

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

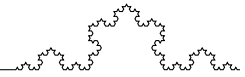
Public Key Cryptography

Author: Timur CRAVTOV
std. gr. FAF-231

Verified by: Maia ZAICA
asist. univ



Chişinău
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

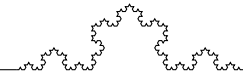
Public key cryptography is a cryptographic system that uses pairs of keys: public keys, which may be disseminated widely, and private keys, which are known only to the owner. This lab focuses on three main algorithms: RSA, ElGamal, and Diffie-Hellman key exchange.

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 .



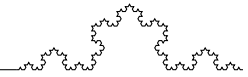
3 ElGamal Algorithm

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]:



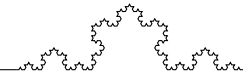
- Public Parameters: Choose a large prime number p and a generator g of the multiplicative group of integers modulo p .
- Key Exchange:
 - Alice chooses a private key a such that $1 < a < p - 2$ and computes her public key $A \equiv g^a \pmod{p}$.
 - Bob chooses a private key b such that $1 < b < p - 2$ 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

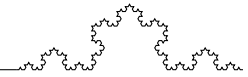
Generate RSA keys and encrypt and decrypt your name. The value of `n` should be at least 2048 bits.

```
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
4     require(message.bitLength() <= 2048)
5
6     val cipherText = message.modPow(publicKey, n)
7
8     return cipherText
9 }
10 fun decryptRsa(encryptedMessage: BigInteger, privateKey:
11     BigInteger, n: BigInteger): String {
12
13     val message = encryptedMessage.modPow(privateKey, n)
14
15     return bigIntToAsciiString(message)
16 }
```



```

Run [labs.Task2Kt.main()] x
Key data. Public key: 65537;
Private key:
2809435085986481429346803074010364641476645225329505024989874499423241416280061813763598678373145659788532247543877340746795855068818351165611156979512110284887037562
77842387365253680048807507354082029948325481633063008769687090390965124027303066551755865501733340888629509715729299608055355469891302092849751633055246208990801686425
98352492381952702715987721159301059244806412951606947855351281458348998752643408774995523606534291877553323240962098565248985440370662147695946028086797154748292705374
43344961785176020904209342937613742055726295254374305186231657894804652427045628359904698859382659040209809627523269863849071794577309752734233929006473728352952865473315
7068964981190241443991536607442614444106335153445800467034576207683451482153583569031167191522841035075104063414470384593461693461774363535363948801022465284222734109
0340683910742555269849424952927156158796344972563238498214980178798069918214986432141404884506989538951836993665,
n:
182335063606947943587939624738975309475594076185565297705841394837462960259331759842171684080551442964508914587768180756637231130287653123196084951425691079141816614980
707165432833892080215755358433796196942306109109335715743156352045224929033210642959241845651326793604487242844102902397813807826328606185293540579871427094338814369412
751854554687664317478479627809181374502562674352293973222839704192495378416715298166654944603034054939513977499783808090809030149265945176020924452247893583709423605602
080215382906123501910399029126003320335051824725403445838578902896933505023231902657334498811371003171362434996060252837917657435807705697266723729401126348175790368
5958004741630965331447180015586642604144336601541057182408715105096997811640753159430481605914159857420703747718109805716779699103451331398443285352870140676236561811
790163613138776006790718442418210340564685856350623659752717861078509457453477262787457887216741760914990425873
ciphertext:
25848288627197951233637219256712876237557945865957662301585941250416989786847248040777607822835901933039039633789441975054622149034736706434129260935657495472499936191
0858144163490759154700807043118735541689643086206142601700453293124218793485766567742083078110973244088136215262605009522440062935368347916358348436318333418068034199461
9429747870494894941791677507070705846343138651001016663940314313993312015022139732618734300378005816903407608228235708970125923748963678277787120603974301729491752932
3844395594857144924188970543867647251329316165979400962066060343392885084806194369895458021092276649009426548195850256725668581947607238178953667090614039871709641916724
4827999605431804027334761820362493787776007544273907654048755829335691994786960510791987353380784481717240618608327723055806310636109804173956556528385971504457719996592
245266634087183406872665806889760381001309386579764990048724964738963221717819617344154989358029335242598606
258482886271979512336372192567128762375579458659576623015859412504169897868472480407776078228359019330390396337894419750546221490347367064341292609356574954724999361910
8581441634907591547008070431187355416896430862061426017004532931242187934857665677420830781109732440881362152626050095224400629353683479163583484363183334180680341994619
4297478704948949417916775070707058463431386510010166639403143139933120150221397326187343003780058169034076082282357089701259237489636782777871206039743017294917529323
844395594857144924188970543867647251329316165979400962066060343392885084806194369895458021092276649009426548195850256725668581947607238178953667090614039871709641916724
8279996054318040273347618203624937877760075442739076540487558293356919947869605107919873533807844817172406186083277230558063106361098041739565565283859715044577199965922
45266634087183406872665806889760381001309386579764990048724964738963221717819617344154989358029335242598606
Timur Cravtov

```

Figure 1: RSA encryption and decryption output

5.2 Task 2

Generate keys and encrypt and decrypt your name using ElGamal algorithm.

```

keys: ElGamalSetupData
(p
=3231700607131100730015351347782516336248805713348907517458843413926980683413621000272905636264016468545855635793533081692882902308057347262527355474246140262025279
1657297286270630032526342821314576693141422365422094111134862999165747826803423055308634905063555771221918789033272956969612974385624174123623722519734640269185579776797
6823014625397933058015226858730761197532436467475855460715043896844940366130497697812854295958659597567051283852132784468522925504568272879113720098931873959143374175837
8260002078034973198552060607533234122603254684088120031105907484281003994966956119696956248629032338072839127039, g=2,
d=16974191124709925506848964948843833872912614107073684359310062126544624915483748240655547629948456141242860924706380880167170034372984214577997060303298903521639224
72190181060161348778673800102155070613719128506072813261380340665457910868838770382072051160206487692216977426052794082964994869255924226926530994143813655946605799482574
067717339762304594370621159713583228307118857270232988799411251258179292447831265437392965604644527851688464251035478505290224421830665554951426059974138665893939772870
9942552624201026602319536198853864476952452199869841773445890281805403860717733581718774059322470458150232988)
ciphertext: ElGamalEncryptionData
(kE
=151836156247031772063344308861324331768682300665579713795323100560044464949222508070732081640144618942871903392867811579820129570116654058428700206754281228058294229
3119693697638545732319620083574921638647838696362114133683104367523056023360577922320900021014452258843584989125558625299236583464942499757630087462702502689085884434
1831755561617430473832313074455288627116866651730834908738731060962041584327405360442132888753120819456877926165466713156501142644646999380416646012869029303548083660
952444035250079125014360469684434467163829542621009971273008794043944349957596030858796497437582413672913998063,
y=69453940397462225013477657886127669154805251679948963282209556972010554122604134687216280737123990814538741440075871361179898666794187965683426308035380512374618444453
87181488363465737069759075365974668084575675411694143958650478443859694118267166873865169872643526705891876573999668177678552837868588647696033957394185156491802309333
88988614033240326970135570724520946590084586894572256993165456869788252149150337720714739853848741869037103332456609610084597367984267497968276153712386731645520088128
0373605004742117249335198929311085572777142937941008365922970138801020122310818043455250196475386714409280)
Timur Cravtov
14:14:32: Execution finished 'lab5.Task2Kt.main()'.

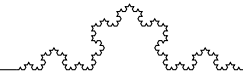
```

Figure 2: ElGamal encryption and decryption output

```

1 fun encryptGamal(message: BigInteger, publicData:
    ElGamalPublicData): ElGamalEncryptionData {
2
3     val (p, g, beta) = publicData
4
5     val i = randomInt(BigInteger.valueOf(2), p-2)
6     val ke = g.modPow(i, p)
7     val km = beta.modPow(i, p)

```

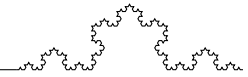


```
8
9     val y = (message * km).mod(p)
10
11     return ElGamalEncryptionData(ke, y)
12
13 }
14
15 fun decryptElGamal(privateData: ElGamalSetupData, encData:
16     ElGamalEncryptionData): BigInteger {
17
18     val (kE, y) = encData
19
20     val d = privateData.d
21     val p = privateData.p
22
23     val km = kE.modPow(d, p)
24     val kminverse = km.modInverse(p)
25     val x = (y * kminverse).mod(p)
26
27     return x
28 }
```

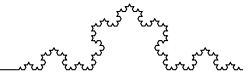
5.3 Task 3

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.randomInt
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.ENCRYPTMODE, 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.DECRYPTMODE, aesKey, ivSpec)
61 val decrypted = cipherDec.doFinal(ciphertext).toString(
62     Charsets.UTF_8)
63
64 println("Decrypted: $decrypted")
65 }

```

```

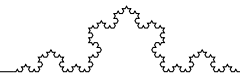
Common key:
258659191764961365420633696905501449999106686082802387998510239965201413119310476073339104642053152826872257794038549340059691247262320139038983953025409231820004949734
5609401764507117140127874489084209108592644822086660426315632899090660870037551664364371613727197346310544635402211793639765343917901728816360644457882569613687366769228
076718936072540550650138303177734575858272537328490608648671682027213932767573051917461890861585113862837809844375444997298004182547874163698341377880599689544015350783
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