Міністерство освіти і науки України
Національний технічний університет України
"Київський політехнічний інститут"
Фізико-технічний інститут

Розрахункова робота

з дисципліни

Симетрична криптографія

Варіант 18

Виконав:
Грубіян Є.О.
Прийняв:
Яковлєв С.В.

Київ 2016 р.

Вступ

Тема розрахункової роботи - дослідження криптографічних властивостей булевих функцій. У запропонованому варіанті пропонуються дві булеві функції для дослідження, а саме: $f(x) = x^{16257}$ та $g(x) = x^{16256}$, де $x \in GF(2^{15})$. В якості полінома генератора поля $GF(2^{15})$ використовується $p(x) = x^{15} + x + 1$.

Програмний код розрахункової роботи написаний на мові C/C++, в силу того, що робота містить багато складних у обчислювальному плані процедур над полем $GF(2^{15})$, тому швидкодія є важливим аспектом. Вихідний код також доступний на ресурсі GitHub: `https://github.com/juja256/boolean`. Всі вказані параметри для двох булевих функцій в роботі обчислюються близько 20 хвилин на процесорі Intel Core i3-2310m @ 2.10 GHz x 2. Програма тестувалась під ОС Windows 10(Visual Studio 2015) та Linux Mint 17.3(GCC 4.8.4, рівень оптимізації -O3).

Програмний код

Listing 1: core.h

```
#ifndef CORE_H
#define CORE_H

#define XOR_EXC 0x00000001
#define RES_EXC 0x00000002
#define FLW_EXC 0x00000003
#define EVL_EXC 0x00000004

typedef unsigned long long u64;
typedef unsigned int u32;
typedef unsigned char u8;

class BooleanException {
        u32 code;
public:
        BooleanException();
        BooleanException(u32 code);
};

class BooleanVector {
        u64 size;
        u32 blocks;
        u32* a;
        bool heap_alloc;
public:
        explicit BooleanVector(u32 a);
        explicit BooleanVector(u64 a);
        BooleanVector(u32* c, u64 size);
        BooleanVector(const BooleanVector& vec);
        BooleanVector();
        ~BooleanVector();

        u32 GetBlocks() const;
        u64 GetSize() const;
        u32 Deg() const;
        BooleanVector GetInverse();
        u32 GetInt(u32 index) const;
        bool operator[](u32 i) const;
        BooleanVector operator^(const BooleanVector& v) const;
```

```
        BooleanVector operator*(const BooleanVector& v);
        BooleanVector& operator=(const BooleanVector& v);
        void operator++();
        BooleanVector Pow(u32 p, const BooleanVector& v) const;
        bool SetBit(u64 index, u32 b);
        void Annulate();
        friend BooleanVector operator%(const BooleanVector& v1, const
           BooleanVector& v2);
        friend BooleanVector operator<<(const BooleanVector& v, u32 p);
};

BooleanVector operator<<(const BooleanVector& v, u32 p);
BooleanVector operator%(const BooleanVector& v1, const BooleanVector& v2);

class BooleanFunction {
        u32 n;
        u32 m;
        BooleanVector* table;
        u64 in_dim;
        double** cached_FS;
        int** cached_WS;
        BooleanVector(*function_ptr)(const BooleanVector&);
public:
        BooleanFunction();
        BooleanFunction(u32 n_, u32 m_, BooleanVector* table_);
        BooleanFunction(u32 n_, u32 m_, BooleanVector (*function_ptr)(const
           BooleanVector&));
        BooleanFunction(const BooleanFunction&);
        ~BooleanFunction();
        BooleanVector GetCordinateVector(u32 i);
        BooleanVector Eval(const BooleanVector& vec);
        double* GetFourierSpectrum(u32 i);
        int* GetWalshSpectrum(u32 i);
        BooleanVector GetAlgebraicNormalForm(u32 i);
        u32 GetAlgebraicDegree();
        u32 GetAlgebraicDegree(u32 i);
        u64 GetUnlinearity(u32 i);
        u32 GetDissballance(u32 i);
        int GetCorrelationImmunityLevel(u32 i);
        u32 GetErrorExpandingCoefficient(u32 index, u32 v);
        u32 GetErrorExpandingCoefficientAverage(u32 v);
        bool GetAvalancheEffectZeroLevel(u32 i);
        bool GetAvalancheEffectZeroLevel();
        bool GetAvalancheEffectAverage();
        BooleanVector* Derivative(const BooleanVector& a);
        double GetMaximumDifferentialProbability();
        void Dump(const char* file);
};

/* algs */
BooleanVector ANF(const BooleanVector& outs, u32 n);
int* FFT(const BooleanVector& outs, u32 n);
int* WAT(const BooleanVector& outs, u32 n);
u32 HW(u32 i);
u32 HW(const BooleanVector& v);

#endif
```

Listing 2: algs.cpp

```
#include "core.h"
```

```
BooleanVector ANF(const BooleanVector& outs, u32 n) { // Fast Mebius
    Transform +++++
        BooleanVector res(outs);
        u64 s = outs.GetSize();
        u32 step, step2;
        for (u32 i = 1; i <= n; i++) {
                step = 1 << i;
                for (u32 j = 0; j < s; j += step) {
                        step2 = (step >> 1);
                        for (u32 k = j; k < j + step2; k += 1) {
                                res.SetBit(k + step2, res[k + step2] ^ res[k
                                    ]);
                        }
                }
        }
        return res;
}

int* FFT(const BooleanVector& outs, u32 n) {
        u64 s = outs.GetSize();
        int* spectre = new int[s];
        for (u32 i = 0; i < s; i++)
                spectre[i] = outs[i];

        u32 step, step2;
        int u0, u1;
        for (u32 i = 1; i <= n; i++) {
                step = 1 << i;
                for (u32 j = 0; j < s; j += step) {
                        step2 = (step >> 1);
                        for (u32 k = j; k < j + step2; k += 1) {
                                u0 = spectre[k];
                                u1 = spectre[k + step2];
                                spectre[k] = u0 + u1;
                                spectre[k + step2] = u0 - u1;
                        }
                }
        }
        return spectre;
}

int* WAT(const BooleanVector& outs, u32 n) {
        u64 s = outs.GetSize();
        int* spectre = new int[s];
        for (u32 i = 0; i < s; i++)
                spectre[i] = outs[i] ? -1 : 1;

        u32 step, step2;
        int u0, u1;
        for (u32 i = 1; i <= n; i++) {
                step = 1 << i;
                for (u32 j = 0; j < s; j += step) {
                        step2 = (step >> 1);
                        for (u32 k = j; k < j + step2; k += 1) {
                                u0 = spectre[k];
                                u1 = spectre[k + step2];
                                spectre[k] = u0 + u1;
                                spectre[k + step2] = u0 - u1;
                        }
```

```
                }
        }
        return spectre;
}

u32 HW(u32 i) {
        i = i - ((i >> 1) & 0x55555555);
        i = (i & 0x33333333) + ((i >> 2) & 0x33333333);
        return (((i + (i >> 4)) & 0x0F0F0F0F) * 0x01010101) >> 24;
}

u32 HW(const BooleanVector& v) {
        u32 w=0;
        for (u32 i = 0; i < v.GetBlocks(); i++) {
                w += HW(v.GetInt(i));
        }
        return w;
}
```

Listing 3: bool_vec.cpp

```
#include "core.h"

#define UN_INV 0x80000000
#define MAX_U32 0xffffffff

#ifdef _DEBUG
#include <iostream>
#define PRINT_VECTOR(v) for (u64 i=0;i<(v).GetBlocks()*32;i++) std::cout<<(v
    )[i];std::cout<<"\n";
#endif

BooleanException::BooleanException(): code(0) {}

BooleanException::BooleanException(u32 code_): code(code_) {}

BooleanVector::BooleanVector(): size(32), blocks(1), heap_alloc(true) {
        this->a = new u32[1];
        this->a[0] = 0;
}

BooleanVector::BooleanVector(u32 vec): size(32), blocks(1), heap_alloc(true)
    {
        this->a = new u32[1];
        this->a[0] = vec;
}

BooleanVector::BooleanVector(u32* c, u64 size_): size(size_), a(c),
    heap_alloc(false) {
        this->blocks = (size_ % 32 == 0) ? size_ / 32 : size_ / 32 + 1;
}

BooleanVector::BooleanVector(u64 size_): size(size_), heap_alloc(true) {
        if (size_ % 32 != 0) {
                this->blocks = size_/32 + 1;
        }
        else {
                this->blocks = size/32;
        }
        this->a = new u32[this->blocks];
```

```
        for (u32 i = 0; i<this->blocks; i++)
                this->a[i] = 0;
}

BooleanVector::BooleanVector(const BooleanVector& vec) { // proper copy

        this->size = vec.size;
        this->blocks = vec.blocks;
        this->a = new u32[vec.blocks];
        this->heap_alloc = true;
        for (u32 i=0; i<vec.blocks; i++) {
                this->a[i] = vec.a[i];
        }
}

BooleanVector& BooleanVector::operator=(const BooleanVector& vec) {
        if ((this->size != 0) && (this->heap_alloc))
                delete[] (this->a);
        this->size = vec.size;
        this->blocks = vec.blocks;
        this->a = new u32[vec.blocks];
        this->heap_alloc = true;
        for (u32 i=0; i<vec.blocks; i++)
                this->a[i] = vec.a[i];
        return *this;
}

BooleanVector::~BooleanVector() {
        if ((this->size != 0) && (this->heap_alloc))
                delete[] (this->a);
}

BooleanVector BooleanVector::GetInverse() {
        BooleanVector res(*this);
        for (u32 i=0; i<this->blocks; i++) {
                this->a[i] = ~(this->a[i]);
        }
        return res;
}

u64 BooleanVector::GetSize() const { return this->size; }
u32 BooleanVector::GetBlocks() const { return this->blocks; }

bool BooleanVector::operator[](u32 i) const {
        return (this->a[i/32] & ( 1 << (i%32) )) >> (i%32);
}

BooleanVector BooleanVector::operator^(const BooleanVector& vec) const {
        u32 b = (vec.blocks > this->blocks) ? this->blocks : vec.blocks;
        int b_ = (vec.blocks - this->blocks);

        u64 res_size = (vec.size > this->size) ? vec.size : this->size;
        BooleanVector res(res_size);
        for (u32 i=0; i<b; i++) {
                res.a[i] = (this->a[i]) ^ (vec.a[i]);
        }
        if (b_>0)
                for (u32 i=b; i<vec.blocks; i++) {
                        res.a[i] = vec.a[i];
                }
```

```
        else if (b_<0)
                for (u32 i=b; i<this->blocks; i++) {
                        res.a[i] = this->a[i];
                }
        return res;
}

BooleanVector operator<<(const BooleanVector& v, u32 p) {
        if (p == 0) {
                return BooleanVector(v);
        }
        u32 deg = v.Deg();
        int offset = deg - v.blocks * 32 + p;
        BooleanVector res;
        if (offset >= 0) {
                BooleanVector v2((u64)(v.blocks * 32 + offset));
                res = v ^ v2;
        }
        else {
                res = v;
        }

        int buf = 0;
        for (u32 i=0; i<res.blocks; i++) {
                u32 cur = res.a[i];
                res.a[i] <<= p;
                res.a[i] += buf;
                buf = (cur & (MAX_U32 << (32 - p))) >> (32 -p);
        }
        return res;
}

u32 BooleanVector::Deg() const { // Danger !!1
        u32 t=1;
        for (int i=(this->blocks)-1; i>=0; i--) {
                if (this->a[i] == 0)
                        continue;
                else {
                        u32 cur = this->a[i];
                        u32 c = 32;
                        while (c) {
                                if ((UN_INV & cur) >> 31)
                                        return c + i*32 -1;
                                cur <<= 1;
                                --c;
                        }
                }
        }
        return 0;
}

BooleanVector BooleanVector::operator*(const BooleanVector& vec) {
        BooleanVector res((u64)0);
        for (u32 i = 0; i < this->blocks; i++) {
                for (u32 j = 0; j<32; j++) {
                        if ((*this)[i*32 + j]) {
                                res = res ^ (vec << (i*32 + j));
                        }
                }
        }
```

```
        return res;
}

BooleanVector operator%(const BooleanVector& v1, const BooleanVector& v2) {
        u32 k;
        u32 n = v2.Deg();
        BooleanVector res(v1);
        while (res.Deg() >= n) {
                k = res.Deg();
                res = res ^ (v2 << k-n);
        }
        return res;
}

BooleanVector BooleanVector::Pow(u32 p, const BooleanVector& v) const {
        if (this->GetInt(0) == 0) {
                return BooleanVector();
        }
        BooleanVector e(p);
        BooleanVector b((u32)0x00000001);
        BooleanVector c(*this);
        u32 s = e.Deg();
        for (u32 i = 0; i <= s; i++) {
                if (e[i]) {
                        b = (b * c) % v;
                }
                c = (c * c) % v;
        }
        return b;
}

bool BooleanVector::SetBit(u64 index, u32 b) {
        u32 block = index / 32;
        u32 bit = index % 32;
        if (index >= this->size) throw BooleanException(FLW_EXC);
        this->a[block] ^= (-b ^ this->a[block]) & (1 << bit);
        return b;
}

u32 BooleanVector::GetInt(u32 index) const {
        if (index >= this->blocks) throw BooleanException(FLW_EXC);
        return this->a[index];
}

void BooleanVector::operator++() {
        this->a[this->blocks - 1]++;
}

void BooleanVector::Annulate() {
        for (u32 i = 0; i < this->blocks; i++)
                this->a[i] = 0;
}
```

Listing 4: bool_func.cpp

```
#include "core.h"
#include <math.h>
#include <stdlib.h>
#include <fstream>
#ifdef _OPENMP
```

```
#include <omp.h>
#endif
BooleanFunction::BooleanFunction(): n(0), m(0), table(0), function_ptr(0),
    cached_FS(0), cached_WS(0) {

}

BooleanFunction::BooleanFunction(u32 n_, u32 m_, BooleanVector* table_): n(
    n_), m(m_), table(table_), function_ptr(0) {
        cached_FS = new double*[m_];
        cached_WS = new int*[m_];
        for (u32 i = 0;i < m_;i++) {
                cached_FS[i] = 0;
                cached_WS[i] = 0;
        }
        this->in_dim = 0x00000001 << n_;
}

BooleanFunction::BooleanFunction(u32 n_, u32 m_, BooleanVector (*
    function_ptr_)(const BooleanVector&)): n(n_), m(m_), function_ptr(
    function_ptr_) {
        cached_FS = new double*[m_];
        cached_WS = new int*[m_];
        for (u32 i = 0;i < m_;i++) {
                cached_FS[i] = 0;
                cached_WS[i] = 0;
        }
        this->in_dim = (u64)1 << n_;
        this->table = new BooleanVector[in_dim];

        for (u64 i = 0; i < in_dim; i++) {
                this->table[i] = this->function_ptr(BooleanVector((u32)i));
        }
}

BooleanFunction::BooleanFunction(const BooleanFunction& f) {
        this->in_dim = f.in_dim;
        this->n = f.n;
        this->m = f.m;
        this->cached_FS = f.cached_FS;
        this->cached_WS = f.cached_WS;
        this->table = f.table;
}

BooleanFunction::~BooleanFunction() {
        if (this->table)
                delete[] this->table;
        for (u32 i = 0; i < this->m; i++) {
                if (this->cached_FS[i])
                        delete[] this->cached_FS[i];
                if (this->cached_WS[i])
                        delete this->cached_WS[i];
        }
        delete[] this->cached_FS;
        delete[] this->cached_WS;
}

BooleanVector BooleanFunction::GetCordinateVector(u32 index) {
        BooleanVector res(this->in_dim);
        for (u64 i = 0; i < this->in_dim; i++) {
```

```
                res.SetBit(i, this->table[i][index]);
        }
        return res;
}

BooleanVector BooleanFunction::GetAlgebraicNormalForm(u32 index) {
        BooleanVector vec = this->GetCordinateVector(index);
        return ANF(vec, this->n);
}

u32 BooleanFunction::GetAlgebraicDegree(u32 index) {
        BooleanVector anf = this->GetAlgebraicNormalForm(index);
        u32 max = 0;
        u32 hw;
        for (int i = 0; i < anf.GetSize(); i++) {
                hw = HW(i);
                if (anf[i] && (hw > max))
                        max = hw;
        }
        return max;
}

u32 BooleanFunction::GetAlgebraicDegree() {
        u32 max = 0, a;
        for (u32 i = 0;i < this->m; i++) {
                a = GetAlgebraicDegree(i);
                if (a > max)
                        max = a;
        }
        return max;
}

int* BooleanFunction::GetWalshSpectrum(u32 i) {
        if (!this->cached_WS[i])
                this->cached_WS[i] = WAT(this->GetCordinateVector(i), this->
                   n);
        return this->cached_WS[i];
}

double* BooleanFunction::GetFourierSpectrum(u32 i) {
        if (this->cached_FS[i])
                return this->cached_FS[i];
        int* v = FFT(this->GetCordinateVector(i), this->n);
        double* v2 = new double[this->in_dim];
        for (u64 j = 0; j < this->in_dim; j++)
                v2[j] = 1. * v[j] / this->in_dim;
        delete[] v;
        this->cached_FS[i] = v2;
        return v2;
}

u32 BooleanFunction::GetDissballance(u32 i) {
        if (!this->cached_WS[i]) {
                GetWalshSpectrum(i);
        }
        return abs(cached_WS[i][0]);
}

u64 BooleanFunction::GetUnlinearity(u32 i) {
        if (!this->cached_WS[i])
```

```
                GetWalshSpectrum(i);
        u64 max=0;
        for (u64 j = 0; j < this->in_dim; j++) {
                if (abs(cached_WS[i][j]) > max)
                        max = abs(cached_WS[i][j]);
        }
        return ((this->in_dim) >> 1) - max / 2;
}

int BooleanFunction::GetCorrelationImmunityLevel(u32 i) {
        if (!this->cached_WS[i])
                GetWalshSpectrum(i);
        bool fl = true;
        for (int k = n; k >= 0; k--) {
                for (u32 j = 0;j < this->in_dim; j++) {
                        if ((HW(j) <= k) && (this->cached_WS[i][j])) {
                                fl = false;
                                break;
                        }
                }
                if (fl)
                        return k;
                else
                        fl = true;
        }
        return -1;
}

BooleanVector BooleanFunction::Eval(const BooleanVector& vec) {
        if (vec.GetBlocks() != 1) throw BooleanException(EVL_EXC);
        return this->table[vec.GetInt(0)];
}

u32 BooleanFunction::GetErrorExpandingCoefficient(u32 index, u32 v) {
        u32 s = 0;
        BooleanVector x;
        BooleanVector e;
        e.SetBit(v, 1);
        for (u32 i = 0; i < this->in_dim; i++) {
                s += (this->Eval(x)[index] ^ this->Eval(x ^ e)[index]);
                ++x;
        }
        return s;
}

u32 BooleanFunction::GetErrorExpandingCoefficientAverage(u32 v) {
        u32 s = 0;
        BooleanVector x;
        BooleanVector e;
        e.SetBit(v, 1);
        for (u32 i = 0; i < this->in_dim; i++) {
                s += HW(this->Eval(x) ^ this->Eval(x ^ e));
                ++x;
        }
        return s;
}

bool BooleanFunction::GetAvalancheEffectZeroLevel(u32 index) {
        for (u32 i = 0; i < this->n; i++) {
                u32 n_ = 1 << (this->n - 1);
```

```
                if (this->GetErrorExpandingCoefficient(index, i) != n_)
                        return false;
        }
        return true;
}

bool BooleanFunction::GetAvalancheEffectZeroLevel() {
        for (u32 i = 0; i < this->m; i++) {
                if (!GetAvalancheEffectZeroLevel(i))
                        return false;
        }
        return true;
}

bool BooleanFunction::GetAvalancheEffectAverage() {
        u32 s = 0;
        for (u32 i = 0; i < this->n; i++) {
                s = this->GetErrorExpandingCoefficientAverage(i);
                if (s != this->m * (this->in_dim >> 1))
                        return false;
        }
        return true;
}

void BooleanFunction::Dump(const char* file) {
        std::ofstream of(file);
        of << "Truth table:\n";
        for (u32 i=0; i<this->in_dim; i++) {
                of << i << ": ";
                for (u32 j=0; j<this->m; j++) {
                        of << this->table[i][j];
                }
                of << "\n";
        }
}

BooleanVector* BooleanFunction::Derivative(const BooleanVector& a) {
        BooleanVector* t = new BooleanVector[this->in_dim];
        BooleanVector x;
        for (u32 i = 0; i < this->in_dim; i++) {
                t[i] = this->Eval(x) ^ this->Eval(x ^ a);
                ++x;
        }
        return t;
}

u32 GetMostFrequentOutputFrequency(BooleanVector* table, u32 in_dim, u32
   out_dim) {
        u32* t = new u32[out_dim]();
        u32 k = 0;
        u32 max = 0;
        for (int i = 0; i < in_dim; i++) {
                k = ++t[table[i].GetInt(0)];
                if (k > max) max = k;
        }
        delete[] t;
        return max;
}

double BooleanFunction::GetMaximumDifferentialProbability() {
```

```
        u32 out_dim = 0x00000001 << this->m;
        BooleanVector a((u32)1);
        u32 max = 0;
        u32 t;
        BooleanVector* D;

        for (int i = 1; i < this->in_dim; i++) {
                D = this->Derivative(a);
                t = GetMostFrequentOutputFrequency(D, this->in_dim, out_dim)
                    ;
                if (t > max)
                        max = t;
                ++a;
                delete[] D;
        }
        return (double)max / this->in_dim;
}
```

Listing 5: main.cpp

```
#include "core.h"
#include <iostream>
#include <iomanip>
#include <stdlib.h>
#include <time.h>

#define SPEEDTEST(block) { clock_t tStart = clock(); block; std::cout << std
    ::setprecision(4)\
         << " | " << (double)(clock() - tStart)/CLOCKS_PER_SEC <<"s"; }

#define PRINT_VECTOR(v) for (u32 in=0;in<(v).GetBlocks()*32;in++) std::cout
    <<(v)[in];std::cout<<"\n";

#define PRINT_INFO_COORD(func, coord) std::cout << "Coordinate function #"
    << coord+1;\
        SPEEDTEST(std::cout << "\nDissb: " << func.GetDissballance(coord);)
            ;\
        SPEEDTEST(std::cout << "\nDeg: " << func.GetAlgebraicDegree(coord);)
            ;\
        SPEEDTEST(std::cout << "\nNL: " << func.GetUnlinearity(coord););\
        SPEEDTEST(std::cout << "\nCIL: " << func.GetCorrelationImmunityLevel
            (coord););\
        std::cout << "\nError expanding coefficients:\n";\
        u32 coefs[15];\
        SPEEDTEST(\
        for (u32 in=0; in<15; in++) {\
                coefs[in] = func.GetErrorExpandingCoefficient(coord, in);\
                std::cout << coefs[in] << " ";\
        }\
        )\
        std::cout << "\nError expanding coefficients deviation:\n";\
        for (u32 in=0; in<15; in++) {\
                std::cout << std::setprecision(2) << (double)abs((int)coefs[
                    in] - 16384) * 100 / 16384 << "% ";\
        }\
        std::cout << "\n\n";

#define PRINT_INFO_FUNCTION(func) {std::cout << "Whole Function";\
        SPEEDTEST(std::cout << "\nDeg: " << func.GetAlgebraicDegree(););\
        std::cout << "\nError expanding coefficients in average:\n";\
```

```
        u32 coefs[15];\
        SPEEDTEST(\
        for (u32 in=0; in<15; in++) {\
                coefs[in] = func.GetErrorExpandingCoefficientAverage(in);\
                std::cout << coefs[in] << " ";\
        }\
        )\
        std::cout << "\nError expanding in average coefficients deviation:\n
           ";\
        for (u32 in=0; in<15; in++) {\
                std::cout << std::setprecision(2) << (double)abs((int)coefs[
                   in] - 245760) * 100 / 245760 << "% ";\
        }\
        SPEEDTEST(std::cout << "\nPresence of avalanche effect of zero level
           : " << func.GetAvalancheEffectZeroLevel(););\
        SPEEDTEST(std::cout << "\nPresence of avalanche effect in average: "
            << func.GetAvalancheEffectAverage(););\
        SPEEDTEST(\
        double mdp = func.GetMaximumDifferentialProbability();\
        std::cout << "\nMDP: " << mdp << " (" << std::setprecision(0) << mdp
            * 32768 << "/32768)";\
        );\
        std::cout << "\n\n";\
        }


BooleanVector f(const BooleanVector& v) {
        BooleanVector p((u32)32771);
        u32 N = 16257;
        return v.Pow(N, p);
}

BooleanVector g(const BooleanVector& v) {
        BooleanVector p((u32)32771);
        u32 N = 16256;
        return v.Pow(N, p);
}

int main() {
        BooleanFunction func1(15, 15, f);
        BooleanFunction func2(15, 15, g);
        std::cout << "------------ f(v) = v^16257 mod (x^15 + x +1)
           ------------\n";
        PRINT_INFO_FUNCTION(func1);
        for (u32 i = 0; i < 15; i++) {
                PRINT_INFO_COORD(func1, i);
        }
        std::cout << "\n------------ g(v) = v^16256 mod (x^15 + x+ 1)
           ------------\n";
        PRINT_INFO_FUNCTION(func2);
        for (u32 i = 0; i < 15; i++) {
                PRINT_INFO_COORD(func2, i);
        }

        #ifdef WIN32
        system("PAUSE");
        #endif
        return 0;
}
```

Звіт

Звіт містить всі криптографічні характеристики, які потрібно було обчислити та приблизний час виконання кожної процедури через вертикальну риску.

Listing 6: output.txt

```
------------ f(v) = v^16257 mod (x^15 + x +1) ------------
Whole Function
Deg: 8 | 0.09369s
Error expanding coefficients in average:
244864 246144 246016 245888 245376 245760 245632 245504 246016 245888 245632
    245376 245376 246016 246144  | 0.1707s
Error expanding in average coefficients deviation:
0.36% 0.16% 0.1% 0.052% 0.16% 0% 0.052% 0.1% 0.1% 0.052% 0.052% 0.16% 0.16%
   0.1% 0.16%
Presence of avalanche effect of zero level: 0 | 0.008724s
Presence of avalanche effect in average: 0 | 0.011s
MDP: 6.104e-05 (2/32768) | 574.2s

Coordinate function #1
Dissb: 0 | 0.001325s
Deg: 8 | 0.005998s
NL: 16256 | 0.000101s
CIL: 0 | 0.0002s
Error expanding coefficients:
16128 16384 16384 16384 16384 16384 16384 16384 16384 16384 16384 16256
   16384 16512 16384  | 0.1255s
Error expanding coefficients deviation:
1.6% 0% 0% 0% 0% 0% 0% 0% 0% 0% 0% 0.78% 0% 0.78% 0%

Coordinate function #2
Dissb: 0 | 0.001307s
Deg: 8 | 0.006072s
NL: 16256 | 7.1e-05s
CIL: 0 | 0.000182s
Error expanding coefficients:
16256 16384 16384 16384 16256 16256 16384 16384 16256 16512 16384 16256
   16384 16256 16384  | 0.1257s
Error expanding coefficients deviation:
0.78% 0% 0% 0% 0.78% 0.78% 0% 0% 0.78% 0.78% 0% 0.78% 0% 0.78% 0%

Coordinate function #3
Dissb: 0 | 0.001744s
Deg: 8 | 0.006086s
NL: 16256 | 7.4e-05s
CIL: 0 | 0.000188s
Error expanding coefficients:
16256 16384 16512 16256 16384 16384 16256 16384 16384 16256 16384 16384
   16512 16384 16384  | 0.1243s
Error expanding coefficients deviation:
0.78% 0% 0.78% 0.78% 0% 0% 0.78% 0% 0% 0.78% 0% 0% 0.78% 0% 0%

Coordinate function #4
Dissb: 0 | 0.0013s
Deg: 8 | 0.006081s
NL: 16256 | 9.3e-05s
CIL: 0 | 0.000205s
Error expanding coefficients:
16384 16256 16512 16384 16384 16384 16384 16384 16512 16384 16512 16384
   16256 16384 16384  | 0.1244s
```

```
Error expanding coefficients deviation:
0% 0.78% 0.78% 0% 0% 0% 0% 0% 0.78% 0% 0.78% 0% 0.78% 0% 0%

Coordinate function #5
Dissb: 0 | 0.001357s
Deg: 8 | 0.006097s
NL: 16256 | 7.2e-05s
CIL: 0 | 0.00018s
Error expanding coefficients:
16256 16384 16384 16384 16256 16384 16512 16256 16512 16384 16384 16384
    16256 16512 16384  | 0.1267s
Error expanding coefficients deviation:
0.78% 0% 0% 0% 0.78% 0% 0.78% 0.78% 0.78% 0% 0% 0% 0.78% 0.78% 0%

Coordinate function #6
Dissb: 0 | 0.001337s
Deg: 8 | 0.006221s
NL: 16256 | 0.000101s
CIL: 0 | 0.000189s
Error expanding coefficients:
16384 16512 16512 16384 16128 16256 16512 16256 16384 16384 16384 16384
    16256 16384 16384  | 0.1254s
Error expanding coefficients deviation:
0% 0.78% 0.78% 0% 1.6% 0.78% 0.78% 0.78% 0% 0% 0% 0% 0.78% 0% 0%

Coordinate function #7
Dissb: 0 | 0.001294s
Deg: 8 | 0.006057s
NL: 16256 | 7.3e-05s
CIL: 0 | 0.000187s
Error expanding coefficients:
16256 16384 16384 16512 16256 16384 16384 16256 16512 16384 16256 16512
    16256 16256 16384  | 0.1238s
Error expanding coefficients deviation:
0.78% 0% 0% 0.78% 0.78% 0% 0% 0.78% 0.78% 0% 0.78% 0.78% 0.78% 0.78% 0%

Coordinate function #8
Dissb: 0 | 0.001306s
Deg: 8 | 0.00602s
NL: 16256 | 7.4e-05s
CIL: 0 | 0.000182s
Error expanding coefficients:
16384 16384 16256 16384 16512 16512 16256 16384 16512 16512 16256 16384
    16512 16512 16384  | 0.1271s
Error expanding coefficients deviation:
0% 0% 0.78% 0% 0.78% 0.78% 0.78% 0% 0.78% 0.78% 0.78% 0% 0.78% 0.78% 0%

Coordinate function #9
Dissb: 0 | 0.001293s
Deg: 8 | 0.006347s
NL: 16256 | 9.6e-05s
CIL: 0 | 0.00019s
Error expanding coefficients:
16256 16384 16256 16384 16256 16384 16384 16384 16384 16384 16256 16512
    16384 16384 16512  | 0.1247s
Error expanding coefficients deviation:
0.78% 0% 0.78% 0% 0.78% 0% 0% 0% 0% 0% 0.78% 0.78% 0% 0% 0.78%

Coordinate function #10
Dissb: 0 | 0.001314s
```

```
Deg: 8 | 0.006571s
NL: 16256 | 9.3e-05s
CIL: 0 | 0.000203s
Error expanding coefficients:
16384 16512 16384 16384 16512 16512 16256 16384 16384 16384 16512 16384
   16384 16512 16512  | 0.125s
Error expanding coefficients deviation:
0% 0.78% 0% 0% 0.78% 0.78% 0.78% 0% 0% 0% 0.78% 0% 0% 0.78% 0.78%

Coordinate function #11
Dissb: 0 | 0.001275s
Deg: 8 | 0.005948s
NL: 16256 | 6.9e-05s
CIL: 0 | 0.000188s
Error expanding coefficients:
16384 16512 16128 16512 16384 16384 16256 16384 16256 16256 16384 16256
   16384 16384 16512  | 0.1239s
Error expanding coefficients deviation:
0% 0.78% 1.6% 0.78% 0% 0% 0.78% 0% 0.78% 0.78% 0% 0.78% 0% 0% 0.78%

Coordinate function #12
Dissb: 0 | 0.001369s
Deg: 8 | 0.006153s
NL: 16256 | 0.000145s
CIL: 0 | 0.000223s
Error expanding coefficients:
16384 16256 16512 16256 16512 16256 16512 16384 16384 16384 16384 16256
   16512 16384 16384  | 0.1233s
Error expanding coefficients deviation:
0% 0.78% 0.78% 0.78% 0.78% 0.78% 0.78% 0% 0% 0% 0% 0.78% 0.78% 0% 0%

Coordinate function #13
Dissb: 0 | 0.001296s
Deg: 8 | 0.006091s
NL: 16256 | 0.00012s
CIL: 0 | 0.000226s
Error expanding coefficients:
16384 16384 16512 16256 16384 16512 16384 16512 16512 16384 16256 16384
   16256 16384 16384  | 0.1324s
Error expanding coefficients deviation:
0% 0% 0.78% 0.78% 0% 0.78% 0% 0.78% 0.78% 0% 0.78% 0% 0.78% 0% 0%

Coordinate function #14
Dissb: 0 | 0.001345s
Deg: 8 | 0.006156s
NL: 16256 | 7.2e-05s
CIL: 0 | 0.000182s
Error expanding coefficients:
16384 16512 16512 16512 16384 16384 16512 16384 16384 16512 16384 16384
   16256 16384 16384  | 0.1256s
Error expanding coefficients deviation:
0% 0.78% 0.78% 0.78% 0% 0% 0.78% 0% 0% 0.78% 0% 0% 0.78% 0% 0%

Coordinate function #15
Dissb: 0 | 0.001286s
Deg: 8 | 0.006061s
NL: 16256 | 9.6e-05s
CIL: 0 | 0.000188s
Error expanding coefficients:
```

```
16384 16512 16384 16512 16384 16384 16256 16384 16256 16384 16512 16256
   16384 16384 16384  | 0.1245s
Error expanding coefficients deviation:
0% 0.78% 0% 0.78% 0% 0% 0.78% 0% 0.78% 0% 0.78% 0.78% 0% 0% 0%


------------ g(v) = v^16256 mod (x^15 + x+ 1) ------------
Whole Function
Deg: 7 | 0.09162s
Error expanding coefficients in average:
245608 245288 244792 244352 245752 245352 244824 246272 246136 246496 246072
    245024 245232 245648 245472  | 0.1697s
Error expanding in average coefficients deviation:
0.062% 0.19% 0.39% 0.57% 0.0033% 0.17% 0.38% 0.21% 0.15% 0.3% 0.13% 0.3%
   0.21% 0.046% 0.12%
Presence of avalanche effect of zero level: 0 | 0.008428s
Presence of avalanche effect in average: 0 | 0.01138s
MDP: 0.0002441 (8/32768) | 562s

Coordinate function #1
Dissb: 0 | 0.001611s
Deg: 7 | 0.00641s
NL: 16044 | 7.5e-05s
CIL: 0 | 0.000182s
Error expanding coefficients:
16328 16008 16008 16360 16008 16520 16360 16256 16008 16480 16520 16136
   16360 16520 16256  | 0.1257s
Error expanding coefficients deviation:
0.34% 2.3% 2.3% 0.15% 2.3% 0.83% 0.15% 0.78% 2.3% 0.59% 0.83% 1.5% 0.15%
   0.83% 0.78%

Coordinate function #2
Dissb: 0 | 0.001215s
Deg: 7 | 0.006144s
NL: 16044 | 7.4e-05s
CIL: 0 | 0.000182s
Error expanding coefficients:
16472 16336 16240 16208 16272 16336 16048 16672 16448 16512 16504 16216
   16152 16384 16376  | 0.1266s
Error expanding coefficients deviation:
0.54% 0.29% 0.88% 1.1% 0.68% 0.29% 2.1% 1.8% 0.39% 0.78% 0.73% 1% 1.4% 0%
   0.049%

Coordinate function #3
Dissb: 0 | 0.001371s
Deg: 7 | 0.006157s
NL: 16044 | 8.3e-05s
CIL: 0 | 0.000224s
Error expanding coefficients:
16488 16416 16304 16456 16432 16392 16352 16344 16216 16288 16528 16312
   16392 16264 16224  | 0.1241s
Error expanding coefficients deviation:
0.63% 0.2% 0.49% 0.44% 0.29% 0.049% 0.2% 0.24% 1% 0.59% 0.88% 0.44% 0.049%
   0.73% 0.98%

Coordinate function #4
Dissb: 0 | 0.001289s
Deg: 7 | 0.006318s
NL: 16044 | 7.3e-05s
CIL: 0 | 0.000197s
```

```
Error expanding coefficients:
16224 16528 16376 16336 16320 16440 16264 16408 16480 16496 16168 16336
   16176 16288 16376  | 0.1247s
Error expanding coefficients deviation:
0.98% 0.88% 0.049% 0.29% 0.39% 0.34% 0.73% 0.15% 0.59% 0.68% 1.3% 0.29% 1.3%
    0.59% 0.049%

Coordinate function #5
Dissb: 0 | 0.001266s
Deg: 7 | 0.006116s
NL: 16044 | 7.3e-05s
CIL: 0 | 0.000182s
Error expanding coefficients:
16360 16424 16168 16288 16296 16312 16616 16304 16448 16512 16352 16096
   16512 16536 16320  | 0.1239s
Error expanding coefficients deviation:
0.15% 0.24% 1.3% 0.59% 0.54% 0.44% 1.4% 0.49% 0.39% 0.78% 0.2% 1.8% 0.78%
   0.93% 0.39%

Coordinate function #6
Dissb: 0 | 0.001278s
Deg: 7 | 0.005925s
NL: 16044 | 7.4e-05s
CIL: 0 | 0.00018s
Error expanding coefficients:
16576 16536 16592 16232 16408 16424 16424 16488 16600 16368 16176 16400
   16224 16328 16336  | 0.1231s
Error expanding coefficients deviation:
1.2% 0.93% 1.3% 0.93% 0.15% 0.24% 0.24% 0.63% 1.3% 0.098% 1.3% 0.098% 0.98%
   0.34% 0.29%

Coordinate function #7
Dissb: 0 | 0.00152s
Deg: 7 | 0.005966s
NL: 16044 | 7.4e-05s
CIL: 0 | 0.000182s
Error expanding coefficients:
16064 16288 16264 16296 16304 16248 16488 16448 16424 16264 16520 16536
   16304 16184 16424  | 0.1243s
Error expanding coefficients deviation:
2% 0.59% 0.73% 0.54% 0.49% 0.83% 0.63% 0.39% 0.24% 0.73% 0.83% 0.93% 0.49%
   1.2% 0.24%

Coordinate function #8
Dissb: 0 | 0.001329s
Deg: 7 | 0.00608s
NL: 16044 | 7.4e-05s
CIL: 0 | 0.000182s
Error expanding coefficients:
16400 16360 16136 16344 16264 16392 16248 16392 16464 16480 16640 16352
   16160 16392 16704  | 0.1234s
Error expanding coefficients deviation:
0.098% 0.15% 1.5% 0.24% 0.73% 0.049% 0.83% 0.049% 0.49% 0.59% 1.6% 0.2% 1.4%
    0.049% 2%

Coordinate function #9
Dissb: 0 | 0.001361s
Deg: 7 | 0.006012s
NL: 16044 | 7.3e-05s
CIL: 0 | 0.000187s
```

```
Error expanding coefficients:
16472 16240 16272 16048 16448 16504 16152 16376 16376 16488 16384 16408
   16392 16432 16216  | 0.123s
Error expanding coefficients deviation:
0.54% 0.88% 0.68% 2.1% 0.39% 0.73% 1.4% 0.049% 0.049% 0.63% 0% 0.15% 0.049%
   0.29% 1%

Coordinate function #10
Dissb: 0 | 0.001298s
Deg: 7 | 0.006029s
NL: 16044 | 9e-05s
CIL: 0 | 0.00019s
Error expanding coefficients:
16224 16376 16320 16264 16480 16168 16176 16376 16472 16576 16344 16480
   16320 16192 16352  | 0.1234s
Error expanding coefficients deviation:
0.98% 0.049% 0.39% 0.73% 0.59% 1.3% 1.3% 0.049% 0.54% 1.2% 0.24% 0.59% 0.39%
    1.2% 0.2%

Coordinate function #11
Dissb: 0 | 0.001529s
Deg: 7 | 0.0062s
NL: 16044 | 7.4e-05s
CIL: 0 | 0.000187s
Error expanding coefficients:
16576 16592 16408 16424 16600 16176 16224 16336 16296 16520 16472 16448
   16488 16432 16304  | 0.1234s
Error expanding coefficients deviation:
1.2% 1.3% 0.15% 0.24% 1.3% 1.3% 0.98% 0.29% 0.54% 0.83% 0.54% 0.39% 0.63%
   0.29% 0.49%

Coordinate function #12
Dissb: 0 | 0.001277s
Deg: 7 | 0.005958s
NL: 16044 | 7.2e-05s
CIL: 0 | 0.00018s
Error expanding coefficients:
16400 16136 16264 16248 16464 16640 16160 16704 16496 16264 16200 16512
   16512 16528 16448  | 0.1244s
Error expanding coefficients deviation:
0.098% 1.5% 0.73% 0.83% 0.49% 1.6% 1.4% 2% 0.68% 0.73% 1.1% 0.78% 0.78%
   0.88% 0.39%

Coordinate function #13
Dissb: 0 | 0.001308s
Deg: 7 | 0.005996s
NL: 16044 | 9.2e-05s
CIL: 0 | 0.000202s
Error expanding coefficients:
16224 16320 16480 16176 16472 16344 16320 16352 16576 16280 16464 16304
   16400 16432 16336  | 0.1233s
Error expanding coefficients deviation:
0.98% 0.39% 0.59% 1.3% 0.54% 0.24% 0.39% 0.2% 1.2% 0.63% 0.49% 0.49% 0.098%
   0.29% 0.29%

Coordinate function #14
Dissb: 0 | 0.001389s
Deg: 7 | 0.006333s
NL: 16044 | 9.3e-05s
CIL: 0 | 0.000203s
```

```
Error expanding coefficients:
16400 16264 16464 16160 16496 16200 16512 16448 16488 16360 16256 16368
   16480 16408 16368  | 0.1235s
Error expanding coefficients deviation:
0.098% 0.73% 0.49% 1.4% 0.68% 1.1% 0.78% 0.39% 0.63% 0.15% 0.78% 0.098%
   0.59% 0.15% 0.098%

Coordinate function #15
Dissb: 0 | 0.001338s
Deg: 7 | 0.006043s
NL: 16044 | 9.2e-05s
CIL: 0 | 0.000202s
Error expanding coefficients:
16400 16464 16496 16512 16488 16256 16480 16368 16344 16608 16544 16120
   16360 16328 16432  | 0.1245s
Error expanding coefficients deviation:
0.098% 0.49% 0.68% 0.78% 0.63% 0.78% 0.59% 0.098% 0.24% 1.4% 0.98% 1.6%
   0.15% 0.34% 0.29%
```

Висновок

В ході розрахункової роботи було досліджено, що найкращими криптографічними параметрами володіє функція $f(x) = x^{16257}$. Оскільки вона має хороші показники нелінійності(вона практично досягає максимального значення на кожній координатній функції) та володіє кореляційним імунітетом 0-го рівня за кожною координатною функцією, тобто є збалансованою за кожною координатною функцією. Разом з тим вона має низький максимум диференціальної ймовірності, всього $1/16384$, тобто є порівняно стійкою до диференціального криптоаналізу. Хоча лавинні ефекти відсутні, відхилення коефіцієнтів розповсюдження помилок від середнього значення не є дуже великими, тобто дана функція володіє "майже"лавинним ефектом. Друга ж функція $g(x) = x^{16256}$ також є непоганою, але все ж гіршою за першу.

Найбільш трудомісткою процедурою виявилась процедура знаходження максимальної диференціальної імовірності, близько 560 секунд(10хв.), інші ж параметри обчислюються відносно швидко: в сумі для двох функцій всі параметри крім MDP займають менше хвилини.

Отже, на мою думку, для криптографічних цілей функція $f(x) = x^{16257}$ підходить добре.