Trabalho 1 - Segurança Computacional

Implementação da encriptação e decriptação do algoritmo S-DES

Emerson Luiz Cruz Junior – 231003531

. Introdução:

O trabalho consistia na implementação do algoritmo S-DES, uma versão simplificada do algoritmo DES apresentando as mesmas propriedades, porém com algumas operações diferentes para simplificar o algoritmo. A implementação desse algoritmo segue as especificações definidas no apêndice G S-DES (1) a implementação detalhada se encontra em um repositório do GitHub presente em (2). Para a realização desse trabalho adota-se como chave padrão, texto em claro padrão e texto cifrado padrão os respectivos valores: 1010000010, 11010111 e 10101000.

. Funções auxiliares:

Para simplificar a implementação das operações de XOR e de permutação, 3 funções que convertem um inteiro para sua representação no formato de uma string binaria foram implementadas, bem como uma função para converter uma string binaria para um inteiro. Em Code (1) é possível observar a implementação de uma dessas 3 funções, todas seguem o mesmo princípio mudando apenas o tamanho da representação da string binaria:

```
string intToBinStr2(int num) {
    string binStr = "";
    while(binStr.size() != 2){
        binStr.push_back((char)((num % 2) + '0'));
        num /= 2;
    }
    reverse(binStr.begin(), binStr.end());
    return binStr;
}
```

Code.(1): exemplo de uma das funções de conversão de um inteiro para a representação em formato de String binaria

Enquanto em Code (2) é possível visualizar a função que realiza a conversão de uma string binaria para inteiro:

```
int binStrToInt(string binStr) {
    int aux = 0, pot2 = 1;
    for(int i = binStr.size()-1; i > -1; i--){
        char x = binStr[i];
        if(x == '1') aux += pot2;
        pot2 *= 2;
    }
    return aux;
}
```

Code (2): Implementação da função que converte uma string binaria para inteiro

. Geração de Chave:

Para a geração das subchaves era necessário realizar uma permutação nos 10 bits da chave inicial seguindo a seguinte tabela presente nas especificações do algoritmo no apêndice G S-DES (1):

P10									
3	5	2	7	4	10	1	9	8	6

Fig. (1): Função de permutação dos 10 bits da chave inicial

Após isso, a chave é dividida em dois blocos de 5 bits, cada bloco passa por um deslocamento a esquerda de 1 bit, a partir do deslocamento a primeira subchave é construída usando a seguinte tabela também presente em (1):

P8							
6	3	7	4	8	5	10	9

Fig. (2): Tabela dos bits selecionados para formarem a subchave de 8 bits

Por fim, os bits são, novamente, separados em dois blocos, cada bloco passa por um deslocamento a esquerda de 2 bits e então seguindo o mesmo critério da primeira subchave a segunda subchave é construída. Para gerar as subchaves usadas na decriptação é preciso apenas inverter as subchaves geradas para a encriptação, isso será melhor desenvolvido na parte de decriptação. A seguir é possível visualizar a implementação da função, que gera a chave, e seu resultado para os padrões adotados:

```
vector<string> genRoundKey(bool decrypt = false, string k
= key) {
    string permKey = "";
    vector<string> roundsKeys(2, "");
    for(int idx = 1; idx <= 10; idx++) {
        switch(idx) {
            case 1 :
                permKey.push_back(k[2]);
                break;
            case 2 :
```

```
                permKey.push_back(k[4]);
                break;i
            case 3 :
                permKey.push_back(k[1]);
                break;
            case 4 :
                permKey.push_back(k[6]);
                break;
            case 5 :
                permKey.push_back(k[3]);
                break;
            case 6 :
                permKey.push_back(k[9]);
                break;
            case 7 :
                permKey.push_back(k[0]);
                break;
            case 8 :
                permKey.push_back(k[8]);
                break;
            case 9 :
                permKey.push_back(k[7]);
                break;
            default:
                permKey.push_back(k[5]);
                break;
        }
    }
    shiftKey(permKey);
    roundsKeys[0] = permutRoundKey(permKey);
    shiftKey(permKey);
    shiftKey(permKey);
    roundsKeys[1] = permutRoundKey(permKey);
        if(decrypt){
        reverse(roundsKeys.begin(), roundsKeys.end());
    }
    return roundsKeys;
}
```

Code (3): Implementação da geração de subchaves

```
--------------
Gerando chave ->
Chave: 1010000010
Chave permutada: 1000001100
Chave pós shift: 0000111000
Subchave 1: 10100100
Chave pós 2 shift's: 0010000011
Subchave 2: 01000011
```

Fig. (3): Resultado da geração de subchaves para encriptação

A manipulação dos bits no shift e a seleção dos bits para compor as subchaves são realizadas por funções auxiliares que podem ser visualizadas com mais detalhes em (2).

. Permutação Inicial e final:

A função de permutação inicial deve ser realizada no início do algoritmo usando o texto em claro e deve seguir o seguinte padrão estabelecido em (1).

IP							
2	6	3	1	4	8	5	7

Fig. (3): Função de permutação inicial

A função de permutação final é a função inversa da permutação inicial e deve ser realizada após as duas rodadas de Feistel, também tem sua tabela especificada em (1):

IP^{-1}							
4	1	3	5	7	2	8	6

Fig. (4): Função de permutação final

A implementação dessas permutações segue o mesmo princípio da implementação da permutação ao gerar as subchaves, portanto para mais detalhes consultar o código presente em (2). Para o padrão determinado no projeto o resultado seria:

```
Permutando o texto com IP ->
Texto: 11010111
Texto Permutado: 11011101
```

Fig. (5): Resultado da Permutação Inicial

```
Permutando o texto com IP(inv) ->
Texto: 00110010
Texto Permutado: 10101000
```

Fig. (6): Resultado da Permutação Final

. Rounds de Feistel:

O algoritmo S-DES conta também com dois rounds da cifra de bloco de Feistel usando o resultado da permutação inicial. Essa cifra consiste na divisão em duas partes do texto a ser cifrado, inicialmente os bits mais significativos são atribuídos ao lado “l” e os bits menos significativos ao lado “r”. Após a divisão, em cada round uma string binaria “X” é gerada a partir da função da rodada cujos parâmetros são o lado “r” e a subchave gerada para aquele round, essa string binaria é utilizada para realizar uma operação de XOR bit a bit com o lado “l” gerando assim um novo valor para o lado

“l”, depois dessa operação os lados “l” e “r” são trocados para o novo round, no último round os valores não são trocados.

É possível visualizar o funcionamento de uma rodada da cifra a partir da Fig. (7):

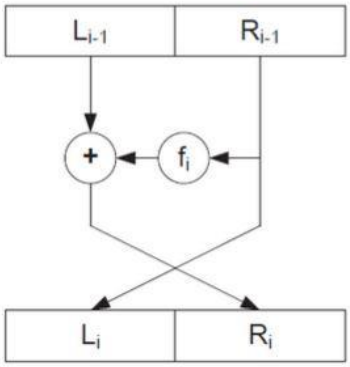

Fig. (7): Diagrama de uma rodada da cifra de Feistel

. Função da rodada:

Uma vez que as subchaves geradas têm tamanho de 8 bits e a parte “r” tem apenas 4 bits, é necessário realizar a expansão desses 4 bits para oito bits. Para isso as especificações do S-DES (1), fornecem a seguinte tabela:

E/P							
4	1	2	3	2	3	4	1

Fig. (8): Função de expansão e permutação

Após essa operação de expansão, um XOR bit a bit é realizado entre o “r” expandido e a subchave desse round, dispondo os bits do resultado por linha em uma matriz 2x4, da seguinte forma Fig. (9), Fig. (10):

$$\left[\begin{array}{c|c|c|c} n_4 \oplus k_{11} & n_1 \oplus k_{12} & n_2 \oplus k_{13} & n_3 \oplus k_{14} \\ n_2 \oplus k_{15} & n_3 \oplus k_{16} & n_4 \oplus k_{17} & n_1 \oplus k_{18} \end{array}\right]$$

$$\left[\begin{array}{c|c|c|c} p_{0,0} & p_{0,1} & p_{0,2} & p_{0,3} \\ p_{1,0} & p_{1,1} & p_{1,2} & p_{1,3} \end{array}\right]$$

Fig. (9) e Fig. (10): Representação matricial dos bits em uma matriz P

Essa matriz é usada para a seleção dos valores em duas S-Boxes “So” e “S1”, dispostas da seguinte forma Fig. (11):

$$S0 = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 1 & 0 & 3 & 2 \\ 3 & 2 & 1 & 0 \\ 0 & 2 & 1 & 3 \\ 3 & 1 & 3 & 2 \end{bmatrix} \end{array} \qquad S1 = \begin{array}{c} \\ 0 \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 0 & 1 & 2 & 3 \end{array} \\ \begin{bmatrix} 0 & 1 & 2 & 3 \\ 2 & 0 & 1 & 3 \\ 3 & 0 & 1 & 0 \\ 2 & 1 & 0 & 3 \end{bmatrix} \end{array}$$

Fig. (11): S-Boxes S0 e S

O valor selecionado de “So” irá compor, inicialmente, os bits mais significativos enquanto o valor de “S1” compõe, inicialmente, os bits menos significativos. A seleção ocorre por meio da concatenação de bits em uma mesma linha, para o valor em “So” a linha o é utilizada, sendo a concatenação entre P[0][0] e P[0][3] o valor que determina a linha de “So” e a concatenação entre P[0][1] e P[0][2] o valor que determina a coluna de “So”. Para “S1” a linha 1 é utilizada, linha e coluna são representadas, respectivamente, pelas concatenações: P[1][0] P[1][3] e P[1][1] P[1][2]. Em posse desse resultado ele é, finalmente, permutado, gerando a saída final, a permutação segue o seguinte padrão especificado em (1):

P4			
2	4	3	1

Fig. (12): Função de permutação para o resultado da função da rodada

A implementação alcançada pode ser visualizada em Code (4), para mais detalhes de funções auxiliares utilizadas consulte (2):

```
int roundFunc(string roundKey, string r) {
    r = expand(r);
    vector<vector<int>> s0 = {
        {1, 0, 3, 2},
        {3, 2, 1, 0},
        {0, 2, 1, 3},
        {3, 1, 3, 2}
    };
    vector<vector<int>> s1 = {
        {0, 1, 2, 3},
        {2, 0, 1, 3},
        {3, 0, 1, 0},
        {2, 1, 0, 3},
     };
    vector<vector<int>> idxs(2, vector<int>(4));
    for(int i = 0; i < 8; i++) {
        int aux = (r[i] - '0') ^ (roundKey[i] -
'0');
        idxs[i / 4][i % 4] = aux;
    }
    string resultPart1 = "";
    resultPart1 += intToBinStr2(s0[idxs[0][0]*2 +
idxs[0][3]][idxs[0][1]*2 + idxs[0][2]]);
    resultPart1 += intToBinStr2(s1[idxs[1][0]*2 +
idxs[1][3]][idxs[1][1]*2 + idxs[1][2]]);
    string result = "";
```

```
    result.push_back(resultPart1[1]);
    result.push_back(resultPart1[3]);
    result.push_back(resultPart1[2]);
    result.push_back(resultPart1[0]);
    return binStrToInt(result);
}
```

Code (4): Implementação da função da rodada

Com essas definições estabelecidas, é possível visualizar os resultados dos rounds da cifra de Feistel para os valores padrões estabelecidos Fig. (12):

```
Round 1 ->
l: 1101 r: 1101
--------------
Funcao F ->
r expandido: 11101011
Matrix XOR subchave e r expandido:
0 1 0 0
1 1 1 1
S0: 11
S1: 11
Resultado funcao F: 1111
--------------
Novo l: 0010
proximo l: 1101 proximo r: 0010
--------------
Round 2 ->
l: 1101 r: 0010
--------------
Funcao F ->
r expandido: 00010100
Matrix XOR subchave e r expandido:
0 1 0 1
0 1 1 1
S0: 01
S1: 11
Resultado funcao F: 1110
--------------
Novo l: 0011
proximo l: 0011 proximo r: 0010
```

Fig. (12): Resultados das rodadas da cifra de Feistel realizadas no algoritmo S-DES

. Decriptação:

Segundo as propriedades do S-DES é possível gerar todas as possibilidades de chaves uma vez que a geração de todas as possibilidades de chave apresenta uma complexidade próxima à $O(2*8*2^{10})$. É possível visualizar a seguir Code (5) a implementação de uma função que gera e decripta o texto cifrado com essas possíveis chaves:

```
void genAllKeys() {
    for(int i = 0; i < (1 << 10); i++){
        for(int j = 0; j < 10; j++){

            key[j] = ((i & (1 << j)) ? 1 : 0) +
'0';
        }
        roundKeys = genRoundKey(true);
        cout << "Chave: " << key << " " << "Texto
em             Claro:              "             <<
finalPerm(desAlgo(iniPerm(cypherTxt))) << '\n';
    }
}
```

Code (5): Implementação de um gerador para todas as chaves

As operações usadas no S-DES apresentam uma propriedade conveniente para decriptar um texto cifrado, todas elas possuem inverso, portanto o algoritmo para decriptar consiste na permutação inicial, o inverso da permutação final, fazer as rodadas de Feistel com as subchaves invertidas e a permutação final, o inverso da permutação inicial. Para a implementação basta um booleano como parâmetro da função de geração de chave como é explicito em Code (3). O resultado da decriptação com o padrão definido pelo projeto é visível a seguir:

```
Decriptando ->
 Permutando o texto com IP ->
Texto: 10101000
Texto Permutado: 00110010
--------------
Round 1 ->
l: 0011 r: 0010
--------------
Funcao F ->
r expandido: 00010100
Matrix XOR subchave e r expandido:
0 1 0 1
0 1 1 1
S0: 01
S1: 11
Resultado funcao F: 1110
--------------
Novo l: 1101
proximo l: 0010 proximo r: 1101
--------------
Round 2 ->
l: 0010 r: 1101
--------------
Funcao F ->
r expandido: 11101011
Matrix XOR subchave e r expandido:
0 1 0 0
1 1 1 1
S0: 11
S1: 11
Resultado funcao F: 1111
--------------
Novo l: 1101
proximo l: 1101 proximo r: 1101
--------------
Permutando o texto com IP(inv) ->
Texto: 11011101
Texto Permutado: 11010111
--------------
11010111
```

Fig. (13): Resultas das rodadas da cifra de Feistel para a decriptação

. Bibliografia:

(1) -> Appendix G Simplified DES, William Stallings, 2010.

(2) -> "https://github.com/EmersonJr/Exercicios-de-Seguranca-Computacional/blob/main/trabalho1/gsdesEncDec.cpp" Repositório no GitHub com a implementação, Emerson Junior.