



Міністерство освіти і науки України

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Фізико-технічний інститут

ЛАБОРАТОРНА РОБОТА №5

з дисципліни

«Криптографія»

на тему:

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

Виконали:

студенти 3 курсу ФТІ

групи ФБ-71

Рейценштейн Кирило і Таран Вікторія

Перевірили:

Чорний О.

Савчук М. М.

Завадська Л. О.

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:

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 , і – 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 ключі, та секретні.
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.

Результати:

Пара p і q , яка була сформована для абонента A :

p : C105BF84A1B147B7E14EDE667D6E9156FDC23812841D53FD4EDBF029E644CEB7

q : E46C0CBA5418AD84A6310DB5BACE13038E6876D34929A56BAB64069038BF7D53

Пара p і q , яка була сформована для абонента B :

p : A10257A4624B2513CC4B95A582DEA714E4013D91B3EFD33F30460E1C41A98513

q : FE102409F2D4E2DFF402159ECE139BD430F7DEC30BF417F667498EEB7602D83F

Експонента, модуль та секретний ключ для абонента A :

e:

569B6C5954B90B3AB9F8B1FE8E5A784233B76CD155A1B9DB63DDCEC596644D2B1ECDB41EB55740DE
C053995AD59AF4755B57719AAEDD9B3399FA963D453C5A69

n:

AC3A9697AE85F6D2309771F1B698A168EEB7F3E136B101D8C58709FF543A3FA1342AA686365A31D7F
53E23B9C7DA47322095B8296D9632AC6F3DBE8383C76055

d:

4CD7410005E121A9ECA64A875D9809243F58367576B262DA51DE4D5B5EC38416E2A855D56D7EE1A7
72BB906BCDF87D705188BAD03C1E5AC048DECB0C81F0469

Експонента, модуль та секретний ключ для абонента В:

e:

2C4A10F429C122CE7B4D2A48E2884A00FE0BD2345C442F095284DC2B1B0AF7FDDC94B353C29CEA73E
5CDF4908FA10D56A442D300DF0BA9D457787FFE2DCAD7B9

n:

9FCA79C529EF0190E0B84B0632D138B8CCF8F80E277689B22C5C7014ABED5183D7EE67AD75CBEF1DC
FA3DB4C4D8A06FEFD4CC42CF4941BD5057AA975FD25C7AD

d:

3025C337A4B503E021444611D56FBB0B661258B3881AFAE86AE56601D6CCA3224D6F0808331B215B0
7DDD73F806BACF61B6798D168ACC78135C2D6F077BB2831

Відкритий текст: 1337 (в байтах)

Шифрований текст ключем абонента А:

56A48ED99DF33B93D51A44AB4CDC77F7AF37CBE2D0BB28D47FDF717927406D49C8168B6C30FE1F4E
B41D37235309B0968FB7DFD3161766AE8F56487B0A1F151

Шифрований текст ключем абонента В:

CDA542D3577578A706BAB58E1B473E6C3C80E78140184F97BCCBD8C88475ECA4DE23C79C274894484
98060C9164F5C8F1A045764BEDD2D38F7D7EBB1B5F97DF

Підпис згенерований ключем абонента А:

7E0375E87FCE9BEB4413CE46C0F45F22B6A4E8DCFA1EF29B8E4010C8C05ADB38000F4ED6EAAF8940CF
B9C6B6048E256D4C7CC3C953A2D97AC231596F9B3869B7

Підпис згенерований ключем абонента В:

6BFC4F295E2F8E95201992F48CE74F363DDF0AB3A4B5D45B2789F59D1855761D2DA1F2529DC896469
47CE104C37FB12AEE6B5ED49363CA1470F4AAD6A2550B5F

Ключ: 16384

Шифрований підпис:

10679726BD3085E44081BCE61DFDD1B448584175A3A3225971D3B974F264A98AB6D539EA8F64AAFB
94198649578FDA95623E974C6A20548C86B2F0186F6C8BA6

Шифрований ключ:

2F1E373DA18D4187E208EA8B849F1E4CC2D5D98B984B4788752F4D3758B4CC0D0198450445ADF0202
E59F444667D3FE9DEA516F157CC0739EC8CF2E2917F5DC6

RSA Testing Environment

Server Key

Encryption

Decryption

Signature

Verification

Send Key

Receive Key

Verify

Clear

Message

1337

Bytes

Signature

7E0375E87FCE9BEB4413CE46C0F45F22B6A4E8DCFA1EF29B8E4010C8C05ADB38000F4ED6EAAF8940CFB

Modulus

AC3A9697AE85F6D2309771F1B698A168EEB7F3E136B101D8C58709FF543A3FA1342AA686365A31D7F53E2

Public exponent

569B6C5954B90B3AB9F8B1FE8E5A784233B76CD155A1B9DB63DDCEC596644D2B1ECDB41EB55740DEC05

Verify

Verification

true

Код:

Source.cpp

<pre>#include <iostream> #include <fstream> #include "RSA256.h" int main() { RSA256* crypto_provider = new RSA256(); std::vector<mp::cpp_int> first_pair, second_pair; crypto_provider->GenerateKeyPair(mp::pow(mp::cpp_int(2), 255), mp::pow(mp::cpp_int(2), 256), &first_pair, &second_pair); for (int i = 0; i < 3; i++) { std::cout << std::hex << std::uppercase << "Abonent A keys: " << first_pair[i] << std::endl; } for (int i = 0; i < 3; i++) { std::cout << "Abonent B keys: " << second_pair[i] << std::endl; } std::cout << std::endl; auto result = crypto_provider->Encrypt(0x1337, first_pair[0], first_pair[2]); std::cout << "Encryption with A keys: " << result << std::endl;</pre>	<pre>mp::cpp_int key = 0x16384; mp::cpp_int encrypted_sign = 0; mp::cpp_int encrypted_key = 0; std::cout << "Our secret decrypted key to send: " << key << std::endl; if (first_pair[2] <= second_pair[2]) { encrypted_key = crypto_provider- >SendKey(key, first_pair[1], first_pair[2], second_pair[0], second_pair[2], &encrypted_sign); std::cout << "Encrypted sign: " << encrypted_sign << std::endl << "Encrypted key: " << encrypted_key << std::endl; auto recieve_result = crypto_provider- >ReceiveKey(encrypted_key, encrypted_sign, second_pair[1], second_pair[2], first_pair[0], first_pair[2]); if (recieve_result) { std::cout << "Receiving the key returned true." << std::endl; } else { std::cout << "Receiving the key returned false." << std::endl; } }</pre>
---	--

<pre> result = crypto_provider->Decrypt(result, first_pair[1], first_pair[2]); std::cout << "Decryption with A keys: " << result << std::endl; result = crypto_provider->Encrypt(0x1337, second_pair[0], second_pair[2]); std::cout << "Encryption with B keys: " << result << std::endl; result = crypto_provider->Decrypt(result, second_pair[1], second_pair[2]); std::cout << "Decryption with B keys: " << result << std::endl; std::cout << std::endl; auto sign_result = crypto_provider->Sign(0x1337, first_pair[1], first_pair[2]); std::cout << "Sign result with A keys: " << sign_result << std::endl; std::cout << "Verify sign returned " << crypto_provider- >Verify(sign_result, first_pair[0], first_pair[2]) << std::endl; sign_result = crypto_provider->Sign(0x1337, second_pair[1], second_pair[2]); std::cout << "Sign result with B keys: " << sign_result << std::endl; std::cout << "Verify sign returned " << crypto_provider- >Verify(sign_result, second_pair[0], second_pair[2]) << std::endl; std::cout << std::endl; </pre>	<pre> else { encrypted_key = crypto_provider- >SendKey(key, second_pair[1], second_pair[2], first_pair[0], first_pair[2], &encrypted_sign); std::cout << "Encrypted sign: " << encrypted_sign << std::endl << "Encrypted key: " << encrypted_key << std::endl; auto recieve_result = crypto_provider- >ReceiveKey(encrypted_key, encrypted_sign, first_pair[1], first_pair[2], second_pair[0], second_pair[2]); if (recieve_result) { std::cout << "Receiving the key returned true." << std::endl; } else { std::cout << "Receiving the key returned false." << std::endl; } system("pause"); return true; } </pre>
--	--

Rsa.h

<pre> #pragma once #include <boost/random/random_device.hpp> #include <boost/random.hpp> #include <boost/multiprecision/cpp_int.hpp> #include <boost/integer/extended_euclidean.hpp> namespace mp = boost::multiprecision; namespace random = boost::random; namespace euclidean = boost::integer; class RSA256 { public: RSA256() {} ~RSA256() {} void GenerateKeyPair(mp::cpp_int range_min, mp::cpp_int range_max, std::vector<mp::cpp_int>* first_pair, std::vector<mp::cpp_int>* second_pair) { random::uniform_int_distribution<mp::cpp_int> get_number(range_min, range_max); mp::cpp_int primes[4]; int k = 0; std::ofstream not_primes("not_primes.txt"); while (k != 4) { auto secret_number = get_number(generator); if (this->MillerRabin(secret_number)) { primes[k++] = secret_number; } else { </pre>	<pre> mp::cpp_int Sign(mp::cpp_int message, mp::cpp_int d, mp::cpp_int n) { return mp::powm(message, d, n); } mp::cpp_int Verify(mp::cpp_int signed_message, mp::cpp_int e, mp::cpp_int n) { return mp::powm(signed_message, e, n); } mp::cpp_int SendKey(mp::cpp_int key, mp::cpp_int d1, mp::cpp_int n1, mp::cpp_int e2, mp::cpp_int n2, mp::cpp_int* encrypted_sign) { auto signed_result = this->Sign(key, d1, n1); auto encrypted_sign_result = this- >Encrypt(signed_result, e2, n2); auto encrypted_key_result = this->Encrypt(key, e2, n2); *encrypted_sign = encrypted_sign_result; return encrypted_key_result; } bool ReceiveKey(mp::cpp_int encrypted_key, mp::cpp_int encrypted_sign, mp::cpp_int d2, mp::cpp_int n2, mp::cpp_int e1, mp::cpp_int n1) { auto decrypted_key_result = this- >Decrypt(encrypted_key, d2, n2); auto decrypted_sign_result = this- >Decrypt(encrypted_sign, d2, n2); auto key = this->Verify(decrypted_sign_result, e1, n1); if (key == decrypted_key_result) { return true; } </pre>
---	---

<pre> not_primes << secret_number << std::endl; } } not_primes.close(); n1 = primes[0] * primes[1]; n2 = primes[2] * primes[3]; mp::cpp_int euler1 = (primes[0] - 1) * (primes[1] - 1); random::uniform_int_distribution<mp::cpp_int> get_number2(2, euler1 - 1); while (true) { auto e1_temp = get_number2(generator); auto euc_result = euclidean::extended_euclidean(e1_temp, euler1); if (euc_result.gcd == 1) { e1 = e1_temp; d1 = mod(mp::cpp_int(euc_result.x), euler1); break; } } mp::cpp_int euler2 = (primes[2] - 1) * (primes[3] - 1); random::uniform_int_distribution<mp::cpp_int> get_number3(2, euler2 - 1); while (true) { auto e2_temp = get_number3(generator); auto euc_result = euclidean::extended_euclidean(e2_temp, euler2); if (euc_result.gcd == 1) { e2 = e2_temp; d2 = mod(mp::cpp_int(euc_result.x), euler2); break; } } first_pair->push_back(e1); first_pair->push_back(d1); first_pair->push_back(n1); second_pair->push_back(e2); second_pair->push_back(d2); second_pair->push_back(n2); } mp::cpp_int Encrypt(mp::cpp_int message, mp::cpp_int e, mp::cpp_int n) { return mp::powm(message, e, n); } mp::cpp_int Decrypt(mp::cpp_int encrypted_message, mp::cpp_int d, mp::cpp_int n) </pre>	<pre> return false; } private: mp::cpp_int mod(mp::cpp_int a, mp::cpp_int b) { return (a % b + b) % b; } bool MillerRabin(mp::cpp_int secret_number) { if (secret_number % 2 == 0) { return false; } mp::cpp_int d = secret_number - 1; while (d % 2 == 0) { d /= 2; } // todo, not full number is used mp::cpp_int s(log((secret_number.convert_to<double>()) - 1) / d.convert_to<double>())); // random::uniform_int_distribution<mp::cpp_int> get_number(2, secret_number - 2); for (int i = 0; i < miller_constant; i++) { auto a = get_number(generator); mp::cpp_int x = mp::powm(a, d, secret_number); if (x == 1 x == secret_number - 1) { continue; } for (mp::cpp_int i = 0; i < s - 1; i++) { x = (x * x) % secret_number; if (x == 1) { return false; } else if (x == secret_number - 1) { break; } } if (x != secret_number - 1) { return false; } } return true; } // number out of my mind const int miller_constant = 9; // random::random_device generator; </pre>
---	--

<pre>{ return mp::powm(encrypted_message, d, n); }</pre>	<pre>mp::cpp_int e1 = 0, d1 = 0, n1 = 0, e2 = 0, d2 = 0, n2 = 0; };</pre>
--	---

Висновки:

Під час данного комп'ютерного практикуму, ми ознайомились з тестами перевірки чисел на простоту і методами генерації ключів для асиметричної криптосистеми типу RSA та з системою захисту інформації на основі криптосхеми RSA.