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 $d \equiv e^{-1} \mod \phi(n)$; 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 $c \equiv m^e \mod n$.
- Decryption:
- Compute the original message m using the formula $m \equiv c^d \mod n$.
- 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 $y \equiv g^x \mod p$.
- 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 $c_1 \equiv g^k \mod p$.
- Compute $c_2 \equiv m \cdot y^k \mod p$.
- The ciphertext is the pair (c_1, c_2).
- Decryption:
- Compute $s \equiv c_1^x \mod p$.
- Compute the modular inverse of s, denoted as s^{-1}.
- Recover the original message $m \equiv c_2 \cdot s^{-1} \mod p$.
- 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 $A \equiv g^a \mod p$.
  - Bob chooses a private key b such that $1 < b < p-2$ and computes his public key $B \equiv g^b \mod p$.
  - Alice and Bob exchange their public keys A and B over the public channel.
- Shared Secret Computation:
  - Alice computes the shared secret key $s \equiv B^a \mod p$.
  - Bob computes the shared secret key $s \equiv A^b \mod p$.
  - 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.

```
fun keyGen(isPublicKeyDefault: Boolean = true): KeyGenData {
    val p = randomPrime(1024, 2100)
    var q = randomPrime(1024, 2100)

    while (p == q) {
        q = randomPrime(1024, 2100)
    }

    val n = p * q
    val phi = (p - BigInteger.ONE) * (q - BigInteger.ONE)

    val e = if (isPublicKeyDefault)
        BigInteger.valueOf(65537)
    else
```

```kotlin
        generateSequence { randomInt(BigInteger.TWO, phi - BigInteger.ONE) }
            .first { it.gcd(phi) == BigInteger.ONE }

    val d = e.modInverse(phi)

    return KeyGenData(e, d, n)
}
```

```kotlin
fun encryptRsa(message: BigInteger, publicKey: BigInteger, n: BigInteger): BigInteger {

    require(message.bitLength() <= 2048)

    val cipherText = message.modPow(publicKey, n)

    return cipherText

}
fun decryptRsa(encryptedMessage: BigInteger, privateKey: BigInteger, n: BigInteger): String {

    val message = encryptedMessage.modPow(privateKey, n)

    return bigIntToAsciiString(message)
}
```

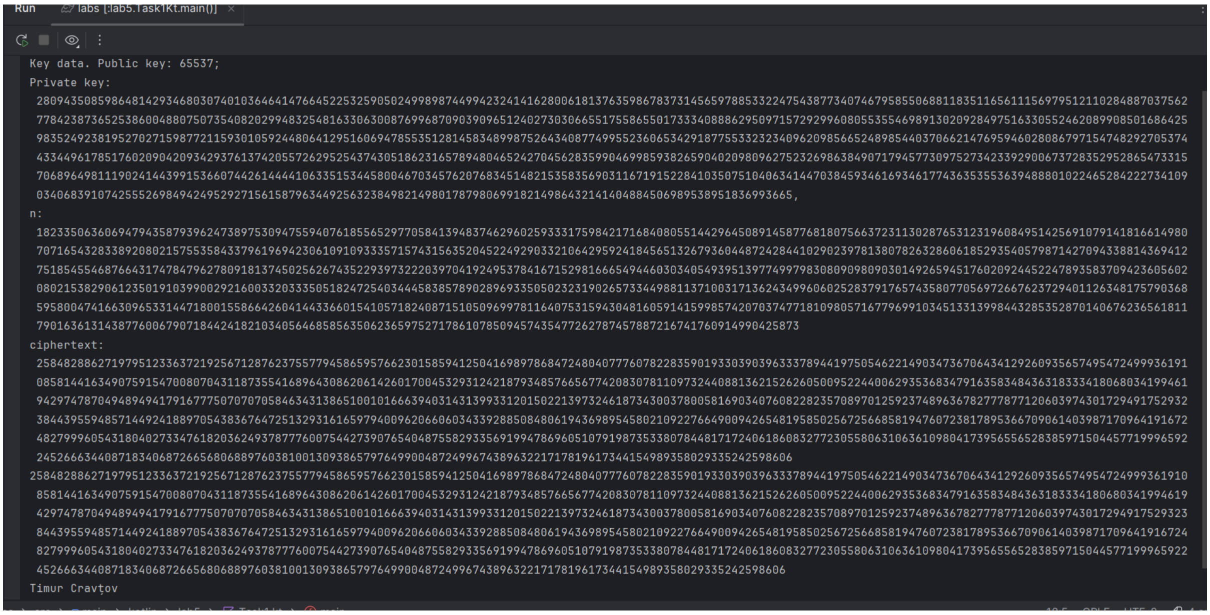

Figure 1: RSA encryption and decryption output

5.2 Task 2

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

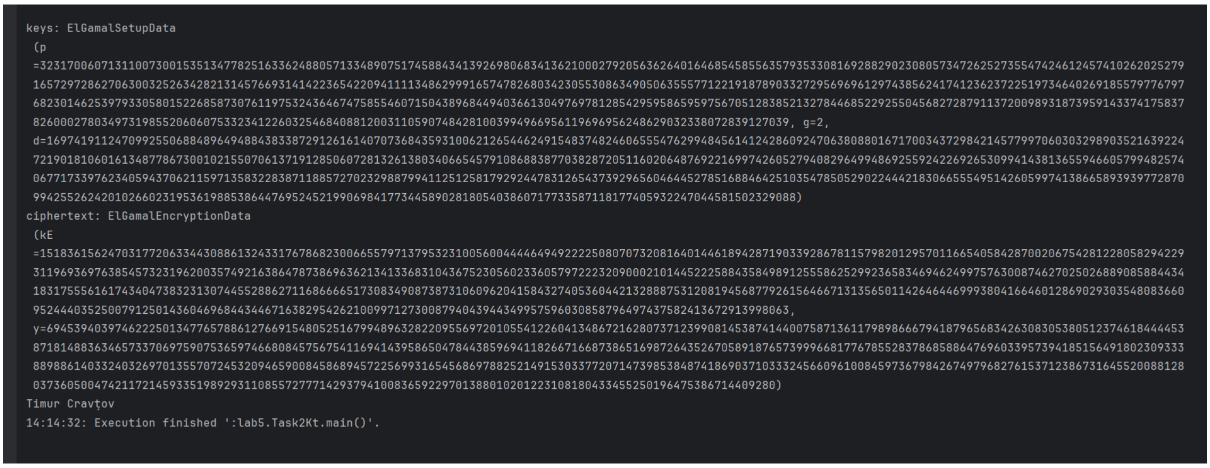

Figure 2: ElGamal encryption and decryption output

```
fun encryptGamal(message: BigInteger, publicData:
    ElGamalPublicData): ElGamalEncryptionData {

    val (p, g, beta) = publicData

    val i = randomInt(BigInteger.valueOf(2), p-2)
    val ke = g.modPow(i, p)
    val km = beta.modPow(i, p)
```

```

    val y = (message * km).mod(p)

    return ElGamalEncryptionData(ke, y)

}

fun decryptElGamal(privateData: ElGamalSetupData, encData:
    ElGamalEncryptionData): BigInteger {

    val (kE, y) = encData

    val d = privateData.d
    val p = privateData.p

    val km = kE.modPow(d, p)
    val kminverse = km.modInverse(p)
    val x = (y * kminverse).mod(p)

    return x
}
```

5.3 Task 3

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

```
package lab5

import lab5.algorithms.randomInt
import java.math.BigInteger
import javax.crypto.Cipher
import javax.crypto.spec.IvParameterSpec
import javax.crypto.spec.SecretKeySpec
import kotlin.random.Random

fun main() {

    val p = BigInteger("7919")
```

```
    val alpha = BigInteger("3")

    // alice private key
    val a = randomInt(BigInteger.ONE, p - BigInteger.ONE)

    // bob private key
    val b = randomInt(BigInteger.ONE, p - BigInteger.ONE)

    // alice public key
    val A = alpha.modPow(a, p)      // alpha^a mod p

    // bob public key
    val B = alpha.modPow(b, p)      // alpha^b mod p

    // alice computed common key
    val kA = B.modPow(a, p)         // (alpha^b)^a mod p

    // bob computed common key
    val kB = A.modPow(b, p)         // (alpha^a)^b mod p

    require(kA == kB)

    // use common key as key for aes
    val raw = kA.toByteArray()
    val secretBytes =
        if (raw.size >= 32) raw.copyOfRange(raw.size - 32, raw.size)
        else ByteArray(32 - raw.size) + raw

    val aesKey = SecretKeySpec(secretBytes, "AES")

    // random iv
    val iv = ByteArray(16).also { Random.nextBytes(it) }
    val ivSpec = IvParameterSpec(iv)

    val plaintext = "Some text which alice encrypted"
    println("plaintext: $plaintext")

```

```kotlin
    // Encrypt
    val cipherEnc = Cipher.getInstance("AES/CBC/PKCS5Padding")
    cipherEnc.init(Cipher.ENCRYPT_MODE, aesKey, ivSpec)
    val ciphertext = cipherEnc.doFinal(plaintext.toByteArray())

    println("Ciphertext (hex): " + ciphertext.joinToString("") {
        "%02x".format(it) })

    // decrypt
    val cipherDec = Cipher.getInstance("AES/CBC/PKCS5Padding")
    cipherDec.init(Cipher.DECRYPT_MODE, aesKey, ivSpec)
    val decrypted = cipherDec.doFinal(ciphertext).toString(
        Charsets.UTF_8)

    println("Decrypted: $decrypted")
}
```

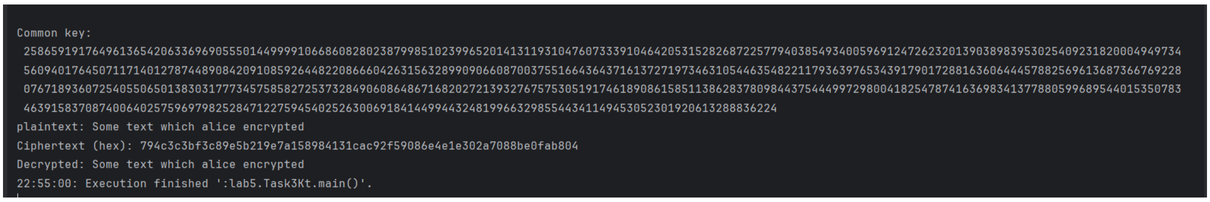

Figure 3: Diffie-Hellman key exchange output

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