APPENDIX P

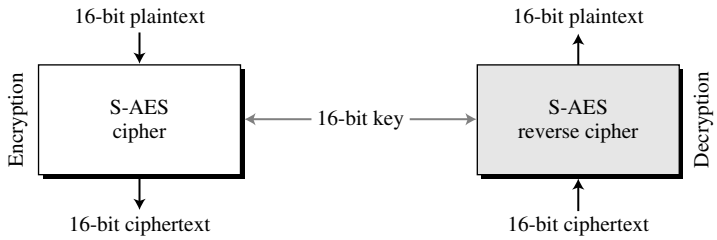
Simplified AES (S-AES)

Simplified AES (S-AES), developed by Professor Edward Schaefer of Santa Clara University, is an educational tool designed to help students learn the structure of AES using smaller blocks and keys. Readers may choose to study this appendix before reading Chapter 7.

P.1 S-AES STRUCTURE

S-AES is a block cipher, as shown in Figure P.1.

Figure P.1 *Encryption and decryption with S-AES*

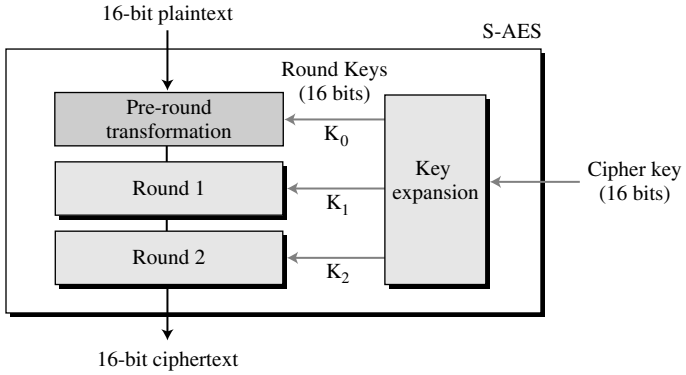


At the encryption site, S-AES takes a 16-bit plaintext and creates a 16-bit ciphertext; at the decryption site, S-AES takes a 16-bit ciphertext and creates a 16-bit plaintext. The same 16-bit cipher key is used for both encryption and decryption.

Rounds

S-AES is a non-Feistel cipher that encrypts and decrypts a data block of 16 bits. It uses one pre-round transformation and two rounds. The cipher key is also 16 bits. Figure P.2

Figure P.2 *General design of S-AES encryption cipher*



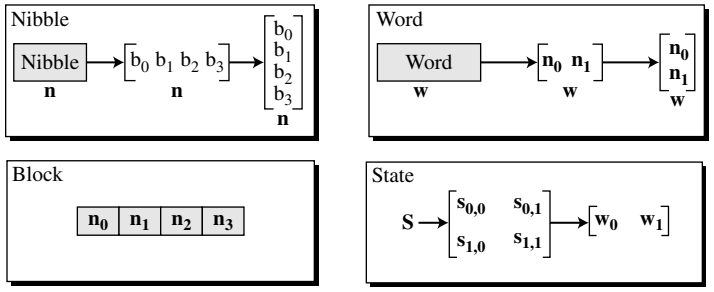
shows the general design for the encryption algorithm (called the cipher); the decryption algorithm (called the inverse cipher) is similar, but the round keys are applied in the reverse order.

In Figure P.2, the round keys, which are created by the key-expansion algorithm, are always 16 bits, the same size as the plaintext or ciphertext block. In S-AES, there are three round keys, K_0, K_1, and K_2.

Data Units

S-AES uses five units of measurement to refer to data: bits, nibbles, words, blocks, and states, as shown in Figure P.3.

Figure P.3 *Data units used in S-AES*



Bit

In S-AES, a *bit* is a binary digit with a value of 0 or 1. We use a lowercase letter b to refer to a bit.

Nibble

A **nibble** is a group of 4 bits that can be treated as a single entity, a row matrix of 4 bits, or a column matrix of 4 bits. When treated as a row matrix, the bits are inserted into the matrix from left to right; when treated as a column matrix, the bits are inserted into the matrix from top to bottom. We use a lowercase bold letter **n** to refer to a nibble. Note that a nibble is actually a single hexadecimal digit.

Word

A *word* is a group of 8 bits that can be treated as a single entity, a row matrix of two nibbles, or a column matrix of 2 nibbles. When it is treated as a row matrix, the nibbles are inserted into the matrix from left to right; when it is considered as a column matrix, the nibbles are inserted into the matrix from top to bottom. We use the lowercase bold letter **w** to refer to a word.

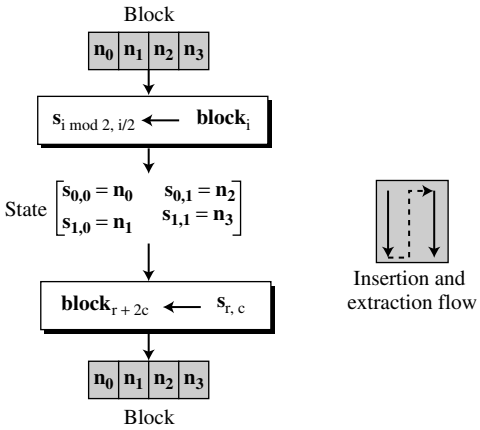
Block

S-AES encrypts and decrypts data blocks. A *block* in S-AES is a group of 16 bits. However, a block can be represented as a row matrix of 4 nibbles.

State

In S-AES, a data block is also referred to as a *state*. We use an uppercase bold letter **S** to refer to a state. States, like blocks, are made of 16 bits, but normally they are treated as matrices of 4 nibbles. In this case, each element of a state is referred to as $\mathbf{s}_{r,c}$, where r (0 to 1) defines the row and the c (0 to 1) defines the column. At the beginning of the cipher, nibbles in a data block are inserted into a state column by column, and in each column, from top to bottom. At the end of the cipher, nibbles in the state are extracted in the same way, as shown in Figure P.4.

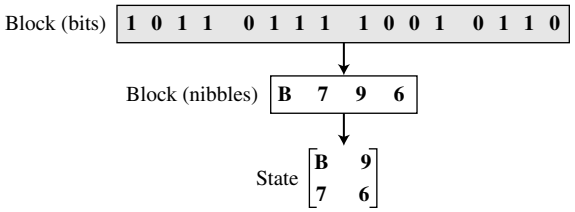
Figure P.4 *Block-to-state and state-to-block transformation*



Example P.1

Let us see how a 16-bit block can be shown as a 2×2 matrix. Assume that the text block is 1011 0111 1001 0110. We first show the block as 4 nibbles. The state matrix is then filled up, column by column, as shown in Figure P.5.

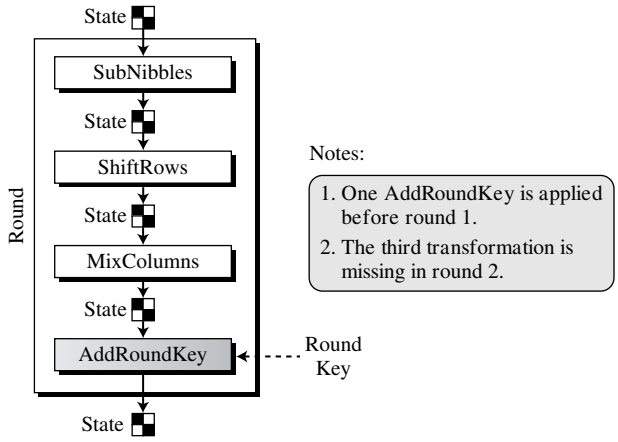
Figure P.5 *Changing ciphertext to a state*



Structure of Each Round

Figure P.6 shows that each transformation takes a state and creates another state to be used for the next transformation or the next round. The pre-round section uses only one transformation (AddRoundKey); the last round uses only three transformations, (MixColumns transformation is missing).

Figure P.6 *Structure of each round at the encryption site*



At the decryption site, the inverse transformations are used: InvSubNibbles, InvShiftRows, InvMixColumns, and AddRoundKey (this one is self-invertible).

P.2 TRANSFORMATIONS

To provide security, S-AES uses four types of transformations: substitution, permutation, mixing, and key-adding. We will discuss each here.

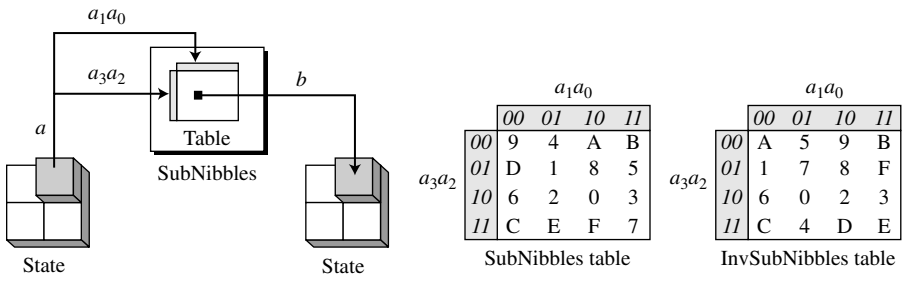
Substitution

Substitution is done for each nibble (4-bit data unit). Only one table is used for transformations of every nibble, which means that if two nibbles are the same, the transformation is also the same. In this appendix, transformation is defined by a table lookup process.

SubNibbles

The first transformation, **SubNibbles,** is used at the encryption site. To substitute a nibble, we interpret the nibble as 4 bits. The left 2 bits define the row and the right 2 bits define the column of the substitution table. The hexadecimal digit at the junction of the row and the column is the new nibble. Figure P.7 shows the idea.

Figure P.7 *SubNibbles transformations*



In the SubNibbles transformation, the state is treated as a 2×2 matrix of nibbles. Transformation is done one nibble at a time. The contents of each nibble is changed, but the arrangement of the nibbles in the matrix remains the same. In the process, each nibble is transformed independently: There are four distinct nibble-to-nibble transformations.

SubNibbles involves four independent nibble-to-nibble transformations.

Figure P.7 also shows the substitution table (S-box) for the SubNibbles transformation. The transformation definitely provides confusion effect. For example, two nibbles, A_{16} and B_{16}, which differ only in one bit (the rightmost bit), are transformed to 0_{16} and 3_{16}, which differ in two bits.

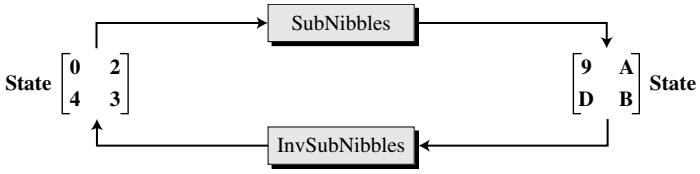
InvSubNibbles

InvSubNibbles is the inverse of SubNibbles. The inverse transformation is also shown in Figure P.7. We can easily check that the two transformations are inverses of each other.

Example P.2

Figure P.8 shows how a state is transformed using the SubNibbles transformation. The figure also shows that the InvSubNibbles transformation creates the original state. Note that if the two nibbles have the same values, their transformation are also the same. The reason is that every nibble uses the same table.

Figure P.8 *SubNibble transformation for Example P.2*



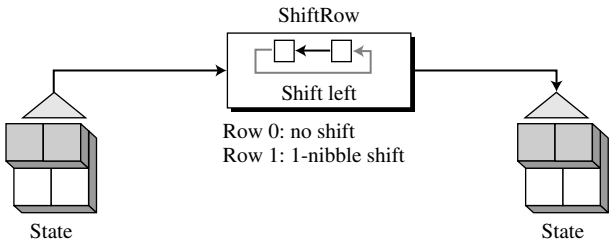
Permutation

Another transformation found in a round is shifting, which permutes the nibbles. Shifting transformation in S-AES is done at the nibble level; the order of the bits in the nibble is not changed.

ShiftRows

In the encryption, the transformation is called *ShiftRows* and the shifting is to the left. The number of shifts depends on the row number (0, 1) of the state matrix. This means row 0 is not shifted at all and row 1 is shifted 1 nibble. Figure P.9 shows the shifting transformation. Note that the ShiftRows transformation operates one row at a time.

Figure P.9 *ShiftRows transformation*



InvShiftRows

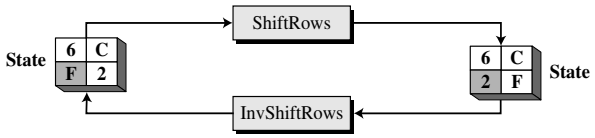
In the decryption, the transformation is called *InvShiftRows* and the shifting is to the right. The number of shifts is the same as the number of the row (0, 1) in the state matrix.

The ShiftRows and InvShiftRows transformations are inverses of each other.

Example P.3

Figure P.10 shows how a state is transformed using ShiftRows. The figure also shows that the InvShiftRows transformation creates the original state.

Figure P.10 *ShiftRows transformation in Example P.3*



Mixing

The substitution provided by the SubNibbles transformation changes the value of the nibble based only on the nibble's original value and an entry in the table; the process does not include the neighboring nibbles. We can say that SubNibbles is an *intra-nibble* transformation. The permutation provided by the ShiftRows transformation exchanges nibbles without permuting the bits inside the bytes. We can say that ShiftRows is a *nibble-exchange* transformation. We also need an *inter-nibble* transformation that changes the bits inside a nibble, based on the bits inside the neighboring nibbles. We need to mix nibbles to provide diffusion at the bit level.

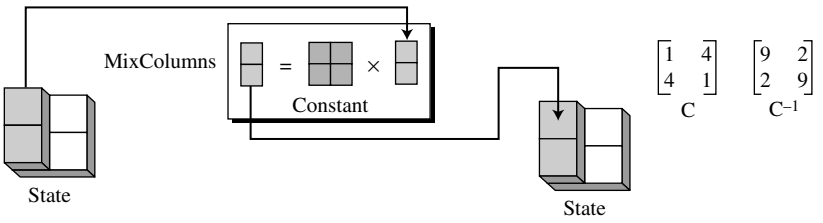
The mixing transformation changes the contents of each nibble by taking 2 nibbles at a time and combining them to create 2 new nibbles. To guarantee that each new nibble is different (even if the old nibbles are the same), the combination process first multiplies each nibble with a different constant and then mixes them. The mixing can be provided by matrix multiplication. As we discussed in Chapter 2, when we multiply a square matrix by a column matrix, the result is a new column matrix. Each element in the new matrix depends on the two elements of the old matrix after they are multiplied by row values in the constant matrix.

MixColumns

The *MixColumns* transformation operates at the column level; it transforms each column of the state into a new column. The transformation is actually the matrix multiplication of a state column by a constant square matrix. The nibbles in the state column and constants matrix are interpreted as 4-bit words (or polynomials) with coefficients in

GF(2). Multiplication of bytes is done in GF(2^4) with modulus ($x^4 + x + 1$) or (10011). Addition is the same as XORing of 4-bit words. Figure P.11 shows the MixColumns transformation.

Figure P.11 *MixColumns transformation*



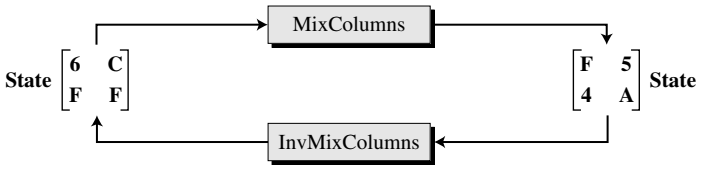
InvMixColumns

The *InvMixColumns* transformation is basically the same as the MixColumns transformation. If the two constant matrices are inverses of each other, it is easy to prove that the two transformations are inverses of each other.

The MixColumns and InvMixColumns transformations are inverses of each other.

Figure P.12 shows how a state is transformed using the MixColumns transformation. The figure also shows that the InvMixColumns transformation creates the original one.

Figure P.12 *The MixColumns transformation in Example 7.5*



Note that equal bytes in the old state, are not equal any more in the new state. For example, the two bytes F in the second row are changed to 4 and A.

Key Adding

Probably the most important transformation is the one that includes the cipher key. All previous transformations use known algorithms that are invertible. If the cipher

key is not added to the state at each round, it is very easy for the adversary to find the plaintext, given the ciphertext. The cipher key is the only secret between Alice and Bob in this case.

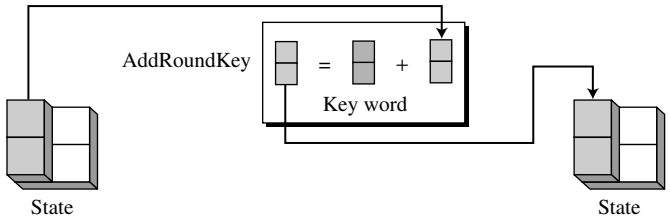
S-AES uses a process called key expansion (discussed later in this appendix) that creates three round keys from the cipher key. Each round key is 16 bits long—it is treated as two 8-bit words. For the purpose of adding the key to the state, each word is considered as a column matrix.

AddRoundKey

AddRoundKey also proceeds one column at a time. It is similar to MixColumns in this respect. MixColumns multiplies a constant square matrix by each state column; AddRoundKey adds a round key word with each state column matrix. The operations in MixColumns are matrix multiplication; the operations in AddRoundKey are matrix addition. The addition is performed in the GF(2^4) field. Because addition and subtraction in this field are the same, the AddRoundKey transformation is the inverse of itself. Figure P.13 shows the AddRoundKey transformation.

The AddRoundKey transformation is the inverse of itself.

Figure P.13 *AddRoundKey transformation*



P.3 KEY EXPANSION

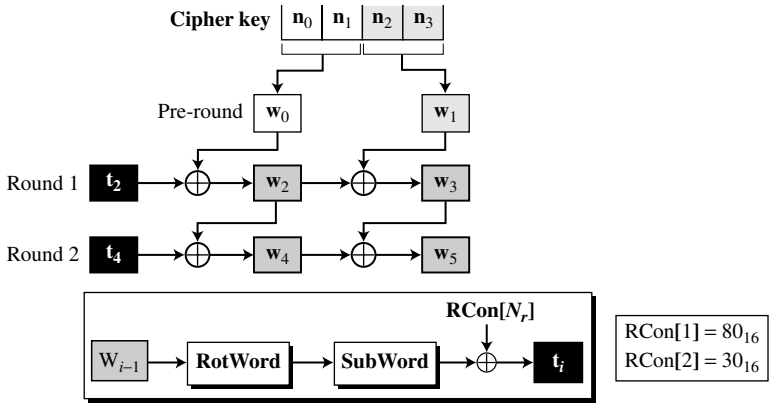
The *key expansion* routine creates three 16-bit round keys from one single 16-bit cipher key. The first round key is used for pre-round transformation (AddRoundKey); the remaining round keys are used for the last transformation (AddRoundKey) at the end of round 1 and round 2.

The key-expansion routine creates round keys word by word, where a word is an array of 2 nibbles. The routine creates 6 words, which are called $\mathbf{w}_0$, $\mathbf{w}_1$, $\mathbf{w}_2$, …, $\mathbf{w}_5$.

Creation of Words in S-AES

Figure P.14 shows how 6 words are made from the original key.

Figure P.14 *Creation of words in S-AES*



Making of $\mathbf{t}_i$ (temporary) words $i = 2N_r$, where N_r is the round number

The process is as follows:

1. The first two words ($\mathbf{w}_0$, $\mathbf{w}_1$) are made from the cipher key. The cipher key is thought of as an array of 4 nibbles ($\mathbf{n}_0$ to $\mathbf{n}_3$). The first 2 nibbles ($\mathbf{n}_0$ to $\mathbf{n}_1$) become $\mathbf{w}_0$; the next 2 nibbles ($\mathbf{n}_2$ to $\mathbf{n}_3$) become $\mathbf{w}_1$. In other words, the concatenation of the words in this group replicates the cipher key.
2. The rest of the words ($\mathbf{w}_i$ for i = 2 to 5) are made as follows:
   a. If (i mod 2) = 0, $\mathbf{w}_i = \mathbf{t}_i \oplus \mathbf{w}_{i-2}$. Here $\mathbf{t}_i$, a temporary word, is the result of applying two routines, SubWord and RotWord, on $\mathbf{w}_{i-1}$ and XORing the result with a round constant, RC[N_r], where N_r is the round number. In other words, we have

$$\mathbf{t}_i = \text{SubWord (RotWord } (\mathbf{w}_{i-1})) \oplus \text{RCon } [N_r]$$

   The words $\mathbf{w}_2$ and $\mathbf{w}_4$ are made using this process.
   b. If (i mod 2) ≠ 0, $\mathbf{w}_i = \mathbf{w}_{i-1} \oplus \mathbf{w}_{i-2}$. Referring to Figure P.14, this means each word is made from the word at the left and the word at the top. The words $\mathbf{w}_3$ and $\mathbf{w}_5$ are made using this process.

RotWord

The *RotWord* (rotate word) routine is similar to the ShiftRows transformation, but it is applied to only one row. The routine takes a word as an array of 2 nibbles and shifts each nibble to the left with wrapping. In S-AES, this is actually swapping the 2 nibbles in the word.

SubWord

The *SubWord* (substitute word) routine is similar to the SubNibble transformation, but it is applied only to 2 nibbles. The routine takes each nibble in the word and substitutes another nibble for it using the SubNibble table in Figure P.7.

Round Constants

Each round constant, RC, is a 2-nibble value in which the rightmost nibble is always zero. Figure P.14 also shows the value of RCs.

Example P.4

Table P.1 shows how the keys for each round are calculated assuming that the 16-bit cipher key agreed upon by Alice and Bob is 2475_{16}.

Table P.1 *Key expansion example*

Round	*Values of t's*	*First word in the round*	*Second word in the round*	*Round Key*
0		$w_0 = 24$	$w_1 = 75$	$K_0 = 2475$
1	$t_2 = 95$	$w_2 = 95 \oplus 24 = B1$	$w_3 = B1 \oplus 75 = C4$	$K_0 = B1C4$
2	$t_4 = EC$	$w_4 = B1 \oplus EC = 5D$	$w_5 = 5D \oplus C4 = 99$	$K_2 = 5D99$

In each round, the calculation of the second word is very simple. For the calculation of the first word we need to first calculate the value of the temporary word ($\mathbf{t}_i$), as shown below:

RotWord (75) = 57 → **SubWord** (57) = 15 → $\mathbf{t_2} = 15 \oplus \mathbf{RC[1]} = 15 \oplus 80 = 95$

RotWord (C4) = 4C → **SubWord** (4C) = DC → $\mathbf{t_4} = DC \oplus \mathbf{RC[2]} = DC \oplus 30 = EC$

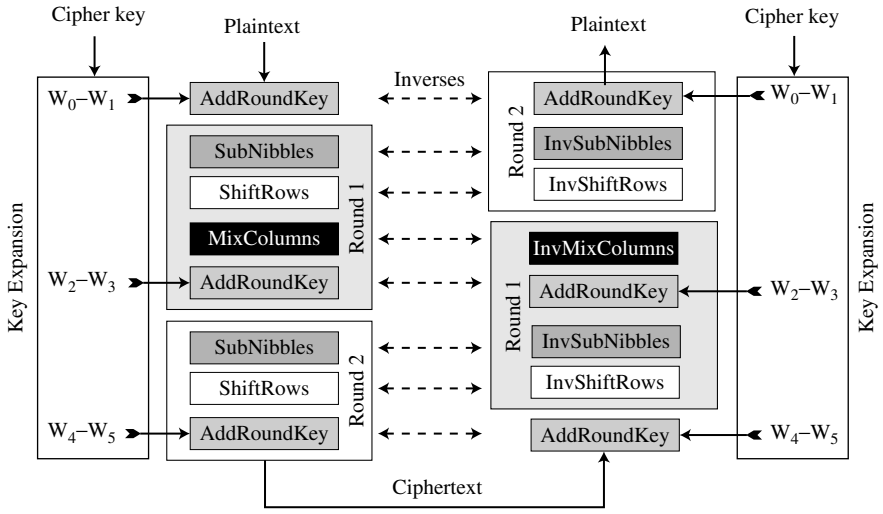
P.4 CIPHERS

Now let us see how S-AES uses the four types of transformations for encryption and decryption. The encryption algorithm is referred to as the *cipher* and the decryption algorithm as the *inverse cipher.*

S-AES is a non-Feistel cipher, which means that each transformation or group of transformations must be invertible. In addition, the cipher and the inverse cipher must use these operations in such a way that they cancel each other. The round keys must also be used in the reverse order. To comply with this requirement, the transformations occur in a different order in the cipher and the reverse cipher, as shown in Figure P.15.

First, the order of SubNibbles and ShiftRows is changed in the reverse cipher. Second, the order of MixColumns and AddRoundKey is changed in the reverse cipher. This difference in ordering is needed to make each transformation in the cipher aligned with its inverse in the reverse cipher. Consequently, the decryption algorithm as a whole is the inverse of the encryption algorithm. Note that the round keys are used in the reverse order.

Figure P.15 *Cipher and inverse cipher of the original design*



Example P.5

We choose a random plaintext block, the cipher key used in Example P.4, and determine what the ciphertext block would be:

Plaintext: $1A23_{16}$	Key: 2475_{16}	Ciphertext: $3AD2_{16}$

Figure P.16 shows the value of states in each round. We are using the round keys generated in Example P.4.

Figure P.16 *Example P.5*