Молдавский Государственный Университет
Факультет Математики и Информатики
Департамент Информатики

Лабораторная работа №1 и практические задания

по курсу “Криптография”

Выполнил: Slavov Constantin,
группа I2302
Проверила: Olga Cerbu, doctor,
conferențiar universitar

Кишинев, 2024

1. Алгоритм RSA

z=1;
for i =s-1 **downto** 0 **do**
z= z^2 **mod** n
if c_i =1 **then** z=z*x **mod** n

7853^4091mod3169 = **451**

c = 4091
x = 7853
n = 3169

$(4091)_{10} = (111111111011)_2$ = Ci

i	Ci	Z = 1
11	1	**z= z^2 mod n** = 1^2 mod 3169 = 1 * 7853 mod 3169 = 1515
10	1	**z= z^2 mod n** = 1515^2 mod 3169 = 1515 * 7853 mod 3169 = 1400
9	1	**z= z^2 mod n** = 1400^2 mod 3169 = 1558 * 7853 mod 3169 = 2634
8	1	**z= z^2 mod n** = 2634^2 mod 3169 = 1015 * 7853 mod 3169 = 760
7	1	**z= z^2 mod n** = 760^2 mod 3169 = 842 * 7853 mod 3169 = 1692
6	1	**z= z^2 mod n** = 1692^2 mod 3169 = 1257 * 7853 mod 3169 = 2955
5	1	**z= z^2 mod n** = 2955^2 mod 3169 = 1430 * 7853 mod 3169 = 2023
4	1	**z= z^2 mod n** = 2023^2 mod 3169 = 1350 * 7853 mod 3169 = 1245
3	1	**z= z^2 mod n** = 1515^2 mod 3169 = 1515 * 7853 mod 3169 = 1833
2	0	**z= z^2 mod n** = 1833^2 mod 3169 = 749
1	1	**z= z^2 mod n** = 749^2 mod 3169 = 88 * 7853 mod 3169 = 222
0	1	**z= z^2 mod n** = 222^2 mod 3169 = 1749 * 7853 mod 3169 = 451

2. Calculul inversei modulo

1. $n_0 = n$; $b_0 = b$; $t_0 = 0$; $t = 1$;

2.$q = n_0 / b_0$;

$r = n_0 – q*b_0$;

3.while $r > 0$ do

3.1.$temp = t_0 – q*t$;

3.2.if $temp >= 0$ then $temp = temp \bmod n$
else $temp = n – ((- temp) \bmod n)$

3.3$n_0 = b_0$; $b_0 = r$; $t_0 = t$; $t = temp$;

3.4.$q = [n_0 / b_0]$; $r = n_0 – q*b_0$;

4.if $b_0 != 1$ then b nu are inversă mod n

else $b^{-1} \bmod n = t$;

$1487^{-1} \bmod 509 = 369$?

b = 1487

n = 509

Step	n_0	b_0	q	r	t_0	t	temp
1	509	1487	0	509	0	1	0
2	1487	509	2	469	1	0	1
3	509	469	1	40	0	1	-2
4	469	40	11	29	1	-2	5
5	40	29	1	11	-2	5	-57
6	29	11	2	7	5	-57	64
7	11	7	1	4	-57	64	-119
8	7	4	1	3	64	-119	183
9	4	3	1	1	-119	183	-302
10	3	1	3	0	183	-302	369

Calcule matematice:

1. $n_0 = 1487$; $b_0 = 509$; $t_0 = 0$; $t = 1$; $q = [1487/ 509] = 2$;

$r = 1487 - 2*509 = 469$; **temp = 0 – 2*1 = - 2;**

temp = $1487 - ((-(-2)) \textbf{ mod } 1487) = 1487 - 2 = 1485 \bmod 1487 = 1485$

2. $n_0 = 509$; $b_0 = 469$; $t_0 = 1$; $t = 1485$; $q = [509/ 469] = 1$; $r = 509 - 1*469 = 40$; temp = $1 - 1*1485 = -1484$;

temp = $1487 - ((-(-1484)) \textbf{ mod } 1487) = 1487 - (1484 \bmod 1487) = 3$;

3. $n_0 = 469$; $b_0 = 40$; $t_0 = 1485$; $t = 3$; $q = 469/ 40 = 11$; $r = 469 - 11*40 = 29$; temp = $1485 - 11*3 = 1452$;

4. $n_0 = 40$; $b_0 = 29$; $t_0 = 3$; $t = 1452$; $q = [40/ 29] = 1$; $r = 40 - 1*29 = 11$; temp = $3 - 1*1452 = -1449$; temp = $1487 - ((-(-1449)) \textbf{ mod } 38) = 1487 - (1449 \bmod 1487) = 38$;

5. $n_0 = 29$; $b_0 = 11$; $t_0 = 1452$; $t = 38$; $q = 29/11 = 2$; $r = 29 - 2*11 = 7$; temp = $1452 - 2*38 = 1376$;

6. $n_0 = 11$; $b_0 = 7$; $t_0 = 38$; t= 1376; q = 11/ 7= 1; r = 11 – 1*7 = **4**;
temp = 38–1*1376 = -1338;
temp = 1487– ((-(-1338)) **mod** 1487) = 1487– (1338 mod 1487) = 149

7. $n_0 = 7$; $b_0 = 4$; $t_0 = 1376$; t=149; q = 7/4= 1; r = 7– 1*4 = 3;
temp = 1376– 1*149= 1227;

8. $n_0 = 4$; $b_0 = 3$; $t_0 = 63$; t=149; q = 4/3 = 1; r = 4– 1*3 = 1;
temp = 149– 1*1227 = -1078;

temp = 1487– ((-(-1078)) **mod** 1487) = 1487– (1078 mod 1487) = 409

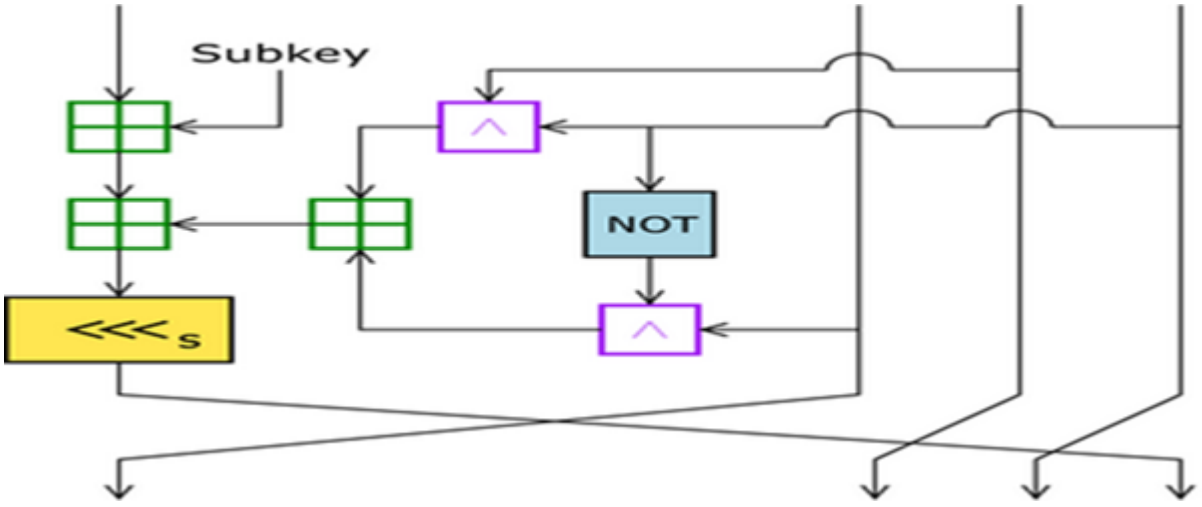
9. $n_0 = 3$; $b_0 = 1$; $t_0 = 1227$; **t=409**; q = 3/1= 3; r = 3– 3*1 = 0;
temp = 1227– 3*409= 0;

Răspuns: b^{-1} **mod** n = t ; 1487^{-1} mod 509= 369

3. Realizare Algoritmul RIVEST CODE 2

Сообщение шифрования m = “domashka” в бинарном виде :01000100 01001111 01001101 01000001 01010011 01001000 01001011 01000001.

Subkey –CS.



a => (01000100 01001111**), b => (**01001101 01000001**)**

c => (01010011 01001000**), d => (**01001011 01000001**)**

Subkey SC=> (01010011 01000011)

Первый шаг : a + subkey => (01000100 01001111 + **01010011 01000011**) mod 2^{32} = (17487+**21315**) mod 2^{32} = (38802 mod 2^{32} = 38802=> **10010111 10010010**

Второй шаг : c^d = 01010011 $01001000^{01001011\ \ 01000001}$ = 01000011 01000000

Третий шаг : not(d)^b = not(01001011 01000001)$^{01001101\ 01000001}$ = 10110110 $10111110^{01001101\ 01000001}$ = 00000100 00000000

Четвертый шаг : (c^d + not(d)^b) mod 2^{32} = (01000011 01000000 +00000100 00000000) mod 2^{32} = (17216 + 1024) mod 2^{32} = 18240 mod 2^{32} = 18240=> 10001110 1000000

Пятый шаг : ((a + subkey) + (c^d + not(d)^b)) mod 2^{32} = (**10010111 10010010** + 10001110 1000000) mod 2^{32} = (38802 + 18240) mod 2^{32} = 57042=> **11011110 11010010**

Шестой шаг : Переместить биты с пятого шага влево на 4 (<<<4) = 00000000000000001101111011010010000

Седьмой шаг : b =>01001101 01000001), c =>01010011 01001000)

d => (01001011 01000001), a => (00000000000011011110110100100000)

Восьмой шаг : Конкатенация (склеивание) b, c, d и a = (01001101 01000001 01010011 01001000 01001011 01000001 00000000000011011110110100100000) =>

Result = "mashka" + "????"

Расшифровка

Первый шаг : Переместить биты с шестого шага шифровки вправо на 4 (>>>4) = 00000000000011011110110100100000=> 00000000000000001101111011010010

Второй шаг : c^d = 01010011 01001000$^{01001011\ \ 01000001}$ = 01000011 01000000

Третий шаг : not(d)^b = not(01001011 01000001)$^{01001011\ \ 01000001}$= 10110110 10111110$^{01001101\ 01000001}$ = 00000100 00000000

Четвертый шаг : (c^d + not(d)^b) mod 2^{32} = (01000011 01000000 + 00000100 00000000) mod 2^{32} = (17216 + 1024) mod 2^{32} = 18240 mod 2^{32} = 18240=> 10001110 1000000

Пятый шаг : (a - (c^d + not(d)^b)) mod 2^{32} = (00000000000000001101111011010010 - 10001110 1000000) mod 2^{32} = (57042 - 18240) mod 2^{32} = 38802=> **10010111 10010010**

Шестой шаг : ((a – SC) mod 2^{32} = (**10010111 10010010** - **01010011 01000011**) mod 2^{32} = (38802 – 21315) mod 2^{32} = 17487 => **01000100 01001111**

Седьмой шаг : Конкатенация (склеивание) a, b, c и d = (01000100 01001111 01001101 0100000101010011 01001000 01001011 01000001) => “domashka”

Result = "domashka"

4. Digital signature with RSA (первых 4х букв имени)

1. Генерация ключей RSA:
p = 19, q = 17

Модуль n:
n=p×q=19×17=323

Функция Эйлера φ(n):
φ(n)=(p−1)×(q−1)=(19−1)×(17−1)=18×16=288

Выбираем открытый ключ e=13 (меньшее простое число, взаимно простое с φ(n)):
Теперь проверим gcd(e,φ(n))=1, что верно, поскольку 13 и 323 взаимно просты.

Вычисление закрытого ключа d:
d должно удовлетворять уравнению $d \times e \equiv 1 \mod \varphi(n)d$. Используя расширенный алгоритм Евклида, находим d.
В данном случае d=133.

2. Шифрование имени "Kost":

Используем ASCII-кодировку:

- **"K" = 75**
- **"o" = 111**
- **"s" = 115**
- **"t" = 116**

Теперь шифруем каждую букву отдельно, используя формулу **c = m^e mod n,** где e=13 и n=323**.**

Для "**S**" (83): c=75^13 mod 323=227

Для "**O**" (79): c=111^13 mod 323=195

Для "**F**" (70): c=115^13 mod 323=115

Для "**I**" (73): c=116^13 mod 323=22

Зашифрованное сообщение для "Kost" будет: 227 195 115 22.

3. Дешифрование:

Теперь дешифруем сообщение, используя формулу m=c^d mod n, где d=133 и n=323.

Для 227: m=227^133 mod 323=75 (буква "K")

Для 195: m=195^133 mod 323=111 (буква "o")

Для 115: m=115^133 mod 323=115 (буква "s")

Для 22: m=22^133 mod 323=116 (буква "t")

В результате мы получаем исходные символы: "Kost".

Лабораторная работа №1. Алгоритм RC5, его принципы и код к нему.

RC5 — это симметричный блочный шифр, разработанный в 1994 году Рональдом Ривестом (соавтором RSA). Алгоритм RC5 обладает рядом особенностей, которые делают его гибким и простым для реализации:

1. Простота: RC5 использует небольшое количество простых операций — сложение, исключающее ИЛИ (XOR), и циклический сдвиг. Благодаря этому его легко реализовать как на программном, так и на аппаратном уровне.

2. Гибкость: RC5 поддерживает переменные параметры:

- Размер блока (обычно 32, 64 или 128 бит),

- Размер ключа (от 0 до 2040 бит),

-Количество раундов (от 0 до 255). Эти параметры позволяют настраивать алгоритм для различных целей и уровней безопасности.

3. Архитектура шифра: Основной принцип RC5 — это несколько раундов преобразования данных, где каждый раунд состоит из сложения, XOR и циклических сдвигов. Эти операции усиливают запутывание и рассеивают информацию по всему блоку.

4. Ключевая экспансия: Перед шифрованием ключ расширяется в массив для использования в каждом раунде.

5. Применение: RC5 использовался в различных приложениях, но был заменен более современными алгоритмами, такими как AES, из-за появления новых угроз.

Для демонстрации работы алгоритма RC5 я попытался написать код, который будет шифровать по данному принципу определенный текст и расшифровывать его.

Давайте рассмотрим данный код получше:

```
import
org.bouncycastle.jce.provider.BouncyCastleProvider;


import javax.crypto.Cipher;

import javax.crypto.KeyGenerator;

import javax.crypto.SecretKey;

import java.security.Security;

import java.util.Base64;


public class RC5Encryption {

    private static final String ALGORITHM = "RC5";

      private static final int KEY_SIZE = 128; //
Длина ключа в битах


     public static void main(String[] args) throws
Exception {

                // 1. Подключаем Bouncy Castle как
криптографический провайдер

                              Security.addProvider(new
BouncyCastleProvider());
```

```
        // 2. Генерация секретного ключа для RC5
                            KeyGenerator keyGen =
KeyGenerator.getInstance(ALGORITHM, "BC");
        keyGen.init(KEY_SIZE);
        SecretKey secretKey = keyGen.generateKey();

        // 3. Текст для шифрования
             String plainText = "Пример текста для
шифрования";

        // 4. Шифрование
           String encryptedText = encrypt(plainText,
secretKey);
         System.out.println("Зашифрованный текст: " +
encryptedText);

        // 5. Расшифровка
                           String decryptedText =
decrypt(encryptedText, secretKey);
          System.out.println("Расшифрованный текст: "
+ decryptedText);
    }
    // Метод шифрования текста
     public static String encrypt(String plainText,
SecretKey secretKey) throws Exception {
```

```
                                    Cipher cipher =
Cipher.getInstance("RC5/ECB/PKCS5Padding", "BC");

        cipher.init(Cipher.ENCRYPT_MODE, secretKey);

                                byte[] encryptedBytes =
cipher.doFinal(plainText.getBytes());

                                                    return
Base64.getEncoder().encodeToString(encryptedBytes);

    }



    // Метод расшифровки текста

            public static String decrypt(String
encryptedText, SecretKey secretKey) throws Exception
{

                                    Cipher cipher =
Cipher.getInstance("RC5/ECB/PKCS5Padding", "BC");

        cipher.init(Cipher.DECRYPT_MODE, secretKey);

                                byte[] decodedBytes =
Base64.getDecoder().decode(encryptedText);

                                byte[] decryptedBytes =
cipher.doFinal(decodedBytes);

        return new String(decryptedBytes);

    }

}
```

Как работает данный код:

1. Подключение провайдера:

```
Security.addProvider(new BouncyCastleProvider());
```

Этот шаг добавляет поддержку дополнительных криптографических алгоритмов в приложение через библиотеку Bouncy Castle. Без этого Java не сможет найти нужный алгоритм.

2. Генерация ключа:

```
KeyGenerator keyGen = KeyGenerator.getInstance(ALGORITHM, "BC");
keyGen.init(KEY_SIZE);
SecretKey secretKey = keyGen.generateKey();
```

Здесь происходит создание ключа для алгоритма. Мы используем генератор ключей и указываем, что хотим работать с алгоритмом. Размер ключа задается 128 бит, а для генерации используется Bouncy Castle. Этот ключ позже используется для шифрования и расшифровки данных.

3. Текст для шифрования:

```
String plainText = "Всем привет, это мой текст, который необходимо зашифровать.";
```

Это текст, который будет преобразован в зашифрованную форму.

4. Шифрование текста:

```
String encryptedText = encrypt(plainText, secretKey);
```

Здесь вызывается функция шифрования, передавая в неё исходный текст и сгенерированный ключ. Шифрование происходит внутри функции `encrypt`.

5. Метод `encrypt`:

```
Cipher cipher = Cipher.getInstance("RC5/ECB/PKCS5Padding", "BC");

cipher.init(Cipher.ENCRYPT_MODE, secretKey);

byte[] encryptedBytes = cipher.doFinal(plainText.getBytes());

return Base64.getEncoder().encodeToString(encryptedBytes);
```

- `Cipher` создается с алгоритмом шифрования и режимом "ECB" (Electronic Codebook) и типом заполнения "PKCS5Padding". Это говорит о том, что данные будут шифроваться блоками фиксированной длины с добавлением выравнивающих байтов (padding).

- Метод `cipher.init(Cipher.ENCRYPT_MODE, secretKey)` указывает, что объект будет использоваться для шифрования.

- Метод `doFinal()` выполняет само шифрование, преобразуя исходный текст в байты. Результат кодируется в формат Base64 для удобного представления в строковом виде.

6. Расшифровка текста:

```
String decryptedText = decrypt(encryptedText, secretKey);
```

Здесь вызывается функция для обратного преобразования зашифрованного текста в исходный, используя тот же ключ.

7. Метод `decrypt`:

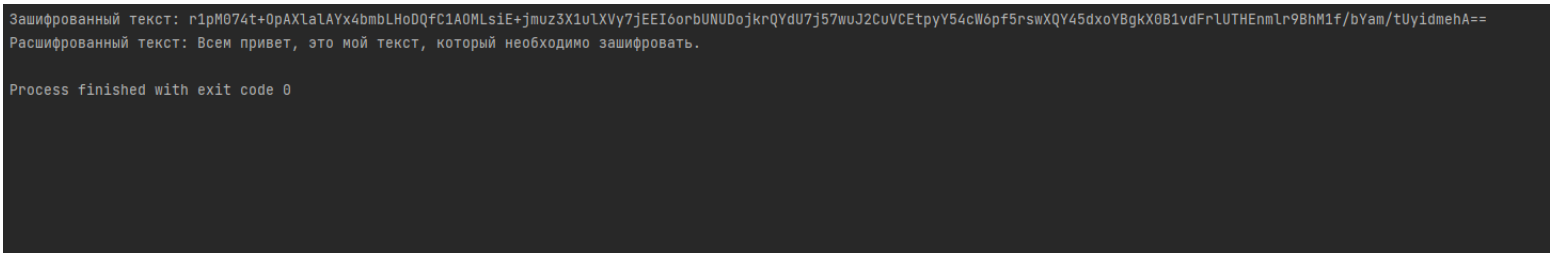
```
Cipher cipher = Cipher.getInstance("RC5/ECB/PKCS5Padding", "BC");

cipher.init(Cipher.DECRYPT_MODE, secretKey);

byte[] decodedBytes = Base64.getDecoder().decode(encryptedText);

byte[] decryptedBytes = cipher.doFinal(decodedBytes);

return new String(decryptedBytes);
```

- Как и в `encrypt`, создается объект `Cipher`, но на этот раз с режимом расшифровки (`Cipher.DECRYPT_MODE`).

- Входной текст сначала декодируется из формата Base64 обратно в байты, а затем эти байты расшифровываются методом `doFinal()`, возвращая исходный текст.

- Код сначала генерирует ключ, использует его для шифрования заданного текста, а затем, используя тот же ключ, восстанавливает исходные данные. Все промежуточные результаты (зашифрованный текст) преобразуются в строку через Base64 для удобства отображения.

Вот что выводится в консоли в редакторе кода Intelij IDEA:



Алгоритм RC5 является надежным способом шифрования информации, и не смотря на то, что имеется много аналогов данного метода шифрования, он до сих пор является популярным с некоторых типах работы с данными.