МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Компьютерний практикум №4

З дисципліни: «Криптографія»

Виконали:
Студенти гр. ФБ-03
Гузенков А.М.
Сірховець А.М
Перевірив:
Чорний О.М.

Київ – 2022

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.

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 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 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 $0 < k < n$.

Хід роботи

У лабораторній роботі використовується самописка бібліотека великих чисел verylong, що має простір імен vl. Бібліотекою передбачені майже всі математичні операції з великими невід’ємними числами. Крім того використовується власна бібліотека vlalgorithm, що є збірником функцій, впроваджуючих модульну арифметику.

1.

```
verylong gen_prime(unsigned len)
{
   verylong prime = gen_verylong(len/32);

   if((prime[0] & 1) == 0)
      prime = prime + 1;
   while(!is_prime(prime))
      prime = prime + 2;
   return prime;
}
```

2, 3.

```
key_pair gen_keys()
{
   verylong p = gen_prime(256);
   cout << "[debug message]   " << "p = 0x" << p.to_hex_string() << endl;
   verylong q = gen_prime(256);
   cout << "[debug message]   " << "q = 0x" << q.to_hex_string() << endl;
   verylong n = p*q;
   cout << "[debug message]   " << "n = 0x" << n.to_hex_string() << endl;
   verylong phi = verylong(p-1) * verylong(q-1);
   cout << "[debug message]   " << "phi = 0x" << phi.to_hex_string() << endl;
   verylong e = gen_in_interval(phi);
   while(gcd(e, phi) != 1)
      e = e - 1;
   cout << "[debug message]   " << "e = 0x" << e.to_hex_string() << endl;
   while(e < 2 || gcd(e, phi) != 1)
      e = gen_in_interval(phi);
   verylong d = invert(e, phi);
   cout << "[debug message]   " << "d = 0x" << d.to_hex_string() << endl;
   return key_pair {{d, n}, {e, n}};
}
```

```
***** Generating Alice's keys pair *****
[debug message]   p = 0x4add427642598ab64dbb686964fac4654dfc9637b8f133fb2fe385de54b621b3
[debug message]   q = 0xdad7ce6a5627127292b86062cbe97f8cb22865e563075ff77b995487e0b4f8d3
[debug message]   n = 0x3fff86c8aa1c691c2e693607b9bb1005bd1d3dd0662a46932ddc1dedce90336d2bfada486db0ef0a1678a5f15cb0047d6a53d6baa44b8f5420bf2741949f2e89
[debug message]   phi = 0x3fff86c8aa1c691c2e693607b9bb1005bd1d3dd0662a46932ddc1dedce90336c0645c967d53051e13604dd252bcbc08b6a2eda9d8852fb6175424cdb5f341404
[debug message]   e = 0xae1586e377763bed3e053f0ea313494d5e8447138dd86e43ca1b9ffa079351dcc89595253fdb0e2db5d7ac364d56474e9dae3c750758d56d455f4061
[debug message]   d = 0x27b2fc88e43cd310315874bfb80169e66f5da91ea62637f30a4fba6357c176b075156d59e3482f3d0e645f7138adae62501cce4fb267c68458c6ba47d32817ed
******** Alice's keys generated ********

****** Generating Bob's keys pair ******
[debug message]   p = 0x6703137173b4af04aec26192b98c14b3d7804bb854e0e25ee955c47c74e3f8f3
[debug message]   q = 0x9b2c351006975edcad1e431b0cd55a7f03b44d70267f045882579addb81f81bd
[debug message]   n = 0x3e70a6a71e6cf8829c37be1f885f57e972611515ad9159230afeb0ae1571ddf695ad864c8e4cf3ccb958a598216f7d40306263fdef003571c88fe119fa2e3e67
[debug message]   phi = 0x3e70a6a71e6cf8829c37be1f885f57e972611515ad9159230afeb0ae1571ddf5937e3dcb1400e5eb5d7800ea5b0e0e0d552dcad573a04eba5ce281bfcd2ac3b8
[debug message]   e = 0xf5def6ebf79b468d443110c70d3a7ac143b708fdb9e0d0cfca2c88da614ef48e78aa0fabfda5a49303768e681b5ba6ae51d3c492e54545c5846f52df
[debug message]   d = 0x4ec2c8208776970b8e1668d0928284d3234995a3ffffaad75208c6224ba12a50f484fb169ac343b7b8928204ddea0722be31ca063c5ff5f745dbd4824172b9f
********* Bob's keys generated *********
```

4.

```
// Шифрування
verylong encrypt(verylong message, public_key key)
{
  return powmod_barret(message, key.e, key.n);
}
// Дешифрування
verylong decrypt(verylong cypher, private_key key)
{
  return powmod_barret(cypher, key.d, key.n);
}
// Підписати повідомлення
vl::verylong sign(vl::verylong message, private_key key)
{
  return powmod_barret(message, key.d, key.n);
}
// Перевірити підпис
vl::verylong verify(vl::verylong signature, public_key key)
{
  return powmod_barret(signature, key.e, key.n);
}
```

5.

Функція, що емулює обмін ключами:

```
void exchange_keys(user& u1, user& u2)
{
  u1.set_pubkey(u2.get_name(), u2.get_pubkey());
  u2.set_pubkey(u1.get_name(), u1.get_pubkey());
}
```

```
***** Generating Alice's keys pair *****
[debug message]   p = 0x4add427642598ab64dbb686964fac4654dfc9637b8f133fb2fe385de54b621b3
[debug message]   q = 0xdad7ce6a5627127292b86062cbe97f8cb22865e563075ff77b995487e0b4f8d3
[debug message]   n = 0x3fff86c8aa1c691c2e693607b9bb1005bd1d3dd0662a46932ddc1dedce90336d2bfada486db0ef0a1678a5f15cb0047d6a53d6baa44b8f5420bf2741949f2e89
[debug message]   phi = 0x3fff86c8aa1c691c2e693607b9bb1005bd1d3dd0662a46932ddc1dedce90336c0645c967d53051e13604dd252bcbc08b6a2eda9d8852fb6175424cdb5f341404
[debug message]   e = 0xae1586e377763bed3e053f0ea313494d5e8447138dd86e43ca1b9ffa079351dcc89595253fdb0e2db5d7ac364d56474e9dae3c750758d56d455f4061
[debug message]   d = 0x27b2fc88e43cd310315874bfb80169e66f5da91ea62637f30a4fba6357c176b075156d59e3482f3d0e645f7138adae62501cce4fb267c68458c6ba47d32817ed
******** Alice's keys generated ********

****** Generating Bob's keys pair ******
[debug message]   p = 0x6703137173b4af04aec26192b98c14b3d7804bb854e0e25ee955c47c74e3f8f3
[debug message]   q = 0x9b2c351006975edcad1e431b0cd55a7f03b44d70267f045882579addb81f81bd
[debug message]   n = 0x3e70a6a71e6cf8829c37be1f885f57e972611515ad9159230afeb0ae1571ddf695ad864c8e4cf3ccb958a598216f7d40306263fdef003571c88fe119fa2e3e67
[debug message]   phi = 0x3e70a6a71e6cf8829c37be1f885f57e972611515ad9159230afeb0ae1571ddf5937e3dcb1400e5eb5d7800ea5b0e0e0d552dcad573a04eba5ce281bfcd2ac3b8
[debug message]   e = 0xf5def6ebf79b468d443110c70d3a7ac143b708fdb9e0d0cfca2c88da614ef48e78aa0fabfda5a49303768e681b5ba6ae51d3c492e54545c5846f52df
[debug message]   d = 0x4ec2c8208776970b8e1668d0928284d3234995a3ffffaad75208c6224ba12a50f484fb169ac343b7b8928204ddea0722be31ca063c5ff5f745dbd4824172b9f
********* Bob's keys generated *********

********** Exchanging pairs **********
********* Exchanging completed *********

[Message]: 0x37ac64c17227ab02d6d48530d661234165ddf08e946f6c517b014b4967c0d01a
[Alice -> Bob]: 0xbcb135ccdb0730a06fbdbd99e3a13d918d2c4857bb942924779f8becce06d5c7a131791ac1bdef76e2d29cbe01220952e2baa5b23afdffab229780131f9631f
[Alice sign]: 0x24f3e8eac90e581fad9a2b5c78b73118c981cfd5c7c53cd274e6aa1fbfd71d0155183b0b9ea644a537a9366339c3e54856e1df5ae354b098c2b57027600c834d
[Bob decrypted]: 37ac64c17227ab02d6d48530d661234165ddf08e946f6c517b014b4967c0d01a
[Message verified]: True
[Message]: 0x4297c84c50152961aaa8ec0c91d9b2f080e50e452947c6e9ee169515391de8ed
[Bob -> Alice]: 0x2e7f0c9368eb1a2e605ad2fe69ea00a1b4009d74ca5bad2de76cce62db4378655ce59afdace66f3d487b61c70fb1172be7b823ddb31a7f164d51de4ea347d72
[Bob sign]: 0x99aa127d90a87c9b233852eb4a76f769e37c02fba58b0911f0c5807d6dffb68dfd2ade45e4d5ce8a6c39c65fc6282e636f0046f2256931aab63f4fb0673e7c
[Alice decrypted]: 4297c84c50152961aaa8ec0c91d9b2f080e50e452947c6e9ee169515391de8ed
[Message verified]: True
```

Опис протоколу обміну ключами. Абонент А передає відкритий ключ абоненту Б (get_pubkey — передача від абонента до абонента), а абонент Б записує ключ абонента А в своє сховище ключів (set_key — збереження ключа до свого сховища відкритих ключів). Потім абонент Б передає ключ абоненту А.

Висновок

Криптосистема RSA — асиметрична криптосистема, що дозволяє шифрувати повідомлення та робити цифровий підпис. В даній лабораторній роботі я ознайомився з принципом роботи цієї криптосистеми, а також дізнався про тести простоти числа.

Пара ключів генерується таким чином:

1. Генеруємо два прості числа p і q
2. Обчислюємо $n = pq$
3. Обчислимо $\varphi(n) = (p-1)(q-1)$

p, q та φ зберігаємо у секреті

4. Обираємо випадкове число $e < \varphi-1$, $gcd(e, \varphi) = 1$
5. Рахуємо $d = e^{-1} \bmod \varphi$

Пара значень (e, n) — відкритий ключ, що використовується для шифрування та перевірки цифрового підпису.

Пара значень (d, n) — закритий ключ, що використовується для розшифрування та створення цифрового підпису.