

Міністерство освіти і науки України

НТУУ«Київський політехнічний інститут»

Фізико-технічний інститут

**CИМЕТРИЧНА КРИПТОГРАФІЯ**

**КОМП’ЮТЕРНИЙ ПРАКТИКУМ №4**

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

**Виконали:**

Студенти 3 курсу ФТІ

групи ФІ-52

Шевченко Олексій

Тузовська Марія

**Прийняв:**

Яковлев С. В.

Київ 2018

**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 (3) – (5) при заданому визначити кількість знаків

вихідної послідовності *N\** , необхідну для знаходження вірного початкового заповнення, а

також поріг *С* для регістрів *L1* та 2 *L* .

3. Організувати перебір всіх можливих початкових заповнень *L1* i обчислення

відповідних статистик *R* з використанням заданої послідовності (zi).

4. Відбракувати випробувані варіанти за критерієм *R* *C* і знайти всі кандидати на

істинне початкове заповнення *L1* .

5. Аналогічним чином знайти початкове заповнення *L2*.

6. Організувати перебір всіх початкових заповнень *L3* та генерацію відповідних

послідовностей (si).

7. Відбракувати невірні початкові заповнення *L*3 за тактами, на яких *x i* *yi* , де

(*x i*), (*y i*)– послідовності, що генеруються регістрами *L*1 та *L*2при знайдених початкових

заповненнях.

8. Перевірити знайдені початкові заповнення ЛРЗ *L*1, *L*2, *L*3шляхом співставлення

згенерованої послідовності (*z i*) із заданою при *i* 0,*N* 1.

**Код програми:**

#include "cuda\_runtime.h"

#include "device\_launch\_parameters.h"

#include "math\_functions.h"

#include "device\_atomic\_functions.h"

#include <fstream>

#include <iostream>

#include <string>

#include <ctime>

#include <iomanip>

using namespace std;

//Params for statistical hypothesis testing

const int Blocks = 14;

\_\_device\_\_ \_\_constant\_\_ int C = 133;

\_\_device\_\_ \_\_constant\_\_ int N = 32 \* Blocks;

\_\_device\_\_ \_\_constant\_\_ int BLOCKS = Blocks;

//L1 Buffer

\_\_device\_\_ \_\_managed\_\_ unsigned int l1\_buffer[8] = { 0 };

\_\_device\_\_ \_\_managed\_\_ unsigned int l1\_gamma\_buffer[8][Blocks] = { 0 };

\_\_device\_\_ \_\_managed\_\_ unsigned int l1\_buffer\_counter = 0;

//L2 Buffer

\_\_device\_\_ \_\_managed\_\_ unsigned int l2\_buffer[24] = { 0 };

\_\_device\_\_ \_\_managed\_\_ unsigned int l2\_gamma\_buffer[24][Blocks] = { 0 };

\_\_device\_\_ \_\_managed\_\_ unsigned int l2\_buffer\_counter = 0;

//L3 Buffer

\_\_device\_\_ \_\_managed\_\_ unsigned int l3\_buffer[16] = { 0 };

\_\_device\_\_ \_\_managed\_\_ unsigned int l3\_buffer\_counter = 0;

//LFSR structure

\_\_device\_\_ \_\_constant\_\_ unsigned int l1\_feedback = 0b00110010100000000000000000000000;

\_\_device\_\_ \_\_constant\_\_ unsigned int l2\_feedback = 0b01001000000000000000000000000000;

\_\_device\_\_ \_\_constant\_\_ unsigned int l3\_feedback = 0b11110101000000000000000000000000;

//Service functions

unsigned int\* read\_data(string data\_file, int& n);

bool gamma\_cheack(const unsigned int\* gamma, int n, unsigned int l1, unsigned int l2, unsigned int l3);

void lfsr\_roll\_back(unsigned int& l1, unsigned int& l2, unsigned int& l3, int n);

unsigned int host\_parity(unsigned int n);

string bin(unsigned int);

//Kernals

\_\_global\_\_ void l1\_register\_brute\_force(const unsigned int\* gamma, const unsigned int state\_prefix);

\_\_global\_\_ void l2\_register\_brute\_force(const unsigned int\* gamma, const unsigned int state\_prefix);

\_\_global\_\_ void l3\_register\_brute\_force(const unsigned int\* gamma, const unsigned int l1\_index, const unsigned int l2\_index, const unsigned int state\_prefix);

int main(int argc, char\* argv[]) {

//Reading data from file

string file(argv[1]);

int n = 0;

unsigned int\* host\_gamma = read\_data(file, n);

unsigned int\* device\_gamma;

unsigned int MAX\_ROUND = n / Blocks;

bool target = false;

time\_t start;

time(&start);

for (int i = 0; i < MAX\_ROUND; i++) {

//Duplication gamma from host memory to device memory

cudaMalloc(&device\_gamma, Blocks \* sizeof(unsigned int));

cudaMemcpy(device\_gamma, &host\_gamma[i \* Blocks], Blocks \* sizeof(unsigned int), cudaMemcpyHostToDevice);

//Brute force

//Kernals params

unsigned int threadsPerBlock = 1 << 5;

unsigned int blocksPerGrig = 1 << 23;

float progress = 0;

cout << "Attack round: " << i << endl;

//L1 brute force

cout << "L1 brute force: " << endl;

for (unsigned int r = 0; r < 4; r++) {

l1\_register\_brute\_force <<< blocksPerGrig, threadsPerBlock >>> (device\_gamma, r);

cudaDeviceSynchronize();

progress = (((float)r) / 4) \* 100;

cout << fixed << setprecision(2) << "\r\t[Progress: " << progress << "%]";

}

cout << "\r\t[Progress: complete]" << endl;

/\*

unsigned int test;

cudaMemcpy(&test, &l1\_buffer\_counter, sizeof(unsigned int), cudaMemcpyDeviceToHost);

cout << "l1 " << test << endl;

\*/

//L2 brute force

cout << "L2 brute force: " << endl;

for (unsigned int r = 0; r < 8; r++) {

l2\_register\_brute\_force <<< blocksPerGrig, threadsPerBlock >>> (device\_gamma, r);

cudaDeviceSynchronize();

progress = (((float)r) / 8) \* 100;

cout << fixed << setprecision(2) << "\r\t[Progress: " << progress << "%]";

}

cout << "\r\t[Progress: complete]" << endl;

/\*

cudaMemcpy(&test, &l2\_buffer\_counter, sizeof(unsigned int), cudaMemcpyDeviceToHost);

cout << "l2 " << test << endl;

\*/

//L3 brute force

unsigned int J, K;

cudaMemcpy(&J, &l1\_buffer\_counter, sizeof(unsigned int), cudaMemcpyDeviceToHost);

cudaMemcpy(&K, &l2\_buffer\_counter, sizeof(unsigned int), cudaMemcpyDeviceToHost);

if ((J == 0) || (J >= 8)) {

unsigned int zero = 0;

cudaMemcpy(&l1\_buffer\_counter, &zero, sizeof(unsigned int), cudaMemcpyHostToDevice);

cudaMemcpy(&l2\_buffer\_counter, &zero, sizeof(unsigned int), cudaMemcpyHostToDevice);

cout << "L3 brute force: [failed]" << endl;

continue;

}

if ((K == 0) || (K >= 24)) {

unsigned int zero = 0;

cudaMemcpy(&l1\_buffer\_counter, &zero, sizeof(unsigned int), cudaMemcpyHostToDevice);

cudaMemcpy(&l2\_buffer\_counter, &zero, sizeof(unsigned int), cudaMemcpyHostToDevice);

cout << "L3 brute force: [failed]" << endl;

continue;

}

cudaDeviceSynchronize();

cout << "L3 brute force:" << endl;

for (int j = 0; j < J; j++) {

for (int k = 0; k < K; k++) {

//L3 brute force kernal

for (int r = 0; r < 16; r++) {

l3\_register\_brute\_force <<< blocksPerGrig, threadsPerBlock >>> (device\_gamma, j, k, r);

cudaDeviceSynchronize();

progress = (((float)(j \* 16 \* K + k \* 16 + r)) / (J \* K \* 16)) \* 100;

cout << fixed << setprecision(2) << "\r\t[Progress: " << progress << "%]";

}

/\*

cudaMemcpy(&test, &l3\_buffer\_counter, sizeof(unsigned int), cudaMemcpyDeviceToHost);

cout << "l3 " << test << endl;

\*/

//Gamma cheack

unsigned int M;

cudaMemcpy(&M, &l3\_buffer\_counter, sizeof(unsigned int), cudaMemcpyDeviceToHost);

if ((M == 0) || (M >= 16)) {

continue;

}

unsigned int l1\_state, l2\_state, l3\_state;

cudaMemcpy(&l1\_state, &l1\_buffer[j], sizeof(unsigned int), cudaMemcpyDeviceToHost);

cudaMemcpy(&l2\_state, &l2\_buffer[k], sizeof(unsigned int), cudaMemcpyDeviceToHost);

for (int m = 0; m < M; m++) {

cudaMemcpy(&l3\_state, &l3\_buffer, sizeof(unsigned int), cudaMemcpyDeviceToHost);

lfsr\_roll\_back(l1\_state, l2\_state, l3\_state, i \* 32 \* Blocks);

target = gamma\_cheack(host\_gamma, 32 \* n, l1\_state, l2\_state, l3\_state);

if (target) {

cout << "\r\t[Progress: complete]" << endl;

cout << "Result:" << endl;

cout << "L1 state: " << bin(l1\_state) << " (" << hex << l1\_state << ")" << endl;

cout << "L2 state: " << bin(l2\_state) << " (" << hex << l2\_state << ")" << endl;

cout << "L3 state: " << bin(l3\_state) << " (" << hex << l3\_state << ")" << endl;

break;

}

}

//Params reset

M = 0;

cudaMemcpy(&l3\_buffer\_counter, &M, sizeof(unsigned int), cudaMemcpyHostToDevice);

if (target)

break;

}

if (target)

break;

}

if (target)

break;

cout << "\r\t[Progress: complete]" << endl;

cout << "Attack round: [failed]" << endl;

//Params reset

unsigned int zero = 0;

cudaMemcpy(&l1\_buffer\_counter, &zero, sizeof(unsigned int), cudaMemcpyHostToDevice);

cudaMemcpy(&l2\_buffer\_counter, &zero, sizeof(unsigned int), cudaMemcpyHostToDevice);

//Memory free

cudaFree(device\_gamma);

}

time\_t end;

time(&end);

double time = difftime(end, start);

cout << "Time: " << time << " seconds" << endl;

//system("pause");

//Memory free

delete[] host\_gamma;

cudaFree(device\_gamma);

return 0;

}

void lfsr\_roll\_back(unsigned int& l1, unsigned int& l2, unsigned int& l3, int n) {

unsigned int l1\_fb = 0b00110010100000000000000000000000;

unsigned int l2\_fb = 0b01001000000000000000000000000000;

unsigned int l3\_fb = 0b11110101000000000000000000000000;

if (n == 0)

return;

for (int i = 0; i < n; i++) {

unsigned int out\_1, out\_2, out\_3;

out\_1 = l1 & 0x1;

out\_2 = l2 & 0x1;

out\_3 = l3 & 0x1;

l1 = l1 >> 1;

l2 = l2 >> 1;

l3 = l3 >> 1;

if (host\_parity(l1 & l1\_fb) != out\_1)

l1 ^= (1 << 29);

if (host\_parity(l2 & l2\_fb) != out\_2)

l2 ^= (1 << 30);

if (host\_parity(l3 & l3\_fb) != out\_3)

l3 ^= (1 << 31);

}

}

string bin(unsigned int n) {

string res = "";

for (int i = 0; i < 32; i++) {

if (i % 4 == 0)

res = " " + res;

if ((n & 0x1) == 1)

res = '1' + res;

else

res = '0' + res;

n = n >> 1;

}

return res;

}

inline \_\_device\_\_ unsigned int parity(unsigned int x) {

return \_\_popc(x) & 0b1;

}

inline \_\_device\_\_ unsigned int weigth(unsigned int x) {

return \_\_popc(x);

}

unsigned int\* read\_data(string data\_file, int& n) {

string data;

ifstream in(data\_file);

in >> data;

in.close();

n = ceill(data.length() / 32);

unsigned int\* gamma = new unsigned int[n];

for (int i = 0; i < n; i++)

gamma[i] = 0;

for (int i = 0; i < data.length(); i++)

if (data[i] == '1')

gamma[int(i / 32)] += (1 << (i % 32));

return gamma;

}

bool gamma\_cheack(const unsigned int\* gamma, int n, unsigned int l1, unsigned int l2, unsigned int l3) {

unsigned int l1\_fb = 0b00110010100000000000000000000000;

unsigned int l2\_fb = 0b01001000000000000000000000000000;

unsigned int l3\_fb = 0b11110101000000000000000000000000;

for (int i = 0; i < (int(n / 32)); i++) {

unsigned int l1\_gamma = 0;

unsigned int l2\_gamma = 0;

unsigned int l3\_gamma = 0;

unsigned int out;

for (int j = 0; j < 32; j++) {

out = host\_parity(l1 & l1\_fb);

l1\_gamma += (((l1 >> 29) & 0x1) << j);

l1 = (l1 << 1) ^ out;

out = host\_parity(l2 & l2\_fb);

l2\_gamma += (((l2 >> 30) & 0x1) << j);

l2 = (l2 << 1) ^ out;

out = host\_parity(l3 & l3\_fb);

l3\_gamma += (((l3 >> 31) & 0x1) << j);

l3 = (l3 << 1) ^ out;

}

unsigned int temp = (((l1\_gamma ^ l2\_gamma) & l3\_gamma) ^ l2\_gamma) ^ gamma[i];

if (temp != 0)

return false;

}

return true;

}

unsigned int host\_parity(unsigned int n) {

n = (n >> 16) ^ n;

n = (n >> 8) ^ n;

n = (n >> 4) ^ n;

n = (n >> 2) ^ n;

n = (n >> 1) ^ n;

return (n & 0x1);

}

\_\_global\_\_ void l1\_register\_brute\_force(const unsigned int\* gamma, const unsigned int state\_prefix) {

unsigned int s = (state\_prefix << 28) ^ (blockIdx.x << 5) ^ (threadIdx.x);

unsigned int R = 0;

if (s == 0)

return;

for (int i = 0; i < BLOCKS; i++) {

unsigned int temp = 0;

for (int j = 0; j < 32; j++) {

unsigned int out = parity(s & l1\_feedback);

temp += (((s >> 29) & 0x1) << j);

s = (s << 1) ^ out;

}

R += weigth(gamma[i] ^ temp);

}

if (R < C) {

unsigned int i = atomicAdd(&l1\_buffer\_counter, 1);

if (i >= 8)

return;

s = (state\_prefix << 28) ^ (blockIdx.x << 5) ^ (threadIdx.x);

l1\_buffer[i] = s;

for (int j = 0; j < BLOCKS; j++) {

unsigned int temp = 0;

for (int k = 0; k < 32; k++) {

unsigned int out = parity(s & l1\_feedback);

temp += (((s >> 29) & 0x1) << k);

s = (s << 1) ^ out;

}

l1\_gamma\_buffer[i][j] = temp;

}

}

}

\_\_global\_\_ void l2\_register\_brute\_force(const unsigned int\* gamma, const unsigned int state\_prefix) {

unsigned int s = (state\_prefix << 28) ^ (blockIdx.x << 5) ^ (threadIdx.x);

unsigned int R = 0;

if (s == 0)

return;

for (int i = 0; i < BLOCKS; i++) {

unsigned int temp = 0;

for (int j = 0; j < 32; j++) {

unsigned int out = parity(s & l2\_feedback);

temp += (((s >> 30) & 0x1) << j);

s = (s << 1) ^ out;

}

R += weigth(gamma[i] ^ temp);

}

if (R < C) {

unsigned int i = atomicAdd(&l2\_buffer\_counter, 1);

if (i >= 24)

return;

s = (state\_prefix << 28) ^ (blockIdx.x << 5) ^ (threadIdx.x);

l2\_buffer[i] = s;

for (int j = 0; j < BLOCKS; j++) {

unsigned int temp = 0;

for (int k = 0; k < 32; k++) {

unsigned int out = parity(s & l2\_feedback);

temp += (((s >> 30) & 0x1) << k);

s = (s << 1) ^ out;

}

l2\_gamma\_buffer[i][j] = temp;

}

}

}

\_\_global\_\_ void l3\_register\_brute\_force(const unsigned int\* gamma, const unsigned int l1\_index, const unsigned int l2\_index, const unsigned int state\_prefix) {

unsigned int s = (state\_prefix << 28) ^ (blockIdx.x << 5) ^ (threadIdx.x);

if (s == 0)

return;

for (int i = 0; i < BLOCKS; i++) {

unsigned int temp = 0;

for (int j = 0; j < 32; j++) {

unsigned int out = parity(s & l3\_feedback);

temp += (((s >> 31) & 0x1) << j);

s = (s << 1) ^ out;

}

temp = (((l1\_gamma\_buffer[l1\_index][i] ^ l2\_gamma\_buffer[l2\_index][i]) & temp) ^ l2\_gamma\_buffer[l2\_index][i]) ^ gamma[i];

if (temp != 0)

return;

}

unsigned int i = atomicAdd(&l3\_buffer\_counter, 1);

if (i >= 16)

return;

l3\_buffer[i] = (state\_prefix << 28) ^ (blockIdx.x << 5) ^ (threadIdx.x);

}

**Варіант 4**

L1 state: 1100111011110101100001

L2 state: 11100010001100001101

L3 state: 101100011100011100001111000

**Варіант 6**

L1 state: 1001001110110001100010011

L2 state: 10100100011010100110101000

L3 state: 111110101001101000100010010

**Висновок:**

Ознайомлені з деякими принципами побудови криптосистем на лінійних регістрах зсуву; практично засвоїли програмну реалізацію лінійних регістрів зсуву (ЛРЗ); ознайомилися з методом кореляційного аналізу криптосистем на прикладі генератора Джиффі.