Project 1

Write an AVR assembly program to code RC5.

Introduction

RC5 is designed by Rivest, which is a symmetric-key block cipher notable for its simplicity (http://people.csail.mit.edu/rivest/pubs/Riv94.pdf). A key feature of RC5 is the heavy use of data-dependent rotations. RC5 has a variable word size, a variable number of rounds, and a variable length secret key. This section presents the encryption, decryption, and key-expansion based on RC5 algorithm with the following parameters:

- the number of rounds ($\boldsymbol{r}$) equals **8-round**,
- the size of the expanded key table ($\boldsymbol{t} = 2*(r+1)$) equals **18 (S[0] … S[17])**,
- the word size in bits ($\boldsymbol{w}$) equals **16-bit**,
- the word size in bytes ($\boldsymbol{u} = w/8$) equals **2-byte**,
- the number of bytes in the **secrete key (*b*)** equals **12-byte**,
- the number of words in the **secrete key** ($\boldsymbol{c} = \lceil b/u \rceil$) equals **6-word**,
- the number of iterations of the key-expansion module ($\boldsymbol{n} = 3*\max(t, c)$) equals **54**,
- the constant $\boldsymbol{P_{16}}$ used in the key-expansion module equals $(\text{b7e1})_{16}$ or $\mathbf{(1011011111100001)_2}$, where $P_w = \text{Odd}((e-2)*2^w$ and $e = 2.718281828459$, and
- the constant $\boldsymbol{Q_{16}}$ used in the key-expansion module equals $(\text{9e37})_{16}$ or $\mathbf{(1001111000110111)_2}$, where $Q_w = \text{Odd}((\phi-1)*2^w$ and $\phi = 1.618033988749$.

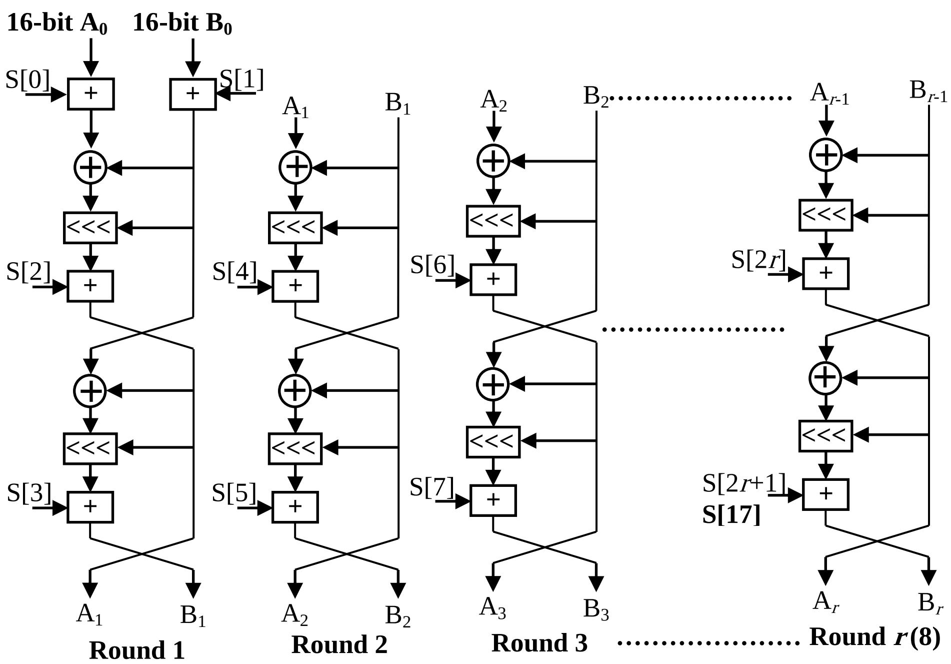

Fig. 1. Unrolled RC5 encryption algorithm with *r* rounds.

RC5 Encryption

The encryption module of RC5 accepts a block of data in two w-bit (16-bit) inputs A_0 and B_0, as shown in Figure 1. Moreover, it accepts the expanded key array $S[0{:}t\text{-}1]$ ($S[0{:}17]$), which stores the round keys generated by the key-expansion module. After r (8) rounds, the encryption module generates an encrypted block in two w-bit outputs (16-bit) A_r and B_r. Listing 1 presents the pseudo-code of the RC5 encryption algorithm, where the main operations are addition (+), XOR (⊕) and Rotate-Left (<<<).

Listing 1: RC5 encryption algorithm

$A_0 = A_0 + S[0]$
$B_0 = B_0 + S[1]$
for $i = 1$ to r ; $r = 8$
  $A_i = ((A_{i-1} \oplus B_{i-1}) \lll B_{i-1}) + S[2*i]$; $S[2], S[4], \ldots\ldots\ldots, S[16]$
  $B_i = ((B_{i-1} \oplus A_i) \lll A_i) + S[2*i+1]$; $S[3], S[5], \ldots\ldots\ldots, S[17]$

Figure 1 shows the block diagram of the unrolled encryption module of RC5 algorithm, where a single clock cycle is required for encrypting 2×w-bit (2×16-bit). It is clear that $(2r+2)\times w$-bit (18×16-bit) adds, $2r\times w$-bit (16×16-bit) shift-left, and $2r\times w$-bit (16×16-bit) XOR are needed for implementing the encryption module with unrolling r rounds.

RC5 Decryption

By reversing the operations, the decryption process can be easily derived from the encryption algorithm. Listing 2 presents the pseudo-code of the RC5 decryption algorithm, where the main operations are subtraction (-), XOR (⊕) and Rotate-right (>>>).

Listing 2: RC5 decryption algorithm

for $i = r$ downto 1
  $B_{i-1} = ((B_i - S[2*i+1]) \ggg A_i) \oplus A_i$
  $A_{i-1} = ((A_i - S[2*i]) \ggg B_{i-1}) \oplus B_{i-1}$
$B_0 = B_0 - S[1]$
$A_0 = A_0 - S[0]$

Like encryption, the unrolled decryption of RC5 algorithm is implemented to decrypt 2×w-bit in a single clock cycle. $(2r+2)\times w$-bit subtractors, $2r\times w$-bit shift-right, and $2r\times w$-bit XOR are needed for implementing the decryption module with unrolling r rounds.

RC5 Key-Expansion

The key-expansion module expands the user's secret key K to fill the expanded key array S, where S resembles an array of $t = 2*(r+1) = 18$ random binary words determined by K. The key-expansion algorithm uses two "magic constants": $P_w = \text{Odd}((e - 2)*2^w$ and $Q_w = \text{Odd}((\phi - 1)*2^w$, where e is the base of natural logarithms (2.718281828459), ϕ is the golden ratio

(1.618033988749), and Odd(x) is the odd integer nearest to x. For w = 16, P_{16} equals $(b7e1)_{16}$ or $(1011011111100001)_2$, and Q_{16} equals $(9e37)_{16}$ or $(1001111000110111)_2$.

As discussed in (http://people.csail.mit.edu/rivest/pubs/Riv94.pdf), the key-expansion algorithm consists of three simple algorithmic parts, see Listing 3. The first step is to copy the secret key $K[0: b-1]$ ($K[0: 11]$) into an array $L[0: c-1]$ ($L[0: 5]$), where b is the number of bytes in the secrete key, c is the number of words in the secrete key ($c = \lceil b/u \rceil = 12/2 = 6$), and u is the number of bytes per word. Note that any unfilled byte positions of L are zeroed. The second step is to initialize array S to a particular fixed (key-independent) pseudo-random bit pattern, using an arithmetic progression modulo 2^w determined by the "magic constants" P_w and Q_w, where $S[0] = P_w$ and $S[i] = S[i - 1] + Q_w$, for $i = 1$ to $t - 1$. Finally, the third step of key-expansion is to mix in the user's secret key in three passes over the arrays S and L. More precisely, due to the potentially different sizes of S and L, the larger array will be processed three times, and the other may be handled more times.

Listing 3: Key-expansion algorithm

```
// First step
    for i = b – 1 downto 0                  // b = 12 = # bytes in the Key, i = 11, 10, …, 0
        L[i / u] = (L[i / u] <<< 8) + K[i];  // u = 2 = # bytes per word
// Second step
    S[0] = Pw
    for i = 1 to t – 1                      // t = size of the expanded key table
        S[i] = S[i – 1] + Qw
// Third step
    i = j = 0
    A = B = 0
    do 3*max(t, c) times           // t = 18 , c = 6 = words per Key , do 3*18 = 54
        A = S[i] = (S[i] + A + B) <<< 3
        B = L[j] = (L[j] + A + B) <<< (A + B)
        i = (i + 1) mod(t)
        j = (j + 1) mod(c)
```