AN953

Data Encryption Routines for the PIC18

Author: David Flowers
Microchip Technology Inc.

INTRODUCTION

This Application Note covers four encryption algorithms: AES, XTEA, SKIPJACK® and a simple encryption algorithm using a pseudo-random binary sequence generator. The science of cryptography dates back to ancient Egypt. In today's era of information technology where data is widely accessible, sensitive material, especially electronic data, needs to be encrypted for the user's protection. For example, a network-based card entry door system that logs the persons who have entered the building may be susceptible to an attack where the user information can be stolen or manipulated by sniffing or spoofing the link between the processor and the memory storage device. If the information is encrypted first, it has a better chance of remaining secure. Many encryption algorithms provide protection against someone reading the hidden data, as well as providing protection against tampering. In most algorithms, the decryption process will cause the entire block of information to be destroyed if there is a single bit error in the block prior to decryption.

ENCRYPTION MODULE OVERVIEW

- Four algorithms to choose from, each with their own benefits
- Advanced Encryption Standard (AES)
  - Modules available in C, Assembly and Assembly written for C
  - Allows user to decide to include encoder, decoder or both
  - Allows user to pre-program a decryption key into the code or use a function to calculate the decryption key
- Tiny Encryption Algorithm version 2 (XTEA)
  - Modules available in C and Assembly
  - Programmable number of iteration cycles
- SKIPJACK
  - Module available in C
- Pseudo-random binary sequence generator XOR encryption
  - Modules available in C and Assembly
  - Allows user to change the feedback taps at run-time
  - KeyJump breaks the regular cycle of the to increase the variability of the sequence
- Out-of-the-box support for MPLAB® C 18
- Various compiling options to customize routines for a specific application
  - Available as a Microchip Application Maestro™ module to simplify customization

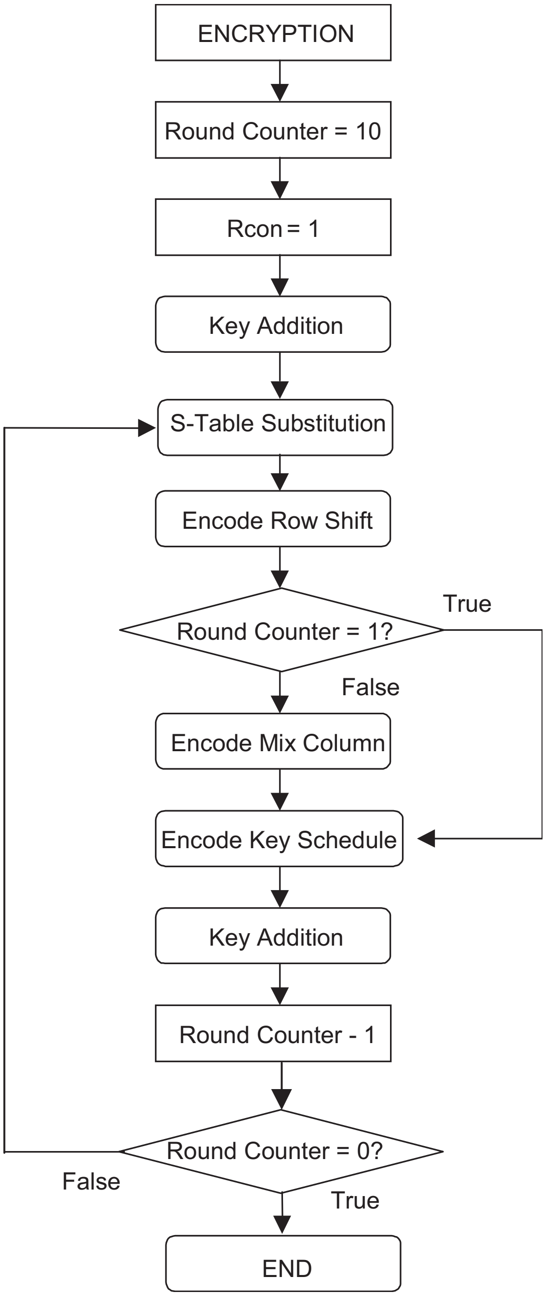
AES

Overview/History/Background

The Advanced Encryption Standard (AES) is a means of encrypting and decrypting data adopted by the National Institute of Standards and Technology (NIST) on October 2, 2000. In the late 1990s, NIST held a contest to initiate the development of encryption algorithms that would replace the Data Encryption Standard (DES). The contest tested the algorithms' security and execution speed to determine which would be named the AES. The algorithm chosen is called the "Rijndael" algorithm after its two designers, Joan Daemen and Vincent Rijmen of Belgium. AES is a symmetric block cipher that utilizes a secret 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

FIGURE 1: AES ENCRYPT FLOWCHART



There are five basic subdivisions of the encryption flowchart. A detailed explanation of each will follow. The number of rounds needed in the transformation is taken from the following table.

	16-Byte Block	24-Byte Block	32-Byte Block
16-byte key	10*****	12	14
24-byte key	12	12	14
32-byte key	14	14	14

***** Used in this implementation.

This implementation of AES uses a 16-byte block and a 16-byte key and thus uses 10 rounds of encryption. On the last encryption round, the `mix column` subdivision is left out. The structures of the key and data blocks are shown below.

TABLE 1: KEY MATRIX:

Key [0]	Key [4]	Key [8]	Key [12]	Key [16]	Key [20]	Key [24]	Key [28]
Key [1]	Key [5]	Key [9]	Key [13]	Key [17]	Key [21]	Key [25]	Key [29]
Key [2]	Key [6]	Key [10]	Key [14]	Key [18]	Key [22]	Key [26]	Key [30]
Key [3]	Key [7]	Key [11]	Key [15]	Key [19]	Key [23]	Key [27]	Key [31]

TABLE 2: DATA MATRIX:

Data [0]	Data [4]	Data [8]	Data [12]	Data [16]	Data [20]	Data [24]	Data [28]
Data [1]	Data [5]	Data [9]	Data [13]	Data [17]	Data [21]	Data [25]	Data [29]
Data [2]	Data [6]	Data [10]	Data [14]	Data [18]	Data [22]	Data [26]	Data [30]
Data [3]	Data [7]	Data [11]	Data [15]	Data [19]	Data [23]	Data [27]	Data [31]

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. For example, take the following quote from “A Tale of Two Cities”, by Charles Dickens. “It was the best of times, it was the worst of times, …”. Broken into 16-byte blocks, the data would now look similar to this:

EXAMPLE 1: PLAIN TEXT DIVIDED INTO 16-BYTE BLOCKS

It was the best	of times, it was	the worst of ti	mes, ... tuvwxyz**(1)**

Note 1: If any of the blocks do not fit into the correct size matrix, the values must be padded at the end of the block. In this case, “tuvwxyz” is added to the end of the quote to complete the last block.

Finally a key must be selected that is 128-bits long. For all of the examples in this document, the key will be [Charles Dickens].

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

S-Table (Encryption Substitution Table):

S-Table Substitution is a direct table lookup and replacement of the data. Below is the C code for this procedure:

```
for(i=0;i<BLOCKSIZE;i++)
        {
            block[i]=STable[block[i]];
        }
```

The values of the table are defined in the following chart.

TABLE 3: S-BOX OR 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

Xtime

`xtime(a)` is a predefined linear feedback shift register that repeats every 51 cycles. The operation is defined as:

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

Key Schedule

Each round of AES uses a different encryption key based on the previous encryption key. An example follows that takes a given key and calculates the next round's key.

EXAMPLE 2: KEY GENERATION

Given the generic key:

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

The key scheduling goes as follows:

1. Column 0 is XORed with the S_Table lookup of column 3:

   K0 ^= S_Table[K13]
   K1 ^= S_Table[K14]
   K2 ^= S_Table[K15]
   K3 ^= S_Table[K12]

2. K0 is XORed with Rcon

   `K0 ^= Rcon;`

3. Rcon is updated with the xtime of Rcon

   `Rcon = xtime(Rcon);`

   (the starting value of Rcon is 0x01 for encoding)

4. Column 1 is XORed with column 0:

   K4 ^= K0
   K5 ^= K1
   K6 ^= K2
   K7 ^= K3

5. Column 2 is XORed with column 1:

   K8 ^= K4
   K9 ^= K5
   K10 ^= K6
   K11 ^= K7

6. Column 3 is XORed with column 2:

   K12 ^= K8
   K13 ^= K9
   K14 ^= K10
   K15 ^= K11

Key Addition:

Key addition is defined as each byte of the key XORed with each of the corresponding data bytes. The key addition process is the same for both the encryption and decryption processes. The following is a C-code example:

```
for(i=0;i<16;i++)
{
data[i] ^= key[i];
}
```

Row Shift

Row shift is a cyclical shift to the left of the rows in the data block according to the table below:

TABLE 4: ENCRYPTION CYCLICAL SHIFT TABLE

	# shifts of row 0	**# shifts of row 1**	**# shifts of row 2**	**# shifts of row 3**
16-byte block	0	1	2	3
24-byte block	0	1	2	3
32-byte block	0	1	3	4

Note that this transformation is different for encryption and decryption.

EXAMPLE 3: TRANSFORMATION

Given the original data:

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

The results of the transformation would be as follows:

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

`mix column`

Chapter 2, Section 4.2.3 of the AES specification defines the `mix column` transformation. In this operation, a 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).

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \overset{\text{FIXED MATRIX } c(x)}{\begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix}} \text{ X } \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

The special rules for multiplication equate to the following:

a ● 1 = a

a ● 2 = xtime(a)

a ● 3 = a ⊕ xtime(a)

a ● 4 = xtime(xtime(a))

a ● 5 = a ⊕ xtime(xtime(a))

...

The first row of the resulting multiplication would be:

EXAMPLE 4:

`b[0]= xtime(a[0]) ⊕ (a[1] ⊕ xtime(a[1])) ⊕ a[2] ⊕ a[3]`(1,2)

Note 1: ⊕ stands for XOR

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

Refer to Application Note 821, “Advanced Encryption Standard Using the PIC16XXX” (DS00821), available at www.microchip.com, for a full example of this multiplication.

EXAMPLE 5: AES ENCRYPTION EXAMPLE:

Plain text: [It was the best][of times, it was][the worst of ti][mes, ... tuvwxyz(1)]

Key: [Charles Dickens.]

Plain hex: [0x49742077617320746865206265737420](2,3)…

Cipher hex: [0x3FD869084483504CA70E246064DD76CA]…

Note 1: The line has been padded with “tuvwxyz” so that the data fits in the block.

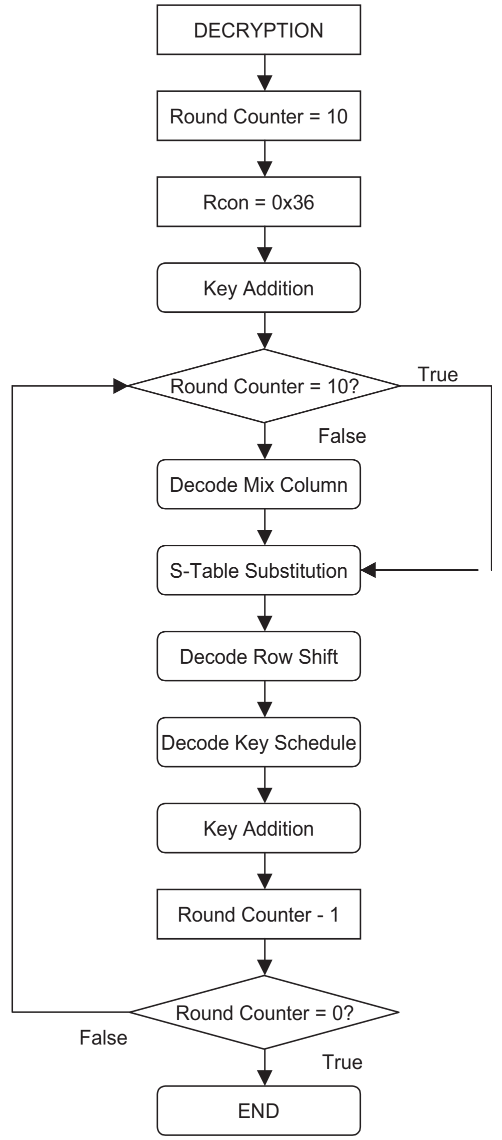
2: Only first block results shown.

3: In between each block, the key must be reset or the change in key must be taken into consideration when decryption begins.

DECRYPTION

The subdivisions of the decryption algorithm are similar to those of the encryption algorithm, with most being the inverse operation. One major difference, however, is in the setup preceding the decryption. The decryption key is different than the encryption key and must be loaded correctly. The decryption key can be calculated by running through the encryption key schedule the appropriate number of rounds. If, during encryption, the key is not reset between blocks, the decryption key will then have to adjust accordingly. The value of Rcon must also be set differently for the decryption process. The value of 0x36 is used for 10 rounds. This is determined by running the encryption key schedule routine for the appropriate number of rounds.

FIGURE 2: DECRYPT FLOWCHART



Si-Table (decryption lookup table):

The Si-Table is similar to the S-Table in function and provides the inverse loop-up results.

TABLE 5: Si-BOX OR 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

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. To obtain the decryption key from the encryption key, cycle the appropriate amount of times through the encryption key schedule. At the end of an encryption cycle, the value of the key at that point is the correct decryption key, so this value can be saved, recalculated later or pre-calculated and stored in the system.

Given the generic key:

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

The key scheduling goes as follows:

Starting from the decryption key:

1. Column 3 is XORed with column 2:

K12 ^= K8
K13 ^= K9
K14 ^= K10
K15 ^= K11

2. Column 2 is XORed with column 1:

K8 ^= K4
K9 ^= K5
K10 ^= K6
K11 ^= K7

3. Column 1 is XORed with column 0:

K4 ^= K0
K5 ^= K9
K6 ^= K10
K7 ^= K11

4. Column 0 is XORed with the S-Table lookup of column 3 (**Note:** This uses the S-Table and not the Si-Table):

K4 ^= S_Table [K13]
K5 ^= S_Table [K14]
K6 ^= S_Table [K15]
K7 ^= S_Table [K16]

5. K0 is XORed with Rcon

   `K0 ^= Rcon;`

6. Rcon is updated with the inverse xtime of Rcon

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

Note: The starting value of Rcon is 0x36 for decoding using a 128-bit key

Row Shift

Row shift is a cyclical shift to the left of the rows in the data based on the below table:

	# shifts of row 0	**# shifts of row 1**	**# shifts of row 2**	**# shifts of row 3**
16-byte block	0	3	2	1
24-byte block	0	5	4	3
32-byte block	0	7	5	4

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

EXAMPLE 6: ROW SHIFT

Given the original data:

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

The results of the transformation would be as follows:

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

`Inverse Mix Column:`

The inverse mix column operation differs from the mix column operation by only the matrix c(x).

(**Note:** all values are in hexadecimal)

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} = \overset{\text{FIXED MATRIX c(x)}}{\begin{bmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{bmatrix}} X \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \end{bmatrix}$$

The operation a[0] ● 0x0E ⊕… is very calculation intensive for an 8-bit processor. There are several different methods of calculating these numbers to reduce the mathematical load. The method chosen for this implementation is a simple lookup table of the xtime(a), xtime(xtime(a)) and xtime(xtime(xtime(a))) values.

Overview of Routines

AESEncode

This function encrypts the input data with the input key.

Syntax

`void Encode(unsigned char* block, unsigned char* key)`

Parameters

`Block` – block of data to encrypt,
`key` – the key used to encrypt

Return Values

None

Pre-condition

Block and key preloaded with the correct values

Side-effects

Values in `block` have changed to the encrypted version, Key contains the decryption key for that block

Remarks

None

Example: Usage of Encode

```
C
...
AESEncode(block,key);
...
Assembly
...
mov1w   0x34
movwf   key+0
mov1w   0x12
movwf   key+1
...
movff   data+0,block+0
movff   data+1,block+1
...
call    AESEncode
```

`AESDecode`

This function decrypts the input data with the input decryption key.

Syntax

`void Decode(unsigned char* block, unsigned char* key)`

Parameters

`Block` – block of data to decrypt,
`key` – the key used to decrypt

Return Values

None

Pre-condition

Block and key preloaded with the correct values

Side Effects

Values in `block` have changed to the decrypted version, `Key` contains the original encryption key for that block

Remarks

None

Example: Usage of `Decode`

```
C
...
AESDecode(block,key);


Assembly
...
mov1w   0x34
movwf   key+0
mov1w   0x12
movwf   key+1
...
movff   data+0,block+0
movff   data+1,block+1
...
call    AESDecode
```

`AESCalcDecodeKey`

This function calculates the decryption key for a block of data from the encryption key.

Syntax

```
void Decode(unsigned char* block, unsigned char* key)
```

Parameters

`key` – the key used to encrypt

Return Values

None

Pre-condition

`key` preloaded with the correct values

Side Effects

`Key` contains the decryption key for that block

Remarks

None

Example: Usage of `calcDecodeKey`

```
C
...
AEScalcDecodeKey(key);

Assembly
...
mov1w   0x34
movwf   key+0
mov1w   0x12
movwf   key+1
...
call    AEScaleDecodeKey
```

XTEA

Overview/History/Background

Tiny Encryption Algorithm version 2 (XTEA) is an encryption algorithm that gets its fame from its size. XTEA is an adaptation of the original algorithm (TEA) after a weakness was found in its structure. XTEA's authors are David Wheeler and Roger Needham of the Cambridge Computer Laboratory. XTEA gets its security from the number of encryption iterations it goes through. The authors recommend 64 iterations for high security, but they believe 32 iterations should be secure for several decades, with as few as 16 iterations being sufficient for applications with lower security needs.

The most notable feature of XTEA is the smallness of the encryption decryption algorithm. Below is the code and flow chart (next page) for the encryption cycle.

```
x1 += ((x2<<4 ^ x2>>5) + x2) ^ (sum + *(key+(sum&0x03)));
sum+=DELTA;
x2 += ((x1<<4 ^ x1>>5) + x1) ^ (sum + *(key+(sum>>11&0x03)));
```

EXAMPLE 7: XTEA ENCRYPTION EXAMPLE

Plain text: [It was t][he best][of times][, it was][the wor][st of ti][mes, ...]

Key: [Charles Dickens.]

Plain hex: [0x4974207761732074](1,2)…

Cipher hex: [0x7C0BA7CED6E78034]…

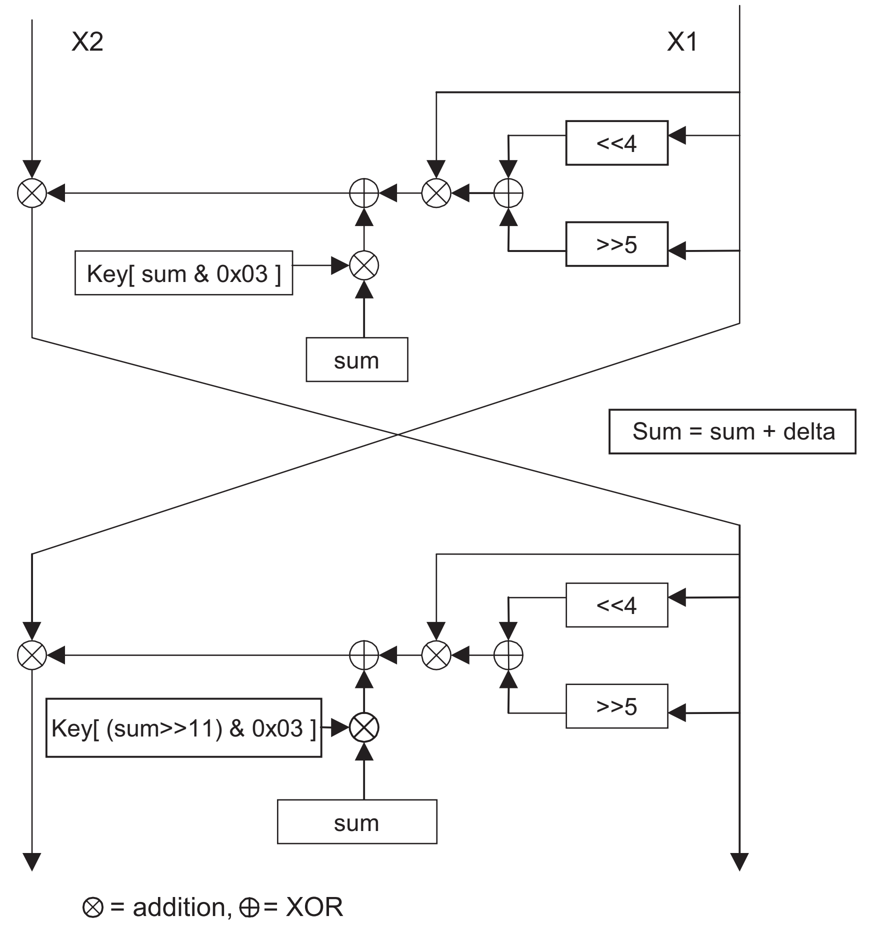
Note 1: Only first block results shown.

2: In between each block, the key must be reset. Otherwise, the decryption flow chart must be changed.

The decoding process is equally as simple. The following C-code describes the reverse operation.

```
x2 -= ((x1<<4 ^ x1>>5) + x1) ^ (sum + *(key+(sum>>11&0x03)));
sum-=DELTA;
x1 -= ((x2<<4 ^ x2>>5) + x2) ^ (sum + *(key+(sum&0x03)));
```

FIGURE 3: XTEA FLOWCHART



Overview of Routines

XTEAEncode

This function encrypts the input data with the input key.

Syntax

`void XTEAEncode(unsigned long* data, unsigned char dataLength)`

Parameters

`data` – block of data to encrypt,
`dataLength` – the amount of the data to encrypt (must be a factor of 2)

Return Values

None

Pre-condition

data and key preloaded with the correct values

Side Effects

Values in data have changed to the encrypted version

Remarks

Note that the assembly version only encrypts 8 bytes at a time

Example: Usage of Encode

```
C
...
XTEAEncode(data,sizeof(data));

Assembly
...
mov1w   0x08
movwf   dataLength
mov1w   0x34
movwf   key+0
mov1w   0x12
movwf   key+1
...
lfsr    pointerNum,data
call    XTEAEncode
```

`TEADecode`

This function decrypts the input data with the input decryption key.

Syntax

`void XTEADecode(unsigned long* data, unsigned char dataLength)`

Parameters

`data` – block of data to decrypt,
`dataLength` – the amount of the data to decrypt (must be a factor of 2)

Return Values

None

Pre-condition

Data and key preloaded with the correct values

Side Effects

Values in data have changed to the decrypted version

Remarks

Note that the assembly version only decrypts 8 bytes at a time

Example: Usage of `Decode`

```
C
...
XTEADecode(data,sizeof(data));

Assembly
...
mov1w   0x08
movwf   dataLength
mov1w   0x34
movwf   key+0
mov1w   0x12
movwf   key+1
...
1fsr    pointerNum,data
call    XTEADecode
```

SKIPJACK

Overview/History/Background

SKIPJACK is an encryption algorithm that was developed in the early 1980s by the NSA for encrypting governmental documents. It remained classified SECRET until 1998 when it was declassified to the public. SKIPJACK uses an 80-bit key on a 64-bit block of data. The smaller key size of SKIPJACK has left it vulnerable to becoming obsolete must faster than AES, XTEA or any of the other encryption standards that support key sizes of 128 and larger.

Encryption

Like many other encryption algorithms, SKIPJACK is based on a Feistel network structure. SKIPJACK alternates between 2 rules over a Feistel network with a substitution table. The counter counts from 1 to 32 and is used to determine the round key by using the counter as an index into the crypto-variable.

FIGURE 4A: SKIPJACK® ENCRYPTION FLOWCHARTS – PAGE 1:

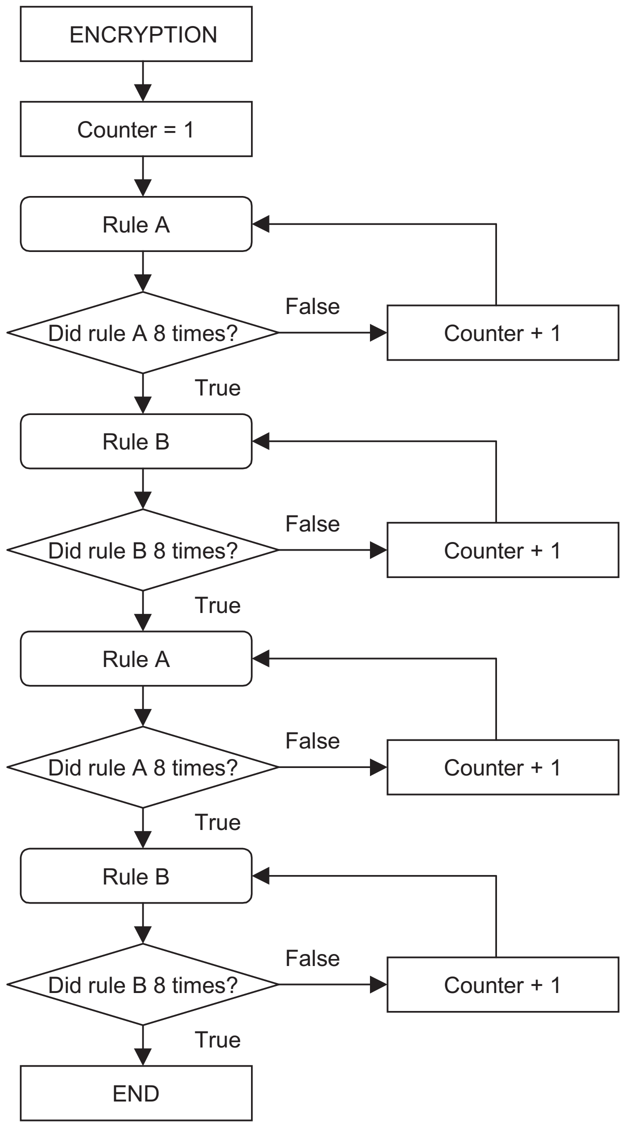
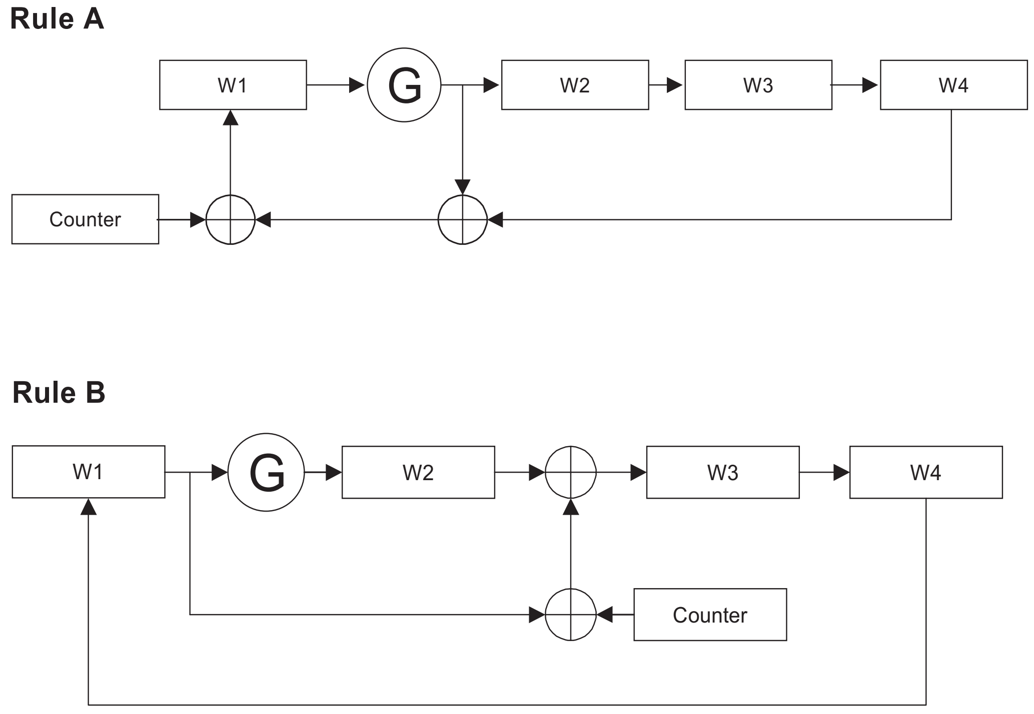


FIGURE 4B: SKIPJACK® ENCRYPTION FLOWCHARTS – PAGE 2



Note 1. (G) stands for G-permutation later described in this document.
W1,2,3 and 4 are the four words that are being encrypted.
2. ⊕ Stands for the XOR operation.

F-TABLE

The F-Table is a simple substitution table that is used both in the encryption and decryption cycles of SKIPJACK. (**Note:** All values are in hexadecimal)

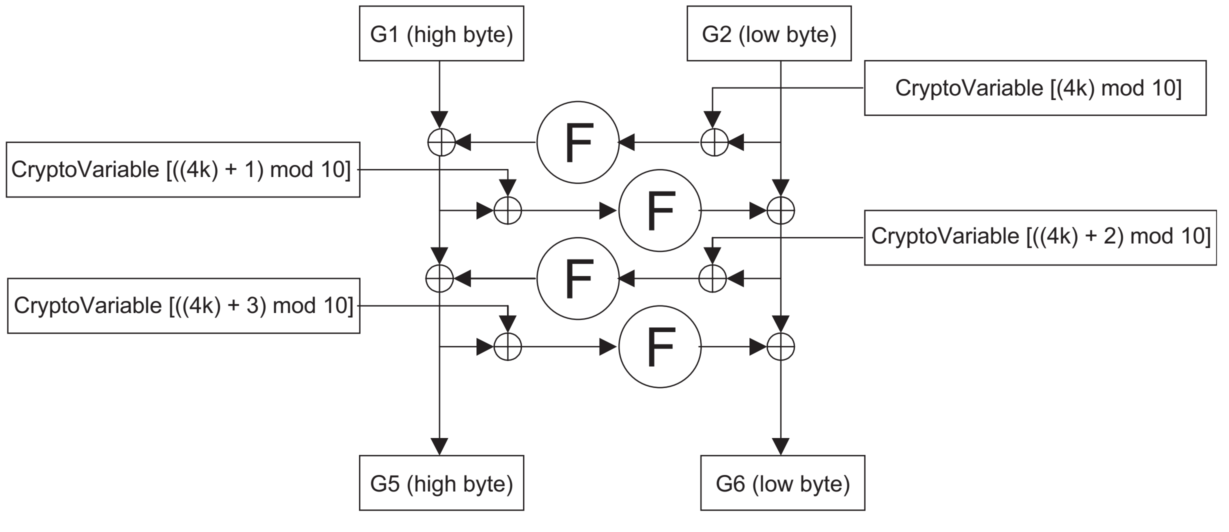
TABLE 6: F-TABLE

		y															
		00	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**	**A0**	**B0**	**C0**	**D0**	**E0**	**F0**
x	**00**	A3	d7	09	83	f8	48	f6	f4	b3	21	15	78	99	b1	af	f9
	01	E7	2d	4d	8a	ce	4c	ca	2e	52	95	d9	1e	4e	38	44	28
	02	0a	Df	02	a0	17	f1	60	68	12	b7	7a	c3	e9	fa	3d	53
	03	96	84	6b	ba	f2	63	9a	19	7c	ae	e5	f5	f7	16	6a	a2
	04	39	b6	7b	0f	c1	93	81	1b	ee	b4	1a	ea	d0	91	2f	b8
	05	55	b9	da	85	3f	41	bf	e0	5a	58	80	5f	66	0b	d8	90
	06	35	d5	c0	a7	33	06	65	69	45	00	94	56	6d	98	9b	76
	07	97	Fc	b2	c2	b0	fe	db	20	e1	eb	d6	e4	dd	47	4a	1d
	08	42	Ed	9e	6e	49	3c	cd	43	27	d2	07	d4	de	c7	67	18
	09	89	Cb	30	1f	8d	c6	8f	aa	c8	74	dc	c9	5d	5c	31	a4
	0A	70	88	61	2c	9f	0d	2b	87	50	82	54	64	26	7d	03	40
	0B	34	4b	1c	73	d1	c4	fd	3b	cc	fb	7f	ab	e6	3e	5b	a5
	0C	Ad	04	23	9c	14	51	22	f0	29	79	71	7e	ff	8c	0e	e2
	0D	0c	Ef	bc	72	75	6f	37	a1	ec	d3	8e	62	8b	86	10	e8
	0E	08	77	11	be	92	4f	24	c5	32	36	9d	cf	f3	a6	bb	ac
	0F	5e	6c	a9	13	57	25	b5	e3	bd	a8	3a	01	05	59	2a	46

G-PERMUTATION

The G-permutation operation is the Feistel network in SKIPJACK. It splits the upper and lower byte of the input word to encrypt. The network also makes use of the crypto-variable, which is an 80-bit long key. The crypto-variable is indexed with the number of iterations through the Feistel network. This number is considered to be mod 10 (so that the index wraps). The resulting byte is then XORed into the data and used to look up into the F-Table. The flow graph below illustrates this process. Note that (F) stands for a loop up into the F-Table. Also note that K = counter – 1.

FIGURE 5: G-PERMUTATION (k)



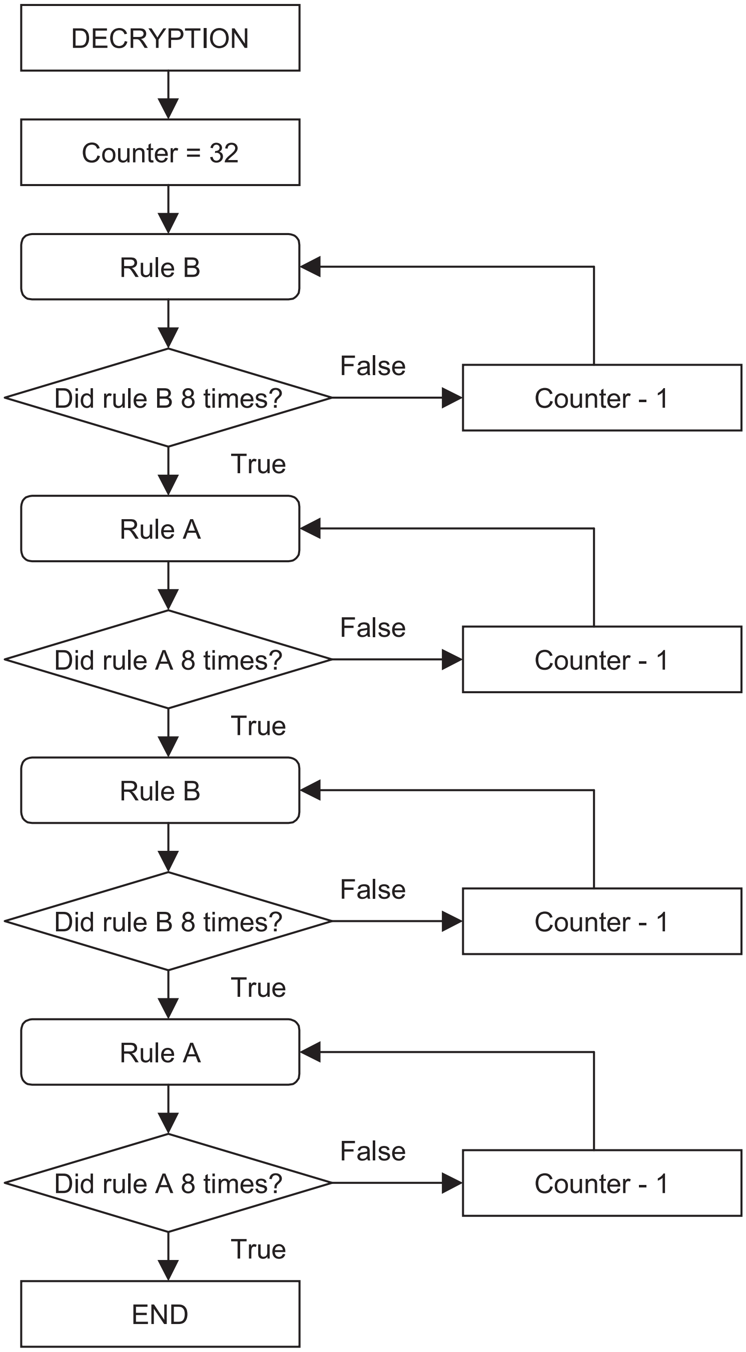
Note: ⊕ represents the XOR operation.

DECRYPTION

Decrypt Flowchart

The decryption flowchart is nearly identical to the encryption flow chart. The only differences being the counter starts at 32 and rule B starts before rule A.

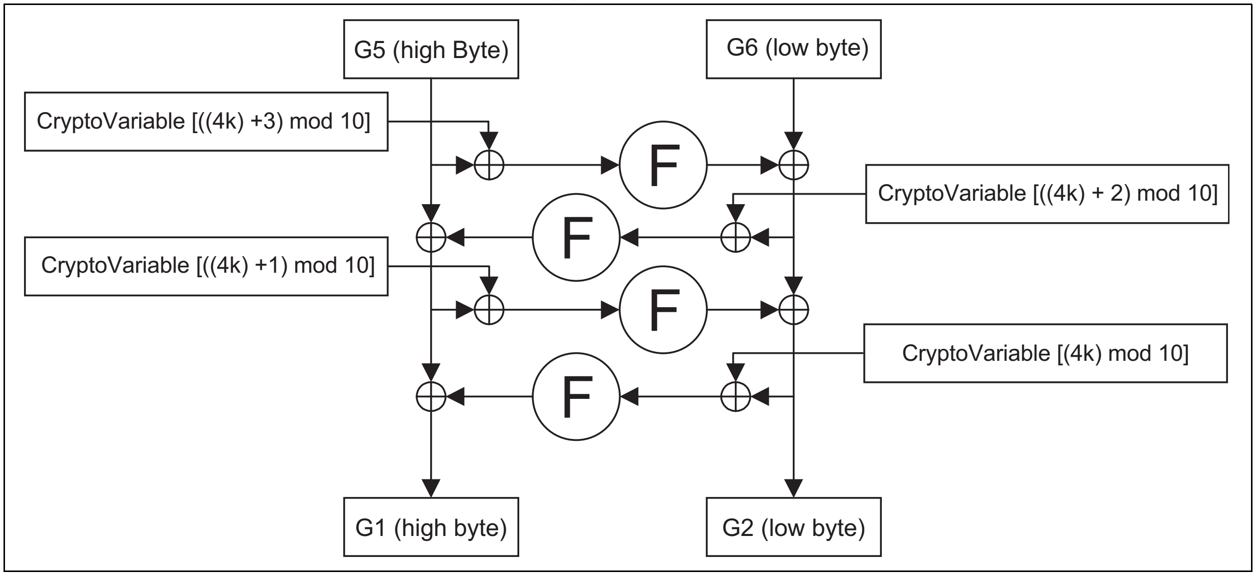
FIGURE 6: DECRYPTION FLOWCHART



INVERSE G-PERMUTATION

The inverse G-permutation is very similar to the G-permutation. Note that (F) stands for a loop up into the F-Table. Also note that K = counter – 1.

FIGURE 7: INVERSE G-PERMUTATION (k)



Overview of Routines

`SKIPJACKEncode`

This function encrypts the input data with the cryptovariable key.

Syntax

void Encode(unsigned int* data, unsigned char dataLength)

Parameters

`data` – block of data to encrypt,
`dataLength` – the amount of the data to encrypt (must be a factor of 4)

Return Values

None

Pre-condition

data preloaded with the correct values and a factor of 4 in size

Side Effects

Values in data have changed to the encrypted version

Remarks

None

Example: Usage of Encode

```
...
SKIPJACKEncode(data,sizeof(data));
...
```

`SKIPJACKDecode`

This function decrypts the input data.

Syntax

`void Decode(unsigned int* data, unsigned char dataLength)`

Parameters

`data` – block of data to decrypt,
`dataLength` – the amount of the data to decrypt (must be a factor of 4)

Return Values

None

Pre-condition

Data preloaded with the correct values and a factor of 4 in size

Side Effects

Values in data have changed to the decrypted version

Remarks

None

Example: Usage of `Decode`

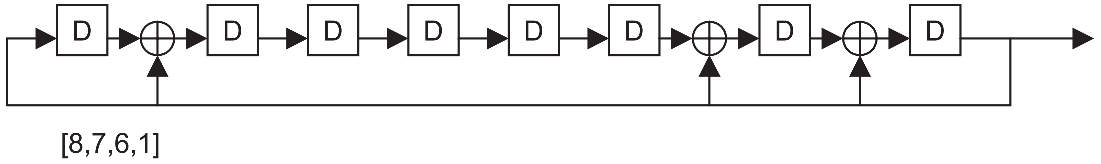
```
...
SKIPJACKDecode(data,sizeof(data));
...
```

PRBS XOR

Overview/History/Background

Pseudo-Random Binary Sequence (PRBS) generators can be used to create a sequence of bits that have very good randomness properties, though the sequences they generate are predictable and eventually repeated. Linear Feedback Shift Registers (LFSRs) can be used to create a PRBS. Specifically, this implementation uses the Galois implementation of LFSR to create the PRBS. The order of the sequence is controlled by where the feedback effects the nodes. PRBSs are used for very simple encryption. While this technique is not secure, it can be used as a fast and simple way to conceal data and deter attacks. This method may not be very secure when the data to encrypt is plain text (as there may be up to 3 consecutive bytes that do not get altered, leaving partial messages visible). The Berlekamp-Massey algorithm can be used to take an output cycle from a LFSR and compute the feedback tabs. From the feedback taps and the data, the key can then be calculated.

FIGURE 8: PRBS FLOWCHART



Note: ⊕ stands for the XOR operation.

The above LFSR is maximal with taps at 8,7,6 and 1. The tap at 8 is the output of the binary sequence. The output of the system always wraps around to the input of the system, as well as all of the other taps. The output of this sequence has good randomness properties. If you think of a binary sequence as a series of coin tosses, you would expect that you would land on heads (1) half of the time and tails (0) the other half. The probability of getting two heads in a row (11) would be (1/2)*(1/2)=1/4. The output binary sequences of LFSRs follow this pattern. The likelyhood of getting a run of length 1 (‘010’ or ‘101’) is 1/2. The probablity of getting a run of length 2 (‘0110’ or ‘1001’) is 1/4 and so on. This is just one of many randomness properties that LFSRs fulfill.

EXAMPLE 8: PRBS XOR EXAMPLES

Plain text: It was the best of times, it was the worst of times, ...
Key: [Char]
Plain hex: [0x4974207761732074][1]…
Cipher hex: [0x3BCD7351F2B5C30A]…

Key: [Char]
Feedback: 0b10000000000000000001000000001111
Plain hex: [0x4974207761732074][1]…
Cipher hex: [0x3BCD73517275A33A]…

Note 1: Only first block results shown.

SUGGESTIONS FOR IMPROVEMENT/VARIATION

Though LFSRs are succeptable to brute force attack due to the simplicity of their nature, there are improvements to the feedback system that can make them more difficult to crack.

- Run through two separate times with different keys and different feedbacks
- Change the module so that the data is operated on in a different (preferably non-linear) manner
  - Replace `data ^= key[0];` with `data ^= (key[3]&0b00110011) + (key[2]&0b10101010)`
  - Or Replace with `data ^= swapf(key[1]) ^ (key[2]&0b11000010)`
  - Or other combinations
- Additional taps for the 32 bit feedback system can be found at http://www.newwaveinstruments.com/resources/articles/m_sequence_linear_feedback_shift_register_lfsr/32stages.txt

Overview of Routines

`PRBSEncodeDecode`

This function encrypts and decrypts data using a PRBS generator.

Syntax

`void EncodeDecode(unsigned char* data, unsigned int dataLength)`

Parameters

`data` – block of data to decrypt,
`dataLength` – the amount of the data to decrypt

Return Values

None

Pre-condition

key preloaded with the correct value

Side Effects

Values in data have changed to the decrypted version

Remarks

None

Example: Usage of `Decode`

```
C
...
PRBSEncodeDecode(data,sizeof(data));

Assembly
...
lfsr    pointerNum.data
mov1w   0x08
movwf   dataLength
call PRBSEncodeDecode
```

PRECAUTIONS

With the exception of the pseudo-random number generator XOR encryption, a single bit error in the encrypted data can cause the destruction of the entire block of data once decrypted. Because of this phenomenon, extra caution should be taken to ensure that the data is correct. A checksum or verification byte embedded into the data can help ensure that the information in the data block remains intact after decryption. This feature can also be used to help prevent key theft. AES, SKIPJACK and XTEA are all relatively secure algorithms, as long as the encryption key remains hidden (even if the encryption algorithm is known). If the key becomes public knowledge, however, then the data is vulnerable. If errors are intentionally introduced into the encrypted data and the attackers are unaware of its existence, then knowing the encryption algorithm and the key will still not allow them to decrypt the data.

```
* i.e. - block[2] ^= block[3]; or  block[6] ^= 0x34;
```

EXAMPLE 9:

Plain text: 0x0102030405060708090A0B0C0D0E0F

Cipher Text: 0x0A940BB5416EF045F1C39458C653EA5A

Added bit errors: 0x0A940BB5416EF045F1C39458C653EA5B **(Least significant byte XORed with 0x01)**

Plain Text results: 0xF0FDF04AD3AFED45BB676E5B3B1685CD **(without correcting the bit error)**

Note: A single bit error in this example caused the destruction of the entire block of data.

RESULTS

TABLE 7: TIMING

C						
Method	**Iterations/ Variables Tested**	**Security**	**Encoding Cycles per Byte (approx.)**	**Decoding Cycles per Byte (approx.)**	**Inst./Sec Bytes/Sec (Encode)**	**Bytes/Sec (Decode)**
PRBS XOR encryption with skipping key	KeyJump = 1	Low	92-146 (**,*)	92-146 (**,*)	68493	68493
SkipJack®		High	2812	2817	3556	2550
XTEA (also referred to as TEAN or TEA-N)	16 iterations	High	1075 (*,***)	1280 (*,***)	9302	7813
XTEA (also referred to as TEAN or TEA-N)	32 iterations	High/ Very High	2133 (*,***)	2194 (*,***)	4688	4558
AES (Rijndael Algorithm)		High/ Very High	2153	2940 (****)	4645	3401
Assembly						
PRBS XOR encryption with skipping key	KeyJump = 1	Low	22-40 (*,**)	22-40 (*,**)	250000	250000
PRBS XOR encryption with skipping key	KeyJump = 5	Low	97-120 (*,**)	97-120 (*,**)	83333	83333
XTEA (also referred to as TEAN or TEA-N)	16 iterations	High	464 (*,***)	464 (*,***)	21552	21552
XTEA (also referred to as TEAN or TEA-N)	32 iterations	High/ Very High	926 (*,***)	926 (*,***)	10799	10799
AES (Rijndael Algorithm)		High/ Very High	367	620-687 (****)	27248	14556

* Depends on size of the data array.

** Depends on value of the key used.

*** Depends on iterations/jumps.

**** Depends if decode key is generated or hard-coded.

TABLE 8: USAGE

C		
Method	**ROM**	**RAM**
PRBS XOR encryption with skipping key	226	12*
SkipJack®	3616	34*
XTEA (also referred to as TEAN or TEA-N)	1950	38*
AES (Rijndael Algorithm)	6104	33*
Assembly		
PRBS XOR encryption with skipping key	48	11
XTEA (also referred to as TEAN or TEA-N)	962	25
AES (Rijndael Algorithm)	4400	45

* This figure does not include the memory holding the data to be encrypted.

Summary

Like many other applications, when choosing an encryption algorithm for an application, it becomes a balancing act between execution speed, code size and security. If the application emphasizes speed over security, then the PRBS algorithm is probably the best choice. If absolute security is needed no matter what the speed, cost or code size, then the best choices are XTEA with 32 or more rounds or AES. A balance between code size, execution speed and security is XTEA with 16 iterations. When developing a system that needs to securely talk to other systems, it will be necessary to implement the same encryption standard so the communication can be deciphered. AES is probably the most common implementation of the four discussed in this application note. When developing products that will remain in use for several decades, it is also important to remember that as technology and cryptography methods improve, the encryption algorithms implemented today will become weaker with time. Brute force attacks will become faster and methods of getting better then brute force attacks are constantly being developed. A more secure encryption implementation may be appropriate for applications where the firmware will remain in the field without updating its encryption alogorithm, but will remain in the field for long periods of time to help keep the data secure as long as possible.

REFERENCES

- **AN821**, *"Advanced Encryption Standard Using the PIC16XXX"* (DS00821), Caio Gubel, Microchip Technology,Inc.
- **Rijndael Web Page:** http://www.esat.kuleuven.ac.be/~rijmen/rijndael
- **NIST AES Web Page:** http://csrc.nist.gov/CryptoToolkit/aes/
- **Russell Web Page:** http://www-users.cs.york.ac.uk/~matthew/TEA/TEA.html
- **NIST DES/SKIPJACK Web Page:** http://csrc.nist.gov/cryptval/des.htm
- **New Wave Instruments Web Page:** http://www.newwaveinstruments.com/resources/articles/m_sequence_linear_feedback_shift_register_lfsr.htm

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's products as critical components in life support systems is not authorized except with express written approval by Microchip. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights.

Trademarks

The Microchip name and logo, the Microchip logo, Accuron, dsPIC, KEELOQ, microID, MPLAB, PIC, PICmicro, PICSTART, PRO MATE, PowerSmart, rfPIC, and SmartShunt are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

AmpLab, FilterLab, Migratable Memory, MXDEV, MXLAB, PICMASTER, 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, dsPICDEM, dsPICDEM.net, dsPICworks, ECAN, ECONOMONITOR, FanSense, FlexROM, fuzzyLAB, In-Circuit Serial Programming, ICSP, ICEPIC, MPASM, MPLIB, MPLINK, MPSIM, PICkit, PICDEM, PICDEM.net, PICLAB, PICtail, PowerCal, PowerInfo, PowerMate, PowerTool, rfLAB, rfPICDEM, Select Mode, Smart Serial, SmartTel and Total Endurance 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.

© 2005, 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

Microchip received ISO/TS-16949:2002 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona and Mountain View, California in October 2003. The Company's quality system processes and procedures are for its PICmicro® 8-bit MCUs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

MICROCHIP

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
Westford, MA
Tel: 978-692-3848
Fax: 978-692-3821

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

San Jose
Mountain View, CA
Tel: 650-215-1444
Fax: 650-961-0286

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

ASIA/PACIFIC

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 - 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 - Qingdao
Tel: 86-532-502-7355
Fax: 86-532-502-7205

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-2229-0061
Fax: 91-80-2229-0062

India - New Delhi
Tel: 91-11-5160-8631
Fax: 91-11-5160-8632

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

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

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

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

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

Taiwan - Hsinchu
Tel: 886-3-572-9526
Fax: 886-3-572-6459

EUROPE

Austria - Weis
Tel: 43-7242-2244-399
Fax: 43-7242-2244-393
Denmark - Ballerup
Tel: 45-4450-2828
Fax: 45-4485-2829

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

Germany - Ismaning
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

England - Berkshire
Tel: 44-118-921-5869
Fax: 44-118-921-5820

10/20/04