

MINISTRY OF EDUCATION OF REPUBLIC OF MOLDOVA
TECHNICAL UNIVERSITY OF MOLDOVA
FACULTY OF COMPUTERS, INFORMATICS AND MICROELECTRONICS
SOFTWARE ENGINEERING DEPARTMENT

CRYPTOGRAPHY AND SECURITY

LABORATORY WORK #5

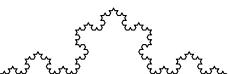
Public Key Cryptography

Author: Timur CRAVTOV
std. gr. FAF-231

Verified by: Maia ZAICA
asist. univ

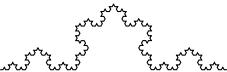


Chișinău
2025



Contents

1	Introduction	2
2	RSA Algorithm	2
3	ElGamal Algorithm	3
4	Diffie-Hellman Key Exchange	3
5	Implementation	4
5.1	Task 1	4
5.2	Task 2	6
5.3	Task 3	7
6	Conclusion	9



1 Introduction

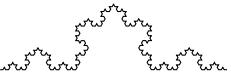
Public key cryptography is a cryptographic system that uses pairs of keys: public keys, which may be disseminated widely, and private keys, which are known only to the owner. This lab focuses on three main algorithms: RSA, ElGamal, and Diffie-Hellman key exchange.

2 RSA Algorithm

RSA (Rivest-Shamir-Adleman) is one of the first public-key cryptosystems and is widely used for secure data transmission. The security of RSA relies on the practical difficulty of factoring the product of two large prime numbers, the "factoring problem".

The steps of RSA algorithm are as follows [2]:

- Key Generation:
 - Choose two distinct large random prime numbers p and q .
 - Compute $n = p \times q$. n is used as the modulus for both the public and private keys.
 - Compute the totient $\phi(n) = (p - 1)(q - 1)$.
 - Choose an integer e such that $1 < e < \phi(n)$ and $\gcd(e, \phi(n)) = 1$; e is released as the public key exponent.
 - Determine d as $d \equiv e^{-1} \pmod{\phi(n)}$; d is kept as the private key exponent.
 - The public key is (e, n) and the private key is (d, n) .
- Encryption:
 - Convert the plaintext message M into an integer m such that $0 \leq m < n$.
 - Compute the ciphertext c using the formula $c \equiv m^e \pmod{n}$.
- Decryption:
 - Compute the original message m using the formula $m \equiv c^d \pmod{n}$.
 - Convert the integer m back to the plaintext message M .



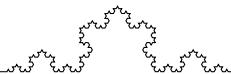
3 ElGamal Algorithm

ElGamal is an asymmetric key encryption algorithm for public-key cryptography which is based on the Diffie-Hellman key exchange. It provides both confidentiality and authentication. The steps of ElGamal algorithm are as follows [2]:

- Key Generation:
 - Choose a large prime number p and a generator g of the multiplicative group of integers modulo p .
 - Choose a private key x such that $1 < x < p - 2$.
 - Compute the public key $y \equiv g^x \pmod{p}$.
 - The public key is (p, g, y) and the private key is x .
- Encryption:
 - Convert the plaintext message M into an integer m such that $0 \leq m < p$.
 - Choose a random integer k such that $1 < k < p - 2$ and $\gcd(k, p - 1) = 1$.
 - Compute $c_1 \equiv g^k \pmod{p}$.
 - Compute $c_2 \equiv m \cdot y^k \pmod{p}$.
 - The ciphertext is the pair (c_1, c_2) .
- Decryption:
 - Compute $s \equiv c_1^x \pmod{p}$.
 - Compute the modular inverse of s , denoted as s^{-1} .
 - Recover the original message $m \equiv c_2 \cdot s^{-1} \pmod{p}$.
 - Convert the integer m back to the plaintext message M .

4 Diffie-Hellman Key Exchange

Diffie-Hellman is a method of securely exchanging cryptographic keys over a public channel. It allows two parties to jointly establish a shared secret key, which can then be used for symmetric encryption of subsequent communications. The steps of Diffie-Hellman key exchange are as follows [2]:



- Public Parameters: Choose a large prime number p and a generator g of the multiplicative group of integers modulo p .
- Key Exchange:
 - Alice chooses a private key a such that $1 < a < p - 2$ and computes her public key $A \equiv g^a \pmod{p}$.
 - Bob chooses a private key b such that $1 < b < p - 2$ and computes his public key $B \equiv g^b \pmod{p}$.
 - Alice and Bob exchange their public keys A and B over the public channel.
- Shared Secret Computation:
 - Alice computes the shared secret key $s \equiv B^a \pmod{p}$.
 - Bob computes the shared secret key $s \equiv A^b \pmod{p}$.
 - Both computations yield the same shared secret key s .
- Symmetric Encryption: Alice and Bob can now use the shared secret key s for symmetric encryption of their communications using algorithms like AES.

5 Implementation

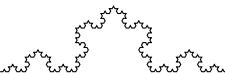
5.1 Task 1

Generate RSA keys and encrypt and decrypt your name. The value of n should be at least 2048 bits.

```

1 fun keyGen(isPublicKeyDefault: Boolean = true): KeyGenData {
2     val p = randomPrime(1024, 2100)
3     var q = randomPrime(1024, 2100)
4
5     while (p == q) {
6         q = randomPrime(1024, 2100)
7     }
8
9     val n = p * q
10    val phi = (p - BigInteger.ONE) * (q - BigInteger.ONE)
11
12    val e = if (isPublicKeyDefault)
13        BigInteger.valueOf(65537)
14    else

```

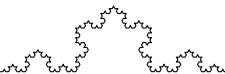


```
15     generateSequence { randomInt(BigInteger.TWO, phi -  
16         BigInteger.ONE) }  
17         .first { it.gcd(phi) == BigInteger.ONE }  
18  
19     val d = e.modInverse(phi)  
20  
21     return KeyGenData(e, d, n)  
22 }
```

```
1 fun encryptRsa(message: BigInteger, publicKey: BigInteger, n:  
2     BigInteger): BigInteger {  
3     require(message.bitLength() <= 2048)  
4  
5     val cipherText = message.modPow(publicKey, n)  
6  
7     return cipherText  
8  
9 }  
10 fun decryptRsa(encryptedMessage: BigInteger, privateKey:  
11     BigInteger, n: BigInteger): String {  
12     val message = encryptedMessage.modPow(privateKey, n)  
13  
14     return bigIntToAsciiString(message)  
15 }
```

```

Run tabs [lab5.Task1Kt.main()]
Key data. Public key: 65537;
Private key:
280943508598648142934680307401036464147664522532590502499898744994232414162800618137359867837314565978853322475438773407467958550688118351165611156979512110284887037562
7784238736253860048807507354082029948325481633063008769968709039095124027305066551755865501733408886295097157292996080553554698913020928497516330552462089908501686425
98352492381952702715930105924480641295160694785535128145834899875264308774995523606534291877553323409620985665248985440370662147695946028086797154748292705374
43344961785176028090420934293761374205572629525437430518623165789480465247204562835990469985938265904020980962752326986384907179457730975273423392900673728352952865473315
706896498111902414399153660744261444160355153445800467034576207683451482153583569031167195284103507510406341447038459346169346177436353553639488801022465284222734109
0340683910742555269849424952972156158796344925632384982149801787980991821498463214104884506989538951836993665,
n:
182335063606947943587939624738795309475594071855652977058143948374629602593331759842171684085514429645089145864497768107566372313028765312319608495142569107914181614980
707165432833892082107553584337961699423061091093335715743156352045224929033210642959241845651326793604487242844102902397813807826328606185293540579871427094338814369412
751854554687664317478476278091813745025626747352293973220397041924953784167152981666594460303405493951397749979830809080903014265954176020924452247893583709423605602
080229964974123501540864260414433660154182406715105096978116407531594304816914598574207037477181098057167799619304513313998443285352870140676236561811
5959800474166309653314471665864260414433660154182406715105096978116407531594304816914598574207037477181098057167799619304513313998443285352870140676236561811
79016361314387760679071844241821034056468585635062365975271786107850945734577262787457887217671769014990425873
ciphertext:
2584828862719795123356721295671287623755779458659576623015859412504169897868472480407776078228590919330390396333789441975054622149034736706434129260935657495472499936191
08581416349075915470808070431187355416896430862041426017004532931242187934857665774208307811097324408813621526260500952244006293536834791635834836318333418068034199461
942974787049489491791677750707058463431386510010166639403143199312015022139732461873430037800516903407608228350879079125923748963678277877120603974301729491752932
3844395594857144292418897054383676475215293161659794009620660603439288508480619436989545802109227764649094265481958025672566858194760723817895366798614039871709644191672
48279996054318040273347618203624937877760075442739076544875852933569199478696051079198735338078481717240618680322723055806310636109804173956556528385971504457719996592
245266344087183408726568068876038100130938657974499004872499674389632217178196173415498935802935324598606
2584828862719795123363721925671287623755779458659576623015859412504169897868472480407776078228359019303903963337894419750546221490347367064341292609356574954724999361910
08581416349075915470808070431187355416896430862041426017004532931242187934857665774208307811097324408813621526260500952244006293536834791635834836318333418068034199461
42974787049489491791677750707058463431386510010166639403143199312015022139732461873430037800516903407608228350879079125923748963678277877120603974301729491752932
844395594857144292418897054383676475215293161659794009620660603439288508480619436989545802109227764649094265481958025672566858194760723817895366798614039871709644191672
82799960543180402733476182036249378777600754427390765448758529335691994786960510791987353380784817172406186803227230558063106361098041739565565283859715044577199965922
45266344087183408726568068876038100130938657974499004872499674389632217178196173415498935802935324598606
2584828862719795123363721925671287623755779458659576623015859412504169897868472480407776078228359019303903963337894419750546221490347367064341292609356574954724999361910
08581416349075915470808070431187355416896430862041426017004532931242187934857665774208307811097324408813621526260500952244006293536834791635834836318333418068034199461
42974787049489491791677750707058463431386510010166639403143199312015022139732461873430037800516903407608228350879079125923748963678277877120603974301729491752932
844395594857144292418897054383676475215293161659794009620660603439288508480619436989545802109227764649094265481958025672566858194760723817895366798614039871709644191672
82799960543180402733476182036249378777600754427390765448758529335691994786960510791987353380784817172406186803227230558063106361098041739565565283859715044577199965922
45266344087183408726568068876038100130938657974499004872499674389632217178196173415498935802935324598606
2584828862719795123363721925671287623755779458659576623015859412504169897868472480407776078228359019303903963337894419750546221490347367064341292609356574954724999361910
08581416349075915470808070431187355416896430862041426017004532931242187934857665774208307811097324408813621526260500952244006293536834791635834836318333418068034199461
42974787049489491791677750707058463431386510010166639403143199312015022139732461873430037800516903407608228350879079125923748963678277877120603974301729491752932
844395594857144292418897054383676475215293161659794009620660603439288508480619436989545802109227764649094265481958025672566858194760723817895366798614039871709644191672
82799960543180402733476182036249378777600754427390765448758529335691994786960510791987353380784817172406186803227230558063106361098041739565565283859715044577199965922
45266344087183408726568068876038100130938657974499004872499674389632217178196173415498935802935324598606
2584828862719795123363721925671287623755779458659576623015859412504169897868472480407776078228359019303903963337894419750546221490347367064341292609356574954724999361910
08581416349075915470808070431187355416896430862041426017004532931242187934857665774208307811097324408813621526260500952244006293536834791635834836318333418068034199461
42974787049489491791677750707058463431386510010166639403143199312015022139732461873430037800516903407608228350879079125923748963678277877120603974301729491752932
844395594857144292418897054383676475215293161659794009620660603439288508480619436989545802109227764649094265481958025672566858194760723817895366798614039871709644191672
82799960543180402733476182036249378777600754427390765448758529335691994786960510791987353380784817172406186803227230558063106361098041739565565283859715044577199965922
45266344087183408726568068876038100130938657974499004872499674389632217178196173415498935802935324598606
2584828862719795123363721925671287623755779458659576623015859412504169897868472480407776078228359019303903963337894419750546221490347367064341292609356574954724999361910
08581416349075915470808070431187355416896430862041426017004532931242187934857665774208307811097324408813621526260500952244006293536834791635834836318333418068034199461
42974787049489491791677750707058463431386510010166639403143199312015022139732461873430037800516903407608228350879079125923748963678277877120603974301729491752932
844395594857144292418897054383676475215293161659794009620660603439288508480619436989545802109227764649094265481958025672566858194760723817895366798614039871709644191672
82799960543180402733476182036249378777600754427390765448758529335691994786960510791987353380784817172406186803227230558063106361098041739565565283859715044577199965922
45266344087183408726568068876038100130938657974499004872499674389632217178196173415498935802935324598606
2584828862719795123363721925671287623755779458659576623015859412504169897868472480407776078228359019303903963337894419750546221490347367064341292609356574954724999361910
08581416349075915470808070431187355416896430862041426017004532931242187934857665774208307811097324408813621526260500952244006293536834791635834836318333418068034199461
42974787049489491791677750707058463431386510010166639403143199312015022139732461873430037800516903407608228350879079125923748963678277877120603974301729491752932
844395594857144292418897054383676475215293161659794009620660603439288508480619436989545802109227764649094265481958025672566858194760723817895366798614039871709644191672
82799960543180402733476182036249378777600754427390765448758529335691994786960510791987353380784817172406186803227230558063106361098041739565565283859715044577199965922
45266344087183408726568068876038100130938657974499004872499674389632217178196173415498935802935324598606
2584828862719795123363721925671287623755779458659576623015859412504169897868472480407776078228359019303903963337894419750546221490347367064341292609356574954724999361910
08581416349075915470808070431187355416896430862041426017004532931242187934857665774208307811097324408813621526260500952244006293536834791635834836318333418068034199461
42974787049489491791677750707058463431386510010166639403143199312015022139732461873430037800516903407608228350879079125923748963678277877120603974301729491752932
844395594857144292418897054383676475215293161659794009620660603439288508480619436989545802109227764649094265481958025672566858194760723817895366798614039871709644191672
82799960543180402733476182036249378777600754427390765448758529335691994786960510791987353380784817172406186803227230558063106361098041739565565283859715044577199965922
45266344087183408726568068876038100130938657974499004872499674389632217178196173415498935802935324598606
2584828862719795123363721925671287623755779458659576623015859412504169897868472480407776078228359019303903963337894419750546221490347367064341292609356574954724999361910
08581416349075915470808070431187355416896430862041426017004532931242187934857665774208307811097324408813621526260500952244006293536834791635834836318333418068034199461
42974787049489491791677750707058463431386510010166639403143199312015022139732461873430037800516903407608228350879079125923748963678277877120603974301729491752932
844395594857144292418897054383676475215293161659794009620660603439288508480619436989545802109227764649094265481958025672566858194760723817895366798614039871709644191672
82799960543180402733476182036249378777600754427390765448758529335691994786960510791987353380784817172406186803227230558063106361098041739565565283859715044577199965922
45266344087183408726568068876038100130938657974499004872499674389632217178196173415498935802935324598606
2584828862719795123363721925671287623755779458659576623015859412504169897868472480407776078228359019303903963337894419750546221490347367064341292609356574954724999361910
08581416349075915470808070431187355416896430862041426017004532931242187934857665774208307811097324408813621526260500952244006293536834791635834836318333418068034199461
42974787049489491791677750707058463431386510010166639403143199312015022139732461873430037800516903407608228350879079125923748963678277877120603974301729491752932
844395594857144292418897054383676475215293161659794009620660603439288508480619436989545802109227764649094265481958025672566858194760723817895366798614039871709644191
```

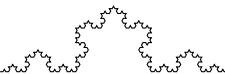


```
8
9     val y = (message * km).mod(p)
10
11    return ElGamalEncryptionData(ke, y)
12
13}
14
15 fun decryptElGamal(privateData: ElGamalSetupData, encData:
16     ElGamalEncryptionData): BigInteger {
17
18     val (kE, y) = encData
19
20     val d = privateData.d
21     val p = privateData.p
22
23     val km = kE.modPow(d, p)
24     val kminverse = km.modInverse(p)
25     val x = (y * kminverse).mod(p)
26
27     return x
}
```

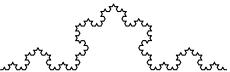
5.3 Task 3

Implement key exchange using Diffie-Hellman algorithm between Alice and Bob which uses AES with 256 bits.

```
1 package lab5
2
3 import lab5.algorithms.randomUUID
4 import java.math.BigInteger
5 import javax.crypto.Cipher
6 import javax.crypto.spec.IvParameterSpec
7 import javax.crypto.spec.SecretKeySpec
8 import kotlin.random.Random
9
10 fun main() {
11
12     val p = BigInteger("7919")
```



```
13  val alpha = BigInteger("3")
14
15  // alice private key
16  val a = randomInt(BigInteger.ONE, p - BigInteger.ONE)
17
18  // bob private key
19  val b = randomInt(BigInteger.ONE, p - BigInteger.ONE)
20
21  // alice public key
22  val A = alpha.modPow(a, p)      // alpha^a mod p
23
24  // bob public key
25  val B = alpha.modPow(b, p)      // alpha^b mod p
26
27  // alice computed common key
28  val kA = B.modPow(a, p)        // (alpha^b)^a mod p
29
30  // bob computed common key
31  val kB = A.modPow(b, p)        // (alpha^a)^b mod p
32
33  require(kA == kB)
34
35  // use common key as key for aes
36  val raw = kA.toByteArray()
37  val secretBytes =
38    if (raw.size >= 32) raw.copyOfRange(raw.size - 32, raw.size)
39    else ByteArray(32 - raw.size) + raw
40
41  val aesKey = SecretKeySpec(secretBytes, "AES")
42
43  // random iv
44  val iv = ByteArray(16).also { Random.nextBytes(it) }
45  val ivSpec = IvParameterSpec(iv)
46
47  val plaintext = "Some text which alice encrypted"
48  println("plaintext: $plaintext")
49
```



```

50 // Encrypt
51 val cipherEnc = Cipher.getInstance("AES/CBC/PKCS5Padding")
52 cipherEnc.init(Cipher.ENCRYPT_MODE, aesKey, ivSpec)
53 val ciphertext = cipherEnc.doFinal(plaintext.toByteArray())
54
55 println("Ciphertext (hex): " + ciphertext.toString("") {
56     "%02x".format(it) })
57
58 // decrypt
59 val cipherDec = Cipher.getInstance("AES/CBC/PKCS5Padding")
60 cipherDec.init(Cipher.DECRYPT_MODE, aesKey, ivSpec)
61 val decrypted = cipherDec.doFinal(ciphertext).toString(
62    Charsets.UTF_8)
63
64 println("Decrypted: $decrypted")
65
66 }

```

```

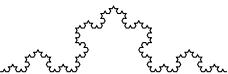
Common key:
2586591917649613654206336969055501449999106686082802587998510239965201413119310476073339104642053152826872257794038549340059691247262320139038983953025409231820004949734
56094017645071171401278744890842091085926448220866042631563289909066087003755166436437161372197346310544635482211793639765343917901728816360644457882569613687366769228
076718936072540550650138303177345758582725373284906086486716820272139327675753051917461890861585113862837809844375444997298004182547874163698341377880599689544015350783
46391583708740064025759697982528471227594540252630069184144994432481996632985544341149453052301920613288836224
plaintext: Some text which alice encrypted
Ciphertext (hex): 794c3c3bf3c89e5b219e7a158984131cac92f59086e4e1e302a7088be0fab804
Decrypted: Some text which alice encrypted
22:55:00: Execution finished ':lab5.Task3Kt.main()'.

```

Figure 3: Diffie-Hellman key exchange output

6 Conclusion

During this lab, I have implemented three important cryptographic algorithms: RSA, ElGamal, and Diffie-Hellman key exchange. Each of these algorithms plays a crucial role in ensuring secure communication in the digital world. RSA was used for encrypting and decrypting my name, demonstrating its effectiveness in public-key cryptography. ElGamal provided another layer of security through its asymmetric encryption method. Finally, the Diffie-Hellman key exchange allowed Alice and Bob to securely share a secret key, which was then used with AES for symmetric encryption. Overall, this lab has enhanced my understanding of cryptographic principles and their practical applications.



References

- [1] GitHub repository <https://github.com/TimurCravtov/CryptographyAndSecurityLabs>
- [2] Lecture Notes CS