

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

«Криптографія»

Комп'ютерний практикум

№ 4

Виконав:

студент групи **ФБ-83**

Гах Валерій

Перевірів:

Київ 2020

Назва: Вивчення криптосистеми RSA та алгоритму електронного підпису; ознайомлення з методами генерації параметрів для асиметричних криптосистем;

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:

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 і довжини щонайменше 256 біт. При цьому пари чисел беруться так, щоб ; p і q – прості числа для побудови ключів абонента A , $p1$, $q1$ – абонента B .

3. Написати функцію генерації ключових пар для RSA. Після генерування функція повинна повертати та/або зберігати секретний ключ та відкритий ключ . За допомогою цієї функції побудувати схеми RSA для абонентів A і B – тобто, створити та зберегти для подальшого використання відкриті та секретні ключі.

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 и 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 .

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().

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

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

Наприклад, для перевірки коректності операції шифрування необхідно а) зашифрувати власною реалізацією повідомлення для серверу та розшифрувати його на сервері, б) зашифрувати на сервері повідомлення для вашої реалізації та розшифрувати його локально.

Варіант: 6(номер у списку групи), номер бригади відсутній(робота виконана самостійно);

Характеристики обладнання:

- Ноутбук - Lenovo G510;
- ОС - Windows 10 Home x64;
- Процесор - Intel Core i5-4200M, CPU - 2.5GHz;
- Тип системи: 64-розрядна ОС, процесор x64;
- ОЗУ - 6.00 ГБ;

Хід роботи:

Програмний код-реалізацію криптоаналізу було написано мовою python. При цьому окремі, математичні операції(НСД, визначення оберненого за модулем) винесено в окремі файли – `Math_operations.py`, `GCD_and_Linear_equations.py`. У файлі `Miller_Rabin.py` містяться функції перевірки простоти пробними діленнями(на малі прості числа), тест Міллера-Рабіна та функція генерації випадкового простого числа заданої довжини в бітах. `RSA.py` – файл з функціями RSA: GenerateKeyPair(), Encrypt(), Decrypt(), Sign(), Verify(), SendKey(), ReceiveKey(). `main.py` – код, що у інтерактивній формі виконує операції криптосистеми RSA та видає чисельні значення для обміну між користувачами-учасниками системи(А-В, або А- сайт).

Результати:

Запустимо на виконання main.py, обмін відбувається між користувачем А та сайтом, В генерується просто за вимогою лабораторної.

Відкритий ключ сайту:

Project 01: RSA with JavaScript

01: RSA with JavaScript

02: RSA with Python

03: RSA with Java

04: RSA with C++

05: RSA with Go

06: RSA with Rust

07: RSA with Kotlin

08: RSA with Swift

09: RSA with Kotlin

10: RSA with Kotlin

11: RSA with Kotlin

12: RSA with Kotlin

13: RSA with Kotlin

14: RSA with Kotlin

15: RSA with Kotlin

16: RSA with Kotlin

17: RSA with Kotlin

18: RSA with Kotlin

19: RSA with Kotlin

20: RSA with Kotlin

21: RSA with Kotlin

22: RSA with Kotlin

23: RSA with Kotlin

24: RSA with Kotlin

25: RSA with Kotlin

26: RSA with Kotlin

27: RSA with Kotlin

28: RSA with Kotlin

29: RSA with Kotlin

30: RSA with Kotlin

31: RSA with Kotlin

32: RSA with Kotlin

33: RSA with Kotlin

34: RSA with Kotlin

35: RSA with Kotlin

36: RSA with Kotlin

37: RSA with Kotlin

38: RSA with Kotlin

39: RSA with Kotlin

40: RSA with Kotlin

41: RSA with Kotlin

42: RSA with Kotlin

43: RSA with Kotlin

44: RSA with Kotlin

45: RSA with Kotlin

46: RSA with Kotlin

47: RSA with Kotlin

48: RSA with Kotlin

49: RSA with Kotlin

50: RSA with Kotlin

51: RSA with Kotlin

52: RSA with Kotlin

53: RSA with Kotlin

54: RSA with Kotlin

55: RSA with Kotlin

56: RSA with Kotlin

57: RSA with Kotlin

58: RSA with Kotlin

59: RSA with Kotlin

60: RSA with Kotlin

61: RSA with Kotlin

62: RSA with Kotlin

63: RSA with Kotlin

64: RSA with Kotlin

65: RSA with Kotlin

66: RSA with Kotlin

67: RSA with Kotlin

68: RSA with Kotlin

69: RSA with Kotlin

70: RSA with Kotlin

71: RSA with Kotlin

72: RSA with Kotlin

73: RSA with Kotlin

74: RSA with Kotlin

75: RSA with Kotlin

76: RSA with Kotlin

77: RSA with Kotlin

78: RSA with Kotlin

79: RSA with Kotlin

80: RSA with Kotlin

81: RSA with Kotlin

82: RSA with Kotlin

83: RSA with Kotlin

84: RSA with Kotlin

85: RSA with Kotlin

86: RSA with Kotlin

87: RSA with Kotlin

88: RSA with Kotlin

89: RSA with Kotlin

90: RSA with Kotlin

91: RSA with Kotlin

92: RSA with Kotlin

93: RSA with Kotlin

94: RSA with Kotlin

95: RSA with Kotlin

96: RSA with Kotlin

97: RSA with Kotlin

98: RSA with Kotlin

99: RSA with Kotlin

100: RSA with Kotlin

RSA Testing Environment

Server Key

Encryption

Decryption

Signature

Verification

Send Key

Receive Key

Get server key

Clear

Key size300

Get key

Modulus

C1BF044C0E642BFAF4EA0AC34B9FA84FAE2100A1B07EC009E42B3B82233993EFCCEE72BC01

Public exponent

10001

Oleh Chorny © 2020

```
-----
Generating keys for new user...
-----
Enter name of new user: A
Preferable key length(in bits): 256
Generating random prime p...
549189221918262482300654096263965049343 didn't pass primality tests(small primes).
549189221918262482300654096263965049345 didn't pass primality tests(small primes).
549189221918262482300654096263965049347 didn't pass primality tests(small primes).
549189221918262482300654096263965049349 didn't pass primality tests(small primes).
549189221918262482300654096263965049351 didn't pass primality tests(small primes).
549189221918262482300654096263965049353 didn't pass primality tests(small primes).
549189221918262482300654096263965049355 didn't pass primality tests(small primes).
549189221918262482300654096263965049357 didn't pass primality tests(small primes).
549189221918262482300654096263965049359 didn't pass primality tests(small primes).
549189221918262482300654096263965049361 didn't pass primality tests(small primes).
549189221918262482300654096263965049363 didn't pass primality tests(small primes).
549189221918262482300654096263965049365 didn't pass primality tests(small primes).
549189221918262482300654096263965049367 didn't pass primality tests(Miller-Rabin).
549189221918262482300654096263965049369 didn't pass primality tests(Miller-Rabin).
549189221918262482300654096263965049371 didn't pass primality tests(small primes).
Generating random prime q...
445110604932986094408366698186948100573 didn't pass primality tests(small primes).
445110604932986094408366698186948100575 didn't pass primality tests(small primes).
445110604932986094408366698186948100577 didn't pass primality tests(small primes).
445110604932986094408366698186948100579 didn't pass primality tests(small primes).
445110604932986094408366698186948100581 didn't pass primality tests(small primes).
445110604932986094408366698186948100583 didn't pass primality tests(small primes).
445110604932986094408366698186948100585 didn't pass primality tests(small primes).
445110604932986094408366698186948100587 didn't pass primality tests(small primes).
445110604932986094408366698186948100589 didn't pass primality tests(small primes).
445110604932986094408366698186948100591 didn't pass primality tests(small primes).
445110604932986094408366698186948100593 didn't pass primality tests(small primes).
445110604932986094408366698186948100595 didn't pass primality tests(small primes).
```

[illegible]

```
7620124193 didn't pass primality tests(small primes).
7620124195 didn't pass primality tests(small primes).
7620124197 didn't pass primality tests(small primes).
7620124199 didn't pass primality tests(small primes).
7620124201 didn't pass primality tests(small primes).
7620124203 didn't pass primality tests(small primes).
7620124205 didn't pass primality tests(small primes).
Public and secret keys are correct.
```

```
=====
Login: A
Modulus: 0x21c71c44db699c1150b498587f52f646434acf0d4b8f326fd941452cbd3da100b
Public_exponent: 0x1c631de2f
=====
```

```
-----
Generating keys for new user...
-----
```

```
Enter name of new user: B
Preferable key length(in bits): 280
Generating random prime p...
```

```
2599059213583596347105508346721075564290455 didn't pass primality tests(small primes).
2599059213583596347105508346721075564290457 didn't pass primality tests(small primes).
2599059213583596347105508346721075564290459 didn't pass primality tests(small primes).
2599059213583596347105508346721075564290461 didn't pass primality tests(small primes).
2599059213583596347105508346721075564290463 didn't pass primality tests(small primes).
2599059213583596347105508346721075564290465 didn't pass primality tests(small primes).
2599059213583596347105508346721075564290467 didn't pass primality tests(small primes).
2599059213583596347105508346721075564290469 didn't pass primality tests(small primes).
2599059213583596347105508346721075564290471 didn't pass primality tests(Miller-Rabin).
2599059213583596347105508346721075564290473 didn't pass primality tests(small primes).
2599059213583596347105508346721075564290475 didn't pass primality tests(small primes).
2599059213583596347105508346721075564290477 didn't pass primality tests(small primes).
2599059213583596347105508346721075564290479 didn't pass primality tests(small primes).
2599059213583596347105508346721075564290481 didn't pass primality tests(small primes).
2599059213583596347105508346721075564290483 didn't pass primality tests(small primes).
2599059213583596347105508346721075564290485 didn't pass primality tests(small primes).
2599059213583596347105508346721075564290487 didn't pass primality tests(Miller-Rabin).
2599059213583596347105508346721075564290489 didn't pass primality tests(small primes).
2599059213583596347105508346721075564290491 didn't pass primality tests(small primes).
2599059213583596347105508346721075564290493 didn't pass primality tests(small primes).
2599059213583596347105508346721075564290495 didn't pass primality tests(small primes).
2599059213583596347105508346721075564290497 didn't pass primality tests(small primes).
2599059213583596347105508346721075564290499 didn't pass primality tests(Miller-Rabin).
2599059213583596347105508346721075564290501 didn't pass primality tests(small primes).
2599059213583596347105508346721075564290503 didn't pass primality tests(small primes).
2599059213583596347105508346721075564290505 didn't pass primality tests(small primes).
2599059213583596347105508346721075564290507 didn't pass primality tests(small primes).
2599059213583596347105508346721075564290509 didn't pass primality tests(small primes).
2599059213583596347105508346721075564290511 didn't pass primality tests(small primes).
2599059213583596347105508346721075564290513 didn't pass primality tests(Miller-Rabin).
2599059213583596347105508346721075564290515 didn't pass primality tests(small primes).
2599059213583596347105508346721075564290517 didn't pass primality tests(small primes).
2599059213583596347105508346721075564290519 didn't pass primality tests(small primes).
2599059213583596347105508346721075564290521 didn't pass primality tests(small primes).
```

[illegible]

```
42997436199 didn`t pass primality tests(small primes).
42997436201 didn`t pass primality tests(small primes).
42997436203 didn`t pass primality tests(small primes).
42997436205 didn`t pass primality tests(small primes).
42997436207 didn`t pass primality tests(small primes).
42997436209 didn`t pass primality tests(small primes).
42997436211 didn`t pass primality tests(small primes).
Public and secret keys are correct.
```

=====

```
Login: B
Modulus: 0x3998a26977de42a598c26b1e30d81bfef20028de783f009a82ac8e4ead0a8b06edccaef
Public_exponent: 0xa02d8cf35
```

=====

Session#0...

=====

```
Login: A
Modulus: 0x21c71c44db699c1150b498587f52f646434acf0d4b8f326fd941452cbd3da100b
Public_exponent: 0x1c631de2f
```

=====

Encrypting for someone out there

```
Plain message from A : 0xad296
Enter modulus of receiver: C1BF044C0E642BF4F4EA0AC34B9FA84FAE2100A1B07EC009E42B3B82233993EFCSEEE72BC01
Enter public exponent of receiver: 10001
Enciphered message : 0x8171092b5eb430077ca90f7ce4dc4b2c886c6101c646d788e2e46ff59830267a973e816e6a5
```

Відкритий ключ узятий із сайт - перевіримо розшифрування повідомлення на ньому:

RSA Testing Environment

[Server Key](#)[Encryption](#)[Decryption](#)[Signature](#)[Verification](#)[Send Key](#)[Receive Key](#)

Decryption

 Clear

Ciphertext

8171092b5eb430077ca90f7ce4dc4b2c886c6101c646d788e2e46ff59830267a973e8

Bytes



Decrypt

Message

0AD296

Oleh Chornyj © 2020

Розшифровується правильно. Далі вивід main.py:

Тепер зашифроване повідомлення для А формує сайт:

RSA Testing Environment

[Server Key](#)[Encryption](#)[Decryption](#)[Signature](#)[Verification](#)[Send Key](#)[Receive Key](#)

Encryption

 Clear

Modulus

21c71c44db699c1150b498587f52f646434acf0d4b8f326fd941452cbd3da100b

Public exponent

1c631de2f

Message

9e4d5

Bytes



Encrypt

Ciphertext

016A468C50555BC0F0937786E13905491B9902258CEE77245848EA122B3DB79C4E

Oleh Chornyj © 2020

Decrypting message for A

Enciphered message : 016A468C50555BC0F0937786E13905491B9902258CEE77245848EA122B3DB79C4E
Plain message for A : 0x9e4d5

Повідомлення від сайту розшифровується правильно.

Signing message for someone out there

Plain message from A : 0x540e4
Signed message : 0x540e4, signature=0xd53bd16909ab9d2826a81840f732679ede7e44114196228fe66017fe84553915

Первірно(верифікуємо) підпис на сайті:

RSA Testing Environment

[Server Key](#)
[Encryption](#)
[Decryption](#)
[Signature](#)
[Verification](#)
[Send Key](#)
[Receive Key](#)

Verify

✕ Clear

Message

540e4

Bytes ▾

Signature

d53bd16909ab9d2826a81840f732679ede7e44114196228fe66017fe84553915

Modulus

21c71c44db699c1150b498587f52f646434acf0d4b8f326fd941452cbd3da100b

Public exponent

1c631de2f

Verify

Verification

true

✓


Oleh Chomyi © 2020

Підпис верифіковано.
Тепер А верифікує підписане повідомлення для нього від сайту.
Підпис сайту:

RSA Testing Environment

[Server Key](#)[Encryption](#)[Decryption](#)[Signature](#)[Verification](#)[Send Key](#)[Receive Key](#)

Sign

 Clear

Message

ABCD

Bytes



Sign

Signature

0889A6D7B56EE1740ED2D9AC3194841A8E768317446677A816E19A2D3EFA8D408E86E7F113B6

Oleh Chorny © 2020

Verifying message for A

Enter message itself: ABCD

Enter its signature : 0889A6D7B56EE1740ED2D9AC3194841A8E768317446677A816E19A2D3EFA8D408E86E7F113B6

Enter modulus of signer : C1BF044C0E642BFAF4EA0AC34B9FA84FAE2100A1B07EC009E42B3B82233993EFCCEEE72BC01

Enter public exponent of signer : 10001

Message is verified : 0xabcd

Повідомлення верифіковано.

Далі надішлемо зашифрований підписаний ключ(по протоколу конфіденційного розсилання ключів) на сайт:

Generating and sending key to someone

Key to send : 0x70aab8b0f50a6cb9c61fda17febafd1eee5e49463532cd849f

Enter modulus of receiver : C1BF044C0E642BFAF4EA0AC34B9FA84FAE2100A1B07EC009E42B3B82233993EFCCEEE72BC01

Enter public exponent of receiver : 10001

Signed&Enciphered key to send : 0x48100052e364b0925b34df4b8f748fb821b9c629b2e363a605cefb6686c835508f08adf7c6d,
signature=0x784ealbe3548f40257c7fb40d7ca5c0fa35588645f38a843f1e392954505cf8d4f074d189b2

Відповідно на сайті:

1) А вибирає ключ, що він хоче розділити з В: 0x165d551004e029f611be60f4cb582166069fe0d26439fee3a2

2) А підписує цей ключ своїм секретним ключем $d=0x186b6dcac79c7ad9b78bc3df3b82246bf29e9e89adff53cb50bcf2823c8b6d617$,
 $n=0x1fe280dbccbff99929ca44e7ad5f39a4907f444c16515fff4ac21411c504aa7f3$

Отримуємо підписаний ключ: $0x165d551004e029f611be60f4cb582166069fe0d26439fee3a2$,
 $sign=0xa6450ef3cfd2e7e9dcc49b2b843ccb0bfe4a18034336b8786b2e850d6eed308$

3) А шифрує ключ та його підпис відкритим ключем В $e=0x9d9c2e0ef$, $n=0x14ce066610c1e1a159b6f57567fe438968991500e0b54cb4300c2036871847c2e64c9c9$

Отримуємо пару значень: $0x96d630e9b5a6b83cceb7ac7b447d03df1c7b167723815ba11d51521a64f90db45f61c4$,
 $sign=0x497232e1119109f5fbc5affb1f724d2dba0817e85d68d4efc97a6aa91241c4ac80eed$

4) А відправляє цю пару значень В по відкритому каналу.

5) В розшифровує обидва значення своїм закритим ключем: $d=0x8dc32d288f19c7bc4f1e07fa5ba78bc1e25829eda238abb77dd1b920cd8aa1e94db08f$,
 $n=0x14ce066610c1e1a159b6f57567fe438968991500e0b54cb4300c2036871847c2e64c9c9$

Отримує: $0x165d551004e029f611be60f4cb582166069fe0d26439fee3a2$, $sign=0xa6450ef3cfd2e7e9dcc49b2b843ccb0bfe4a18034336b8786b2e850d6eed308$

Ключ вже отримали відкритий (отже він призначався саме В).

6) Тепер В перевірить підпис цього ключа відкритим ключем А і тим самим пересвідчиться, що ключ йому відправив саме А:

$e=0x10025158f$, $n=0x1fe280dbccbff99929ca44e7ad5f39a4907f444c16515fff4ac21411c504aa7f3$

Перевірка полягає в розшифровці самого підпису відкритим ключем А та порівняння отриманого значення із тим ключем, що був підписаний:

Ключ, що підписував А: $0x165d551004e029f611be60f4cb582166069fe0d26439fee3a2$

Розшифрований підпис: $0x165d551004e029f611be60f4cb582166069fe0d26439fee3a2$

Отже підпис справжній. Конфіденційна передача ключового значення $0x165d551004e029f611be60f4cb582166069fe0d26439fee3a2$ між користувачами А та В пройшла успішно.