Міністерство освіти і науки України
Національний технічний університет україни
"київський політехнічний інститут імені ігоря сікорського"
Фізико-технічний інститут

Криптографія
Комп’ютерний практикум №4

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

Виконали:
Студенти гр. ФБ-11
Поліщук Олександра
Маленко Сергій
3 варіант

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:

Увесь код даного практикума міститься у main.py

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.

Створена ф-ція generate_prime() та її допоміжна check_prime(). У випадку, что згенероване число не просте, повертається False. Інакше – True. If True -> використовуємо сгенероване число як ключ. Для наглядності, можна вивести у консоль.

```
def generate_prime(length=256):
    lower = 1 << (length - 1)
    higher = (1 << length) - 1
    while True:
        p = randint(lower, higher)
        print(p, check_prime(p))
        if check_prime(p):
            return p
```

```
71554311055429602073760833965297375545353824661155561129362565250007590386354 False
11057225419146222946112432844964612140359111197944189227419370949783029790491 1 False
643813088498593935119332442279612724192534374448422375392171232523102658608 42 False
59872041996062204193561856578740894748557217806939480493262247919249383032527 False
98947025967417534540488187439851793556073926565175620504932790838991027414954 False
60979286943093859150055517532330194070751599548708668546985611428227733469779 False
11212168503710304975458400550584552023609811865621586015049555569184956487271 3 False
90488679834900862684079819107770584084185875489607860278825118190274408198673 False
90245993835000135854172790730323238163606463528602342618568762833685986725774 False
91804923893029858208000334797811054425388487504468615160582912925213276582227 7 False
94840963542390483676044564782272004705479446884001428191545672123475564483987 True
```

2. За допомогою цієї функції згенерувати дві пари простих чисел p, q і 1 1 p , q довжини щонайменше 256 біт. При цьому пари чисел беруться так, щоб pq ≤ p1q1 ; p і q – прості числа для побудови ключів абонента A, 1 p і q1 – 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) , (,) 1 n1 e та секретні d і d1 .

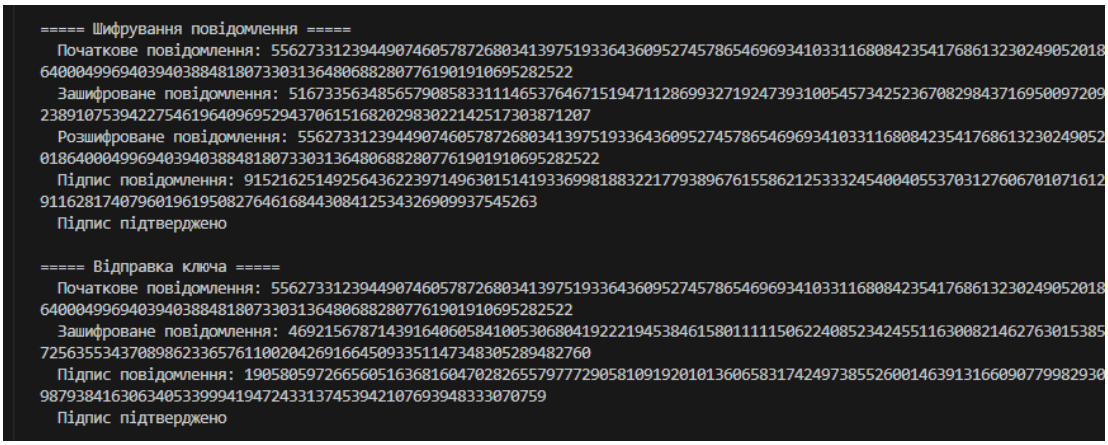
Створили функцію generate_key_pair(), яка за допомогою generate_prime() генерує ключі та повертає публічний та секретний ключ користувача. У консолі це виглядає наступним чином.

```
===== User A =====
  Public Key:
    n = 730072940902812061676591558035528480106764158927029876075381446837268488244495024054146608568445524525943036923113753694364405356696803840611049187925656 7
    e = 65537
  Private Key:
    p = 769786507468891158921637987000871760449895296591929031627787703749646502913 41
    q = 948409635423904836760445647822720047054794468840014281915456721234755644839 87
    d = 164758514975549542887175017827264998714909774791808777746233297201934799291023807689569166977425697355220823456437164861427013302553865495097154651750427 3

===== User B =====
  Public Key:
    n = 770824252760996541196639422813151046216611141062522817568768240482330739715698032089819271606529749845376609503518912067787495302313201501909044377875909 7
    e = 65537
  Private Key:
    p = 668811873266273805213301192603232972501232731999586194012814789152921066149 27
    q = 115252776389349264421737018886783630405846018651626551177972609437167860198711
    d = 667956728848585601585338808016565620423387755192970891748903934345973191314725614285359214654960325478501179415620054626186204867043538529407114818597425 3
```

3. 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.

Створили функції encrypt(), decrypt(), sign_message() та verify()



4. 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 $0 < k < n$.

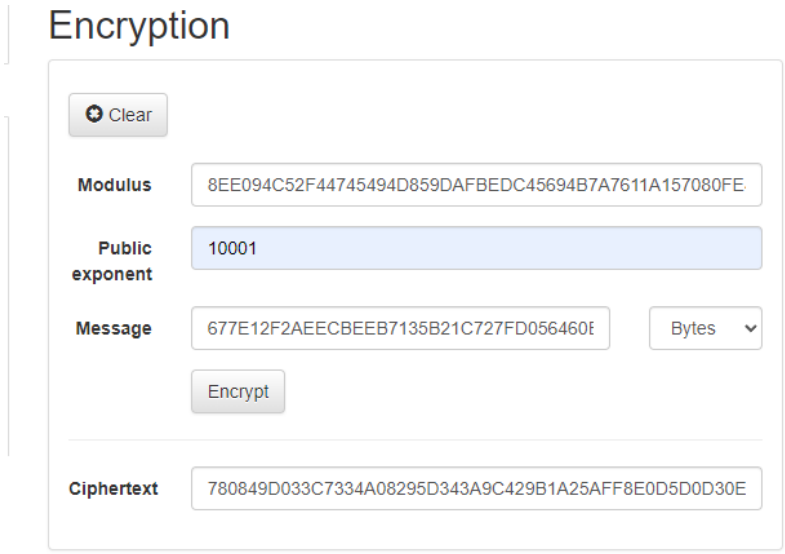
Візьмемо одну із робот нашої програми. Для прикладу одразу переведемо початкове повідомлення та n у hex-формат, оскільки сайт приймає саме його.

```
===== User B =====
Generating p, q....
Calculating n....
Calculating Oyler...
Calculating secret key for deciphering...
  Public Key:
    n = 9343325059646857234941733677017053788587598961256740589800529944679194758691 9
    e = 65537
  Private Key:
    p = 28174143730085356666424599370232563533 7
    q = 331627649420618278513089844949435739887
    d = 919705270187647294022195372443119393551150786997092795166392230392714276317 61
Generating message...

Hex n(USER A) 8EE094C52F44745494D859DAFBEDC45694B7A7611A157080FE40E33DBDB2A7B7


===== Шифрування повідомлення =====
  Початкове повідомлення: 468109769084015804239170017543002440203113967329467377901422595309388509371 94
  Started in hex: 677E12F2AEECBEEB7135B21C727FD056460EC91082D423A5D888672C36A7096A
  Зашифроване повідомлення: 54292186047115289303294179100872918282463868657425397028193599815612209202778
  Cyphered in hex 780849D033C7334A08295D343A9C429B1A25AFF8E0D5D0D30E1E3CDAB15A865A
  Розшифроване повідомлення: 468109769084015804239170017543002440203113967329467377901422595309388509371 94
  Підпис повідомлення: 561351330907194650556344186419570150757832851663019236006461575976654936273 71
  Підпис підтверджено
```

Перевіряємо за допомогою онлайн сервісу.



Шифрування повністю вірно працює

Висновки: під час виконання комп’ютерного практикуму, ми детальніше розглянули алгоритм RSAб його особливості, переваги та захист інформації.