Міністерство освіти і науки України

Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Лабораторна робота

із Криптографії №4

Побудова регістрів зсуву з лінійним зворотним зв’язком та дослідження їх властивостей

Виконали:

Топчій Микита ФБ - 74

Височанська Вікторія ФБ - 71

Перевірено_____

Київ 2019

КРИПТОГРАФІЯ
КОМП'ЮТЕРНИЙ ПРАКТИКУМ №4
Побудова регістрів зсуву з лінійним зворотним зв'язком та дослідження їх властивостей

Мета роботи
Ознайомлення з принципами побудови регістрів зсуву з лінійним зворотним зв'язком; практичне освоєння їх програмної реалізації; дослідження властивостей лінійних рекурентних послідовностей та їх залежності від властивостей характеристичного полінома регістра.

Порядок виконання роботи
0. Уважно прочитати методичні вказівки до виконання комп'ютерного практикуму.
1. Вибрати свій варіант завдання згідно зі списком. Варіанти завдань містяться у файлі Crypto_CP4 LFSR_Var.
2. За даними характеристичними многочленами $p_1(x)$, $p_2(x)$ скласти лінійні рекурентні співвідношення для ЛРЗ, що задаються цими характеристичними многочленами.
3. Написати програми роботи кожного з ЛРЗ L_1 , L_2 .
4. За допомогою цих програм згенерувати імпульсні функції для кожного з ЛРЗ і підрахувати їх періоди.
5. За отриманими результатами зробити висновки щодо влавстивостей кожного з характеристичних многочленів $p_1(x)$, $p_2(x)$: многочлен примітивний над F_2 ; не примітивний, але може бути незвідним; звідний.
6. Для кожної з двох імпульсних функцій обчислити розподіл k-грам на періоді, $k \le n_i$, де n_i - степінь полінома $f_i(x)$, $i=1,2$ а також значення функції автокореляції $A(d)$ для $0 \le d \le 10$. За результатами зробити висновки.

Варіант 18:

$P_1(X)= X_{23}+X_{20}+X_{19}+X_{18}+X_{17}+X_{13}+X_{12}+X_{11}+X_7+X+1$
$P_2(X)= X_{22}+X_{20}+X_{19}+X_{17}+X_{16}+X_{15}+X_{12}+X_9+X_7+X_5+X_4+X_3+X_2+X+1$

Довжини періодів:
L_1: $2^{23}-1$ = 8 388 607 => $P_1(x)$ – примітивний поліном поля F_2
L_2: 35 805 => $P_2(x)$ – не примітивний та звідний

Розподіл К-грам полінома Р1:

2-грами:		3-грами		4-грами		5-грами	
00	0.250349	000	0.124932	0000	0.0624561	00000	0.0312703
01	0.249847	001	0.125061	0001	0.0626397	00001	0.0312009
10	0.249721	010	0.125264	0010	0.0624937	00010	0.0311286
11	0.250083	011	0.124761	0011	0.0622511	00011	0.0312152
		100	0.124918	0100	0.0623906	00100	0.0314956
		101	0.124745	0101	0.0626427	00101	0.0313074
		110	0.125008	0110	0.0626206	00110	0.0312939
		111	0.125312	0111	0.0623888	00111	0.0315635
				1000	0.0624662	01000	0.0311808
				1001	0.0625831	01001	0.0313575
				1010	0.0624251	01010	0.0312846
				1011	0.0623798	01011	0.0310836
				1100	0.0627202	01100	0.0314455
				1101	0.0624299	01101	0.031095
				1110	0.06244	01110	0.0313775
				1111	0.0626725	01111	0.0311801
						10000	0.0311093
						10001	0.0311601
						10010	0.031183
						10011	0.0309133
						10100	0.0312266
						10101	0.0310721
						10110	0.0313275
						10111	0.0313868
						11000	0.0311844
						11001	0.031311
						11010	0.0313389
						11011	0.0315213
						11100	0.0313103
						11101	0.0311844
						11110	0.0311293
						11111	0.0311615

Розподіл К-грам полінома Р2:

2-грами		3-грами		4-грами		5-грами	
00	0.247926	000	0.128239	0000	0.0601871	00000	0.0318365
01	0.250691	001	0.120755	0001	0.0608854	00001	0.0308311
10	0.247005	010	0.128463	0010	0.0624214	00010	0.0330094
11	0.254378	011	0.123995	0011	0.0668901	00011	0.0306635
		100	0.125335	0100	0.0608854	00100	0.0289879
		101	0.119861	0101	0.0620025	00101	0.0309987
		110	0.121984	0110	0.0638179	00110	0.0306635
		111	0.131367	0111	0.05907	00111	0.0313338
				1000	0.0638179	01000	0.0308311
				1001	0.0624214	01001	0.0299933
				1010	0.0606061	01010	0.0348525
				1011	0.0650747	01011	0.036193
				1100	0.0624214	01100	0.0309987
				1101	0.0635386	01101	0.0309987
				1110	0.0620025	01110	0.0306635
				1111	0.0639575	01111	0.0311662
						10000	0.0341823
						10001	0.027815
						10010	0.0325067
						10011	0.0303284
						10100	0.0266421
						10101	0.0291555
						10110	0.0309987
						10111	0.0299933
						11000	0.0298257
						11001	0.0293231
						11010	0.0286528
						11011	0.0333445
						11100	0.0343499
						11101	0.0365282
						11110	0.0309987
						11111	0.0313338

Значення автокореляції:

L_1:	L_2:
d = 1 : 4194304	d = 1 : 17864
d = 2 : 4194304	d = 2 : 17808
d = 3 : 4194304	d = 3 : 17864
d = 4 : 4194304	d = 4 : 17864
d = 5 : 4194304	d = 5 : 17864
d = 6 : 4194304	d = 6 : 17864
d = 7 : 4194304	d = 7 : 17808
d = 8 : 4194304	d = 8 : 17976
d = 9 : 4194304	d = 9 : 17976
d = 10 : 4194304	d = 10 : 17976

Висновок:

В даному комп'ютерному практикумі було набуто навичок роботи з лінійними регістрами зсуву, а саме: їх програмна реалізація, дослідження властивостей характеристичного полінома регістра. Окрім цього було досліджено властивості лінійних рекурентних послідовностей

Програмна реалізація:

```
#include<iostream>
#include<fstream>
#include<map>
#include<vector>
#include<ctime>
#include <string>
using namespace std;

bool LFSRoneTact(vector<bool>, vector<bool>& ); // One tact of linear feedback shift register
int LFSR(vector<bool>, vector<bool>& , vector<bool>& , string);
int GetAutoCorCoef(vector<bool> Sequence, int Step); // Auto correlation Coefficient
void GetAutoCorCoef(vector<bool> Sequence);
void PolynomialType(int PolyDegree, int Period);

void initNgramMap( vector<bool> Sequence, int NgramSize, string FilePath);

int main() {
	//-------------- First polynomial -----------
	vector<bool> CoefP1 = { 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0 };
	vector<bool> InitialStateP1(CoefP1.size(), 0);
	InitialStateP1.back() = 1; // //Initial state of register for impulse function
	vector<bool> SequenceP1; // Generated sequence

	//-------------- Second polynomial ----------
	vector<bool> CoefP2 = { 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 0 };
	vector<bool> InitialStateP2(CoefP2.size(), 0);
	InitialStateP2.back() = 1; // //Initial state of register for impulse function
	vector<bool> SequenceP2; // Generated sequence

	// First LFSR
	int PeriodP1;
	PeriodP1 = LFSR(CoefP1, InitialStateP1, SequenceP1, "..\\..\\Generated Sequence Var.18 P1.txt");
	cout << PeriodP1 << endl;
	// Second LFSR
	int PeriodP2;
	PeriodP2 = LFSR(CoefP2, InitialStateP2, SequenceP2, "..\\..\\Generated Sequence Var.18 P2.txt");
	cout << PeriodP2 << endl;

	GetAutoCorCoef(SequenceP1);
	GetAutoCorCoef(SequenceP2);

	PolynomialType(CoefP1.size(), PeriodP1);
	PolynomialType(CoefP2.size(), PeriodP2);

	for (int i = 2; i < 6; i++) {
		initNgramMap(SequenceP1, i, "..\\..\\NgramsP1\\" + to_string(i) + "gram Map.txt");
	}
	for (int i = 2; i < 6; i++) {
		initNgramMap(SequenceP2, i, "..\\..\\NgramsP2\\" + to_string(i) + "gram Map.txt");
	}


	system("pause");
	return 0;
}
bool LFSRoneTact(vector<bool>FeedbackCoef, vector<bool>& Register) {
	bool ShiftedBit = 0; // First bit of register, that will be shifted
	bool GeneratedBit; // Bit generated by LFSR in one tact
	ShiftedBit = Register[0];
	int Sum = 0; // Sum of composition Feedback[i] and Register[i]
	for (int i = 0; i < Register.size(); i++) {
		if (FeedbackCoef[i] != 0) {
			Sum += FeedbackCoef[i] * Register[i];
		}
	}
	GeneratedBit = Sum % 2;
	//Bits swaping
	for (int i = 0; i < Register.size() - 1; i++) {
		Register[i] = Register[i + 1];
	}
	Register.back() = GeneratedBit;
	return ShiftedBit;
}
int LFSR(vector<bool>FeedbackCoef, vector<bool>& InitialState, vector<bool>& Sequence, string FilePath) {
	ofstream fout(FilePath);
	vector<bool> Register;
	int Period = 0;
	Register = InitialState;
	bool TempBool;
	do {
```

```
                Period++;
                TempBool = LFSRoneTact(FeedbackCoef, Register);
                fout << TempBool;
                Sequence.push_back(TempBool);
                //if (Period % 100000 == 0) cout << Period << " ";
        } while (Register != InitialState);
        return Period;
}

int GetAutoCorCoef(vector<bool> Sequence, int Step) {
        int ACF = 0;
        for (int i = 0; i < Sequence.size(); i++) {
                ACF += (Sequence[i] + Sequence[(i + Step) % Sequence.size()]) % 2;
        }
        return ACF;
}
void GetAutoCorCoef(vector<bool> Sequence) {
        for (int i = 1; i < 11; i++) {
                cout << "AutoCorCoef d = " << i << " : " << GetAutoCorCoef(Sequence, i) << endl;
        }
        cout << endl;
}

void PolynomialType(int PolyDegree, int Period) {
        if (Period == pow(2, PolyDegree) - 1) {
                cout << "Polynomial is primitive" << endl;
        }
        else if(pow(2, PolyDegree) - 1 % Period == 0){
                cout << "Polynomial isn't primitive and not reducible" << endl;
        }
        else {
                cout << "Polynomial isn't primitive, but reducible" << endl;
        }
}

void initNgramMap(vector<bool> Sequence, int NgramSize, string FilePath) {
        ofstream fout(FilePath);
        map<string, double> Map;
        string TempNgram;
        bool TempBool;
        double Amount = 0;
        for (int i = 0; i < Sequence.size(); i += 1 + NgramSize) {
                for (int j = 0; j < NgramSize; j++) {
                        TempBool = Sequence[(i + j) % Sequence.size()];
                        TempNgram += to_string(TempBool);
                }
                if (Map.count(TempNgram)) {
                        Amount++;
                        Map.at(TempNgram)++;
                }
                else {
                        Amount++;
                        Map.emplace(TempNgram, 1);
                }
                TempNgram.clear();
        }
        for (auto it = Map.cbegin(); it != Map.cend(); it++) {
                fout << it->first << "      " << it->second / Amount << endl;
        }
                                                        }
```