



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 групи ФБ-93
Приходько Андрій
Шахова Катерина

Мета та основні завдання 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 і рекомендації щодо 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 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 A , p_1 і 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) , (e_1, n_1) 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()`.

Виконання :

Згенерували p , q , $p1$, $q1$:

p	199678154031684671539430416713290906166999585391355487044350862215685268344879
q	205174100210673205200058779618100018279183148869024122379502109050216467008623
$p1$	226252153381013260702479181319917934576168461407559243409221620798063872767099
$q1$	186914172463734799398029317962633444089705791146136750986504945743094240325439

Кандидати, що не пройшли перевірку зберігаються у файлі not_passed_test.txt

Далі за допомогою цих чисел змогли згенерувати публічні та приватні ключі для так званих Аліси та Боба :

Аліса	
e	31811810397541411518390776294835057193314687426271862377448009668698747444709905182745849045723700219230766301737381579326438243281828096105520784648772369
n	40968785585179110685184299376332342832605494307488528671728507393914766727141639329418216226852424329933127753248984922917069362009463494923614991430891617
d	7195740480726333941257813343609756758476520065810118550975628695567716773384083132290873373314233700176193109954563146216146339386404509588001316335249745
p	199678154031684671539430416713290906166999585391355487044350862215685268344879
q	205174100210673205200058779618100018279183148869024122379502109050216467008623

Боб	
e	13437496771523038521314953327205306261853919451770944010713823 61635266673526050706642638391337053264766868458323636862212275 8221149114579494587168605441355
n	42289734017350091106740379689670948367559293508426883343057712 11104435137799270227171673604013086794484751058302247689527643 3613203138860780237099211931461
d	33386278783561399406060190743965534352984733098170016997745157 82148608763060739087225647645161314427622211994332101825539646 0314418506314578762639442257007
p1	22625215338101326070247918131991793457616846140755924340922162 0798063872767099
q1	18691417246373479939802931796263344408970579114613675098650494 5743094240325439

Далі створимо повідомлення 12345678901234567890

Та зашифруємо його публічним ключем Аліси

```
# частина для Аліси
encrypted_alice = encrypt(msg,alice_public)
print("Encrypted message by Alice:",encrypted_alice)
decrypted_alice = decrypt(encrypted_alice,alice_private)
print("Decrypted message by Alice:",decrypted_alice)
```

encrypted	348091275952585643600520742836696173656663646156745801226 295621314577911247985814929013242616395965370128311317382 7410525964641213123023774340381482843743
decrypted	12345678901234567890

Також Аліса підписала це повідомлення :

```
alice_sign = sign(msg,alice_private)
print("Signature for Alice:",alice_sign)
print("Sign check: ",verify(msg,alice_sign,alice_public))
print("\n")
```

Alice signature	337042234217611498960472252907540862198585317116834389814 941918984427500972693806887954674214582130592843508708891 94743528024130562444744168658895313904082
--------------------	---

```
Signature for Alice:
3370422342176114989604722529075408621985853171168343898149419189844275
0097269380688795467421458213059284350870889194743528024130562444744168
658895313904082
Sign check: True
```

Зробимо таке саме для Боба :

```
# частина Боба
encrypted_bob = encrypt(msg, bob_public)
print("Encrypted message by Bob:", encrypted_bob)
decrypted_bob = decrypt(encrypted_bob, bob_private)
print("Decrypted message by Bob:", decrypted_bob)
```

encrypted	354613949746155063596491683385824792357902890217794177356 458025150719599593954940405308648682916998956005151491014 27843694542390534088536242356975280088437
decrypted	12345678901234567890

```
bob_sign = sign(msg, bob_private)
print("Signature for Bob:", bob_sign)
print("Sign check: ", verify(msg, bob_sign, bob_public))
print("\n")
```

Bob signature	374048817531805180320725544243278946561526096625986279837 534312533151230543931009173642881314841106163294496521784 79971173631233686656787865719400426670926
------------------	---

```
Decrypted message by Bob: 12345678901234567890
Signature for Bob:
3740488175318051803207255442432789465615260966259862798375343125331512
3054393100917364288131484110616329449652178479971173631233686656787865
719400426670926
Sign check: True
```

Протокол конфіденційного розсилання ключів по відкритих каналах зв'язку з підтвердженням справжності відправника

Нехай Аліса має свої ключі та відкритий ключ Боба. Тоді вона формує повідомлення (k_1, s_1) та відправляє його Бобу.

В свою чергу за допомогою приватного ключа Боб розшифровує повідомлення та може перевірити підпис Аліси за її публічний ключем.

```
174         k1,s1 = send_key(msg,bob_public,alice_private)
175
176     recieve_key(k1,s1,bob_private,alice_public)
177
recieve_key() > else
main x
↑ message: 12345678901234567890
↓ message verified!
```

Перевірка з сайтом

1) Згенерували ключі у себе. Закинули на сайт. Шифроване повідомлення однакове

The screenshot shows a web application interface on the left and a Python script on the right. The web application has a sidebar with options: Server Key, Encryption, Decryption, Signature, Verification, Send Key, and Receive Key. The 'Encryption' tab is active. It contains a 'Clear' button, a 'Modulus' field with the value '30E3B40A80AFC4EC61F90A2C19358F9BC18AE2884178DA6EA09D', a 'Public exponent' field with the value '25F64E3E3DAA16FDDCCA1C6E88C6E9930DC594671B0809900AC3A18E', a 'Message' field with the value 'Great! We did it!', a 'Text' dropdown menu, an 'Encrypt' button, and a 'Ciphertext' field with the value 'CC65DB8E114E40023B3B7C68FAB4966490A744514AB003E65D38'. The Python script on the right is a file named 'main.py' with the following code:

```
220 my_open = [e,n]
221 my_private = [d,p,q]
222
223
224 #.site_encryption
225 print('mod = ', int_to_hex(n))
226 print('e = ', int_to_hex(e))
227 enc = encrypt(hex_to_int(encode_to_hex(msg)),my_open)
228 print('enc= ',int_to_hex(enc))
229
```

The script is run in a terminal window, showing the output of the encryption process.

2) Зашифрували у себе ключем з серверу. Розшифрували на ньому

RSA Testing Environment

The screenshot shows a web application interface on the left and a Python script on the right. The web application has a sidebar with options: Server Key, Encryption, Decryption, Signature, Verification, Send Key, and Receive Key. The 'Decryption' tab is active. It contains a 'Clear' button, a 'Ciphertext' field with the value '8E88C65103D110FA7A06225B76C206404BE8CE7BE678F60E813D5', a 'Text' dropdown menu, a 'Decrypt' button, and a 'Message' field with the value 'Great! We did it!'. The Python script on the right is a file named 'main.py' with the following code:

```
222
223
224 #.site_encryption
225 print('mod = ', int_to_hex(n))
226 print('e = ', int_to_hex(e))
227 enc = encrypt(hex_to_int(encode_to_hex(msg)),my_open)
228 print('enc= ',int_to_hex(enc))
229
230
231 #.site_decryption
232 encrypted = encrypt(hex_to_int(encode_to_hex(msg)),server_open)
233 print('Encrypted = ',int_to_hex(encrypted))
234
235 if __name__ == '__main__':
236     main()
237
```

The script is run in a terminal window, showing the output of the decryption process.

3) Підписали на сервері. Перевірили у себе

RSA Testing Environment

Server Key

Encryption

Decryption

Signature

Verification

Send Key

Receive Key

Sign

Clear

Message

hello

Text

Sign

Signature

1D6B5ED18B2F0635CE5E7A19D44850EAF7BBC67A42841BD0BCF

```
222
223
224 # site encryption
225 print('mod = ', int_to_hex(n))
226 print('e = ', int_to_hex(e))
227 enc = encrypt(hex_to_int(encode_to_hex(msg)), my_open)
228 print('enc = ', int_to_hex(enc))
229
230
231 # site decryption
232 encrypted = encrypt(hex_to_int(encode_to_hex(msg)), server_open)
233 print('Encrypted = ', int_to_hex(encrypted))
234
235 # site sign
236 sg = '6CA04253EBCD2B2C45C82D7B39F8701229E8C72105BA636723D9619351F'
237 print('verify(hex_to_int(encode_to_hex(msg)), hex_to_int(sg), server'
238
```

Run: main x

Encrypted =
8E88C65103D110FA7A06225B76C206404BE8CE7BE678F60E813D550EAFCA1A0BE
True

4) Підписали у себе. Перевірили на сервері

Verify

Clear

Message

Great! We did it!

Text

Signature

E22A83F061CD0D667B6DE14AFC4FC0F6DD0E3303CEB912E3CA4

Modulus

30E3B40A80AFC4EC61F90A2C19358F9BC18AE2884178DA6EA09D

Public exponent

25F64E3E3DAA16FDDCCA1C6E88C6E9930DC594671B0809900AC

Verify

Verification

true

```
229
230
231 # site decryption
232 encrypted = encrypt(hex_to_int(encode_to_hex(msg)), server_open)
233 print('Encrypted = ', int_to_hex(encrypted))
234
235 # site sign
236 site_sg = '6CA04253EBCD2B2C45C82D7B39F8701229E8C72105BA636723D9619351F'
237 print('verify(hex_to_int(encode_to_hex(msg)), hex_to_int(site_sg), server'
238
239
240 # my sign
241 my_sg = sign(hex_to_int(encode_to_hex(msg)), my_private)
242 print('My sign = ', int_to_hex(my_sg))
243
244
245 if __name__ == '__main__':
246     91CC1932C36109AB0DAE5211F615510BDA3313DFE8580284967E79D51
247     Encrypted =
248     8E88C65103D110FA7A06225B76C206404BE8CE7BE678F60E813D550EAFCA1A0BE
249     True
250     My sign =
251     E22A83F061CD0D667B6DE14AFC4FC0F6DD0E3303CEB912E3CA41C6629479EC06E60ACB,
252     82FC91583261611A550F1E1E1155CE5137BE298736B305EEC53595753
253
254     Process finished with exit code 0
```

Висновок: виконуючи дану ЛР ми дізналися про принцип роботи асиметричної криптографії на прикладі алгоритму RSA. Ми змогли зробити функцію перевірки чисел на простоту та за допомогою цього могли генерувати великі прості числа для ключів. Дізналися про цифрові підписи RSA та протокол розсилання ключів.