INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO

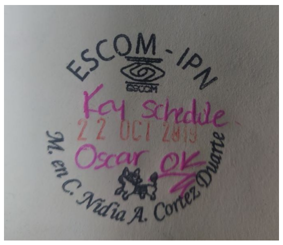
CRYPTOGRAPHY

Practice 7: DES - Key generation

STUDENT:
Rosas Hernandez Oscar Andres

PROFESSOR:
Msc. Nidia Asunción Cortez Duarte

Abstract:
In this practice we show a detail an implementation on the first part of the DES algorithm; the key generation



October 22, 2019

Contents

I. Literature review

A. DES

The DES (Data Encryption Standard) algorithm is the most widely used encryption algorithm in the world. For many years, and among many people, "secret code making" and DES have been synonymous. And despite the recent coup by the Electronic Frontier Foundation in creating a 220,000 machine to crack DES-encrypted messages, DES will live on in government and banking for years to come through a life- extending version called "triple-DES."
Since the creation of DES, many other algorithms (recipes for changing data) have emerged which are based on design principles similar to DES.

DES works on bits, or binary numbers–the 0s and 1s common to digital computers. Each group of four bits makes up a hexadecimal, or base 16, number. Binary "0001" is equal to the hexadecimal number "1", binary "1000" is equal to the hexadecimal number "8", "1001" is equal to the hexadecimal number "9", "1010" is equal to the hexadecimal number "A", and "1111" is equal to the hexadecimal number "F".
DES works by encrypting groups of 64 message bits, which is the same as 16 hexadecimal numbers. To do the encryption, DES uses "keys" where are also apparently 16 hexadecimal numbers long, or apparently 64 bits long. However, every 8th key bit is ignored in the DES algorithm, so that the effective key size is 56 bits. But, in any case, 64 bits (16 hexadecimal digits) is the round number upon which DES is organized.
For example, if we take the plaintext message "8787878787878787", and encrypt it with the DES key "0E329232EA6D0D73", we end up with the ciphertext "0000000000000000".
If the ciphertext is decrypted with the same secret DES key "0E329232EA6D0D73", the result is the original plaintext "8787878787878787".

DES is a block cipher–meaning it operates on plaintext blocks of a given size (64-bits) and returns ciphertext blocks of the same size. Thus DES results in a permutation among the 2^{64} possible arrangements of 64 bits, each of which may be either 0 or 1.
Each block of 64 bits is divided into two blocks of 32 bits each, a left half block L and a right half R. (This division is only used in certain operations.)
[1]

1) Key schedule: In cryptography, the so-called product ciphers are a certain kind of cipher, where the (de-)ciphering of data is typically done as an iteration of rounds.
The setup for each round is generally the same, except for round-specific fixed values called a round constant, and round-specific data derived from the cipher key called a round key. A key schedule is an algorithm that calculates all the round keys from the key.

DES has a key schedule in which the 56-bit key is divided into two 28-bit halves; each half is thereafter treated separately. In successive rounds, both halves are rotated left by one or two bits (specified for each round), and then 48 round key bits are selected by Permuted Choice 2 (PC-2) – 24 bits from the left half and 24 from the right.
The rotations have the effect that a different set of bits is used in each round key; each bit is used in approximately 14 out of the 16 round keys.

2) Process:

- The 64-bit key is permuted according to the following table, PC-1. Since the first entry in the table is "57", this means that the 57th bit of the original key K becomes the first bit of the permuted key K+. The 49th bit of the original key becomes the second bit of the permuted key. The 4th bit of the original key is the last bit of the permuted key.
  Note only 56 bits of the original key appear in the permuted key.

```
                  PC-1

57   49    41   33    25    17    9
 1   58    50   42    34    26   18
10    2    59   51    43    35   27
19   11     3   60    52    44   36
63   55    47   39    31    23   15
 7   62    54   46    38    30   22
14    6    61   53    45    37   29
21   13     5   28    20    12    4
```

[1]

- Next, split this key into left and right halves, C0 and D0, where each half has 28 bits.
- With C0 and D0 defined, we now create sixteen blocks Cn and Dn, 1<=n<=16. Each pair of blocks Cn and Dn is formed from the previous pair Cn-1 and Dn-1, respectively, for n = 1, 2, ..., 16, using the following schedule of "left shifts" of the previous block. To do a left shift, move each bit one place to the left, except for the first bit, which is cycled to the end of the block.

Iteration Number	Number of Left Shifts
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

[1]

This means, for example, C3 and D3 are obtained from C2 and D2, respectively, by two left shifts, and C16 and D16 are obtained from C15 and D15, respectively, by one left shift.

In all cases, by a single left shift is meant a rotation of the bits one place to the left, so that after one left shift the bits in the 28 positions are the bits that were previously in positions 2, 3,..., 28, 1.

- We now form the keys Kn, for 1<=n<=16, by applying the following permutation table to each of the concatenated pairs CnDn. Each pair has 56 bits, but PC-2 only uses 48 of these.

  Therefore, the first bit of Kn is the 14th bit of CnDn, the second bit the 17th, and so on, ending with the 48th bit of Kn being the 32th bit of CnDn.

PC-2

14	17	11	24	1	5
3	28	15	6	21	10
23	19	12	4	26	8
16	7	27	20	13	2
41	52	31	37	47	55
30	40	51	45	33	48
44	49	39	56	34	53
46	42	50	36	29	32

II. Sotware

- C++ 17 [3]

III. Procedure

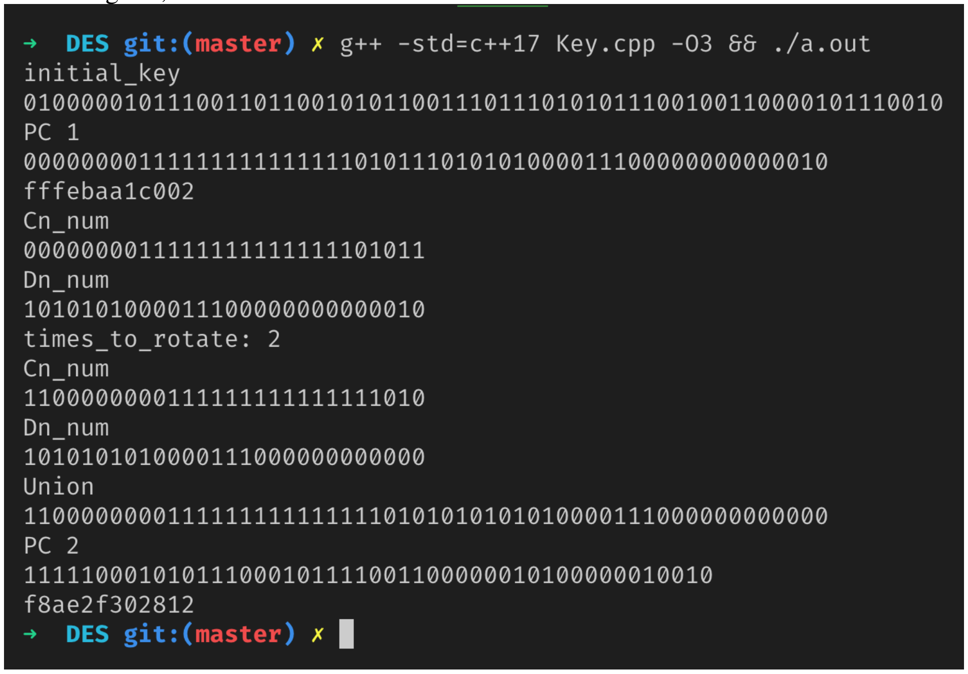
- Create the 64 bits integer that represent the 8 byte message
- Create a bitset of 64 bits (called result)
- Iterate of each position in the permutation map and save the value in the result (be careful about endian)
- Separate the result in 2 variables (28 bits) called Cn and Dn
- Calculate the number of times we have to rotate the parts
- Do the rotation using modular arithmetic
- Join the parts to the result using bit shifts and bitwise or
- Do the final permutation

IV. Memory map

- The initial permutation is an static array of 656 8-bits integers (448 bits)
- The information uses 2 64-bits integers (128 bits)
- Note that using constexpr (and C++11 or more) we may do all calculation in compilation time

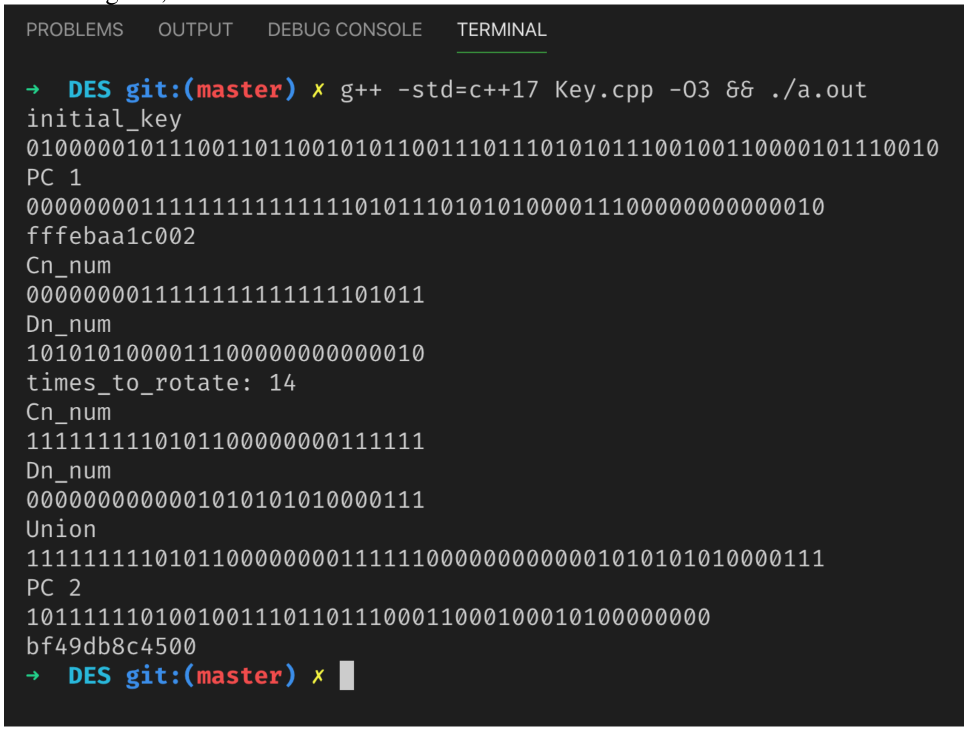
V. Results and evidence

Test: Asegurar, 2



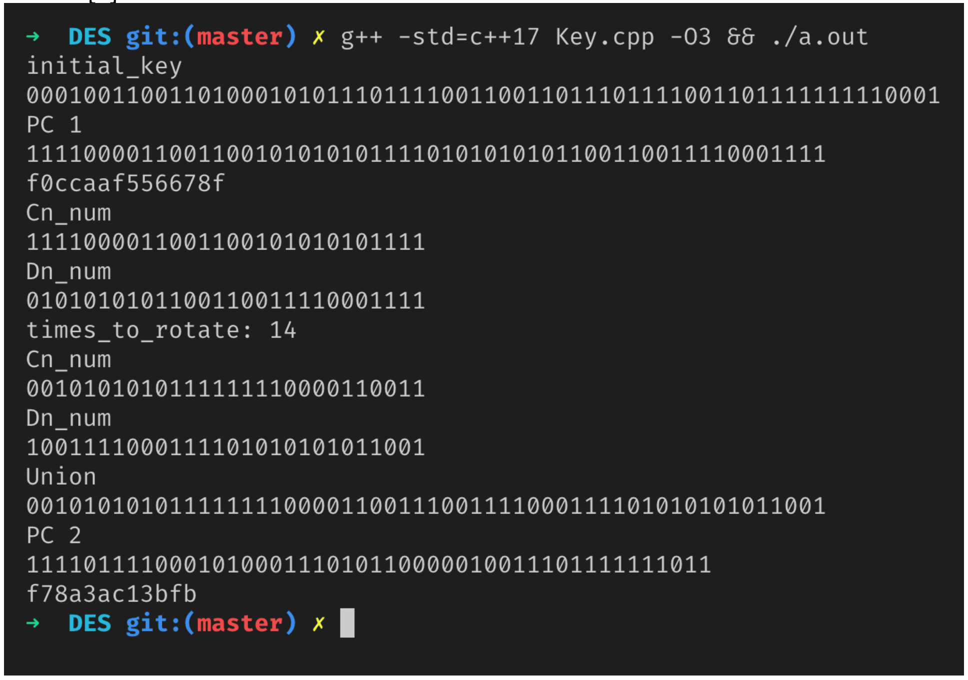
```
→  DES git:(master) ✗ g++ -std=c++17 Key.cpp -O3 && ./a.out
initial_key
0100000101110011011001010110011101110101011100100110000101110010
PC 1
00000000111111111111111010111010101000011100000000000010
fffebaa1c002
Cn_num
0000000011111111111111101011
Dn_num
1010101000011100000000000010
times_to_rotate: 2
Cn_num
1100000000111111111111111010
Dn_num
1010101010000111000000000000
Union
11000000001111111111111110101010101010000111000000000000
PC 2
111110001010111000101111001100000010100000010010
f8ae2f302812
→  DES git:(master) ✗
```

Test: Asegurar, 8



```
→  DES git:(master) ✗ g++ -std=c++17 Key.cpp -O3 && ./a.out
initial_key
0100000101110011011001010110011101110101011100100110000101110010
PC 1
00000000111111111111111010111010101000011100000000000010
fffebaa1c002
Cn_num
0000000011111111111111101011
Dn_num
1010101000011100000000000010
times_to_rotate: 14
Cn_num
1111111110101100000000111111
Dn_num
0000000000001010101010000111
Union
11111111101011000000001111110000000000001010101010000111
PC 2
101111110100100111011011100011000100010100000000
bf49db8c4500
→  DES git:(master) ✗
```

Test in [1]



```
➜  DES git:(master) ✗ g++ -std=c++17 Key.cpp -O3 && ./a.out
initial_key
0001001100110100010101110111100110011011101111001101111111110001
PC 1
11110000110011001010101011110101010101100110011110001111
f0ccaaf556678f
Cn_num
1111000011001100101010101111
Dn_num
0101010101100110011110001111
times_to_rotate: 14
Cn_num
0010101010111111110000110011
Dn_num
1001111000111101010101011001
Union
00101010101111111100001100111001111000111101010101011001
PC 2
111101111000101000111010110000010011101111111011
f78a3ac13bfb
➜  DES git:(master) ✗ 
```

VI. Discussion and Conclusions

We can see that implementing the key generation part is not trivial (using just the necessary memory, because of the way the computer store numbers / little and big endian), but it is more easy using a high level language.
I learned a lot about how to do bit manipulation in C++ and about little endian and all the little problems it may take.

VII. Code

See it in: https://github.com/SoyOscarRH/LibroCriptografia/tree/master/Code/DES

```cpp
#include <bitset>

using namespace std;

constexpr array<uint8_t, 64> initial_permutation {
    58, 50, 42, 34, 26, 18, 10, 2,  60, 52, 44, 36, 28, 20, 12, 4,  62, 54, 46, 38, 30, 22,
    14, 6,  64, 56, 48, 40, 32, 24, 16, 8,  57, 49, 41, 33, 25, 17, 9,  1,  59, 51, 43, 35,
    27, 19, 11, 3,  61, 53, 45, 37, 29, 21, 13, 5,  63, 55, 47, 39, 31, 23, 15, 7};

constexpr array<uint8_t, 64> final_permutation {
    40, 8,  48, 16, 56, 24, 64, 32, 39, 7,  47, 15, 55, 23, 63, 31, 38, 6,  46, 14, 54, 22,
    62, 30, 37, 5,  45, 13, 53, 21, 61, 29, 36, 4,  44, 12, 52, 20, 60, 28, 35, 3,  43, 11,
    51, 19, 59, 27, 34, 2,  42, 10, 50, 18, 58, 26, 33, 1,  41, 9,  49, 17, 57, 25};

constexpr array<uint8_t, 56> permuted_choice_1 {
    57, 49, 41, 33, 25, 17, 9,  1,  58, 50, 42, 34, 26, 18, 10, 2,  59, 51, 43,
    35, 27, 19, 11, 3,  60, 52, 44, 36, 63, 55, 47, 39, 31, 23, 15, 7,  62, 54,
    46, 38, 30, 22, 14, 6,  61, 53, 45, 37, 29, 21, 13, 5,  28, 20, 12, 4};

constexpr array<uint8_t, 56> permuted_choice_2 {
    14, 17, 11, 24, 1,  5,  3,  28, 15, 6,  21, 10, 23, 19, 12, 4,  26, 8,  16, 7,  27, 20, 13, 2,
    41, 52, 31, 37, 47, 55, 30, 40, 51, 45, 33, 48, 44, 49, 39, 56, 34, 53, 46, 42, 50, 36, 29, 32};

template <size_t num_bits, typename message_t, typename permutation_t>
auto permute(message_t message, permutation_t permutation) {
  const auto data = bitset<num_bits> {message};
  auto result = bitset<num_bits> {0};

  auto i = 0;
  for (const auto pos : permutation) result[i++] = data[pos - 1];

  return result.to_ullong();
}

template <size_t in_bits, size_t out_bits, typename message_t, typename permutation_t>
auto permute_endian(message_t message, permutation_t permutation) {
  const auto data = bitset<in_bits> {message};
  auto result = bitset<out_bits> {0};

  auto x = data.to_string();
  auto i = out_bits;
  for (const auto pos : permutation) {
    result[--i] = data[in_bits - pos];
  }

  return result.to_ullong();
}
```

```
#include <array>
#include <bit>
#include <bitset>
#include <cstdint>
#include <iostream>
#include <numeric>
#include <string>

#include "Utility.cpp"
#include "Permutation.cpp"

using namespace std;

template <size_t num_bits>
auto right_rotate(const bitset<num_bits> data, int times) {
  auto result = bitset<num_bits> {};

  for (int i = 0; i < num_bits; i++) {
    result[(num_bits + i - times) % num_bits] = data[i];
  }

  return result;
}

auto generate_key(const uint64_t initial_key, const int key_id) {
  constexpr static array<uint8_t, 16> rounds {1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1};

  cout << "initial_key" << endl;
  print_bin(initial_key);

  cout << "PC 1" << endl;
  const auto permuted_message = permute_endian<64, 56>(initial_key, permuted_choice_1);
  print_bin<56>(permuted_message);
  print_hex(permuted_message);

  const auto data = bitset<56> {permuted_message};
  auto Cn = bitset<28> {0}, Dn = bitset<28> {0};

  auto index = 56;
  for (auto i = 27; i >= 0; i--) Cn[i] = data[--index];
  for (auto i = 27; i >= 0; i--) Dn[i] = data[--index];

  cout << "Cn_num" << endl;
  print_bin<28>(Cn.to_ullong());

  cout << "Dn_num" << endl;
  print_bin<28>(Dn.to_ullong());

  const auto times_to_rotate = accumulate(begin(rounds), begin(rounds) + key_id, 0);
  cout << "times_to_rotate: " << dec << times_to_rotate << endl;

  Cn = right_rotate<28>(Cn, times_to_rotate);
  Dn = right_rotate<28>(Dn, times_to_rotate);

  cout << "Cn_num" << endl;
  print_bin<28>(Cn.to_ullong());

  cout << "Dn_num" << endl;
  print_bin<28>(Dn.to_ullong());

  auto result = Cn.to_ullong();
  result = (result << 28) | Dn.to_ullong();

  cout << "Union" << endl;
  print_bin<56>(result);

  result = permute_endian<56, 48>(result, permuted_choice_2);

  cout << "PC 2" << endl;
  print_bin<48>(result);

  return result;
}

auto main() -> int {
  //const auto key = create_number("Asegurar");
  const auto key = 0b00010011'00110100'01010111'01111001'10011011'10111100'11011111'11110001;
  const auto result = generate_key(key, 8);

  cout << hex << result << endl;
}
```

```
#include <array>
#include <bitset>
#include <cstdint>
#include <iostream>

using namespace std;

template <size_t T = 64>
auto print_bin(const uint64_t data) {
  cout << bitset<T> {data} << endl;
}

auto print_hex(const uint64_t data) { cout << hex << data << endl; }

auto print_bin_spaces(const uint64_t data, const char separator = ' ') {
  auto to_show = bitset<64> {data}.to_string();

  for (auto it = begin(to_show); distance(it, end(to_show)) > (4 + 2); ++it) {
    advance(it, 4);
    it = to_show.insert(it, separator);
  }

  cout << to_show << endl;
}

template <typename num = uint64_t>
auto create_number(string key_text) -> num {
  union key_t {
    char text[sizeof(num)];
    num id;
  };

  std::reverse(begin(key_text), end(key_text));

  auto key = key_t {};
  strcpy(key.text, key_text.c_str());

  return key.id;
}
```

References

[1] The DES Algorithm Illustrated, by J. Orlin Grabbe *http://page.math.tu-berlin.de/ kant/teaching/hess/krypto-ws2006/des.htm*.
[2] Handbook of Applied Cryptography, Alfred J. Menezes, Paul C. van Oorschot, Scott A. Vanstone
[3] *https://en.cppreference.com/w/cpp/algorithm*.