МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г. ЧЕРНЫШЕВСКОГО»

Кафедра теоретических основ компьютерной безопасности и криптографии

Схема Миньотта (китайская теорема об остатках)

ОТЧЁТ
ПО ДИСЦИПЛИНЕ
«КРИПТОГРАФИЧЕСКИЕ ПРОТОКОЛЫ»

студента 5 курса 531 группы
специальности 10.05.01 Компьютерная безопасность
факультета компьютерных наук и информационных технологий
Алексеева Александра Александровича

Преподаватель
профессор, д.ф.-м.н. ___________ В. Е. Новиков
подпись, дата

Саратов 2023

СОДЕРЖАНИЕ

1 Теоретическая часть

Цель работы – изучение схемы разделения секрета Миньотта.

1.1 Описание алгоритма

Последовательность Миньотта

Пороговая схема разделения секрета Миньотта использует специальные последовательности чисел, названные последовательностями Миньотта. Пусть n – целое, $n \geq 2$, и $2 \leq k \leq n$. (k, n)-последовательность Миньотта – последовательность взаимно простых положительных $p_1 < p_2 < \ldots < p_n$ таких, что $\prod_{i=0}^{k-2} p_{n-1} < \prod_{i=1}^{k} p_i$.

Алгоритм

Имея открытый ключ-последовательность Миньотта, схема работает так:

1. Секрет S выбирается, как случайное число такое, что $\beta < S < \alpha$, где $\alpha = \prod_{i=1}^{k} p_i$, $\beta = \prod_{i=0}^{k-2} p_{n-1}$. Другими словами, секрет должен находится в промежутке между $p_1 \bullet p_2 \bullet \ldots \bullet p_k$ и $p_{n-k+2} \bullet \ldots \bullet p_n$.
2. Доли вычисляются как $I_i = S \bmod p_i$ для всех $1 \leq i \leq n$.
3. Имея k различных теней $I_{i_1}, \ldots, I_{i_k}$, можно получить секрет S, используя стандартный вариант китайской теоремы об остатках – им будет единственное решение по модулю $p_{i_1}, \ldots, p_{i_k}$ системы.

$$\begin{cases} x \equiv I_{i_1} \; mod \; p_{i_1} \\ \quad\quad \ldots \\ x \equiv I_{i_k} \; mod \; p_{i_k} \end{cases}$$

Секретом является решение приведённой выше системы, более того, S лежит в пределах $Z_{p_{i_1}}, \ldots, Z_{p_{i_k}}$, т.к. $S < \alpha$. С другой стороны, имея всего $k - 1$ различных теней $I_{i_1}, \ldots, I_{i_{k-1}}$, можно сказать, что $S \equiv x_0 \bmod p_{i_1}, \ldots, p_{i_{k-1}}$, где x_0 – единственное решение по модулю $p_{i_1}, \ldots, p_{i_{k-1}}$ исходной системы (в данном случае $S > \beta \geq p_{i_1}, \ldots, p_{i_{k-1}} > x_0$. Для того, чтобы получить приемлемый уровень безопасности, должны быть использованы (k, n) последовательности Миньотта с большим значением $\frac{\alpha - \beta}{\beta}$. Очевидно, что схема Миньотта не обладает значительной криптографической стойкостью, однако может оказаться удобной в приложениях, где компактность теней является решающим факторов.

2 Описание программы

Программа, представленная ниже, содержит следующие функции:

• powClosed(x, y, mod) – возводит число x в степень y по модулю mod;

• binForm(x) – представление числа x в 2-чной системе счисления;

• decForm(x) – представление числа x в 10-чной системе счисления;

• usualEuclid(a, b) – вычисление НОД чисел a и b обычным алгоритмом Евклида;

• funEuler(n) – вычисление функции Эйлера от n;

• miller_rabin(n, $k = 10$) – проверка числа n на простоту с вероятностью $\frac{1}{2^k}$;

• generatesimpleNum(m) – генерация простого числа размером $\approx m$ бит;

• quicksort(mas) – сортировка массива mas;

• gcTheorem($raws$) – поиск решения системы линейных сравнений $raws$ с помощь греко-китайской теоремы об остатках;

• guillouQuisquater (J, M) – подпись сообщения M с атрибутами J;

• selectPrimeNums(m) – генерация последовательности Миньотта размером m;

• selectS(m) – генерация секрета размерности $\approx m$ бит;

• selectIs() – вычисление теней секрета.

2.1 Примеры работы программы

```
        Схема Миньотта (китайскай теорема об остатках)
Количество людей, между которыми необходимо разделить секрет: 10
Количество людей, которые могут восстановить секрет: 5
Параметр безопасности > 5: 10

Последовательность Миньотта:
p1 = 947
p2 = 4423
p3 = 9767
p4 = 9973
p5 = 13841
p6 = 46681
p7 = 61363
p8 = 92557
p9 = 99719
p10 = 187477

alpha = 5647046888460665711, beta = 1061797658975293469 33
S (beta < S < alpha) = 4532921905242456424

Доли пользователей:
I1: 83
I2: 4383
I3: 403
I4: 2369
I5: 1745
I6: 18891
I7: 22271
I8: 80379
I9: 17553
I10: 142679

Количество пользователей, которые хотят восстановить секрет: 5


По греко-китайской теореме об остатках:
x = 83 (mod 947)
x = 4383 (mod 4423)
x = 403 (mod 9767)
x = 1745 (mod 13841)
x = 18891 (mod 46681)
Решение данной системы линейных сравнений: x = 4532921905242456424 (mod 26432346916698319067)
Изначальное значение S: 4532921905242456424
```

```
        Схема Миньотта (китайскай теорема об остатках)
Количество людей, между которыми необходимо разделить секрет: 10
Количество людей, которые могут восстановить секрет: 5
Параметр безопасности > 5: 10

Последовательность Миньотта:
p1 = 8081
p2 = 9421
p3 = 39679
p4 = 46549
p5 = 49261
p6 = 56041
p7 = 77291
p8 = 86719
p9 = 90437
p10 = 312563

alpha = 692685997174639740133 1, beta = 189464087022463085099
S (beta < S < alpha) = 189464087022756835746

Доли пользователей:
I1: 2856
I2: 660
I3: 9958
I4: 39623
I5: 11304
I6: 41586
I7: 44847
I8: 33394
I9: 11271
I10: 253990

Количество пользователей, которые хотят восстановить секрет: 4


По греко-китайской теореме об остатках:
x = 660 (mod 9421)
x = 39623 (mod 46549)
x = 11304 (mod 49261)
x = 253990 (mod 312563)
Решение данной системы линейных сравнений: x = 401245375476097630 (mod 6752244344545740647)
Изначальное значение S: 189464087022756835746
```

3 Листинг кода

```
#include "iostream"
#include "vector"
#include "string"
#include "boost/multiprecision/cpp_int.hpp"

using namespace std;
using namespace boost::multiprecision;

class Pattern {
private:
	static vector <cpp_int> deg2(cpp_int el, cpp_int n) {//Раскладываем
число на степени двойки
		vector <cpp_int> res;
		while (n != 0) {
			if (n / el == 1) {
				res.push_back(el);
				n -= el;
				el = 1;
			}
			else
				el *= 2;
		}
		return res;
	}


	static cpp_int multMod(cpp_int n, cpp_int mod, vector <pair <cpp_int,
cpp_int>> lst) {//Умножаем число по модулю
		if (lst.size() == 1) {
			cpp_int res = 1;
			for (int i = 0; i < lst[0].second; i++)
				res = res * lst[0].first % mod;
			return res;
		}
		else if (lst[0].second == 1) {
			cpp_int el = lst[0].first;
			lst.erase(lst.begin());
			return (el * multMod(n, mod, lst)) % mod;
		}
		else {
			for (int i = 0; i < lst.size(); i++)
				if (lst[i].second > 1) {
					lst[i].first = (lst[i].first *
lst[i].first) % mod;
					lst[i].second /= 2;
				}
			return multMod(n, mod, lst);
		}
	}


	static int partition(vector <cpp_int>& a, int start, int end) { //Для
quicksort
		cpp_int pivot = a[end];
		int pIndex = start;

		for (int i = start; i < end; i++) {
			if (a[i] <= pivot) {
				swap(a[i], a[pIndex]);
				pIndex++;
			}
```

```
                }

                swap(a[pIndex], a[end]);
                return pIndex;
        }


        static bool checkMutualSimplicity(vector <pair <cpp_int, cpp_int>> raws)
{ //Для gcTheorem
                for (unsigned short i = 0; i < raws.size(); i++)
                        for (unsigned short j = i + 1; j < raws.size(); j++) {
                                if (raws[i].second > raws[j].second) {
                                        if (usualEuclid(raws[i].second,
raws[j].second) != 1)
                                                return false;
                                }
                                else
                                        if (usualEuclid(raws[j].second,
raws[i].second) != 1)
                                                return false;
                        }
                return true;
        }
public:
        static cpp_int powClosed(cpp_int x, cpp_int y, cpp_int mod) {//Возводим
число в степени по модулю
                if (y == 0)
                        return 1;

                vector <cpp_int> lst = deg2(1, y);
                vector <pair <cpp_int, cpp_int>> xDegs;
                for (int i = 0; i < lst.size(); i++)
                        xDegs.push_back(make_pair(x, lst[i]));

                cpp_int res = multMod(x, mod, xDegs);
                return res;
        }


        static string binForm(cpp_int x) {
                string bitter = "";
                while (x != 0) {
                        bitter = (x % 2 == 0 ? "0" : "1") + bitter;
                        x = x / 2;
                }
                if (bitter == "")
                        return "0";
                return bitter;
        }


        static cpp_int decForm(string x) {
                cpp_int res = 0, deg = 1;
                if (x.back() == '1')
                        res += 1;
                for (int i = 1; i < x.length(); i++) {
                        deg = deg * 2;
                        if (x[x.length() - i - 1] == '1')
                                res += deg;
                }
                return res;
        }
```

```
static cpp_int usualEuclid(cpp_int a, cpp_int b) {
	if (a < b)
		swap(a, b);
	if (a < 0 || b < 0)
		throw string{ "Выполнение невозможно: a < 0 или b < 0" };
	else if (b == 0)
		return a;

	cpp_int r = a % b;
	return usualEuclid(b, r);
}


static cpp_int funEuler(cpp_int n) {
	cpp_int res = 1;
	for (int i = 2; i < n; i++)
		if (usualEuclid(n, i) == 1)
			res++;
	return res;
}


static bool miller_rabin(cpp_int n, int k = 10) {
	if (n == 0 || n == 1)
		return false;

	cpp_int d = n - 1;
	cpp_int s = 0;
	while (d % 2 == 0) {
		s++;
		d = d / 2;
	}

	cpp_int nDec = n - 1;
	for (int i = 0; i < k; i++) {
		cpp_int a = rand() % nDec;
		if (a == 0 || a == 1)
			a = a + 2;

		cpp_int x = powClosed(a, d, n);
		if (x == 1 || x == nDec)
			continue;

		bool flag = false;
		for (int j = 0; j < s; j++) {
			x = (x * x) % n;
			if (x == nDec) {
				flag = true;
				break;
			}
		}
		if (!flag)
			return false;
	}

	return true;
}


static cpp_int generateSimpleNum(unsigned short m) {
	cpp_int q = rand() % 1000;
	while (funEuler(q) != q - 1)
```

```
				q++;

			cpp_int s, n = 2, nDec;
			while (!miller_rabin(n)) {
				string sBin = "1";
				int sBinSize = rand() % (m / 2) + m / 2;
				for (int i = 0; i < sBinSize; i++)
					sBin = sBin + to_string(rand() % 2);
				s = decForm(sBin);

				n = (q * s) + 1;
				nDec = n - 1;
			}

			return n;
		}


		static void quicksort(vector <cpp_int>& a, int start, int end) {
			if (start >= end) {
				return;
			}

			int pivot = partition(a, start, end);
			quicksort(a, start, pivot - 1);
			quicksort(a, pivot + 1, end);
		}


		static pair <cpp_int, cpp_int> gcTheorem(vector <pair <cpp_int, 
cpp_int>> raws) {
			if (!checkMutualSimplicity)
				throw string{ "Модули m не являются попарно взаимно 
простыми!" };

			cpp_int M = 1;
			for (unsigned short i = 0; i < raws.size(); i++)
				M *= raws[i].second;

			cpp_int res = 0;
			for (unsigned short i = 0; i < raws.size(); i++) {
				cpp_int Mi = 1;
				for (unsigned short j = 0; j < raws.size(); j++) {
					if (i == j)
						continue;
					Mi *= raws[j].second;
				}
				for (cpp_int j = 1;; j++)
					if ((Mi * j) % raws[i].second == raws[i].first) {
						res += Mi * j;
						break;
					}
			}

			return make_pair(res % M, M);
		}
};


class Mignotte {
private:
		bool checkMutualSimplicity(vector <pair <cpp_int, cpp_int>> raws) {
```

```
		for (unsigned short i = 0; i < raws.size(); i++)
			for (unsigned short j = i + 1; j < raws.size(); j++) {
				if (raws[i].second > raws[j].second) {
					if (Pattern::usualEuclid(raws[i].second, 
raws[j].second) != 1)
						return false;
				}
				else
					if (Pattern::usualEuclid(raws[j].second, 
raws[i].second) != 1)
						return false;
			}
		return true;
	}
public:
	int n, k;
	vector <cpp_int> simpleNums, Is;
	cpp_int alpha = 1, beta = 1, S;


	Mignotte(int n, int k) {
		this->n = n;
		this->k = k;
	}
	~Mignotte() {
	}


	void selectPrimeNums(unsigned short m) {
		for (unsigned short i = 0; i < n; i++) {
			cpp_int p = Pattern::generateSimpleNum(m);
			this->simpleNums.push_back(p);
		}
		Pattern::quicksort(simpleNums, 0, n - 1);

		cout << "\nПоследовательность Миньотта: ";
		for (unsigned short i = 0; i < simpleNums.size(); i++)
			cout << "\np" << i + 1 << " = " << simpleNums[i];
	}


	void selectS(unsigned short m) {
		for (unsigned short i = 0; i < k; i++)
			alpha *= simpleNums[i];
		for (unsigned short i = n - k + 1; i < n; i++)
			beta *= simpleNums[i];
		cout << "\n\nalpha = " << alpha << ", beta = " << beta;

		S = (Pattern::generateSimpleNum(2 * m) + beta) % alpha;
		cout << "\nS (beta < S < alpha) = " << S;
	}


	void selectIs() {
		for (unsigned short i = 0; i < n; i++)
			this->Is.push_back(S % simpleNums[i]);

		cout << "\n\nДоли пользователей:";
		for (unsigned short i = 0; i < n; i++)
			cout << "\nI" << i + 1 << ": " << Is[i];
	}
};
```

```
int main() {
	setlocale(LC_ALL, "ru");
	srand(time(NULL));

	cout << "\tСхема Миньотта (китайскай теорема об остатках)";
	int n, k;
	cout << "\nКоличество людей, между которыми необходимо разделить секрет: 
";
	cin >> n;
	cout << "Количество людей, которые могут восстановить секрет: ";
	cin >> k;
	if (n < 2 || (k < 2 || k > n)) {
		cout << "\nПараметры k или n указаны неверно!";
		return 0;
	}

	cout << "Параметр безопасности > 5: ";
	unsigned short m;
	cin >> m;

	try {
		Mignotte mignotte(n, k);
		mignotte.selectPrimeNums(m);
		mignotte.selectS(m);
		mignotte.selectIs();

		cout << "\n\nКоличество пользователей, которые хотят восстановить 
секрет: ";
		cin >> m;
		if (m < 1 || m > n)
			throw string{ "Неверное количество пользователей!" };

		cout << "\n\nПо греко-китайской теореме об остатках: ";
		vector <pair <cpp_int, cpp_int>> raws;
		for (unsigned short i = 0; i < mignotte.Is.size(); i++)
			raws.push_back(make_pair(mignotte.Is[i], 
mignotte.simpleNums[i]));
		while (raws.size() != m)
			raws.erase(raws.begin() + rand() % raws.size());
		for (unsigned short i = 0; i < raws.size(); i++)
			cout << "\nx = " << raws[i].first << " (mod " << 
raws[i].second << ")";

		pair <cpp_int, cpp_int> res = Pattern::gcTheorem(raws);
		cout << "\nРешение данной системы линейных сравнений: x = " << 
res.first << " (mod " << res.second << ")";
		cout << "\nИзначальное значение S: " << mignotte.S << endl;
	}
	catch (string& message) {
		cout << message;
	}

	cout << endl;
	return 0;
}
```