Федеральное государственное автономное образовательное учреждение
высшего профессионального образования Национальный
исследовательский университет «Высшая школа экономики»

Московский институт электроники и математики Факультет прикладной
математики и кибернетики

Кафедра «Компьютерная безопасность»

ОТЧЕТ

по дисциплине «Теоретико-числовые методы в криптографии»

Программная реализация метода простоты Фробениуса

Выполнили студенты
группы СКБ-181:
Конищев М.А.
Лапушкин С.М.

Проверил: Доцент
Нестеренко А. Ю.

МОСКВА
2022

Введение

Тема данной работы - реализация на языке C++ алгоритма надежности метода Фробениуса проверки чисел на простоту. Данный метод реализует проверку простоты чисел основываясь на исследованиях использования последовательностей Лукаса (“Lucas sequences”).

Согласно исследованиям, проведенным в 2003 году [1], существует реализация теста Фробениуса которая всего в два раза больше трудоёмкости метода Ферма или Миллера-Рабина, (то есть равна трудоёмкости двух таких проверок), но при этом имеет гораздо меньшую вероятность ошибки, а именно 256/331776*t для t итераций теста в худшем случае. При оценке среднего случая в работе указана граница вероятности ошибки этого алгоритма в среднем случае по сравнению с более ранними аналогичными результатами для теста Миллера-Рабина, Результаты показывают, что тест в среднем случае равноценен 9 тестам Миллера-Рабина, но при этом требует время, эквивалентное примерно 2 таким тестам.

Всё это вместе взятое привело к тому, что метод Фробениуса оказался сильно недооценён. В данной работе мы реализовали одну из вариаций данного метода построенную на использовании последовательностей Лукаса (“Lucas sequences”) [2].

Теоретическая часть
Последовательности Люкаса и Псевдопростые Фробениуса

Рассмотрим полином

$$f(x) = x^2 - ax + b \in \mathbb{Z}[x]$$

Для данного полинома последовательности Люкаса будут представлять следующие выражения:

$$\begin{aligned} U_j := U_j(a,b) &:= \frac{x^j - (a-x)^j}{x - (a-x)} \pmod{f(x)} \\ V_j := V_j(a,b) &:= x^j + (a-x)^j \pmod{f(x)} \end{aligned}$$

Данные последовательности удовлетворяют следующему рекурентному соотношению:

$$U_j = aU_{j-1} - bU_{j-2}\ ;\ V_j = aV_{j-1} - bV_{j-2} \text{ for } j \geq 2$$

с начальными значениями равными:

$$U_0 = 0,\ U_1 = 1 \quad V_0 = 2,\ V_1 = a$$

Теорема 1 (основная)

Возьмем такие числа a,b что

$$a, b \in \mathbb{Z} \setminus \{0\},\ \Delta := a^2 - 4b$$

и вышеописанные последовательности

$$(U_j), (V_j)$$

Тогда, если число p простое и

$$\gcd(p, 2ab\Delta) = 1,$$

то

$$U_{p - \left(\frac{\Delta}{p}\right)} \equiv 0 \pmod{p}$$

Определение 2 (псевдопростое число Лукаса)

Возьмем такие числа a,b что

$$a, b \in \mathbb{Z} \setminus \{0\},\ \Delta := a^2 - 4b$$

не являются квадратами.

Тогда псевдопростым числом Лукаса над полиномом

$$f(x) := x^2 - ax + b$$

будем называть составное число n при условии, что

$$\gcd(2ab\Delta, n) = 1$$

и

$$\text{if } U_{n - \left(\frac{\Delta}{n}\right)} \equiv 0 \pmod{n}$$

Определение 3 (псведопростое число Фробениуса)

Возьмем такие числа a,b что

$$a, b \in \mathbb{Z} \setminus \{0\},\ \Delta := a^2 - 4b$$

не являются квадратами.

Тогда псевдопростым числом Фробениуса над полиномом

$$f(x) := x^2 - ax + b$$

будем называть составное число n при условии, что

$$\gcd(2ab\Delta, n) = 1$$

и

$$x^n \equiv \begin{cases} a - x \pmod{f(x), n} & , \text{if } \left(\frac{\Delta}{n}\right) = -1 \\ x \pmod{f(x), n} & , \text{if } \left(\frac{\Delta}{n}\right) = 1 \end{cases}$$

Эффективная реализация

В зарубежных публикациях было найдено алгоритмическое описание реализации данного алгоритма [2] и [3]:

We get the following algorithm:

Input : An odd integer $n \in \mathbb{N}_{\geq 3}$

Output: A boolean value which indicates that n is Frobenius-pseudoprime with respect to the polynomial $x^2 - ax + b$

choose $a, b \in \{1, \ldots, n-1\}$ uniformly at random,
such that $\Delta = a^2 - 4b$ not a square and $\gcd(2\Delta ab, n) = 1$;
$W_1 \leftarrow a^2 b^{-1} - 2 \pmod{n}$;
$m \leftarrow \frac{1}{2}(n - (\frac{\Delta}{n}))$
compute W_m, W_{m+1} using equations (9) and (10)
if $(W_1 W_m \neq 2W_{m+1} \pmod{n})$ **then**
  return *false*;
end
$B \leftarrow b^{(n-1)/2} \pmod{n}$;
if $(BW_m \equiv 2 \pmod{n})$ **then**
  return *true*;
else
  return *false*;
end

Algorithm 1: Frobenius-Test

Допустим, мы хотим применить тест Фробениуса для некого числа n. Возьмем такие числа a,b что

$$a, b \in \mathbb{Z} \setminus \{0\},\ \Delta := a^2 - 4b$$

не являются квадратами и

$$\gcd(2ab\Delta, n) = 1$$

Исходя из алгоритма Евклида и символа Якоби, число

$$n - \left(\tfrac{\Delta}{n}\right)$$

всегда четное, то есть представимо в виде

$$n - \left(\tfrac{\Delta}{n}\right) = 2m$$

Воспользуемся последовательностями Лукаса (модифицированные Вильясом [3]):

$$W_j := b^{-j} V_{2j} \pmod{n}$$

$$W_0 \equiv 2 \pmod{n} \quad \text{and} \quad W_1 \equiv a^2 b^{-1} - 2 \pmod{n}$$

$$\begin{cases} W_{2j} \equiv W_j^2 - 2 \pmod{n} \\ W_{2j+1} \equiv W_j W_{j+1} - W_1 \pmod{n} \end{cases}$$

$$W_m \equiv 2b^{-(n-1)/2} \pmod{n}$$

Пусть

$$B := b^{(n-1)/2},$$

Тогда

$$BW_m \equiv 2 \pmod{n}$$

Практическая часть

Для реализации алгоритма был выбран язык C++ (C++17), вместе с библиотекой Boost 1.79.0 [4] для реализации поддержки размерностей переменных вплоть до int1024.
Исходный код приведен в приложении к документу.

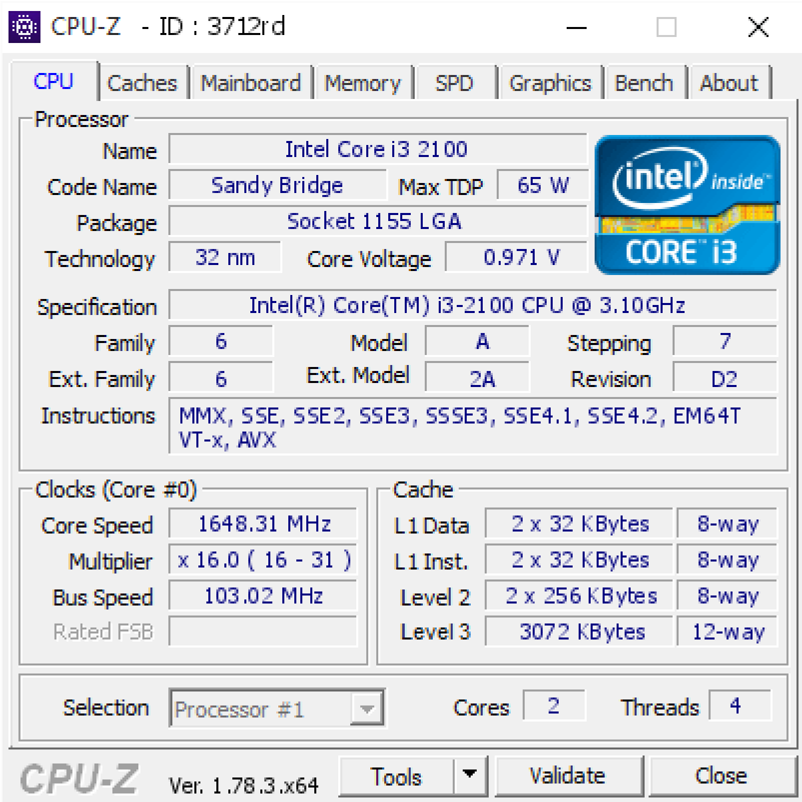
Для проверки корректности алгоритма из открытых источников были взяты таблицы простых чисел [5].

Для наглядности в файле отчета приведены частичные тесты и измерения.

```
Windows PowerShell
2222222222222271
Input a number: Input iterations: Number is not prime
PS C:\Users\AndreyKrylov\Desktop\Frobenius> .\tests.ps1
1299583
Input a number: Input iterations: Number is Prime
1299601
Input a number: Input iterations: Number is Prime
1299631
Input a number: Input iterations: Number is Prime
1299637
Input a number: Input iterations: Number is Prime
1299647
Input a number: Input iterations: Number is Prime
1299653
Input a number: Input iterations: Number is Prime
1299673
Input a number: Input iterations: Number is Prime
1299689
Input a number: Input iterations: Number is Prime
1299709
Input a number: Input iterations: Number is Prime
8191
Input a number: Input iterations: Number is Prime
131071
Input a number: Input iterations: Number is Prime
524287
Input a number: Input iterations: Number is Prime
6700417
Input a number: Input iterations: Number is Prime
2147483647
Input a number: Input iterations: Number is not prime
67280421310721
Input a number: Input iterations: Number is Prime
20988936657440586486151264256610222593863921
Input a number: Input iterations: Number is Prime
4
Input a number: Input iterations: Number is not prime
324
Input a number: Input iterations: Number is not prime
183
Input a number: Input iterations: Number is not prime
2222222222222271
Input a number: Input iterations: Number is not prime
PS C:\Users\AndreyKrylov\Desktop\Frobenius>
```

Практическое время работы алгоритма

Тестирование проводилось с помощью виртуальной машины в среде KVM с операционной системой Windows 10 19044, выделенными 4 гб оперативной памяти и 2 ядрами хостовой машины Intel Core i3 2100



Тестируемое значение	Среднее время одной попытки, мс
11	225
13	231
121	0
8191	869
131071	8729
524287	33164
6700417	557389
1299583	86135
1299601	88668
1299631	87713
1299637	84815
1299647	103072

1299653	80511
1299673	77258
1299689	92408
1299709	92301
2147483647	226
6,72804E+13	261
2,09889E+43	296
4	0
324	0
183	247
2,22222E+15	270

Ссылки и литература

1. https://www.brics.dk/RS/03/9/BRICS-RS-03-9.pdf
2. https://eprint.iacr.org/2008/124.pdf
3. http://thales.doa.fmph.uniba.sk/macaj/skola/teoriapoli/primes.pdf
4. https://www.boost.org/
5. https://en.wikipedia.org/wiki/Largest_known_prime_number

Приложение

Листинг файла Frobenius.h:

```
#pragma once
#include <boost/multiprecision/cpp_int.hpp>

namespace mp = boost::multiprecision;
/**
* Основная функция, проверка числа на простоту, реализуя тест Фробениуса
* \param[in] number Число для проверки
* \param[in] iterations Количество итераций проверки
* \return true, если число простое, false - иначе
*/
bool CheckIsPrime(mp::uint1024_t number, int iterations);

/**
* Функция, реализующая алгоритм Фробениуса для проверки числа на простоту
* \param[in] number Число для проверки
* \return true, если число простое, false - иначе
*/
bool FrobeniusAlgorithm(mp::uint1024_t number);

/**
* Функция, генерирующая 2 случайных числа, удовлетворяющих условиям теста Люка
* \param[in] number Число для проверки на простоту
* \param[out] a Первое сгенерированное число
* \param[out] b Второе сгенерированное число
* \param[out] delta = a ^ 2 - 4 * b
*/
```

```
void ABSelect(mp::uint1024_t number, mp::uint1024_t& a, mp::uint1024_t& b, mp::uint1024_t& delta);

/**
* Функция, генерирующая рекуррентную последовательность последовательность Wj
* \param[in] w1 Первый элемент последовательности
* \param[in] m Индекс последовательности
* \paran[in] number Число для проверки на простоту (по нему будет браться модуль)
* \param[out] Wm m-й элемент последовательности Wj
* \param[out] Wm1 (m+1)-й элемент последовательности Wj
*/
void CalculateSequence(mp::uint1024_t w1, mp::uint1024_t m, mp::uint1024_t number, mp::uint1024_t& Wm, mp::uint1024_t& Wm1);

/**
* Функция, проверяющая, является ли число квадратом
* \param[in] Проверяемое число
* \return true, если число является квадратом, false - иначе
*/
bool CheckIsSquare(mp::uint1024_t number);

/**
* Функция для возведения в степень числа по модулю
* \param[in] Число, возводимое в степень
* \param[in] Степень, в которую необходимо возвести число
* \param[in] Модуль, по которому будет производиться возведение
* \return Результат возведения в степень по модулю
*/
mp::uint1024_t ModulePow(mp::uint1024_t number, mp::uint1024_t degree, mp::uint1024_t module);

/**
* Функция для вычисления символа Якоби
```

```
* \param[in] number Нижний аргумент
* \param[in] delta Верхний аргумент
* \return Значение символа Якоби
*/
int CountJacobi(mp::uint1024_t number, mp::uint1024_t delta);

/**
* Расширенный алгоритм Евклида
* \param[in] a Первое число, для которого считается НОД
* \param[in] b Второе число, для которого считается НОД
* \param[out] x Множитель при числе a (ax+by=d)
* \param[out] y Множитель при числе b (ax+by=d)
* \return НОД чисел a и b
*/
mp::uint1024_t ExtendedGCD(mp::uint1024_t a, mp::uint1024_t b, mp::int1024_t& x, mp::int1024_t& y);
```

Листинг файла Frobenius.cpp

```
#include <stdint.h>
#include <cmath>
#include <random>
#include <numeric>
#include <iostream>
#include <chrono>
#include <fstream>

#include <boost/multiprecision/cpp_int.hpp>
#include <boost/random/random_device.hpp>
#include <boost/random.hpp>
#include <boost/math/common_factor_rt.hpp>
#include <boost/integer/extended_euclidean.hpp>
```

```cpp
#include <boost/multiprecision/cpp_bin_float.hpp>

#include "Frobenius.h"



bool CheckIsSquare(mp::uint1024_t number)
{
	if (number == 0 || number == 1)
	{
	return true;
	}

	mp::uint1024_t root = mp::sqrt(number);

	if (root * root == number)
	{
	return true;
	}
	return false;
}

void ABSelect(mp::uint1024_t number, mp::uint1024_t& a, mp::uint1024_t& b, mp::uint1024_t& delta)
{
	boost::random::random_device prng{};
	// распределение в промежутке [1, a]
	boost::random::uniform_int_distribution<mp::uint1024_t> distA(1, a - 1);
	// распределение в промежутке [1, b]
	boost::random::uniform_int_distribution<mp::uint1024_t> distB(1, b - 1);
```

```
    // генерация двух рандомных чисел
    a = distA(prng);
    b = distB(prng);
    // подсчет дельты
    delta = a * a - 4 * b;

    // должно удовлетворять 2 условиям:
    // 1) delta - не является полным квадратом
    // 2) НОД(number, 2ab*delta) = 1
    while (!(!CheckIsSquare(delta) && boost::math::gcd(2 * delta * a * b, number) == 1))
    {
    a = distA(prng);
    b = distB(prng);
    delta = a * a - 4 * b;
    }
}

bool CheckIsPrime(mp::uint1024_t number, int iterations)
{
    std::vector<long long> time;
    bool flag = false;
    for (int i = 0; i < iterations; ++i)
    {
    auto begin = std::chrono::high_resolution_clock::now();
    bool rc = FrobeniusAlgorithm(number);
    auto end = std::chrono::high_resolution_clock::now();
    auto duration = std::chrono::duration_cast<std::chrono::microseconds>(end - begin).count();
    time.push_back(duration);
    if (rc)
    {
```

```
				flag = true;
			}
		}
		if (flag)
		{
		std::cout << "Number is prime\n";
		}
		else
		{
		std::cout << "Number is not prime\n";
		}
		long long resTime = 0;
		for (int i = 0; i < time.size(); ++i)
		{
		resTime += time[i];
		}
		resTime /= time.size();
		std::ofstream fout("out.dat", std::ios_base::app);
		fout << number << ' ' << resTime << '\n';
		return flag;
}

mp::uint1024_t ExtendedGCD(mp::uint1024_t a, mp::uint1024_t b, mp::int1024_t& x, mp::int1024_t& y)
{
		if (a == 0) {
		x = 0; y = 1;
		return b;
		}
		mp::int1024_t x1 = 0, y1 = 0;
		mp::uint1024_t d = ExtendedGCD(b % a, a, x1, y1);
```

```
		x = y1 - (mp::int1024_t)(b / a) * x1;
		y = x1;
		return d;
}

mp::uint1024_t ModulePow(mp::uint1024_t number, mp::uint1024_t degree, mp::uint1024_t module)
{
		mp::uint1024_t res = 1;

		if ((degree >> 24) == 0)
		{
		for (int i = 0; i < degree; ++i)
		{
				res *= (number % module);
				res %= module;
		}
		}
		else
		{
		while (degree != 0)
		{
				if ((degree & 1) != 0)
				{
				res *= number;
				res %= module;
				}

				number *= number;
				number %= module;
				degree >>= 1;
```

```
        }
        }
        return res;
}

void CalculateSequence(mp::uint1024_t w1, mp::uint1024_t m, mp::uint1024_t number, mp::uint1024_t& Wm, mp::uint1024_t& Wm1)
{
        Wm = 2;
        Wm1 = w1;

        int log = 0;
        while (m)
        {
        log++;
        m >>= 1;
        }
        log -= 2;
        if (log < 0)
        {
        log = 0;
        }
        mp::uint1024_t mask = 1 << log;
        while (mask <= m)
        {
        mask <<= 1;
        }
        mask >>= 1;
        while (mask)
        {
        if (mask & m != 0)
```

```
    {
        Wm = (Wm * Wm1 - w1) % number;
        Wm1 = (Wm1 * Wm1 - 2) % number;
    }
    else
    {
        Wm = (Wm * Wm - 2) % number;
        Wm1 = (Wm * Wm1 - w1) % number;
    }
    mask >>= 1;
    }
}

int CountJacobi(mp::uint1024_t number, mp::uint1024_t delta)
{
    int res = 1;
    while (number)
    {
    // number % 2 == 0
    while (!(number & 1))
    {
        number >>= 1;
        // delta % 8 == 3 || d % 8 == 5
        if ((delta & 7) == 3 || (delta & 7) == 5)
        {
        res = -res;
        }
    }
    std::swap(number, delta);
    // delta % 4 == 3
```

```
        if ((delta & 3) == 3 && (delta & 3) == (number & 3))
        {
                res = -res;
        }
        number %= delta;
        }
        return delta == 1 ? res : 0;
}


bool FrobeniusAlgorithm(mp::uint1024_t number)
{
        // первичная проверка чисел
        if (number < 3 || number % 2 == 0)
        {
        return false;
        }

        if (CheckIsSquare(number))
        {
        return false;
        }


        // выбор чисел A и B
        mp::uint1024_t a = 0, b = 0, delta = 0;
        ABSelect(number, a, b, delta);
        // Алгоритм Евклида и поиск коэффициентов
        mp::int1024_t x = 0, y = 0;
        mp::uint1024_t gcd = ExtendedGCD(a, b, x, y);
```

```
    // первый элемент последовательности Wj
    mp::int1024_t w1 = ((mp::int1024_t)(a % number) * (mp::int1024_t)(a % number) * (x % number)) % number - 2;
    if (w1 < 0)
    w1 += number;
    // подсчет символа Якоби для (delta/number)
    mp::uint1024_t m = (number - CountJacobi(delta, number)) >> 1;

    mp::uint1024_t wm = 0, wm1 = 0;
    // подсчет m и (m+1) символа последовательности
    CalculateSequence((mp::uint1024_t)w1, m, number, wm, wm1);


    if ((w1 * wm) != (2 * wm1 % number))
    {
    return false;
    }
    // возведение b в степень (number-1)/2
    b = ModulePow(b, (number - 1) >> 1, number);
    return b * wm % number == 2;
}
```

Листинг файла main.cpp:

```
#include <stdint.h>
#include <cmath>
```

```cpp
#include "Frobenius.h"
#include <boost/multiprecision/cpp_int.hpp>

int main()
{
	mp::uint1024_t input;
	int iterations;
	std::cout << "Input a number: ";
	std::cin >> input;
	std::cout << "Input iterations: ";
	std::cin >> iterations;

	bool rc = CheckIsPrime(input, iterations);

	return 0;
}
```