Meet in the Middle

March 10, 2021

My optimizations

F	: 4
Sbox	: 26
Sbox 8−bit	: 12
Sbox 16−bit	: 12
Round function	: 14
Next roundkey	: 6
Encrypt	: 318
Encrypt unrolled	: 314

Sbox

```c
uint64_t* generate_8_bit_sbox(uint8_t* sbox){
    uint64_t i, word;
    uint64_t *sbox_8 = calloc(256, sizeof(uint64_t));
    for(word=0; word < 256; word++){
        sbox_8[word] = sbox[word & 0xF] |
                       (sbox[(word >> 4) & 0xF] << 4);
    }
    return sbox_8;
}
```

Sbox cont.

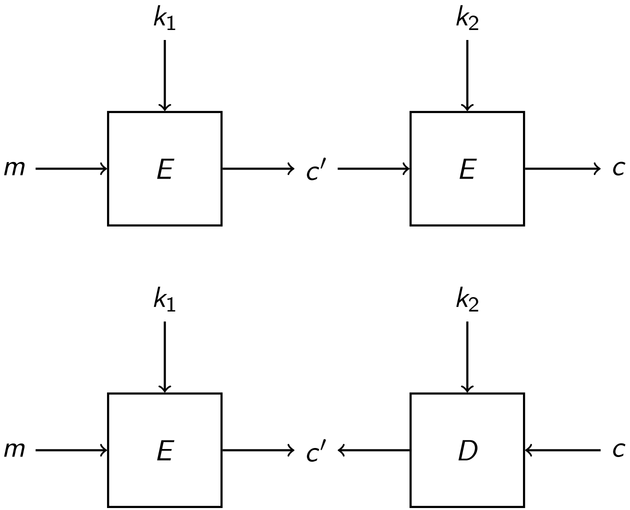
```
inline uint64_t apply_sbox8(uint64_t word){
    uint8_t block;
    int i;
    uint64_t word_new;

    word_new = 0;
    int shift = 0;
    for(i=0; i < 8; i++){
        word_new |= SBOX8[word & 0xFF] << shift;
        word >>= 8;
        shift += 8;
    }
    return word_new;
}
```

Last week's exercise (cont.)

- Use `inttypes.h` or `stdint.h` (`uint32_t, uint64_t, etc.`)
- In/output to the system is hexadecimal (and without the `0x`)
- I linked to a makefile tutorial on the website.
- Try all optimization levels, can sometimes save you some cycles.
- Try combining operations, for `TC01` I combined two 4-bit sboxes into an 8-bit sbox.
- You can also combine the linear layer and the sboxes (did not try).
- Use the reference implementation to check your own implementation.
- Please hand in reports in `pdf` format (I do not have Word).
- If you have any problems with the exercise, please ask questions!

MitM on 2DES (or 2AES)



Schoolbook MitM implementation on 2AES/2DES/2ETC

Algorithm 1 MitM attack

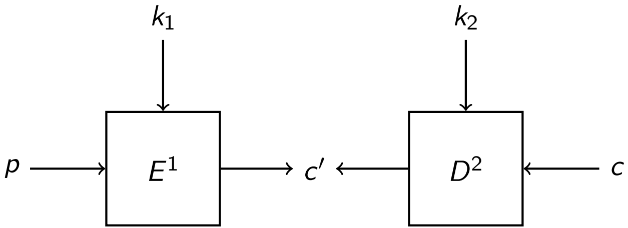
Given a plaintext ciphertext pair: (p, m)
Instantiate Hashmap H
for $k_1 \in K$ **do**
  $c' = E_{k_1}(p)$
  $H[c'] = k_1$
end for
for $k_2 \in K$ **do**
  $c' = D_{k_2}(c)$
  if $c' \in H$ **then**
    Output $(H[c'], k_2)$ as a probable key.
  end if
end for

Questions

Questions

- What is the running time of this attack?
- What is the memory consumption? Peak/Sustained?
- How many pairs do we need?

MitM



- Divide the cipher into two sub-ciphers E^1 and E^2 (and D^1, D^2 for decryption).
- Compute $c'_1 = E^1_{k_1}(p)$ for each $k_1 \in K_1$.
- Compute $c'_2 = D^2_{k_2}(p)$ for each $k_2 \in K_2$.
- If $c'_1 = c'_2$, then k_1 and k_2 are probable keys.

Schoolbook MitM implementation

Algorithm 2 MitM attack

For a plaintext ciphertext pair: (p, m)
for $k_1 \in K_1$ **do**
  $c' = E_{k_1}(p)$
  $H[c'] = k_1$
end for
for $k_2 \in K_2$ **do**
  $c' = D_{k_2}(c)$
  if $c' \in H$ **then**
    Output $(H[c'], k_2)$ as a probable key.
  end if
end for

Questions

Questions

- What is the running time of this attack?
- What is the memory consumption? Peak/Sustained?
- How many pairs do we need?

Schoolbook MitM implementation (2)

Algorithm 3 MitM attack

```
For a plaintext ciphertext pair: (p, m)
for k_c ∈ K_1 ∩ K_2 do
    Instantiate Hashmap H
    for k_1 ∈ K_1 \ K_2 do
        c' = E_{k_1+k_c}(p)
        H[c'] = k_1
    end for
    for k_2 ∈ K_2 \ K_1 do
        c' = D_{k_2+kc}(c)
        if c' ∈ H then
            Output (k_c, H[c'], k_2) as a probable key.
        end if
    end for
end for
```

Questions

Questions

- What is the running time of this attack?
- What is the memory consumption? Peak/Sustained?
- How many pairs do we need?

Finding MitM attacks

- For every key-bit/cell find the influence after r rounds.
- Find partial key sets K_1 and K_2 s.t. we have at least one common known bit in the middle

TC03

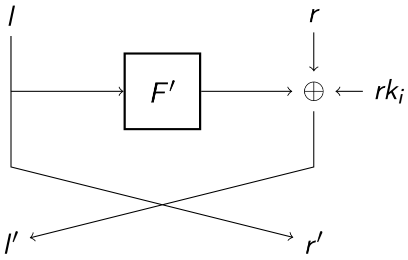
TC03 is a Feistel network with a block size of 8 bits, and a key size of 64-bit.

Round Function

$F'(w) = ((w \lll 1) \& (w \lll 2)) \oplus w$

Key Schedule

$K = k_0|k_1|k_2|k_3|\ldots|k_{15}$
The i-th round key is given by: $rk_i = k_{(i \mod 16)}$



Breaking TC03

Questions?

- How many rounds can we break of TC03?
- How many rounds of TC03 can we break practically?
- How to increase/decrease the resistance against MitM attacks?

MitM attack

Given we have found a MitM attack which guesses n_1 and n_2 key bits for the two partial ciphers and without loss of generality we assume that $n_1 < n_2$.

Forward Phase

- We have to build a filter, mapping 2^{n_1} words of size $n_1 + n_2$ bits to words of n_1 bits. Thus, mapping word to key.
- This takes $2^{n_1} \cdot I$ time, where I is the time to insert an element into the filter.
- This takes $O(2^{n_1} \cdot (n_2 + n_1))$ memory.

Backward Phase

- For each key guess in the backward phase we have to retrieve a value from the filter.
- This takes $2^{n_2} \cdot R$ time, where R is the time needed to retrieve a value from the filter.

Implementing MitM attacks

When implementing a MitM attack, there are three parts:

- Fast computation of the partial encryption/decryption
- Storing a filter
- Querying a filter

There are also two limiting factors:

- Time complexity
- Memory complexity

Partial encryption/decryption

- Expand 'key' into roundkeys $\rightarrow$ Fast key enumeration/schedule.
  - By using the key schedule.
  - Use an expansion function to expand masks and a value to round keys.
- Not computing the full state, but only a partial state.
- Fast implementation of the cipher.

Size of the filter

For effective filtering we need to have a properly sized filter. Given that we guess n_1 keybits in the forward direction and n_2 keybits in the backward direction the filter word size w needs to be at least:

$$w = n_1 + n_2$$

bits.

Size of the filter

For effective filtering we need to have a properly sized filter. Given that we guess n_1 keybits in the forward direction and n_2 keybits in the backward direction the filter word size w needs to be at least:

$$w = n_1 + n_2$$

bits.

Proof.

The probability of two random w-bit words being the same is 2^{-w}. Thus the probability that two random keys in the forward and backward direction produce the same w-bit state is: 2^{-w}. Since we are trying $2^{n_1+n_2}$ combinations of keys we get:

$$\begin{aligned} 2^{n_1} \cdot 2^{n_2} \cdot 2^{-w} &= 1 \\ 2^{n_1+n_2-w} &= 1 \\ n_1 + n_2 - w &= 0 \\ n_1 + n_2 &= w \end{aligned}$$

Storing a filter

We guess n_1 bits in the forward direction and n_2 bits in the backward direction. As seen before the filter word size is: $w = n_1 + n_2$ bits.

- Create a (hash)map H with 2^{n_1} elements mapping w-bit states to n_1-bit keys.
- For every key $k_1 \in \{0 \dots 2^{n_1}\}$ append k_1 to the set of keys in $H[E'_{k_1}(p)]$.

Storing a filter

We guess n_1 bits in the forward direction and n_2 bits in the backward direction. As seen before the filter word size is: $w = n_1 + n_2$ bits.

- Create a (hash)map H with 2^{n_1} elements mapping w-bit states to n_1-bit keys.
- For every key $k_1 \in \{0 \ldots 2^{n_1}\}$ append k_1 to the set of keys in $H[E'_{k_1}(p)]$.
- This takes: $2^{n_1} \cdot (w + n_1) \cdot C$ bits of RAM.
- Given a machine with 2^{40} bits of RAM and $C = 1$ what can we do?

Storing a filter

We guess n_1 bits in the forward direction and n_2 bits in the backward direction. As seen before the filter word size is: $w = n_1 + n_2$ bits.

- Create a (hash)map H with 2^{n_1} elements mapping w-bit states to n_1-bit keys.
- For every key $k_1 \in \{0 \ldots 2^{n_1}\}$ append k_1 to the set of keys in $H[E'_{k_1}(p)]$.
- This takes: $2^{n_1} \cdot (w + n_1) \cdot C$ bits of RAM.
- Given a machine with 2^{40} bits of RAM and $C = 1$ what can we do?

n_1	n_2	Filter size	RAM
20	44	$2^{26.4}$	0.01GB
24	40	$2^{30.4}$	0.17GB
28	36	$2^{34.4}$	2.83GB
32	32	$2^{38.6}$	52GB
36	28	$2^{42.6}$	549GB

Storing a filter (2)

- We can choose not to store the forward key this saves a (Memory) factor n_1, but adds 2^{n_1} time.
- If $w < 2 \cdot n_1$ we can store a bit array of size 2^w.
- We can use an ordinary list to store the (filter, key) pairs and sort after filling the list. This is better for IO complexity.
- Etc.

Questions

- Can we match on smaller than $(n_1 + n_2)$-bit words?
- What is the lower bound on memory if $n_1 = 32$ and $n_2 = 20$?
- And what is the lower bound for $n_1 = n_2 = 32$?

For next week

- Do this weeks exercise.
- Send me an email with what processor you have and the amount of RAM.
- If you get stuck, email me.