

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

Криптографія  
Комп’ютерний практикум №4

Підготували:  
студенти групи ФБ-02  
Єсаф'єв Євгеній  
Сапегін Валентин

Київ – 2022

## 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.

## Постановка задачі:

- 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.
- За допомогою цієї функції згенерувати дві пари простих чисел  $p, q$  і  $p_1, q_1$  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 А,  $p_1, q_1$  – абонента В.
- 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 А і В – тобто, створити та зберегти для подальшого використання відкриті ключі  $(e, n)$ ,  $(e_1, n_1)$  та секретні  $d$  і  $d_1$ .
- Написати програму шифрування, розшифрування і створення повідомлення з цифровим підписом для абонентів А і В. 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 А і В, перевірити правильність розшифрування. Скласти для А і В повідомлення з цифровим підписом і перевірити його.
- 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  $0 < k < n$ .

Кожну операцію рекомендується перевіряти шляхом взаємодії із тестовим середовищем, розташованим за адресою

<http://asymcryptwebservice.appspot.com/?section=rsa>

## Хід роботи:

Генеруємо  $p$ ,  $q$ ,  $p1$ ,  $q1$  (при чому  $pq \leq p1q1$ ):

```
q: 75377582710393124700210720757628726523704117994094646238195983202757295736371
p: 73663677631378435941766617475144322848313903124345113882745543388735929177389
q1: 84744505678031382420452358536678624978186159377862123837156406986108288771311
p1: 82318779325839488747096458804955644683490189815989735038807809648674072920169
```

Для юзера Lil та Big:

Generated message: 1072612107

Encrypted message from Lil to Big:

49358891473056675278664951210516704514626618478949693783502612148659570654929746111506259782380157334314504522057967431943386364735154061890783137

Signature:

41870980194993116384629092635664182706404426101128329288272328358052093199864468221282351995746494260724606157562330386803419830448409247270316964

Big decrypted message: 1072612107

Signature verify: True

## Пересилання:

Generated key: 1050099265

Success

Все працює, переконаємось в цьому перевіркою на сайті:

Зашифруємо нашим кодом:

```
# Server modulus here
n_hex = 'A5F15327E0D32F463F7E98381FC735B66BE14229F9A86A5D0D0E204DB2AA04AD'
e_hex = '10001'

mes = 'ABCD'

me.hex_encrypt(mes, e_hex, n_hex)
```

✓ 0.8s

'1710FC739005A706CB30DA7E31DD53415AB1A4F5945E5857EFC9CAF420627DBF'

Розшифруємо на сайті:

✖ Clear

Ciphertext

I0FC739005A706CB30DA7E31DD53415AB1A4F5945E5857EFC9CAF420627DBF

Bytes

Decrypt

Message

ABCD

Робимо навпаки:

✖ Clear

Modulus

3C7EDF816E4B1971394C73122460C6A671FD81841ABFBC616C84B431B99B212628A3A33C86183E213D4601

Public exponent

A2BBEDB0166CF6C861B5364684E13229B62FE3C56A78E4F7917379053FC8615E98EB9B5CE06127AC04DE1

Message

ABCD

Bytes

Encrypt

Ciphertext

22E8067A0755638046B78F87E4BBD9F99F864F22CC99E593681FA575B76F794E695558B85E80BB0867AA76E3

```
me.hex_decrypt('22E8067A0755638046B78F87E4BBD9F99F864F22CC99E593681FA575B76F794E695558B85E80BB0867AA76E3')
```

✓ 0.5s

'ABCD'

Отримуємо підпис:

```
me.hex_sign(msg)
```

✓ 0.9s

'568F88F6FB24193F901D3DD46E0AC30ABC81483D34F5C78974DAF672B6A07A21A967C0A582BE8F5CF4C6F33A963709FB81AC444DE5F66FE0D2518C270EC440F5'

Та перевіряємо:

## Verify

✖ Clear

Message

ABCD

Bytes

▼

Signature

5C78974DAF672B6A07A21A967C0A582BE8F5CF4C6F33A963709FB81AC444DE5F66FE0D2518C270EC440F5

Modulus

2C7EDF816E4B1971394C73122460C6A671FD81841ABFBC616C84B431B99B212628A3A33C86183E213D4601

Public exponent

42BBEDB0166CF6C861B5364684E13229B62FE3C56A78E4F7917379053FC8615E9BEB9B5CE06127AC04DE1

Verify

Verification

true

✓

Та навпаки:

## Sign

✖ Clear

Message

ABCD

Bytes

▼

Sign

Signature

5E1E10C85F4D6DDBB5060D0F440B2F6F47E1B6A7F0AA917BB681FEC534973D83

me.hex\_verify(mes, '5E1E10C85F4D6DDBB5060D0F440B2F6F47E1B6A7F0AA917BB681FEC534973D83', '10001', '10001')

✓ 0.1s

True

Отримуємо згенерований з код ключ на сайті:

```
# Server modulus here
n_hex = '839439A8E3FC1E38EDF9EA6E3E80C246A9552028F4450800B08EE6A83856CD56157EBF04D941AF421B6F6CBA5FF4C8D88AFF5C6C10C5E5C74E39A39A4A9EE
e_hex = '10001'

key = 'ABCD'

me.hex_regenerate_keys(n_hex, range_min, range_max)
me.print_public()
me.hex_send_key(key, e_hex, n_hex)
```

✓ 0.2s

Modulus:  
69B6B7102D96F1EBAF1FF91C4A93D1610EB9A6D8F785AA0034C1E33324528595437AA4C36B2FA4B98CC5C975B46890CEB916EDB46DF10A846BD605CFAE819A05

E:  
3E9DF0E16111DDA7E9EA595A90F74D744E383B55971A87B6AE794A35637208FF07BAD1FCAD38EEB86FE2FD85E6AC9CFADEEC8B17CF3DD37B7ECBBFE4446FF893

('7EAC94FFE663B486EB48FE200954847BF194849F6A6142DC58CC318F0CFA04E5597D52CA9BAF6FEA72EAD092C8C3357CAC57FC9C6856A399161FCB222CBF87E6',  
'21898545AA18BB49D2D53357616AB87F17A8F75112D037F901840D54AA0FEC2D53DFB415E419AE49FB8734933A587F74612383701D61FF01AD329AC40E0D6623')

## Receive key

✖ Clear

Key	2DC58CC318F0CFA04E5597D52CA9BAF6FEA72EAD092C8C3357CAC57FC9C6856A399161FCB222CBF87E6
Signature	0037F901840D54AA0FEC2D53DFB415E419AE49FB8734933A587F74612383701D61FF01AD329AC40E0D6623
Modulus	5AA0034C1E33324528595437AA4C36B2FA4B98CC5C975B46890CEB916EDB46DF10A846BD605CFAE819A05
Public exponent	'B6AE794A35637208FF07BAD1FCAD38EEB86FE2FD85E6AC9CFADEEC8B17CF3DD37B7ECBBFE4446FF893

Receive

Key	ABCD
Verification	true ✓

І навпаки:

## Send key

✖ Clear

Modulus

ECD8360F3D5948DCD49FEBA0A089FF365601A5AFAED2C12069391517E6E456C4965780898432F01E5FEDB

Public exponent

1492A4DBFDD4030E283DF89857C85C1797BBD60BCE4746361AF2772AF1D0B4AABA7AE14CF1B8A1AD043F

Send

Key

4AE352FA394198BFA9AFBA4A6069AC8ED5CAA9690776758A16611394D2A60BBF25FB7A744D3CA85F30D3C

Signature

4E1CDF2EFD8B1146E93AF5460896F4DF279E664FB289A4034FD3C7538F7ADE2D1DD44F1A0E4EEDB2DC44

Все збігається.

### Висновок:

Під час виконання практикума ми розібрали перевірку простоти чисел за допомогою тесту Міллера-Рабіна. Реалізували принцип роботи криптосистеми типу RSA, навчилися шифрувати, розшифровувати, використовувати цифрові підписи, яки генерували з використанням даної системи, організували захищений зв'язок.