INSTITUTO POLITÉCNICO NACIONAL
ESCUELA SUPERIOR DE CÓMPUTO

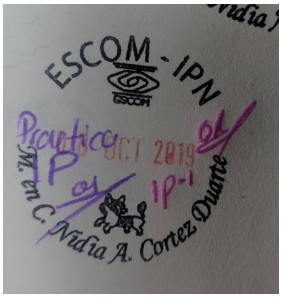
CRYPTOGRAPHY

Practice 6: DES - Initial Permutation

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 initial permutation and its inverse process (IP^{-1})



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) The Initial Permutation: There is an initial permutation IP of the 64 bits of the message data M. This rearranges the bits according to the following table, where the entries in the table show the new arrangement of the bits from their initial order. The 58th bit of M becomes the first bit of IP. The 50th bit of M becomes the second bit of IP. The 7th bit of M is the last bit of IP.

IP

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

2) Why they are important (permutations): The initial and final permutation have no influence on security (they are unkeyed and can be undone by anybody).
The usual explanation is that they make implementation easier in some contexts, namely a hardware circuit which receives data over a 8-bit bus: it can accumulate the bits into eight shift registers, which is more efficient (in terms of circuit area) than a single 64-bit register. This process "naturally" performs the initial permutation of DES.
In more details: Suppose that you are designing a hardware circuit which should do some encryption with DES, and receives data by blocks of 8 bits. This means that there are 8 "lines", each yielding one bit at each clock. A common device for accumulating data is a shift register: the input line plugs into a one-bit register, which itself plugs into another, which plugs into a third register, and so on. At each clock, each register receives the contents from the previous, and the first register accepts the new bit. Hence, the contents are "shifted".
With an 8-bit bus, you would need 8 shift registers, each receiving 8 bits for an input block. The first register will receive bits 1, 9, 17, 25, 33, 41, 49 and 57. The second register receives bits 2, 10, 18,... and so on. After eight clocks, you have received the complete 64-bit block and it is time to proceed with the DES algorithm itself.

IP^{-1}

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

If there was no initial permutation, then the first step of the first round would extract the "left half" (32 bits) which, at that point, would consist of the leftest 4 bits of each of the 8 shift registers. The "right half" would also get bits from the 8 shift registers.
If you think of it as wires from the shift registers to the units which use the bits, then you end up with a bunch of wires which heavily cross each other. Crossing is doable but requires some circuit area, which is the expensive resource in hardware designs.
However, if you consider that the wires must extract the input bits and permute them as per the DES specification, you will find out that there is no crossing anymore. In other words, the accumulation of bits into the shift registers inherently performs a permutation of the bits, which is exactly the initial permutation of DES. By defining that initial permutation, the DES standard says: "well, now that you have accumulated the bits in eight shift registers, just use them in that order, that's fine".
The same thing is done again at the end of the algorithm.
Remember that DES was designed at a time when 8-bit bus where the top of the technology, and one thousand transistors were an awfully expensive amount of logic. [2]

II. Sotware

- C++ 17 [4]

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
- Return the result

IV. Memory map

- The initial permutation is an static array of 64 8-bits integers (512 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

```
→  DES git:(master) ✗ g++ -std=c++17 InitialPermutation.cpp -O3 && ./a.out
initial input
0100 0100 0110 1001 0110 0001 0110 1101 0110 0001 0110 1110 0111 0100 0110 0101

permuted
1111 1111 0100 0000 1110 1001 1001 1110 0000 0000 1111 1110 0010 1010 0010 0000

recovered
0100 0100 0110 1001 0110 0001 0110 1101 0110 0001 0110 1110 0111 0100 0110 0101
→  DES git:(master) ✗ g++ -std=c++17 InitialPermutation.cpp -O3 && ./a.out
initial input
0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111

permuted
1100 1100 0000 0000 1100 1100 1111 1111 1111 0000 1010 1010 1111 0000 1010 1010

recovered
0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111
```

We can check the output that to [1]

VI. Discussion and Conclusions

We can see that implementing the initial permutation is trivial using a high level language, and that the reason for its existance is rather for electronic reasons.
I learned a lot about how to do bit manipulation in C++ and it was my first opint of contact with DES, so really good.

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();
}
```

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

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

auto main() -> int {
  const auto message = 0x0123456789ABCDEFull;
  constexpr auto num_bits = 64;
  //const auto message =  create_number("Diamante");

  cout << "initial input" << endl;
  print_bin_spaces(message);

  cout << endl << "permuted" << endl;
  auto permuted_message = permute<num_bits>(message, initial_permutation);
  print_bin_spaces(permuted_message);

  cout << endl << "recovered" << endl;
  auto recovered_message = permute<num_bits>(permuted_message, final_permutation);
  print_bin_spaces(recovered_message);

  return 0;
}
```

```cpp
#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] Crypto Stack *https://crypto.stackexchange.com/questions/3/what-are-the-benefits-of-the-two-permutation-tables-in-des*.
[3] Handbook of Applied Cryptography, Alfred J. Menezes, Paul C. van Oorschot, Scott A. Vanstone
[4] *https://en.cppreference.com/w/cpp/algorithm*.