Звіт про створення захищеного месенджера

Вступ

У цьому проєкті ми створили захищений месенджер — застосунок, що дозволяє людям обмінюватися повідомленнями так, щоб ці повідомлення не могли прочитати сторонні. Навіть якщо хтось перехопить їх у мережі, він не зможе їх розшифрувати чи підробити.

Ми хотіли не просто написати чат, а зробити його безпечним, доступним і водночас сучасним. Тому ми реалізували шифрування та цифровий підпис. У звіті ми пояснюємо, як саме це працює, як ми обирали алгоритми, чому вибрали саме ECC та DSA, і як побудували сам застосунок.

Перший етап: вивчення та реалізаціїї алгоритмів

Спершу ми реалізували чотири надані нам алгоритми шифрування:

- **RSA** — це метод шифрування, який ґрунтується на математичній задачі розкладу великого числа на прості множники. Його суть у тому, що повідомлення можна зашифрувати відкритим ключем, але розшифрувати — лише закритим. Цей алгоритм вважається надійним, однак потребує великих обчислювальних ресурсів і працює повільніше, ніж сучасні альтернативи.

- **ElGamal** — алгоритм, який шифрує дані з використанням випадкових чисел і спеціальної математичної операції, що ускладнює злам. Він генерує два зашифрованих фрагменти для кожного повідомлення, що трохи збільшує обсяг даних, але гарантує високу безпеку. Його часто використовують у відкритих криптосистемах.

- **Rabin** — це шифр, який працює за простим принципом: підносить число до квадрату, щоб зашифрувати його. Розшифрувати важче, бо результат може мати кілька можливих варіантів (зазвичай 4), і не завжди однозначно зрозуміло, який з них правильний. Через це він більше цікавий для теорії, ніж для реального застосування.

- **ECC (Elliptic Curve Cryptography)** — це сучасний спосіб захисту даних, який використовує геометричні властивості еліптичних кривих. Його головна перевага — висока безпека при мінімальних обчисленнях. Іншими словами, він дозволяє шифрувати дані надійно, швидко і з меншими витратами пам'яті, що особливо зручно для смартфонів і сучасних вебзастосунків.

Кожен з цих алгоритмів був реалізований нами **від нуля** — ми не використовували готові реалізаціїї з бібліотек, щоб повністю зрозуміти механізм і сильні та слабкі сторони кожного.

Другий етап: Детальний аналіз кожного алгоритму

RSA

Як працює:

1. **Генерація простих чисел p і q**
   На початку обираються два великі прості числа p і q. Це такі числа, що діляться тільки на 1 і на себе. Наприклад: 7, 13, 101 — це прості. У RSA ці числа мають бути дуже великими — зазвичай 1024 або 2048 біт, щоб їх не можна було підібрати.
2. **Обчислення $n = p \cdot q$**
   Ми перемножуємо p і q, отримуючи число n. Воно буде основою для всіх подальших обчислень: як під час шифрування, так і розшифрування.
3. **Обчислення функції Ейлера $\varphi(n)$**
   Це спеціальна математична функція, яка для RSA обчислюється як:

   $$\phi(n) = (p-1)(q-1)$$

   Це кількість чисел, менших за n, які не мають спільних дільників з n (тобто взаємно прості до нього). Ця функція потрібна для обрахунку приватного ключа.
4. **Вибір показника e**
   e — це число, яке:
   - більше за 1,
   - менше за $\varphi(n)$,
   - взаємно просте з $\varphi(n)$ (тобто $\gcd(e, \varphi(n)) = 1$).
5. **Обчислення d — закритого ключа**
   d — це число, яке є оберненим до e за модулем $\varphi(n)$. Тобто воно задовольняє рівняння:

   $$e * d \equiv 1 (mod \phi(n))$$

   Це називається обернене за модулем. Щоб знайти d, використовують розширений алгоритм Евкліда.
6. **Шифрування і розшифрування**
   Шифрування:

   $$C = M^e \, mod \, n$$

де M — повідомлення (перетворене в число), C — зашифрований результат.
Розшифрування:

$$M = C^d \, mod \, n$$

Тобто ми беремо повідомлення, підносимо його до степеня e, беремо залишок за модулем n — і передаємо. А одержувач, маючи d, робить те саме, але у зворотному напрямку — і отримує початковий текст.

Чому не вибрали:

Попри те, що RSA — перевірений часом алгоритм, у нашому випадку він має кілька суттєвих недоліків:

- Великий **розмір** ключів. Щоб досягти належного рівня безпеки, RSA потребує ключів розміром 2048 або навіть 3072 біти, що ускладнює зберігання та передачу.
- **Повільність**. Операції шифрування й розшифрування потребують багато часу і ресурсів, особливо на слабших пристроях.
- **Громіздкість зашифрованих даних**. RSA шифрує повідомлення блоками, які за розміром дорівнюють модулю n, через що вихідні дані суттєво збільшуються.
- Менша придатність для **сучасних** застосунків. У порівнянні з новішими алгоритмами, RSA є менш ефективним і важче інтегрується у ресурсообмежені системи (наприклад, мобільні або веб-додатки).

ElGamal

Як працює:

1. **Вибір параметрів**
   - Обирається велике просте число p (модуль)
   - Вибирається примітивний корінь g, такий, що його степені утворюють всю групу $\mathbb{Z}_p^*$ (тобто g є генератором групи)
2. **Генерація ключів**
   - Закритий ключ: випадкове число a у межах $2 < a < p - 2$
   - Відкритий ключ:

     $$e = g^a \, mod \, p$$

     Це — класична операція піднесення до степеня за модулем, або modular exponentiation.
3. **Шифрування повідомлення** M
   Для кожного повідомлення:
   - Вибирається випадкове число k (одноразовий ключ)

– Обчислюються:

$$c1 = g^k \, mod \, p$$
$$c2 = M * e^k \, mod \, p$$

Тут використано два піднесення до степеня та одне множення — все по модулю p.

* $c1$ — частина, яка маскує ключ
* $c2$ — частина, яка містить зашифроване повідомлення

4. **Розшифрування** Одержувач знає a (приватний ключ), тому може відновити s:

$$s = c1^a \, mod \, p$$

А далі знайти обернене до s за модулем p (тобто s^{-1}), і відновити повідомлення:

$$M = c2 * s^{-1} \, mod \, p$$

Тут знову:

– піднесення до степеня за модулем
– обчислення оберненого елемента
– множення по модулю

Чому не вибрали:

– **Подвоєний об'єм зашифрованих даних**. Оскільки результат — це пара (a, b), обсяг шифру приблизно вдвічі більший за розмір початкового повідомлення.
– **Складність у зберіганні та передачі**. Обробка пар чисел замість одного ускладнює реалізацію, особливо в мережевих застосунках.
– **Повільніше, ніж ECC**. Хоча ElGamal швидший за RSA, він все ж поступається криптографії на еліптичних кривих у швидкодії та компактності.

Rabin

Як працює:

1. **Генерація ключів**
   – Обираються два великі прості числа p і q, бажано такі, що $p \equiv 3 \mod 4$ і $q \equiv 3 \mod 4$ (це значно спрощує обчислення квадратних коренів у модульній арифметиці).
   – Обчислюється $n = p \cdot q$ — це модуль, який використовується для шифрування.
   – Публічний ключ: n.
   – Приватний ключ: пара (p, q).

2. **Шифрування**
   Дуже просте:

$$C = M^2 \, mod \, n$$

   Тобто береться повідомлення M (як число), підноситься до квадрату, і результат береться за модулем n.
   Це дуже швидка і легка операція — модульне піднесення до квадрату.

3. **Розшифрування (складна частина)**
   Щоб відновити M з C, треба знайти всі такі x, що $x^2 \equiv C \mod n$.
   Це не так просто, бо:

   – Якщо $n = p \cdot q$, то в модулі n є 4 різних квадратичних корені, тобто існують 4 різних числа, які при піднесенні до квадрату дадуть одне й те саме C.

   Що робимо:

   (а) Знаходимо два корені по модулю p:

$$r_p = C^{\frac{p+1}{4}} \, mod \, p$$

   (б) Знаходимо два корені по модулю q:

$$r_q = C^{\frac{q+1}{4}} \, mod \, q$$

   (в) Тепер маємо по 2 корені в кожному модулі → всього 4 комбінації.

   (г) Об'єднуємо їх у розв'язки по модулю n за допомогою Китайської теореми про залишки (CRT):

   – CRT дозволяє зібрати окремі розв'язки по p і q в один спільний розв'язок по $n = p \cdot q$.
   – В результаті отримуємо 4 можливі значення M.

Чому не вибрали:

– Його безпека прямо зводиться до задачі факторизації (а це один з **найважчих** класичних криптозавдань).

– Розшифрування **не однозначне**. Алгоритм повертає 4 варіанти, з яких треба вручну визначати правильний. Це потребує додаткових меток, перевірок або логіки — що ускладнює реалізацію.

– **Складний у використанні**. Якщо неправильно обрати повідомлення або модуль, алгоритм може працювати некоректно.

– **Практично не використовується**. Більшість сучасних систем або не підтримують Rabin взагалі, або замінюють його на ефективніші схеми.

ECC

Як працює:
ECC — це сучасний криптографічний алгоритм, який використовує особливі математичні об'єкти — еліптичні криві над скінченими полями — для створення ключів

і шифрування. На відміну від RSA, який ґрунтується на розкладі великих чисел, *ECC* використовує геометричні операції над точками на кривій, які створюють односторонню функцію: легко виконати, але дуже важко звернути назад.

Що таке еліптична крива?
Це рівняння вигляду:

$$y^2 = x^3 + ax + b$$

. . . але всі обчислення виконуються **по модулю простого числа** p — тобто з залишками (як у RSA, але геометрично).

Кожна пара чисел (x, y), яка задовольняє це рівняння в обраному полі, — це точка на кривій. І над цими точками можна робити спеціальні операції: **додавати точки, множити на число тощо.**

Генерація ключів:

1. Вибирається точка G на кривій — так звана **базова точка**.
2. Користувач обирає випадкове число p — **закритий ключ**.
3. Обчислюється $P = p \cdot G$ — **відкритий ключ** (множенням точки на число).

Множення точки G на p — це багаторазове додавання $G + G + ... + G$ (p разів).

Безпека полягає в тому, що знаючи G і P, практично неможливо відновити p. Це називається **проблема дискретного логарифма на еліптичній кривій**, і вона на кілька порядків складніша, ніж у класичних системах.

Шифрування:
ECC сам по собі не шифрує напряму текст. Зазвичай його використовують для:

- генерації **спільного секрету** (як у протоколі Ель-Гамаля на кривій),
- або для створення **ключа для симетричного шифрування** (наприклад, AES).

У нашій реалізації *ECC* ми створюємо спільну точку, яка обчислюється обома сторонами, і з її координат отримуємо симетричний ключ. Потім текст шифрується, наприклад, через AES, з використанням цього ключа.

Чому обрали: *ECC* став очевидним вибором після практичного тестування усіх чотирьох алгоритмів. Ось основні аргументи:

1. **Компактність**
   - Для забезпечення тієї ж криптостійкості, що RSA-2048, ECC достатньо 256-бітного ключа.

- Це економить пам'ять, пропускну здатність і робить алгоритм ідеальним для мобільних, веб- і хмарних застосунків.

2. **Висока швидкодія**
   - Генерація ключів, шифрування і дешифрування — відбуваються значно швидше, ніж у RSA чи ElGamal.
   - Алгоритм чудово працює навіть на слабших пристроях.
3. **Стандарти безпеки**
   - ECC рекомендований багатьма сучасними криптографічними стандартами (NIST, ISO, NSA Suite B).
   - Він уже активно використовується у протоколах TLS, Signal, Apple iMessage, WhatsApp та ін.
4. **Стійкість до атак**
   - Завдяки складності задачі дискретного логарифма на кривій, ECC є значно важчим для зламу, ніж RSA або ElGamal — за однакової довжини ключа.

Підпис: DSA

Коли користувач надсилає повідомлення, він додає до нього підпис, створений за допомогою свого приватного ключа. Інший користувач, маючи публічний ключ відправника, може переконатися, що цей підпис справжній.

Використовувані параметри:

- p — велике просте число (модуль)
- q — просте число, яке ділить $p - 1$
- g — елемент порядку q за модулем p (тобто $g^q \mod p = 1$)
- x — приватний ключ ($0 < x < q$)
- $y = g^x \mod p$ — публічний ключ

Ці параметри можуть бути спільними для всіх користувачів системи, крім x та y — вони індивідуальні.

Підпис повідомлення
Щоб підписати повідомлення m, робимо так:

1. Обираємо **випадкове** число k таке, що $0 < k < q$
   (важливо: кожен підпис — з новим k!)
2. **Обчислюємо**:

$$r = (g^k \bmod p) \bmod q$$

3. **Обчислюємо**:

$$s = (k^{-1} * H(m) + x * r) \bmod p$$

де $H(m)$ — хеш повідомлення (наприклад, SHA-256), а k^{-1} — обернене до k за модулем q.

Пара (r, s) — це **цифровий підпис**.

Перевірка підпису Одержувач повідомлення m з підписом (r, s) перевіряє його так:

1. **Обчислює**:

$$w = s^{-1} \bmod q$$

2. **Далі**:

$$t_1 = H(m) * w \bmod q, t_2 = r * w \bmod q$$

3. **Обчислює**:

$$v = (q^{t_1} * y^{t_2} \bmod p) \bmod q$$

Якщо $v = r$, підпис дійсний.
Якщо ні — повідомлення або підроблене, або надіслане кимось іншим. **У нашому месенджері:**

- перед відправленням повідомлення — **підписується**,
- при отриманні — **перевіряється** за публічним ключем.

Реалізаціїя застосунку

Ми створили месенджер з такими складовими:

- **Сервер** на FastAPI, який обробляє WebSocket-з'єднання та веде список активних користувачів.
- **Клієнт** з UI (PyQt6).
- **Зашифроване передавання даних** через ECC.
- **Підпис повідомлень** через DSA.
- **Обмін публічними ключами** через сервер.
- **Вивід перевірених повідомлень** у віконці або терміналі.

Висновок

Ми протестували **4 алгоритми** шифрування, проаналізували їх сильні і слабкі сторони — **і обєктивно обрали ECC**, як найефективніший, сучасний і безпечний варіант. Алгоритм DSA став логічним доповненням — завдяки ньому наш месенджер не просто захищений, а **достовірний**.

Цей проєкт для нас — не просто навчальна вправа. Ми хочемо, щоб майбутні технології, ***навіть у найпростіших речах, як чат****, поважали безпеку і довіру між людьми.*