Міністерство освіти та науки

Прикарпатський національний університет

Фізико-технічний факультет

Кафедра комп’ютерної інженерії та електроніки

Лабораторна робота № 8

З курсу “Захист інформації у комп’ютерних мережах та системах”

Виконав студент групи КІ-41

Воробій Віталій

Івано-Франківськ 2020

5 варіант

Програмна реалізація мовою *Java:*

Реалізував клас, що містить ряд ефективних алгоритмів, що будуть корисні у подальшому.

package solution;  
  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Random;  
  
public class Algorithms {  
  
 */\*\*  
 \* Дискретне піднесення до степеня по модулю  
 \*/* public static int discretePower(int x, int y, int p) {  
 int res = 1;  
 x = x % p;  
 if (x == 0) return 0;  
 while (y > 0)  
 {  
 if ( (y & 1) == 1)  
 res = (res \* x) % p;  
 y = y >> 1;  
 x = (x \* x) % p;  
 }  
 return res;  
 }  
  
 */\*\*  
 \* Знаходження випадкового числа між двома межами  
 \* @param minQuote Мінімальне співвідношення між згенерованим  
 \* випадковим числом і правою межою  
 \*/* public static int findRandomInRangeWithMinQuote (  
 int a,  
 int b,  
 float minQuote)  
 {  
 final Random r = new Random();  
 int leftBound = (int)( a + ((b - a) \* minQuote));  
 return leftBound + Math.*abs*(r.nextInt(b - leftBound));  
 }  
  
 */\*\*  
 \* Рекурсивний алгоритм Евкліда  
 \*/* public static int euclidGCD(int a, int b) {  
 if (b == 0) {  
 return a;  
 }  
 return *euclidGCD*(b, a % b);  
 }

*/\*\*  
 \* Знаходження взаємнопростого випадкового числа  
 \* в межах від 1 до p - 1  
 \*/* public static int findRandomCoprimeNumberBetween(int p) {  
 final Random r = new Random();  
 int a = 2 + Math.*abs*(r.nextInt(p - 3));  
 while (*euclidGCD*(p - 1, a ) != 1) {  
 a = 2 + Math.*abs*(r.nextInt(p - 3));  
 }  
 return a;  
 }  
  
 */\*\*  
 \* Функція для генерування простих чисел  
 \*/* public static List<Integer> generatePrimeNumbers(int length) {  
 List<Integer> numbers = new ArrayList<>();  
 for (int n = 2; numbers.size() != length; n++) {  
 boolean isPrime = true;  
 for (Integer r: numbers) {  
 if (n % r == 0) {  
 isPrime = false;  
 break;  
 }  
 }  
 if (isPrime) {  
 numbers.add(n);  
 }  
 }  
 return numbers;  
 }  
  
}

Написав клас конфігурації Ель Гамаля.

package solution;  
  
*/\*\*  
 \* Клас, що описує конфігурацію Ель Гамаля  
 \*/*public class ElHamal {  
 private static final float *MIN\_QUOTE* = 0.6f;  
 private final int p;  
 private final int q;  
 private final int x;  
 private final int y;  
  
 public ElHamal(int p, int q) {  
 this.p = p;  
 this.q = q;  
 this.x = calcX();  
 this.y = calcY();  
 }  
  
 */\*\*  
 \* Обрахунок значення x  
 \*/* private int calcX() {  
 return Algorithms.*findRandomInRangeWithMinQuote*(p, q, *MIN\_QUOTE*);  
 }

*/\*\*  
 \* Обрахунок значення y  
 \*/* private int calcY() {  
 return Algorithms.*discretePower*(q, x, p);  
 }  
  
 public int getP() {  
 return p;  
 }  
  
 public int getQ() {  
 return q;  
 }  
  
 public int getY() {  
 return y;  
 }  
  
 public int getX() {  
 return x;  
 }  
}

Написав клас для знаходження хеш-значення повідомлення:

package solution;  
  
import java.util.List;  
import java.util.Random;  
  
public class ElHamalHasher {  
 private final int H\_0;  
 private final int p;  
 private final int n;  
  
 public ElHamalHasher(ElHamal elHamal, int h\_0) {  
 this.n = elHamal.getP() \* elHamal.getQ();  
 this.p = elHamal.getP();  
 H\_0 = h\_0;  
 }  
  
 public ElHamalHasher(ElHamal elHamal) {  
 this.n = elHamal.getP() \* elHamal.getQ();  
 this.p = elHamal.getP();  
 H\_0 = new Random().nextInt();  
 }  
  
 public int hash(List<Integer> M) {  
 int prevH = H\_0;  
 for (Integer num : M) {  
 prevH = Algorithms.*discretePower*(  
 num + prevH,  
 2,  
 n  
 );  
 }  
 return 2 + Math.*abs*(prevH % (p - 2));  
 }  
  
}

Додаткова операція взяття по модулю в кінці функції хешування необхідна щоб вихідне число було у межах від *1* до *p – 1*.

Написав клас, що описує підпис Ель-Гамаля:

package solution;  
  
public class ElHamalKey {  
 private final int a;  
 private final int b;  
  
 public ElHamalKey(ElHamal elHamal, int m) {  
 final int p = elHamal.getP();  
 final int q = elHamal.getQ();  
 final int x = elHamal.getX();  
 final int k = Algorithms.*findRandomCoprimeNumberBetween*(p);  
  
 a = calcA(q, k, p);  
 b = calcB(x, a, k, p, m);  
 }  
  
 */\*\*  
 \* Розрахунок значення а  
 \*/* private int calcA(int q, int k, int p) {  
 return Algorithms.*discretePower*(q, k, p);  
 }  
  
 */\*\*  
 \* Розрахунок значення b  
 \*/* private int calcB(int x, int a, int k, int p, int m) {  
 final int r = m % (p - 1);  
 for (int i = 1; ; i++) {  
 int o = (a \* x + k \* i) % (p - 1);  
 if (o == r) {  
 return i;  
 }  
 }  
 }  
  
 public int getA() {  
 return a;  
 }  
  
 public int getB() {  
 return b;  
 }  
}

Таким чином пошук коефіцієнту *b* зводиться до ітеративного підбору значення починаючи з одиниці, що задовольняє рівності вище.

Клас для перевірки достовірності ключа:

package solution;  
import static solution.Algorithms.*discretePower*;  
  
public class ElHamalKeyValidator {  
 private final ElHamal elHamal;  
  
 public ElHamalKeyValidator(ElHamal elHamal) {  
 this.elHamal = elHamal;  
 }  
  
 */\*\*  
 \* Перевірка на достовірність ключа  
 \*/* public boolean validate(ElHamalKey key, int m) {  
 final int a = key.getA();  
 final int b = key.getB();  
 final int y = elHamal.getY();  
 final int p = elHamal.getP();  
 final int q = elHamal.getQ();  
  
 int l = (*discretePower*(y, a, p) \* *discretePower*(a, b, p)) % p;  
 int r = *discretePower*(q, m, p);  
  
 return l == r;  
 }  
  
}

Для перевірки ключа використовується наступна рівність:

Ліву частину дуже важко безпосередньо порахувати. Особливо користуючись високорівневою мовою програмування.

Тому було використано наступне перетворення:

В такому випадку можна застосувати два рази алгоритм дискретного піднесення до степеня по модулю.

Клас для демонстрації роботи програми:

package solution;  
  
import java.util.ArrayList;  
import java.util.List;  
import java.util.Random;  
import java.util.Scanner;  
import java.util.stream.Collectors;  
  
public class ElHamalDemo {  
 private static final int *NUMBER\_OF\_PRIMES* = 5000;

*/\*\*  
 \* Функція для вибору двох випадкових чисел зі списку  
 \*/* private static List<Integer> pickRandomTwo(List<Integer> inputList) {  
 Random random = new Random();  
 final int N = inputList.size();  
 int a = Math.*abs*(random.nextInt()) % N;  
 int b = Math.*abs*(random.nextInt()) % N;  
 while (b == a) {  
 b = Math.*abs*(random.nextInt()) % N;  
 }  
 return List.*of*(inputList.get(a), inputList.get(b));  
 }  
  
 private static List<Integer> stringToIntList(String text) {  
 final int N = text.length();  
 List<Integer> integers = new ArrayList<>();  
 for (int i = 0; i < N; i++) {  
 integers.add((int)text.charAt(i));  
 }  
 return integers;  
 }  
  
 public static void main(String[] args) {  
 List<Integer> primes = Algorithms.*generatePrimeNumbers*(*NUMBER\_OF\_PRIMES*);  
  
 *// Фільтруємо прості числа менші 10 000* primes = primes  
 .stream()  
 .filter(prime -> prime > 10\_000)  
 .collect(Collectors.*toList*());  
  
 *// Залишаємо два випадкові* primes = *pickRandomTwo*(primes);  
  
 final int p = Math.*max*(primes.get(0), primes.get(1));  
 final int q = Math.*min*(primes.get(0), primes.get(1));  
  
 System.*out*.println("p = " + p);  
 System.*out*.println("q = " + q);  
  
 final ElHamal elHamal = new ElHamal(p, q);  
  
 System.*out*.println("x = " + elHamal.getX());  
 System.*out*.println("y = " + elHamal.getY());  
  
 final ElHamalHasher hasher = new ElHamalHasher(elHamal);  
 final ElHamalKeyValidator validator = new ElHamalKeyValidator(elHamal);  
  
 final Scanner scanner = new Scanner(System.*in*);  
  
 System.*out*.println("Введіть текст: ");  
 while (scanner.hasNext()) {  
 final String inputText = scanner.nextLine();  
 if (inputText.equals("stop")) {  
 break;  
 }  
 List<Integer> inputList = *stringToIntList*(inputText);  
  
 final int hash = hasher.hash(inputList);  
 final ElHamalKey key = new ElHamalKey(elHamal, hash);  
  
 System.*out*.println("Вхідний текст: " + inputText);  
 System.*out*.println("Хеш-код: " + hash);

System.*out*.println("a: " + key.getA());  
System.*out*.println("b: " + key.getB());

System.*out*.println("Ключ достовірний : " + (validator.validate(key, hash) ? "Так" : "Ні"));  
 }  
 System.*out*.println("Програму завершено");  
  
 }  
  
}

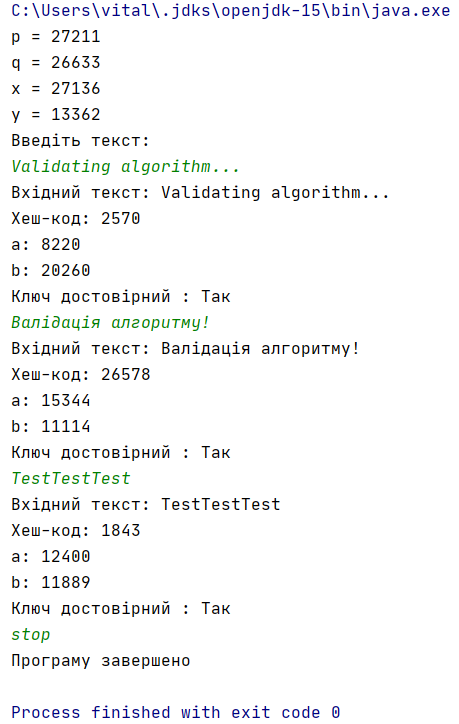


Рисунок 1. Демонстрація роботи програми

Висновок: на цій лабораторній роботі я навчися формувати цифровий підпис Ель Гамаля і перевіряти його на достовірність.