Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Фізико-технічний інститут

КРИПТОГРАФІЯ
КОМП’ЮТЕРНИЙ ПРАКТИКУМ №4

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

Виконали:

ФБ-31 Федорович Дарина

ФБ-31 Шваюк Олександра

Мета та основні завдання 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.

Генерація числа (generate_prime()) відбувається або за довжини 256 біт (дефолтне значення), або в інтервалі від найменшого до найбільшого числа з кількістю цифр 78 (що відповідає умові не менше 256 біт). Під час генерації проходять 2 умови: перевірка діленням на пробні ділення (if_prime_trial_div()) та сам тест Міллера-Рабіна (if_prime_mil_rab()). У випадку коли кандитат не проходить - генеруємо заново. Також важливою функцією є схема Горнера швидкого піднесення до степеня (horny_power()), яка була основою майже усіх інших функцій, це дуже сильна функція в ЛР.

2. За допомогою цієї функції згенерувати дві пари простих чисел p, q і p1,q1довжини щонайменше 256 біт. При цьому пари чисел беруться так, щоб pq <= p1 q1 ; p і q – прості числа для побудови ключів абонента A, p1 і q1 – абонента B.

Функція (generate_p_q()) генерує 4 числа: p q та p1 q1 такі, що виконується умова pq <= p1q1, інакше відкидає ці значення та генерує їх заново. Для демонстрації та тестування ми обрали згенеровані ключі які пройшли тести на простоту та умову множення та помістили їх в статичні змінні (згенеровані прості числа за довжиною 256 біт)

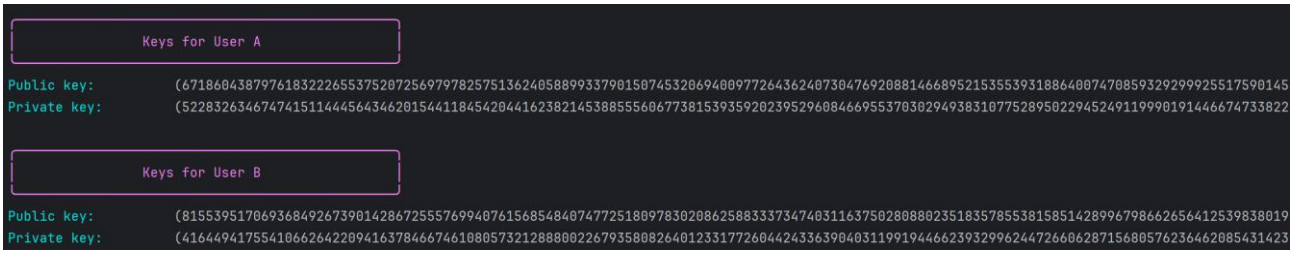
```
Generating Valid Numbers
These candidates are invalid:
p = 976077084984653007166434405981414984985783063991835233964728886131912506734 73
q = 947855554042428862091388793431734657417815233363780718637317899498109823849 83
p_1 = 651983533074824084815383120368716840448432554744268675856946554099751954878 47
q_1 = 645362667332365426818387722939823828205666562737874502399984137154792355474 63
These candidates are invalid:
p = 106245865823500997143878033906622073480216833241649498396471972466300243282433
q = 819498831399188560471410369686527162056538354106553002985160071405869431716 11
p_1 = 676888889409330146625370460727682486365714802027684113025423761381921800624 23
q_1 = 715393321444855919468704859589346903481260288004516239820867351265354882261 61
p * q <= p_1 * q_1: True
p: 736766981399830045281214860115489744904311780115201078215954574948173761777 13
q: 111924287397058094732590000629393699019009276865391446581994015305929082019047
p_1: 109645489619571930811222827295949905908436159866454823732335133320266370027333
q_1: 103111118175564000188312415408721748648964048740319618241553227170339955091909
len of p_1: 256
len of q_1: 256
```

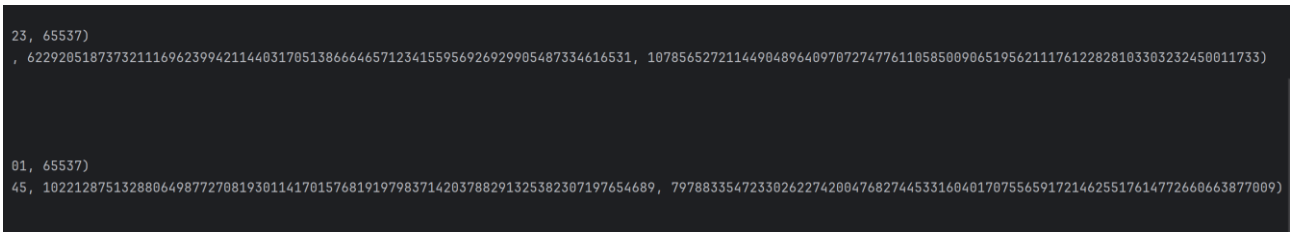
```
User A Static Numbers
p: 622920518737321116962399421144031705138666465712341559569269299054873346165 31 (len=256)
q: 107856527211449048964097072747761105850090651956211176122828103303232450011733 (len=256)

User B Static Numbers
p: 102212875132880649877270819301141701576819197983714203788291325382307197654689 (len=256)
q: 797883354723302622742004768274453316040170755659172146255176147726606638770 09 (len=256)

p * q <= p_1 * q_1 : True
```

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) , (e1, n1) та секретні d і d1.

Функція (GenerateKeyPair()) генерує там пару закритих (d, p, q) та відкритих ключів (n, e), використовуючи функцію Ойлера а також функції, написані у 3 ЛР (знаходження НСД та оберненого за модулем, е обрано яу в сучасних криптографічних системах, 2**16 + 1, але за бажанням також можна і згенерувати випадкове яке буде виконувати умову gcd(e, phi) = 1. Пара n, e - відкритий ключ, d - секретний ключ.

```
keys_a = GenerateKeyPair(p, q)
keys_b = GenerateKeyPair(p_1, q_1)

public_key_a, private_key_a = keys_a[1], keys_a[0]
public_key_b, private_key_b = keys_b[1], keys_b[0]

print_keys(public_key_a, private_key_a, user: "A")
print_keys(public_key_b, private_key_b, user: "B")
```





Вийшли дуже довгими (другий скрін це продовження правої частини першої). На цьому етапі ключі згенеровані.

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 М і знайти криптограму для абонентів А и В, перевірити правильність розшифрування. Скласти для А і В повідомлення з цифровим підписом і перевірити його.

Шифрування (Encrypt()) приймає повідомлення, яке повинно бути менше за модуль ключа, та публічний ключ, та використовує піднесення до степеня по модулю для шифрування.

Розшифрування (Decrypt()) приймає також повідомлення яке повинно задовольняти умову та приватний ключ, який використовується в піднесенні до степеня по модулю для розшифрування.

Цифровий підпис (Sign()) створюється як пара повідомлення та його значення-підпису за приватним ключем.

Перевірка підпису (Verify()) відбувається за наявності підписаної пари та публічного ключа.

Перевіримо чи працюють функції. Число M оберемо невелике для з.меншення навантаження. Приклади тривіальні для перевірки правильності без помилок.

```
# ---------- Encryption / Decryption ----------
M = 'Hello Mario'
print_section( title: "Testing Encryption/Decryption", CYAN)
print(f"Original Message: {M}")

cryptogram_a = Encrypt(M, public_key_b)
cryptogram_b = Encrypt(M, public_key_a)

decrypted_a = Decrypt(cryptogram_b, private_key_a, text=True)
decrypted_b = Decrypt(cryptogram_a, private_key_b, text=True)

print(f"Encrypted for B: {cryptogram_a}")
print(f"Encrypted for A: {cryptogram_b}")
print(f"Decrypted by A: {decrypted_a}")
print(f"Decrypted by B: {decrypted_b}")
```

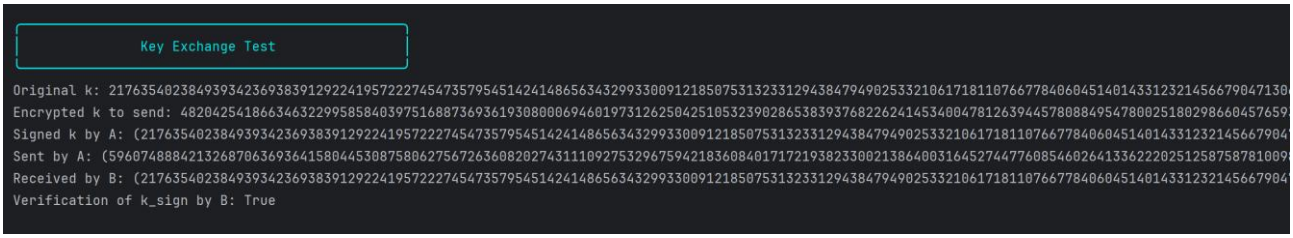
```
# ---------- Digital Signatures ----------
signed_a = Sign(M, private_key_a)
signed_b = Sign(M, private_key_b)

print_section( title: "Testing Signatures", PINK)
print(f"Signature by A: {signed_a}")
print(f"Signature by B: {signed_b}")

verification_a = Verify(signed_a, public_key_a)
verification_b = Verify(signed_b, public_key_b)

print(f"Verification for A: {verification_a}")
print(f"Verification for B: {verification_b}")
```

```
Testing Encryption/Decryption

Original Message: Hello Mario
Encrypted for B: 586003971784678356218393928879849218495465504997610438231968744729990406618172356212617241537046323486428671647665319844232567019089146511799068
Encrypted for A: 355549463997881972440880025341988072169710669455444539460409880808809263742359810618145363906869788903271436195773115318469635973695328528419912222
Decrypted by A: Hello Mario
Decrypted by B: Hello Mario

Testing Signatures

Signature by A: (875216180888824906075610 71, 542238963883972965976740612795340442475007949167712713232639816182266745694154701837331449527384030773076711775399581
Signature by B: (875216180888824906075610 71, 238822464629111614435803091166474437460747739954894144928615359124162864823597316281131409073497353524239117715286145
Verification for A: True
Verification for B: True
k is verified
```

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.

```
# ---------- Key Exchange ----------
k = random.randint( a: 0, public_key_a[0])
k_encrypted = Encrypt(k, public_key_a)
k_sign = Sign(k, private_key_a)
sent = SendKey(k, public_key_a, private_key_a, public_key_b)
received = ReceiveKey(sent, private_key_b, public_key_a)
b_verify = Verify(k_sign, public_key_a)
```



Перевірка підпису відбувається при отриманні ключа.

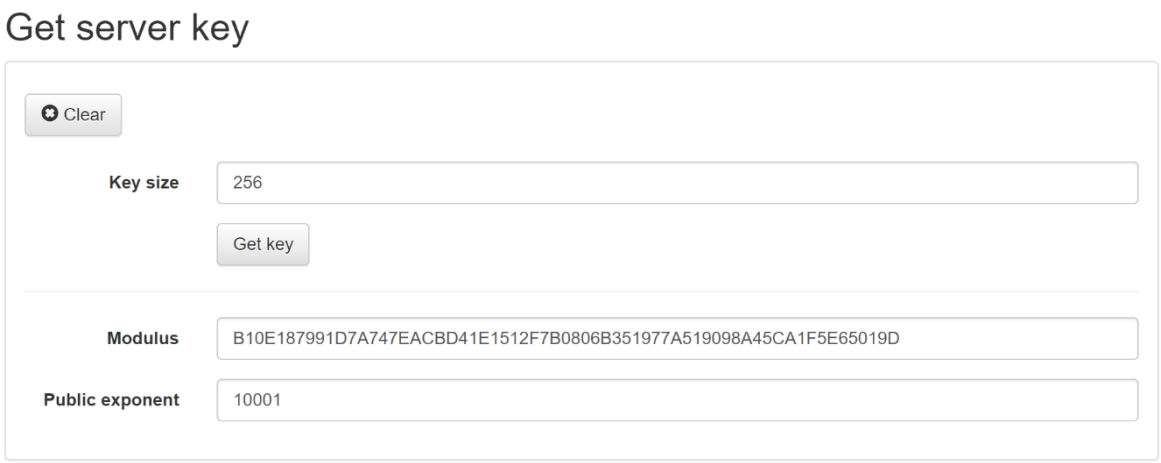
Кожна з наведених операцій повинна бути реалізована у вигляді окремої процедури, інтерфейс якої повинен приймати лише ті дані, які необхідні для її роботи; наприклад, функція Encrypt(), яка шифрує повідомлення для абонента, повинна приймати на вхід повідомлення та відкритий ключ адресата (і тільки його), повертаючи в якості результату шифротекст. Відповідно, програмний код повинен містити сім високорівневих процедур: GenerateKeyPair(), Encrypt(), Decrypt(), Sign(), Verify(), SendKey(), ReceiveKey()

Відповідні функції було реалізовано. У примітивних прикладах ми тестували числа. Також ми реалізували шифрування та дешифрування стрінгів через їх ASCII значення.

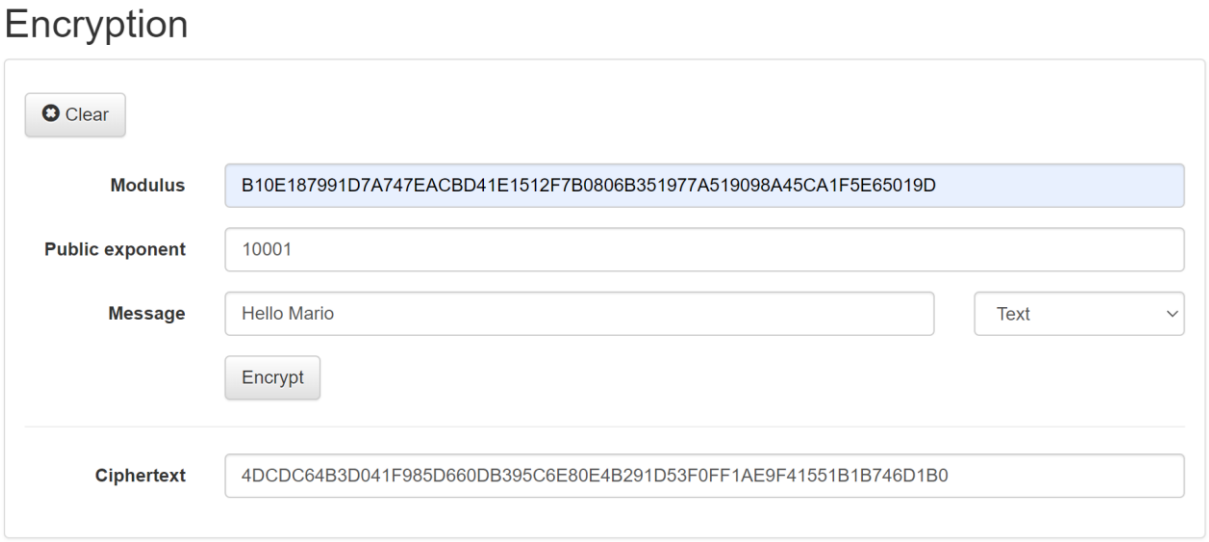
Кожну операцію рекомендується перевіряти шляхом взаємодії із тестовимсередовищем, розташованим за адресою

http://asymcryptwebservice.appspot.com/?section=rsa

Для початку генеруємо публічний ключ довжиною 256 біт на сервері



Зашифровуємо повідомлення на сервері



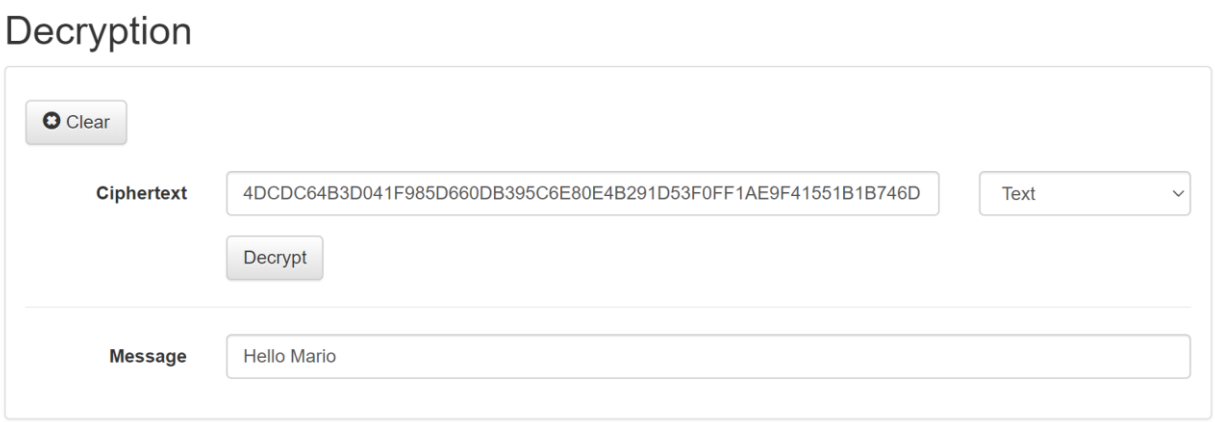
Перевіряємо у себе

```
pubkey_serv = (int("B10E187991D7A747EACBD41E1512F7B0806B351977A519098A45CA1F5E65019D", 16), int("10001", 16))
message = 'Hello Mario'

ciphertext_serv = int("4DCDC64B3D041F985D660DB395C6E80E4B291D53F0FF1AE9F41551B1B746D1B0", 16)
ciphertext_our = Encrypt(message, pubkey_serv)

print(f"{CYAN}Ciphertext encrypted by server:{RESET} {ciphertext_serv}")
print(f"{CYAN}Ciphertext encrypted by us:     {RESET} {ciphertext_our}")
num = 351916615633894913280667065562955231885012362574766252631767390618199160795536
```

```
┌──────────────────────────────────────┐
│            Server-Side Testing       │
└──────────────────────────────────────┘
Ciphertext encrypted by server: 351916615633894913280667065562955231885012362574766252631767390618199160795536
Ciphertext encrypted by us:     351916615633894913280667065562955231885012362574766252631767390618199160795536
Hex value: 0x4dcdc64b3d041f985d660db395c6e80e4b291d53f0ff1ae9f41551b1b746d1b0
```

Бачимо, що наша функція правильно його зашифрувала.

Тепер спробуємо наш шифротекст отриманий через ключ сервера розшифрувати на сервері.



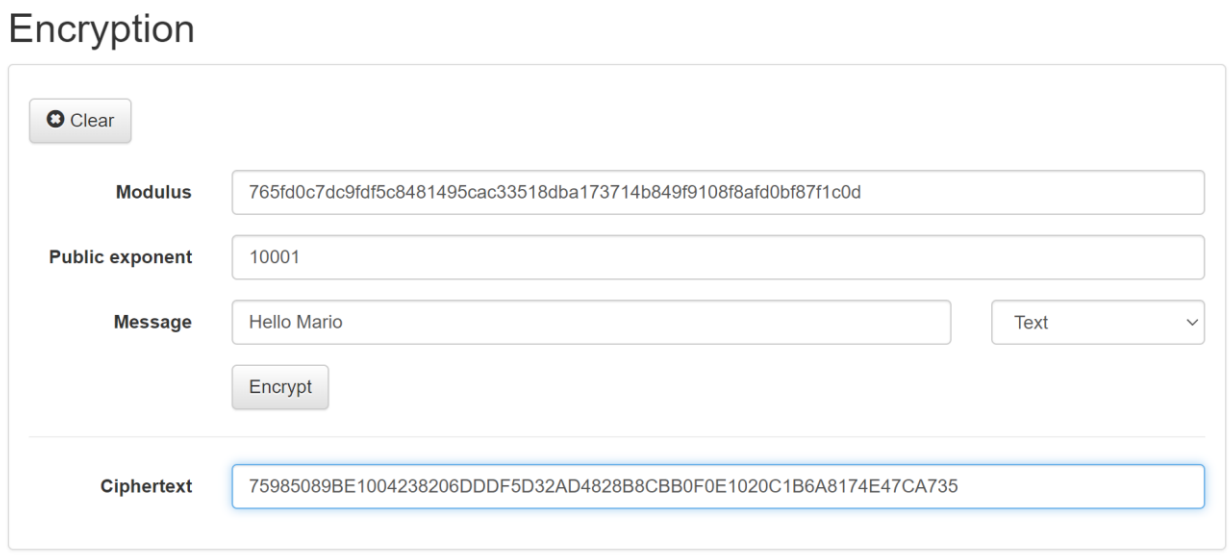
Отримали наш початковий текст.

Тепер передамо серверу наші відкриті ключі, та нехай він зашифрує в себе.

```
Public key a: (535422075554833714407701119326825463962135307936229886283655625549834375075 97, 65537)
Private key a: (505569743832640718200280274765429302318901294453523993996627176589138773201 17, 181207594698208181078258173571185530019, 295474412342678748021019
Public key a in hex: (0x765fd0c7dc9fdf5c8481495cac33518dba173714b849f9108f8afd0bf87f1c0d, 0x10001)
Private key a in hex: (0x6fc63bc36ced3d2104317c4f587ec0c2eb53881b0da90235889f2c6bde84e1b5, 0x88535032fc1c9d00c0490334088054a3, 0xde4a4d463ea2e343898ead497819278
```

Дамо серверу відкритий ключ А та зашифруємо текст.

У себе розшифровуємо.

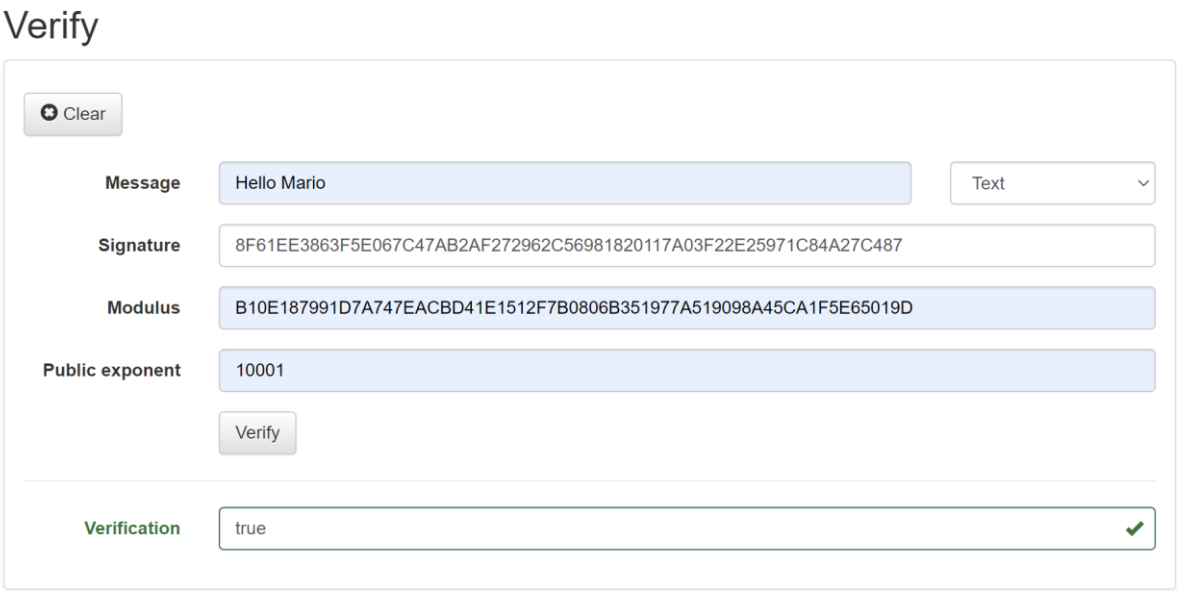



```
{Decrypt(int('75985089BE1004238206DDDF5D32AD4828B8CBB0F0E1020C1B6A8174E47CA735', 16), private_key_a, text=True)}")
```

```
Decrypted with our key:          Hello Mario
```

Тепер генеруємо підпис на сервері.

Та перевіримо його на сервері.



А тепер у нас

```
{Verify((message, int('8F61EE3863F5E067C47AB2AF272962C56981820117A03F22E25971C84A27C487', 16)), pubkey_serv)}"
```

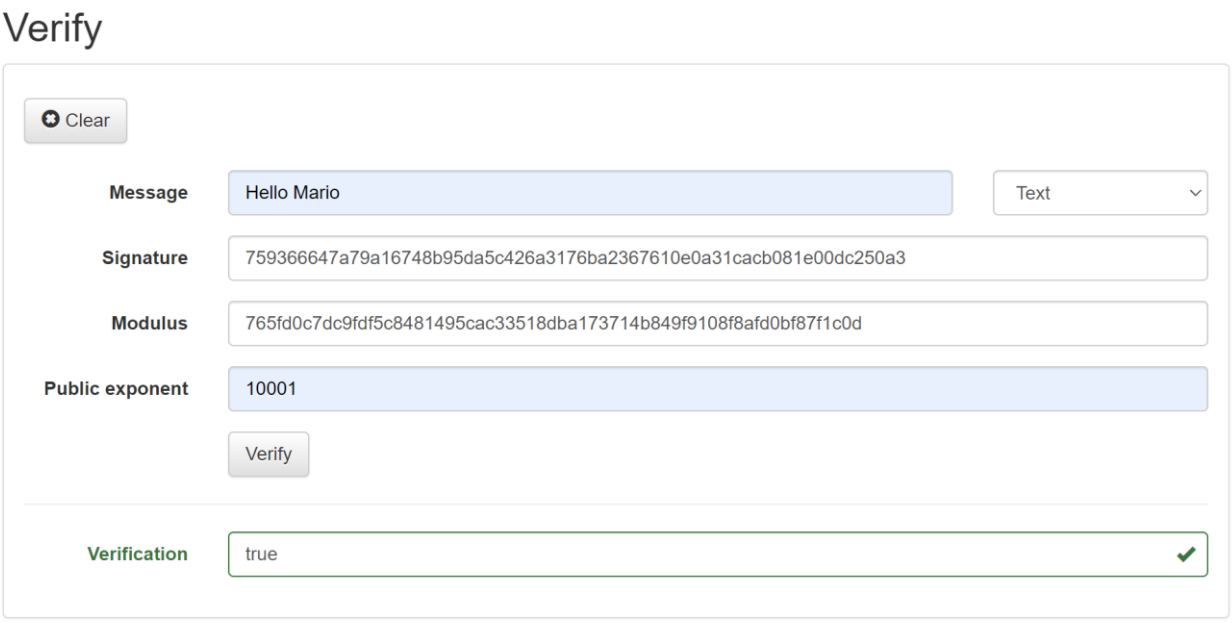
```
Server signature verified:        True
```

Тепер генеруємо підпис у нас

```
generated_sign = Sign(message, private_key_a)
print(f"{CYAN}Our generated signature (hex):  {RESET} {hex(generated_sign[1])}")
```

```
Our generated signature (hex):   0x759366647a79a16748b95da5c426a3176ba2367610e0a31cacb081e00dc250a3
```

Та перевіряємо на сервері



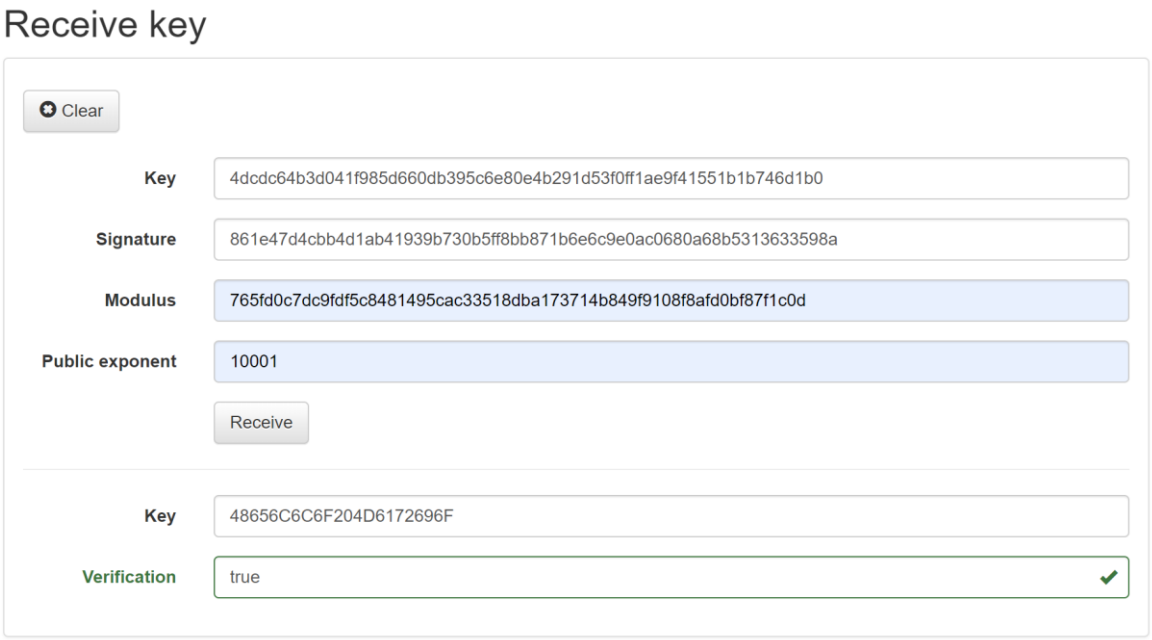
Підпис справився та підтвердився

Тепер відішлемо на сервері наш відкритий ключ A. Забула зробити скрін.

Перевіряємо

```
print(f"{CYAN}Received key:{RESET}")
received = ReceiveKey( received: (int('5B39213FD1B35D81CBD84937835B65EB3C4C1E275F19E015B2828D3E023E38D7', 16),
                       int('694883EE7AF784A5EA11EF55369384DFADB9015B9092F0B78DAED9D485653B4E', 16)),
                      private_key_a, pubkey_serv)
print(f"  {hex(received[0])},")
print(f"  {hex(received[1])}")

sent = SendKey(message, public_key_a, private_key_a, pubkey_serv)
print(f"{CYAN}Sent key:{RESET}")
print(f"  {hex(sent[0])},")
print(f"  {hex(sent[1])}")
```

```
k is verified
Received key:
k is verified
  0x232c18a1c09fa5c6,
  0x1665d9362b9f8e948b7a12502a9d8ee0cea08936e3c00bf73c30f7dbe333dcef
Sent key:
  0x4dcdc64b3d041f985d660db395c6e80e4b291d53f0ff1ae9f41551b1b746d1b0,
  0x861e47d4cbb4d1ab41939b730b5ff8bb871b6e6c9e0ac0680a68b5313633598a
```




```
key_ = 0x48656C6C6F204D6172696F
print(f"{CYAN}Key as text:{RESET} {key_.to_bytes((key_.bit_length() + 7) // 8, byteorder='big').decode('utf-8')}")
```

```
Key as text: Hello Mario
```

Програма успішно пройшла перевірки на сайті сервера

Висновки:

В ході виконання лабораторної роботи було вивчено роботу криптосистеми RSA та алгоритму електронного підпису та методами генерації параметрів для асиметричних криптосистем

Ми знову зрозуміли практичну важливість алгоритму Евкліда та лінійних конгруенцій, знаходження оберненого елементу за модулем.

Ключові етапи побудови криптосистеми RSA:

1. Написання допоміжних математичних функцій

2. Написання функції, що вирішує схему Горнера швидкого піднесення до степеня

3. Написання тестів перевірків натуральних чисел на простоту (Міллера-Рабіна)

4. Написання функції генерації простого числа

5. Написання функцій шифрування-дешифрування, цифрового підпису та

верифікації

6. Експлуатація функцій та використання криптосистеми

Одним з основних етапів побудови тестів на простоту є використання псевдопростих чисел. Зокрема, такі числа, що зберігають деякі властивости простих чисел використовуються у імовірнісному тесті Міллера-Рабіна. При виконанні лабораторної роботи було написано функції if_prime_mil_rab(), if_prime_trial_div() та generate_prime(), що слугують для генерації простих чисел. Для того, щоб просте число згенерували - кандидат має пройти 2 тести

Робота RSA відбувається наступним чином:

Абонент А генерує великі прості числа p та q та обчислює їх добуток та функцію Ойлера n=pq та φ(n) =(p-1)(q-1) відповідно.

Далі А обирає випадкове число e (зазвичай це 2**16 +1), НСД між числом e та φ повинно бути 1.

Далі знаходиться обернений за модулем елемент d, який і є секретним ключем абонента d. Пара чисел n та e - наш публічний ключ.

Зашифрування відкритого повідомлення відбувається формулою:

$$C = M^e \bmod n$$

А розшифрування за формулою:

$$M = C^d \bmod n$$

Відкрите повідомлення підписується користувачем А за формулою

$$S = M^d \bmod n$$

та цей підпис S додається до повідомлення, користувач В верифікує його за допомогою

відкритого ключа за формулою

$$M = S^e \bmod n$$

Розсилання та отримання повідомлень відбуваються наступним чином:

Абонент А має відкритий ключ (e,n) , секретний d і відкритий ключ (e1, n1) абонента В. Повідомлення (k1, S1) формується абонентом А та відправляється абоненту В за

формулою:

$$k_1 = k^{e_1} \bmod n_1, \quad S_1 = S^{e_1} \bmod n_1, \quad S = k^d \bmod n$$

Абонент В розшифровує повідомлення за допомогою свого секретного ключа та перевіряє підпис абонента А

$$k = k_1^{d_1} \bmod n_1, \quad S = S_1^{d_1} \bmod n_1 \qquad k = S^e \bmod n$$