Міністерство освіти та науки

Прикарпатський національний університет імені Василя Стефаника

Фізико-технічний факультет

Кафедра комп’ютерної інженерії та електроніки

Лабораторна робота №2

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

Виконав студент групи КІ-41

Воробій Віталій

Івано-Франківськ 2020

5 варіант

|  |  |  |
| --- | --- | --- |
| № варіанту | Вид генератора ПВЧ | Кількість розрядів |
| 5 | Лінійні конгруентні датчики ПВЧ | 7 |

Виконання роботи

Завдання виконано з використанням мови програмування *Java* і додатково для автоматизованого тестування була використана бібліотека *jUnit*.

1. Розрахунок параметра *M* відповідно до варіанту
2. Визначення параметрів *A*, *C* і *T(0)*

Параметри повинні задовольняти наступним умовам:

2. *C* – не парне
3. *A* mod 4 = 1

Відповідно до цього і

Значення *T(0)* може бути довільним, тому

1. Створив абстракцію генератора послідовностей

package solution;  
  
public interface SequenceGenerator<T> {  
 T[] generateSequence(int length);  
}

1. Створив клас, що реалізує алгоритм додавання за модулем 2 між вхідним текстом і маскою

package solution.helpers;  
  
public final class Algorithms {  
  
 */\*\*  
 \* @throws IndexOutOfBoundsException коли довжина маски менше довжини тексту  
 \*/* public static String xorTextWithMask(String text, Integer [] mask) {  
 final int n = text.length();  
 final StringBuilder builder = new StringBuilder();  
 for (int i = 0; i < n; i++) {  
 builder.append((char) ( mask[i] ^ ((int) text.charAt(i)) ));  
 }  
 return builder.toString();  
 }  
  
}

1. Розробив реалізацію лінійного конгруентного датчику ПВЧ

package solution;  
  
public class LinearCongruentSequenceGenerator implements SequenceGenerator<Integer> {  
 private final int A;  
 private final int C;  
 private final int M;  
 private final int T\_0;  
  
 public LinearCongruentSequenceGenerator(int a, int c, int m, int t\_0) {  
 A = a;  
 C = c;  
 M = m;  
 T\_0 = t\_0;  
 }  
  
 public Integer[] generateSequence(int length) {  
 final Integer[] sequence = new Integer[length];  
 int previousValue = T\_0;  
 for (int i = 0; i < length; i++) {  
 sequence[i] = (A \* previousValue + C ) % M;  
 previousValue = sequence[i];  
 }  
 return sequence;  
 }  
  
}

1. Написав клас для тестування реалізації лінійного конгруентного датчику ПВЧ

package solution;  
  
import org.junit.jupiter.api.Test;  
  
import static org.junit.jupiter.api.Assertions.\*;  
  
*/\*\*  
 \* Тест для тестування правильності роботи генератора псевдовипадкових чисел  
 \* Параметри генератора були взяті з методичних рекомендацій до курсу  
 \*/*class LinearCongruentSequenceGeneratorTest {  
 private static final int *A* = 5;  
 private static final int *C* = 3;  
 private static final int *M* = 32;  
 private static final int *T\_0* = 7;  
  
 @Test  
 public void testCorrectness() {  
 LinearCongruentSequenceGenerator gen = new LinearCongruentSequenceGenerator(*A*, *C*, *M*, *T\_0*);  
 Integer[] expected = {6, 1, 8};  
 *assertArrayEquals*(gen.generateSequence(3), expected);  
 }  
  
}

1. Написав класи, що реалізує шифрування і дешифрування. Так-як реалізація алгоритму шифрування і дешифрування за цим методом є буквально еквівалентною, то достатньо написати всього лиш один інтеграційний тест для них

package solution;  
  
import solution.helpers.Algorithms;  
  
public class CipherPseudoRandom {  
 private final SequenceGenerator<Integer> sequenceGenerator;  
  
 public CipherPseudoRandom(SequenceGenerator<Integer> sequenceGenerator) {  
 this.sequenceGenerator = sequenceGenerator;  
 }  
  
 public String cipher(String text) {  
 final int length = text.length();  
 Integer[] sequence = sequenceGenerator.generateSequence(length);  
 return Algorithms.*xorTextWithMask*(text, sequence);  
 }  
  
}

package solution;  
  
import solution.helpers.Algorithms;  
  
public class DecipherPseudoRandom {  
 private final SequenceGenerator<Integer> sequenceGenerator;  
  
 public DecipherPseudoRandom(SequenceGenerator<Integer> sequenceGenerator) {  
 this.sequenceGenerator = sequenceGenerator;  
 }  
  
 public String decipher(String text) {  
 final int length = text.length();  
 Integer[] sequence = sequenceGenerator.generateSequence(length);  
 return Algorithms.*xorTextWithMask*(text, sequence);  
 }  
  
}

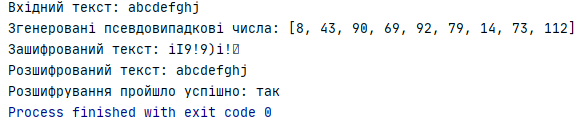
Клас, для тестування вище наведених класів.

package solution;  
  
import org.junit.jupiter.api.BeforeEach;  
import org.junit.jupiter.api.Test;  
  
import static org.junit.jupiter.api.Assertions.*assertEquals*;  
  
public class GeneralTest {  
 private static final int *A* = 5;  
 private static final int *C* = 3;  
 private static final int *M* = 128;  
 private static final int *T\_0* = 1;  
  
 private CipherPseudoRandom cipherObj;  
 private DecipherPseudoRandom decipherObj;  
  
 @BeforeEach  
 public void setup() {  
 SequenceGenerator<Integer> generator = new LinearCongruentSequenceGenerator(*A*, *C*, *M*, *T\_0*);  
 cipherObj = new CipherPseudoRandom(generator);  
 decipherObj = new DecipherPseudoRandom(generator);  
 }  
  
 @Test  
 public void testCorrectnessAscii() {  
 final String initialText = "Some text to check correctness";  
 String encodedText = cipherObj.cipher(initialText);  
 *assertEquals*(initialText, decipherObj.decipher(encodedText));  
 }  
  
 @Test  
 public void testCorrectnessUkrainianEncoding() {  
 final String initialText = "Деякий текст для перевірки алгоритму";  
 String encodedText = cipherObj.cipher(initialText);  
 *assertEquals*(initialText, decipherObj.decipher(encodedText));  
 }  
  
}

1. Реалізував демонстрацію шифрування і дешифрування

package solution;  
  
import java.util.Arrays;  
  
public class Demo {  
 private static final int *A* = 5;  
 private static final int *C* = 3;  
 private static final int *M* = 128;  
 private static final int *T\_0* = 1;  
  
 public static void main(String[] args) {  
 String text = "abcdefghj";  
 LinearCongruentSequenceGenerator gen = new LinearCongruentSequenceGenerator(*A*, *C*, *M*, *T\_0*);  
 CipherPseudoRandom cipher = new CipherPseudoRandom(gen);  
 DecipherPseudoRandom decipher = new DecipherPseudoRandom(gen);  
 System.*out*.printf("Вхідний текст: %s%n", text);  
 System.*out*.println("Згенеровані псевдовипадкові числа: " + Arrays.*toString*(gen.generateSequence(text.length())));  
 String encoded = cipher.cipher(text);  
 System.*out*.printf("Зашифрований текст: %s%n", encoded);  
 String decoded = decipher.decipher(encoded);  
 System.*out*.printf("Розшифрований текст: %s%n", decoded);  
 System.*out*.printf("Розшифрування пройшло успішно: %s", decoded.equals(text)? "так" : "ні");  
 }  
  
}

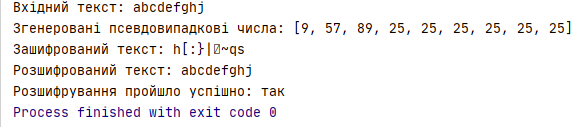
Результат виконання



1. Змінив значення параметрів

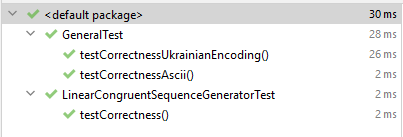
package solution;  
  
import java.util.Arrays;  
  
public class DemoChangedParam {  
 private static final int *A* = 6; *// Змінено параметр, що не задовільняє вимогам* private static final int *C* = 3;  
 private static final int *M* = 128;  
 private static final int *T\_0* = 1;  
  
 public static void main(String[] args) {  
 String text = "abcdefghj";  
 LinearCongruentSequenceGenerator gen = new LinearCongruentSequenceGenerator(*A*, *C*, *M*, *T\_0*);  
 CipherPseudoRandom cipher = new CipherPseudoRandom(gen);  
 DecipherPseudoRandom decipher = new DecipherPseudoRandom(gen);  
 System.*out*.printf("Вхідний текст: %s%n", text);  
 System.*out*.println("Згенеровані псевдовипадкові числа: " + Arrays.*toString*(gen.generateSequence(text.length())));  
 String encoded = cipher.cipher(text);  
 System.*out*.printf("Зашифрований текст: %s%n", encoded);  
 String decoded = decipher.decipher(encoded);  
 System.*out*.printf("Розшифрований текст: %s%n", decoded);  
 System.*out*.printf("Розшифрування пройшло успішно: %s", decoded.equals(text)? "так" : "ні");  
 }  
  
}

Результат виконання



У цій програмі значення параметру було наміренно змінено на неправильне, щоб показати до чого це може призвести. Як бачимо, на відмінну від попередньої програми, яка завжди повертала псевдовипадкові числа, у цьому випадку генератор починає повертати константу після деякої кількості ітерацій.

Результат проходження тестів.



Висновок: на цій лабораторній роботі я засвоїв принципи шифрування гамуванням і написав програмну її реалізацію.