БЛОЧНЫЕ АЛГОРИТМЫ СИММЕТРИЧНОГО ШИФРОВАНИЯ. РЕЖИМЫ РАБОТЫ

Вариант №5

отчет о лабораторной работе №7
по дисциплине
МЕТОДЫ И СРЕДСТВА КРИПТОГРАФИЧЕСКОЙ ЗАЩИТЫ ИНФОРМАЦИИ

Выполнила _____

ст. гр. №230711, Павлова В.С.

Проверила _____

доцент каф. ИБ, Басалова Г.В.

ХОД РАБОТЫ

Задание 1. Реализовать программно генератор псевдослучайных чисел, имеющий в качестве выхода последовательность бит. Согласно заданию варианта, необходимо реализовать генератор «стоп-пошел» (Stop-and-Go) Both-Piper. Разработать программу шифрования произвольных данных, записанных в файле, с помощью генерируемой последовательности бит, используемой в качестве гаммы.

Принципиальная схема данного генератора приведена на рисунке 1.

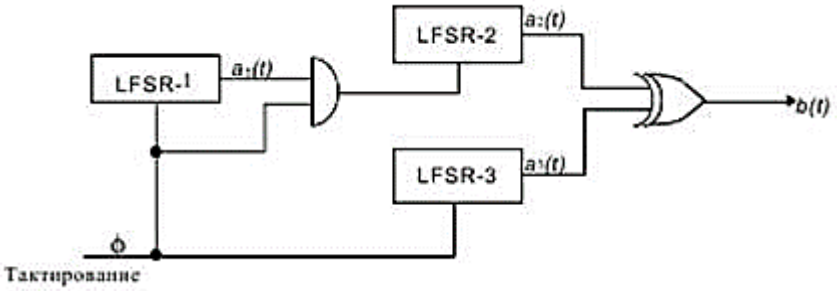

Рисунок 1 – Схема ГПСЧ «Stop-and-Go»

Программная реализация генератора представлена в листинге 1.

Листинг 1 – Код программы шифрования с использованием ГПСЧ

```
#pragma once
#include <iostream>
#include <vector>
#include <bitset>
#include <string>
```

Листинг 1 – Код программы шифрования с использованием ГПСЧ (продолжение)

```
using namespace std;

// Размеры регистров

const int N = 16;
const int M = 16;
const int K = 16;

const int gammaSize = 8;

// Начальные значения регистров

const uint16_t INIT1 = 0xc332;
const uint16_t INIT2 = 0x13e5;
const uint16_t INIT3 = 0x669a;


inline void Print(uint16_t &LFSR1, uint16_t& LFSR2, uint16_t& LFSR3)
{
	cout << "Initial state of registers:\n";

	cout << "LFSR1:\t";
	for (size_t i = 0; i < N; i++)
	{
		cout << ((LFSR1 >> (N - i - 1) & 0x01));
	}
	cout << hex << "\t(" << INIT1 << ")\n";

	cout << "LFSR2:\t";
	for (size_t i = 0; i < M; i++)
	{
		cout << ((LFSR2 >> (M - i - 1) & 0x01));
	}
	cout << hex << "\t(" << INIT2 << ")\n";

	cout << "LFSR3:\t";
	for (size_t i = 0; i < K; i++)
	{
		cout << ((LFSR3 >> (K - i - 1) & 0x01));
	}
	cout << hex << "\t(" << INIT3 << ")\n";
}

inline void Initialization(uint16_t &LFSR1, uint16_t &LFSR2, uint16_t&LFSR3)
{

	LFSR1 = INIT1;
	LFSR2 = INIT2;
	LFSR3 = INIT3;

	Print(LFSR1, LFSR2, LFSR3);
}

inline char MakeGamma(uint16_t& LFSR1, uint16_t& LFSR2, uint16_t& LFSR3)
{
	char gamma;

	uint16_t bit1 = 0x00;
	uint16_t bit2 = 0x00;
	uint16_t bit3 = 0x00;

	uint16_t newbit1 = 0x00;
	uint16_t newbit2 = 0x00;
	uint16_t newbit3 = 0x00;
```

Листинг 1 – Код программы шифрования с использованием ГПСЧ (продолжение)

```
	for (size_t i = 0; i < gammaSize; i++)
	{
		// Получение нового бита для 1-го регистра

		newbit1 =
			((LFSR1 >> 1) & 0x01)		// Бит первого разряда
			^ ((LFSR1 >> 5) & 0x01)		// Бит пятого разряда
			^ ((LFSR1 >> 8) & 0x01)		// Бит восьмого разряда
			^ ((LFSR1 >> 11) & 0x01)	// Бит одиннадцатого разряда
			^ ((LFSR1 >> 14) & 0x01);	// Бит четырнадцатого разряда

		// Получение бита из 1-го регистра после сдвига

		bit1 = LFSR1 & 0x01;

		LFSR1 = (LFSR1 >> 1) | (newbit1 << 15);

		// Изменение состояния 2-го регистра,
		// если с выхода 1-го получена единица

		if (bit1 == 0x01)
		{
			// Получение нового бита для 2-го регистра
			newbit2 =
				((LFSR2 >> 1) & 0x01)		// Бит первого разряда
				^ ((LFSR2 >> 5) & 0x01)		// Бит пятого разряда
				^ ((LFSR2 >> 8) & 0x01)		// Бит восьмого разряда
				^ ((LFSR2 >> 11) & 0x01)	// Бит 11-го разряда
				^ ((LFSR2 >> 14) & 0x01);	// Бит 14-го разряда

			// Получение бита из 2-го регистра после сдвига

			bit2 = LFSR2 & 0x01;

			LFSR2 = (LFSR2 >> 1) | (newbit2 << 15);
		}


		// Получение нового бита для 3-го регистра

		newbit3 =
			((LFSR3 >> 1) & 0x01)		// Бит первого разряда
			^ ((LFSR3 >> 5) & 0x01)		// Бит пятого разряда
			^ ((LFSR3 >> 8) & 0x01)		// Бит восьмого разряда
			^ ((LFSR3 >> 11) & 0x01)	// Бит одиннадцатого разряда
			^ ((LFSR3 >> 14) & 0x01);	// Бит четырнадцатого разряда

		// Получение бита из 3-го регистра после сдвига

		bit3 = LFSR3 & 0x01;

		LFSR3 = (LFSR3 >> 1) | (newbit3 << 15);

		// Наполнение гаммы битовой последовательностью

		gamma = (gamma << 1) | (bit2 ^ bit3);

	}

	return gamma;
}
```

Листинг 1 – Код программы шифрования с использованием ГПСЧ (продолжение)

```
#include <fstream>

// Для расшифрования поменять data на encrypted

const string DATA_PATH = "D:\\WORK\\NIX2TWIX\\CODE\\Crypto_Lab7\\data.txt";
const string OUTPUT_PATH = "D:\\WORK\\NIX2TWIX\\CODE\\Crypto_Lab7\\encrypted.txt";

int main_ENCRYPTOR()
{
	vector<unsigned char> LFSR1;
	vector<unsigned char> LFSR2;
	vector<unsigned char> LFSR3;
	vector<unsigned char> gammaSeed;
	vector<unsigned char> gamma;

	int gammaSize = 64;

	char byte = 0;

	Initialization(LFSR1, LFSR2, LFSR3);

	gammaSeed = MakeGamma(gammaSize, LFSR1, LFSR2, LFSR3);
	cout << "Gamma:\t";
	for (size_t i = 0; i < gammaSize / 8; i++)
	{
		for (size_t j = 0; j < 8; j++)
		{
			byte <<= 1;
			byte += gammaSeed[j * i + j];
		}
		gamma.push_back(byte);
		cout << byte;
	}

	ifstream dataFile(DATA_PATH, ios::binary);
	// Определение размера файла
	dataFile.seekg(0, ios::end);
	const int fileSize = dataFile.tellg();
	dataFile.seekg(0, ios::beg);
	vector<unsigned char> data(fileSize);
	dataFile.read(reinterpret_cast<char*>(data.data()), fileSize);
	dataFile.close();

	// Шифрование с помощью полученной гаммы
	ofstream encryptedFile(OUTPUT_PATH, ios::binary);

	for (int gammaIndex = 0, i = 0; i < data.size(); i++)
	{
		// Гаммирование
		data[i] ^= gamma[gammaIndex];

		// Переход к следующему байту гаммы
		gammaIndex = (gammaIndex + 1) % gamma.size();
	}

	encryptedFile.write(reinterpret_cast<const char*>(data.data()),
data.size());
	encryptedFile.close();

	return EXIT_SUCCESS;

	}
```

Задание 2. Исследовать равномерность датчика ПСЧ (проверить гипотезу о равномерности распределения совокупности чисел, генерируемых датчиком ПСЧ). Определить период датчика ПСЧ для заданных параметров.

Для проверки датчика на равномерность распределения разделим интервал (0;1) на 10 частей. Вычислим, сколько чисел попало в каждый из интервалов и занесём эти данные в таблицу 1.

Таблица 1 – Данные полученного распределения

Интервал	Количество чисел, попавших в интервал
[0; 0.1)	91
[0.1; 0.2)	103
[0.2; 0.3)	86
[0.3; 0.4)	108
[0.4; 0.5)	115
[0.5; 0.6)	98
[0.6; 0.7)	98
[0.7; 0.8)	112
[0.8; 0.9)	92
[0.9; 1)	94

На основе таблицы 1 вычислим значения критерия $\chi^2_{\text{набл}} = \sum_{i=1}^{k} \frac{(n_i - p_i * N)^2}{p_i * N} = \frac{(91-100)^2}{100} + \frac{(103-100)^2}{100} + \frac{(86-100)^2}{100} + \frac{(108-100)^2}{100} + \frac{(115-100)^2}{100} + \frac{(98-100)^2}{100} + \frac{(98-100)^2}{100} + \frac{(112-100)^2}{100} + \frac{(92-100)^2}{100} + \frac{(94-100)^2}{100} = 8.27,$ где число интервалов k = 10, количество чисел N = 1000, n_i – количество случайных чисел, попавших в каждый интервал, а $p_i * N = \frac{1}{k} * N = 100.$

По таблице критерия согласия Пирсона имеем $\chi^2_{\text{кр}}(\alpha, \nu) = \chi^2_{\text{кр}}(0.3, 9) = 10.7$, где α – уровень значимости, а $\nu = k - 1$ – число степеней свободы. Поскольку $\chi^2_{\text{набл}} < \chi^2_{\text{кр}}$, нет оснований отвергать гипотезу о том, что полученное распределение равномерно.

Период полученного распределения превышает 1000.