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 p_1, q_1 довжини щонайменше 256 біт. При цьому пари чисел беруться так, щоб $p\,q \le p_1\,q_1$; p і q – прості числа для побудови ключів абонента А, p_1 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), (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().

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:
337006691912260326019707219943404581804742469546076071474687569395876552635347293812367046854481395582381651491174117803870137649904752305039331340575237 3

User 1 public key:
n1:
550577279517731646176082564462941694571541997423437305153598385529043042005817500363903267792245911086368503576863402823473122330340049798800134210691455 7
e1:
65537

User 2 private key:
d2:
155370037010047544517008384298712867131143275247406262521631761169368669533056020665705971967507812662366298698820858964926667488955622623158616698048547 3

User 2 public key:
n2:
101267887772525966434720820306163552194666701440967321997813831245717697614998465279865855263233813469799057731787453979366778963572097760955238668724053 83
e2:
65537

Signature:
405577165462261452158973880545273509860627946089639941575004595371788806318062400729951617279505653402003416590635426707009501355710989525477955059860661 2

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.