Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Фізико-технічний інститут

ЛАБОРАТОРНА РОБОТА №4

з дисципліни

«Криптографія»

на тему:

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

Виконали:
студенти 3 курсу ФТІ
групи ФБ-71
Рейценштейн Кирило і Таран Вікторія
Перевірили:
Чорний О.
Савчук М. М.
Завадська Л. О.

Київ 2019

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:

0. Уважно прочитати методичні вказівки до виконання комп'ютерного практикуму.
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=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 $x_i' = y_i$, 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 $i=0,N-1$.

Вихідні дані:

Характеристичні многочлени:
– для *L1* : $p(x) = x^{30} \oplus x^6 \oplus x^4 \oplus x \oplus 1$, що відповідає співвідношенню між членами послідовності $x_{i+30} = x_{i+6} \oplus x_{i+4} \oplus x_{i+1} \oplus x_i$;
– для *L2* : $p(x) = x^{31} \oplus x^3 \oplus 1$, відповідна рекурента: $y_{i+31} = y_i \oplus y_{i+3}$;
– для *L3* : $p(x) = x^{32} \oplus x^7 \oplus x^5 \oplus x^3 \oplus x^2 \oplus x \oplus 1$, відповідна рекурента: $s_{i+32} = s_i \oplus s_{i+1} \oplus s_{i+2} \oplus s_{i+3} \oplus s_{i+5} \oplus s_{i+7}$;
Імовірність помилки першого роду $\alpha = 0{,}01$.

Наш підхід до вирішення:

Після одержання послідовностей для L1 та L2, ми відібрали невірні послідовності L2 за критерієм (x=y)!=z. Тобто якщо в L1 та L2 біти були однакові, а в послідовності z інший біт, то послідовність L2 відкидувалась. Далі порівнюючи L1, L2 та послідовність z, ми відновили 14 початкових бітів L3. Інші біти були згенеровані випадковим чином та підставлені до послідовності L3. Якщо послідовність підходила за рівнянням генератора Джиффі, то послідовність L3 знайдена. Тобто всі три послідовності відновлені.

Результати:

Знайдені критерії та довжини для регістрів:
Для L1 N* = 258 C = 81
Для L2 N* = 265 C = 83
Для L3 N* = 273 C = 85

Варіант 13

```
11101001000011101011001110101111000011010101101000111001101110110110001101000110010110011011000001000110
01100110110000001101010101001110101100001010011100100101010100110100100101111110010111111111000000011111
11000100011001100011110100000010111001010100001010000000000011101011111010111010111001101101110100111111
01000001001010100000010001000110010010000001000011101101111000010110001100110111011110000010100000100000110
00000000000000011100101101010110010001000101100001110000101011001011000011010110000010010001011010110001111
11100110110101010100111011110111100001100110011000101000110010000010111000011010101010000011000111110111
00100100111001111101011111110111001001100101101011100000111010100111001001001000010001000001011100101000 1
11000100000001101010101010101000111100010100111111000000100111111111110001111001010101000101000010100001100
01110111000110100101001001111001100010010100111010100001001010011000011111111110000110001011100111101010 1
00111111011000101110011011001110111000101111110100110101101110010011111000110011011101001010100001010111 0
10101011100010101011100011111001010101111110001110011011011110100110000010010000000011011111010011011000 0
01011010111101101110010101000000101001110111111110110011001000001010111011100001100001001100110011011011 01
11011110001110011111001010010100010100111000111001101100010000001010100001010111010100101111000010110000
11110001000100110111010000111010100111010010011111011111001001011001101100101000010010001101011010111 0
00101100110110110011100001101000001011100100110011010110000010110011101000111011001001111011111111011101110
11001011101000000000101101101001000111111001010100010010000010111101010110110001001100011000111011010111 00
00100000101111001101100101010111111001011011000011110101000100010000100101011110100100100010011100000010 0
01111011100110100100100011011100011110000010010010001110010001110111101011100100100110000000011100100100 1
00110000110011101011111100010100100100011101000010000001000011101011000001111001011011001100010110010101 1
01010000011001001010000000010011101101001000011010111
```

Початкові значення послідовностей:
L1: 10101111000011101011101000001 0
L2: 11101001111011001011000110111 11
L3: 00000000111001110000001000010011

Повні послідовності:
L1:
10101111000011101011101000001 0110000100101011011101100011011111110110001011101110100110111011110
01100100111110011111000001110111011100111010010000101000110010010001111010010011000111100011010
11011100100001101001100011010110001100010001000001000110010101000001
L2:
11101001111011001011000110111111101001101000100100111100010001010010010110000001101111001101100 0
00010011000110001011010000110001000101111011101000101011011100110101010011010110111000011101001
11110010011000011110111101001100110000101101110100101010010101010110101000011
L3:
00000000111001110000001000010011011010101110110001010101001110110110100001000111001110001010001
00111000010010111001010110000101010001110001011000110110101110111000010000001110101111100100110
00011101011010100001001100001111101000010000110100000111011101111100100001101000 1

Код:

```
#include <Windows.h>
#include <iostream>
#include <fstream>
#include <bitset>
#include <vector>

// Calculated constants
const int l1_size = 258;
const int l1_criteria = 81;
const int l2_size = 265;
const int l2_criteria = 83;
const int l3_size = 272;
//

struct ThreadData
{
	unsigned long long start;
	unsigned long long end;
	FILE* file_ptr;
};

CRITICAL_SECTION thread_cs;
	l_data->start = l_per_thread * i;
	l_data->end = l_per_thread * (i + 1);
	l_data->file_ptr = file_ptr;
	hThreads[i] = CreateThread(NULL, NULL,
(LPTHREAD_START_ROUTINE)func_addr, l_data, NULL, &ThreadID);
	}

	WaitForMultipleObjects(threads_count, hThreads, TRUE,
INFINITE);

	fclose(file_ptr);
}

void l1_calc(ThreadData* struct_ptr)
{
	for (unsigned long long i = struct_ptr->start; i < struct_ptr->end;
i++)
	{
		std::string bitset_string =
std::bitset<30>(i).to_string();

		for (int j = 0; j < l1_size - 30; j++)
```

```
std::string input_data = "";
std::string final_l1 = "", final_l2 = "", final_l3 = "";

std::vector<std::string> final_data_l1;
std::vector<std::string> final_data_l2;

void ThreadCreator(void* func_addr, int pow_num, int threads_count, const
char* file_name);
void l1_calc(ThreadData* struct_ptr);
void l2_calc(ThreadData* struct_ptr);
void l3_calc();

int main()
{
	SetConsoleCP(1251);
	SetConsoleOutputCP(1251);

	InitializeCriticalSection(&thread_cs);

	std::string input_filename = "";
	std::cout << "Введите название файла с входящей
последовательностью: ";
	std::cin >> input_filename;

	std::string num_of_threads = "";
	std::cout << "Введите количество потоков (будьте осторожны,
слишком большое число может сильно нагрузить ваш процессор): ";
	std::cin >> num_of_threads;

	int threads_count = atoi(num_of_threads.c_str());

	std::ifstream input_sequence(input_filename);
	input_sequence >> input_data;
	input_sequence.close();

	std::cout << "1. Посчитать L1\n2. Посчитать L2\n3. Посчитать
L3\n4. Посчитать всё\nЧто вы хотите сделать: ";
	std::string action = "";
	std::cin >> action;

	if (action == "1" || action == "4")
	{
		std::cout << "Поиск подходящих L1..." << std::endl;

		ThreadCreator(l1_calc, 30, threads_count, "l1.txt");

		std::cout << "Последовательности L1:" << std::endl;

		for (int i = 0; i < final_data_l1.size(); i++)
		{
			std::cout << final_data_l1[i] << std::endl;
		}
	}

	if (action == "2" || action == "4")
	{
		std::cout << "Поиск подходящих L2..." << std::endl;

		ThreadCreator(l2_calc, 31, threads_count, "l2.txt");

		std::cout << "Последовательности L2:" << std::endl;

		for (int i = 0; i < final_data_l2.size(); i++)
		{
			std::cout << final_data_l2[i] << std::endl;
		}
	}

	if (action == "3" || action == "4")
	{
		if (action != "4")
		{
			std::string file_name_l1 = "",
file_name_l2 = "";
			std::cout << "Введите название файла с
последовательностями L1: ";
			std::cin >> file_name_l1;
			std::cout << "Введите название файла с
последовательностями L2: ";
			std::cin >> file_name_l2;

			std::ifstream l1_data(file_name_l1);
			std::ifstream l2_data(file_name_l2);

			while (!l1_data.eof())
			{
				std::string temp = "";
				l1_data >> temp;

				if (temp[0] != '\n')
				{
					bitset_string += ((bitset_string[j] ^
bitset_string[j + 1] ^ bitset_string[j + 4] ^ bitset_string[j + 6]) + 0x30);
				}

				int R = 0;

				for (int j = 0; j < l1_size; j++)
				{
					R += (bitset_string[j] ^ input_data[j]);
				}

				if (R < l1_criteria)
				{
					EnterCriticalSection(&thread_cs);
					final_data_l1.push_back(bitset_string);
					fputs(bitset_string.c_str(), struct_ptr-
>file_ptr);
					fputs("\n", struct_ptr->file_ptr);
					LeaveCriticalSection(&thread_cs);
				}
			}

			ExitThread(NULL);
}

void l2_calc(ThreadData* struct_ptr)
{
	for (unsigned long long i = struct_ptr->start; i < struct_ptr->end;
i++)
	{
		std::string bitset_string =
std::bitset<31>(i).to_string();

		for (int j = 0; j < l2_size - 31; j++)
		{
			bitset_string += ((bitset_string[j] ^
bitset_string[j + 3]) + 0x30);
		}

		int R = 0;

		for (int j = 0; j < l2_size; j++)
		{
			R += (bitset_string[j] ^ input_data[j]);
		}

		if (R < l2_criteria)
		{
			EnterCriticalSection(&thread_cs);
			final_data_l2.push_back(bitset_string);
			fputs(bitset_string.c_str(), struct_ptr-
>file_ptr);
			fputs("\n", struct_ptr->file_ptr);
			LeaveCriticalSection(&thread_cs);
		}
	}

	ExitThread(NULL);
}

void l3_calc()
{
	final_l1 = final_data_l1[0];

	// finding l2
	for (int j = 0; j < final_data_l2.size(); j++)
	{
		for (int k = 0; k < l1_size; k++)
		{
			if (final_l1[k] != input_data[k] &&
final_data_l2[j][k] != input_data[k])
			{
				break;
			}

			if (k == l1_size - 1)
			{
				final_l2 = final_data_l2[j];
			}
		}
	}
	//

	// calculating l3
	std::string l3_generated = "";
	for (int i = 0; i < 32; i++)
	{
		if (final_l1[i] == input_data[i] && final_l2[i] ==
input_data[i])
```

```
                                        {

                final_data_l1.push_back(temp);
                                        }
                                }
                                l1_data.close();

                                while (!l2_data.eof())
                                {
                                        std::string temp = "";
                                        l2_data >> temp;

                                        if (temp[0] != '\n')
                                        {

                final_data_l2.push_back(temp);
                                        }
                                }
                                l2_data.close();
                        }

                        std::cout << "Генерация L3 относительно L1 и
L2..." << std::endl;

                        l3_calc();
                }

                std::cout << "Финальные последовательности L1,L2,L3:" <<
std::endl;
                std::cout << "L1: " << final_l1 << std::endl;
                std::cout << "L2: " << final_l2 << std::endl;
                std::cout << "L3: " << final_l3 << std::endl;

                system("pause");

                return true;
}

void ThreadCreator(void* func_addr, int pow_num, int threads_count, const
char* file_name)
{
                unsigned long long l_start = pow(2, pow_num);
                unsigned long long l_per_thread = l_start / threads_count;

                DWORD ThreadID = NULL;
                HANDLE* hThreads = new HANDLE[threads_count];

                FILE* file_ptr = NULL;
                fopen_s(&file_ptr, file_name, "wb");

                for (int i = 0; i < threads_count; i++)
                {
                        ThreadData* l_data = new ThreadData();
```

```
                        {
                                l3_generated += "?";
                        }
                        else if (final_l1[i] == input_data[i])
                        {
                                l3_generated += "1";
                        }
                        else if (final_l2[i] == input_data[i])
                        {
                                l3_generated += "0";
                        }
                }
                //

                // l3 brute-force
                // 18 unknown bits found during testing our sequence ?
                unsigned long long l3_start = pow(2, 18);
                //

                for (unsigned long long i = 0; i < l3_start; i++)
                {
                        std::string bitset_string =
std::bitset<18>(i).to_string();

                        std::string l3_not_full = l3_generated;

                        for (int j = 0, k = 0; j < 32; j++)
                        {
                                if (l3_not_full[j] == '?')
                                {
                                        l3_not_full[j] =
bitset_string[k++];
                                }
                        }

                        for (int l = 0; l < l3_size - 32; l++)
                        {
                                l3_not_full += ((l3_not_full[l] ^
l3_not_full[l + 1] ^ l3_not_full[l + 2] ^ l3_not_full[l + 3] ^ l3_not_full[l + 5] ^
l3_not_full[l + 7]) + 0x30);
                        }

                        for (int m = 0; m < l1_size; m++)
                        {
                                if (l3_not_full[m] == '1' && final_l1[m]
== input_data[m] || l3_not_full[m] == '0' && final_l2[m] == input_data[m] ||
final_l1[m] == final_l2[m])
                                {
                                        if (m == l1_size - 1)
                                        {
                                                final_l3 =
l3_not_full;
                                        }
                                        continue;
                                }
                                break;
                        }
                }
                //
}
```

Висновок:

Під час данного комп’ютерного практикуму, ми ознайомились з деякими принципами побудови криптосистем на лінійних регістрах зсуву та з методом кореляційного аналізу криптосистем на прикладі генератора Джиффі.