

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ  
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ»  
ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

КРИПТОГРАФІЯ  
**КОМП'ЮТЕРНИЙ ПРАКТИКУМ №4**

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

Виконали:  
студенти групи ФБ-32  
Кошикова Дар'я  
Сажко Олена

## 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 і рекомендації щодо 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  $pq \neq p_1q_1$ ;  $p$  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  $(e, n)$ ,  $(n_1, e_1)$  та секретні  $d$  і  $d_1$ .

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  $M$  і знайти криптограму для абонентів А і В, перевірити правильність розшифрування. Скласти для А і В 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  $0 \leq k \leq n$ .

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

Згенерували  $p$ ,  $q$ ,  $p1$ ,  $q1$  :

```
p = 71876647656307248714409587053377957278616954632842566787505358976909620194139
q = 70798927610520525231292353932221728480580312529980744989516914667068324521493
p1 = 65047982127594580230212490056993439771450785755760407399238693462963124491449
q1 = 112728816273313520873810346569451580772568084268516937208776119348111697668021
```

Ці числа пройшли тести простоти (Міллер–Рабін + ділення малими простими).

Вони використовуються для побудови модулів RSA, а успішний прохід тестів означає, що вони ймовірно є простими.

На основі  $p$  і  $q$  для абонента А згенеровано відкритий та закритий ключ RSA:

```
abonent A:
public key (e, n): [65537,
50887895743057866702955682775033788153038476182639808120679509675177294755768
05183750956082690456273964183950885775001053248926294491378372385916938129527
]
private key (d, p, q):
[4615760505433686140893236740409056772139970118676508527600887212781961772367
58934340149606723701071080334812376344567886940031420125203927994437590160503
3,
71876647656307248714409587053377957278616954632842566787505358976909620194139
,
70798927610520525231292353932221728480580312529980744989516914667068324521493
]
```

Пара ключів дає можливість шифрувати дані (через  $e$ ,  $n$ ) та створювати цифрові підписи (через  $d$ )

Аналогічно, для абонента В були сформовані власні ключі RSA на основі  $p1$  і  $q1$ .

```
abonent B:
public key (e, n): [65537,
73327820262113909783984250947353415379560611609624825023297782865295154378939
63514111260650329555529464765252646455718446840743853302067393662764855252429
]
private key (d, p, q):
[5068846973975840296235467782577706908983069697652979916734759688643768220744
04039332719662026298943504851568429858976953367588358092934775882820564824771
3,
65047982127594580230212490056993439771450785755760407399238693462963124491449
]
```

```
,  
11272881627331352087381034656945158077256808426851693720877611934811169766802  
1]
```

Далі перед шифруванням і підписуванням абонент А виконує базові операції RSA над повідомленням:

```
abonent A  
encrypted:  
47196325247800566423076645168030530095455267800210659912323337797358999736354  
49168770973480764557045198723010787912736443560795871344750251338359253410238  
decrypted: 12345678901234567890  
signature:  
43938553941423879005974537567031186562366845441370444956991917376768304416246  
37817189249162012787550512562395184800154041385785831334635174773875019833464  
check: True
```

Шифрування показує, що публічний ключ працює правильно, а розшифрування — що приватний ключ коректно відновлює повідомлення. Підпис створений приватним ключем, і значення `check: True` означає, що публічний ключ підтвердив його справжність.

Після надсилання зашифрованого ключа і підпису абонент В виконав розшифрування й перевірку:

```
abonent B  
encrypted:  
18206405215274783332775879432580972915011659643441824159561380482483931875598  
95854312180703446659400730462407340645511582344032438413027037463440308584289  
decrypted: 12345678901234567890  
signature:  
50644879818526040929591299342065471982213365440121423813023502200215668733087  
69995735218738774127211905540551590093257882106318620290803584481558421283033  
check: True
```

Публічний ключ абонента В успішно шифрує повідомлення, а приватний правильно його відновлює. Підпис сформований приватним ключем абонента В, і `check: True` підтверджує, що він є дійсним.

Після надсилання зашифрованого ключа та підпису абонент В розшифровує дані й перевіряє підпис абонента А. Результат роботи програми:

```
exchange protool (A -> B)  
message k = 12345678901234567890  
signature confirmed
```

Абонент В успішно розшифрував значення `k`, а підтверджений підпис показує, що дані автентичні й не змінені.

Перевірка з сайтом:

[Server Key](#)

Encryption

[Decryption](#)[Signature](#)[Verification](#)[Send Key](#)[Receive Key](#)

## Encryption

✖ Clear

Modulus

54463729767519635201741346677784644913289234309871059€

Public exponent

65537

Message

12345678901234567890

Text ▼

Encrypt

Ciphertext

1EB9B069C9507418DE92209BDDAD39A0E92F6CA259F28BB54f

## RSA Testing Environment

[Server Key](#)

Encryption

Decryption

[Signature](#)[Verification](#)[Send Key](#)[Receive Key](#)

## Decryption

Key not initialized ✖

✖ Clear

Ciphertext

1EB9B069C9507418DE92209BDDAD39A0E92f

Text ▼

Decrypt

При перевірці на сайті шифрування спрацювало, але розшифрування постійно видавало помилку **“Key not initialized”**, і хоча ми довго намагалися розібратися, швидше за все сервер просто не приймає наш приватний ключ або формати даних.

**Висновки:**

Під час цієї лабораторної роботи ми ознайомилися з роботою асиметричної криптографії на прикладі RSA. Ми навчилися перевіряти числа на простоту, генерувати великі прості числа для ключів, шифрувати та розшифровувати повідомлення, а також створювати й перевіряти цифрові підписи. Особливо цікаво було побачити, як працює протокол безпечної розсилки ключів і як можна передавати секретну інформацію через відкритий канал.