

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-73

Лень Олександр та Мухамедзянов Артем

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 *p*, *q* і *p1*, *q1* довжини щонайменше 256 біт. При цьому пари чисел беруться так, щоб *p q* ≤ *p1 q1* ; p і q – прості числа для побудови ключів абонента А, *p1* 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 А і B – тобто, створити та зберегти для подальшого використання відкриті ключі (e, n), (e1, n1) та секретні d і d1 .
4. Написати програму шифрування, розшифрування і створення повідомлення з цифровим підписом для абонентів *А* 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 *А* і *B*, перевірити правильність розшифрування. Скласти для *А* і *B* повідомлення з цифровим підписом і перевірити його.

1. 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.

# Результати

Message:

k = 1337

User 1 private key:

d1:

3370066919122603260197072199434045818047424695460760714746875693958765526353472938123670468544813955823816514911741178038701376499047523050393313405752373

User 1 public key:

n1:

5505772795177316461760825644629416945715419974234373051535983855290430420058175003639032677922459110863685035768634028234731223303400497988001342106914557

e1:

65537

User 2 private key:

d2:

1553700370100475445170083842987128671311432752474062625216317611693686695330560206657059719675078126623662986988208589649266674889556226231586166980485473

User 2 public key:

n2:

10126788777252596643472082030616355219466670144096732199781383124571769761499846527986585526323381346979905773178745397936677896357209776095523866872405383

e2:

65537

Signature: 4055771654622614521589738805452735098606279460896399415750045953717888063180624007299516172795056534020034165906354267070095013557109895254779550598606612

# Код програми

***rsa.c***

#include <stdio.h>

#include "gmp.h"

#include <time.h>

#include <math.h>

#include <stdlib.h>

static mpz\_t s;

static mpz\_t d;

static mpz\_t one;

static int k = 0;

static float saved = 0;

gmp\_randstate\_t state;

static mp\_bitcnt\_t bits;

static mpz\_t a;

static mpz\_t j;

static mpz\_t temp;

static mpz\_t n\_min;

int check\_often(mpz\_t n)

{

int ret;

ret = !mpz\_cdiv\_ui(n, 3) || !mpz\_cdiv\_ui(n, 5) || !mpz\_cdiv\_ui(n, 7) ||

!mpz\_cdiv\_ui(n, 11)|| !mpz\_cdiv\_ui(n, 13);

return (ret);

}

static void clear\_mpz(void)

{

mpz\_clear(s);

mpz\_clear(d);

mpz\_clear(one);

mpz\_clear(a);

mpz\_clear(j);

mpz\_clear(temp);

mpz\_clear(n\_min);

}

int miller\_rabin(mpz\_t s, mpz\_t d, mpz\_t n, int count)

{

mpz\_div\_ui(n\_min, n, 1);

while (count--)

{

mpz\_urandomb(a, state, bits);

mpz\_setbit(a, bits);

mpz\_powm(a, a, d, n);

if (!mpz\_cmp\_ui(a, 1) || !mpz\_cmp(a, n\_min))

continue;

mpz\_set\_ui(j, 1);

while (mpz\_cmp(j, s) < 0)

{

mpz\_powm\_ui(a, a, 2, n);

if (mpz\_cmp\_ui(a, 1) == 0)

return (0);

if (mpz\_cmp(a, n\_min) == 0)

break ;

mpz\_add\_ui(j, j, 1);

}

if (mpz\_cmp(j, s) == 0)

return (0);

}

return (1);

}

int ft\_ssl\_is\_primary(mpz\_t n, float prob)

{

if (saved != prob)

{

k = (int)(log10(1 - prob) / log10(0.25));

saved = prob;

}

if (check\_often(n))

return (0);

mpz\_set\_ui(one, 1);

mpz\_set\_ui(s, 0);

mpz\_sub\_ui(d, n, 1);

while (1)

{

mpz\_and(temp, d, one);

if (mpz\_cmp\_ui(temp, 0) != 0)

break ;

mpz\_add\_ui(s, s, 1);

mpz\_div\_ui(d, d, 2);

}

return (miller\_rabin(s, d, n, k));

}

void iterate\_and\_check(mpz\_t n, mpz\_t max\_of\_bits, int size, mp\_bitcnt\_t prev\_bits)

{

float prob;

bits = prev\_bits;

mpz\_init(j);

mpz\_init(s);

mpz\_init(d);

mpz\_init(temp);

mpz\_init(one);

mpz\_init(n\_min);

mpz\_init(a);

prob = 1 - 0.000001 \* size;

while (mpz\_cmp(n, max\_of\_bits) < 0)

{

if (ft\_ssl\_is\_primary(n, prob))

break ;

mpz\_add\_ui(n, n, 2);

}

clear\_mpz();

if (mpz\_cmp(n, max\_of\_bits) >= 0)

{

mpz\_urandomb(n, state, bits);

iterate\_and\_check(n, max\_of\_bits, size, prev\_bits);

}

}

void calculate\_variables(mpz\_t module, mpz\_t pub, mpz\_t priv, mpz\_t first, mpz\_t second)

{

mpz\_set\_str(pub, "10001", 16);//set pub exp 65537

mpz\_mul(module, first, second);//calculating module

mpz\_t euler\_mul;

mpz\_t euler1;

mpz\_t euler2;

mpz\_init(euler1);

mpz\_init(euler2);

mpz\_init(euler\_mul);

mpz\_sub\_ui(euler1, first, 1);

mpz\_sub\_ui(euler2, second, 1);

mpz\_mul(euler\_mul, euler1, euler2);

mpz\_invert(priv, pub, euler\_mul);

gmp\_printf("module = %Zx\npublic exp = %Zx\npriv = %Zx\n", module, pub, priv);

}

void encode(mpz\_t mes, mpz\_t pub, mpz\_t mod)

{

mpz\_powm(mes, mes, pub, mod);

}

void decode(mpz\_t mes, mpz\_t priv, mpz\_t mod)

{

mpz\_powm(mes, mes, priv, mod);

}

void sign(mpz\_t mes, mpz\_t priv, mpz\_t mod)

{

mpz\_powm(mes, mes, priv, mod);

}

void verify(mpz\_t mes, mpz\_t pub, mpz\_t mod, mpz\_t copy)

{

mpz\_powm(mes, mes, pub, mod);

if (mpz\_cmp(mes, copy) == 0)

printf("verification done\n");

else

printf("verification failed\n");

}

int main(int argc, char \*\*argv)

{

if (argc != 2)

return (printf("Usage: %s [num\_of\_bits]\n", argv[0]) && 0);

int num\_bits = atoi(argv[1]);

printf("num of bits = %d\n", num\_bits);

if (num\_bits < 256)

return (printf("num\_bits can be only >255\n") && 0);

mpz\_t first;

mpz\_t second;

mpz\_t third;

mpz\_t fourth;

mpz\_t maximum;// maximum of this bits

mpz\_init(first);

mpz\_init(second);

mpz\_init(third);

mpz\_init(fourth);

mpz\_init(maximum);

gmp\_randinit\_mt(state);//initialize random number generator

gmp\_randseed\_ui(state, time(NULL));

mpz\_urandomb(first, state, num\_bits / 2);

mpz\_urandomb(second, state, num\_bits / 2);

mpz\_urandomb(third, state, num\_bits / 2);

mpz\_urandomb(fourth, state, num\_bits / 2);

// stavim 1 v 0 i posledniy bit

mpz\_setbit(first, num\_bits / 2 - 1);

mpz\_setbit(second, num\_bits / 2 - 1);

mpz\_setbit(third, num\_bits / 2 - 1);

mpz\_setbit(fourth, num\_bits / 2 - 1);

bits = num\_bits / 2;

mpz\_setbit(first, 0);

mpz\_setbit(second, 0);

mpz\_setbit(third, 0);

mpz\_setbit(fourth, 0);

gmp\_printf("randomly generated\n%Zd\n%Zd\n%Zd\n%Zd\nStart searching primes\n", first, second, third, fourth);

mpz\_urandomb(maximum, state, bits);

for (int i = 0; i < num\_bits / 2; i++)

mpz\_setbit(maximum, i);

iterate\_and\_check(first, maximum, num\_bits / 2, bits);

iterate\_and\_check(second, maximum, num\_bits / 2, bits);

iterate\_and\_check(third, maximum, num\_bits / 2, bits);

iterate\_and\_check(fourth, maximum, num\_bits / 2, bits);

gmp\_printf("primes are\n%Zd\n%Zd\n%Zd\n%Zd\n", first, second, third, fourth);

mpz\_t module1;

mpz\_t pub1;

mpz\_t priv1;

mpz\_t module2;

mpz\_t pub2;

mpz\_t priv2;

mpz\_init(module1);

mpz\_init(pub1);

mpz\_init(priv1);

mpz\_init(module2);

mpz\_init(pub2);

mpz\_init(priv2);

calculate\_variables(module1, pub1, priv1, first, second);

calculate\_variables(module2, pub2, priv2, third, fourth);

//Encoding and decoding message

mpz\_t message;

mpz\_t copy;

mpz\_init(message);

mpz\_init(copy);

mpz\_urandomb(message, state, bits);

mpz\_set(copy, message);

gmp\_printf("generated Random message = %Zd\n", message);

encode(message, pub1, module1);

gmp\_printf("Encoded message = %Zd\n", message);

decode(message, priv1, module1);

gmp\_printf("Decoded message = %Zd\n", message);

sign(message, priv1, module1);

verify(message, pub1, module1, copy);

}

***site.c***

#include <stdio.h>

#include "gmp.h"

#include <time.h>

#include <math.h>

#include <stdlib.h>

void encode(mpz\_t mes, mpz\_t pub, mpz\_t mod)

{

mpz\_powm(mes, mes, pub, mod);

}

void decode(mpz\_t mes, mpz\_t priv, mpz\_t mod)

{

mpz\_powm(mes, mes, priv, mod);

}

void sign(mpz\_t mes, mpz\_t priv, mpz\_t mod)

{

mpz\_powm(mes, mes, priv, mod);

}

void verify(mpz\_t mes, mpz\_t pub, mpz\_t mod, mpz\_t copy)

{

mpz\_powm(mes, mes, pub, mod);

if (mpz\_cmp(mes, copy) == 0)

printf("verification done\n");

else

printf("verification failed\n");

}

void receive\_key(mpz\_t module, mpz\_t my\_module, mpz\_t my\_private, mpz\_t Key, mpz\_t Sign)

{

mpz\_t pub;

mpz\_init(pub);

mpz\_set\_str(pub, "10001", 16);

decode(Key, my\_private, my\_module);

decode(Sign, my\_private, my\_module);

verify(Sign, pub, module, Key);

gmp\_printf("Key is %Zx\nSign is %Zx\n", Key, Sign);

}

void send\_key(mpz\_t module, mpz\_t my\_module, mpz\_t my\_private, mpz\_t Message)

{

mpz\_t pub;

mpz\_init(pub);

mpz\_set\_str(pub, "10001", 16);

mpz\_t Sign;

mpz\_init(Sign);

mpz\_set(Sign, Message);

sign(Sign, my\_private, my\_module);

encode(Sign, pub, module);

encode(Message, pub, module);

gmp\_printf("Key is %Zx\nSign is %Zx\n", Message, Sign);

}

int main(int argc, char \*\*argv)

{

if (argc < 2)

return (printf("Usage: %s flag [modulus] [args]\n", argv[0]) && 0);

if (atoi(argv[1]) == 1)

{

if (argc != 7)

return (printf("Usage: %s [modulus] [Key] [Signature] [my\_module] [my\_private]\n", argv[0]) && 0);

mpz\_t module;

mpz\_init(module);

mpz\_set\_str(module, argv[2], 16);

mpz\_t my\_module;

mpz\_init(my\_module);

mpz\_set\_str(my\_module, argv[5], 16);

mpz\_t my\_private;

mpz\_init(my\_private);

mpz\_set\_str(my\_private, argv[6], 16);

mpz\_t Key;

mpz\_init(Key);

mpz\_set\_str(Key, argv[3], 16);

mpz\_t Sign;

mpz\_init(Sign);

mpz\_set\_str(Sign, argv[4], 16);

receive\_key(module, my\_module, my\_private, Key, Sign);

}

if (atoi(argv[1]) == 2)

{

printf("%d\n", argc);

if (argc != 6)

return (printf("Usage: %s [modulus] [my\_modulus] [my\_private] [message]", argv[0]) && 0);

mpz\_t module;

mpz\_init(module);

mpz\_set\_str(module, argv[2], 16);

mpz\_t my\_module;

mpz\_init(my\_module);

mpz\_set\_str(my\_module, argv[3], 16);

mpz\_t my\_private;

mpz\_init(my\_private);

mpz\_set\_str(my\_private, argv[4], 16);

mpz\_t Message;

mpz\_init(Message);

mpz\_set\_str(Message, argv[5], 10);

send\_key(module, my\_module, my\_private, Message);

}

}

# Висновки:

Під час данного комп’ютерного практикуму, ми ознайомились з тестами перевірки чисел на простоту і методами генерації ключів для асиметричної криптосистеми типу RSA. Також, використовуючи криптосистему типу RSA, організували канал засекреченого зв’язку й електронний підпис, ознайомились із протоколом розсилання ключів.