Timing attack against a DES software implementation

Zhang Yiyi

Lab Description:

In this lab, you will try to exploit a flaw in a **DES software** implementation which computation time depends on the input messages and on the secret key: its P permutation was implemented by a not-too-smart software designer who did not know anything about timing attacks (and not much about programming). The pseudo-code of his implementation of the P permutation is the following:

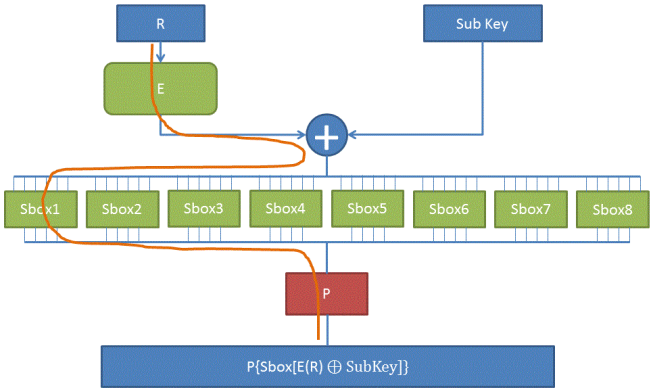
```
// Permutation table. Input bit #16 is output bit #1 and
// input bit #25 is output  bit #32.
p_table = {16,  7, 20, 21,
           29, 12, 28, 17,
            1, 15, 23, 26,
            5, 18, 31, 10,
            2,  8, 24, 14,
           32, 27,  3,  9,
           19, 13, 30,  6,
           22, 11,  4, 25};

p_permutation(val) {
  res = 0;                    // Initialize the result to all zeros
  for i in 1 to 32 {          // For all input bits #i (32 of them)
    if get_bit(i, val) == 1   // If input bit #i is set
      for j in 1 to 32        // For all 32 output bits #j (32 of them)
        if p_table[j] == i    // If output bits #j is input bit #i
          k = j;              // Remember output bit index
        endif
      endfor                  // output bit #k is now input bit #i
      set_bit(k, res);        // Set bit #k of result
    endif
  endfor
  return res;                 // Return result
}
```

Do you understand why, apart from being very inefficient, this implementation is a nice target for a timing attack? Open your lab report and explain what model of the computation time could be used by an attacker.

Lab comprehension:

First, the idea to attack the software is based on the flaw of this DES algorithm. From the code of the P permutation, we can see that some lines of codes are executed only when the input bit is 1, as a result the operation time is longer than when the input bit is 0. **So the computation time depends on the input messages and the secret key.** Since the time is data depending, we can exploit a timing attack to find the secret key.

In order to understand the DES algorithm more clearly, we have a picture below to help.



Here, *R*, which can be retrieved by the ciphertext, is expanded by *E*, XOR with the unknown *Subkey* then the resulting 6-bit value is used to output a 4-bit value from the substitution box (*Sbox*).

We know the ciphertext, so we can easily compute *R*. All the internal operations, i.e. *E* expansion, *Sbox* output values and *P* permutations are known. **The only unknown part is the *Sub key***. As highlighted above the execution time depends on the number of input bits to the block *P*. The number of input bits is denoted by *Hamming Weight*. We have 8 best subkeys in total, we try to find the 8 best subkeys based on the fact that **if the subkey is the best, then the computation time is linear related to *HW*.**

As usual intuition says that the above demonstration should not work because of a variety of reasons: we compute only 4 bits of one round and there are 16 rounds in DES, each round has 32 bit input into the *P* permutation, timing measurements are not perfect, the DES calculation done by PC is too quick.

However, thanks to the *law of big numbers* all of the noisy parts shall be averaged after certain number of encryptions. At the end all the noise will be averaged to a constant. So by trying all possible 6 bits of the first *SubKey* input to the *Sbox1* we can find the correct *SubKey* value.

Lab Steps:

1. **Generate a 64-bits DES secret key, build a table with 100000 pairs of 64-bits ciphertexts and the enciphering time.**

   $ ./ta_acquisition 100000
   100%
   Acquisitions stored in: ta.dat
   Secret key stored in: ta.key
   Last round key (hex):
   0x3cf34048397d

2. **Using timing attack to find the last round key.**

   The idea is to find the 8 best subkeys, then combine them together. I use three models to do the

attack, and manage to find the best one.

- **Basic attack**

  The idea of basic attack is that if the HW equals to 0, the computation is fast, then this computation time set as fast. If the HW equals to 4, the computation is slow, then set this computation time as slow. After we compare the average of the fast set and the average of slow set. **The bigger the gap, the better the subkey**. In the real practice, I find that if HW(0 and 1) set as **fast set**, HW(3 and 4) set as **slow set**, it works better.

- **Attack with Pearson Correlation Coefficient**

  To automate the comparison side-channel attacks use statistical tools known as distinguishers. These statistical tools show if dependency exists between two variables. For this concrete example we can apply the **Pearson Correlation Coefficient (PCC)**. PCC shows if there is any linear dependency between two variables, i.e. should two variables x and y have a relationship y = ax + b (a,b are constants), then PCC will give a result equal to 1, otherwise PCC will give a much smaller value. Here we set x as HW, y as Time. We look for the subkey that will make PCC result closest to 1.

- **Attack combined by basic attack and Pearson coefficient**

  There are limitations in both methods, but we can try to combine the advantages of both. I merge them by final_score = score(average(slow)- average(fast)) * PCC *PCC.

Lab result:

As I mentioned before, the goal is to retrieve the last round key **0x3cf34048397d** by timing attack.

1. Basic attack

```
-genis$ ./ta.py ta.dat 6569
Sbox 0
best current key guess 0x00000000003D
Sbox 1
best current key guess 0x00000000097D
Sbox 2
best current key guess 0x00000000397D
Sbox 3
best current key guess 0x00000048397D
Sbox 4
best current key guess 0x00000048397D
Sbox 5
best current key guess 0x00034048397D
Sbox 6
best current key guess 0x00F34048397D
Sbox 7
best current key guess 0x3CF34048397D
Average timing: 132460.277668
Last round key (hex):
```

0x3CF34048397D

2. Attack with Pearson coefficient

```
-genis$ ./ta.py ta.dat 3206
Sbox 0
best current key guess 0x00000000003D
Sbox 1
best current key guess 0x00000000097D
Sbox 2
best current key guess 0x00000000397D
Sbox 3
best current key guess 0x00000048397D
Sbox 4
best current key guess 0x00000048397D
Sbox 5
best current key guess 0x00034048397D
Sbox 6
best current key guess 0x00F34048397D
Sbox 7
best current key guess 0x3CF34048397D
Average timing: 132497.342514
Last round key (hex):
0x3CF34048397D
```

3. Attack combined by basic attack and Pearson Correlation Coefficient

```
-genis$ ./ta.py ta.dat 3174
Sbox 0
best current key guess 0x00000000003D
Sbox 1
best current key guess 0x00000000097D
Sbox 2
best current key guess 0x00000000397D
Sbox 3
best current key guess 0x00000048397D
Sbox 4
best current key guess 0x00000048397D
Sbox 5
best current key guess 0x00034048397D
Sbox 6
best current key guess 0x00F34048397D
Sbox 7
best current key guess 0x3CF34048397D
Average timing: 132504.229175
Last round key (hex):
0x3CF34048397D
```

According to the result, the **combine model** has the best result. I also tried with other data sets, the result is the same : the combine model has the best performance, but the improvement is not so bigger than the attack model only with **Pearson Correlation Coefficient**. The PCC model has much better performance than the basic attack.

Counter measure

The basic idea is to make the **computation time independent with data,** so I try to change the code when the input bit equal to 0, make the computation time similar to when the input bit is 1.

```
/* Applies the P permutation to a 32 bits word and returns the result as another
 * 32 bits word. */
uint64_t
des_p_ta(uint64_t val) {
  uint64_t res;
  int i, j, k;

  res = UINT64_C(0);
  k = 0;
  for(i = 1; i <= 32; i++) {
    if(get_bit(i, val) == 1) {
      for(j = 1; j <= 32; j++) {
        if(p_table[j - 1] == i) { /* C array index starts at 0, not 1 */
          k = j;
        }
      }
      res = set_bit(k, res);
    }
```

//---------- zhang yiyi----------

```
    else {
      for(j = 1; j <= 32; j++) {
        if(p_table[j - 1] == i) { /* C array index starts at 0, not 1 */
          k = j;
        }
      }
      res = unset_bit(k, res);
    }
```

//---------- zhang yiyi----------

```
  }
  return res;
}
```

- **Test result:**

```
-genis$ ./ta_acquisition 100000
100%
Acquisitions stored in: ta.dat
Secret key stored in:  ta.key
Last round key (hex):
0xaffbcffceb83

-genis$ ./ta.py ta.dat 100000
Sbox 0
best current key guess 0x000000000033
Sbox 1
best current key guess 0x000000000EF3
Sbox 2
best current key guess 0x000000018EF3
Sbox 3
```

```
best current key guess 0x0000008D8EF3
Sbox 4
best current key guess 0x0000208D8EF3
Sbox 5
best current key guess 0x0006E08D8EF3
Sbox 6
best current key guess 0x01D6E08D8EF3
Sbox 7
best current key guess 0x49D6E08D8EF3
Average timing: 210048.792006
Last round key (hex):
0x49D6E08D8EF3
```

- **Conclusion:**
  Even with 100000 ciphertexts, the attack model can't find any right subkey, which means the counter measure can prevent the DES software from timing attack.