Specification of TC07

March 11, 2021

1 Cipher

The cipher takes a 64-bit key and 64-bit words and computes the ciphertext in 10 rounds. The words are divided into 16 4-bit nibbles, called the state of the cipher.

1.1 Round function

The round function of TC07 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:

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

with key: `89AB CDEF 0123 4567` .

1.1.1 Add Roundkey

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

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

1.1.2 Sub Cells

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

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

After applying substitution layer the state is:

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

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:

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

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:

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

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

DE7D 3C15 AAC7 74BA

D	E	7	D
3	C	1	5
A	A	C	7
7	4	B	A

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:

$$k_i = k_{i-1} \oplus \texttt{0xF3F3} \ggg 16$$

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

$$rk_i = k_i \ \texttt{\&} \ \texttt{0x00000000FFFFFFFF}$$

Where & denotes bitwise and.

2 Test Vectors

Plaintext	Key	Ciphertext
0000000000000000	0000000000000000	B8B825255959E1E1
000000000000002A	0000000000000001	938892A8785DEBD5
0000000000000000	0123456789ABCDEF	B98E1F711262ABEC

2.1 Reference Implementation

```
#!/usr/bin/env python3
"""
    A very slow implementation of TC07

    Author: Eran Lambooij
"""
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 ^ 0xF3F3, 16, 64)

def add_roundkey(word, keystate):
    return word ^ (keystate & 0x00000000FFFFFFFF)

def apply_sbox(word):
    """ apply the sbox to every nibble """
    word_new = 0
    sbox = [0xA, 0x5, 0x4, 0x2, 0x6, 0x1, 0xF, 0x3, 0xB, 0xE, 0x7, 0x0, 0x8, 0xD, 0xC
    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 ^ row_3) << 0

    return new_word

def round_function(word, keystate):
    word = add_roundkey(word, keystate)
    word = apply_sbox(word)
    word = shift_rows(word)
    word = mix_columns(word)

    return word

def encrypt(word, key, rounds=10):
    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

def create_test_vectors():
    state = 0xFEDCBA9800000000
    first_roundkey = 0x01234567

    print("%016X"%state)

    state = add_roundkey(state, first_roundkey)
    print("%016X"%state)
```

```
    state = apply_sbox(state)
    print("%016X"%state)

    state = shift_rows(state)
    print("%016X"%state)

    state = mix_columns(state)
    print("%016X"%state)

    print("%016X"%0, "%016X"%0,"%016X"%encrypt(0, 0))
    print("%016X"%42, "%016X"%1,"%016X"%encrypt(42, 1))
    print("%016X"%0, "%016X"%0x0123456789ABCDEF,
            "%016X"%encrypt(0, 0x0123456789ABCDEF))

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

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

    key = int(sys.argv[1], 16)
    # We seed the random generator with a hash of the key to get the same messages fo
    random.seed(hashlib.sha256(sys.argv[1].encode()).digest())

    for i in range(16):
        word = random.getrandbits(64)
        cipher = encrypt(word, key, rounds=4)
        print("%016X %016X"%(word, cipher))
```