Міністерство освіти і науки України
НТУУ«Київський політехнічний інститут»
Фізико-технічний інститут

СИМЕТРИЧНА КРИПТОГРАФІЯ

КОМП'ЮТЕРНИЙ ПРАКТИКУМ №4

Побудова генератора псевдовипадкових послідовностей на лінійних регістрах зсуву (генератора Джиффі) та його кореляційний криптоаналіз

Підготували:
студенти 3 курсу
групи ФІ-84
Ковальчук Ольга Миронівна
Коломієць Андрій Юрійович

Перевірив:
Деркач О. Г.

Київ 2021

Побудова генератора псевдовипадкових послідовностей на лінійних регістрах зсуву (генератора Джиффі) та його кореляційний криптоаналіз

Мета роботи:

Ознайомлення з деякими принципами побудови криптосистем на лінійних регістрах зсуву; практичне освоєння програмної реалізації лінійних регістрів зсуву (ЛРЗ); ознайомлення з методом кореляційного аналізу криптосистем на прикладі генератора Джиффі.

Порядок виконання роботи:

1. За даними характеристичними многочленами написати програму роботи ЛРЗ L1 , L2 , L3 і побудованого на них генератора Джиффі.
2. За допомогою формул (4) – (6) при заданому α визначити кількість знаків вихідної послідовності N^*, необхідну для знаходження вірного початкового заповнення, а також поріг C для регістрів L1 та L2
3. Організувати перебір всіх можливих початкових заповнень L1 і обчислення відповідних статистик R з використанням заданої послідовності $(z)_i$, $i = \overline{0, N^* - 1}$
4. Відбракувати випробувані варіанти за критерієм R > C і знайти всі кандидати на істинне початкове заповнення L1
5. Аналогічним чином знайти кандидатів на початкове заповнення L2.
6. Організувати перебір всіх початкових заповнень L3 та генерацію відповідних послідовностей $(s)_i$.
7. Відбракувати невірні початкові заповнення L3 за тактами, на яких $x_i \neq y_i$, де $(x)_i$, $(y)_i$ – послідовності, що генеруються регістрами L1 та L2 при знайдених початкових заповненнях.
8. Перевірити знайдені початкові заповнення ЛРЗ L1 , L2 , L3 шляхом співставлення згенерованої послідовності $(z)_i$ із заданою при $i = \overline{0, N - 1}$

Обчислення параметрів

Задано формули (4) – (6):

$$C = Np_1 + t_{1-\alpha}\sqrt{Np_1(1-p_1)}$$

$$t_{1-\beta} = \frac{Np_2 - C}{\sqrt{Np_2\,(1-p_2)}}$$

$$\beta M < 1$$

Де $p_1 = \frac{1}{4}$, $p_2 = \frac{1}{2}$, $M_1 = 2^{n_1}$, $M_2 = 2^{n_2}$, $M_3 = 2^{n_3}$

$n_1 = 25$, $n_2 = 26$, $n_3 = 27$

Із цих формул виражаємо шукані константи:

$$\beta = \frac{1}{M}$$

$$N^* = (2t_{1-\beta} + \sqrt{3}t_{1-\alpha})^2$$

$$C = \frac{N^*}{4} + t_{1-\alpha}\frac{\sqrt{3N^*}}{4}$$

Були отримані такі значення β, C та N^* для L1, L2 та L3:
Для L1: $\beta = 2.98023 \cdot 10^{-8}$, $N^* = 222$, $C = 66$

Для L2: $\beta = 1.49012 \cdot 10^{-8}$, $N^* = 229$, $C = 68$

Для L3: $\beta = 7.45058 \cdot 10^{-9}$, $N^* = 236$, $C = 70$

Результати роботи

Початкові заповнення регістрів:

Olga Kovalchuk var №19

L1 register: (110011011010001011101011 1)

L2 register: (100011001001001101110000 111)

L3 register: (110011100111010001110110011)

Час перебору для регістра L1: 32.462 сек

Час перебору для регістра L2: 64.784 сек

Час перебору для регістра L3: 15.698 сек

Час перебору для пошуку початкового заповнення: 2.005 сек

Час роботи програми: 114.949 сек = 1 хв 54.949 сек

Andrew Kolomiets var №4

L1 register: (000110011101111010110000 1)

L2 register: (000000111000100011000011 01)

L3 register: (101100011100011100001111000)

Час перебору для регістра L1: 29.961 сек

Час перебору для регістра L2: 66.334 сек

Час перебору для регістра L3: 296.843 сек

Час перебору для пошуку початкового заповнення: 210.875 сек

Час роботи програми: 604.013 сек = 10 хв 4.013 сек

Висновки

У даному комп'ютерному практикумі розглянуто приклад роботи регістрів зсуву та побудованого на них генератора Джиффі. За заданою вихідною послідовністю (z_i) використовуючи засоби криптоаналізу генератору Джиффі було знайдено початкові заповнення регістрів L1, L2, L3.

Код програми

```
#include <iostream>
#include <fstream>
#include <string>
#include <ctime>
#include <time.h>


using namespace std;


struct register_result
{
	int res[1024];
};

register_result L1(int start_x[], int size);
register_result L2(int start_y[], int size);
register_result L3(int start_s[], int size);
register_result Giffy(register_result X, register_result Y, register_result S);
register_result short_L1(int start_x[], int size);

//For Jiffy generator

register_result L1(int start_x[], int size)
{
	register_result X;
	for (int i = 0; i < 25; i++)
	{
		X.res[i] = start_x[i];
	}
	for (int i = 0; i < size - 25; i++)
	{
		X.res[i+25] = X.res[i] xor X.res[i+3];
	}
	return X;
}

register_result short_L1(int start_x[], int size)
{
	register_result X;
	for (int i = 0; i < 4; i++)
	{
		X.res[i] = start_x[i];
```

```
	}
	for (int i = 0; i < size - 4 ; i++)
	{
		X.res[i + 4] = X.res[i] xor X.res[i + 3];
	}
	return X;
}

register_result L2(int start_y[], int size)
{
	register_result Y;
	for (int i = 0; i < 26; i++)
	{
		Y.res[i] = start_y[i];
	}
	for (int i = 0; i < size - 26; i++)
	{
		Y.res[i + 26] = Y.res[i] xor Y.res[i+1] xor Y.res[i+2] xor Y.res[i+6];
	}
	return Y;
}

register_result L3(int start_s[], int size)
{
	register_result S;
	for (int i = 0; i < 27; i++)
	{
		S.res[i] = start_s[i];
	}
	for (int i = 0; i < size - 27; i++)
	{
		S.res[i + 27] = S.res[i] xor S.res[i+1] xor S.res[i+2] xor S.res[i+5];
	}
	return S;
}

register_result Giffy(register_result X, register_result Y, register_result S)
{
	register_result Gif;
	for (int i = 0; i < 1024; i++)
	{
		Gif.res[i] = (S.res[i] * X.res[i]) xor ((1 xor S.res[i]) * Y.res[i]);
	}
	return Gif;
}
```

```
int statistic_R(register_result x, int z[], int size)
{
	int R = 0;
	for (int i = 0; i < size; i++)
	{
		R = R + (x.res[i] xor z[i]);
	}
	return R;
}

//for sorting out all possible variants

int* sort_out(int arr[], int size_of_vector, int num_in_decimal)
{
	for (int i = size_of_vector-1; num_in_decimal > 0; i--)
	{
		arr[i] = num_in_decimal % 2;
		num_in_decimal = num_in_decimal / 2;
	}
	return arr;
}


int main()
{
	register_result X_for_L1;
	register_result Y_for_L2;
	register_result Z_for_L3;

	clock_t start, finish;
	double t_for_L1;
	double t_for_L2;
	double t_for_L3;
	double t_for_final_fill;
	srand(time(NULL));
	//var 3
	//string sequence_of_z =
"01010100000100011110010000111010011110110010111111010101111100100000110000001110000100101000000001100111000000011001110100110000
10001101110100011101000101001110011000101111110101011100101100110100101100111000101010101010001111110011001001100110000101000011
11101100101010111001101000100000001000101000011110010000000110010000100110100011001010000001010010010110001110111100101011100001
00000001011111011110111000000101001001001111110000010100101110001000111001001001101011111010101100011100000101110100000100000001
00101010000111110100001001111101110101001111110101111110110111011011000010100010000101001000110001000010100101010111010100100101
00111001011001110010001111111000010110111000111111100100001000001000110010111100100110100000010110011010010001101011000000000100
```

```
10110001100001000010010001011111111100101101101001011000101110100000100111101110100111010110001100010110100010000001110010010 1
11001101100110110101110110000000111100101001000100100011111000100000101101011011010101010001100010100011111111000101110001000
10110010111010101010101000001001010111011011001011000010110011100011111101010011010110110011000101001001110110110110111100000
11000011011010111111001111010100110110110000011111010110000011110110100010000100001000100100000111110001111001010111010101010100
01010011010100101011010011111111001010011001001011101110000100000000111000000000101111110011111010000010010011101110000000001010
10101010010000000111111001100000111000010111111010011100111000001001111100100010101011001011101100001011111110111011010110001 11
11100010100001001001111000111001000101101110111001001001101110111100010010011111011111000111001000101010101011000011010001011 1
00011111001001011000110101110101011001001100100111101100100101101000101011100111101000011011111111110100011100100000110011101 01
01111010100010001111101111010110001001111110001111001110111101000111011101010001011101111110101011100011101000101111111001110110
11101010001100111110000011101010000111111101000101011001011110100011011000101111001110011101100001100000000000110110010110010 0
11001011101011010010001110000110 0";
      //var 19
      //string sequence_of_z =
"110011001010001101100011110100000101100101011010111011010000111100010110011001111111110010011100000000110001101011011110011100
11001001001000101001110101010010000111111000110110000000100100011101111000101111101001110000011000101101011010111010101011011110
11101111111010001111011010111100010101000010011011010100011011111101010000011001011001111010100101100110111001101001000010110 1
10100010011011111010011110000000101001010110100000000111011100000011000000100101100101011001001001000110101111010000101000100110 1
01011001100101101000100101111000010110000000111110001010101110111111110001011111110110010011110101111001100010101010110001010111
11110010111111111111000100100010111011101010100000011000111011101101111010110011001111011000010110101110111111101110100010111 11
00010010110101110101101111011100001100011010010110011110100110001110000110110000001101000011101100001111011111110010101001101111
11111101100101111001000000101100100001101101011100110110000111010111001011011111111011010101101010010000000110001011000011101011
00000111011111100101100010010100000010010000111010100000001111001101111110000101110010001010000111110111011101001111111010011010 01
10101110110100010101110000010111100011000011011011001101000110101111101011100110011000000001101100010101110110100110000000100111 1
10101100100001010101100011110100011000111010011100111011000100001100111111111000011101011010011100111011110110001100000001110100
10001010101100010010000000101110011001100000101011100100001011010010101001110010000111110000000100111110001110011110110100101010 0
00111011101101000011000110110001110111011110101100000011111000101000011110010101011000110001101110111111100011011011011010000 0
11111010010111101010001111000010000111000000111011000010110100010011110111101110000000111111010100111110010011000111111111000
00101110110100010001100010110100011011111111010011000110101011110011010101011010111001001001010010000001001001101011010111110 1
01100111010101100111000100100001111010011101111010111111011101111110010100010000000111110110101100011011010001000100101110100 01
11110110100011010010101110101011 0";
      //var 4
      string sequence_of_z =
"0001001111001110110000000111101110010100101111011001101110011001000000101010001110101100100110110101111110001001010000001000 11
01101100010000011000110000100001010010010010011010001010000000000000100101010101001111011011001000001011011000001000101100010 10
11010001000000111100011101100101100110101101101010100010111110010011010111010001010000110111110111111111111001111010010000101 1
10010100100100110111011110001000100000000011101111011001011111101100000100111001011000110100011100110111001111010010101011011 1
00111011001001001010111111000000101001010010111100011010000001000000111110111101001010001011101000000101010111110111111010010111 01
10110011010100000000100011100110111100111011011101101010100000001000000111001111100010000011011011010000101011100000101011100111
11111100010000000100111111101101100110111110100111011011110010111110000111000010010110100110010010111011000110101010101001011 10
10110000000111011110111011101100110010011011001000010000101010001100110001001011011010011110101110001110100110000110001001000 11
10100000001110100110001001001110000101001111111101101111010101101011000001111010000000001010000011011001101010000000101010001101 0
1001010000001111110101100111011110000111111100000001011110111010010010011111100110010010000111011000111000100111010000000101001010
01010101010001010111011010110010001100100001110011111001010100101100110101110100110101111101101100101011001111100001110110111 1
0111001101100010100101110011110000000010011110000011011111100010010000011010111110010001011101001111011111110101011010000110100 1
1100110111100000111011011100001010001111000000001101000110111111000110111111100100101010111000010110011101111100110100000001100
```

```
10111011010110001101111011001010010110110101111000010000011101000110110001110011111111100110001001001110011111010011000011010
10110011101111110110111010100010110000100000001011000011011010110111001101001000110000011110110110100111111001110101110001111 0
00111011101100111100101000101010011001100000101000100011011111100010001011000100000110110101101100010001111101111000010111011 1
00101110001101000100101010110011";
	int z[2048];
	for (int i = 0; i < 2048; i++)
	{
		z[i] = int(sequence_of_z.at(i) - '0');
	}
	int start_x[25];
	int start_y[26];
	int start_s[27];
	for (int i = 0; i < 25; i++)
	{
		start_x[i] = 0;
		start_y[i] = 0;
		start_s[i] = 0;
	}
	start_s[25] = start_s[26] = 0;
	start_y[25] = 0;
	double Betta1;
	int M1 = pow(2, 25);
	Betta1 = (double)1 / M1;
	cout << "Betta1: " << Betta1 << endl;
	cout << "1-Betta1: " << (double)1 - Betta1 << endl;
	double Kvantil_1_betta1 = 5.42;
	double Kvantil_1_alfa = 2.33;
	int N1_ = 222;
	int C1 = 66;
	cout << "Number N1*:  " << N1_ << endl;
	cout << "Constant C1:  " << C1 << endl << endl;

	double Betta2;
	int M2 = pow(2, 26);
	Betta2 = (double)1 / M2;
	cout << "Betta2: " << Betta2 << endl;
	cout << "1-Betta2: " << (double)1 - Betta2 << endl;
	double Kvantil_1_betta2 = 5.54;
	int N2_ = 229;
	int C2 = 68;
	cout << "Number N2*:  " << N2_ << endl;
	cout << "Constant C2:  " << C2 << endl << endl;

	cout << "////////////////////////////////////////////////////////////////" << endl;
```

```
ofstream L1_reg;
L1_reg.open("L1 gen.txt");
ofstream L2_reg;
L2_reg.open("L2 gen.txt");

int R = 0;
int* sequence_of_x;
int* sequence_of_y;
int candidates_for_L1 = 0;
int candidates_for_L2 = 0;
int candidates_for_L3 = 0;

//finding true initial candidats for L1 and L2
cout << "Begin finding true initial candidats for L1 and L2" << endl;
start = clock();
for (int i = 1; i < 33554432; i++)
{
      sort_out(start_x, 25, i);
      X_for_L1 = L1(start_x, N1_);
      R = statistic_R(X_for_L1, z, N1_);
      if (R < C1)
      {
            candidates_for_L1++;
            for (int j = 0; j < 25; j++)
            {
                  L1_reg << to_string(start_x[j]);
            }
            L1_reg << endl;
      }

}

finish = clock();
t_for_L1 = (double)(finish - start) / CLOCKS_PER_SEC;
cout << "End of finding true initial candidats for L1" << endl;
cout << "Time for L1: " << t_for_L1 << endl;
cout << "Begin finding true initial candidats for L2" << endl;
start = clock();
for (int i = 0; i < 25; i++)
{
      start_x[i] = 0;
}
//finding true initial filling for L2
for (int i = 1; i < 67108864; i++)
{
```

```
      sort_out(start_y, 26, i);
      Y_for_L2 = L2(start_y, N2_);
      R = statistic_R(Y_for_L2, z, N2_);
      if (R < C2)
      {
            candidates_for_L2++;
            for (int j = 0; j < 26; j++)
            {
                  L2_reg << to_string(start_y[j]);
            }
            L2_reg << endl;
      }

}
cout << "End of finding true initial candidats for L2" << endl;

cout << "Candidats for L1: " << candidates_for_L1 << endl;
cout << "Candidats for L2: " << candidates_for_L2 << endl;
finish = clock();
t_for_L2 = (double)(finish - start) / CLOCKS_PER_SEC;
cout << "Time for L2: " << t_for_L2 << endl;
L1_reg.close();
L2_reg.close();


//finding true initial candidats for L3
string x = "";
string y = "";
int correct = 0;
int that_var_of_input_is_wrong = 0;
ofstream L3_reg;
L3_reg.open("L3 gen.txt");
ifstream L1_for_check;
L1_for_check.open("L1 gen.txt");
ifstream L2_for_check;
L2_for_check.open("L2 gen.txt");
int **Candidates_x;
Candidates_x = new int* [25];
for (int i = 0; i < 25; i++)
{
      Candidates_x[i] = new int[candidates_for_L1];
}
int** Candidates_y;
Candidates_y = new int* [26];
for (int i = 0; i < 26; i++)
```

```
{
	Candidates_y[i] = new int[candidates_for_L2];
}
int j = 0;
if (L1_for_check.is_open())
{
	while (getline(L1_for_check, x))
	{
		for (int k = 0; k < 25; k++)
		{
			Candidates_x[k][j] = int(x[k] - '0');
		}
		j = j + 1;
	}
}
j = 0;
if (L2_for_check.is_open())
{
	while (getline(L2_for_check, y))
	{
		for (int k = 0; k < 26; k++)
		{
			Candidates_y[k][j] = int(y[k] - '0');
		}
		j = j + 1;
	}
}

cout << "Right candidates for L1:" << endl;
for (int i = 0; i < 25; i++)
{
	for (int j = 0; j < candidates_for_L1; j++)
	{
		cout << Candidates_x[i][j] << " ";
	}
	cout << endl;
}
cout << endl;
cout << "Right candidates for L2:" << endl;
for (int i = 0; i < 26; i++)
{
	for (int j = 0; j < candidates_for_L2; j++)
	{
		cout << Candidates_y[i][j] << " ";
	}
```

```
      cout << endl;
}
cout << endl;

register_result* x_seq = new register_result[candidates_for_L1];
register_result* y_seq = new register_result[candidates_for_L2];
for (int i = 0; i < candidates_for_L1; i++)
{
      for (int j = 0; j < 25; j++)
      {
            start_x[j] = Candidates_x[j][i];
      }
      x_seq[i] = L1(start_x, 1024);
}
for (int i = 0; i < candidates_for_L2; i++)
{
      for (int j = 0; j < 26; j++)
      {
            start_y[j] = Candidates_y[j][i];
      }
      y_seq[i] = L2(start_y, 1024);
}

int x_check_num = 0;
int y_check_num = 0;
int for_no_writing_same = 0;
cout << "Begin finding true initial candidats for L3" << endl;
start = clock();
for (int i = 1; i < 134217728; i++)
{
      for_no_writing_same = 1;
      sort_out(start_s, 27, i);
      x_check_num = 0;
      while (x_check_num < candidates_for_L1)
      {
            y_check_num = 0;
            while (y_check_num < candidates_for_L2)
            {
                  that_var_of_input_is_wrong = 0;
                  correct = 0;
                  for (int p = 0; p < 27; p++)
                  {
                        if (that_var_of_input_is_wrong > 0)
                        {
                              break;
```

```
                }
                if (x_seq[x_check_num].res[p] != y_seq[y_check_num].res[p])
                {
                      if (z[p] == x_seq[x_check_num].res[p])
                      {
                            if (start_s[p] == 1)
                            {
                                  correct = correct + 1;
                            }
                            else {
                                  correct = 0;
                                  that_var_of_input_is_wrong++;
                                  break;
                            }
                      }
                      else {
                            if (z[p] == y_seq[y_check_num].res[p]) {
                                  if (start_s[p] == 0)
                                  {
                                        correct = correct + 1;
                                  }
                                  else {
                                        correct = 0;
                                        that_var_of_input_is_wrong++;
                                        break;
                                  }
                            }
                      }
                }
          }
          if (correct > 0 && for_no_writing_same ==1)
          {
                candidates_for_L3++;
                for (int t = 0; t < 27; t++)
                {
                      L3_reg << to_string(start_s[t]);
                }
                L3_reg << endl;
                for_no_writing_same++;
          }
          correct = 0;
          y_check_num++;
    }
    x_check_num++;
}
```

```
}

L1_for_check.close();
L2_for_check.close();
L3_reg.close();
finish = clock();
t_for_L3 = (double)(finish - start) / CLOCKS_PER_SEC;
cout << "Time for L3: " << t_for_L3 << endl;
cout << "End of finding true initial filling for L3" << endl;

correct = 0;
string s;
string text;
register_result* s_seq = new register_result[candidates_for_L3];
register_result L1_seq;
register_result L2_seq;
register_result L3_seq;
ifstream L3_reg_read;
cout << "Begin finding initial fillings for L1, L2, L3 so that we will get Z, that is io our variant" << endl;
start = clock();
L3_reg_read.open("L3 gen.txt");
ofstream Fin;
Fin.open("Final result.txt");
int l = 0;
while (getline(L3_reg_read, s))
{
      for (int i = 0; i < 27; i++)
      {
            start_s[i] = int(s[i] - '0');
      }
      s_seq[l] = L3(start_s, 1024);
      l = l + 1;
}
for (int i = 0; i < candidates_for_L3; i++)
{
      for (int j = 0; j < candidates_for_L2; j++)
      {
            for (int m = 0; m < candidates_for_L1; m++)
            {
                  correct = 0;
                  Z_for_L3 = Giffy(x_seq[m], y_seq[j], s_seq[i]);
                  for (int p = 0; p < 1024; p++)
                  {
                        if (z[p] != Z_for_L3.res[p])
                        {
```

```
                        correct = 0;
                        break;
                  }
                  else {
                        correct += 1;
                  }
            }
            if (correct == 1024)
            {
                  text = "L1 register:";
                  Fin << text;
                  Fin << endl;
                  for (int q = 0; q < 25; q++)
                  {
                        Fin << to_string(x_seq[m].res[q]);
                  }
                  Fin << endl;

                  L1_seq = x_seq[m];
                  L2_seq = y_seq[j];
                  L3_seq = s_seq[i];

                  text = "L2 register:";
                  Fin << text;
                  Fin << endl;
                  for (int q = 0; q < 26; q++)
                  {
                        Fin << to_string(y_seq[j].res[q]);
                  }
                  Fin << endl;

                  text = "L3 register:";
                  Fin << text;
                  Fin << endl;
                  for (int q = 0; q < 27; q++)
                  {
                        Fin << to_string(s_seq[i].res[q]);
                  }
                  Fin << endl;
                  i = candidates_for_L3;
                  j = candidates_for_L2;
                  break;
            }
      }
}
```

```
}

L3_reg_read.close();
Fin.close();
cout << "End of finding initial fillings for L1, L2, L3 so that we will get Z, that is io our variant" << endl;
finish = clock();
t_for_final_fill = (double)(finish - start) / CLOCKS_PER_SEC;
cout << "Time for finding final correct fill: " << t_for_final_fill << endl;

Z_for_L3 = Giffy(L1_seq, L2_seq, L3_seq);
cout << "Z in variant:" << endl;
for (int i = 0; i < 1024; i++)
{
      cout << z[i] <<" ";
}
cout << endl;
cout << "Z we got:" << endl;
for (int i = 0; i < 1024; i++)
{
      cout << Z_for_L3.res[i] << " ";
}
cout << endl;

cout << "L1:" << endl;
for (int i = 0; i < 25; i++)
{
      cout << L1_seq.res[i] << " ";
}
cout << endl;
cout << "L2:" << endl;
for (int i = 0; i < 26; i++)
{
      cout << L2_seq.res[i] << " ";
}
cout << endl;
cout << "L3:" << endl;
for (int i = 0; i < 27; i++)
{
      cout << L3_seq.res[i] << " ";
}
cout << endl;


for (int i = 0; i < candidates_for_L1; i++)
{
```

```
            delete Candidates_x[i];
      }
      delete [] Candidates_x;
      for (int i = 0; i < candidates_for_L2; i++)
      {
            delete Candidates_y[i];
      }
      delete[] Candidates_y;

      return 0;
}
```