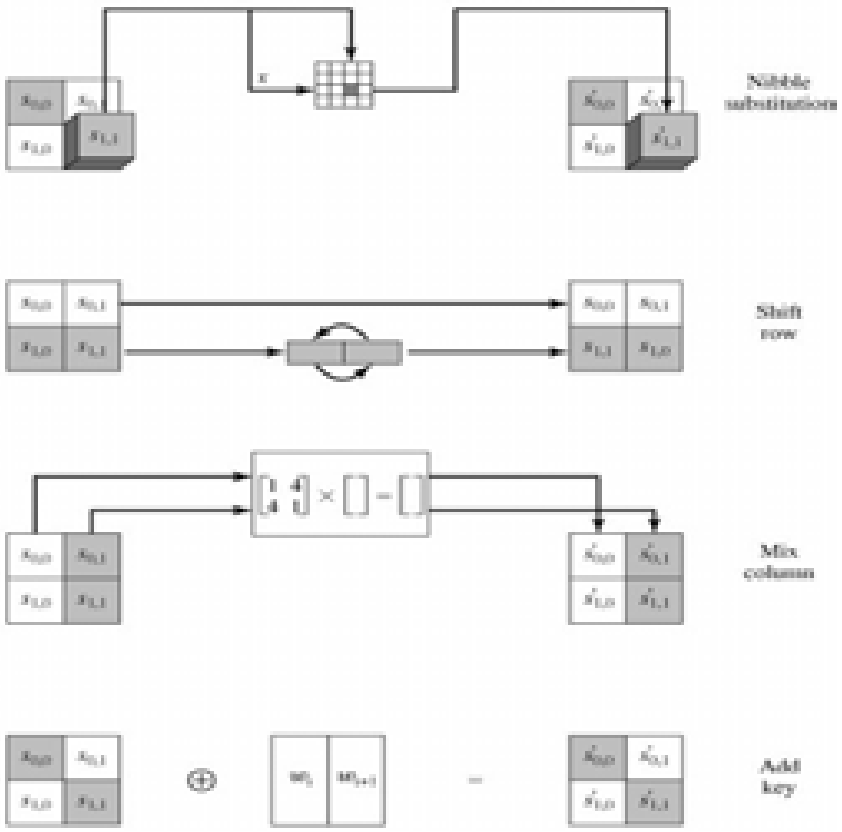


Figure 5.11. S-AES Transformations

[View full size image]



The inverse of the add key function is identical to the add key function, because the XOR operation is its own inverse.

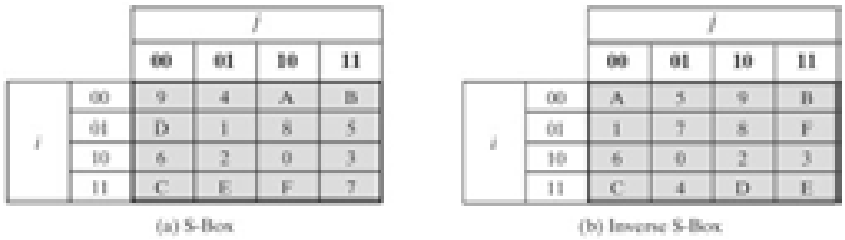
Nibble Substitution

The nibble substitution function is a simple table lookup (Figure 5.11). AES defines a 4 x 4 matrix of nibble values, called an S-box (Table 5.5a), that contains a permutation of all possible 4-bit values. Each individual nibble of the state matrix is mapped into a new nibble in the following way: The leftmost 2 bits of the nibble are used as a row value and the rightmost 2 bits are used as a column value. These row and column values serve as indexes into the S-box to select a unique 4-bit output value. For example, the

hexadecimal value A references row 2, column 2 of the S-box, which contains the value 0. Accordingly, the value A is mapped into the value 0.

Table 5.5. S-AES S-Boxes

Note: Hexadecimal numbers in shaded boxes; binary numbers in unshaded boxes.



		j			
		00	01	10	11
i	00	9	4	A	B
	01	D	1	8	5
	10	6	2	0	3
	11	C	E	F	7

(a) S-Box

		j			
		00	01	10	11
i	00	A	5	9	B
	01	1	7	8	F
	10	6	0	2	3
	11	C	4	D	E

(b) Inverse S-Box

Here is an example of the nibble substitution transformation:

8	1	→	6	4
A	C		0	C

The inverse nibble substitution function makes use of the inverse S-box shown in Table 5.5b. Note, for example, that the input 0 produces the output A, and the input A to the S-box produces 0.

Shift Row

The shift row function performs a one-nibble circular shift of the second row of the state matrix; the first row is not altered (Figure 5.11). The following is an example:

6	4	→	6	4
0	C		C	0

The inverse shift row function is identical to the shift row function, because it shifts the second row back to its original position.

Mix Column

The mix column function operates on each column individually. Each nibble of a column is mapped into a new value that is a function of both nibbles in

that column. The transformation can be defined by the following matrix multiplication on the state matrix (Figure 5.11):

$$\begin{bmatrix} 1 & 4 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} S_{0,0} & S_{0,1} \\ S_{1,0} & S_{1,1} \end{bmatrix} = \begin{bmatrix} S'_{0,0} & S'_{0,1} \\ S'_{1,0} & S'_{1,1} \end{bmatrix}$$

Performing the matrix multiplication, we get:

$S'_{0,0} = S_{0,0} \oplus (4 \cdot S_{1,0})$

$S'_{1,0} = (4 \cdot S_{0,0}) \oplus S_{1,0}$

$S'_{0,1} = S_{0,1} \oplus (4 \cdot S_{1,1})$

$S'_{1,1} = (4 \cdot S_{0,1}) \oplus S_{1,1}$

Where arithmetic is performed in GF(2^4), and the symbol · refers to multiplication in GF(2^4). Appendix E provides the addition and multiplication tables. The following is an example:

$$\begin{bmatrix} 1 & 4 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} 6 & 4 \\ C & 0 \end{bmatrix} = \begin{bmatrix} 3 & 4 \\ 7 & 3 \end{bmatrix}$$

The inverse mix column function is defined as follows:

$$\begin{bmatrix} 9 & 2 \\ 2 & 9 \end{bmatrix} \begin{bmatrix} S_{0,0} & S_{0,1} \\ S_{1,0} & S_{1,1} \end{bmatrix} = \begin{bmatrix} S'_{0,0} & S'_{0,1} \\ S'_{1,0} & S'_{1,1} \end{bmatrix}$$

We demonstrate that we have indeed defined the inverse in the following fashion:

$$\begin{bmatrix} 9 & 2 \\ 2 & 9 \end{bmatrix} \begin{bmatrix} 1 & 4 \\ 4 & 1 \end{bmatrix} \begin{bmatrix} S_{0,0} & S_{0,1} \\ S_{1,0} & S_{1,1} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} S_{0,0} & S_{0,1} \\ S_{1,0} & S_{1,1} \end{bmatrix} = \begin{bmatrix} S_{0,0} & S_{0,1} \\ S_{1,0} & S_{1,1} \end{bmatrix}$$

The preceding matrix multiplication makes use of the following results in GF(2^4): $9 + (2 \cdot 4) = 9 + 8 = 1$; $(9 \cdot 4) + 2 = 2 + 2 = 0$. These operations can be verified using the arithmetic tables in Appendix E or by polynomial arithmetic.

The mix column function is the most difficult to visualize. Accordingly, we provide an additional perspective on it in Appendix E.

Key Expansion

For key expansion, the 16 bits of the initial key are grouped into a row of two 8-bit words. Figure 5.12 shows the expansion into 6 words, by the calculation of 4 new words from the initial 2 words. The algorithm is as follows:

$w_2 = w_0 \oplus g(w_1) = w_0 \oplus RCON(1) \oplus SubNib(RotNib(w_1))$

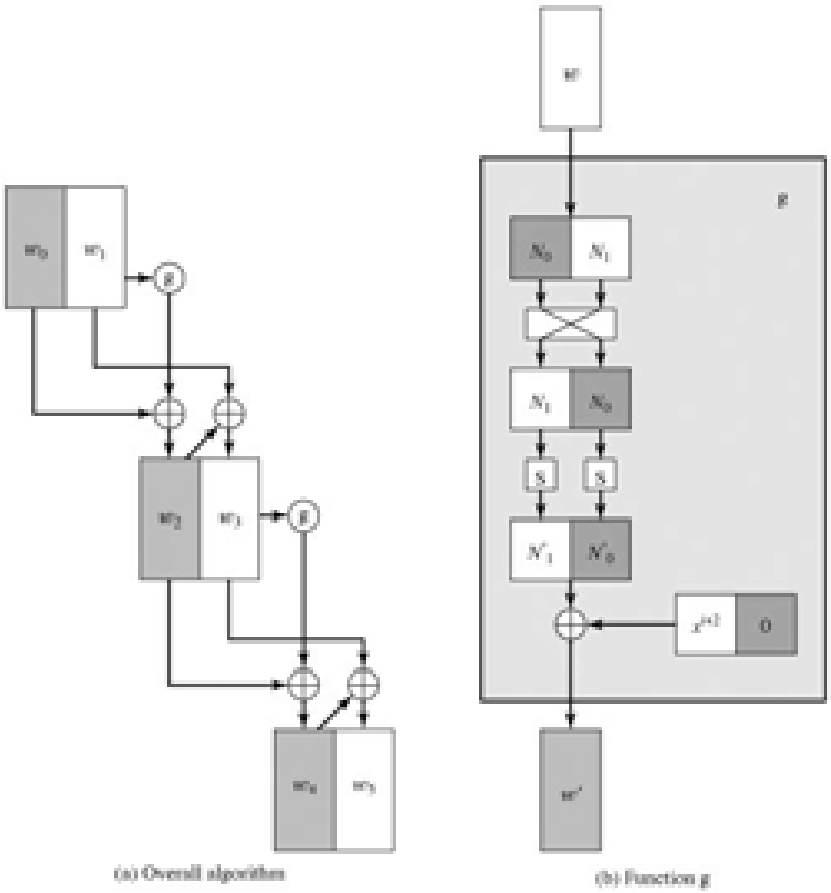
$w_3 = w_2 \oplus w_1$

$w_4 = w_2 \oplus g(w_3) = w_2 \oplus RCON(2) \oplus SubNib(RotNib(w_3))$

$w_5 = w_4 \oplus w_3$

Figure 5.12. S-AES Key Expansion

(This item is displayed on page 171 in the print version)



(a) Overall algorithm (b) Function g

RCON is a round constant, defined as follows: RC[i] = x^{i+2}, so that RC[1] = x^3 = 1000 and RC[2] = x^4 mod (x^4 + x + 1) = x + 1 = 0011. RC[i] forms the leftmost nibble of a byte, with the rightmost nibble being all zeros. Thus, RCON(1) = 10000000 and RCON(2) = 00110000.

For example, suppose the key is 2D55 = 0010 1101 0101 0101 = w_0w_1. Then

w_2 = 00101101 ⊕10000000 ⊕SubNib(01010101)

= 00101101 ⊕10000000 ⊕00010001 = 10111100

w_3 = 10111100 ⊕01010101 = 11101001

w_4 = 10111110 ⊕00110000 ⊕SubNib(10011110)

= 10111100 ⊕00110000 ⊕00101111 = 10100011

w_5 = 10100011 ⊕11101001 = 01001010

The S-Box

The S-box is constructed as follows:

> Initialize the S-box with the nibble values in ascending sequence row by row. The first row contains the hexadecimal values 0, 1, 2, 3; the second row contains 4, 5, 6, 7; and so on. Thus, the value of the nibble at row i, column j is 4i + j.

1. Treat each nibble as an element of the finite field GF(2^4) modulo x^4 +x + 1. Each nibble $a_0a_1a_2a_3$ represents a polynomial of degree 3.
2. Map each byte in the S-box to its multiplicative inverse in the finite field GF(2^4) modulo x^4 + x + 1; the value 0 is mapped to itself.
3. Consider that each byte in the S-box consists of 4 bits labeled (b_0, b_1, b_2, b_3). Apply the following transformation to each bit of each byte in the S-box: The AES standard depicts this transformation in matrix form as follows:

$$\begin{bmatrix} b_0' \\ b_1' \\ b_2' \\ b_3' \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

> The prime (') indicates that the variable is to be updated by the value on the right. Remember that addition and multiplication are being calculated modulo 2.

Table 5.5a shows the resulting S-box. This is a nonlinear, invertible matrix. The inverse S-box is shown in Table 5.5b.

S-AES Structure

We can now examine several aspects of interest concerning the structure of AES. First, note that the encryption and decryption algorithms begin and end with the add key function. Any other function, at the beginning or end, is easily reversible without knowledge of the key and so would add no security but just a processing overhead. Thus, there is a round 0 consisting of only the add key function.

The second point to note is that round 2 does not include the mix column function. The explanation for this in fact relates to a third observation, which is that although the decryption algorithm is the reverse of the encryption algorithm, as clearly seen in Figure 5.8, it does not follow the same sequence of functions. Thus

Encryption: A_{K2} º SR º NS º A_{K1} º MC º SR º NS º A_{K0}

Decryption: A_{K0} º INS º ISR º IMC º A_{K1} º INS º ISR º A_{K2}

From an implementation point of view, it would be desirable to have the decryption function follow the same function sequence as encryption. This allows the decryption algorithm to be implemented in the same way as the encryption algorithm, creating opportunities for efficiency.

Note that if we were able to interchange the second and third functions, the fourth and fifth functions, and the sixth and seventh functions in the decryption sequence, we would have the same structure as the encryption algorithm. Let's see if this is possible. First, consider the interchange of INS and ISR. Given a state N consisting of the nibbles (N_0, N_1, N_2, N_3) the transformation INS(ISR(N)) proceeds as follows:

$$\begin{pmatrix} N_0 & N_2 \\ N_1 & N_3 \end{pmatrix} \rightarrow \begin{pmatrix} N_0 & N_2 \\ N_3 & N_1 \end{pmatrix} \rightarrow \begin{pmatrix} \text{IS}[N_0] & \text{IS}[N_2] \\ \text{IS}[N_3] & \text{IS}[N_1] \end{pmatrix}$$

Where IS refers to the inverse S-Box. Reversing the operations, the transformation ISR(INS(N) proceeds as follows:

$$\begin{pmatrix} N_0 & N_2 \\ N_1 & N_3 \end{pmatrix} \rightarrow \begin{pmatrix} IS[N_0] & IS[N_2] \\ IS[N_1] & IS[N_3] \end{pmatrix} \rightarrow \begin{pmatrix} IS[N_0] & IS[N_2] \\ IS[N_3] & IS[N_1] \end{pmatrix}$$

which is the same result. Thus, INS(ISR(N)) = ISR(INS(N)).

Now consider the operation of inverse mix column followed by add key: IMC(A_{K1}(N)) where the round key K_1 consists of the nibbles ($k_{0,0}$, $k_{1,0}$, $k_{0,1}$, $k_{1,1}$) Then:

$$\begin{pmatrix} 9 & 2 \\ 2 & 9 \end{pmatrix}\left(\begin{pmatrix} k_{0,0} & k_{0,1} \\ k_{1,0} & k_{1,1} \end{pmatrix} \oplus \begin{pmatrix} N_0 & N_2 \\ N_1 & N_3 \end{pmatrix}\right) = \begin{pmatrix} 9 & 2 \\ 2 & 9 \end{pmatrix}\begin{pmatrix} k_{0,0} \oplus N_0 & k_{0,1} \oplus N_2 \\ k_{1,0} \oplus N_1 & k_{1,1} \oplus N_3 \end{pmatrix}$$

$$= \begin{pmatrix} 9(k_{0,0} \oplus N_0) \oplus 2(K_{1,0} \oplus N_1) & 9(k_{0,1} \oplus N_2) \oplus 2(K_{1,1} \oplus N_3) \\ 2(k_{0,0} \oplus N_0) \oplus 9(K_{1,0} \oplus N_1) & 2(k_{0,1} \oplus N_2) \oplus 9(K_{1,1} \oplus N_3) \end{pmatrix}$$

$$= \begin{pmatrix} (9k_{0,0} \oplus 2k_{1,0}) \oplus (9N_0 \oplus 2N_1) & (9k_{0,1} \oplus 2k_{1,1}) \oplus (9N_2 \oplus 2N_3) \\ (2k_{0,0} \oplus 9k_{1,0}) \oplus (2N_0 \oplus 9N_1) & (2k_{0,1} \oplus 9k_{1,1}) \oplus (2N_2 \oplus 9N_3) \end{pmatrix}$$

$$= \begin{pmatrix} (9k_{0,0} \oplus 2k_{1,0}) & (9k_{0,1} \oplus 2k_{1,1}) \\ (2k_{0,0} \oplus 9k_{1,0}) & (2k_{0,1} \oplus 9k_{1,1}) \end{pmatrix} \oplus \begin{pmatrix} (9N_0 \oplus 2N_1) & (9N_2 \oplus 2N_3) \\ (2N_0 \oplus 9N_1) & (2N_2 \oplus 9N_3) \end{pmatrix}$$

$$= \begin{pmatrix} 9 & 2 \\ 2 & 9 \end{pmatrix}\begin{pmatrix} k_{0,0} & k_{0,1} \\ k_{1,0} & k_{1,1} \end{pmatrix} \oplus \begin{pmatrix} 9 & 2 \\ 2 & 9 \end{pmatrix}\begin{pmatrix} N_0 & N_2 \\ N_1 & N_3 \end{pmatrix}$$

All of the above steps make use of the properties of finite field arithmetic.

The result is that IMC(A_{K1}(N)) = IMC(K_1 ⊕ IMC(N). Now let us define the inverse round key for round 1 to be IMC(K_1) and the inverse add key operation IA_{K1} to be the bitwise XOR of the inverse round key with the state vector. Then we have IMC(A_{K1}(N)) = IA_{K1}(IMC(N)). As a result, we can write the following:

Encryption: A_{K2} º SR º NS º A_{K1} º MC º SR º NS º A_{K0}

Decryption: A_{K0} º INS º ISR º IMC º A_{K1} º INS º ISR º A_{K2}

Decryption: A_{K0} º ISR º INS º $A_{IMC(K1)}$ º IMC º ISR º INS º A_{K2}

Both encryption and decryption now follow the same sequence. Note that this derivation would not work as effectively if round 2 of the encryption algorithm included the MC function. In that case, we would have

Encryption: A_{K2} º MC º SR º NS º A_{K1} º MC º SR º NS º A_{K0}

Decryption: A_{K0} º INS º ISR º IMC º A_{K1} º INS º ISR º IMC º A_{K2}

There is now no way to interchange pairs of operations in the decryption algorithm so as to achieve the same structure as the encryption algorithm.

The AES Cipher

The Rijndael proposal for AES defined a cipher in which the block length and the key length can be independently specified to be 128, 192, or 256 bits. The AES specification uses the same three key size alternatives but limits the block length to 128 bits. A number of AES parameters depend on the key length (Table 5.3). In the description of this section, we assume a key length of 128 bits, which is likely to be the one most commonly implemented.

Table 5.3. AES Parameters

Key size (words/bytes/bits)	4/16/128	6/24/192	8/32/256
Plaintext block size (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Number of rounds	10	12	14
Round key size (words/bytes/bits)	4/16/128	4/16/128	4/16/128
Expanded key size (words/bytes)	44/176	52/208	60/240

Rijndael was designed to have the following characteristics:

- Resistance against all known attacks
- Speed and code compactness on a wide range of platforms
- Design simplicity

Figure 5.1 shows the overall structure of AES. The input to the encryption and decryption algorithms is a single 128-bit block. In FIPS PUB 197, this block is depicted as a square matrix of bytes. This block is copied into the

State array, which is modified at each stage of encryption or decryption. After the final stage, State is copied to an output matrix. These operations are depicted in Figure 5.2a. Similarly, the 128-bit key is depicted as a square matrix of bytes. This key is then expanded into an array of key schedule words; each word is four bytes and the total key schedule is 44 words for the 128-bit key (Figure 5.2b). Note that the ordering of bytes within a matrix is by column. So, for example, the first four bytes of a 128-bit plaintext input to the encryption cipher occupy the first column of the in matrix, the second four bytes occupy the second column, and so on. Similarly, the first four bytes of the expanded key, which form a word, occupy the first column of the w matrix.

Figure 5.1. AES Encryption and Decryption

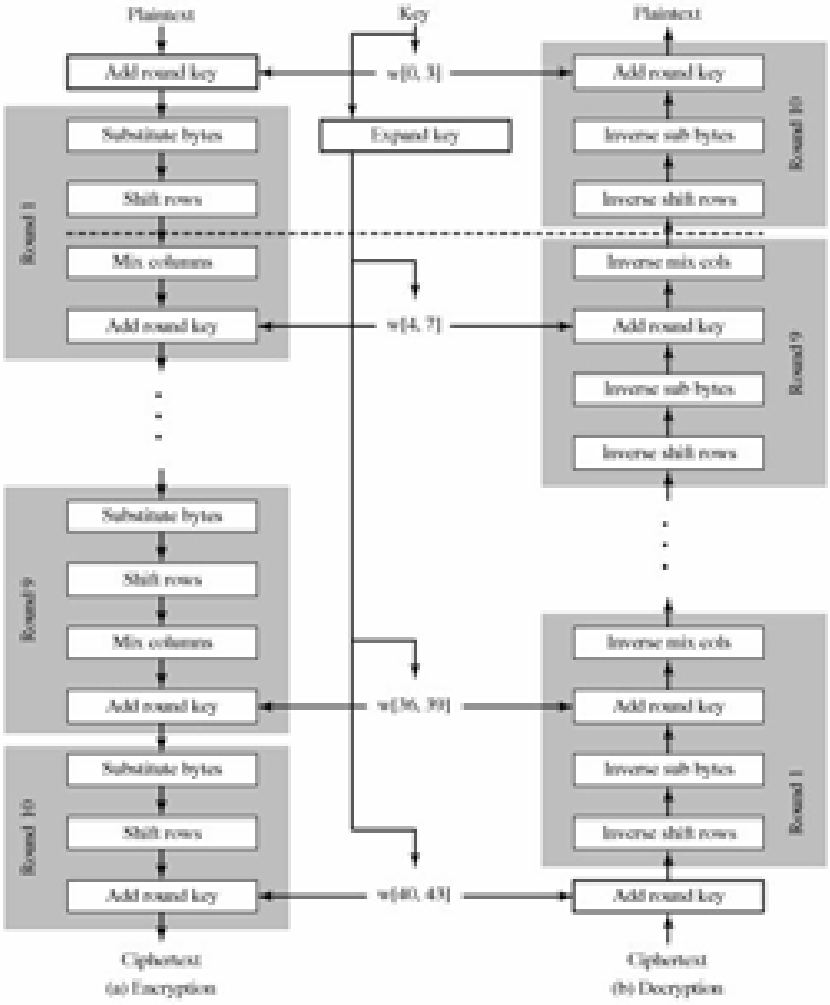
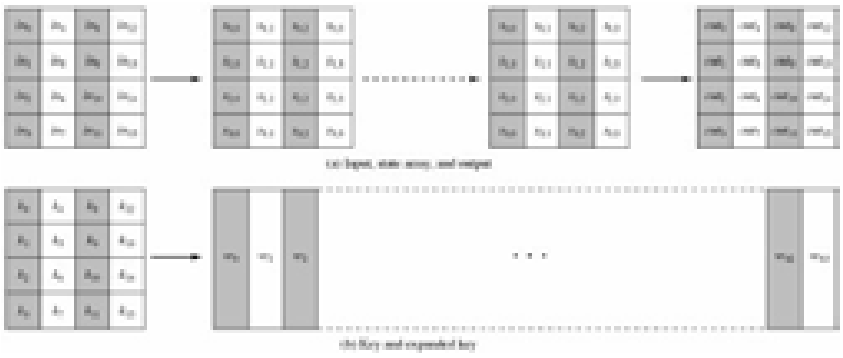


Figure 5.2. AES Data Structures

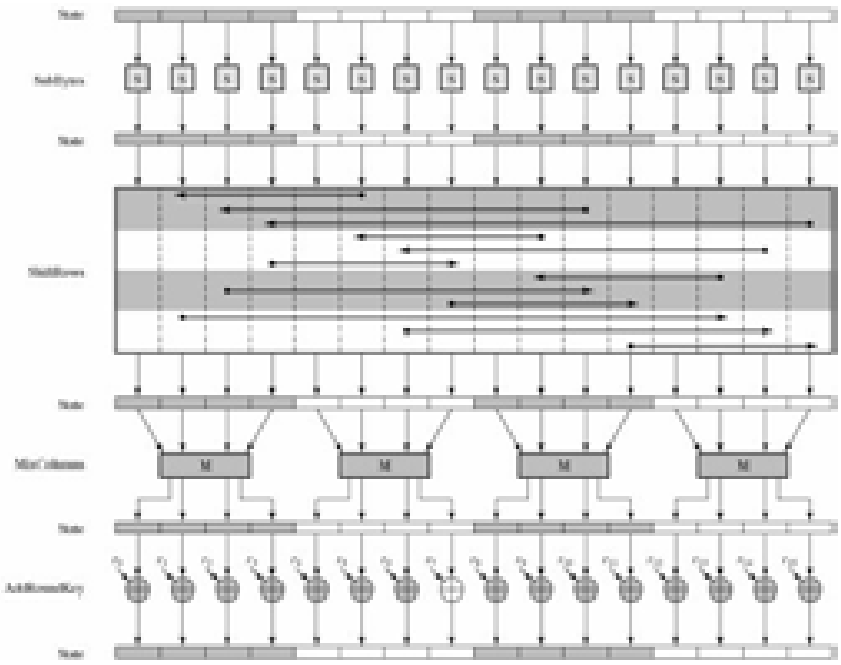


Before delving into details, we can make several comments about the overall AES structure:

1. One noteworthy feature of this structure is that it is not a Feistel structure. Recall that in the classic Feistel structure, half of the data block is used to modify the other half of the data block, and then the halves are swapped. Two of the AES finalists, including Rijndael, do not use a Feistel structure but process the entire data block in parallel during each round using substitutions and permutation.
2. The key that is provided as input is expanded into an array of forty-four 32-bit words, w[i]. Four distinct words (128 bits) serve as a round key for each round; these are indicated in Figure 5.1.
3. Four different stages are used, one of permutation and three of substitution:
    - Substitute bytes: Uses an S-box to perform a byte-by-byte substitution of the block
    - ShiftRows: A simple permutation
    - MixColumns: A substitution that makes use of arithmetic over GF(2^8)
    - AddRoundKey: A simple bitwise XOR of the current block with a portion of the expanded key
4. The structure is quite simple. For both encryption and decryption, the cipher begins with an AddRoundKey stage, followed by nine rounds that each includes all four stages, followed by a tenth round of three stages. Figure 5.3 depicts the structure of a full encryption round.

Figure 5.3. AES Encryption Round

(This item is displayed on page 144 in the print version)



5. Only the AddRoundKey stage makes use of the key. For this reason, the cipher begins and ends with an AddRoundKey stage. Any other stage, applied at the beginning or end, is reversible without knowledge of the key and so would add no security.
6. The AddRoundKey stage is, in effect, a form of Vernam cipher and by itself would not be formidable. The other three stages together provide confusion, diffusion, and nonlinearity, but by themselves would provide no security because they do not use the key. We can view the cipher as alternating operations of XOR encryption (AddRoundKey) of a block, followed by scrambling of the block (the other three stages), followed by XOR encryption, and so on. This scheme is both efficient and highly secure.
7. Each stage is easily reversible. For the Substitute Byte, ShiftRows, and MixColumns stages, an inverse function is used in the decryption algorithm. For the AddRoundKey stage, the inverse is achieved by XORing the same round key to the block, using the result that $A \oplus A \oplus B = B$.

8. As with most block ciphers, the decryption algorithm makes use of the expanded key in reverse order. However, the decryption algorithm is not identical to the encryption algorithm. This is a consequence of the particular structure of AES.

   Once it is established that all four stages are reversible, it is easy to verify that decryption does recover the plaintext. Figure 5.1 lays out encryption and decryption going in opposite vertical directions. At each horizontal point (e.g., the dashed line in the figure), State is the same for both encryption and decryption.

9. The final round of both encryption and decryption consists of only three stages. Again, this is a consequence of the particular structure of AES and is required to make the cipher reversible.

We now turn to a discussion of each of the four stages used in AES. For each stage, we describe the forward (encryption) algorithm, the inverse (decryption) algorithm, and the rationale for the stage. This is followed by a discussion of key expansion.

AES uses arithmetic in the finite field $GF(2^8)$, with the irreducible polynomial $m(x) = x^8 + x^4 + x^3 + x + 1$. The developers of Rijndael give as their motivation for selecting this one of the 30 possible irreducible polynomials of degree 8 that it is the first one on the list .

Substitute Bytes Transformation

Forward and Inverse Transformations

The forward substitute byte transformation, called SubBytes, is a simple table lookup (Figure 5.4a). AES defines a 16 x 16 matrix of byte values, called an S-box (Table 5.4a), that contains a permutation of all possible 256
8-bit values. Each individual byte of State is mapped into a new byte in the following way: The leftmost 4 bits of the byte are used as a row value and the rightmost 4 bits are used as a column value. These row and column values serve as indexes into the S-box to select a unique 8-bit output value. For example, the hexadecimal value {95} references row 9, column 5 of the S-box, which contains the value {2A}. Accordingly, the value {95} is mapped into the value {2A}.

In FIPS PUB 197, a hexadecimal number is indicated by enclosing it in curly brackets. We use that convention in this chapter.

Figure 5.4. AES Byte-Level Operations

(This item is displayed on page 145 in the print version)

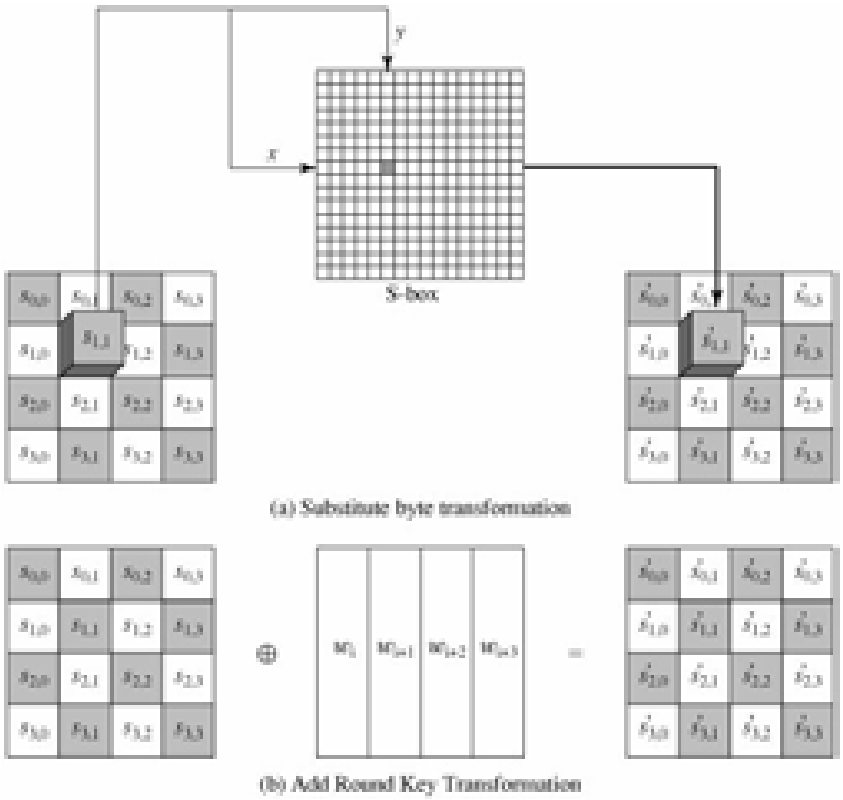


Table 5.4. AES S-Boxes

(This item is displayed on page 146 in the print version)

(a) S-box

		y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x	0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
	1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
	2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
	3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
	4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
	5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
	6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
	7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
	8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
	9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
	A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
	B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
	C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
	D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
	E	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
	F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

(b) Inverse S-box

		y															
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
x	0	52	09	6A	D5	30	36	A5	38	BF	40	A3	9E	81	F3	D7	FB
	1	7C	E3	39	82	9B	2F	FF	87	34	8E	43	44	C4	DE	E9	CB
	2	54	7B	94	32	A6	C2	23	3D	EE	4C	95	0B	42	FA	C3	4E
	3	08	2E	A1	66	28	D9	24	B2	76	5B	A2	49	6D	8B	D1	25
	4	72	F8	F6	64	86	68	98	16	D4	A4	5C	CC	5D	65	B6	92
	5	6C	70	48	50	FD	ED	B9	DA	5E	15	46	57	A7	8D	9D	84
	6	90	D8	AB	00	8C	BC	D3	0A	F7	E4	58	05	B8	B3	45	06
	7	D0	2C	1E	8F	CA	3F	0F	02	C1	AF	BD	03	01	13	8A	6B
	8	3A	91	11	41	4F	67	DC	EA	97	F2	CF	CE	F0	B4	E6	73
	9	96	AC	74	22	E7	AD	35	85	E2	F9	37	E8	1C	75	DF	6E
	A	47	F1	1A	71	1D	29	C5	89	6F	B7	62	0E	AA	18	BE	1B
	B	FC	56	3E	4B	C6	D2	79	20	9A	DB	C0	FE	78	CD	5A	F4
	C	1F	DD	A8	33	88	07	C7	31	B1	12	10	59	27	80	EC	5F
	D	60	51	7F	A9	19	B5	4A	0D	2D	E5	7A	9F	93	C9	9C	EF
	E	A0	E0	3B	4D	AE	2A	F5	B0	C8	EB	BB	3C	83	53	99	61
	F	17	2B	04	7E	BA	77	D6	26	E1	69	14	63	55	21	0C	7D

Here is an example of the SubBytes transformation:

EA	04	65	85
83	45	5D	96
5C	33	98	B0
F0	2D	AD	C5

→

87	F2	4D	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6

The S-box is constructed in the following fashion:

1. Initialize the S-box with the byte values in ascending sequence row by row. The first row contains {00}, {01}, {02},.... {0F}; the second row contains {10}, {11}, etc.; and so on. Thus, the value of the byte at row x, column y is {xy}.
2. Map each byte in the S-box to its multiplicative inverse in the finite field GF(2^8); the value {00} is mapped to itself.
3. Consider that each byte in the S-box consists of 8 bits labeled (b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0). Apply the following transformation to each bit of each byte in the S-box:

   Equation 5-1

$$b'_i = b_i \oplus b_{(i+4) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+6) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus c_i$$

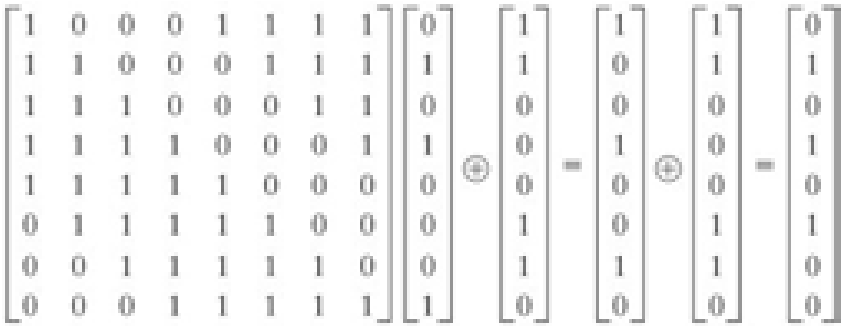
   where c_i is the ith bit of byte c with the value {63}; that is, ($c_7c_6c_5c_4c_3c_2c_1c_0$) = (01100011). The prime (') indicates that the variable is to be updated by the value on the right. The AES standard depicts this transformation in matrix form as follows:

   Equation 5-2

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Equation (5.2) has to be interpreted carefully. In ordinary matrix multiplication,[5] each element in the product matrix is the sum of products of the elements or one row and one column. In this case, each element in the product matrix is the bitwise XOR of products of elements of one row and one column. Further, the final addition shown in Equation (5.2) is a bitwise XOR.

As an example, consider the input value {95}. The multiplicative inverse in GF(2^8) is $\{95\}^{-1} = \{8A\}$, which is 10001010 in binary. Using Equation (5.2),



The result is {2A}, which should appear in row {09} column {05} of the S-box. This is verified by checking Table 5.4a.

The inverse substitute byte transformation, called InvSubBytes, makes use of the inverse S-box shown in Table 5.4b. Note, for example, that the input {2A} produces the output {95} and the input {95} to the S-box produces {2A}. The inverse S-box is constructed by applying the inverse of the transformation in Equation (5.1) followed by taking the multiplicative inverse in GF(2^8). The inverse transformation is:

$$b_i' = b_{(i+2) \bmod 8} \oplus b_{(i+5) \bmod 8} \oplus b_{(i+7) \bmod 8} \oplus d_i$$

where byte d = {05}, or 00000101. We can depict this transformation as follows:

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

To see that InvSubBytes is the inverse of SubBytes, label the matrices in SubBytes and InvSubBytes as X and Y, respectively, and the vector versions of constants c and d as C and D, respectively. For some 8-bit vector B, Equation (5.2) becomes B' = XB ⊕C. We need to show that Y(XB ⊕C) ⊕D = B. Multiply out, we must show YXB ⊕YC ⊕D = B. This becomes

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} \oplus$$

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} =$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix}$$

We have demonstrated that YX equals the identity matrix, and the YC = D, so that YC ⊕D equals the null vector.

Rationale

The S-box is designed to be resistant to known cryptanalytic attacks. Specifically, the Rijndael developers sought a design that has a low correlation between input bits and output bits, and the property that the output cannot be described as a simple mathematical function of the input . In addition, the constant in Equation (5.1) was chosen so that the S-box has no fixed points [S-box(a) = a] and no "opposite fixed points" [S-box(a) = ā], where ā is the bitwise complement of a.

Of course, the S-box must be invertible, that is, IS-box[S-box(a)] = a. However, the S-box is not self-inverse in the sense that it is not true that S- box(a) = IS-box(a). For example, [S-box({95}) = {2A}, but IS-box({95}) = {AD}.

ShiftRows Transformation

Forward and Inverse Transformations

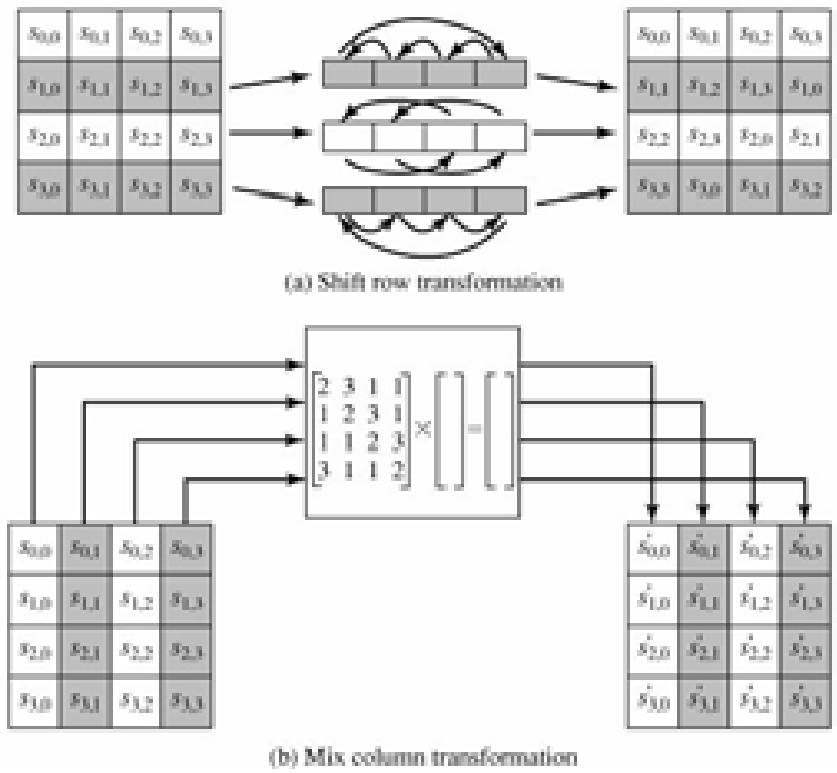
The forward shift row transformation, called ShiftRows, is depicted in Figure 5.5a. The first row of State is not altered. For the second row, a 1-byte circular left shift is performed. For the third row, a 2-byte circular left shift is performed. For the fourth row, a 3-byte circular left shift is performed. The following is an example of ShiftRows:

87	F2	4D	97
EC	6E	4C	90
4A	C3	46	E7
8C	D8	95	A6

→

87	F2	4D	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95

Figure 5.5. AES Row and Column Operations



The inverse shift row transformation, called InvShiftRows, performs the circular shifts in the opposite direction for each of the last three rows, with a one-byte circular right shift for the second row, and so on.

Rationale

The shift row transformation is more substantial than it may first appear. This is because the State, as well as the cipher input and output, is treated as an array of four 4-byte columns. Thus, on encryption, the first 4 bytes of the plaintext are copied to the first column of State, and so on. Further, as will be seen, the round key is applied to State column by column. Thus, a row shift moves an individual byte from one column to another, which is a linear distance of a multiple of 4 bytes. Also note that the transformation ensures

that the 4 bytes of one column are spread out to four different columns. Figure 5.3 illustrates the effect.

MixColumns Transformation

Forward and Inverse Transformations

The forward mix column transformation, called MixColumns, operates on each column individually. Each byte of a column is mapped into a new value that is a function of all four bytes in that column. The transformation can be defined by the following matrix multiplication on State (Figure 5.5b):

Equation 5-3

$$\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

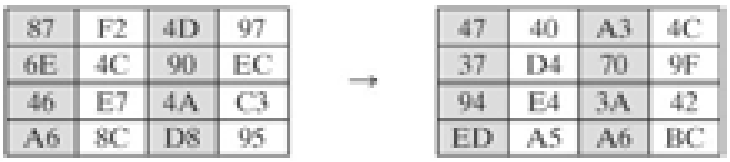
Each element in the product matrix is the sum of products of elements of one row and one column. In this case, the individual additions and multiplications are performed in GF(2^8). The MixColumns transformation on a single column j($0 \leq j \leq 3$) of State can be expressed as

We follow the convention of FIPS PUB 197 and use the symbol · to indicate multiplication over the finite field GF(2^8) and ⊕to indicate bitwise XOR, which corresponds to addition in GF(2^8).

Equation 5-4

$$\begin{aligned} s'_{0,j} &= (2 \cdot s_{0,j}) \oplus (3 \cdot s_{1,j}) \oplus s_{2,j} \oplus s_{3,j} \\ s'_{1,j} &= s_{0,j} \oplus (2 \cdot s_{1,j}) \oplus (3 \cdot s_{2,j}) \oplus s_{3,j} \\ s'_{2,j} &= s_{0,j} \oplus s_{1,j} \oplus (2 \cdot s_{2,j}) \oplus (3 \cdot s_{3,j}) \\ s'_{3,j} &= (3 \cdot s_{0,j}) \oplus s_{1,j} \oplus s_{2,j} \oplus (2 \cdot s_{3,j}) \end{aligned}$$

The following is an example of MixColumns:



87	F2	4D	97
6E	4C	90	EC
46	E7	4A	C3
A6	8C	D8	95

→

47	40	A3	4C
37	D4	70	9F
94	E4	3A	42
ED	A5	A6	BC

Let us verify the first column of this example, in GF(2^8), addition is the bitwise XOR operation and that multiplication can be performed according to the rule established in Equation (4.10). In particular, multiplication of a value by x (i.e., by {02}) can be implemented as a 1-bit left shift followed by a conditional bitwise XOR with (0001 1011) if the leftmost bit of the original value (prior to the shift) is 1. Thus, to verify the MixColumns transformation on the first column, we need to show that

$$(\{02\} \cdot \{87\}) \oplus (\{03\} \cdot \{6E\}) \oplus \{46\} \oplus \{A6\} = \{47\}$$

$$\{87\} \oplus (\{02\} \cdot \{6E\}) \oplus (\{03\} \cdot \{46\}) \oplus \{A6\} = \{37\}$$

$$\{87\} \oplus \{6E\} \oplus (\{02\} \cdot \{46\}) \oplus (\{03\} \cdot \{A6\}) = \{94\}$$

$$(\{03\} \cdot \{87\}) \oplus \{6E\} \oplus \{46\} \oplus (\{02\} \cdot \{A6\}) = \{ED\}$$

For the first equation, we have {02} · {87} = (0000 1110) ⊕ (0001 1011) = (0001 0101); and {03} · {6E} = {6E} ⊕ ({02} · {6E}) = (0110 1110) ⊕ (1101 1100) = (1011 0010). Then

$\{02\} \cdot \{87\}$ = 0001 0101

$\{03\} \cdot \{6E\}$ = 1011 0010

{46} = 0100 0110

{A6} = 1010 0110

0100 0111 = {47}

The other equations can be similarly verified.

The inverse mix column transformation, called InvMixColumns, is defined by the following matrix multiplication:

Equation 5-5

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s'_{0,0} & s'_{0,1} & s'_{0,2} & s'_{0,3} \\ s'_{1,0} & s'_{1,1} & s'_{1,2} & s'_{1,3} \\ s'_{2,0} & s'_{2,1} & s'_{2,2} & s'_{2,3} \\ s'_{3,0} & s'_{3,1} & s'_{3,2} & s'_{3,3} \end{bmatrix}$$

It is not immediately clear that Equation (5.5) is the inverse of Equation (5.3). We need to show that:

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix} = \begin{bmatrix} s_{0,0} & s_{0,1} & s_{0,2} & s_{0,3} \\ s_{1,0} & s_{1,1} & s_{1,2} & s_{1,3} \\ s_{2,0} & s_{2,1} & s_{2,2} & s_{2,3} \\ s_{3,0} & s_{3,1} & s_{3,2} & s_{3,3} \end{bmatrix}$$

which is equivalent to showing that:

Equation 5-6

$$\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

That is, the inverse transformation matrix times the forward transformation matrix equals the identity matrix. To verify the first column of Equation (5.6), we need to show that:

$(\{0E\} \cdot \{02\}) \oplus \{0B\} \oplus \{0D\} \oplus (\{09\} \cdot \{03\}) = \{01\}$

$(\{09\} \cdot \{02\}) \oplus \{0E\} \oplus \{0B\} \oplus (\{0D\} \cdot \{03\}) = \{00\}$

$(\{0D\} \cdot \{02\}) \oplus \{09\} \oplus \{0E\} \oplus (\{0B\} \cdot \{03\}) = \{00\}$

$(\{0B\} \cdot \{02\}) \oplus \{0D\} \oplus \{09\} \oplus (\{0E\} \cdot \{03\}) = \{00\}$

For the first equation, we have {0E} · {02}) ⊕00011100; and {09} · {03} = {09} ⊕ ({09} · {02}) = 00001001 ⊕00010010 = 00011011. Then

{0E} · {02}	=	00011100
{0B}	=	00001011
{0D}	=	00001101
{09} · {03}	=	00011011
		00000001

The other equations can be similarly verified.

The AES document describes another way of characterizing the MixColumns transformation, which is in terms of polynomial arithmetic. In the standard, MixColumns is defined by considering each column of State to be a four-term polynomial with coefficients in GF(2^8). Each column is multiplied modulo ($x^4 + 1$) by the fixed polynomial a(x), given by

Equation 5-7

$$a(x) = \{03\}x^3 + \{01\}x^2 + \{01\}x + \{02\}$$

demonstrates that multiplication of each column of State by a(x) can be written as the matrix multiplication of Equation (5.3). Similarly, it can be seen that the transformation in Equation (5.5) corresponds to treating each column as a four-term polynomial and multiplying each column by b(x), given by

Equation 5-8

$$b(x) = \{0B\}x^3 + \{0D\}x^2 + \{09\}x + \{0E\}$$

It can readily be shown that $b(x) = a^1 (x) \bmod (x^4 + 1)$.

Rationale

The coefficients of the matrix in Equation (5.3) are based on a linear code with maximal distance between code words, which ensures a good mixing among the bytes of each column. The mix column transformation combined with the shift row transformation ensures that after a few rounds, all output bits depend on all input bits.

In addition, the choice of coefficients in MixColumns, which are all {01}, {02}, or {03}, was influenced by implementation considerations. As was discussed, multiplication by these coefficients involves at most a shift and an XOR. The coefficients in InvMixColumns are more formidable to implement. However, encryption was deemed more important than decryption for two reasons:

1. For the CFB and OFB cipher modes only encryption is used.
2. As with any block cipher, AES can be used to construct a message authentication code (Part Two), and for this only encryption is used.

AddRoundKey Transformation

Forward and Inverse Transformations

In the forward add round key transformation, called AddRoundKey, the 128 bits of State are bitwise XORed with the 128 bits of the round key. As shown in Figure 5.4b, the operation is viewed as a columnwise operation between the 4 bytes of a State column and one word of the round key; it can also be viewed as a byte-level operation. The following is an example of AddRoundKey:

The first matrix is State, and the second matrix is the round key.

The inverse add round key transformation is identical to the forward add round key transformation, because the XOR operation is its own inverse.

Rationale

The add round key transformation is as simple as possible and affects every bit of State. The complexity of the round key expansion, plus the complexity of the other stages of AES, ensure security.

AES Key Expansion

Key Expansion Algorithm

The AES key expansion algorithm takes as input a 4-word (16-byte) key and produces a linear array of 44 words (176 bytes). This is sufficient to provide a 4-word round key for the initial AddRoundKey stage and each of the 10 rounds of the cipher. The following pseudocode describes the expansion:

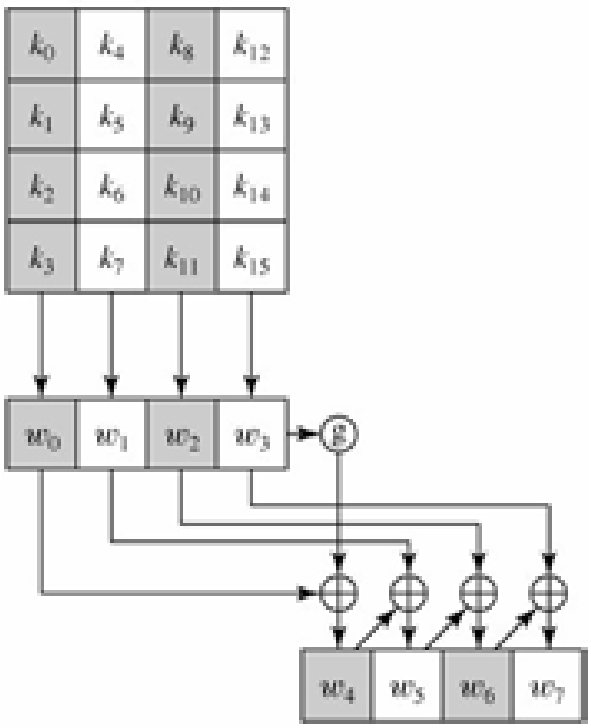
```
KeyExpansion (byte key[16], word w[44])
{
   word temp
   for (i = 0; i < 4; i++) w[i] = (key[4*i],
    key[4*i+1],
                              key[4*i+2],
                              key[4*i+3]);
   for (i = 4; i < 44; i++)
   {
    temp = w[i  1];
    if (i mod 4 = 0) temp = SubWord (RotWord (temp))
                         ⊕ Rcon[i/4];
    w[i] = w[i4] ≈temp
   }
}
```

The key is copied into the first four words of the expanded key. The remainder of the expanded key is filled in four words at a time. Each added word w[i] depends on the immediately preceding word, w[i 1], and the word four positions back,w[i 4]. In three out of four cases, a simple XOR is used. For a word whose position in the w array is a

multiple of 4, a more complex function is used. Figure 5.6 illustrates the generation of the first eight words of the expanded key, using the symbol g to represent that complex function. The function g consists of the following subfunctions:

1. RotWord performs a one-byte circular left shift on a word. This means that an input word [b0, b1, b2, b3] is transformed into [b1, b2, b3, b0].
2. SubWord performs a byte substitution on each byte of its input word, using the S-box (Table 5.4a).
3. The result of steps 1 and 2 is XORed with a round constant, Rcon[j].

Figure 5.6. AES Key Expansion



The round constant is a word in which the three rightmost bytes are always 0. Thus the effect of an XOR of a word with Rcon is to only perform an XOR on the leftmost byte of the word. The round constant is different for each round and is defined as Rcon[j] = (RC[j], 0, 0, 0), with RC[1] = 1,

RC[j] = 2 · RC[j - 1] and with multiplication defined over the field GF(2^8). The values of RC[j] in hexadecimal are

j	1	2	3	4	5	6	7	8	9	10
RC[j]	01	02	04	08	10	20	40	80	1B	36

For example, suppose that the round key for round 8 is

EA D2 73 21 B5 8D BA D2 31 2B F5 60 7F 8D 29 2F

Then the first 4 bytes (first column) of the round key for round 9 are calculated as follows:

i (decimal)	temp	After RotWord	After SubWord	Rcon (9)	After XOR with Rcon	w[i 4]	w[i] = temp ⊕w[i 4]
36	7F8D292F	8D292F7F	5DA515D2	1B000000	46A515D2	EAD27321	AC7766F3

Rationale

The Rijndael developers designed the expansion key algorithm to be resistant to known cryptanalytic attacks. The inclusion of a round-dependent round constant eliminates the symmetry, or similarity, between the ways in which round keys are generated in different rounds:

- Knowledge of a part of the cipher key or round key does not enable calculation of many other round key bits
- An invertible transformation [i.e., knowledge of any Nk consecutive words of the Expanded Key enables regeneration the entire expanded key (Nk = key size in words)]
- Speed on a wide range of processors
- Usage of round constants to eliminate symmetries
- Diffusion of cipher key differences into the round keys; that is, each key bit affects many round key bits
- Enough nonlinearity to prohibit the full determination of round key differences from cipher key differences only
- Simplicity of description

The authors do not quantify the first point on the preceding list, but the idea is that if you know less than Nk consecutive words of either the cipher key or one of the round keys, then it is difficult to reconstruct the remaining unknown bits. The fewer bits one knows, the more difficult it is to do the reconstruction or to determine other bits in the key expansion.

Equivalent Inverse Cipher

As was mentioned, the AES decryption cipher is not identical to the encryption cipher (Figure 5.1). That is, the sequence of transformations for decryption differs from that for encryption, although the form of the key schedules for encryption and decryption is the same. This has the disadvantage that two separate software or firmware modules are needed for applications that require both encryption and decryption. There is, however, an equivalent version of the decryption algorithm that has the same structure as the encryption algorithm. The equivalent version has the same sequence of transformations as the encryption algorithm (with transformations replaced by their inverses). To achieve this equivalence, a change in key schedule is needed.

Two separate changes are needed to bring the decryption structure in line with the encryption structure. An encryption round has the structure SubBytes, ShiftRows, MixColumns, AddRoundKey. The standard decryption round has the structure InvShiftRows, InvSubBytes, AddRoundKey, InvMixColumns. Thus, the first two stages of the decryption round need to be interchanged, and the second two stages of the decryption round need to be interchanged.

Interchanging InvShiftRows and InvSubBytes

InvShiftRows affects the sequence of bytes in State but does not alter byte contents and does not depend on byte contents to perform its transformation. InvSubBytes affects the contents of bytes in State but does not alter byte sequence and does not depend on byte sequence to perform its transformation. Thus, these two operations commute and can be interchanged. For a given State S_i,

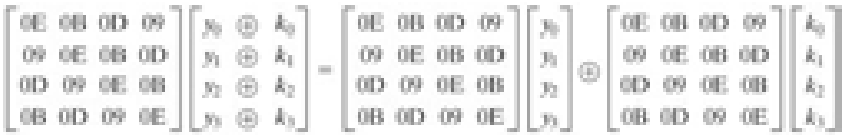
InvShiftRows [InvSubBytes (S_i)] = InvSubBytes [InvShiftRows (S_i)]

Interchanging AddRoundKey and InvMixColumns

The transformations AddRoundKey and InvMixColumns do not alter the sequence of bytes in State. If we view the key as a sequence of words, then both AddRoundKey and InvMixColumns operate on State one column at a time. These two operations are linear with respect to the column input. That is, for a given State S_i and a given round key w_j:

$$\text{InvMixColumns}\ (S_i \oplus w_j) = [\text{InvMixColumns}\ (S_i)] \oplus [\text{InvMixColumns}\ (w_j)]$$

To see this, suppose that the first column of State S_i is the sequence (y_0, y_1, y_2, y_3) and the first column of the round key w_j is (k_0, k_1, k_2, k_3). Then we need to show that



Let us demonstrate that for the first column entry. We need to show that:

$$[\{0E\} \cdot (y_0 \oplus k_0)] \oplus [\{0B\} \cdot (y_1 \oplus k_1)] \oplus [\{0D\} \cdot (y_2 \oplus k_2)] \oplus [\{09\} \cdot (y_3 \oplus k_3)]$$

$$= [\{0E\} \cdot y_0] \oplus [\{0B\} \cdot y_1] \oplus [\{0D\} \cdot y_2] \oplus [\{09\} \cdot y_3]$$

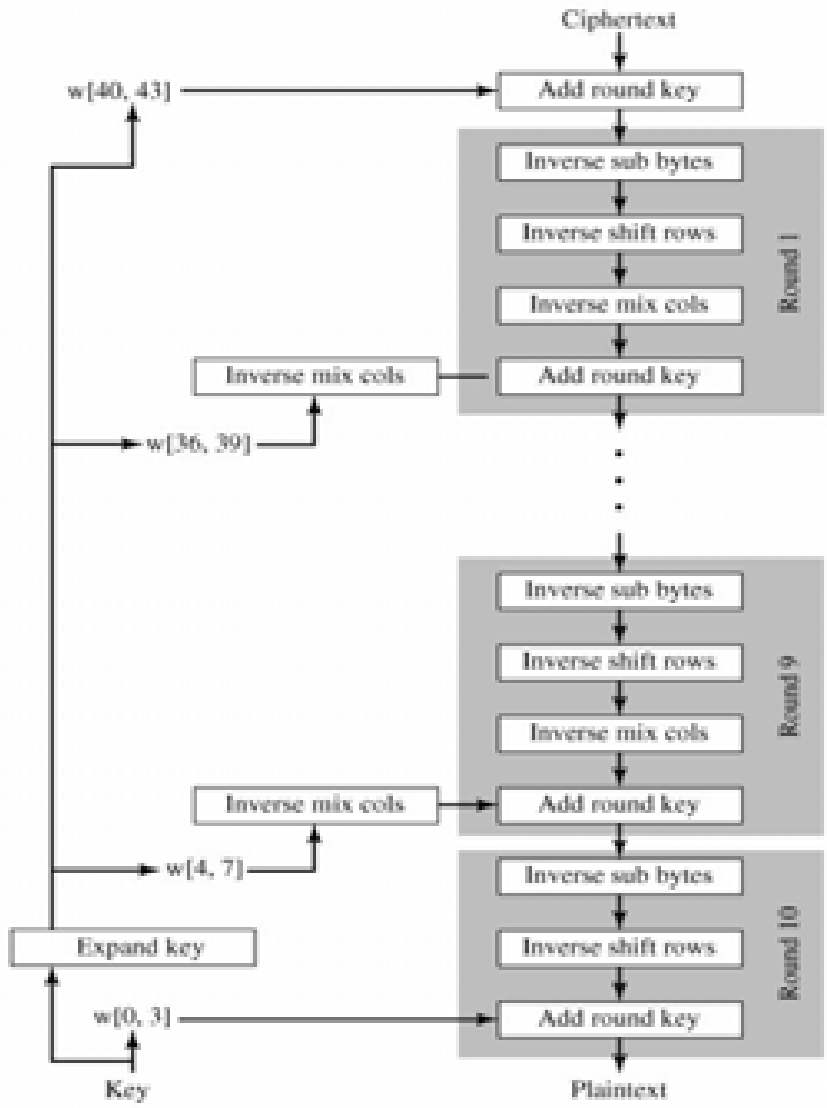
$$\oplus [[\{0E\} \cdot k_0] \oplus] [\{0B\} \cdot k_1] \oplus [\{0D\} \cdot k_2] \oplus [\{09\} \cdot k_3]$$

This equation is valid by inspection. Thus, we can interchange AddRoundKey and InvMixColumns, provided that we first apply InvMixColumns to the round key. Note that we do not need to apply InvMixColumns to the round key for the input to the first AddRoundKey transformation (preceding the first round) nor to the last AddRoundKey transformation (in round 10). This is because these two AddRoundKey transformations are not interchanged with InvMixColumns to produce the equivalent decryption algorithm.

Figure 5.7 illustrates the equivalent decryption algorithm.

Figure 5.7. Equivalent Inverse Cipher

(This item is displayed on page 158 in the print version)



Implementation Aspects

The Rijndael proposal provides some suggestions for efficient implementation on 8-bit processors, typical for current smart cards, and on 32-bit processors, typical for PCs.

8-Bit Processor

AES can be implemented very efficiently on an 8-bit processor. AddRoundKey is a bytewise XOR operation. ShiftRows is a simple byte shifting operation. SubBytes operates at the byte level and only requires a table of 256 bytes.

The transformation MixColumns requires matrix multiplication in the field GF(2^8), which means that all operations are carried out on bytes. MixColumns only requires multiplication by {02} and {03}, which, as we have seen, involved simple shifts, conditional XORs, and XORs. This can be implemented in a more efficient way that eliminates the shifts and conditional XORs. Equation Set (5.4) shows the equations for the MixColumns transformation on a single column. Using the identity {03} · x = ({02} · x) ⊕x, we can rewrite Equation Set (5.4) as follows:

Equation 5-9

$$
\begin{aligned}
Tmp &= s_{0,j} \oplus s_{1,j} \oplus s_{2,j} \oplus s_{3,j} \\
s'_{0,j} &= s_{0,j} \oplus Tmp \oplus [2 \cdot (s_{0,j} \oplus s_{1,j})] \\
s'_{1,j} &= s_{1,j} \oplus Tmp \oplus [2 \cdot (s_{1,j} \oplus s_{2,j})] \\
s'_{2,j} &= s_{2,j} \oplus Tmp \oplus [2 \cdot (s_{2,j} \oplus s_{3,j})] \\
s'_{3,j} &= s_{3,j} \oplus Tmp \oplus [2 \cdot (s_{3,j} \oplus s_{0,j})]
\end{aligned}
$$

Equation Set (5.9) is verified by expanding and eliminating terms.

The multiplication by {02} involves a shift and a conditional XOR. Such an implementation may be vulnerable to a timing attack of the sort described in Section 3.4. To counter this attack and to increase processing efficiency at the cost of some storage, the multiplication can be replaced by a table lookup. Define the 256-byte table X2, such that X2[i] = {02} · i. Then Equation Set (5.9) can be rewritten as

$Tmp = s_o, j \oplus s_1, j \oplus s2, j \oplus s3, j$

$s'_0, j = s_0, j \oplus Tmp \oplus X2[s_o, j \oplus s_1, j]$

$s'_1, c = s_1, j \oplus Tmp \oplus X2[s_1, j \oplus s_2, j]$

$s'_2, c = s_2, j \oplus Tmp \oplus X2[s_2, j \oplus s_3, j]$

$s'_3, j = s_3, j \oplus Tmp \oplus X2[s_3, j \oplus s_0, j]$

32-Bit Processor

The implementation described in the preceding subsection uses only 8-bit operations. For a 32-bit processor, a more efficient implementation can be achieved if operations are defined on 32-bit words. To show this, we first define the four transformations of a round in algebraic form. Suppose we begin with a State matrix consisting of elements $a_{i,j}$ and a round key matrix consisting of elements $k_{i,j}$. Then the transformations can be expressed as follows:

SubBytes	$b_{i,j} = S[a_{i,j}]$

ShiftRows

$$\begin{bmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{bmatrix} = \begin{bmatrix} b_{0,j} \\ b_{1,j-1} \\ b_{2,j-2} \\ b_{3,j-3} \end{bmatrix}$$

MixColumns

AddRoundKey

$$\begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} c_{0,j} \\ c_{1,j} \\ c_{2,j} \\ c_{3,j} \end{bmatrix}$$

$$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = \begin{bmatrix} d_{0,j} \\ d_{1,j} \\ d_{2,j} \\ d_{3,j} \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}$$

In the ShiftRows equation, the column indices are taken mod 4. We can combine all of these expressions into a single equation:

$$\begin{bmatrix} e_{0,j} \\ e_{1,j} \\ e_{2,j} \\ e_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S[a_{0,j}] \\ S[a_{1,j-1}] \\ S[a_{2,j-2}] \\ S[a_{3,j-3}] \end{bmatrix} \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}$$

$$= \left(\begin{bmatrix} 02 \\ 01 \\ 01 \\ 03 \end{bmatrix} \cdot S[a_{0,j}] \right) \oplus \left(\begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix} \cdot S[a_{1,j-1}] \right) \oplus \left(\begin{bmatrix} 01 \\ 03 \\ 02 \\ 01 \end{bmatrix} \cdot S[a_{2,j-2}] \right)$$

$$\oplus \left(\begin{bmatrix} 01 \\ 01 \\ 03 \\ 02 \end{bmatrix} \cdot S[a_{3,j-3}] \right) \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}$$

In the second equation, we are expressing the matrix multiplication as a linear combination of vectors. We define four 256-word (1024-byte) tables as follows:

$$T_0[x] = \left(\begin{bmatrix} 02 \\ 01 \\ 01 \\ 03 \end{bmatrix} \cdot S[x] \right) \quad T_1[x] = \left(\begin{bmatrix} 03 \\ 02 \\ 01 \\ 01 \end{bmatrix} \cdot S[x] \right) \quad T_2[x] = \left(\begin{bmatrix} 01 \\ 03 \\ 02 \\ 01 \end{bmatrix} \cdot S[x] \right) \quad T_3[x] = \left(\begin{bmatrix} 01 \\ 01 \\ 03 \\ 02 \end{bmatrix} \cdot S[x] \right)$$

Thus, each table takes as input a byte value and produces a column vector (a 32-bit word) that is a function of the S-box entry for that byte value. These tables can be calculated in advance.

We can define a round function operating on a column in the following fashion:

$$\begin{bmatrix} s'_{0,j} \\ s'_{1,j} \\ s'_{2,j} \\ s'_{3,j} \end{bmatrix} = T_0[s_{0,j}] \oplus T_1[s_{1,j-1}] \oplus T_2[s_{2,j-2}] \oplus T_3[s_{3,j-3}] \oplus \begin{bmatrix} k_{0,j} \\ k_{1,j} \\ k_{2,j} \\ k_{3,j} \end{bmatrix}$$

As a result, an implementation based on the preceding equation requires only four table lookups and four XORs per column per round, plus 4 Kbytes to store the table. The developers of Rijndael believe that this compact, efficient implementation was probably one of the most important factors in the selection of Rijndael for AES.

t a few rounds.

Elliptic Curve Arithmetic

Most of the products and standards that use public-key cryptography for encryption and digital signatures use RSA. As we have seen, the key length for secure RSA use has increased over recent years, and this has put a heavier processing load on applications using RSA. This burden has ramifications, especially for electronic commerce sites that conduct large numbers of secure transactions. Recently, a competing system has begun to challenge RSA: elliptic curve cryptography (ECC). Already, ECC is showing up in standardization efforts, including the IEEE P1363 Standard for Public-Key Cryptography.

The principal attraction of ECC, compared to RSA, is that it appears to offer equal security for a far smaller key size, thereby reducing processing overhead. On the other hand, although the theory of ECC has been around for some time, it is only recently that products have begun to appear and that there has been sustained cryptanalytic interest in probing for weaknesses. Accordingly, the confidence level in ECC is not yet as high as that in RSA.

ECC is fundamentally more difficult to explain than either RSA or Diffie-Hellman, and a full mathematical description is beyond the scope of this book. This section and the next give some background on elliptic curves and ECC. We begin with a brief review of the concept of abelian group. Next, we examine the concept of elliptic curves defined over the real numbers.

This is followed by a look at elliptic curves defined over finite fields. Finally, we are able to examine elliptic curve ciphers.

Abelian Groups

that an abelian group G, sometimes denoted by {G, • }, is a set of elements with a binary operation, denoted by •, that associates to each ordered pair (a, b) of elements in G an element (a • b) in G, such that the following axioms are obeyed:

The operator • is generic and can refer to addition, multiplication, or some other mathematical operation.

(A1) Closure:	If a and b belong to G, then a • b is also in G.
(A2) Associative:	a • (b • c) = (a • b) • c for all a, b, c in G.
(A3) Identity element:	There is an element e in G such that a • e = e • a = a for all a in G.
(A4) Inverse element:	For each a in G there is an element a' in G such that a • a' = a' • a = e.
(A5) Commutative:	a • b = b • a for all a, b in G.

A number of public-key ciphers are based on the use of an abelian group. For example, Diffie-Hellman key exchange involves multiplying pairs of nonzero integers modulo a prime number q. Keys are generated by exponentiation over the group, with exponentiation defined as repeated multiplication. For example, $a^k \bmod q = \underbrace{(a \times a \times \ldots \times a)}_{k \text{ times}} \bmod q$. To attack Diffie-Hellman, the attacker must determine k given a and a^k; this is the discrete log problem.

For elliptic curve cryptography, an operation over elliptic curves, called addition, is used. Multiplication is defined by repeated addition. For example, $a \times k = \underbrace{(a + a + \ldots + a)}_{k \text{ times}}$, where the addition is performed over an elliptic

curve. Cryptanalysis involves determining k given a and (a x k).

An elliptic curve is defined by an equation in two variables, with coefficients. For cryptography, the variables and coefficients are restricted to

elements in a finite field, which results in the definition of a finite abelian group. Before looking at this, we first look at elliptic curves in which the variables and coefficients are real numbers. This case is perhaps easier to visualize.

Elliptic Curves over Real Numbers

Elliptic curves are not ellipses. They are so named because they are described by cubic equations, similar to those used for calculating the circumference of an ellipse. In general, cubic equations for elliptic curves take the form

$$y^2 + axy + by = x^3 + cx^2 + dx + e$$

where a, b, c, d, and e are real numbers and x and y take on values in the real numbers. For our purpose, it is sufficient to limit ourselves to equations of the form

Equation 10-1

$$y^2 = x^3 + ax + b$$

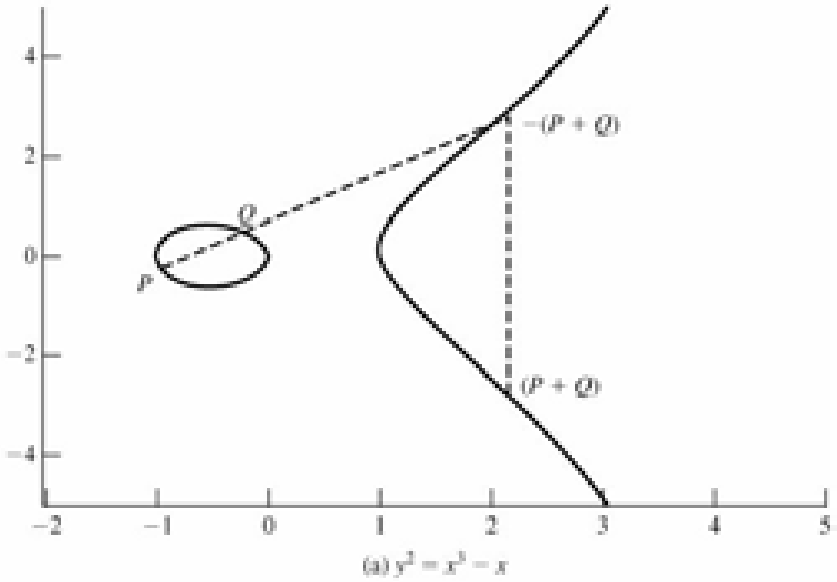
Such equations are said to be cubic, or of degree 3, because the highest exponent they contain is a 3. Also included in the definition of an elliptic curve is a single element denoted O and called the point at infinity or the *zero point*, which we discuss subsequently. To plot such a curve, we need to compute

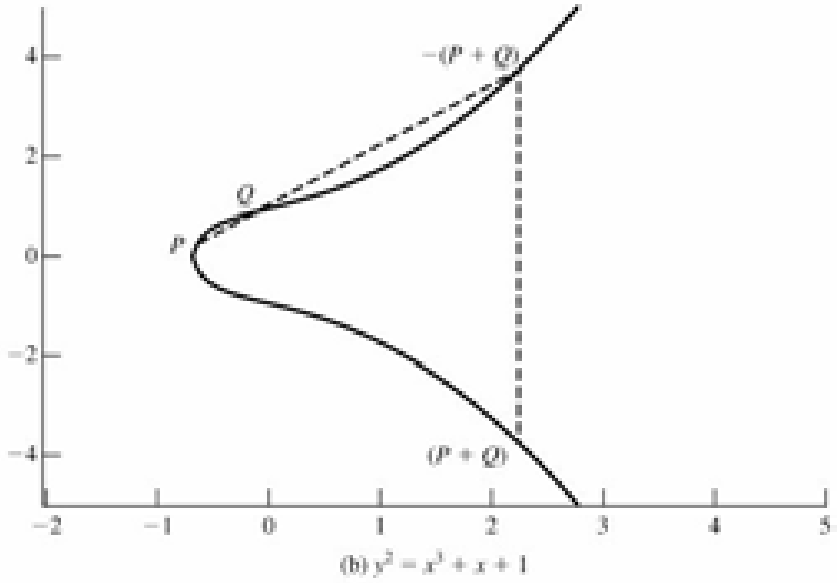
$$y = \sqrt{x^3 + ax + b}$$

For given values of a and b, the plot consists of positive and negative values of y for each value of x. Thus each curve is symmetric about y = 0. Figure 10.9 shows two examples of elliptic curves. As you can see, the formula sometimes produces weird-looking curves.

Figure 10.9. Example of Elliptic Curves

(This item is displayed on page 304 in the print version)





Now, consider the set of points E(a, b) consisting of all of the points (x, y) that satisfy Equation (10.1) together with the element O. Using a different value of the pair (a, b) results in a different set E(a, b). Using this

terminology, the two curves in Figure 10.9 depict the sets E(1,0) and E(1, 1), respectively.

Geometric Description of Addition

It can be shown that a group can be defined based on the set E(a, b) for specific values of a and b in Equation (10.1), provided the following condition is met:

Equation 10-2

$$4a^3 + 27b^2 \neq 0$$

To define the group, we must define an operation, called addition and denoted by +, for the set E(a, b), where a and b satisfy Equation (10.2). In geometric terms, the rules for addition can be stated as follows: If three points on an elliptic curve lie on a straight line, their sum is O. From this definition, we can define the rules of addition over an elliptic curve:

1. O serves as the additive identity. Thus O = O; for any point P on the elliptic curve, P + O = P. In what follows, we assume P ≠ O and Q ≠ O.
2. The negative of a point P is the point with the same x coordinate but the negative of the y coordinate; that is, if P = (x, y), then P = (x, y). Note that these two points can be joined by a vertical line. Note that P + (P) = P P = O.
3. To add two points P and Q with different x coordinates, draw a straight line between them and find the third point of intersection R. It is easily seen that there is a unique point R that is the point of intersection (unless the line is tangent to the curve at either P or Q, in which case we take R = P or R = Q, respectively). To form a group structure, we need to define addition on these three points as follows: P + Q = R. That is, we define P + Q to be the mirror image (with respect to the x axis) of the third point of intersection. Figure 10.9 illustrates this construction.
4. The geometric interpretation of the preceding item also applies to two points, P and P, with the same x coordinate. The points are joined by a vertical line, which can be viewed as also intersecting the curve at the

infinity point. We therefore have P + (P) = O, consistent with item (2).

5. To double a point Q, draw the tangent line and find the other point of intersection S. Then Q + Q = 2Q = S.

With the preceding list of rules, it can be shown that the set E(a, b) is an abelian group.

Algebraic Description of Addition

In this subsection we present some results that enable calculation of additions over elliptic curves. For two distinct points P = (x_P, y_P) and Q = (x_Q, y_P) that are not negatives of each other, the slope of the line l that joins them is Δ = (y_Q y_P). There is exactly one other point where l intersects the elliptic curve, and that is the negative of the sum of P and Q. After some algebraic manipulation, we can express the sum R = P + Q as follows:

Equation 10-3

$$x_R = \Delta^2 - x_P - x_Q$$
$$y_R = -y_P + \Delta(x_P - x_R)$$

We also need to be able to add a point to itself: P + P = 2P = R. When $y_P \neq 0$, the expressions are

Equation 10-4

$$x_R = \left(\frac{3x_P^2 + a}{2y_P}\right)^2 - 2x_P$$
$$y_R = \left(\frac{3x_P^2 + a}{2y_P}\right)(x_P - x_R) - y_P$$

Elliptic Curves over Z_p

Elliptic curve cryptography makes use of elliptic curves in which the variables and coefficients are all restricted to elements of a finite field. Two families of elliptic curves are used in cryptographic applications: prime curves over Z_p and binary curves over GF (2^m). For a prime curve over Z_p, we use a cubic equation in which the variables and coefficients all take on values in the set of integers from 0 through p 1 and in which calculations are performed modulo p. For a binary curve defined over GF(2^m), the variables and coefficients all take on values in GF(2^n) and in calculations are performed over GF(2^n). points out that prime curves are best for software applications, because the extended bit-fiddling operations needed by binary curves are not required; and that binary curves are best for hardware applications, where it takes remarkably few logic gates to create a powerful, fast cryptosystem. We examine these two families in this section and the next.

There is no obvious geometric interpretation of elliptic curve arithmetic over finite fields. The algebraic interpretation used for elliptic curve arithmetic over real numbers does readily carry over, and this is the approach we take.

For elliptic curves over Z_p, as with real numbers, we limit ourselves to equations of the form of Equation (10.1), but in this case with coefficients and variables limited to Z_p:

Equation 10-5

$$y^2 \bmod p = (x^3 + ax + b) \bmod p$$

For example, Equation (10.5) is satisfied for a = 1, b = 1, x = 9, y = 9, y = 7, p = 23:

$7^2 \bmod 23 = (9^3 + 9 + 1) \bmod 23$

$49 \bmod 23 = 739 \bmod 23$

$3 = 3$

Now consider the set E_p (a, b) consisting of all pairs of integers (x, y) that satisfy Equation (10.5), together with a point at infinity O. The coefficients a and b and the variables x and y are all elements of Z_p.

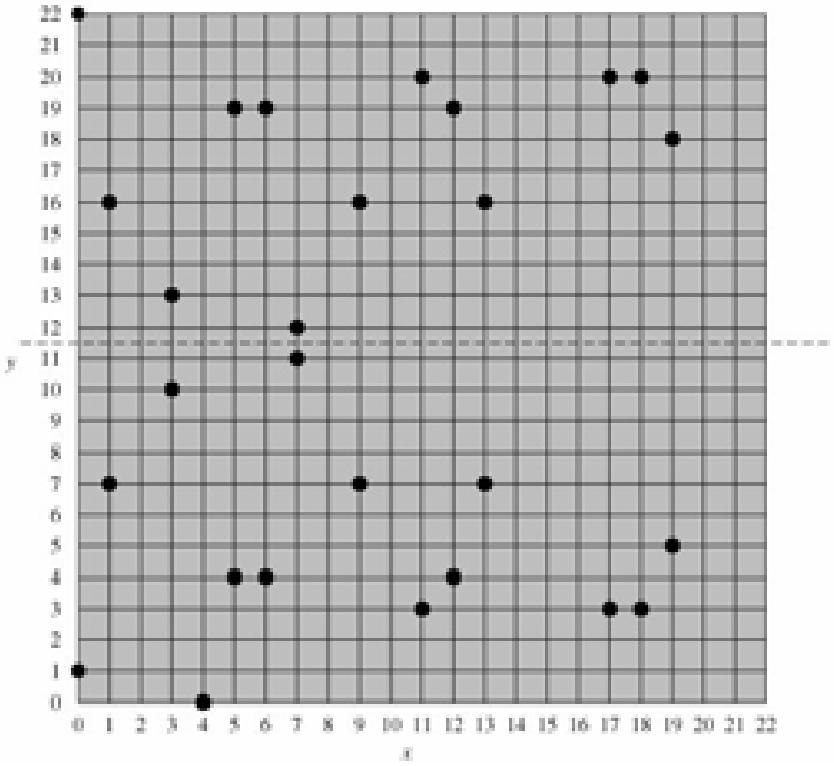
For example, let p = 23 and consider the elliptic curve $y^2 = x^3 + x + 1$. In this case, a = b = 1. Note that this equation is the same as that of Figure 10.9b. The figure shows a continuous curve with all of the real points that satisfy the equation. For the set $E_{23}(1, 1)$, we are only interested in the nonnegative integers in the quadrant from (0, 0) through (p 1, p 1) that satisfy the equation mod p. Table 10.1 lists the points (other than O) that are part of $E_{23}(1,1)$. Figure 10.10 plots the points of $E_{23}(1,1)$; note that the points, with one exception, are symmetric about y = 11.5.

Table 10.1. Points on the Elliptic Curve $E_{23}(1,1)$

(0, 1)	(6, 4)	(12, 19)
(0, 22)	(6, 19)	(13, 7)
(1, 7)	(7, 11)	(13, 16)
(1, 16)	(7, 12)	(17, 3)
(3, 10)	(9, 7)	(17, 20)
(3, 13)	(9, 16)	(18, 3)
(4, 0)	(11, 3)	(18, 20)
(5, 4)	(11, 20)	(19, 5)
(5, 19)	(12, 4)	(19, 18)

Figure 10.10. The Elliptic Curve E_{23} (1, 1)

(This item is displayed on page 307 in the print version)



It can be shown that a finite abelian group can be defined based on the set $E_p(a, b)$ provided that $(x^3 + ax + b) \bmod p$ has no repeated factors. This is equivalent to the condition

Equation 10-6

$$(4a^3 + 27b^2) \bmod p \neq 0 \bmod p$$

Note that Equation (10.6) has the same form as Equation (10.2).

The rules for addition over $E_p(a, b)$ correspond to the algebraic technique described for elliptic curves defined over real number. For all points P, Q

Ep(a, b);

1. P + O = P.

   If P = (x_P, y_P) then P + (x_P, y_P) = O. The point (x_P, y_P) is the negative of P, denoted as P. For example, in E_{23} (1, 1), for P = (13, 7), we have P = (13, 7). But 7 mod 23 = 16. Therefore, P = (13, 16), which is also in $E_{23}(1,1)$.

2. If P = (x_P, y_Q) and Q = (x_Q, y_Q) with P ≠Q, then R = P + Q = (x_R, y_R) is determined by the following rules:

   $x_R = (\lambda^2 x_P x_Q) \bmod p$

   $y_R = (\lambda (x_P x_R) y_P) \bmod p$

   where

$$\lambda = \begin{cases} \left(\frac{y_Q - y_P}{x_Q - x_P}\right) \bmod p & \text{if } P \neq Q \\ \left(\frac{3x_P^2 + a}{2y_P}\right) \bmod p & \text{if } P = Q \end{cases}$$

3. Multiplication is defined as repeated addition; for example, 4P = P + P + P + P.

   For example, let P = (3,10) and Q = (9,7) in $E_{23}(1,1)$. Then

$$\lambda = \left(\frac{7 - 10}{9 - 3}\right) \bmod 23 = \left(\frac{-3}{6}\right) \bmod 23 = \left(\frac{-1}{2}\right) \bmod 23 = 11$$

   $x_R = (11^2 3 9) \bmod 23 = 17$

   $y_R = (11(3 17) 10) \bmod 23 = 164 \bmod 23 = 20$

So P + Q = (17, 20). To find 2P,

$$\lambda = \left(\frac{3(3^2) + 1}{2 \times 10}\right) \bmod 23 = \left(\frac{5}{20}\right) \bmod 23 = \left(\frac{1}{4}\right) \bmod 23 = 6$$

The last step in the preceding equation involves taking the multiplicative inverse of 4 in Z_{23}. To confirm, note that (6 x 4) mod 23 = 24 mod 23 = 1.

$x_R = (6^2\ 3\ 3) \bmod 23 = 30 \bmod 23 = 7$

$y_R = (6(3\ 7)\ 10) \bmod 23 = (\ 34) \bmod 23 = 12$

and 2P = (7, 12).

For determining the security of various elliptic curve ciphers, it is of some interest to know the number the number of points in a finite abelian group defined over an elliptic curve. In the case of the finite group $E_p(a,b)$, the number of points N is bounded by

$$p + 1 - 2\sqrt{p} \le N \le p + 1 + 2\sqrt{p}$$

Note that the number of points in $E_p(a, b)$ is approximately equal to the number of elements in Zp, namely p elements.

Elliptic Curves over $GF(2^m)$

a finite field $GF(2^m)$ consists of 2^m elements, together with addition and multiplication operations that can be defined over polynomials. For elliptic curves over $GF(2^m)$, we use a cubic equation in which the variables and coefficients all take on values in $GF(2^m)$, for some number m, and in which calculations are performed using the rules of arithmetic in $GF(2^m)$.

It turns out that the form of cubic equation appropriate for cryptographic applications for elliptic curves is somewhat different for $GF(2^m)$ than for Z_p. The form is

Equation 10-7

$$y^2 + xy = x^3 + ax^2 + b$$

where it is understood that the variables x and y and the coefficients a and b are elements of GF(2^m) of and that calculations are performed in GF(2^m).

Now consider the set $E_2^m(a, b)$ consisting of all pairs of integers (x, y) that satisfy Equation (10.7), together with a point at infinity O.

For example, let us use the finite field GF(2^4) with the irreducible polynomial $f(x) = x^4 + x + 1$. This yields a generator that satisfies f(g) = 0, with a value of $g^4 = g + 1$, or in binary 0010. We can develop the powers of g as follows:

$g^0 = 0001$	$g^4 = 0011$	$g^8 = 0101$	$g^{12} = 1111$
$g^1 = 0010$	$g^5 = 0110$	$g^9 = 1010$	$g^{13} = 1101$
$g^2 = 0100$	$g^6 = 1100$	$g^{10} = 0111$	$g^{14} = 1001$
$g^3 = 1000$	$g^7 = 1011$	$g^{11} = 1110$	$g^{15} = 0001$

For example, $g^5 = (g^4)(g) = g^2 + g = 0110$.

Now consider the elliptic curve $y^2 + xy = x^3 + g^4x^2 + 1$. In this case $a = g^4$ and $b = g^0 = 1$. One point that satisfies this equation is (g^5, g^3):

$(g^3)^2 + (g^5)(g^3) = (g^5)^3 + (g^4)(g^5)^2 + 1$

$g^6 + g^8 = g^{15} + g^{14} + 1$

$1100 + 0101 = 0001 + 1001 + 0001$

$1001 = 1001$

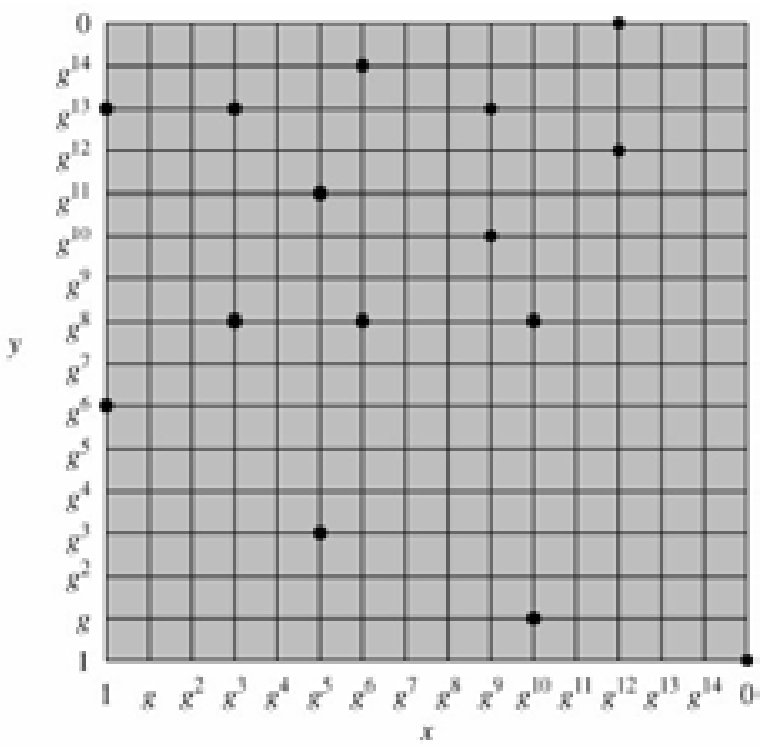
Table 10.2 lists the points (other than O) that are part of $E_2^4(g^4, 1)$. Figure 10.11 plots the points of $E_2^4(g^4, 1)$.

Table 10.2. Points on the Elliptic Curve E_2^4 $(g^4, 1)$

(0, 1)	(g^5, g^3)	(g^9, g^{13})
$(1, g^6)$	(g^5, g^{11})	(g^{10}, g)
$(1, g^{13})$	$g^6, g^8)$	(g^{10}, g^8)
(g^3, g^8)	(g^6, g^{14})	$(g^{12},0)$

Table 10.2. Points on the Elliptic Curve E_2^4 (g^4, 1)		
(0, 1)	(g^5, g^3)	(g^9, g^{13})
(g^3, g^{13})	(g^9, g^{10})	(g^{12}, g^{12})

Figure 10.11. The Elliptic Curve E_2^4(g^4, 1)



It can be shown that a finite abelian group can be defined based on the set $E_{2m}(a, b)$, provided that $b \neq 0$. The rules for addition can be stated as follows. For all points P, $Q \neq E_2^m(a, b)$:

1. P + O = P.

   If $P = (x_P, y_P)$, then $P + (x_P, x_p + y_P) = O$. The point $(x_P, x_P + y_P)$ is the negative of P, denoted as P.

2. If $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$ with $P \neq Q$ and $P \neq Q$, then $R = P + Q = (x_R, y_R)$ is determined by the following rules:

   $x_R = \lambda^2 + \lambda + x_P + x_Q + a$

   $y_R = \lambda (x_P + x_R) + x_R + y_P$

   where

   $$\lambda = \frac{y_Q + y_P}{x_Q + x_P}$$

3. If $= (x_P, y_P)$ then $R = 2P = (x_R, y_R)$ is determined by the following rules:

   $$x_R = \lambda^2 + \lambda + a$$
   $$y_R = x_P^2 + (\lambda + 1)x_R$$

   where

   $$\lambda = x_P + \frac{y_P}{x_P}$$

Elliptic Curve Cryptography

The addition operation in ECC is the counterpart of modular multiplication in RSA, and multiple additions are the counterpart of modular exponentiation. To form a cryptographic system using elliptic curves, we need to find a "hard problem" corresponding to factoring the product of two primes or taking the discrete logarithm.

Consider the equation Q = kP where Q, P $E_p(a, b)$ and k < p. It is relatively easy to calculate Q given k and P, but it is relatively hard to determine k given Q and P. This is called the discrete logarithm problem for elliptic curves.

Consider the group $E_{23}(9, 17)$. This is the group defined by the equation $y^2 \bmod 23 = (x^3 + 9x + 17) \bmod 23$. What is the discrete logarithm k of Q = (4, 5) to the base P = (16.5)? The brute-force method is to compute multiples of P until Q is found.

Thus

P = (16, 5); 2P = (20, 20); 3P = (14, 14); 4P = (19, 20); 5P = (13, 10); 6P = (7, 3); 7P = (8, 7); 8P (12, 17); 9P = (4, 5).

Because 9P = (4, 5) = Q, the discrete logarithm Q = (4, 5) to the base P = (16, 5) is k = 9. In a real application, k would be so large as to make the brute-force approach infeasible.

In the remainder of this section, we show two approaches to ECC that give the flavor of this technique.

Analog of Diffie-Hellman Key Exchange

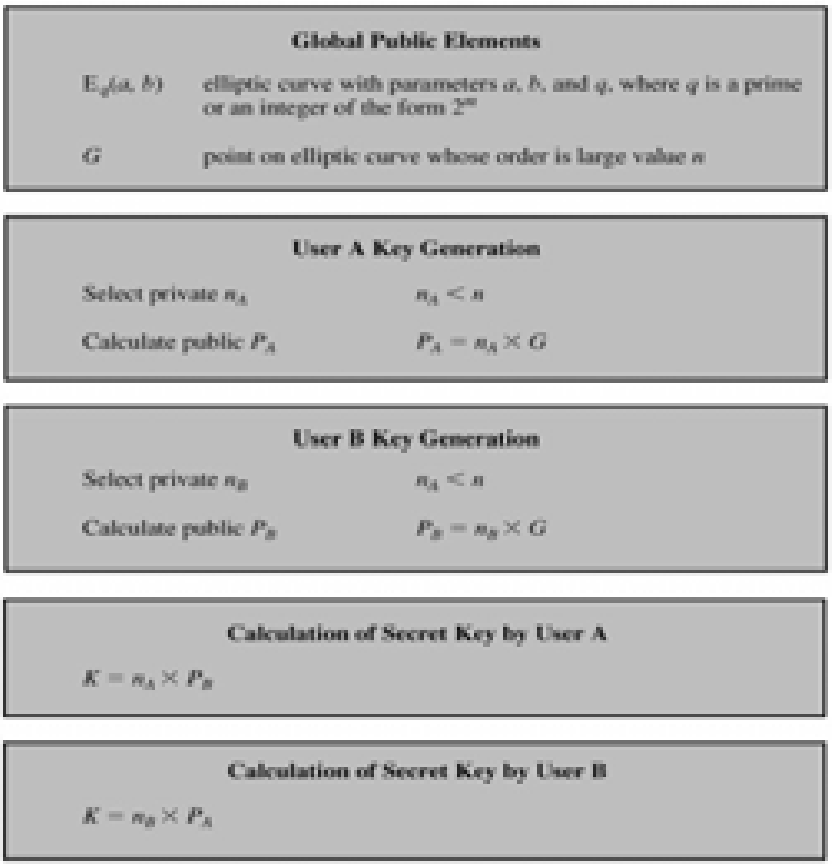
Key exchange using elliptic curves can be done in the following manner. First pick a large integer q, which is either a prime number p or an integer of the form 2^m and elliptic curve parameters a and b for Equation (10.5) or Equation (10.7). This defines the elliptic group of points $E_q(a, b)$. Next, pick a base point $G = (x_1, y_1)$ in $E_p(a, b)$ whose order is a very large value n. The order n of a point G on an elliptic curve is the smallest positive integer n

such that nG = O. $E_q(a, b)$ and G are parameters of the cryptosystem known to all participants.

A key exchange between users A and B can be accomplished as follows (Figure 10.12):

1. A selects an integer n_A less than n. This is A's private key. A then generates a public key $P_A = n_A \times G$; the public key is a point in Eq(a, b).
2. B similarly selects a private key n_B and computes a public key P_B.
3. A generates the secret key $K = n_A \times P_B$. B generates the secret key $K = n_B \times P_A$.

Figure 10.12. ECC Diffie-Hellman Key Exchange



The two calculations in step 3 produce the same result because

$n_A \times P_B = n_A \times (n_B \times G) = n_B \times (n_A \times G) = n_B \times P_A$

To break this scheme, an attacker would need to be able to compute k given G and kG, which is assumed hard.

As an example, take p = 211; Ep(0, 4), which is equivalent to the curve $y^2 = x^3$ 4; and G = (2, 2). One can calculate that 240G = O. A's private key is n_A = 121, so A's public key is P_A = 121(2, 2) = (115, 48). B's private key is n_B = 203, so B's public key is 203(2, 2) = (130, 203). The shared secret key is 121(130, 203) = 203(115, 48) = (161, 69).

Note that the secret key is a pair of numbers. If this key is to be used as a session key for conventional encryption, then a single number must be generated. We could simply use the x coordinates or some simple function of the x coordinate.

Elliptic Curve Encryption/Decryption

Several approaches to encryption/decryption using elliptic curves have been analyzed in the literature. In this subsection we look at perhaps the simplest. The first task in this system is to encode the plaintext message m to be sent as an x-y point P_m. It is the point P_m that will be encrypted as a ciphertext and subsequently decrypted. Note that we cannot simply encode the message as the x or y coordinate of a point, because not all such coordinates are in Eq(a, b); for example, see Table 10.1. Again, there are several approaches to this encoding, which we will not address here, but suffice it to say that there are relatively straightforward techniques that can be used.

As with the key exchange system, an encryption/decryption system requires a point G and an elliptic group Eq(a, b) as parameters. Each user A selects a private key n_A and generates a public key $P_A = n_A \times G$.

To encrypt and send a message P_m to B, A chooses a random positive integer k and produces the ciphertext C_m consisting of the pair of points:

$C_m = \{kG, P_m + kP_B\}$

Note that A has used B's public key P_B. To decrypt the ciphertext, B multiplies the first point in the pair by B's secret key and subtracts the result from the second point:

$$P_m + kP_B\ n_B(kG) = P_m + k(n_BG)\ n_B(kG) = P_m$$

A has masked the message P_m by adding kP_B to it. Nobody but A knows the value of k, so even though P_B is a public key, nobody can remove the mask kP_B. However, A also includes a "clue," which is enough to remove the mask if one knows the private key n_B. For an attacker to recover the message, the attacker would have to compute k given G and kG, which is assumed hard.

As an example of the encryption process (taken from [KOBL94]), take p = 751; $E_p(1, 188)$, which is equivalent to the curve $y^2 = x^3 x + 188$; and G = (0, 376). Suppose that A wishes to send a message to B that is encoded in the elliptic point P_m = (562, 201) and that A selects the random number k = 386. B's public key is P_B = (201, 5). We have 386(0, 376) = (676, 558), and (562, 201) + 386(201, 5) = (385, 328). Thus A sends the cipher text {(676, 558), (385, 328)}.

Security of Elliptic Curve Cryptography

The security of ECC depends on how difficult it is to determine k given kP and P. This is referred to as the elliptic curve logarithm problem. The fastest known technique for taking the elliptic curve logarithm is known as the Pollard rho method. Table 10.3 compares various algorithms by showing comparable key sizes in terms of computational effort for cryptanalysis. As can be seen, a considerably smaller key size can be used for ECC compared to RSA. Furthermore, for equal key lengths, the computational effort required for ECC and RSA is comparable . Thus, there is a computational advantage to using ECC with a shorter key length than a comparably secure RSA.

Table 10.3. Comparable Key Sizes in Terms of Computational Effort for Cryptanalysis

Symmetric Scheme (key size in bits)	ECC-Based Scheme (size of n in bits)	RSA/DSA (modulus size in bits)
56	112	512

Table 10.3. Comparable Key Sizes in Terms of Computational Effort for Cryptanalysis

Symmetric Scheme (key size in bits)	ECC-Based Scheme (size of n in bits)	RSA/DSA (modulus size in bits)
80	160	1024
112	224	2048
128	256	3072
92	384	7680
256	512	15360

Source: Certicom