AN1044

Data Encryption Routines for PIC24 and dsPIC® Devices

Authors: David Flowers and
Howard Henry Schlunder
Microchip Technology Inc.

INTRODUCTION

Currently, there are three data encryption standards approved for use in the Federal Information Processing Standards (FIPS). This application note discusses the implementation of two of these for PIC24 and dsPIC30/33 devices: Triple Data Encryption Standard (TDES) and Advanced Encryption Standard (AES).

TDES ENCRYPTION

Background

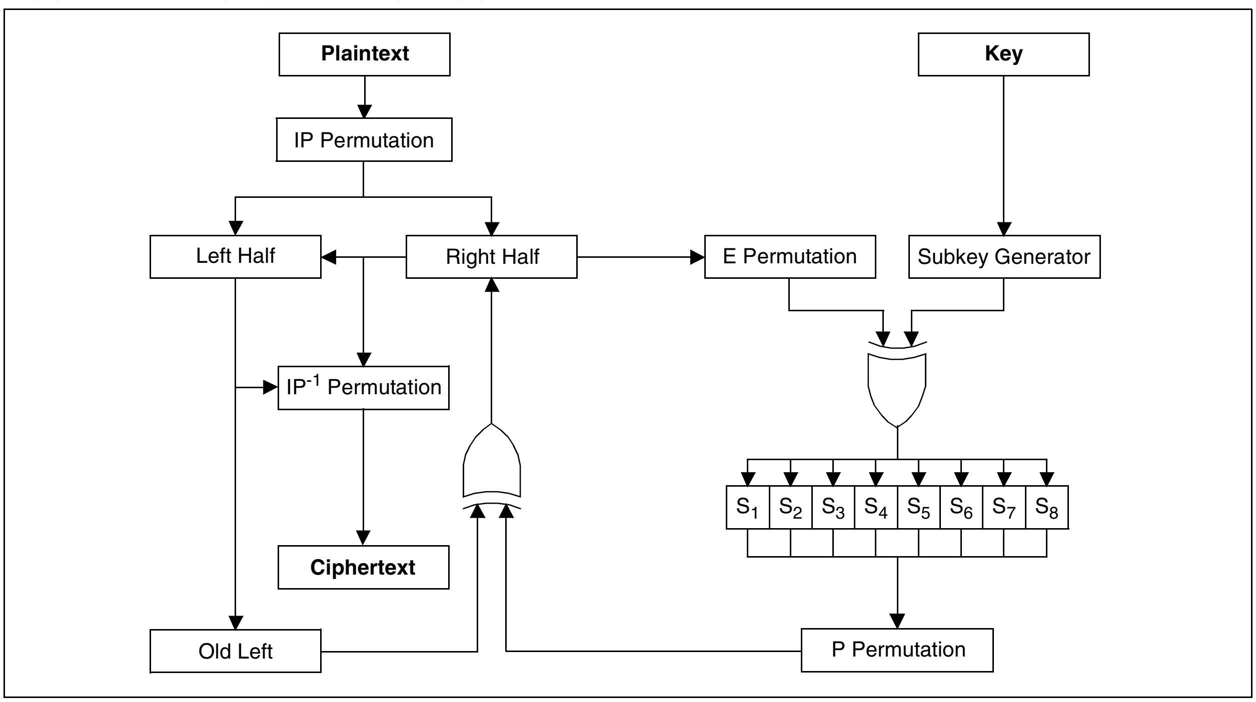
The original Data Encryption Standard (DES), a 64-bit block cipher, was invented in the early 1970s by IBM®. DES uses a 64-bit encryption key: 56 bits for encoding and decoding, the remainder for parity. It was adopted by the United States government in 1977 as standard for encrypting sensitive data. By the mid 1990s, several public organizations had demonstrated that they were able to crack a DES code within days.

Triple DES (TDES) is a variant of DES, and is described in FIPS 46-2 and 46-3. TDES uses three cycles of DES to extend the key from 56 bits to 112 or 168 bits, depending on the mode of operation. Because of known weaknesses in the DES algorithm, the actual security is believed to be on the order of 80 and 112 bits, respectively, for the two different methods. The use of TDES was suggested by the American government in 1999 for use in all systems, except in legacy systems, where only DES was available.

There are several different modes of TDES. The most common involves using two different keys. The data is encrypted with the first key. That result is then decrypted with the second key. The data is then finally encrypted once again with the first key. Other modes of operation include using three different keys, one for each of the stages, and encrypting in all rounds instead of decrypting during the second round. For most new applications, TDES has been replaced with Advanced Encryption Standard (AES). AES provides a slightly higher security level than TDES and is much faster and smaller in implementation than TDES.

The original DES algorithm is outlined in Figure 1. The cycle is run 32 times before the ciphertext is valid.

FIGURE 1: ORIGINAL DES ALGORITHM



In the original DES, the plaintext is permuted by the initial permutation matrix, IP (Figure 2). It is then split into a left portion and a right portion. The right portion is permuted by E (Figure 3), XORed with the round subkey, substituted with an S-Box value (Figure 6), permuted by P (Figure 4) and XORed with the left half of the data from the last round. The left data is replaced with the right data from the last round and the right data is replaced with this new calculated value. The cycle is repeated for 32 iterations, with the result permuted by the inverse permutation matrix, IP^{-1} (Figure 5), to get the final cipher text.

FIGURE 2: INITIAL PERMUTATION MATRIX (IP)

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

FIGURE 3: EXPANSION PERMUTATION MATRIX (E)

32	1	2	3	4	5	4	5
6	7	8	9	8	9	10	11
12	13	12	13	14	15	16	17
16	17	18	19	20	21	20	21
22	23	24	25	24	25	26	27
28	29	28	29	30	31	32	1

FIGURE 4: PERMUTATION BOX MATRIX (P)

16	7	20	21	29	12	28	17
1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9
19	13	30	6	22	11	4	25

FIGURE 5: INVERSE PERMUTATION (IP^{-1}) MATRIX

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	14	52	20	60	28
35	3	43	13	51	19	59	27
34	2	42	12	50	18	58	26
33	1	41	11	49	17	57	25

An optional implementation, shown in Figure 7, can be used to reduce the execution time required for each encryption. Because the S-Box substitution and P permutation are both linear operations, they can be combined into one operation, instead of two separate operations, thus resulting in a PS table. Unrolling the DES loop once removes the need for some temporary variables and reduces the overhead of shuffling data. It does, however, increase the code size.

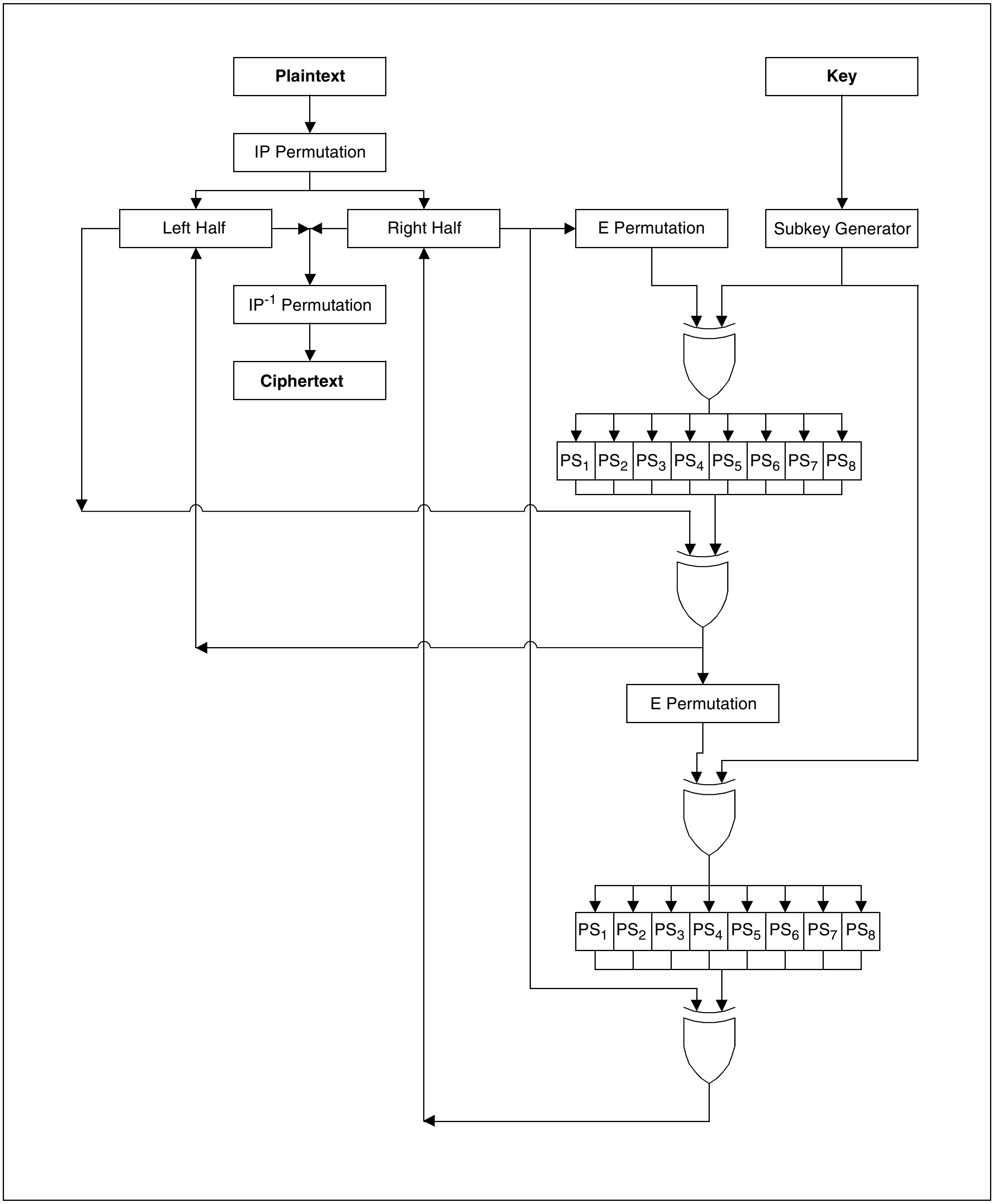
For a more detailed description of how the permutations and substitutions work, please refer to Microchip application note *AN583, "Implementation of the Data Encryption Standard Using PIC17C42"* (DS00583).

FIGURE 6: S-BOX MATRICES (S_n)

S_1 =	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
	4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	3
S_2 =	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
S_3 =	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
S_4 =	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

S_4 =	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
S_6 =	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
	9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
S_7 =	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
S_8 =	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

FIGURE 7: SPEED-OPTIMIZED DES ALGORITHM



Using the TDES Algorithm

This implementation of TDES is accessed through three function calls: `initTDES`, `TDES_encrypt` and `TDES_decrypt`. Their usage is discussed below.

initTDES

This function precalculates the subkey groups needed for TDES. By precalculating the subkeys, the encryption and decryption routines can be significantly enhanced for speed.

Syntax

```
void initTDES(unsigned int *KeyLocation);
```

Parameters

`KeyLocation`: word-aligned starting address in RAM where the calculated subkeys will be stored. This requires a 384-byte (192-word) block of memory.

Return Values

None

Pre-Conditions

`KeyLocation` is either reserved or allocated memory of 384 bytes (192 words).

`unsigned int Key[12]` is loaded with the Encryption/Decryption Keys, where `Key[0-3]` is the first DES key, `Key[4-7]` is the second key and `Key[8-11]` is the third key.

The same keys used to encrypt a block must also be used to decrypt it.

Side Effects

Values at reserved addresses are changed.

Example

```
...
unsigned int *KeyPointer;
KeyPointer = (unsigned int*)malloc(384);
if(KeyPointer != NULL)
{
initTDES(KeyPointer);
}
...
```


TDES_encrypt

This function uses a set of precalculated subkeys generated from `initDES` function and encrypts the data using TDES.

Syntax

```
void TDES_encrypt(unsigned int *KeyLocation);
```

Parameters

`KeyLocation`: pointer to the RAM where the subkeys are located.

Return Values

None

Pre-Conditions

`initTDES()` has been called resulting in a precalculated subkey

`unsigned int M[4]` is loaded with the data that will be encrypted

Side Effects

`unsigned int M[4]` will be translated to the ciphertext.

Example

```
...
TDES_encrypt(KeyPointer);
...
```

TDES_decrypt

This function uses a set of precalculated subkeys and decrypts the data using TDES.

Syntax

```
void TDES_decrypt(unsigned int *KeyLocation);
```

Parameters

`KeyLocation`: the address in RAM where the subkeys are located. The subkeys must be generated from the same key used to encrypt the data (refer to the `initTDES` function for details).

Return Values

None

Pre-Conditions

`initTDES()` has been called resulting in a precalculated subkey

`unsigned int M[4]` is loaded with the data that will be decrypted

Side Effects

`unsigned int M[4]` will be translated to the plaintext.

Example

```
...
TDES_decrypt(KeyPointer);
...
```

AES ENCRYPTION

Background

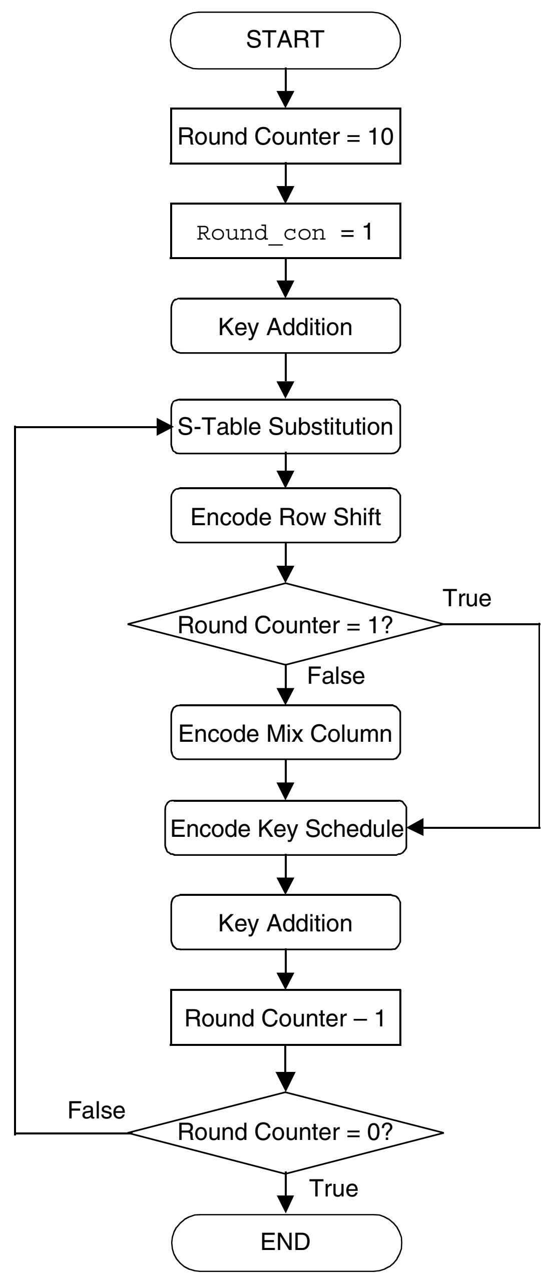
In the late 1990s, the National Institute of Standards and Technology (NIST) held a contest to initiate the development of encryption algorithms that would replace DES. The competition tested the algorithms' security and execution speed to determine which would be named the new Advanced Encryption Standard, or AES. The algorithm finally chosen is called the "Rijndael" algorithm after its two designers, Joan Daemen and Vincent Rijmen of Belgium. It was adopted by NIST on October 2, 2000, and is described in FIPS 197.

Rijndael/AES is a symmetric block cipher that utilizes a single key to encrypt data. The implementation of AES in this application note is based on a 16-byte block of data and a 16-byte key size.

Encryption

There are five basic subdivisions of the encryption algorithm, shown in Figure 8. A detailed explanation of each follows.

FIGURE 8: AES ENCRYPTION FLOWCHART



The number of rounds needed in the transformation is taken from Table 1. The implementation of AES discussed here uses 16-byte block and key sizes, and thus, uses 10 rounds of encryption.

TABLE 1: DETERMINING AES ROUNDS

Key Size	Rounds Needed for Block Size		
	16-Byte	**24-Byte**	**32-Byte**
16-byte	10*	12	14
24-byte	12	12	14
32-byte	14	14	14

***** Used in this implementation.

The structures of the key and data blocks are shown in Table 2 and Table 3. To fit into the data matrix structure, the plain text to be encrypted needs to be broken into the appropriate size blocks, with any leftover space being padded with an application specified value. Finally, a key must be selected that is 128 bits (16 bytes) long.

With a key selected and the data sectioned off into appropriate size blocks, the encryption cycle may begin.

TABLE 2: KEY MATRIX

Key [0]	Key [4]	Key [8]	Key [12]
Key [1]	Key [5]	Key [9]	Key [13]
Key [2]	Key [6]	Key [10]	Key [14]
Key [3]	Key [7]	Key [11]	Key [15]

TABLE 3: DATA MATRIX

Data [0]	Data [4]	Data [8]	Data [12]
Data [1]	Data [5]	Data [9]	Data [13]
Data [2]	Data [6]	Data [10]	Data [14]
Data [3]	Data [7]	Data [11]	Data [15]

KEY ADDITION

Once the key has been selected, each byte of the key is XORed with each of the corresponding data bytes. On subsequent rounds, the key generated by the key schedule for that round is XORed in a bytewise manner with the data.

S-TABLE SUBSTITUTION

During each round, each data byte is replaced with a corresponding byte from a fixed substitution table, or S-Table. A fixed S-Table defined by AES is shown in Table 4.

TABLE 4: S-TABLE ENCRYPTION SUBSTITUTION TABLE (VALUES IN HEXADECIMAL)

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

ENCODE ROW SHIFT

Row shift is a cyclical shift to the left of the rows in the data block. The values of each row are shifted differently, as shown in Table 5.

TABLE 5: ENCRYPTION CYCLICAL SHIFT

Before Row Shift:			
0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15
After Row Shift:			
0	4	8	12
5	9	13	1
10	14	2	6
15	3	7	11

ENCODE MIX COLUMN

Chapter 2, Section 4.2.3 of the AES specification (FIPS 197) defines the `mix column` transformation. In this operation, a fixed 4x4 matrix, $c(x)$, is cross-multiplied by the input vector $(a(x))$ using the special rules of Polynomials with coefficients in $GF(2^8)$ to form the output vector, $b(x)$, shown in Equation 1:

EQUATION 1:

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \times \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

FIXED MATRIX $c(x)$

The special rules for multiplication equate to the following:

$a \bullet 1 = a$

$a \bullet 2 = xtime(a)$

$a \bullet 3 = a \oplus xtime(a)$

$a \bullet 4 = xtime(xtime(a))$

$a \bullet 5 = a \oplus xtime(xtime(a))$

...

where `xtime` is a linear feedback shift procedure. It can be described in C as shown in Example 1:

EXAMPLE 1: `xtime` ROUTINE

```
if(a<0x80)
{
        a<<=1;
}
else
{
        a=(a<<1)^0x1b;
}
```

As an example, the first row of the resulting multiplication is shown in Equation 2, below. A more complete demonstration is provided in Microchip application note *AN821, "Advanced Encryption Standard Using the PIC16XXX"* (DS00821).

ENCODE KEY SCHEDULING

Each round of AES uses a different encryption key based on the previous encryption key. The key schedule algorithm also uses the S-table, the `xtime` routine and `Round_con`, an initial encryption value.

Consider the generic key:

K0	K4	K8	K12
K1	K5	K9	K13
K2	K6	K10	K14
K3	K7	K11	K15

Starting with key matrix created from the original plaintext key, the key scheduling is as follows:

1. The values of column 3 of the key matrix (K12 through K15) are used to obtain values from the S-Table.
2. Column 0 of the key matrix (K0 through K3) is XORed with the S-Table look-up values of column 3.
3. K0 is XORed with `Round_con` (the original value of `Round_con` is 01h for encoding).
4. `Round_con` is then updated with the `Xtime` of `Round_con` for the next round.
5. Column 1 is XORed with column 0.
6. Column 2 is XORed with column 1.
7. Column 3 is XORed with column 2.

EQUATION 2:

$$b[0] = xtime(a[0]) \oplus (a[1] \oplus xtime(a[1])) \oplus a[2] \oplus a[3]$$

where: "⊕" is the XOR operation

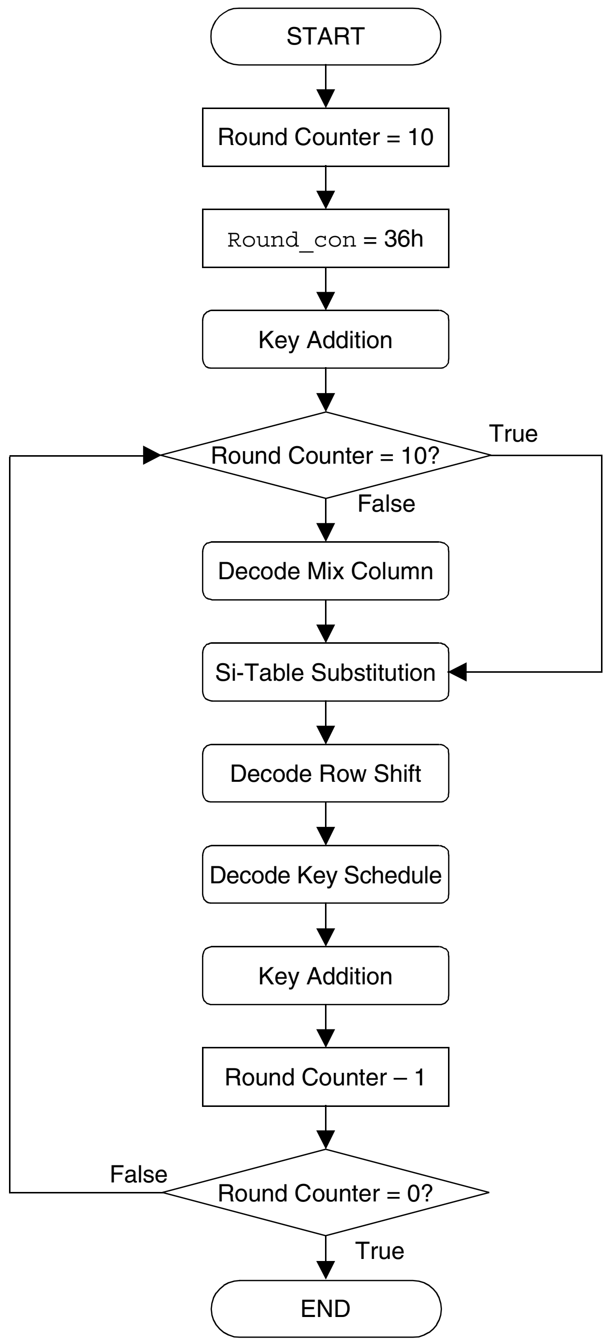
Note: The members of the multiplication are XORed together rather then added together as they would in regular matrix multiplication.

Decryption

The functional divisions of the decryption algorithm are similar to those for the encryption algorithm, with most being the inverse operation. One major difference, however, is in the setup preceding the decryption. The decryption key differs from than the encryption key and must be loaded correctly. It can be calculated by running through the encryption key schedule the appropriate number of rounds. After the completion of an encryption cycle, the key is transformed into a decryption key. The decryption key can be precalculated and stored in the system, or recalculated each time as needed.

The value of `Round_con` must also be set differently for the decryption process. The value of 36h is used for 10 rounds.

FIGURE 9: DECRYPT FLOWCHART



KEY ADDITION

In a manner like the encryption process, each byte of the initial decryption key is XORed with each of the corresponding data bytes. On subsequent rounds, the key generated by the key schedule for that round is XORed in a bytewise manner with the data.

DECODE MIX COLUMN

The inverse mix column operation (Equation 3) differs from the encode mix column operation by only the matrix $c(x)$. Note that the coefficients for $c(x)$ are in hexadecimal.

EQUATION 3:

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix} \times \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

FIXED MATRIX c(x)

Si-TABLE SUBSTITUTION

To undo the S-Table substitutions of the encryption process, a fixed Si-Table is used (Table 7). During each round, each data byte is replaced with a corresponding byte from the Si-Table.

DECODE ROW SHIFT

As with encryption, row shift is a cyclical left shift of the rows in the data. For decryption, the different row shift values are shown in Table 6.

TABLE 6: DECRYPTION CYCLICAL SHIFT

Before Row Shift:			
0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15
After Row Shift:			
0	4	8	12
13	1	5	9
10	14	2	6
7	11	15	3

Note that this transformation is different for encryption and decryption. Also note that the results of this transformation are equivalent to the row shift transformation used during encryption if the blocks are shifted to the right instead of to the left.

TABLE 7: Si-TABLE DECRYPTION SUBSTITUTION TABLE (VALUES IN HEXADECIMAL)

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

DECODE KEY SCHEDULE

Each round of AES decryption uses the same key that was used to encrypt the data. The key for the next iteration can be determined from the previous decryption key by performing the inverse operation to the encryption key schedule.

Starting from the decryption key of the previous round, the key scheduling is as follows:

1. Column 3 is XORed with column 2.
2. Column 2 is XORed with column 1.
3. Column 1 is XORed with column 0.
4. Column 0 is XORed with the S-Table look-up of column 3. (**Note:** This step uses the S-Table (Table 4), not the Si-Table (Table 7).)
5. K0 is XORed with `Round_con`
6. `Round_con` is updated with the inverse `xtime` of `Round_con`. The inverse `xtime` function can be defined in C, as shown is Example 2.

EXAMPLE 2: INVERSE `xtime` ROUTINE

```
if(a&0x01)
{
        a=0x80;
}
else
{
        a>>=1;
}
```

Using the AES Algorithm

The implementation of AES discussed here is accessed through three function calls: `AESEncrypt`, `AESDecrypt` and `AESCalcDecKey`. Their usage is discussed below.

`AESEncrypt`

This function encrypts a 16-byte block of data in place with a 128-bit (16-byte) key using the AES algorithm.

Syntax

```
void AESEncrypt(int *DataBlock, const int *EncryptKey)
```

Parameters

`*DataBlock`: Pointer to the 16-byte block of data to encrypt. The block of data must begin on an even memory address.

`*EncryptKey`: Pointer to the 16-byte key to use for encryption. The key must begin on an even memory address.

Return Values

`DataBlock`: 16 bytes of plaintext at `*DataBlock` is replaced with 16 bytes of cipher text.

Pre-Condition

None

Side Effects

None

Remarks

1. Peak stack memory usage is 40 bytes, including the 4-byte return address.
2. `AESEncrypt` requires 2808 instruction cycles (including `CALL`, `RETURN` and two parameter load instructions) when `EncryptKey` is stored in data memory. If `EncryptKey` is stored in program memory, eight additional instruction cycles are required.
3. `AESEncrypt` is interrupt and re-entrant safe.

Example

```
int block[8];
int key[8] = {0x0100,0x0302,0x0504,0x0706,0x0908,0x0B0A,0x0D0C,0x0F0E};
...
// Load block[] with application data
...
AESEncrypt(block, key);
```

AESDecrypt

This function decrypts a 16-byte block of data in place with a 128-bit (16-byte) key using the AES algorithm.

Syntax

```
void AESDecrypt(int *DataBlock, const int *DecryptKey)
```

Parameters

`*DataBlock`: Pointer to the 16-byte block of data to decrypt. The block of data must begin on an even memory address.

`*DecryptKey`: Pointer to the 16-byte key to use for decryption. This key is not the same key used for encryption. Use the `AESCalcDecKey` function to derive a decryption key from an encryption key. The key must begin on an even memory address.

Return Values

`DataBlock`: 16 bytes of cipher text at `*DataBlock` is replaced with 16 bytes of plaintext.

Pre-Condition

If necessary, calculate the decryption key.

Side Effects

None

Remarks

1. Peak stack memory usage is 40 bytes, including the 4-byte return address.
2. When `DecryptKey` is stored in data memory, `AESDecrypt` requires 4490 instruction cycles (including `CALL`, `RETURN` and two parameter load instructions). If `DecryptKey` is stored in program memory, nine additional instruction cycles are required.
3. `AESDecrypt` is interrupt and re-entrant safe.

Example

```
int block[8];
int key[8] = {0x0100,0x0302,0x0504,0x0706,0x0908,0x0B0A,0x0D0C,0x0F0E};
// Assuming key is loaded with the encryption key, calculate a decryption key
// first
AESCalcDecKey(key);
...
// Load block[] with application data
...
AESDecrypt(block, key);
```

AESCalcDecKey

This function derives a 128-bit (16-byte) decryption key from a 128-bit (16-byte) encryption key.

Syntax

```
void AESCalcDecKey(int *Key)
```

Parameters

`*Key`: Pointer to the 16-byte encryption key to translate. The key must begin on an even memory address.

Return Values

`Key`: 16-byte encryption key at `*Key` is replaced with 16-byte decryption key.

Pre-Condition

None

Side Effects

None

Remarks

1. Peak stack memory usage is 6 bytes, including the 4-byte return address.
2. `AESCalcDecKey` requires 497 instruction cycles (including `CALL`, `RETURN` and one parameter load instruction).
3. `AESCalcDecKey` is interrupt and re-entrant safe.
4. If this function is not needed, it may be deleted to save program memory.

Example

```
int key[8] = {0x0100,0x0302,0x0504,0x0706,0x0908,0x0B0A,0x0D0C,0x0F0E};

// Assuming key is loaded with an encryption key, calculate the decryption key
AESCalcDecKey(key);
```

PERFORMANCE

The 16-bit implementations of TDES and AES were evaluated on the PIC24FJ128GA010, running at a clock speed of 32 MHz (16 MIPS). The results are shown in Table 8.

TABLE 8: EXECUTION TIME AND THROUGHPUT PERFORMANCE FOR PIC24/dsPIC® DEVICE ENCRYPTION ALGORITHMS

Algorithm	Execution Time (Instruction Cycles)		Throughput (Kbit/s) @ 16 MIPS	
	Encrypt	Decrypt	Encrypt	Decrypt
TDES (8 bytes/block)	6403(1) 13557(2)	6403(1) 13557(2)	159(1) 75.5(2)	159(1) 75.5(2)
AES (16 bytes/block)	2808	4490	729	456

Note 1: Key value is constant for each block and does not require recalculation.
2: Key value is recalculated for each block.

RESOURCE USAGE

The memory requirements of the algorithms are shown in Table 9.

TABLE 9: MEMORY USAGE FOR ENCRYPTION ALGORITHMS

Algorithm	Data RAM (Bytes)	Program Memory (Bytes)
TDES	430(1)	7500
AES	40	3018

Note 1: An additional reduction of data RAM usage can be achieved if Key 1 and Key 3 are always equal to the same value. If the application code is modified to do this, the application will use 302 bytes.

SUMMARY

TDES and AES are two of only three encryption algorithms that are used as Federal Information Processing Standards. Both of these algorithms are available for PIC24 and dsPIC30/33 devices as compact and efficient implementations.

This purpose of this document has been to introduce the reader to the algorithms and their practical use in application code. A full discussion of the algorithms, usage modes and test vectors for the algorithms are provided in the FIPS documentation.

It is important to remember when working data encryption algorithms, that no encryption algorithm is secure. Data encryption algorithms only provide a probability of security. It is also important to be aware of any exportation control laws that may affect the source code or end product that have cryptographic elements.

REFERENCES

A. Lovrich and M. Palmer, *AN583, "Implementation of the Data Encryption Standard Using PIC17C42"* (DS00583), Microchip Technology, Inc., 1997.

C. Gübel, *AN821, "Advanced Encryption Standard Using the PIC16XXX"* (DS00821), Microchip Technology, Inc., 2002.

D. Flowers, *AN953, "Data Encryption Routines for the PIC18"* (DS00953), Microchip Technology, Inc., 2005.

D.C. Feldmeier, *"A High-Speed Software DES Implementation"*, Computer Communication Research Group, Bellcore, June 1989.

Institute for Applied Information Processing and Communications, Graz University of Technology, *"AES Lounge"* (AES public home page), http://www.iaik.tu-graz.ac.at/research/krypto/AES/

Computer Security Resource Center, National Institute of Standards and Technology, *"Cryptographic Toolkit"* (home page, link to archival information on AES), http://csrc.nist.gov/CryptoToolkit/tkencryption.html.

APPENDIX A: SOFTWARE DISCUSSED IN THIS APPLICATION NOTE

Because of statutory export license restrictions on encryption software, the source code listings for the AES and TDES algorithms are not provided here. These applications may be ordered from Microchip Technology, Inc. through its sales offices, or through the corporate web site:

www.microchip.com

Interested users are encouraged to check the web site or their nearest sales office for more information.

NOTES:

WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
http://support.microchip.com
Web Address:
www.microchip.com

Atlanta
Alpharetta, GA
Tel: 770-640-0034
Fax: 770-640-0307

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Farmington Hills, MI
Tel: 248-538-2250
Fax: 248-538-2260

Kokomo
Kokomo, IN
Tel: 765-864-8360
Fax: 765-864-8387

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608

Santa Clara
Santa Clara, CA
Tel: 408-961-6444
Fax: 408-961-6445

Toronto
Mississauga, Ontario,
Canada
Tel: 905-673-0699
Fax: 905-673-6509

ASIA/PACIFIC

Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Habour City, Kowloon
Hong Kong
Tel: 852-2401-1200
Fax: 852-2401-3431

Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing
Tel: 86-10-8528-2100
Fax: 86-10-8528-2104

China - Chengdu
Tel: 86-28-8676-6200
Fax: 86-28-8676-6599

China - Fuzhou
Tel: 86-591-8750-3506
Fax: 86-591-8750-3521

China - Hong Kong SAR
Tel: 852-2401-1200
Fax: 852-2401-3431

China - Qingdao
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

China - Shanghai
Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

China - Shenyang
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

China - Shenzhen
Tel: 86-755-8203-2660
Fax: 86-755-8203-1760

China - Shunde
Tel: 86-757-2839-5507
Fax: 86-757-2839-5571

China - Wuhan
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

China - Xian
Tel: 86-29-8833-7250
Fax: 86-29-8833-7256

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-4182-8400
Fax: 91-80-4182-8422

India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

India - Pune
Tel: 91-20-2566-1512
Fax: 91-20-2566-1513

Japan - Yokohama
Tel: 81-45-471- 6166
Fax: 81-45-471-6122

Korea - Gumi
Tel: 82-54-473-4301
Fax: 82-54-473-4302

Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

Malaysia - Penang
Tel: 60-4-646-8870
Fax: 60-4-646-5086

Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

Taiwan - Hsin Chu
Tel: 886-3-572-9526
Fax: 886-3-572-6459

Taiwan - Kaohsiung
Tel: 886-7-536-4818
Fax: 886-7-536-4803

Taiwan - Taipei
Tel: 886-2-2500-6610
Fax: 886-2-2508-0102

Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

EUROPE

Austria - Wels
Tel: 43-7242-2244-3910
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

UK - Wokingham
Tel: 44-118-921-5869
Fax: 44-118-921-5820

07/21/06