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

![Original DES Algorithm Diagram]
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 5 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 |

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 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 |

**FIGURE 6: S-BOX MATRICES (Sₙ)**

| S₁ = 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 |
| S₂ = 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 |
| S₃ = 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 |
| S₄ = 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 |
| S₅ = 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 |
| S₆ = 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 |
| S₇ = 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 5 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 |
| S₈ = 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

- START
- Round Counter = 10
- Round_con = 1
- Key Addition
- S-Table Substitution
- Encode Row Shift
- Round Counter = 1?
  - True
  - Round Counter – 1
  - False
    - Encode Mix Column
    - Encode Key Schedule
    - Key Addition
    - Round Counter – 1
    - False
    - Round Counter = 0?
      - True
      - END
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**

<table>
<thead>
<tr>
<th>Key Size</th>
<th>Rounds Needed for Block Size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>16-Byte</td>
</tr>
<tr>
<td>16-byte</td>
<td>10*</td>
</tr>
<tr>
<td>24-byte</td>
<td>12</td>
</tr>
<tr>
<td>32-byte</td>
<td>14</td>
</tr>
</tbody>
</table>

* 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**

|---------|---------|---------|----------|

**TABLE 3: DATA MATRIX**

|----------|----------|----------|-----------|

**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 byte wise 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)**

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

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:

\[
\begin{pmatrix}
 b_0 \\
 b_1 \\
 b_2 \\
 b_3
\end{pmatrix} =
\begin{pmatrix}
 02 & 03 & 01 & 01 \\
 01 & 02 & 03 & 01 \\
 01 & 01 & 02 & 03 \\
 03 & 01 & 01 & 02
\end{pmatrix}
\begin{pmatrix}
 a_0 \\
 a_1 \\
 a_2 \\
 a_3
\end{pmatrix}
\]

The special rules for multiplication equate to the following:
\[a \cdot 1 = a\]
\[a \cdot 2 = \text{xtime}(a)\]
\[a \cdot 3 = a \oplus \text{xtime}(a)\]
\[a \cdot 4 = \text{xtime}(\text{xtime}(a))\]
\[a \cdot 5 = a \oplus \text{xtime}(\text{xtime}(a))\]
...

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

\[
\begin{align*}
\text{if}(a<0x80) \\
&\quad \{a<<=1;\} \\
\text{else} \quad \{a=(a<<1)^0x1b;\}
\end{align*}
\]

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 \text{xtime} routine and \text{Round_con}, an initial encryption value.

Consider the generic key:

\[
\begin{align*}
K0 & \quad K4 & \quad K8 & \quad K12 \\
K1 & \quad K5 & \quad K9 & \quad K13 \\
K2 & \quad K6 & \quad K10 & \quad K14 \\
K3 & \quad K7 & \quad K11 & \quad K15
\end{align*}
\]

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 \text{Round_con} (the original value of \text{Round_con} is 01h for encoding).
4. \text{Round_con} is then updated with the \text{Xtime} of \text{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.

\[
b[0] = \text{xtime}(a[0]) \oplus (a[1] \oplus \text{xtime}(a[1])) \oplus a[2] \oplus a[3]
\]

where: “\(\oplus\)” 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.

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

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

```c
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**

```c
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**

```c
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**

```c
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**

```c
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

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Execution Time (Instruction Cycles)</th>
<th>Throughput (Kbit/s) @ 16 MIPS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Encrypt</td>
<td>Decrypt</td>
</tr>
<tr>
<td>TDES (8 bytes/block)</td>
<td>6403(1)</td>
<td>13557(2)</td>
</tr>
<tr>
<td>AES (16 bytes/block)</td>
<td>2808</td>
<td>4490</td>
</tr>
</tbody>
</table>

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

**Note 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

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Data RAM (Bytes)</th>
<th>Program Memory (Bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TDES</td>
<td>430(1)</td>
<td>7500</td>
</tr>
<tr>
<td>AES</td>
<td>40</td>
<td>3018</td>
</tr>
</tbody>
</table>

**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


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/

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.
Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip’s Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as “unbreakable.”

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip’s code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Microchip disclaims all liability arising from this information and its use. Use of Microchip devices in life support and/or safety applications is entirely at the buyer’s risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights.

Trademark List:
The Microchip name and logo, the Microchip logo, Accuron, dsPIC, KEELoCo, microID, MPLAB, PIC, PICmicro, PICSTART, PRO MATE, PowerSmart, rPIC, and SmartShunt are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.
AmpLab, FilterLab, Migratable Memory, MXDEV, MXLAB, SEEVAL, SmartSensor and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.
Analog-for-the-Digital Age, Application Maestro, CodeGuard, dsPICDEM, dsPICDEM.net, dsPICWorks, ECAN, ECONOMONITOR, FanSense, FlexROM, fuzzyLAB, In-Circuit Serial Programming, ICSP, ICEPIC, Linear Active Thermistor, Mindi, Micro, MPASM, MPLIB, MPLINK, PICkit, PICDEM, PICDEM.net, PICLAB, PICtail, PowerCal, PowerInfo, PowerMate, PowerTool, REAL ICE, rLAB, rfPICDEM, Select Mode, Smart Serial, SmartTel, Total Endurance, UNI/O, WiperLock and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.
SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.
All other trademarks mentioned herein are property of their respective companies.
© 2006, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.
© Printed on recycled paper.

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
ISO/TS 16949:2002
<table>
<thead>
<tr>
<th>AMERICAS</th>
<th>ASIA/PACIFIC</th>
<th>ASIA/PACIFIC</th>
<th>EUROPE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Corporate Office 2355 West Chandler Blvd. Chandler, AZ 85224-6199 Tel: 480-792-7200 Fax: 480-792-7277 Technical Support: <a href="http://support.microchip.com">http://support.microchip.com</a> Web Address: <a href="http://www.microchip.com">www.microchip.com</a></td>
<td>Asia Pacific Office Suites 3707-14, 37th Floor Tower 6, The Gateway Harbour City, Kowloon Hong Kong Tel: 852-2401-1200 Fax: 852-2401-3431</td>
<td>India - Bangalore Tel: 91-80-4182-8400 Fax: 91-80-4182-8422</td>
<td>Austria - Wels Tel: 43-7242-2244-3910 Fax: 43-7242-2244-393</td>
</tr>
<tr>
<td>Atlanta Alpharetta, GA Tel: 770-640-0034 Fax: 770-640-0307</td>
<td>China - Beijing Tel: 86-10-8528-2100 Fax: 86-10-8528-2104</td>
<td>India - New Delhi Tel: 91-11-4160-8631 Fax: 91-11-4160-8632</td>
<td>Denmark - Copenhagen Tel: 45-4450-2828 Fax: 45-4485-2829</td>
</tr>
<tr>
<td>Chicago Itasca, IL Tel: 630-285-0071 Fax: 630-285-0075</td>
<td>China - Fuzhou Tel: 86-591-8750-3506 Fax: 86-591-8750-3521</td>
<td>Japan - Yokohama Tel: 81-45-471-6166 Fax: 81-45-471-6122</td>
<td>Germany - Munich Tel: 49-89-627-144-0 Fax: 49-89-627-144-44</td>
</tr>
<tr>
<td>Dallas Addison, TX Tel: 972-818-7423 Fax: 972-818-2924</td>
<td>China - Hong Kong SAR Tel: 852-2401-1200 Fax: 852-2401-3431</td>
<td>Korea - Gumi Tel: 82-54-473-4301 Fax: 82-54-473-4302</td>
<td>Italy - Milan Tel: 39-0331-742611 Fax: 39-0331-466781</td>
</tr>
<tr>
<td>Detroit Farmington Hills, MI Tel: 248-538-2250 Fax: 248-538-2260</td>
<td>China - Qingdao Tel: 86-532-8502-7355 Fax: 86-532-8502-7205</td>
<td>Korea - Seoul Tel: 82-2-554-7200 Fax: 82-2-558-5932 or 82-2-558-5934</td>
<td>Netherlands - Drunen Tel: 31-416-690399 Fax: 31-416-690340</td>
</tr>
<tr>
<td>Kokomo Kokomo, IN Tel: 765-864-8360 Fax: 765-864-8387</td>
<td>China - Shanghai Tel: 86-21-5407-5533 Fax: 86-21-5407-5066</td>
<td>Malaysia - Penang Tel: 60-4-646-8870 Fax: 60-4-646-5086</td>
<td>Spain - Madrid Tel: 34-91-708-08-90 Fax: 34-91-708-08-91</td>
</tr>
<tr>
<td>Santa Clara Santa Clara, CA Tel: 408-961-6444 Fax: 408-961-6445</td>
<td>China - Shenzhen Tel: 86-755-8203-2660 Fax: 86-755-8203-1760</td>
<td>Singapore Tel: 65-6334-8870 Fax: 65-6334-8850</td>
<td></td>
</tr>
<tr>
<td>Toronto Mississauga, Ontario, Canada Tel: 905-673-0699 Fax: 905-673-6509</td>
<td>China - Shunde Tel: 86-757-2839-5507 Fax: 86-757-2839-5571</td>
<td>Taiwan - Hsin Chu Tel: 886-3-572-9526 Fax: 886-3-572-6459</td>
<td></td>
</tr>
<tr>
<td></td>
<td>China - Wuhan Tel: 86-27-5980-5300 Fax: 86-27-5980-5118</td>
<td>Taiwan - Kaohsiung Tel: 886-7-536-4818 Fax: 886-7-536-4803</td>
<td></td>
</tr>
<tr>
<td></td>
<td>China - Xian Tel: 86-29-8833-7250 Fax: 86-29-8833-7256</td>
<td>Taiwan - Taipei Tel: 886-2-2500-6610 Fax: 886-2-2508-0102</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Thailand - Bangkok Tel: 66-2-694-1351 Fax: 66-2-694-1350</td>
<td></td>
</tr>
</tbody>
</table>