



Міністерство освіти і науки України
Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Фізико-технічний інститут

ЛАБОРАТОРНА РОБОТА №5

з дисципліни

«Криптографія»

**на тему: «Вивчення криптосистеми RSA та алгоритму електронного підпису;
ознайомлення з методами генерації параметрів для асиметричних криптосистем»**

Виконали:

студенти 3 курсу ФТІ

групи ФБ-71

Бабенко Іван та Гончаренко Дарина

Перевірили:

Чорний О.

Савчук М. М.

Завадська Л. О.

Мета роботи :

Ознайомлення з тестами перевірки чисел на простоту і методами генерації ключів для асиметричної криптосистеми типу RSA; практичне ознайомлення з системою захисту інформації на основі криптосхеми RSA, організація з використанням цієї системи засекреченого зв'язку й електронного підпису, вивчення протоколу розсилання ключів.

Порядок виконання роботи

1. Написати функцію пошуку випадкового простого числа з заданого інтервалу або заданої довжини, використовуючи датчик випадкових чисел та тести перевірки на простоту. В якості датчика випадкових чисел використовуйте вбудований генератор псевдовипадкових чисел вашої мови програмування. В якості тесту перевірки на простоту рекомендовано використовувати тест Міллера-Рабіна із попередніми пробними діленнями. Тести необхідно реалізовувати власноруч, використання готових реалізацій тестів не дозволяється.
2. За допомогою цієї функції згенерувати дві пари простих чисел p, q і $1 < p, q$ довжини щонайменше 256 біт. При цьому пари чисел беруться так, щоб $pq \leq p_1q_1$; p і q – прості числа для побудови ключів абонента A, $1 < p < q_1$ – абонента B.
3. Написати функцію генерації ключових пар для RSA. Після генерування функція повинна повертати та/або зберігати секретний ключ (d, p, q) та відкритий ключ (n, e) . За допомогою цієї функції побудувати схеми RSA для абонентів A і B – тобто, створити та зберегти для подальшого використання відкриті ключі (e, n) , (n, d) та секретні d і d_1 .
4. Написати програму шифрування, розшифрування і створення повідомлення з цифровим підписом для абонентів A і B. Кожна з операцій (шифрування, розшифрування, створення цифрового підпису, перевірка цифрового підпису) повинна бути реалізована окремою процедурою, на вхід до якої повинні подаватись лише ті ключові дані, які необхідні для її виконання. За допомогою датчика випадкових чисел вибрати відкрите повідомлення M і знайти криптограму для абонентів A і B, перевірити правильність розшифрування. Скласти для A і B повідомлення з цифровим підписом і перевірити його.
5. За допомогою раніше написаних на попередніх етапах програм організувати роботу протоколу конфіденційного розсилання ключів з підтвердженням справжності по відкритому каналу за допомогою алгоритму RSA. Протоколи роботи кожного учасника (відправника та приймаючого) повинні бути реалізовані у вигляді окремих процедур, на вхід до яких повинні подаватись лише ті ключові дані, які необхідні для виконання. Перевірити роботу програм для випадково обраного ключа $0 < k < n$.

Кожна з наведених операцій повинна бути реалізована у вигляді окремої процедури, інтерфейс якої повинен приймати лише ті дані, які необхідні для її роботи; наприклад, функція `Encrypt()`, яка шифрує повідомлення для абонента, повинна приймати на вхід повідомлення та відкритий ключ адресата (і тільки його), повертаючи в якості результату шифротекст. Відповідно, програмний код повинен містити сім високорівневих процедур: `GenerateKeyPair()`, `Encrypt()`, `Decrypt()`, `Sign()`, `Verify()`, `SendKey()`, `ReceiveKey()`.

Труднощі та етапи розробки програмного коду

Перш за все було розроблено функцію перевірки числа на простоту. Для цього був використаний тест Мілера-Рабіна. Першою перешкодою була необхідність швидко піднести число в степінь по модулю, так як звичайна математична операція піднесення числа в степінь довго працює з великими числами. Тому ми створили функцію за схемою Горнера, яка вирішує це питання.

Далі ми отримали пари простих чисел p_1, q_1, p, q та написали функції за шифрування, розшифрування, підписання ЦП, його перевірки, та функції відправки та отримання ключів. На цих етапах проблем не було.

Хід роботи:

Первый набор:

n: 875e47792ff13fea078d2daea19bfa1a738f407f76635de5ce4f23c8370beb6a176f3539ef1a026e45d770beffeb34721a27fbf3f65342294db018c00e8ae1cd

e: 483dbf7a6f8464c4313e409cf19c6de8c3a940b3d7aa7968d5ce568cc5d6095e836bcca1d5abd5ca21b528fd7a02cc5649b9719183d29bb8f220367599da27

d: 75f93bc02e65272e768508beb575909156e719545a7ec5479b6cb88879cf624b5e74c53b575211f82672d58b14cfc209612ea90d365d20540a02b60f08a204f7q:
77411323652058046381906949344302699013880408630970156912557360290829216614613

p: 77411323652058046381906949344302699013880408630970156912557360290829216614613

q: 91586239165236362813175585875842432664712139524423315035682526413095299510553

c: 26f3df01896b46792e422554d88e938424e3f7be1b40b01d8377ebb99858426d2946f9a9742c30d8cd1b3836c5033ffa80cdc81776d6023c4437e99df97b4e3a

m: 9c2178fb

s: 4564c17b9756672c7666aa233021c43b98b1733d9665a8acd46c326448ac1fb97c148d8085a01a9f2edd3274963b1274ea9ec13c77a6ea57f00116c9d54b456d

Второй набор

n1: 9516534687edb77c7ce6c382fd3c6068040b26c8d78a095c79a894b4a9df982f357f64a5a8f13aae83f856dcecc84506858f23be3a85b10e29da3a6a04db0cccd

e1: 18a0e3037e5ec8c75ef34a166d7aa609ec292cf3586e3a915dc3dc3bf5ef06aed07ada7e24b1622ab5a564968888e346cb20ef08c7abe3817c3d8198b88de0a5

d1: 2c583aada0e061d74fda5a993be3ce1c4f282a510e6ea54594e13592e16a2c24efe3eba8bd0be302d6efbe572c1cd63cfa40503e7060865d8b161e04669422d

q1: 82342592552581730173566128439722272527001990049004456868839717794276970230977

p1: 94827360061771511511419892304651598008267233833681659609119322294818006474509

c1:43fca23ae28a7209af5fbea8b17c12e7679c5a216f4ec7d51f71c27738d6d808658d0e79cab2166f66db52a85279256d09df3c9c5e5d1735b22a1620c5abeaa6

m1: 89ebeccb

s1:211a96013b6d3296c4de3d0050ea8d6c50f95815cff5be77e2bebe22c7e14ce39f66f39b0c409de1278a561462153a2cc41e5a9dbc5a255c5fdbe583a76af208

k:4afdfae47bb7104f5e87fa36c2242dca542bbc9974c02e40bae16e6945ef8341b76cfad11c567dd257f1cb60a821d8c0c9d4ab3810034e3ea6d5d117eeca079

S:1ff65f937796d6141f70621076986e0377c5b5fad5008c9a91fbf3d30500e838455acfbbe5214652de436230a58a642ba05a410cd6f4ae2c31ecaa10bed84b14

RSA Testing Environment

Server Key

Encryption

Decryption

Signature

Verification

Send Key

Receive Key

Verify

Clear

Message

9c2178fb

Bytes

Signature

4564c17b9756672c7666aa233021c43b98b1733d9665a8acd46c326448ac1fb97c148d8085a01a9f2edd3274963b

Modulus

875e47792ff13fea078d2daea19bfa1a738f407f76635de5ce4f23c8370beb6a176f3539ef1a026e45d770beffb3472'

Public exponent

483dbf7a6f8464c4313e409cf19c6de8c3a940b3d7aa7968d5ce568cc5d6095e836bcca1d5abd5ca21b528fd7a02c

Verify

Verification

true

Код:

```
import AppKit
import Darwin.C.math
import BigInt

// ----- SOURCE ----- //

func dataWithHexString(hex: String) -> Data {
    var hex = hex
```

```
var data = Data()
while(hex.count > 0) {
    let subIndex = hex.index(hex.startIndex, offsetBy: 2)
    let c = String(hex[..
```

```

        data.append(&char, count: 1)
    }

    return data
}

//let keyn1 = "90DECD167D2F5F2C6394A5F9EEC60B2A0CE4C105ABC2E3F542D31
2596DE4DF55"

//let key_n1 : BigUInt = BigUInt(keyn1, radix:16)!

//let keye1 = "10001"

//let key_e1 : BigUInt = BigUInt(keye1, radix:16)!

let keyn = "875e47792ff13fea078d2daea19bfa1a738f407f76635de5ce4f23c8370beb6a176
f3539ef1a026e45d770beffeb34721a27fb3f65342294db018c00e8ae1cd"

let key_n : BigUInt = BigUInt(keyn, radix:16)!

let keye = "483dbf7a6f8464c4313e409cf19c6de8c3a940b3d7aa7968d5ce568cc5d6095e83
6bcc1d5abd5ca21b528fd7a02cc5649b9719183d29bb8f220367599da27"

let key_e : BigUInt = BigUInt(keye, radix:16)!

let keyd = "75f93bc02e65272e768508beb575909156e719545a7ec5479b6cb88879cf624b5
e74c53b575211f82672d58b14cfc209612ea90d365d20540a02b60f08a204f7"

let key_d : BigUInt = BigUInt(keyd, radix:16)!

let keyn1 = "9516534687edb77c7ce6c382fd3c6068040b26c8d78a095c79a894b4a9df982f
357f64a5a8f13aae83f856dccc84506858f23be3a5b10e29da3a6a04db0cccd"

let key_n1 : BigUInt = BigUInt(keyn, radix:16)!

let keye1 = "2c583aada0e061d74fda5a993be3ce1c4f282a510e6ea54594e13592e16a2c24e
fe3eba8bd0be302d6efbe572c1cd63cfa40503e7060865d8b161e04669422d"

let key_e1 : BigUInt = BigUInt(keye, radix:16)!

let keyd1 = "4f34d55ef421e2348eadd3d298d81d58986b2ddcafbfeed6d63fb01cc2873fb7c
74917d2f489ec9d431b8bc66ffe3d18c239fb07e3844753960fe5aedf677eaf"

let key_d1 : BigUInt = BigUInt(keyd, radix:16)!

print("Сообщение сгенерировано: ")

let M = BigUInt.randomInteger(withExactWidth: 32)

let M_hex = String(M, radix:16)

print(M_hex)

// ----- TEXT EDIT -----
- //

func gorner (num : BigUInt, power : BigUInt , mod : BigUInt) -> BigUInt{

    var a : BigUInt = 0

    var power = power

    var to_binary = [BigUInt]()

    var x2 = num.power(2, modulus: mod)

    repeat {

        a = power%2

        power = power/2

        to_binary.append(a)

        //to_binary.insert(a, at: 0)

    } while power >= 1

    var result = num.power(Int(to_binary[0]))

    let length = to_binary.count

    for i in 1...(length-1){

        result = (result * x2.power(to_binary[i], modulus: mod))

        x2 = x2.power(2, modulus: mod)

```

```

    }

    //print(result)

    //print(length)

    return a
}

func prime_check(p: BigUInt, k: Int) -> Bool{

    var flag : Bool = true

    if (p%2 == 0){flag = false}

    var s : Int = -1

    var d : BigUInt = 0

    var num : BigUInt = p-1

    while (d%2 != 1){

        d = num

        num = num/2

        s+=1

    }

    if (s<=1){return false}

    for var i in 0...k{

        let x = BigUInt.randomInteger(lessThan: p)

        if (p.greatestCommonDivisor(with: x) != 1){flag = false; break}

        else {

            var xd = x.power(d, modulus: p)

            if (xd == 1) || (xd == p-1) {i += 1; continue}

            else {

                for var j in 1...(s-1){

                    xd = xd.power(2, modulus: p)

                    if (xd == 1){flag = false; return flag}

                    else if (xd == (p-1)){i += 1; break}

                }

                j+=1

                if (j==s){flag = false; return flag}

            }

            i+=1

        }

    }

    return flag
}

func generate(_ width: Int) -> BigUInt{

    while true {

        var random = BigUInt.randomInteger(withExactWidth: width)

        random |= BigUInt(1)

        if (prime_check(p: random, k: 9) == true) {

            print(random)

            return random

        }

        else{print("Число \{random\} вероятно составное!")}

```

```

    }
}

func GenKeys() -> (n: BigUInt, e: BigUInt, d: BigUInt){

    let p = generate(256)

    let q = generate(256)

    let n = p*q

    //let n1 = p1*q1

    let fin=(p-1)*(q-1)

    var e : BigUInt = 0

    var d : BigUInt = 0

    while (d == 0){

        e = BigUInt.randomInteger(lessThan: fin)

        var result : BigUInt = e.greatestCommonDivisor(with: fin)

        while ((result != 1) && (e<2)){

            e = BigUInt.randomInteger(lessThan: fin)

            result = e.greatestCommonDivisor(with: fin)

        }

        d = e.inverse(fin) ?? 0

    }

    return (n, e, d)

}

func Encrypt(message: BigUInt, e: BigUInt, n : BigUInt) -> BigUInt{

    //let c : BigUInt = gorner(num: message, power: e, mod: n)

    let c : BigUInt = message.power(e, modulus: n)

    return c

}

func Decrypt(message: BigUInt, d: BigUInt, n : BigUInt) -> BigUInt{

    //let c : BigUInt = gorner(num: message, power: e, mod: n)

    let m : BigUInt = message.power(d, modulus: n)

    return m

}

func Sign(message: BigUInt, d: BigUInt, n : BigUInt) -> BigUInt{

    //let c : BigUInt = gorner(num: message, power: e, mod: n)

    let m : BigUInt = message.power(d, modulus: n)

    return m

}

func Verify(message: BigUInt, signature: BigUInt, e: BigUInt, n : BigUInt) -> Bool{

    if (signature.power(e, modulus: n) == message){return true}

    else{return false}

}

func SendKey(e1: BigUInt, n1 : BigUInt) -> (k1: BigUInt, S1: BigUInt){

    let k = BigUInt.randomInteger(lessThan: key_n)

```

```

    let S : BigUInt = k.power(key_d, modulus: key_n)

    let S_hex = String(S, radix:16)

    print("S: \S_hex")

    let S1 : BigUInt = S.power(e1, modulus: n1)

    let k1 : BigUInt = k.power(e1, modulus: n1)

    return (k1, S1)

}

func RecieveKey(k1: BigUInt, S1: BigUInt, n: BigUInt, e: BigUInt) -> Bool{

    let S : BigUInt = S1.power(key_d1, modulus: key_n1)

    let S_hex = String(S, radix:16)

    let k : BigUInt = k1.power(key_d1, modulus: key_n1)

    // let S_hex = String(S, radix:16)

    let Se : BigUInt = S.power(e, modulus: n)

    let k_hex = String(k, radix:16)

    let Se_hex = String(Se, radix:16)

    // print("S: \S_hex")

    print("k: \k_hex")

    print("S^e: \Se_hex")

    print("S: \S_hex")

    if ((S.power(e, modulus: n)) == k)

    {return true}

    else{return false}

}

// ----- MAIN -----
- //

//print("Выполняется генерация ключей, ожидайте...")

//let n = GenKeys()

//let result = n.n

//print("Первый набор")

//let npr = String(n.n, radix:16)

//let epr = String(n.e, radix:16)

//let dpr = String(n.d, radix:16)

//print("n: \npr")

//print("e: \epr")

//print("d: \dpr")

//var n1 = GenKeys()

//var result1 = n1.n

//while (result1<result) {

//    n1 = GenKeys()

//    result1 = n1.n

//}

//print("Второй набор")

//let npr1 = String(n1.n, radix:16)

//let epr1 = String(n1.e, radix:16)

//let dpr1 = String(n1.d, radix:16)

//print("n1: \npr1")

//print("e1: \epr1")

```

```

//print("d1: \dpr1")

//print("\n")

//

print("\nВыберите действие:\n1 - Зашифровать\n2 - Расшифровать\n3 - Подписать\n4 - Проверить цифровую подпись\n5 - Отправить\n6 - Получить")

let answer = readLine()!

switch Int(answer) {

case 1:

    print("Введите сообщение: ")

    let mess_string = readLine()!

    let data = Data(mess_string.utf8)

    let hexString = data.map{ String(format:"%02x", $0) }.joined()

    print(hexString)

    let message : BigUInt = BigUInt(hexString, radix:16)!

    print(message)

    let methodStart = Date()

    print("Зашифрованное сообщение: ")

//    let encr_mess : BigUInt = Encrypt(message: message, e: n.e, n: n.n)

    let encr_mess : BigUInt = Encrypt(message: message, e: key_e, n: key_n)

    print(String(encr_mess, radix:16))

    let methodFinish = Date()

        let executionTime = methodFinish.timeIntervalSince(methodStart)

        print("Execution time: \(executionTime)")

case 2:

    print("Введите зашифрованное сообщение: ")

    let mess_string = readLine()!

    let message : BigUInt = BigUInt(mess_string, radix:16)!

    let methodStart = Date()

    print("Расшифрованное сообщение: ")

//let encr_mess : BigUInt = Decrypt(message: message, d: n.d, n: n.n)

    let encr_mess : BigUInt = Decrypt(message: message, d: key_d, n: key_n)

    let hex = String(encr_mess, radix:16)

    let data = dataWithHexString(hex: hex)

    let emess_data = String(data: data, encoding: .utf8)

    let encrypted_message : String = emess_data!

    print(encrypted_message)

    let methodFinish = Date()

        let executionTime = methodFinish.timeIntervalSince(methodStart)

        print("Execution time: \(executionTime)")

case 3:

    let methodStart = Date()

    print("Сообщение подписано: ")

    let signed_m = Sign(message: M, d: key_d, n: key_n)

    let signed_m_hex = String(signed_m, radix:16)

    print(signed_m_hex)

```

```

print("Verify: ")

print(Verify(message: M, signature: signed_m, e: key_e, n: key_n))

let methodFinish = Date()

let executionTime = methodFinish.timeIntervalSince(methodStart)

print("Execution time: \(executionTime)")

case 4:

    let methodStart = Date()

    let mess_hex = "9954cf27"

    print("Исходное сообщение: \mess_hex)")

    let signed_hex = "1c668f7d0536da18f01eadbdcc462658fc3dc0f9a736136a2f8e156954456c3a7cf8f6d790110c15a4076c726895d455aa97dbeaa76ed9654dcd81ac34aa36c"

    print("Подпись: \signed_hex)")

    let signed : BigUInt = BigUInt(signed_hex, radix:16)!

    let message : BigUInt = BigUInt(keyd, radix:16)!

    print("Verify: ")

    print(Verify(message: message, signature: signed, e: key_e, n: key_n))

    let methodFinish = Date()

        let executionTime = methodFinish.timeIntervalSince(methodStart)

        print("Execution time: \(executionTime)")

case 5:

    let methodStart = Date()

    print("N1 и E1 отправлены...")

    let result = SendKey(e1: key_e1, n1: key_n1)

    let k1_hex = String(result.k1, radix:16)

    let S1_hex = String(result.S1, radix:16)

    print("k1: \k1_hex)")

    print("S1: \S1_hex)")

    let methodFinish = Date()

        let executionTime = methodFinish.timeIntervalSince(methodStart)

        print("Execution time: \(executionTime)")

case 6:

    let methodStart = Date()

    let k1_hex = "54ee2d93d975b94735b712aadd383241f7bae40dc251724ad38bcb35752f8ebc0d31a0a7a6d55935409a223922de61e0fbb71c8ed969e4aac880da2bfb0c5b8"

    let S1_hex = "4afdfae47bb7104f5e87fa36c2242dca542bbe9974c02e40bae16e6945ef8341b76cfad11c567dd257f1cb60a821d8c0c9d4ab3810034c3ea6d5d117eacea079"

    let k1 : BigUInt = BigUInt(k1_hex, radix:16)!

    let S1 : BigUInt = BigUInt(S1_hex, radix:16)!

    print("K1 и S1 отправлены...")

    let result = RecieveKey(k1: k1, S1: S1, n: key_n, e: key_e)

    print("Verify: \result)")

    let methodFinish = Date()

    let executionTime = methodFinish.timeIntervalSince(methodStart)

    print("Execution time: \(executionTime)")

default:

    print("Упс, что-то пошло не так")

}

```

Висновок:

У ході комп'ютерного практикуму було набуто навичок роботи з числами великої розрядності, написання тестів перевірки чисел на простоту та методів генерації ключів для асиметричної криптосистеми RSA. Набуто навичок побудови цифрового підпису на основі криптосистеми RSA.