The BIG Cipher (Revision 1)

William Diehl

11/5/2018

The BIG cipher is a block cipher, capable of encryption and decryption, which operates on 128-bit blocks of plaintext or ciphertext, and uses a 128-bit secret key. The BIG cipher requires a number of iterations of rounds, or NUM_rounds. NUM_rounds = 12 for moderate security, and 18 for high security. This cipher is designed using a Feistel structure which operates separately on 64-bit left (high) and right (low) operands, which could provide opportunities for parallelization in 64-bit microprocessors. Diffusion is achieved through simple bitwise permutations, including half-word swaps and circular shifts. Confusion is achieved through four-bit non-linear substitutions.

Caution: This cipher is for educational purposes only. The security of this cipher has not been formally evaluated.

Basic operation

Strings of plaintext (PT), ciphertext (CT), and secret key (K) are 128 bits, or 16 bytes long. The order of a string W, consisting of PT, CT, or K, is as follows:

$$W_0 \parallel W_1 \parallel W_2 \parallel \cdots \parallel W_{15}$$

Where W_i is byte i of the relevant string. Furthermore, W is split into an upper half W_H and a lower half W_L, such that $W = W_H \parallel W_L$, and where each of the halves are 64 bits, or 8 bytes long.

The BIG encryption operation ENC is defined as follows, and is shown in Figure 1:

ALGORITHM 1: BIG ENCRYPTION

$CT = ENC(PT, K)$

1. $Partition\ W = PT\ into\ W_H\ and\ W_L$
2. $for\ i = 0\ to\ NUM_{ROUNDS} - 1$
3. $\quad W_H = W_H \oplus RK_{H\_i}$
4. $\quad W_H = SubBytes(W_H)$
5. $\quad W_L = AddRoundConstant(W_L, i)$
6. $\quad W_L = W_L \oplus RK_{L\_i}$
7. $\quad W_L = Perm_2(Perm_1(W_L))$
8. $\quad tmp = W_H$
9. $\quad W_H = W_H \oplus W_L$
10. $\quad W_L = tmp$
11. $return\ CT = W$

The BIG decryption operation DEC is defined as follows, and is shown in Figure 2:

ALGORITHM 2: BIG DECRYPTION

$PT = DEC(CT, K)$

1. $Partition\ W = CT\ into\ W_H\ and\ W_L$
2. $for\ i = NUM_{ROUNDS} - 1\ down\ to\ 0$

3. $tmp = W_L$
4. $W_L = W_H \oplus W_L$
5. $W_L = Perm_1{}^{-1}(Perm_2{}^{-1}(W_L))$
6. $W_H = InvSubBytes(tmp)$
7. $W_H = W_H \oplus RK_{H\_i}$
8. $W_L = W_L \oplus RK_{L\_i}$
9. $W_L = AddRoundConstant(W_L, i)$
10. $return\ PT = W$

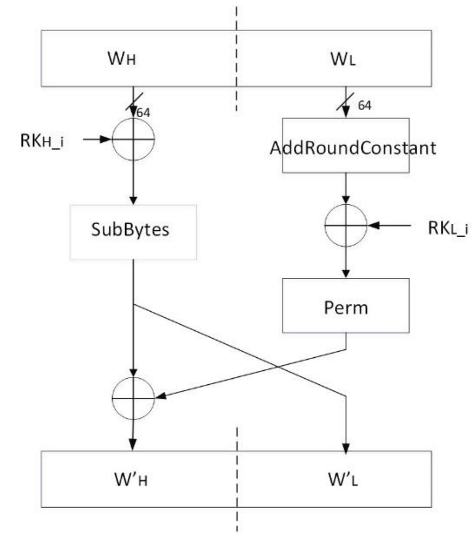

Figure 1 - Encryption

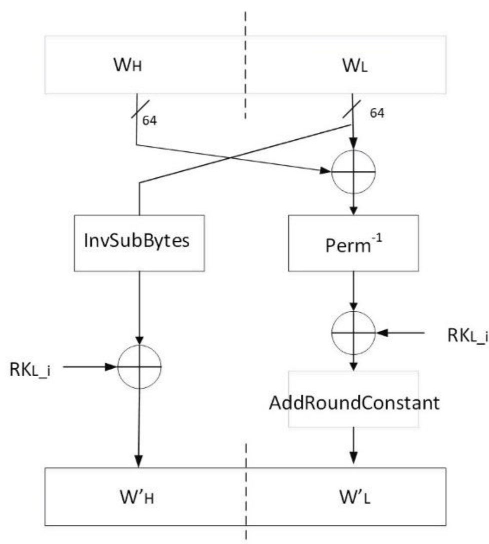

Figure 2 – Decryption

SubBytes

$S = SubBytes(Y)$ is defined as a string of 4-bit S-Boxes performed on each of 16 nibbles (64 bits) of Y as follows:

$$S = S_1 \parallel S_2 \parallel \cdots \parallel S_{15} = Sbox(Y_0) \parallel Sbox(Y_1) \parallel \cdots \parallel Sbox(Y_{15})$$

S_i, Y_i, are 4-bit nibbles, where $i = 0\ to\ 15$.

$Sbox(Y)$ is generated as $AY^{-1} + b$, where A is the binary matrix $\begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{matrix}$, Y^{-1} is the inverse of Y modulo $P(x) = x^4 + x + 1$, and b is the column vector constant $(1\ 1\ 0\ 0)^T$. The resulting values are shown in Table 1.

Table 1 – S-Boxes

Y	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
S	C	9	D	2	5	F	3	6	7	E	0	1	A	4	B	8

<u>InvSubBytes</u>

$\sigma = InvSubBytes(v)$ is defined as a string of 4-bit Inverse S-Boxes performed on each of 16 nibbles (64 bits) of v as follows:

$$\sigma = \sigma_0 \parallel \sigma_1 \parallel \cdots \parallel \sigma_{15} = InvSbox(v_0) \parallel InvSbox(v_1) \parallel \cdots \parallel InvSbox(v_{15})$$

σ_i, v_i, are 4-bit nibbles, where $i = 0\ to\ 15$.

$InvSbox(v)$ is generated as $[A^{-1}(v + b)]^{-1}$ where A^{-1} is the binary matrix $\begin{matrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{matrix}$, $[\ \cdot\ ]^{-1}$ is the inverse of $[\ \cdot\ ]$ modulo $P(x) = x^4 + x + 1$, and b is the column vector constant $(1\ 1\ 0\ 0)^T$. The resulting values are shown in Table 2.

Table 2 –Inverse S-Boxes

v	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
σ	A	B	3	6	D	4	7	8	F	1	C	E	0	2	9	5

<u>AddRoundConstant</u>

$Y = AddRoundConstant(X, i)$ adds a 7-bit value c_i at bits 20 down to 14 of the 64-bit argument X. Specifically: $Y = X_{63..21} \parallel X_{20..14} \oplus c_i \parallel X_{13..0}$, where $c_i = 0x5A * 2^i$,where multiplications are performed modulo $P(x) = x^8 + x^4 + x^3 + x + 1$ (i.e., the AES polynomial). The computed round constants for rounds 0 to 11 are shown in Table 3.

Table 3 – Round Constants

i	0	1	2	3	4	5	6	7	8	9	A	B
c_i	0x5A	0x34	0x73	0x66	0x57	0x35	0x71	0x62	0x5F	0x25	0x51	0x22

<u>Permutations</u>

$Y = Perm_1(X)$
Permutation 1 swaps the position of 16-bit words with one of its neighbors as follows

$Y_1 \parallel Y_0 \parallel Y_3 \parallel Y_2 = Perm_1(X_0 \parallel X_1 \parallel X_2 \parallel X_3)$, where X_i and Y_i are each two bytes, or 16 bits long. Note that $Perm_1 = {Perm_1}^{-1}$.

$Y = Perm_2(X)$

Permutation 2 is a 43-bit right circular shift (i.e., rotation) on a 64-bit operand as follows:

$$Y_{42..0} \parallel Y_{63..43} = Perm_2(X_{63..0})$$

$\upsilon = {Perm_2}^{-1}(\xi)$

The inverse of Permutation 2 is a 43-bit left circular shift (i.e., rotation) on a 64-bit operand as follows:

$\upsilon_{20..0} \parallel \upsilon_{63..21} = {Perm_2}^{-1}(\xi_{63..0})$.

<u>Round Keys</u>

In every round i of encryption and decryption, a round key is generated. Round keys for encryption are generated as follows, and as shown in Figure 3:

ALGORITHM 3 – ROUND KEY GENERATION DURING ENCRYPTION
$RK_i = generateRoundKey_{enc}(RK_{i-1})$
1. $if\ i = 0$
2. $\quad let\ RK_{i-1} = K$
3. $Partition\ RK_{i-1} = RK_{H_{i-1}} \parallel RK_{L_{i-1}}$
4. $tmp = Perm_1\left(RK_{L_i-1}\right)$
5. $RK_{Li} = tmp \oplus RK_{H_i-1}$
6. $RK_{H_i} = tmp$
7. $return\ RK_i$

Round keys for decryption are generated as follows, and as shown in Figure 4.:

ALGORITHM 4 – ROUND KEY GENERATION DURING DECRYPTION
$RK_{i-1} = generateRoundKey_{dec}(RK_i)$
1. $if\ i = NUM_{rounds} - 1$
2. $\quad let\ RK_i = K$
3. $Partition\ RK_i = RK_{Hi} \parallel RK_{Li}$

4. $tmp = \ Perm_1(RK_{Hi})$
5. $RK_{H_i-1} = RK_{H_i} \oplus RK_{L_i}$
6. $RK_{L_i-1} = tmp$
7. $return\ RK_{i-1}$

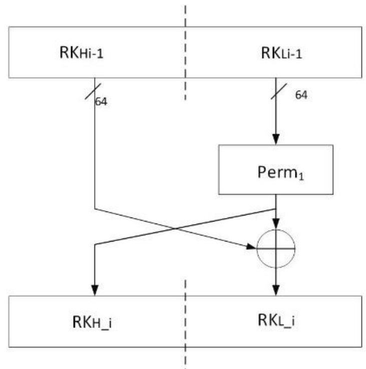

Figure 3 – Round Key generation for Encryption

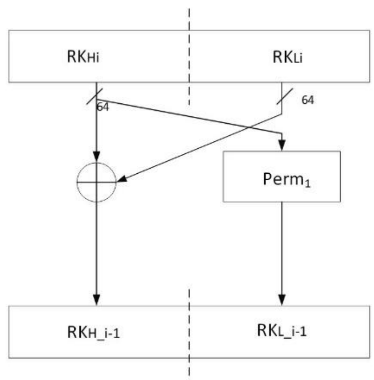

Figure 4 – Round Key generation for Decryption

<u>Test Vectors:</u>

Test Vectors are for NUM_rounds = 12

TV1:

PT = 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
KEY = 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00
CT = 0xa8 0x5e 0x68 0x2c 0x0e 0x14 0x0e 0x79 0x67 0x9e 0xc7 0x22 0x13 0x5b 0x6c 0x64

TV2:

PT = 0xde 0xad 0xbe 0xef 0xfe 0xfe 0xba 0xbe 0x12 0x34 0x56 0x78 0x9a 0xbc 0xde 0xf0
KEY = 0x01 0x23 0x45 0x67 0x89 0xab 0xcd 0xef 0xff 0xee 0xdd 0xcc 0xaa 0x99 0x88 0x77
CT = 0xda 0xb1 0xc4 0xc0 0xca 0x4d 0xcf 0x5b 0x50 0xea 0xf6 0x17 0xdb 0x92 0x55 0x13

Change Log:
11/5/18 – Changed BIG Encryption Algorithm Step 8 to $tmp = \ W_H$