



**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ "КИЇВСЬКИЙ
ПОЛІТЕХНІЧНИЙ ІНСТИТУТ"
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ**

Комп'ютерний практикум

№4

З дисципліни

Криптографія

Виконав студент групи ФБ-81

Середа А. С.

Перевірив

Чорний О. М.

Київ – 2020

19.11

Тема: 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.

Завдання: Варіант (18)

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 А, $1 < p$ і q_1 – абонента В.

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 А і В – тобто, створити та зберегти для подальшого використання відкриті ключі (e, n) , $(,)$ і n і секретні d і d_1 .

4. 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.

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()`. 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.

Хід роботи:

Спершу я створив функції для обробки даних, такі як `find_random_bin` (для створення зовсім випадкового числа), `bin_to_int` (для переведу бінарного масива в `crr_int`), `mod` та перегрузив функцію `pow`.

Потім я створив функції для пошуку p та q та функцію перевірки простоти. Також я створив декілька допоміжних функцій.

Потім я створив функції `Encrypt()`, `Decrypt()`, `Sign()`, `Verify()`, та вирішив створити псевдосистему з двома користувачами і можливістю змінювати користувача.

Потім я створив функції `SendKey()` та `ReceiveKey()`.

Я перевіряв працювання функцій на малих розмірах (5, 10, 15), та почав чекати (поки виконається програма для 256 бітів).

Проблеми з якими я зіткнувся:

- Час обробки програми для 256 бітних чисел. Було вирішено за допомогою функції `powm(a, b, c)` ($a^b \bmod c$);
- Спочатку я не знав що можна використовувати бібліотеки для великих чисел, та почав (та майже повністю реалізував) власну бібліотеку для обробки чисел у двійковій формі.

Параметри криптосистеми:

$P1 = 97745765517377919795220861649417698800994744977913532517543914221364281846643$

Відсіяні кандидати:

97745765517377919795220861649417698800994744977913532517543914221364281846419
97745765517377919795220861649417698800994744977913532517543914221364281846421
97745765517377919795220861649417698800994744977913532517543914221364281846423
97745765517377919795220861649417698800994744977913532517543914221364281846425
97745765517377919795220861649417698800994744977913532517543914221364281846427
97745765517377919795220861649417698800994744977913532517543914221364281846429
97745765517377919795220861649417698800994744977913532517543914221364281846431
97745765517377919795220861649417698800994744977913532517543914221364281846433
97745765517377919795220861649417698800994744977913532517543914221364281846435
97745765517377919795220861649417698800994744977913532517543914221364281846437
97745765517377919795220861649417698800994744977913532517543914221364281846439
97745765517377919795220861649417698800994744977913532517543914221364281846441
97745765517377919795220861649417698800994744977913532517543914221364281846443
97745765517377919795220861649417698800994744977913532517543914221364281846445
97745765517377919795220861649417698800994744977913532517543914221364281846447
97745765517377919795220861649417698800994744977913532517543914221364281846449
97745765517377919795220861649417698800994744977913532517543914221364281846451
97745765517377919795220861649417698800994744977913532517543914221364281846453
97745765517377919795220861649417698800994744977913532517543914221364281846455
97745765517377919795220861649417698800994744977913532517543914221364281846457
97745765517377919795220861649417698800994744977913532517543914221364281846459
97745765517377919795220861649417698800994744977913532517543914221364281846461
97745765517377919795220861649417698800994744977913532517543914221364281846463
97745765517377919795220861649417698800994744977913532517543914221364281846465
97745765517377919795220861649417698800994744977913532517543914221364281846467
97745765517377919795220861649417698800994744977913532517543914221364281846469
97745765517377919795220861649417698800994744977913532517543914221364281846471
97745765517377919795220861649417698800994744977913532517543914221364281846473
97745765517377919795220861649417698800994744977913532517543914221364281846475
97745765517377919795220861649417698800994744977913532517543914221364281846477
97745765517377919795220861649417698800994744977913532517543914221364281846479
97745765517377919795220861649417698800994744977913532517543914221364281846481
97745765517377919795220861649417698800994744977913532517543914221364281846483
97745765517377919795220861649417698800994744977913532517543914221364281846485

Q1 = 98219030841433552464311142587428173157294506280006971099954355202407009277971

[illegible]

~ 6 ~

~ 7 ~

Відсіяні кандидати:

282

$\sim 9 \sim$

~ 10

~ 11

[illegible]

Відсіяні кандидати:

[illegible]

107807658232732500280764221714368918213579073154766277077524352432091454255491
107807658232732500280764221714368918213579073154766277077524352432091454255493
107807658232732500280764221714368918213579073154766277077524352432091454255495
107807658232732500280764221714368918213579073154766277077524352432091454255497
107807658232732500280764221714368918213579073154766277077524352432091454255499
107807658232732500280764221714368918213579073154766277077524352432091454255501
107807658232732500280764221714368918213579073154766277077524352432091454255503
107807658232732500280764221714368918213579073154766277077524352432091454255505
107807658232732500280764221714368918213579073154766277077524352432091454255507
107807658232732500280764221714368918213579073154766277077524352432091454255509
107807658232732500280764221714368918213579073154766277077524352432091454255511
107807658232732500280764221714368918213579073154766277077524352432091454255513
107807658232732500280764221714368918213579073154766277077524352432091454255515
107807658232732500280764221714368918213579073154766277077524352432091454255517
107807658232732500280764221714368918213579073154766277077524352432091454255519
107807658232732500280764221714368918213579073154766277077524352432091454255521
107807658232732500280764221714368918213579073154766277077524352432091454255523

Alice Open keys:

e = 4097;

n =

960049435797087414333526051709038794067396743914473829321591329379192898369546602115768863622
6938542019264910630702054754493211505517548303857109980201353;

Alice Close keys:

d =

164265231753419154856676046435937562763301224672698597596884432974082065354415042946284896440
1520621435326988868899043109047580910991441316606729416900853;

p = 97745765517377919795220861649417698800994744977913532517543914221364281846643;

q = 98219030841433552464311142587428173157294506280006971099954355202407009277971;

Bob Open keys:

e = 4097;

n =

113296673373141981233162160426083194648904854185348657049835232106093092465193538254643662457
74270895941927259788050453234285942360081675766343945947516687;

Bob Close keys:

d =

321334475127144208427958092299508596978343765104650084188207321716573525615212149289246340166
0631721965036301572134077390089088510296153956258066796791249;

p = 105091489074514475704275531956670955006138302959923392226748007036381458447269;

q = 107807658232732500280764221714368918213579073154766277077524352432091454255523;

Процедура пересилання ключів:

Key sending has begun ...

Encryption has begun ...

Message k = 23001523707977109526751469666956383760224992265869994285080028118503487852208;

e = 4097;

n =

960049435797087414333526051709038794067396743914473829321591329379192898369546602115768863622
6938542019264910630702054754493211505517548303857109980201353

236357737817339757917944686997110516638624047852305142013788344740258292488841660938290045213
4337244285275309879276846086329673508279253761396882030470036

Encrypted message k1 =

236357737817339757917944686997110516638624047852305142013788344740258292488841660938290045213
4337244285275309879276846086329673508279253761396882030470036

Encryption end.

Signing ...

Encryption has begun ...

Message k = 23001523707977109526751469666956383760224992265869994285080028118503487852208;

d =

321334475127144208427958092299508596978343765104650084188207321716573525615212149289246340166
0631721965036301572134077390089088510296153956258066796791249;

$n =$
113296673373141981233162160426083194648904854185348657049835232106093092465193538254643662457
74270895941927259788050453234285942360081675766343945947516687
592638808357935433572574921256910615454900005353992245323675311891664102949196814367740245905
5738136623571952722301187722295491710075691363266345049631427
Encrypted message $S =$
592638808357935433572574921256910615454900005353992245323675311891664102949196814367740245905
5738136623571952722301187722295491710075691363266345049631427
Encryption end.
Encryption has begun ...
Message $S =$
592638808357935433572574921256910615454900005353992245323675311891664102949196814367740245905
5738136623571952722301187722295491710075691363266345049631427;
 $d = 4097$;
 $n =$
960049435797087414333526051709038794067396743914473829321591329379192898369546602115768863622
6938542019264910630702054754493211505517548303857109980201353
254256554772135345782617062555832668914576501103577822152818601005618055504900709407832183526
7462541120346473174677586592710311081628045376174836517566031
Encrypted message $S1 =$
254256554772135345782617062555832668914576501103577822152818601005618055504900709407832183526
7462541120346473174677586592710311081628045376174836517566031
Encryption end.
 $k1 =$
236357737817339757917944686997110516638624047852305142013788344740258292488841660938290045213
4337244285275309879276846086329673508279253761396882030470036
 $S =$
592638808357935433572574921256910615454900005353992245323675311891664102949196814367740245905
5738136623571952722301187722295491710075691363266345049631427
 $S1 =$
254256554772135345782617062555832668914576501103577822152818601005618055504900709407832183526
7462541120346473174677586592710311081628045376174836517566031
Key sended.
User switched to Alice ...
Key receiving has begun ...
Decryption has begun ...
Encrypted message $k1 =$
236357737817339757917944686997110516638624047852305142013788344740258292488841660938290045213
4337244285275309879276846086329673508279253761396882030470036;
 $d =$
164265231753419154856676046435937562763301224672698597596884432974082065354415042946284896440
1520621435326988868899043109047580910991441316606729416900853;
 $n =$
960049435797087414333526051709038794067396743914473829321591329379192898369546602115768863622
6938542019264910630702054754493211505517548303857109980201353;
Decrypted message $k =$
23001523707977109526751469666956383760224992265869994285080028118503487852208
Decryption end.
Decryption has begun ...
Encrypted message $S1 =$
254256554772135345782617062555832668914576501103577822152818601005618055504900709407832183526
7462541120346473174677586592710311081628045376174836517566031;
 $d =$
164265231753419154856676046435937562763301224672698597596884432974082065354415042946284896440
1520621435326988868899043109047580910991441316606729416900853;
 $n =$
960049435797087414333526051709038794067396743914473829321591329379192898369546602115768863622
6938542019264910630702054754493211505517548303857109980201353
Decrypted message $S1 =$
592638808357935433572574921256910615454900005353992245323675311891664102949196814367740245905
5738136623571952722301187722295491710075691363266345049631427
Decryption end.
 $k = 23001523707977109526751469666956383760224992265869994285080028118503487852208$

S1 =
592638808357935433572574921256910615454900005353992245323675311891664102949196814367740245905
5738136623571952722301187722295491710075691363266345049631427
Verifying ...
Decryption has begun ...
Encrypted message S1 =
592638808357935433572574921256910615454900005353992245323675311891664102949196814367740245905
5738136623571952722301187722295491710075691363266345049631427;
e = 4097;
n =
113296673373141981233162160426083194648904854185348657049835232106093092465193538254643662457
74270895941927259788050453234285942360081675766343945947516687
Decrypted message k =
23001523707977109526751469666956383760224992265869994285080028118503487852208
Decryption end.
Message verified!
Key receiving end.

Висновок

Під час цієї лабораторної я розібрався та реалізував власноруч алгоритм шифрування та електронного підпису RSA.