КРИПТОГРАФІЯ
КОМП'ЮТЕРНИЙ ПРАКТИКУМ №4
Вивчення криптосистеми RSA та алгоритму електронного підпису; ознайомлення з методами генерації параметрів для асиметричних криптосистем

Мета

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

Побудувати криптосистему RSA і виконати в ній шифрування/розшифрування/підпис повідомлень, перевірку цифрового підпису повідомлення, конфіденційне розсилання ключів з підтвердженням справжності по відкритому каналу.

Хід роботи

Мовою програмування для створення скриптів було обрано Python 3. Було розроблено код ср4.py, який містить усі функції для виконання цього комп'ютерного практикуму.

У ході виконання роботи виникли несуттєві труднощі, загалом пов'язані з обрахуванням математичних функцій, збільшенням зручності використання застосунку і визначенням типу даних сервера RSA, наприклад:

- Деякий час функція **miller_rabin()**, яка перевіряє, чи є вхідне число простим, працювала неприпустимо повільно при розмірі вхідних даних понад 256 біт. З'ясувалось, причиною цього була некоректна робота **randint()** за умови, що верхня межа задана степенем двійки з використанням базового оператор "**".

  Розв'язання. Застосувати замість **randint()** іншу функцію бібліотеки random - **getrandbits()**. Вона дозволяє швидко згенерувати випадкове число заданої кількості біт.
- Спочатку код імпортувався в оболонку Python3 як бібліотека. Через те, що код працює з десятковими числами, а сервер - з шістнадцятковими, доводилося робити значну кількість ручних перетворень з однієї систему в іншу.

  Розв'язання. Не імпортувати код в оболонку, а створити функцію **RSA()**, що підтримувала цикл роботи за допомогою введення команд.
- На певному етапі розробка зупинилася через інтерпретацію значення "Public exponent" сервера як двійкове число.

  Розв'язання. Це число є шістнадцятковим.

Була розроблена функція **tests()**, яка містить тестування більшості функцій коду.

```
# Test all functions.
def tests() -> None:
    # Find gcd and inverses.
    print(f"""\n=== GCD & INVERSES TEST ===
euclid(155, 29) = {euclid(155, 29)}
euclid(2, -5) = {euclid(2, -5)}
```

```
euclid(-15, 40) = {euclid(-15, 40)}""")

    # Find euler function.
    print(f"""\n=== EULER FUNCTION TEST ===
phi(1) = {euler(1)}
phi(5) = {euler(5)}
phi(6) = {euler(6)}
phi(23147) (factorization is unknown) = {euler(23147)}
phi(23147) (factorization is given) = {euler(23147, p=79, q=293)}""")
    #print(f"phi(-1) = {euler(-1)}")

    # Calculate x ^ a (mod m).
    print(f"""\n=== HORNER'S SCHEME TEST ===
14 ^ 8 mod 23 = {horner(14, 8, 23)}
1 ^ 25 mod 456 = {horner(1, 25, 456)}""")
    '''
    print(f"""3 ^ 2 mod 2 = {horner(3, 2, 2)}
2 ^ 0 mod 3 = {horner(2, 0, 3)}
0 ^ 145 mod 89 = {horner(0, 145, 89)}
12 ^ -5 mod 9 = {horner(12, -5, 9)}
-7 ^ 1 mod 5 = {horner(-7, 1, 5)}""")
    '''

    # Check prime number.
    print(f"""\n=== PRIME CHECK TEST ===
9 is prime == {miller_rabin(9)}
13 is prime == {miller_rabin(13)}
23 is prime == {miller_rabin(23)}
293 is prime == {miller_rabin(293)}""")
    '''
    print(f"""0 is prime == {miller_rabin(0)}
1 is prime == {miller_rabin(1)}
2 is prime == {miller_rabin(2)}""")
    '''

    # Find some prime number on interval.
    print(f"""\n=== PRIME GENERATION TEST ===
(1, 100) = {get_prime(1, 100)}
(250, 1000) = {get_prime(250, 1000)}
(500, 10000) = {get_prime(500, 10000)}
(5000, 10000) = {get_prime(5000, 10000)}
RSA prime = {get_RSA_prime(k=10)}""")

    # Generate key pair for RSA.
    pubkey, privkey = generate_key_pair()
    print(f"""\n=== RSA KEY PAIR GENERATION TEST ===
Public key:
\te = {pubkey[0]}
\tn = {pubkey[1]}
Private key:
\td = {privkey[0]}
\tp = {privkey[1]}
\tq = {privkey[2]}""")

    # Encrypt and decrypt a message.
    M = randint(1, 2 ** 16)
    C = encrypt(M, pubkey)
    print(f"""\n=== RSA ENCRYPTION/DECRYPTION TEST ===
Original:\t{M}
```

```
Encrypted:\t{C}
Decrypted:\t{decrypt(C, privkey)}""")

    # Sign and verify a message.
    signed = sign(M, privkey)
    print(f"""\n=== RSA SIGN/VERIFY TEST ===
Message:\t{signed[0]}
Signature:\t{signed[1]}
Correct (right key):\t{verify(signed, pubkey)}""")
#Correct (wrong key):\t{verify(signed, [29, 79])}

    # Key transferring.
    k = get_prime(1, 100)
    # Generate such pair of keys that: nA <= nB.
    pubkeyA, privkeyA = generate_key_pair()
    pubkeyB, privkeyB = generate_key_pair()
    while pubkeyA[1] > pubkeyB[1]:
        pubkeyA, privkeyA = generate_key_pair()
    # Send and receive key.
    signed = send_key(k, privkeyA, pubkeyA, pubkeyB)
    received = receive_key(signed, privkeyB, pubkeyA, pubkeyB)
    print(f"""\n=== RSA SEND/RECEIVE KEY TEST ===
Secret key:\t{k}
A keys:
\te = {pubkeyA[0]}
\tn = {pubkeyA[1]}
\td = {privkeyA[0]}
\tp = {privkeyA[1]}
\tq = {privkeyA[2]}
B keys:
\te1 = {pubkeyB[0]}
\tn1 = {pubkeyB[1]}
\td1 = {privkeyB[0]}
\tp1 = {privkeyB[1]}
\tq1 = {privkeyB[2]}
Encrypted key:\t{signed[0]}
Encrypted signature:\t{signed[1]}
Decrypted key:\t{received[0]}
Decrypted signature:\t{received[1]}""")
```

Лістинг 1. tests().

У її вихідних даних можна переглянути приклад p, q, p_1, q_1 разом з параметрами деякої системи RSA з користувачами A і B. Розглянемо приклад протоколу конфіденційного розсилання ключів з підтвердженням справжності.

```
=== RSA SEND/RECEIVE KEY TEST ===
Secret key:     59
A keys:
        e = 65537
        n = 971901754136457734677044559762196320918843838388399868658542074152695851436 9
        d = 308919929816997285061225189197038490631152186737389058328951062732425778312 1
        p = 22969257439931813438609183625741573163 7
        q = 42313155167428996486945535150495097437
B keys:
        e1 = 65537
        n1 = 339616907749235144616651349752613907971893660453472863747704154407735284814 03
        d1 = 231835165156862259484867297636944642181924603894749651428421552555482963413 13
        p1 = 23498332370759483475165181083455440168 1
        q1 = 144528089223830514348274806503221535563
Encrypted key:  75805883879172535296159267687769616261744273852632127524272685304635067781 79
Encrypted signature:    55404890072635135220126890884090032215046830597303534667935087882474577377 80
Decrypted key:  59
Decrypted signature:    60947466403999936460392466264528586509721795923550724427089914023922586826 94
```

Знімок 1. Приклад протоколу конфіденційного розсилання ключів з підтвердженням справжності.

Код не зберігає числа, що не пройшли перевірки на простоту, тому вони не були наведені. Те саме стосується чисел, які не відповідали вимогам криптосистеми RSA.

Перевіримо справність роботи системи при взаємодії із тестовим середовищем. Для цього була розроблена функція **RSA()**, яка дозволяє у працювати у циклі.

1. Як аргумент вона отримує список, в якому перший елемент - *n* (Modulus) сервера, а другий - *e* (Public exponent) сервера.
2. Генерує пару ключів для клієнта (поточного застосунку).
3. Входить в цикл обробки.

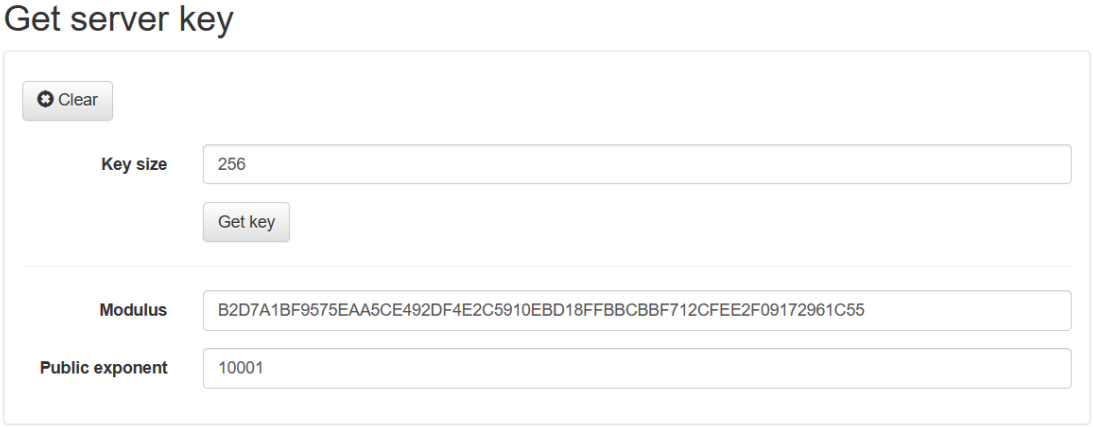
```
# Establish RSA communication between client and server.
# Server link: http://asymcryptwebservice.appspot.com/?section=rsa
# Receives input and shows output.
def RSA(pubkey_server: list) -> None:
    # Create client key pair.
    pubkey_client, privkey_client = generate_key_pair()
    e_client, n_client = pubkey_client
    # Store server's public key.
    e_server, n_server = pubkey_server
    if n_client > n_server:
        print("[INFO] Client's modulus is greater than server's: " + \
            "possible data loss during encryption and key sending.")
    # Processing loop.
    while True:
        cmd = input("\nEnter command (\'h\' for help): ")

        if cmd == 'h':
            print("""[INFO] All input data must be hexadecimal.
[INFO] Command list:
\th\t- show this message.
\tk\t- list client's and server's public keys.
\tg[+-]\t- generate new key pair for client (modulus greater or less than
server's respectively).
\tu\t- update server public key.
\tq\t- terminate program.
\te\t- encrypt specified message.
\td\t- decrypt specified ciphertext.
\tsi\t- sign specified message.
\tv\t- verify server's signed message.
\tse\t- send encrypted signed key to server.
\tr\t- receive encrypted signed key from server and verify it.""")

        # Show client's and server's public keys.
        elif cmd == 'k':
            print(f"""\n=== CLIENT KEYS ===
Modulus:\t{to_hex(n_client)}
Public exponent:\t{to_hex(e_client)}
d:\t{to_hex(privkey_client[0])}
p:\t{to_hex(privkey_client[1])}
q:\t{to_hex(privkey_client[2])}
phi(n):\t{to_hex(privkey_client[3])}\n
=== SERVER PUBLIC KEY ===
Modulus:\t{to_hex(n_server)}
Public exponent:\t{to_hex(e_server)}""")
```

```
        # Generate new key pair for client.
        elif cmd[0] == 'g':
            # Case incorrect parameter.
            if len(cmd) < 2 or cmd[1] not in "+-":
                print(f"[ERROR] Incorrect command \'g[+-]\' syntax: {cmd}")
                continue
            # Case client's modulus should be less than server's.
            elif cmd[1] == '-':
                while n_client > n_server:
                    pubkey_client, privkey_client = generate_key_pair()
                    e_client, n_client = pubkey_client
            # Case client's modulus should be greater than server's.
            else:
                while n_client < n_server:
                    pubkey_client, privkey_client = generate_key_pair()
                    e_client, n_client = pubkey_client
            print("[INFO] Generated new key pair for client (\'k\' to
show).")

        # Update server public key (modulus and public exponent).
        elif cmd == 'u':
            n_server = int(input("Enter server's modulus: "), 16)
            e_server = int(input("Enter server's public exponent: "), 16)
            print("[INFO] Updated server's public key (\'k\' to show).")
            if n_client > n_server:
                print("[INFO] Client's modulus is greater than server's: "
+ \
                      "possible data loss during encryption and key
transferring.")

        # Encrypt message on client to decrypt it on server.
        elif cmd == 'e':
            M = int(input("Enter message M: "), 16)
            C = encrypt(M, pubkey_server)
            print(f"Encrypted message C:\t{to_hex(C)}")

        # Decrypt encrypted message by server on client.
        elif cmd == 'd':
            C = int(input("Enter encrypted message C: "), 16)
            M = decrypt(C, privkey_client)
            print(f"Decrypted message M:\t{to_hex(M)}")

        # Sign a message for server.
        elif cmd == "si":
            M = int(input("Enter message M: "), 16)
            signed = sign(M, privkey_client)
            S = signed[1]
            print(f"Message:\t{to_hex(M)}\nSignature:\t{to_hex(S)}")

        # Verify a message signed by server.
        elif cmd == 'v':
            M = int(input("Enter message M: "), 16)
            S = int(input("Enter signature S: "), 16)
            signed = [M, S]
            result = verify(signed, pubkey_server)
            print(f"[INFO] {result}")

        # Send a key to server.
        elif cmd == "se":
```

```
            k = get_RSA_prime()
            while k >= n_client:
                k = get_RSA_prime()
            signed    =    send_key(k,    privkey_client,    pubkey_client,
pubkey_server)
            k1, S1 = signed
            print(f"""Key k:\t{to_hex(k)}
Encrypted key k1:\t{to_hex(k1)}
Encrypted signature S1:\t{to_hex(S1)}""")

        # Receive and verify a key from server.
        elif cmd == 'r':
            k1 = int(input("Enter encrypted key k1: "), 16)
            S1 = int(input("Enter encrypted signature S1: "), 16)
            signed = [k1, S1]
            received  =  receive_key(signed,  privkey_client,  pubkey_server,
pubkey_client)
            k, S = received
            result = verify(received, pubkey_server)
            print(f"""Decrypted key k:\t{to_hex(k)}
Decrypted signature S:\t{to_hex(S)}
[INFO] {result}""")

        # Quit program.
        elif cmd == 'q':
            print("[INFO] Finishing program...")
            break

        # Case incorrect command.
        else:
            print(f"[ERROR] Incorrect command passed: {command}")
```

Лістинг 2. RSA().

Переглянемо, як працює програма.



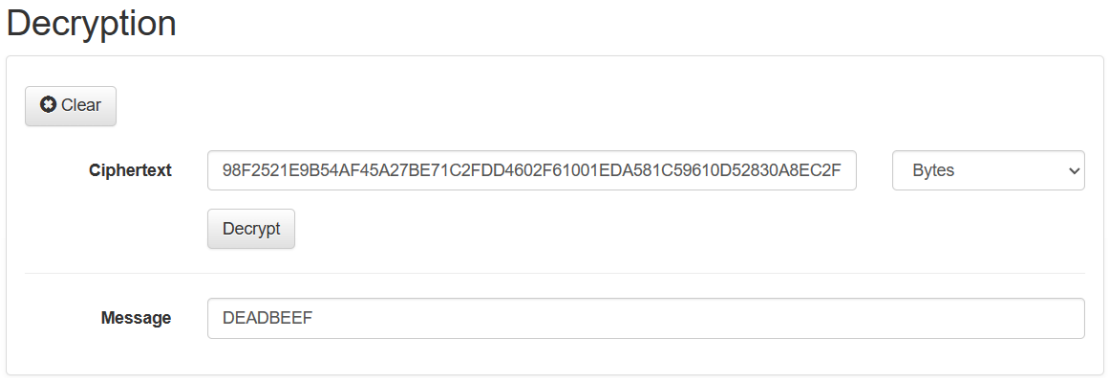
Знімок 2. Отримано відкритий ключ сервера.

```
kryva@kryva-virtual-machine:~/crypto-23-24/cp4/tyshchenko_fb-13_cp4$ ./cp4.py B2D
7A1BF9575EAA5CE492DF4E2C5910EBD18FFBBCBBF712CFEE2F09172961C55 10001
[INFO] Starting RSA...

Enter command ('h' for help): k

=== CLIENT KEYS ===
Modulus:        261C4331CF20A5AE83B76B89F90836B7C3F3C120C674FB9E47F8B4E417E977E5
Public exponent:        10001
d:      4C54CF41345B6F7DDAD3C5D0D3BA5FDCA47798BC8E09F7981AAD043804E4511
p:      6C705C99BCBEDEF6FE2008CE4F15B0E3
q:      59F85A10E2A18190694A52050A8CF697
phi(n): 261C4331CF20A5AE83B76B89F90836B6FD8B0A7627149B16E08E5A10BE46D06C

=== SERVER PUBLIC KEY ===
Modulus:        B2D7A1BF9575EAA5CE492DF4E2C5910EBD18FFBBCBBF712CFEE2F09172961C55
Public exponent:        10001

Enter command ('h' for help):
```

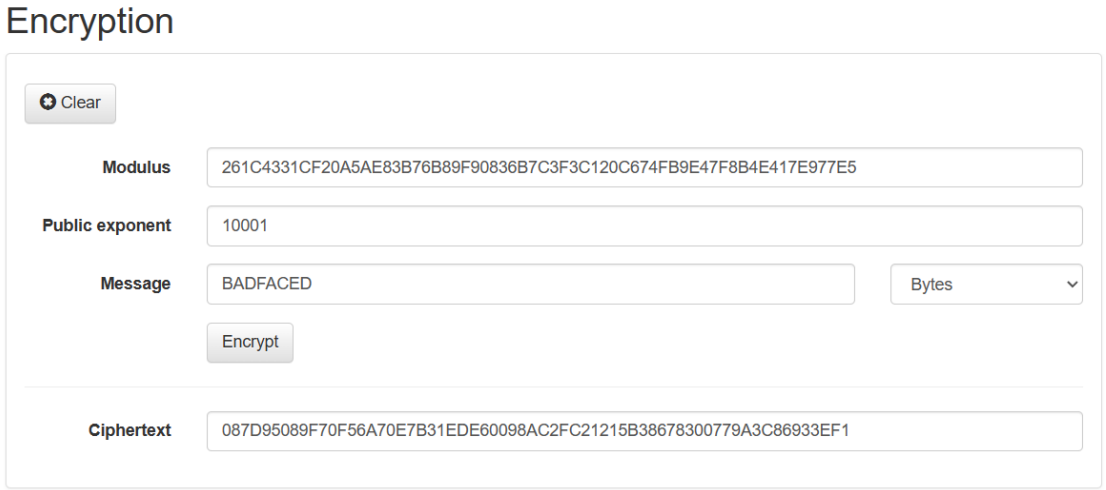
Знімок 3. Параметри побудованої криптосистеми RSA.

```
Enter command ('h' for help): e
Enter message M: DEADBEEF
Encrypted message C:    98F2521E9B54AF45A27BE71C2FDD4602F61001EDA581C59610D52830A8EC2F51

Enter command ('h' for help):
```

Знімок 4. Шифрування повідомлення клієнтом.



Знімок 5. Розшифрування повідомлення сервером.



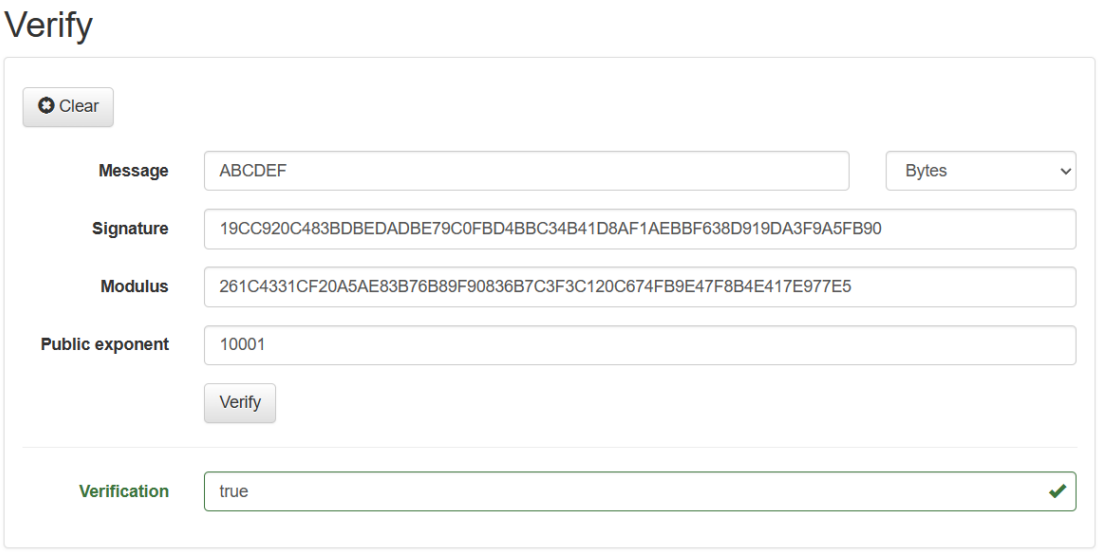
Знімок 6. Шифрування повідомлення сервером.

```
Enter command ('h' for help): d
Enter encrypted message C: 087D95089F70F56A70E7B31EDE60098AC2FC21215B38678300779A3C86933EF1
Decrypted message M:    BADFACED

Enter command ('h' for help):
```

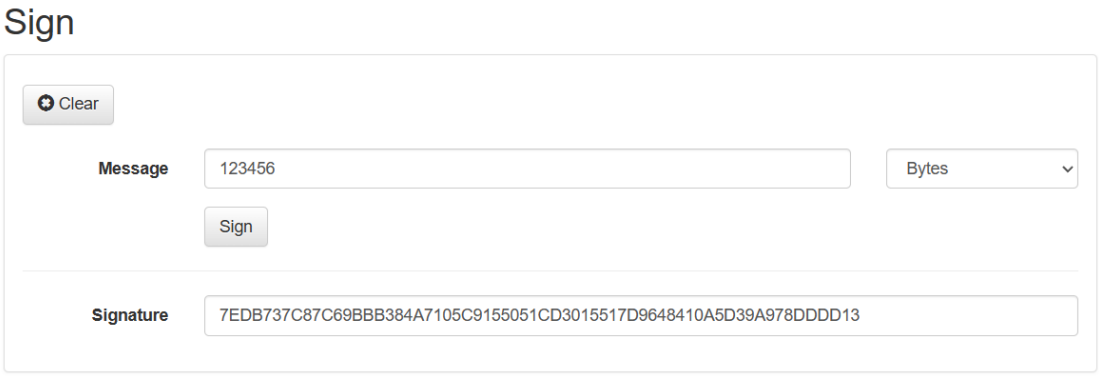
Знімок 7 Розшифрування повідомлення клієнтом.

```
Enter command ('h' for help): si
Enter message M: ABCDEF
Message:        ABCDEF
Signature:      19CC920C483BDBEDADBE79C0FBD4BBC34B41D8AF1AEBBF638D919DA3F9A5FB90

Enter command ('h' for help):
```

Знімок 8. Підпис повідомлення клієнтом.



Знімок 9. Перевірка підпису сервером.



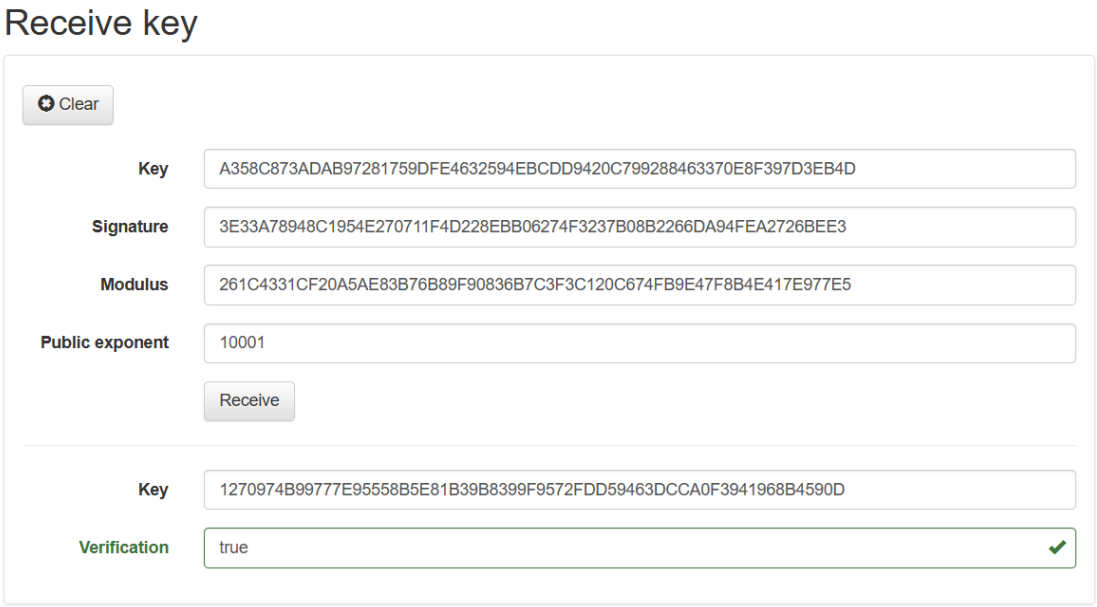
Знімок 10. Підпис повідомлення сервером.

```
Enter command ('h' for help): v
Enter message M: 123456
Enter signature S: 7EDB737C87C69BBB384A7105C9155051CD3015517D9648410A5D39A978DDDD13
[INFO] True

Enter command ('h' for help):
```

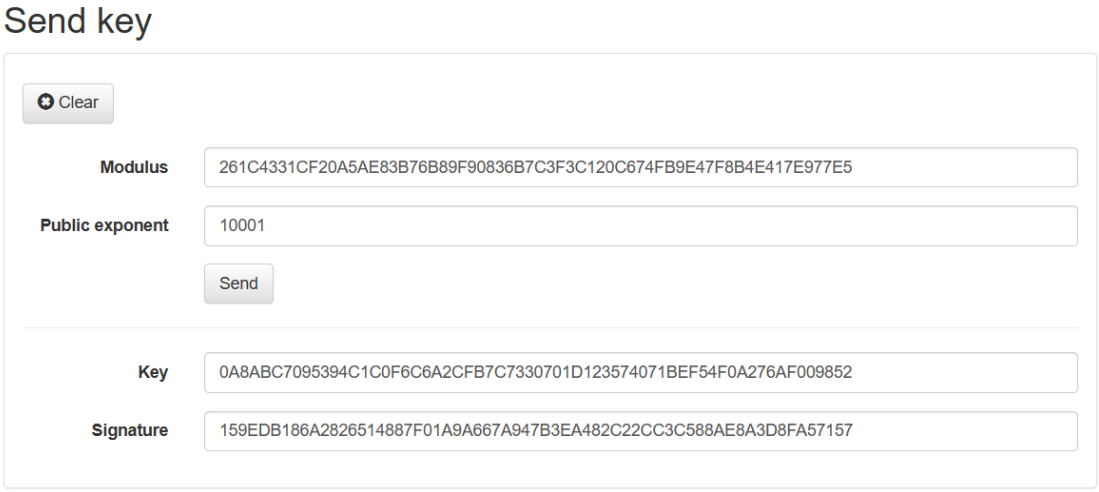
Знімок 11. Перевірка підпису клієнтом.

```
Enter command ('h' for help): se
Key k:  1270974B99777E95558B5E81B39B8399F9572FDD59463DCCA0F3941968B4590D
Encrypted key k1:       A358C873ADAB97281759DFE4632594EBCDD9420C799288463370E8F397D3EB4D
Encrypted signature S1: 3E33A78948C1954E270711F4D228EBB06274F3237B08B2266DA94FEA2726BEE3

Enter command ('h' for help):
```

Знімок 12. Надсилання ключа клієнтом.



Знімок 13. Отримання ключа сервером.



Знімок 14. Надсилання ключа сервером.

```
Enter command ('h' for help): r
Enter encrypted key k1: 0A8ABC7095394C1C0F6C6A2CFB7C7330701D123574071BEF54F0A276AF009852
Enter encrypted signature S1: 159EDB186A2826514887F01A9A667A947B3EA482C22CC3C588AE8A3D8FA57157
Decrypted key k:        5C66F030E943C93C
Decrypted signature S:  470BF1EABF9951E8C25BDF6B45D9CB15A4BB790109C838D6AF80D0F85FC4CFD
[INFO] True

Enter command ('h' for help):
```

Знімок 15. Отримання ключа клієнтом.

Таким чином, застосунок коректно взаємодіє з середовищем перевірки (сервером).

Висновок

У ході виконання комп'ютерного практикуму були отримані навички побудови криптосистеми RSA.

Крім того, була досягнута головна мета комп'ютерного практикуму: створено криптосистему RSA з можливістю шифрування/розшифрування повідомлень, підпису повідомлень і перевірки справжності повідомлення та реалізовано протокол конфіденційної передачі секретного ключа відкритим каналом.

Output тестування функцій програми міститься у файлі tests.txt.