Specification of TC02

March 29, 2019

1 Cipher

The cipher takes a 64-bit key and 64-bit words and computes the ciphertext in 8 rounds. The words are divided into 16 4-bit nibbles, called the state of the cipher.

1.1 Round function

The round function of TC02 is very similar to the round function of AES. As in AES it consists of a sequential application of 4 layers. Add Roundkey (AR), Sub Cells (SC), Shift Rows (SR) and Mix Columns (MC). To clarify how each layer works we apply the first round to the plaintext:

0	0	0	0
0	0	0	0
F	E	D	C
B	A	9	8

with key: `0123 4567 89AB CDEF`.

1.1.1 Add Roundkey

In the Add Roundkey layer we XOR the 32 most significant bits of the key state with the cipher state. After adding the round key the state of the cipher is:

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

1.1.2 Sub Cells

In the Sub Cells (SC) layer we apply following substitution box (sbox) to every nibble of the iternal state.

$$S = [\texttt{2 4 5 6 1 A F 3 B E 0 7 9 8 C D}]$$

After applying substitution layer the state is:

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

1.1.3 Shift Rows

In the shift rows layer (SR) we rotate the nibbles in the rows by 0, 1, 2 and 3 places to the left.

After the shift rows layer the state is:

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

1.1.4 Mix Columns

In the Mix Columns (MC) layer we mix the nibbles in every column according to a matrix. The matrix is given by:

$$M = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

We multiply each column with the matrix. After the mix columns layer the state is:

A	D	8	A
2	6	E	D
9	3	5	8
8	9	D	C

1.2 Key schedule

Given a master key $K = k_0$, the keystate for the i-th round is given by applying the following recurrence relationship:

$$k_i = k_{i-1} \oplus \texttt{0x3} \lll 16$$

The roundkey for the i-th round rk_i is given by:

$$rk_i = k_i \ \texttt{\&}\ \texttt{0xFFFFFFFF00000000}$$

Where & denotes bitwise and.

1.3 Reference Implementation

```
#!/usr/bin/env python3
"""
    A very slow implementation of TC02

    64 bit state divided in 16 4-bit nibbles arranged in a matrix representation

    F E D C -> row_0
    B A 9 8 -> row_1
    7 6 5 4 -> row_2
    3 2 1 0 -> row_3

    Round function:
      add roundkey
      S-layer [same as TC01]
      AES shift rows (0, 1, 2, 3)
      Mix Columns:
      1 0 1 0
      0 1 1 0
      1 0 0 1
      0 0 1 0

    Key expansion:
        The Key has a 64-bit state like the cipher state. Keystate in round i
        K_i is computed as follows.

        K_{i+1} = (K_{i} ^ 0x3) >>> 16      # the rotation is given in bits

        The round key in the i-th round is given by:
        k_i = K{i} & 0xFFFFFFFF00000000     # mask the two most significant rows
"""
def get_rows(word):
    row_0 = (word >> 48) & 0xFFFF
    row_1 = (word >> 32) & 0xFFFF
    row_2 = (word >> 16) & 0xFFFF
    row_3 = (word >> 0) & 0xFFFF
    return row_0, row_1, row_2, row_3

def rotate_left(word, n, word_size=64):
    mask = 2**word_size - 1
    return ((word << n) & mask) | ((word >> (word_size - n) & mask))

def rotate_right(word, n, word_size=64):
    mask = 2**word_size - 1
    return ((word >> n) & mask) | ((word << (word_size - n) & mask))

def next_keystate(keystate):
    return rotate_right(keystate ^ 0x3, 16, 64)
```

```
def add_roundkey(word, keystate):
    return word ^ (keystate & 0xFFFFFFFF00000000)

def apply_sbox(word, sbox):
    """ apply the sbox to every nibble """
    word_new = 0
    for i in range(16): # 16 nibbles
        nibble = (word >> (i*4)) & 0xF # retrieve the ith nibble
        # insert the permuted nibble in the correct position
        word_new |= sbox[nibble] << i*4
    return word_new

def shift_rows(word):
    row_0, row_1, row_2, row_3 = get_rows(word)

    # apply the shiftrows transformation
    row_0 = row_0
    row_1 = rotate_left(row_1, 4, 16)
    row_2 = rotate_left(row_2, 8, 16)
    row_3 = rotate_left(row_3, 12, 16)

    # reconstruct the word
    new_word = row_0 << 48      # a |= b <==> a = a | b
    new_word |= row_1 << 32
    new_word |= row_2 << 16
    new_word |= row_3 << 0

    return new_word

def mix_columns(word):
    row_0, row_1, row_2, row_3 = get_rows(word) # split up the word into rows
    # Apply the mix culomns transformation and reconstruct the word
    new_word = (row_0 ^ row_2) << 48
    new_word |= (row_1 ^ row_2) << 32   # a |= b <==> a = a | b
    new_word |= (row_0 ^ row_3) << 16
    new_word |= row_2 << 0

    return new_word

def round_function(word, keystate):
    sbox = [0x2, 0x4, 0x5, 0x6, 0x1, 0xA, 0xF, 0x3,
            0xB, 0xE, 0x0, 0x7, 0x9, 0x8, 0xC, 0xD]

    word = add_roundkey(word, keystate)
    word = apply_sbox(word, sbox)
    word = shift_rows(word)
    word = mix_columns(word)

    return word
```

```
def encrypt(word, key, rounds=8):
    keystate = key
    for i in range(rounds):
        # apply the roundfunction to word
        word = round_function(word, keystate)
        # go to the next key state
        keystate = next_keystate(keystate)
    return word

if __name__ == "__main__":
    import sys
    import random

    if len(sys.argv) == 1:
        print("Error occured")
        exit()

    key = int(sys.argv[1], 16) % 2**64
    print("%016X"%(key))
    for i in range(16):
        word = random.getrandbits(64)
        cipher = encrypt(word, key, rounds=8)
        print("%016X %016X"%(word, cipher))
```