Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем факультету інформатики

Методи захисту інформації в клієнт-серверних архітектурах

Текстова частина до курсової роботи
за спеціальністю «Інженерія програмного забезпечення» 121

Керівник курсової роботи доц., к. н. Олецький О.В.
_____
(підпис)
“____” _____ 2021 р.

Виконав студент 3-го курсу
Кузан О.О.
“____” _____ 2021 р.

Київ 2021
Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ

Зав. кафедри мультимедійних систем,

доцент, к.ф.-м.н.

_____ О.П. Жежерун

(підпис)

“____”_____ 2020 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студенту Кузану Олегу

3-го курсу факультету інформатики

ТЕМА: Методи захисту інформації в клієнт-серверних архітектурах

Вихідні дані:

- Клієнт-серверне застосування для роботи з медичними висновками у клініці

Зміст ТЧ до курсової роботи:

Вступ

1. Аналіз предметної області. Постановка завдання курсової роботи 2. Теоретичні відомості

3. Опис реалізації програмного продукту

Висновки

Список джерел

Додатки (за необхідністю)

Дата видачі “___” _____ 2020 р.

Керівник _____ Завдання отримано _____

Календарний план виконання курсової роботи

Тема: Розробка клієнт-серверної системи діагностування пацієнтів

Календарний план виконання роботи:

№ п/п	Назва етапу курсової роботи	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу	Жовтень-листопад 2020р.	
2.	Вивчення та аналіз задачі	Листопад-грудень 2020р.	
3.	Розробка архітектури та загальної структури програми	Січень-лютий 2021р.	
4.	Створення застосунку	Лютий-березень 2021р.	
5.	Написання текстової частини	Березень-квітень 2021р.	
6.	Перегляд курсової роботи науковим керівником	Травень 2021р.	
7.	Створення презентації	Травень 2021р.	
8.	Захист курсової роботи		

Студент Кузан О. О. _____

Керівник Олецький О. В. _____

“_____” _____ 2020 р.

Зміст

Анотація

Розглянуто основні аспекти криптографії: конфіденційність, цілісність даних, автентифікація, та інструменти їх досягнення (симетричне, асиметричне шифрування, протоколи узгодження ключа, хешування, генератори випадкових чисел, наскрізне шифрування) в контексті клієнт-серверної архітектури. Показані актуальні атаки на безпеку в клієнт-серверних застовуваннях, проаналізована їхня ефективність та способи захисту. Надано рекомендації з оптимальних параметрів безпеки. Спираючись на розглянуті технології та концепції, розроблено застосунок для захищеного спілкування.

Ключові слова: криптографія, цілісність, конфіденційність, автентифікація, клієнт-сервери, шифрування, хешування, AES, DH, E2EE, RSA, PFS, MD, SHA, CHF, MITM.

Вступ

З дедалі швидшою цифровізацією, проникнення технологій в усі сфери нашого життя стало буденністю. Ми покладаємось на них більше, ніж на найближчих друзів, довіряючи тексти приватних повідомлень, історію пошуку та покупок, банківські дані та документи. Це обумовлює важливість та актуальність питання захисту цієї чутливої інформації. Від безпеки даних може залежати не тільки приватність користувачів, а й репутації корпорацій та навіть державні інтереси.

Метою цієї курсової роботи є систематизація уявлень про безпековий аспект архітектури клієнт-серверних застосувань, визначення її ключових складових, з’ясування оптимальних параметрів налаштування засобів захисту з врахунням рівня сьогоднішніх загроз.

Робота складається з двох розділів. У першому розібрано теоретичні основи безпеки в клієнт-серверних застосуваннях, зокрема з яких компонентів вона складається, якими засобами їх можна досягти, які недоліки і переваги є у цих засобів, та як рекомендовано ними послуговуватись задля досягнення оптимальних результатів. У другому описано засоби розробки програмного застосунку, обґрунтування безпекової архітектури та результати тестування.

Розроблено застосунок для захищеного спілкування, у якому втілені ідеї, розглянуті у роботі, покладаючись на ті алгоритми і протоколи, безпека яких аналізувалась. Для уникнення помилок реалізації, у створенні застосунку використовувались лише публічно обговорені та протестовані імплементації безпекових компонентів.

Джерелами слугували роботи іноземних спеціалістів у галузі кібербезпеки, оригінальні роботи винахідників відомих алгоритмів, безпекові рекомендації державних установ США, аналітичні статті.

1 Аналіз предметної області

1.1 Основи криптографії

Окреслимо та визначимо основні поняття, на яких базуватиметься подальший матеріал. Криптографія – наука про забезпечння секретності інформації [26]. Ціль криптографії – захистити приватність комунікації в публічному світі. Виокремимо три основи безпечної передачі даних:

- Конфіденційність: повідомлення може прочитати лише адресат;
- Цілісність: зміст повідомлення залишився без змін у процесі передачі;
- Автентичність: автор повідомлення є тим, ким представляється.

Сучасна криптографія оперує трьома типами криптографічних функцій:

- Симетричне шифрування;
- Асиметричне шифрування;
- Хешування.

З криптографією нерозривно пов'язана інша наука – криптоаналіз. Вона вивчає математичні методи отримання вихідних даних зашифрованої інформації без знання ключа. Деякі види атак:

- CPA (англ. Chosen plaintext attack) – атака на основі підібраного відкритого тексту.
- Брут-форс – метод вирішення задачі шляхом перебору усіх можливих варіантів розв’язку (метод грубої сили).
- MITM (англ. Man-in-the-middle) – ситуація, коли зловмисник здатен втручатись у комунікацію, перехоплюючи та змінюючи повідомлення, без можливості бути поміченим.
- Атака стороннім каналом – атака спрямова на вразливості імплементації;
- Атака за часом (англ. timing attack) – вид атаки стороннім каналом, в якій час, потрібний на виконання криптографічних алгоритмів дає якусь додаткову, корисну для зловмисника інформацію.

Результатом шифурвання повідомлення, або відкритого тексту називається шифротекст. Систему шифрів можуть називати криптосистемою.

1.2 Криптографічні алгоритми

1.2.1 Симетрична криптографія

У симетричній криптографії один ключ (англ. secret key) використовується і для шифрування, і для дешифрування (рисунок 1.1). Це простий і давній метод шифрування.

Рисунок 1.1 – Симетричне шифрування

Найбільш широковикористовуваний алгоритм симетричного шифрування сьогодні – Advanced Encryption Standard (AES), який прийшов на зміну застарілому DES, прийнятий Національним інститутом стандартів і технології США (NIST) як переможець відкритого конкурсу, проведеного у 2001 р. Основними критеріями були вартість, складність та очікувана криптоскійкість 1[17]. Це один з найпопулярніших алгоритмів шифрування у світі.

Можливі довжини ключів у бітах – 128, 192, 256, чим довший ключ, тим більше часу займе брутфорс атака (повний перебір) [1]. Уряд США використовує довжину 128 для файлів з поміткою SECRET, та 192, 256 для TOP SECRET файлів [18].

AES має ключовий розклад – алгоритм розширення базового ключа і добування з нього ключів для раундів. Важливість використання ключового розкладу, а не однакових ключів для раудів в тому, що це допомагає захиститись від атак на пов’язаних ключах та атак зсувом [19, 20].

Порядок операцій всередині раунду, хоч і змінить шифротекст, ніяк не впливає на безпеку алгоритму [21].

Процес шифрування складається з раундів, одного початкового, кількох основних та одного кінцевого, на яких застосовуються операції, які забезпечують дифузію (рисунок 1.2). У процесі дешифрування використовується ця ж послідовність кроків, але оскільки не всі операції є зворотними самі до себе, так як у DES, приходиться провести деяку додаткову підготовку. Тому теоретично для шифрування одного блоку даних алгоритм дешифрування не буде слабшим варіантом [15, 16].

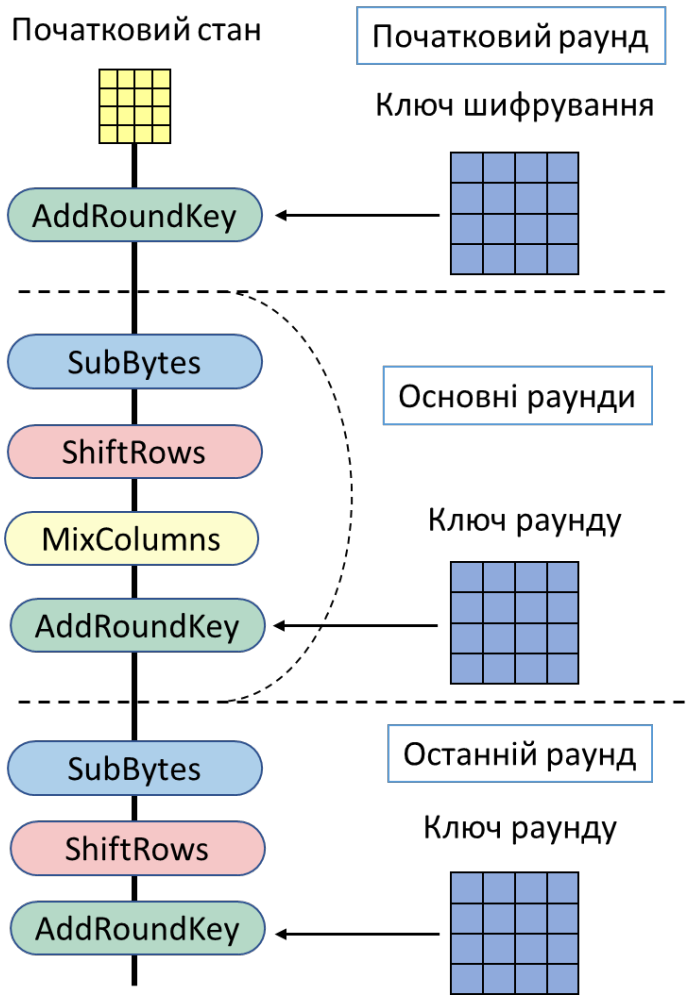

Рисунок 1.2 – Раунди AES

Кількість раундів відображає баланс між продуктивністю і безпекою, який часто доводиться шукати в криптографії. Як казали автори, у випадку з ключем довжиною 128 біт вони не передбачають успішних атак на алгоритм, з кількістю раундів більше 6, а 4 додали для більшої безпеки у майбутньому (як відомо, «з часом атаки стають лише кращими»). Далі на кожні 32 біти ключа вирішили додавати один раунд. Таким чином маємо 10, 12, 14 раундів для довжин ключа у 128, 192 та 256 відповідно [1].

Більше раундів в загальному випадку означають більше роботи для криптоаналітиків, однак є атаки, для яких цей параметр не має значення, наприклад зсувна атака (англ. slide attack) [14].

Симетричні шифри можуть бути блочними (за одну ітерацію перетворюється блок фіксованого розміру) або потоковими (за одну ітерацію перетворюється один біт або байт). Блочні шифри зазвичай простіші в реалізації, можуть забезпечувати цілісність та автентифікацію у деяких імплементаціях, але є повільнішими. Потоковий шифр можна використовувати як генератор псевдовипадкових чисел, якщо ним шифрувати послідовність нулів.

У більшості режимів дії блочні шифри мають отримати на вхід текст, довжина якого кратна довжині блоку шифру. Тому виникає потреба у доповненні тексту безсенсовим навантаженням до потрібної довжини. Є різні конвенції щодо того яким може бути це доповнення. Наприклад бітове доповнення полягає в додаванні до кінця повідомлення 1 та необхідної кількості 0 до кратного числа.

Бувають і байтові доповнення, популярним є стандарт PKCS7 (і його частковий випадок PKCS5 для восьмибайтових блоків) [2]. Згідно нього, якщо до кратності бракує N байтів, то до повідомлення додаються N байтів, кожен із значенням N. У процесі розшифрування перевіряється, чи мають останні N байтів значення N кожен, якщо так, то доповнення відкидається, а повідомлення приймається, якщо ні, викидається повідомлення про помилку розшифрування (рисунок 1.3) [3].

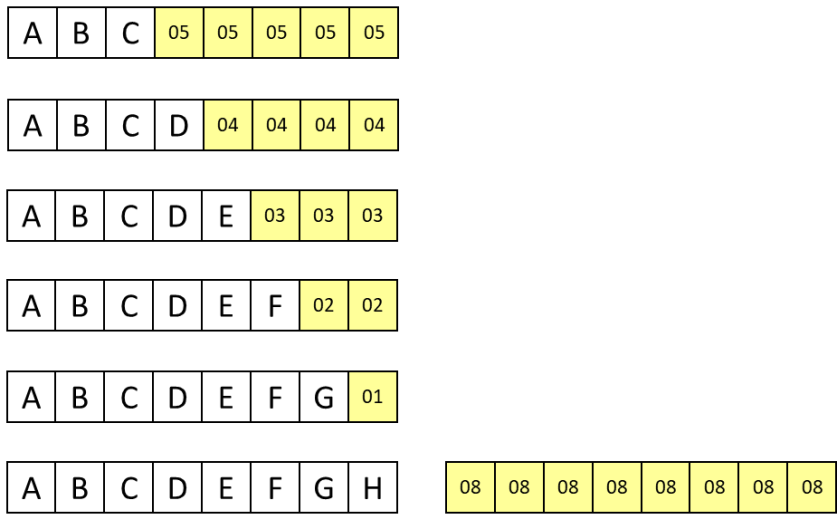

Рисунок 1.3 – Доповнення PKCS7

Важливо пам'ятати, що якщо доповнення використовується в обраному режимі, то доповнюватись будуть і повідомлення, які кратні довжині блоку – адже інакше не можливо відрізнити доповнення від вмісту повідомлення.

Доповнення можуть використовувати і в інших випадках, для ускладнення криптоаналізу, наприклад, щоб приховати справжню довжину повідомлення.

Блочні шифри використовуються не напряму, а через режими блочного шифрування, або режими дії (англ. modes of operation), найпростіший з яких, режим електронної книги кодів (ECB), просто ділить повідомлення на блоки і застовує алгоритм послідовно до кожного з них. Принцип його роботи продемонстровано на рисунку 1.4:

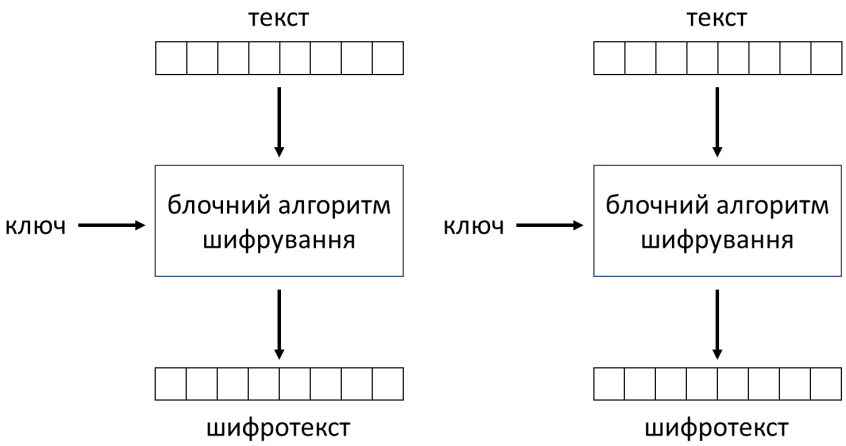

Рисунок 1.4 – Режим дії ECB

Оскільки AES детерміністичний алгоритм, однакові вхідні дані результують в однаковий шифротекст, що полегшує криптоаналіз.

Сематично стійкими називають криптосистеми, котрих з шифротексту можна витягнути лише незначну інформацію про відкритий текст. AES-ECB не є семантично стійким, адже можна визначити, чи мають повідомлення повторювані дані, та де саме. Для наочної ілюстрації цього часто використовують рисунок 1.5:

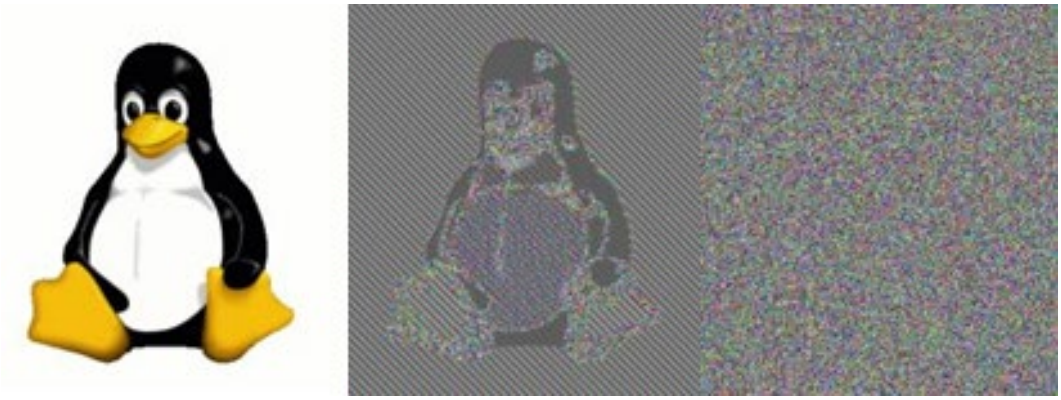

Рисунок 1.5 – ECB-пінгвін

На першому зображенні – оригінал, на другому – результат режиму ECB, а на третьому – інших режимів. Як бачимо, сліпе застовування алгоритму шифрування зовсім не означає, що дані справді буде захищено.

Як наслідок, алгоритм криптографічно «податливий» (malleable), тобто легко комбінувати відомі блоки шифротексту, які будуть валідними та перетворяться у звичайний текст. З цих причин, ECB може використовуватись хіба що для шифрування одного блоку даних, або для шифрування справді випадкових байтів, тим не менше, прикрі випадки його невдалого використання трапляються навіть у великих корпораціях і у наш час [4]. CPA дозволяє зламати N байтів AES-EBC за $N \cdot (2^8)$ операцій (хоч це не дає даних про сам ключ) [5].

Інший популярний режим – CBC (Cipher Block Chaining). При шифрування блоку використовується попередній блок, а для першого блоку його заміняє

ініціалізаційний вектор (IV – Initialization vector). Таким чином кожен блок залежить від всіх попередніх. Принцип дії можна побачити на рисунку 1.6:

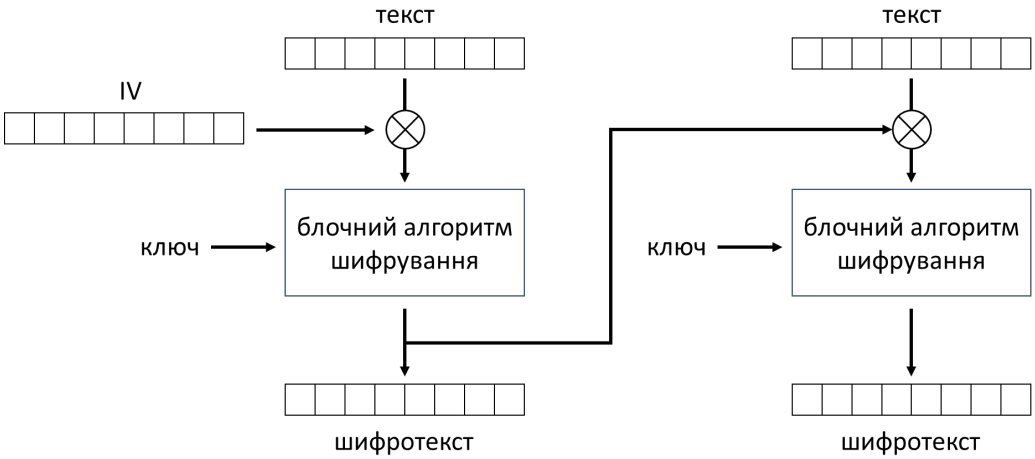

Рисунок 1.6 – Режим дії CBC

IV має бути випадковим та різним для кожного повідомлення зашифрованого одним ключем, але він не обов’язково має бути секретним. Іншими словами, не можна допустити шифрування за допомогою певної пари ключ-IV двічі, принаймі один елемент має відрізнятись.

CBC може бути вразливим до атак padding oracle (оракул доповнення). Оракул в контексті криптографії – абстрактна модель, математичний опис для витоку даних, для подальшого розгляду можливості якихось атак. Оракул дає відповідь миттєво (O(1)) зазвичай на складні задачі, які ми би не могли вирішити самі.

Щоб атака була можливою, хакеру треба мати доступ до такого оракула, який, в нашому випадку, повідомлятиме, чи є доповнення надісланого шифротексту валідним (рисунок 1.7). Цього достатньо, щоб повністю розшифрувати повідомлення, не знаючи ключа шифрування [33].

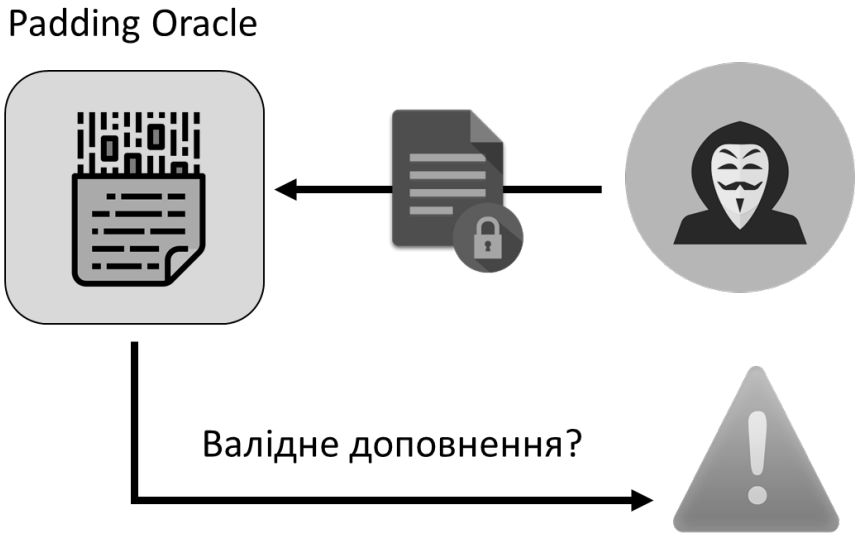

Рисунок 1.7 – Оракул доповнення

Атака відбувається так: нехай нас цікавить блок m1. Позначимо останній байт блоку m[1] за lb, і припустимо, що він дорівнює g . Замінюємо останній байт шифротексту c[0], який відповідає блоку m[0], на g ⊕ 0x01. Згодовуємо цей шифротекст оракулу – якщо він його прийме, значить lb ⊕ g ⊕ 0x01 дало правильне доповнення, 0x01, значить g – правильна здогадка. Якщо оракул поверне помилку, продовжуємо здогадки, всього є 28 варіантів (рисунок 1.8). Процес аналогічний для злому інших байтів, хіба доповнення треба змінити на відповідне – для двох байтів це буде два байти виду 0x02 і т.д [].

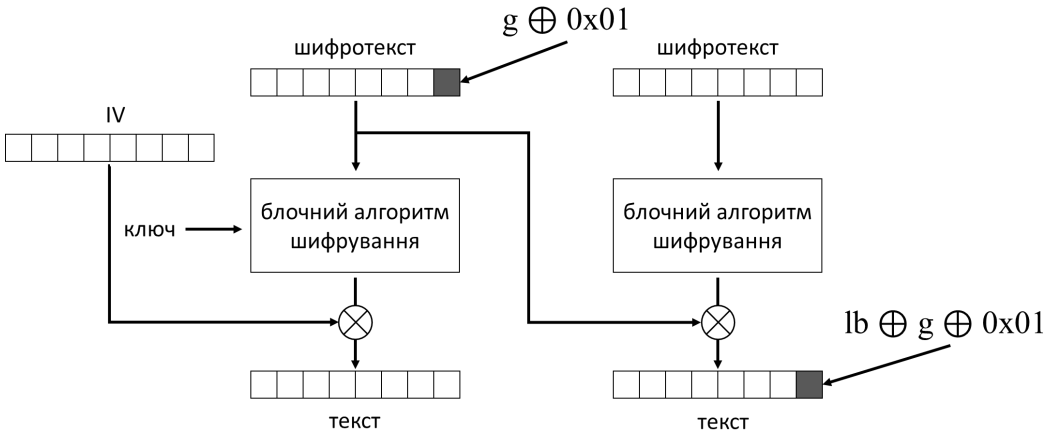

Рисунок 1.8 – Атака оракула доповнення

Спершу цю атаку відбили тим, що завжди повертали стандартне повідомлення про помилку, без вказування, що саме пішло не так. Але це все ще не унеможлювало атаки по часу. Було з'ясовано, що якщо доповнення не валідне, то сповіщення про помилку приходить швидше, адже воно перевіряється першим, тобто оракул все ще «працював». Щоб вирішити цю проблему, процес перевірки не зупиняють щойно знадейно помилку в доповненні.

Це приклад атаки на основі підібраного шифротексту (CCA - chosen-ciphertext attack), а також атаки сторонніми каналами (Side-channel attack). Реальними атаками типу paddling oracle є SSL-атаки Lucky 13, POODLE (SSL 3.0) та BEAST (TLS 1.0) [].

NIST наразі одобрив 14 режимів шифрування, 8 з яких забезпечують лише конфіденційність (ECB, CBC, OFB, CFB, CTR, XTS-AES, FF1, FF3), один лише автентифікацію (CMAC), та 5 гарантують і конфіденційність, і автентифікацію (CCM, GCM, KW, KWP, TKW). У режимах OFB, CTR та CFB блочний шифр веде себе як потоковий, отже не потребує алгоритмів доповнення і не є вразливим до padding oracle атак [6].

Саме режими з забезпеченням цілісності і автентифікації бажано використовувати. Режими типу CBC варто окремо супроводжувати перевірками цілісності, наприклад за допомогою HMAC.

Через таку популярність алгоритму, питання його безпеки завжди актуальне і є темою багатьох досліджень. Технічно, будь-який спосіб знаходження правильного ключа з кількістю необхідних операцій меншою, ніж повний перебір, може вважитись «зламом» алгоритму. У цьому сенсі, AES теж був зламаний. Інша справа, що ніякої практичної загрози це не несе, затребувана кількість обчислень все ще недосяжна для сучасних комп'ютерів (рівні криптостійкості від 80-90 біт можна вважати достатнім – усієї потужності майнерів біткоїнів впродовж року станом на 2021 р. вистачило би для зламу приблизно 93 бітів) [7].

Брутфорс AES навіть з найменшою довжиною ключа, 128, справа безнадійна, адже потрібно 2^{128} операцій, що зайняло би мільярди років обчислень на суперкомп'ютерах.

Атака на пов'язаних ключах (передбачає можливість хакера отримати шифротекст на невідомих ключах, які пов'язані відомими математичними залежностями) може зламати AES-192 та AES-256 із складностями 2^{176} ($< 2^{192}$) та $2^{99.5}$ ($< 2^{256}$) відповідно. Атака є суто теоритичною, тому не варто думати, що AES-192/256 незважаючи на довший ключ менш захищені. Також варто зазначити, що захист від такого роду атак не був критерієм на відборі кращого алгоритму для AES.

Більш небезпечений і реальний сценарій – атака з використанням одного ключа, дозволяє зменшити складність злому AES-128 з 2^{128} лише до $2^{126.1}$.

Більш успішними є атаки на AES із меншою кількість раундів (Square, Impossible Differences, Collision, Boomerang атаки). Наприклад якби AES-128 мав не 10 раундів, а 6, то складність зламу можна зменшити до 2^{32} [].

Атака по повному дводольному графу (biclique attack) приносить результати $2^{126.1}$, $2^{198.7}$ та $2^{254.4}$ для AES128, AES192 та AES256 відповідно [8]. Утім навіть якщо в майбутньому буде знайдено вразливість для повнораундових версії AES, імовірно, що достатньо буде просто збільшити кількістів раундів.

Головною проблемою, яку треба розв’язати, якщо маємо справу з симетричним шифрування, це безпечена передача спільного ключа. Тут у нагоді стають алгоритми асиметричного шифрування.

1.2.2 Асиметрична криптографія

Асиметрична криптографія характеризується використанням пари ключів – відкритого (public), та закритого (private). Відповідно до назв, закритий ключ має бути відомим лише його власнику, а відкритий може бути відомий усім. Щоб відправити повідомлення, його шифрують закритим ключем, а приймаюча сторона розшифрує його відповідним відкритим ключем (рисунок 1.9).

Рисунок 1.9 – Асиметричне шифрування

Технологію було винайдено 1976 році, вона стала справжньою революцією в криптографії, адже вирішила проблему безпечної передачі ключа, який до цього доводилось передавати, наприклад, через дипломатів. Математична основа цих алгоритмів - це функції, які легко (тобто зі складністю не більшою ніж $O(n^k)$ для вхідних даних довжиною n) обчислити лише в одну сторону (one-way functions).

Якщо закритий ключ скомпрометовано, потрібно згенерувати нову пару ключів. З відкритого ключа дістати приватний практично не можливо, на чому і базується безпека алгоритму, отримання ж відкритого ключа із закритого може бути тривіальним або теж складним, залежно від конкретної імплементації.

Найпопулярнішим асиметричним шифром є RSA (Rivest–Shamir–Adleman), винайдений у 1978. Він базується на популярній проблемі факторизації, тобто знаходження простих множників цілих чисел (адже множити набагато легше, ніж знаходити дільники). Як і AEC-CBC потребує використання доповнення і теж може бути вразливим до padding oracle атак, зокерма атаки Бляйхенбахера [9].

Важливою перевагою асиметричного шифрування є можливість гарантувати виконання криптографічної «тріади» – не тільки конфіденційності, але й цілісності та автентифікації [10].

За такої архітектури нема потреби передавати якусь критично важливу інформацію через незахищені канали зв’язку, на відміну від симетричного шифрування, що є важливою перевагою. Саме з цієї причини деякі асиметричні алгоритми зручно використовувати у якості протоколу узгоджнення ключів (KAP – Key agreement protocol).

Такий протокол дозволяє сторонам спілкуючись через відкритий канал встановити спільний секретний ключ, який можна в подальшому використати для симетричного шифрування, що часто і трапляється на практиці. Одним із найвідомих таких протоколів та першим у своєму роді є протокол Діффі — Геллмана (DH).

Оригінальна імплементація алгоритму виглядала так:

- X та Y погоджуються використовувати базові цілі числа g, p
- X загадує секретне ціле число a, і відправляє Y $A = g^a \cdot mod\ p$
- Y загадує секретне ціле число b, і відправляє X $B = g^b \cdot mod\ p$
- X обраховує $s = B^a \cdot mod\ p$
- Y обраховує $s = A^b \cdot mod\ p$
- X та Y отримали спільний секретний ключ

Для наочності цей процес можна схематично зобразити як на рисунку 1.10:

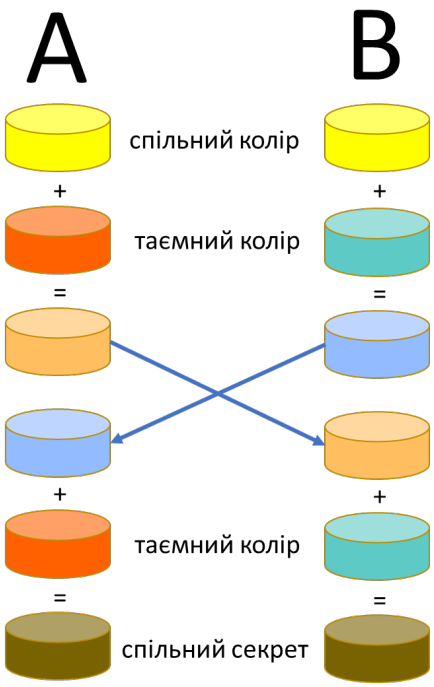

Рисунок 1.10 – Принцип роботи алгоритму Діффі – Геллмана

Всі дані крім a, b відправляються потенційно незахищеними каналами та можуть бути перехопленими. Отже, використовуючи вищеописаний алгоритм

сторони отримують однаковий ключ, ніколи не обмінюючись ним через незахищені канали; інформації ж, якою вони обмінюються, на практиці замало, що відновити ключ.

DH не є обмеженим двома учасниками, його модифіковану версію можна використовувати і для одночасного узгодження ключа багатьма користувачами. Технічно, цей алгоритм можна застосовувати і для генерації пар закритий-відкритий ключ, як типовий асиметричний алогоритм, але на практиці для таких цілей використовують звичний RSA, тому що останній додатково надає можливість підписувати сертифікати.

Найоптимальнішим варіантом алогоритму є протокол на еліптичних кривих ECDH (Elliptic-curve Diffie–Hellman). За винятком того, що ECDH покладається іншу математичну базу, та, як наслідок, має менші довжини ключів, в цілому він працює аналогічно до DH.

Традиційний DH математично базується на складності знаходження дискретних логарифмів, і доволі ресурсозатратний, в рази повільніший за традиційний RSA. ECDH набагато ефективніший, особливо при довших ключах, лише незначно поступаючись RSA (рисунок 1.11) [11].

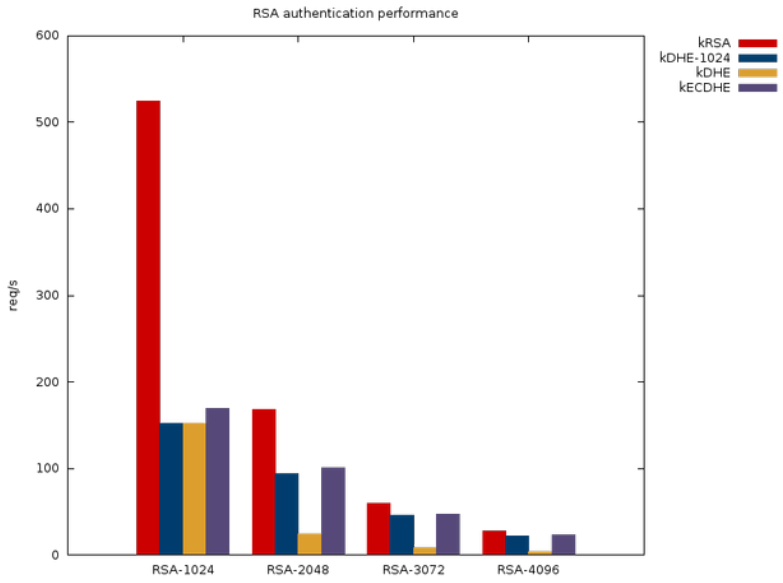

Рисунок 1.11 – Порівняння швидкості алгоритмів автентифікації

Для RSA та стандартного DH рекомендується обирати просте число розміром 2048 біт (у десятковому представленні таке число матиме 617 цифр). Використання ключів у 512, 1024 біт робить алогоритм вразливим до атак Logjam, що показала атака дослідників у 2015 році []

Цікаво, що DH має таку саму силу ключа, як і RSA для однакової довжини ключа, оскільки їх безпека зав'язана на спільних математичних проблемах, а еліптична криптографія дозволяє використовувати значно коротші ключі (в середньому лише у два рази довші, ніж відовідні для AES). Рекомендації NIST дають уявлення, які довжини ключів за різних алгоритмів дають співрозмірний рівень захису [12, 13]:

Таблиця 1.1 – Порівння розмірів ключів для різних алгоритмів

Розмір ключа (у бітах)			
Симетричні	Асиметричні		
AES	RSA	DH	ECDH
80	1024	1024	160
112	2048	2048	224
128	3072	3072	256
192	7680	7680	384
256	15360	15360	521

Властивості DH роблять його дуже корисним для імплементації прямої секретності. Пряма секретність (forward security), або цілковита пряма секретність (perfect forward security) — це властивість криптосистеми, яка гарантує, що навіть при втраті довготермінового ключа ключі попередніх сесій не будуть скомпрометовані [35]. Пряма секретність не є гарантією криптостійкості, вона просто гарантує, що злам одної сесії не приведе до зламу попередніх. Важливість прямої секретності була продемонстрована багом The Heartbleed, знайденим в OpenSSL у 2014, який зачепив гігантський сегмент мережі [37].

Щоб забезпечити пряму секретність, треба послуговатись протоколом Діффі — Геллмана з ефемерним (різним для кожної сесії) ключем (у TSL позначається

як EDH). У ECDH теж є свій "ефемерний" аналог - ECDHE. Навідміну від статичного DH, у EDH приватні ключі будуть зберігатись лише в оперативній пам’яті, і будуть стерті після закінчення сесії.

На практиці DH рідко використовується один, не в зв’язці з іншими алгоритмами, адже він не забезпечує автентифікацію і тому є вразлими для атак типу MITM. Зазвичай його використовують разом з RSA, який відповідатиме за автентифікацію. Поєднюють його також з Digital Signature Standard (DSS) з цих самих міркувань.

Тому наприклад набір шифрів (англ.cipher suite) в TLS виглядає так:

TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384

Як бачимо, протоколом обміну ключів є еліптичний Діффі — Геллман з ефемерним ключем, автентифікація відбувається за допомогою RSA, шифрування основних даних алгоритмом AES з довжиною ключа 256 у режимі дії GCM, SHA384 позначає хеш-функцію.

1.2.3 Хешування

Важливе місце в криптографії посідає хешування. Хеш-функція – математичний алгоритм, одностороння детерміністична функція, яка бере на вхід дані довільної довжини, і повертає дані фіксованої довжини, які називають значенням хешу, дайджестом повідомлення (англ. message digest), контрольною сумою, цифровим відбитком, тощо. Через це її ще називають функцією згортання. Технологію почали використовувати в кінці 1970-их.

Криптографічна хеш функція (CHF) має генерувати такий хеш, що обчислювально нереально:

- маючи дайджест h знайти повідомленн m таке, що h = hash(m), тобто не бути вразливим до атак знаходження першовзору першого типу
- маючи повідомлення m1 знайти таке повідомлення m2, що hash(m1) = hash(m2), тобто не бути вразливим до атак знаходження першовзору другого типу

- знайти такі повідомлення m1, m2, що hash(m1) = hash(m2), тобто не бути вразливим до колізійної атаки

На практиці ці вимоги означають, що найменші зміни вхідних даних до невпізнаваності змінять дайджест, це називається лавиновий ефект (англ. avalanche effect). Це робить їх зручним інструментом для перевірок цілісності даних.

Принципова різнция між хешування там шифруванням у тому, що дія першого задумувалась як незворотня, а шифрують з розрахунком на те, що до даних згодом потрібно буде отримати доступ.

З того, що повідомленю будь-якої довжини буде поставлено у відповідність значення хешу фіксованої довжини за принципом Діріхле випливає, що колізії неминучі, адже неможливо нескінченній кількості можливих вхідних даних поставити у відповідність скінченні варіанти дайджесту. Але на практиці величезна множина значень, які може набувати хеш (для SHA-256 це 2^{256}, для порівняння, кількість атомів у Всесвіті оцінюють у 2^{260}), означає, що шанс знайти колізії при достатній довжині хешу дуже малий, для SHA-256, на приклад, досі не було знайдено жодної.

Щоб знайти першовзір дайджесту теоритично потрібно 2^n операцій, де n – довжина хешу в бітах. Для того, щоб знайти колізії – $2^{\frac{n}{2}}$, через парадокс днів народження. Для знаходження першовзорів K хешів знадобиться $\log_2 K\, 2^n$, відповідно до Coupon collector's problem [].

Хеші слід застосовувати тоді, коли треба мати змогу порівнювати інформацію без потреби її зберігати. Це робить їх чудовим інструментом для проведення автентифікації, наприклад через логін та пароль. Якщо застосунок не потребує оперувати паролями від імені користувача (це може), нема потреби зберігати пароль навіть у зашифрованому вигляді – достатньо зберегти його хеш, що дозволить визначити, чи ввів користувач його правильно (рисунок 1.14).

Проте просто хешувати паролі не достатньо для їх захисту, адже це залишає простір для інших атак. Брутфорс-пошук першовзору сучасних хешів задача

непроста, але деколи цього і непотрібно. Можна взяти заготовані паролі, знайти для них хеші і пробувати всі по черзі. За базу можна взяти популярні паролі із різних витоків даних, і доповнити їх специфічною інформацією про жертву, як-от дата народження, якщо така є в наявності. Атака відома за назвою перебір за словником (dictionary attack). Такі атаки можуть бути двох видів: онлайн та офлайн.

Онлайн атака – це спроби пройти автентифікацію цільової системи в реальному часі, вона не вимагає ніяких попередніх даних. Але зазвичай вона мало перспективна, адже сильно обмежена швидкістю мережі (зазвичай виходить зробити 3 – 5 спроб в секунду), і кількістю спроб – після невдалого введення якусь кількість разів, можуть заблокувати IP-адресу нападника, або аккаунт користувача. Офлайн атака обмежена лише обчислювальними потужностями комп'ютера, тобто у тисячі разів швидша. Для її проведення треба заволодіти хешованим паролем.

Для брут-форс атак не треба багато простору, але треба багато часу. Перебір за словником займає багато фізичного місця (теребайти для пароля довжиною сім символів), але є набагато швидшим. Як компроміс між цими варіантами є ще третій, найоптимальніший: атака за райдужними таблицями (rainbow tables attack).

У такій таблиці зберігається лише початковий елемент та кінцевий результат певного ланцюжка обчислень. Щоб сформувати таку таблицю треба визначити детерміністичну функцію зведення R (англ. reduction function), яка співставляє дайджесту якийсь текст. Важливо розуміти, що вона не є зворотньою дією для хешування. Далі до тексту почергово застовують функції хешування та зведення. Зберігаються хіба значення першого та останнього стовпчиків (рисунок 1.12) [27].

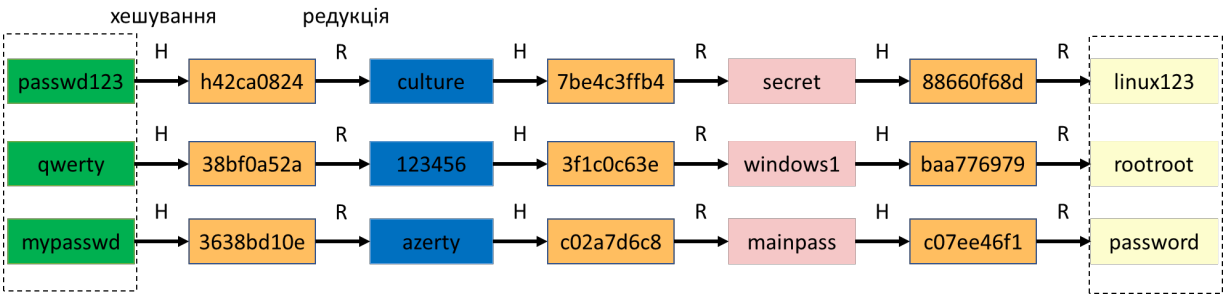

Рисунок 1.12 – Райдужна таблиця

Тепер, отримавши на вхід якийсь хеш h, ми знаходемо R(h) і пробуємо знайти це значення у другому стовпчику. Якщо воно там, це означає, що цей хеш був передостаннім елементом ланцюжка обчислень. Якщо ні, шукаємо R(H(R(h))) у другому стовпичку, і так далі. Отримавши потрібне значення другого стопця, ми використовуємо відповідне йому значення з першого стовпця, щоб відтворити текст, адже ми не можемо знайти першовзор дайджеста. До значення першого стовпця застосовують операціх H та R потрібну кількість разів (залежно від позиції h в даному ланцюжку) і отримують шуканий текст (рисунок 1.13).

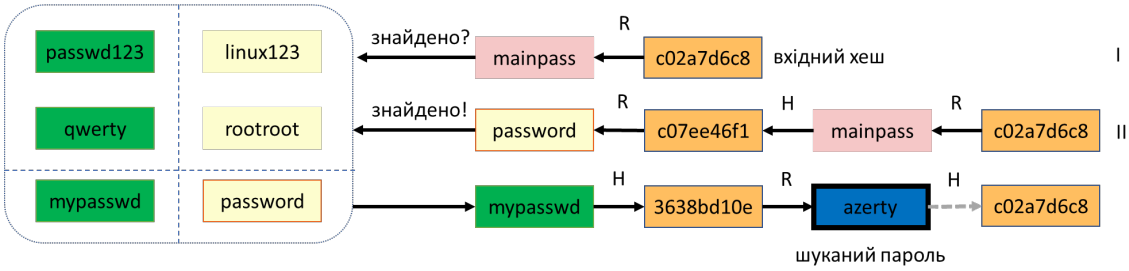

Рисунок 1.13 – Атака за райдужною таблицею

Щоб убезпечитись від такого роду атак, потрібно до тексту додавати модифікатор, «сіль» (англ. salt), яка має бути випадковою, але не обов'язково таємною, і лише потім хешувати.

Вона має генеруватись наново для кожного облікового запису, адже якщо використовувати однакову, однакові паролі будуть результувати в однакові хеші, чого варто уникати (reverse lookup table attack). Крім того, якщо така сіль буде скомпротована, всі записи опиняться під загрозою. Також сіль має бути достатньо

довгою, інакше можна створити таблиці для всіх її варіантів. NIST рекомендує довжину солі 128 бітів [25]. Використовувати у якості солі якісь інші дані користувача, як-то username – теж погана практика.

Сіль зберігається поряд із хешем в базі даних, і використовується в процесі верифікації (рисунок 1.14)

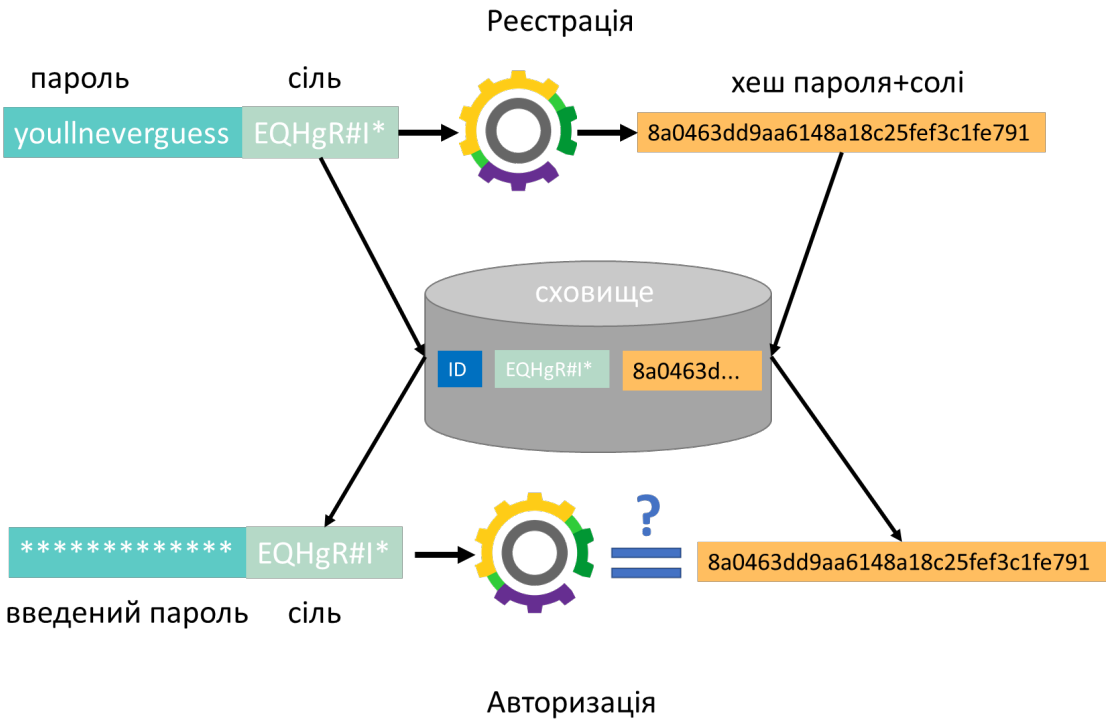

Рисунок 1.14 – Реєстрація та авторизація з хешем і сіллю

Хоча швидкість є перевагою для хеш-фенкції, в реальному житті це означає, що брутфорс атаки теж стануть ефективнішими, тому рекомендується сповільнювати автентифікацію. Зробити це допомагають функції формування ключа (англ. key derivation function, KDF). Наприклад, у 1Password (відомий менеджер паролів) використовують популярний PBKDF2 з кількістю ітерацій 100000 [24]. Це сильно сповільнює процес розрахунку хешу та автентифікації і для сервера, і для зловмисника. Тут варто знайти баланс між досвідом користувача і безпекою –автентифікація має займати стільки часу, щоб користувач не помічав різниці, але задача зловмисників набагато ускладнювалась.

Для безпеки критично важливо користуватись актуальними алгоритмами хешування. MD5, SHA-1 є застарілими і не рекомендовані до використання у

криптографії, хоча можуть використовуватись наприклад для перевірок цілісності. Атаку на MD5 здійснили ще у 2005 році, знайшовши 12 колізій, у 2017 Google заявив про можливість атаки колізіями на SHA-1. Є веб-ресурси, які ламають MD5 хеші за лічені секунди [23]. Популярним рішенням є SHA-256. Найбезпечнішим є SHA-3, який проте ще не набув значного поширення.

Хеші використовують у цифрових підписів. Дані, які потрібно підписати, хешуються, ключ шифрується і відправляється разом з даними. Отримувач розшифровує хеш, і порівнює з власним, якщо вони співпадають, підпис валідний (рисунок 1.15). Цифрові підписи гарантують цілісність та автентифікацію.

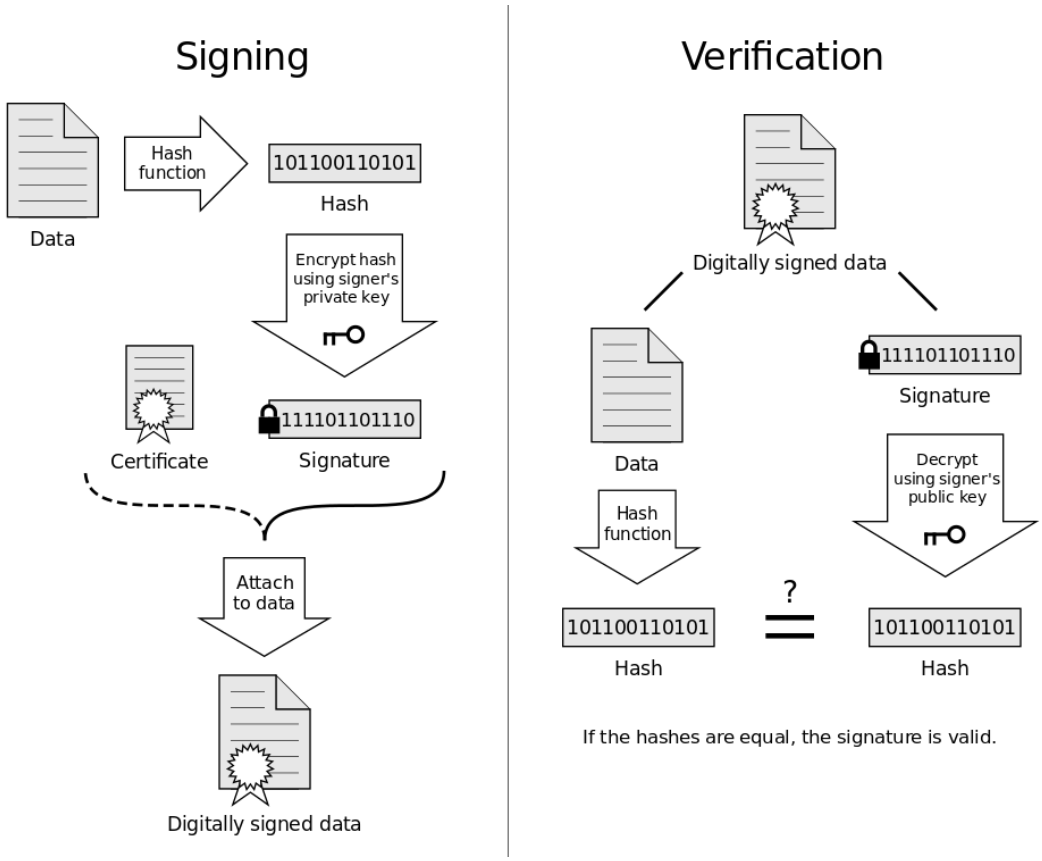

Рисунок 1.15 – Цифрові підписи

1.5 Джерело випадковості

Для отримання випадкових чисел використовують генератори випадкових чисел. Серед них розрізняють TRNGs (True random number generators) – генератори справді випадкових чисел, та DRNG/PRNG (Deterministic/Pseudo

random number generator) – генератори псевдовипадкових чисел. Ще є гібридні генератори, які поєднюють властивості двох попередніх [34].

TRNG не є алгоритмічними, вони генерують випадкові числа беручи дані якихось випадкових природних явищ, як-от флуктуації температури, радіоактивний розпад, тощо. Джерелом випадковості для TRNG також можуть слугувати дії людини, наприклад взаємодія користувача з програмним забезпенням. TRNG як правило повільніші, і можуть мати статистичні відхилення.

PRNG генерують послідовності псевдовипадкових значень за математичними законами, у яких є період, зазвичай доволі довгий, відштовхуючись від породжуючого елементу (seed), який подається йому на вхід. Важливо розуміти, що подаючи генератору однаковий породжуючий елемент, ми отримаємо ідентичні послідовності.

Проте далеко не кожен PRNG має властивості, потрібні для використання в криптографії. Ці властивості перевіряються спеціальними тестами, і лише генератор, що їх проходить, може називатись CPRNG (cryptographically secure pseudorandom number generator) – криптографічно стійким генератор псевдовипадкових чисел. Саме його зазвичай використовуть в прикладних застосунках для криптографічних алгоритмів.

У мовах програмування зазвичай є імплементації як звичайни PRNG, так і криптографічно стійких, тому наприклад в Java варто використувати клас java.security.SecureRandom, а не стандартний java.util.Random, яка бере за породжуючий елемент значення внтутрішнього годинника. На рисунку 1.16 різниця між ними продемонстрована графічно:

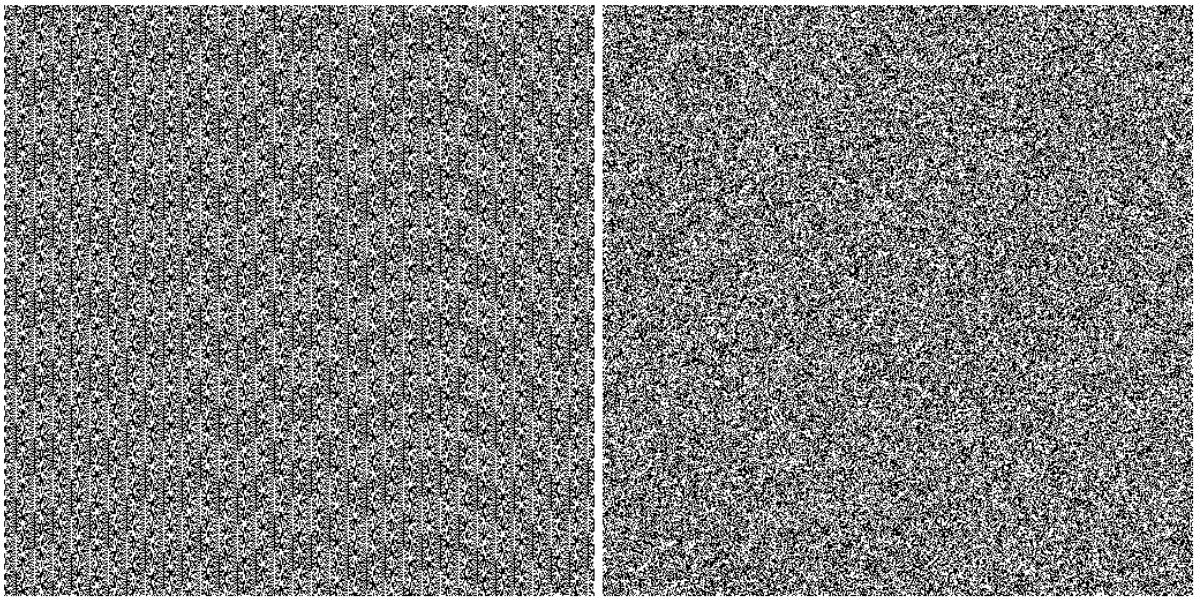

Рисунок 1.16 – Порівняння Random (зліва) та SecureRandom (справа)

Властивості різнх типів генераторів можна підсумувати таблицею:

Таблиця 1.2 – Порівняння властивостей генераторів

	PRNG	CPRNG	TRNG
Випадковість	+	+	+
Непередбачуваність	-	+	+
Невідтворюваність	-	-	+

1.7 Наскрізне шифрування

Наскрізне шифрування (англ. End-to-end encryption, E2EE) – спосіб передачі даних, в якому доступ читання та редагування повідомлень мають лише користувачі, які ці повідомлення надсилають. Це досягається за рахунок того, що закритий ключ ніколи не покидає кінцевий девайс, на якому був згенерований (рисунок 1.17). Таким чином, в системі з наскрізним шифруванням гарантується конфіденційність та цілісність.

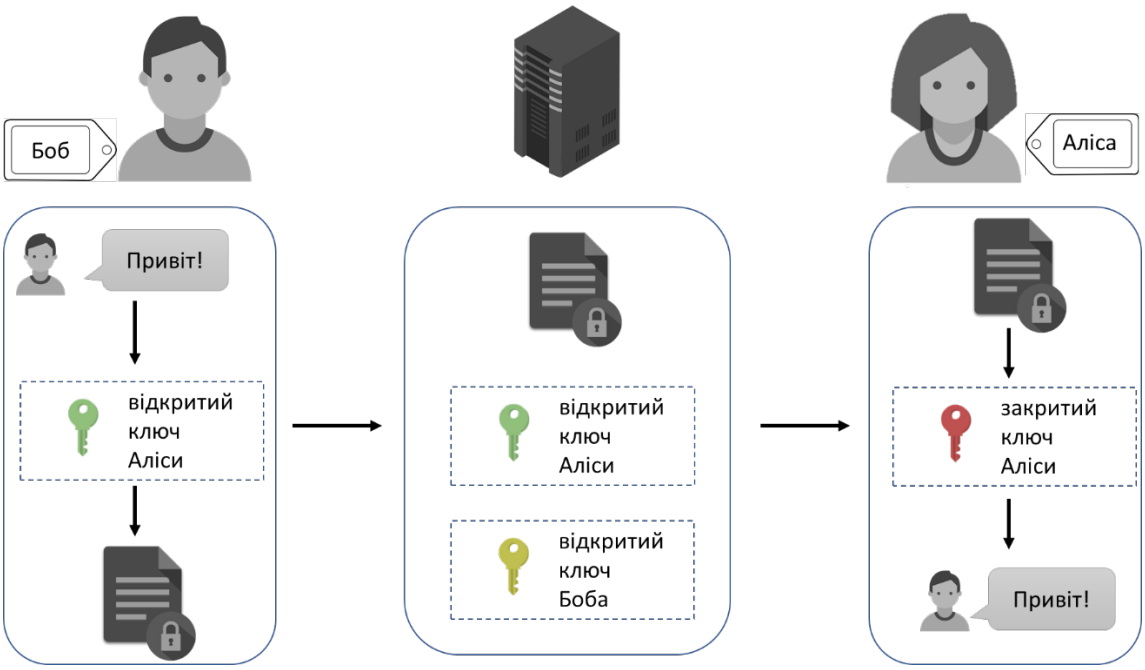

Рисунок 1.17 – Наскрізне шифрування

Це означає, що не потрібно довіряти ніякій третій стороні, навіть якщо її дані буде скомпротевоно, це не загрожуватиме конфіденційності користувачів. Зазвичай гарантується лише захищеність даних у процесі їх передачі по каналу між сервером та клієнтом (англ. encryption-in-transit), тобто користувач повинен довіряти власниками серверу.

Тому компанії, у застосунках яких імплементовано наскрізне шифрування, навіть на вимогу влади не в стані розкрити дані користувачів. Такий захист приватності може бути особливо важливим для активістів, журналістів, опозиціонерів тощо. Наскрізне шифрування втілене у багатьох відомих додатках: Telegram, Signal, WhatsApp, ProtonMail, Bitwarden, MEGA (хоча воно не завжди встановлене за промовчанням). Важливий поштовх публічній дискусії щодо використання наскрізного шифрування дав Едвард Сноуден після свої втечі у 2014 році.

Не варто вважати наскрізне шифрування панацеєю. Слід пам'ятати загальне правило криптографії – сила вашої криптосистеми визначається її найслабшим елементом.

У сервера все ще є інформація, хто і з ким і в який час обмінювався повідомленнями. Не варто також забувати про можливість заволодіти кінцевим пристроєм, на розшифровується повідомлення, або можливість перехопити ввід користувача. Технологія вразлива до атак MITM.

Тим не менше, технологія залишається золотим стандартом в індустрії і забезпечує найбільш безпечний спосіб комунікації.

1.8 Квантова загроза

Використовуючи алгоритм Шора для знаходження простих дільників цілих чисел, квантовий комп'ютер за розрахунками може нести загрозу для асиметричних алгоритмів як-от RSA, Diffie Hellman, ECC. Для зламу алгоритмів симетичного шифрування застосовують алгоритм Гровера, який допомагає знизити час затребуваний для брутфорс атаки до квадратного кореня (складність O(sqrt(N))), що все ще не становить загрози для AES-256, але вже робить небезпечним використання AES-128).[]

Виникає питання - чи можна досягти абсолютної криптографічної стійкості? Ця властивість доведена для шифру Вернама (або схеми одноразових блокнотів), винайденого у 1917 році співробітниками AT&T, при чому до ключа є такі вимоги:

1) Бути справді випадковим

2) Зберігатись в цілковитій таємниці

3) Не застосовуватись повторно

4) Мати розмір не менший, ніж розмір повідомлення

Було доведено, що не можливо досягти абсолютної криптографічної стійкості, використовуючи ключ, довжина якого менша за довжину повідомлення.

Абсолютна криптографічна стійкість означає, що криптосистему неможливо зламати, навіть маючи безлімітні часові і обчислювальні ресурси. Іншими словами, навіть перебравши усі варіанти ключів, ми не дізнаємось, яким було повідомлення, адже всі ці варіанти будуть рівноможливі.

Очевидно, що вимоги 3 і 4 викликають багато труднощів на практиці, тому широко ця модель не застосовується, адже якщо мати безпечний канал зв'язку для пересилки ключа довжиною не меншою, ніж саме повідомлення, доцільнише переслати власне саме повідомлення. З іншого боку, можна передавати ключі в нецифровий спосіб, наприклад передавати шифроблокнот з справді випадковими ключами, і використовувати як буде потреба, звідси і назва шифру.

2 Розробка захищеного застосунку

2.1 Обгрунтування вибору засобів розробки

Для розробки було обрано мову Java, що є типовим рішенням для клієнт-серверних застосунків, враховуючи її мультиплатформовість, і мінімальні вимоги – встановлена JRE (англ. Java Runtime Environment). Графічний інтерфейс реалізовано засобами JavaFX, ця технологія прийшла на зміну застарілому Swing, вона пропонує сучасніший інтерфейс та кращий досвід розробки.

2.2 Опис функціоналу

Безпекова архітектура реалізована вбудованими засобами – бібліотеками java.security.*, та javax.crypto.*, які надають усі необхідні для нашого застосунку можливості. Застосунок складається з двох частин – клієнта і сервера.

Після зупинки сервера спілкування припиняється, сервер же від приєднань та від’єднань користувачів не залежить. Усі користувачі спілкуються в одному просторі, біля кожного повідомлення наведено час його відправки та ім’я користувача-автора. Дублювання імен користувача не допускається. Пусте повідомлення відправити не можна. Про приєднання та від’єднання користувачів сервер сповіщає усіх. Кожен користувач спілкується через окремий сокет (англ. socket).

При приєднанні нового користувача, клієнт і сервер за допомогою протоколу узгодження ключа Діффі-Геллмана генерують спільний «секрет», який в подальшому використовується для створення ключа для AES, на якому базуватиметься подальше спілкування.

Зі сторони сервера цей відбувається так (рисунок 2.1):

```
DataOutputStream dos = new DataOutputStream(outStream);
DataInputStream dis = new DataInputStream(inStream);
System.out.println("SERVER generates DH keypair ...");

// generating key pair, size 2048
KeyPairGenerator serverKeyPairGenerator = KeyPairGenerator.getInstance("DH");
serverKeyPairGenerator.initialize(2048);
KeyPair serverKeyPair = serverKeyPairGenerator.generateKeyPair();

// Server creates and initializes her DH KeyAgreement object
System.out.println("SERVER initializes DH KeyAgreement object ...");
KeyAgreement serverKeyAgree = KeyAgreement.getInstance("DH");
serverKeyAgree.init(serverKeyPair.getPrivate());

// Server sends public key to the client
PublicKey serverPublicKey = serverKeyPair.getPublic();
byte[] serverPublicKeyEnc = serverPublicKey.getEncoded();
dos.writeInt(serverPublicKeyEnc.length);
dos.write(serverPublicKeyEnc);

//Server received client's public key
int bLen = dis.readInt();
byte[] theBytes = new byte[bLen];
dis.readFully(theBytes);

//Server instantiates client's public key
KeyFactory serverKeyFactory = KeyFactory.getInstance("DH");
X509EncodedKeySpec x509KeySpec = new X509EncodedKeySpec(theBytes);
PublicKey clientPublicKey = serverKeyFactory.generatePublic(x509KeySpec);
System.out.println("SERVER executes PHASE1 ...");
serverKeyAgree.doPhase(clientPublicKey, true);

//shared secret generated
byte[] sharedSecret = serverKeyAgree.generateSecret();
System.out.println("SECRET: " + (Arrays.toString(sharedSecret)));
System.out.println(bytesToHex(sharedSecret));

//deriving AES key from shared secret
SecretKeySpec aesKey = new SecretKeySpec(sharedSecret, 0, 16, "AES");
Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
cipher.init(Cipher.ENCRYPT_MODE, aesKey);

byte[] cleartext = "If you see this it means that encryption works!".getBytes();
byte[] ciphertext = cipher.doFinal(cleartext);
byte[] encodedParams = cipher.getParameters().getEncoded();
```

Рисунок 2.1 – Сервер узгоджує ключ

А зі сторони клієнта так (рисунок 2.2):

```
DataInputStream dis = new DataInputStream(inStream);
DataOutputStream dos = new DataOutputStream(outStream);

KeyFactory clientKeyFactory = KeyFactory.getInstance("DH");

//receiving info from server
int byteLength = dis.readInt();
byte[] theBytes = new byte[byteLength];
dis.readFully(theBytes);
X509EncodedKeySpec x509KeySpec = new X509EncodedKeySpec(theBytes);
PublicKey serverPublicKey = clientKeyFactory.generatePublic(x509KeySpec);
DHParameterSpec dhParamFromServerPubKey = ((DHPublicKey) serverPublicKey).getParams();

// Client creates his own DH key pair
System.out.println("CLIENT generates DH keypair ...");
KeyPairGenerator clientKeyPairGen = KeyPairGenerator.getInstance("DH");
clientKeyPairGen.initialize(dhParamFromServerPubKey);
KeyPair clientKeyPair = clientKeyPairGen.generateKeyPair();

// Client creates and initializes his DH KeyAgreement object
System.out.println("CLIENT initializes ...");
KeyAgreement clientKeyAgreement = KeyAgreement.getInstance("DH");
clientKeyAgreement.init(clientKeyPair.getPrivate());

// Client encodes his public key, and sends it over to Server.
byte[] clientPublicKeyEnc = clientKeyPair.getPublic().getEncoded();
dos.writeInt(clientPublicKeyEnc.length);
dos.write(clientPublicKeyEnc);

System.out.println("CLIENT executes PHASE1 ...");
clientKeyAgreement.doPhase(serverPublicKey, true);

//shared secret generated
byte[] sharedSecret = clientKeyAgreement.generateSecret();
System.out.println("Shared Secret:" + bytesToHex(sharedSecret));

System.out.println("Use shared secret as SecretKey object ...");
SecretKeySpec aesKey = new SecretKeySpec(sharedSecret, 0, 16, "AES");

//receiving AES-encrypted text
int msgL = dis.readInt();
byte[] ciphertext = new byte[msgL];
dis.readFully(ciphertext);
int msgL2 = dis.readInt();
byte[] encodedParams = new byte[msgL2];
dis.readFully(encodedParams);
```

Рисунок 2.2 – Клієнт узгоджує ключ

Після генерації спільного ключа шляхом спілкування по незахищеному каналу, одна сторона тепер може надсилати повідомлення, зашифровані за допомогою AES (рисунок 2.3):

```
byte[] cleartext = "If you see this it means that encryption works!".getBytes();
byte[] ciphertext = cipher.doFinal(cleartext);
byte[] encodedParams = cipher.getParameters().getEncoded();

//sending text encrypted with AES and cipher info
dos.writeInt(ciphertext.length);
dos.write(ciphertext);
dos.writeInt(encodedParams.length);
dos.write(encodedParams);
```

Рисунок 2.3 – Шифрування та надсилання повідомлення

А інша – розшифровувати (рисунок 2.4):

```
//decrypting
AlgorithmParameters aesParams = AlgorithmParameters.getInstance("AES");
aesParams.init(encodedParams);
Cipher cipher = Cipher.getInstance("AES/CBC/PKCS5Padding");
cipher.init(Cipher.DECRYPT_MODE, aesKey, aesParams);
byte[] recovered = cipher.doFinal(ciphertext);
System.out.println(new String(recovered));
```

Рисунок 2.4 – Отримання та розшифровка повідомлення

2.3 Тестування програми

При запуску сервера відображається таке вікно (рисунок 2.5):

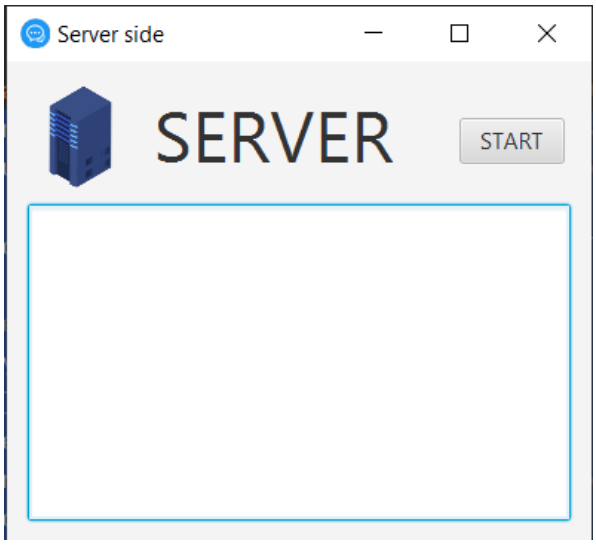

Рисунок 2.5 – Початкове вікно сервера

Для запуску сервера треба натиснути на кнопку START. Після запуску сервер сигналізує, на якому порті приймає сигнал, а кнопка тепер дає можливість його зупинити (рисунок 2.6).

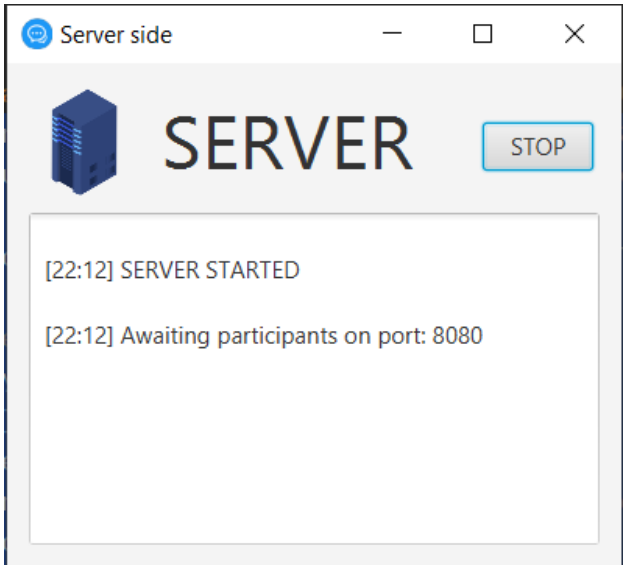

Рисунок 2.6 – Запущений сервер

Користувачу, щоб зв’язатись із сервером, потрібно ввести свій нікнейм у відопідне поле та натиснути кнопку Connect, яка після цього зміниться на Disconnect (рисунок 2.7).

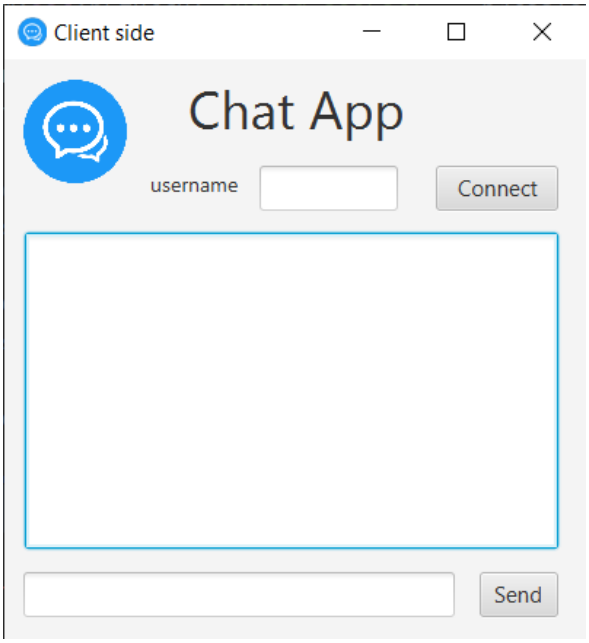

Рисунок 2.7 – Початкове вікно клієнта

Сервер інформує про етапи у формуванні спільного ключа за протоколом Діффі -Геллмана (Рисунок 2.8):

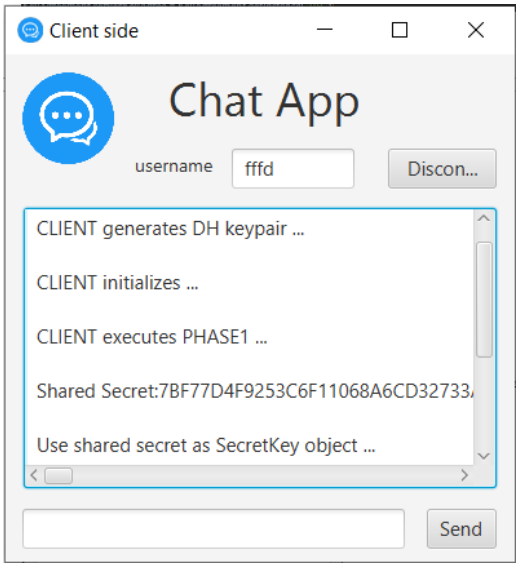

Рисунок 2.8 – Генерація спільного ключа

Інформація про доєднання відображається на обох сторонах (рисунок 2.9):

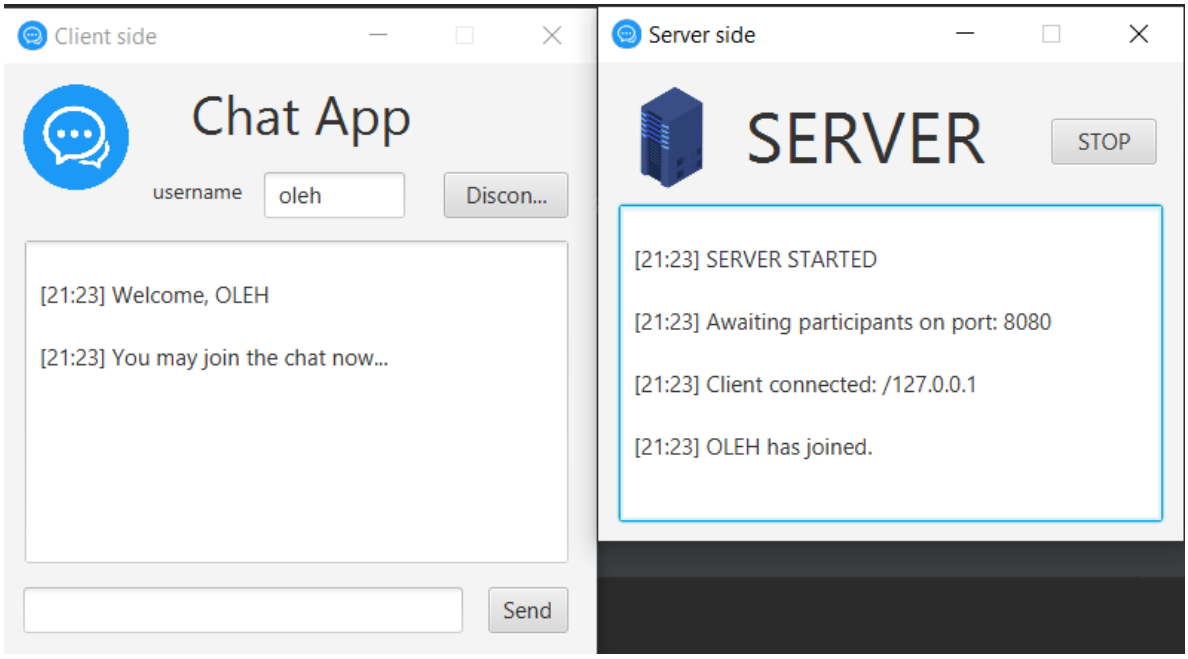

Рисунок 2.9 – Доєднання клієнта

Два користувачі після приєднання можуть спілкуватись у чаті (рисунок 2.10):

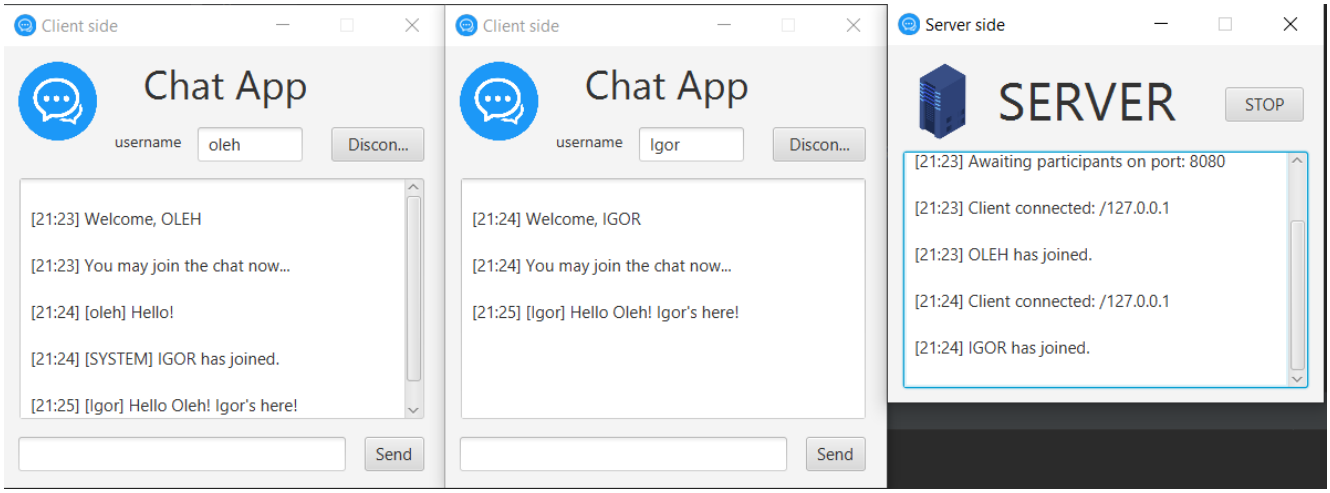

Рисунок 2.10 – Спілкування користувачів

Коли користувачі від’єднуються, сервер сповіщає про це учасників (рисунок 2.11):

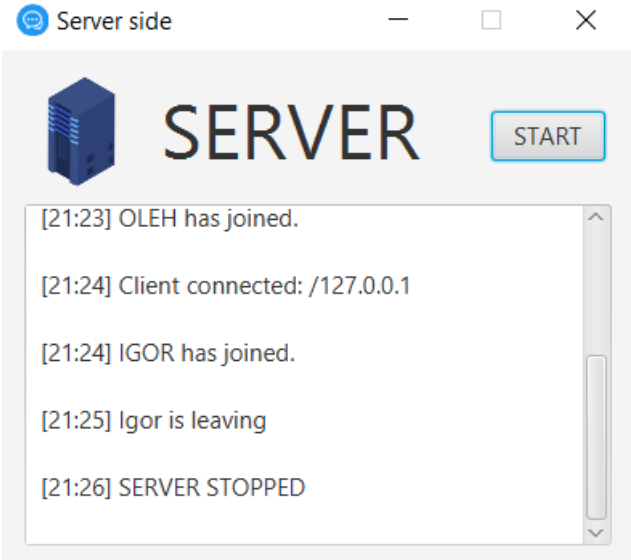

Рисунок 2.11 – Від’єднання користувача

Висновок

У цій роботі було проаналізовані засоби, якими досягається безпека інформації в контексті клієнт-серверної архітектури, розглянуто атаки багатьох типів на різні компоненти захисної архітектури та актуальні способи захисту від них. Показано еволюцію рівнів захисту, пояснено, чим обумовлені ті чи інші зміни. Надано практичні поради щодо оптимальних параметрів захисних алгоритмів, які базуються на рекомандаціях урядових організацій, а також наведено порівняння їх ефективності. Розроблено клієнт-серверний застосунок, на якому продемонстровано імплементацію та роботу попередньо розглянутих алгоритмів на практиці.

Робота залишає великий простір для доопрацювання, адже тема роботи включає у себе багато аспектів. Можна додатково розглянути питання надійного зберігання інформації на фізичному сервері, регулярної заміни ключів шифрування, тощо. Захист безпеки в застосунку можна доповнити автентифікацією користувачів, перевірками цілісності, наскрізним шифруванням, можливістю спілкуватись у приватних чатах.

Список використаних джерел

1. Joan Daemen, Vincent Rijmen, «The Design of Rijndael», Springer Science+Business Media, 2002 [Електронний ресурс] https://autonome-antifa.org/IMG/pdf/Rijndael.pdf
2. B. Kaliski, «PKCS #7: Cryptographic Message Syntax», RSA Laboratories, East, 1998 [Електронний ресурс] https://www.ietf.org/rfc/rfc2315.txt
3. «Padding schemes for block ciphers», CryptoSys PKI Pro Manual [Електронний ресурс] https://www.cryptosys.net/pki/manpki/pki_paddingschemes.html
4. Bill Buchanan, «Surely No-one Uses ECB Mode in AES? », 2020 [Електронний ресурс] https://medium.com/asecuritysite-when-bob-met-alice/surely-no-one-uses-ecb-mode-in-aes-332ed90f29d0
5. Daniel Voigt, «Breaking ECB by prepending your own message», 2020 [Електронний ресурс] http://bit.ly/ecb_breaking
6. NIST, «Block Cipher Techniques» [Електронний ресурс] https://csrc.nist.gov/projects/block-cipher-techniques/bcm
7. Blockchain.com, «Total Hash Rate», 2021 [Електронний ресурс] https://www.blockchain.com/en/charts/hash-rate?scale=1&timespan=all
8. Andrey Bogdanov, Dmitry Khovratovich, Christian Rechberger, «Biclique Cryptanalysis of the Full AES», 2011 [Електронний ресурс] https://eprint.iacr.org/2011/449.pdf
9. David Naccache, «Padding attacks on RSA», Information Security Technical Report, 1999 [Електронний ресурс] https://www.sciencedirect.com/science/article/abs/pii/S1363412799800855

10. Jon Crowcroft, «Public Key Cryptography», 1998 [Електронний ресурс] https://www.cl.cam.ac.uk/~jac22/books/mm/book/node335.html
11. Hubert Kario, «RSA And ECDSA Performance», SecurityPitfalls 2014 [Електронний ресурс] http://bit.ly/rsa_tests
12. N.I.o.S.a. Technology, «Recommended Elliptic Curves for Federal Government Use», 1999.
13. Elaine Barker, Quynh Dang, «Recommendation for Key Management», NIST, 2015 https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57Pt3r1.pdf
14. Alex Biryukov, David Wagner, “Slide Attacks”, Fast Software Encryption, 1999 [Електронний ресурс] https://link.springer.com/content/pdf/10.1007/3-540-48519-8_18.pdf
15. «Data Encryption Standard (DES), Lecture 4», Laboratory for Computer Science at Université Paris-Sud [Електронний ресурс] https://www.lri.fr/~fmartignon/documenti/systemesecurite/4-DES.pdf
16. «Reid», «Can AES decryption be used as encryption?», Cryptography Stack Exchange, 2013 [Електронний ресурс] https://crypto.stackexchange.com/a/9671
17. Dr. Lily Chen, «AES Development», NIST, 2016 [Електронний ресурс] https://csrc.nist.gov/projects/cryptographic-standards-and-guidelines/archived-crypto-projects/aes-development
18. «National Policy on the Use of the Advanced Encryption Standard (AES) to Protect National Security Systems and National Security Information», NIST, 2003 [Електронний ресурс] https://csrc.nist.gov/csrc/media/projects/cryptographic-module-validation-program/documents/cnss15fs.pdf
19. «hardyrama», «Why do block ciphers use key schedules instead of round constants?», Cryptography Stack Exchange, 2018 [Електронний ресурс]

https://crypto.stackexchange.com/a/62067

20. Jialin Huang, Xuejia Lai, «Revisiting Key Schedule's Diffusion In Relation With Round Function's Diffusion», Department of Computer Science and Engineering Shanghai Jiaotong University, 2012 [Електронний ресурс] https://eprint.iacr.org/2012/415.pdf
21. «Biv», «Does changing the order of the steps within a round affect the security of AES?», Cryptography Stack Exchange, 2016 [Електронний ресурс] https://crypto.stackexchange.com/a/41037
22. Haseeb Qureshi, «Hash Functions», nacamoto.com, 2013 [Електронний ресурс] https://nakamoto.com/hash-functions/#collision-resistance
23. [Електронний ресурс] https://crackstation.net/
24. «How PBKDF2 strengthens your Master Password», 1Password Support, 2020 [Електронний ресурс] http://bit.ly/1passwd_pbkfd2
25. Meltem Sönmez Turan, Elaine Barker, William Burr, Lily Chen, «Recommendation for Password-Based Key Derivation», 2010 [Електронний ресурс] https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-132.pdf
26. Енциклопедія сучасної України, «Криптографія», 2014 [Електронний ресурс] http://esu.com.ua/search_articles.php?id=1576
27. Ryan Sheasby, «Rainbow Tables (probably) aren't what you think», 2010 [Електронний ресурс] https://rsheasby.medium.com/rainbow-tables-probably-arent-what-you-think-30f8a61ba6a5
28. OWASP, «Password Storage Cheat Sheet», [Електронний ресурс] https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html

29. «Timing vulnerabilities with CBC-mode symmetric decryption using padding», Microsoft, 2020 [Електронний ресурс] http://bit.ly/cbc_danger
30. Antoine Joux, «Authentication Failures in NIST version of GCM», Universit´e de Versailles St-Quentin-en-Yvelines, [Електронний ресурс]
31. Salil Vadhan, «Private-Key Encryption in Practice», Harvard, 2013 [Електронний ресурс] https://people.seas.harvard.edu/~salil/cs127/fall13/lec13.pdf
32. Robert Heaton, «The Padding Oracle Attack», 2013 https://robertheaton.com/2013/07/29/padding-oracle-attack/
33. Nick Sullivan, «Padding oracles and the decline of CBC-mode cipher suites», Cloudfare, 2016 [Електонний ресурс] http://bit.ly/cloud_fare_oracle_attack
34. Oto Petura, «True random number generators for cryptography: Design, securing and evaluation», Université de Lyon, 2019 [Електронний ресурс] https://tel.archives-ouvertes.fr/tel-02895861/document
35. Colin Boyd, Kai Gellert, «A Modern View on Forward Security», NTNU, Norwegian University of Science and Technology, Trondheim, 2019 [Електронний ресурс] https://eprint.iacr.org/2019/1362.pdf
36. Josh Lake, «What is the Diffie–Hellman key exchange and how does it work? », Comparitech, 2021 [Електронний ресурс] https://www.comparitech.com/blog/information-security/diffie-hellman-key-exchange/
37. Zakir Durumeric1, James Kasten1, David Adrian1, J. Alex Halderman1, Michael Bailey, «The Matter of Heartbleed», IMC '14: Internet Measurement Conference Vancouver BC Canada, 2014 [Електронний ресурс] https://dl.acm.org/doi/pdf/10.1145/2663716.2663755