Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

КВАЛІФІКАЦІЙНА РОБОТА

освітній ступінь – бакалавр

на тему: **«ЗАСТОСУВАННЯ КРИПТОГРАФІЧНИХ МЕТОДІВ ШИФРУВАННЯ В РОЗРОБЦІ ВЕБ-СЕРВІСУ ДЛЯ БЕЗПЕЧНОГО ЗБЕРІГАННЯ ЧУТЛИВИХ ДАНИХ»**

Виконала:
студентка 4-го року навчання
спеціальності 122-Комп’ютерні науки
Бойко Інна Іванівна
Керівник ст. викладач
Кирієнко Оксана Валентинівна
Рецензент_____
Кваліфікаційна робота оцінена
за оцінкою_____
Секретар ЕК _____

Київ 2025

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ

Зав. кафедри інформатики,

к. ф.-м. н. С. С. Гороховський

_____ (підпис)

“____” ___ ____ 2024 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

кваліфікаційної роботи

студентки Бойко Інни Іванівни факультету інформатики 4 курсу

Тема: Застосування криптографічних методів шифрування в розробці веб-сервісу для безпечного зберігання чутливих даних.

Зміст ТЧ до кваліфікаційної роботи:

Вступ

1. Теоретичні відомості
2. Аналіз поточного рівня захищеності обраного веб-сервісу організації та аналіз алгоритмів шифрування.
3. Реалізація практичної частини

Висновки

Список джерел

Дата видачі “_25_” ___жовтня___ 2024р.

Керівник _____ Кирієнко О.В.
(підпис) (прізвище та ініціали)

Завдання отримано _____ Бойко І.І.
(підпис) (прізвище та ініціали)

Календарний план виконання кваліфікаційної роботи

№ п/п	Перелік робіт	Термін виконання	Дата ознайомлення наукового керівника	Підпис наукового керівника	Примітка
1.	Вибір теми, затвердження її на засіданні кафедри та закріплення наукового керівника	жовтень 2024р.	20 жовтня 2024р.		
2.	Узгодження календарного графіка підготовки кваліфікаційної роботи.	жовтень 2024р.	25 жовтня 2024р.		
3.	Складання плану кваліфікаційної роботи та узгодження з науковим керівником	листопад 2024р.	2 листопада 2024р.		
4.	Вивчення джерел літератури, матеріалів, наукових статей, збір та узагальнення фактів, даних	листопад 2024р.	28 листопада 2024р.		
5.	Написання розділу 1 кваліфікаційної роботи (постановка проблеми, теоретичні основи, обґрунтування обраних методів)	грудень 2024р.	20 грудня 2024р.		
6.	Розробка веб-сервісу спираючись на зібрані дані.	січень – березень 2025р.	25 березня 2025р.		
7.	Написання розділу 2 (опис архітектури веб-сервісу, обґрунтування реалізації захисту даних)	січень 2025р.	30 січня 2025р.		
8.	Проміжний контроль виконання роботи	березень 2025р.	25 березня 2025р.		
9.	Написання розділу 3 (тестування та оцінка рівня безпеки веб-сервісу, аналіз результатів)	березень 2025р.	2 квітня 2025р.		

10.	Повне завершення написання кваліфікаційної роботи, оформлення її згідно з вимогами й подання на відгук науковому керівнику	квітень 2025р.	29 квітня 2025р.		
11.	Подання кваліф. роботи для перевірки письмових робіт студентів НаУКМА на відповідність вимогам академічної доброчесності	травень 2025р.	1 червня 2025р.		
12.	Подання на зовнішню рецензію	травень 2025р.	2 червня 2025р.		
13.	Підготовка до захисту кваліфікаційної роботи на засіданні кафедри: написання доповіді та створення презентації	травень 2025р.	20 травня 2025р.		
14.	Попередній захист кваліфікаційної роботи на засіданні кафедри	травень 2025р.	28 травня 2025р.		
15.	Подання кваліфікаційної роботи на кафедру з усіма супроводжувальними документами	травень 2025р.	1 червня 2025р.		
16.	Публічний захист кваліфікаційноїроботи перед екзаменаційною комісією	згідно з роботи ЕК	5 червня 2025р.		

Графік узгоджено “__25___” __жовтня___ __2024____ р.

Науковий керівник _____Кирієнко О.В._____ _____

Виконавець кваліфікаційної роботи ___Бойко Інна Іванівна__ _____

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА ПРИЙНЯТИХ СКОРОЧЕНЬ

ADD – Additional Authenticated Data; додаткові дані автентифікації;

AEAD – Authenticated Encryption with Associated Data; алгоритм з автентифікованим шифруванням;

AES – Advanced Encryption Standard; просунутий стандарт шифрування даних;

API – Application Programming Interface;

CA – Certificate Center; центр сертифікації.

CBC – Cipher Block Chaining; режим ланцюгового шифрування;

CRUD – Create, Read, Update, Delete; створення, читання, оновлення, видалення;

CTR – Counter; лічильниковий режим;

DTO – Data Transfer Object; Об’єкт передачі даних;

ECIES – Elliptic Curve Integrated Encryption Scheme; еліптична шифрувальна схема з інтегрованою аутентифікацією;

ECB – Electronic Codebook; електронний шифрувальний блок;

GCM – Galois/Counter mode; режим Galois/Counter;

GHASH – Galois Hash; Galois шифрування;

GNFS – General Number Field Sieve; загальний багатофакторний метод;

HR – Human Resources; управління персоналом;

HTTP – HyperText Transfer Protocol; протокол передачі гіпертексту;

ID – identifier; ідентифікатор;

IBM – International Business Machines Corporation;

JDK – Java Development Kit; комплект розробника Java;

JPA – Java Persistence API; API для збереження даних у Java;

JSON – JavaScript Object Notation; нотація об’єктів JavaScript;

KV – “key-value”; пара ключ-значення;

MAC – Message Authentication Code; код автентифікації повідомлення;

NIST – National Institute of Standards and Technology; Національний Інститут Стандартів та Технологій;

NPM – Node package manager; менеджер пакетів Node.js;

OAuth – Open Authorization; відкритий протокол авторизації;

OWASP – Open Web Application Security Project; проєкт безпеки веб-додатків;

PCI – Payment Card Information; інформація про платіжні картки;

PII – Personally Identifiable Information; персональні дані ідентифікації;

PHI – Protected health Information; захищені медичні дані;

POODLE – Padding Oracle on Downgraded Legacy Encryption, атака зниження версії TLS;

RC4 – Rivest Cipher 4;

RBAC – Role-based access; керування доступом на основі ролей;

RSA – Rivest-Shamir-Adleman; алгоритм Рівеста-Шаміра-Адлемана;

RxJS – Reactive Extensions for JavaScript; реактивні розширення для JavaScript;

SAC – Strict Avalanche Criterion; критерій стійкості шифру;

S-box – Substitution box; коробка підстановки;

SQL – Structured Query Language; мова структурованих запитів;

SSO – Single Sign-On, єдиний логін;

TLS – Transport Layer Security; захищений шар транспортного рівня;

URL – Uniform Resource Locator; уніфікований локатор ресурсу;

XOR – Exclusive OR ; виключне або;

СУБД – Система управління бази даних

ВСТУП

Стрімкий розвиток технологій спричиняє щоденне збільшення обсягу даних, які обробляються та зберігаються у веб-сервісах. Значна їх частина є високочутливими: це персональна інформація, фінансові дані документи тощо. Оскільки дані стають одним з найцінніших цифрових ресурсів, кількість кібератак, основною метою яких є несанкціонований доступ до даних, постійно зростає – постійно розробляються нові способи використання вразливостей інформаційних систем організацій.

У руках зловмисників дані набувають реальної цінності, адже вони можуть бути використані для шантажу, маніпуляцій, подальших атак, нанесення різного роду шкоди компаніям, а також окремим особам.

Витік даних є одним із найнебезпечніших наслідків таких атак; він може призвести до значних фінансових втрат, репутаційних ризиків і юридичних наслідків. Це підтверджує звіт IBM за 2024 рік, згідно з яким середня вартість витоку даних досягла 4,88 мільйонів доларів [1], що підкреслює масштаб фінансових втрат від таких інцидентів.

Тому питання забезпечення конфіденційності, цілісності даних та недоступності для сторонніх осіб набуває першочергового значення для будь-якої сучасної організації.

Одним із ключових засобів захисту даних є шифрування. За результатами дослідження Operationalizing Encryption and Key Management (Fortanix, ESG, 2024)[2], 90% опитаних фахівців підтвердили, що шифрування позитивно впливає на безпеку даних, мережі та організації загалом, а понад 50% вважають цей вплив суттєвим. Водночас відсутність шифрування залишалася основною причиною втрати чутливих даних для майже 33% організацій. Саме шифрування даних стає останньою лінією оборони, що дозволяє зберегти інформацію

недоступною для зловмисників навіть у разі її викрадення. Дослідження IBM показують, що компанії, які впровадили шифрування, знизили витрати при витоку даних в середньому на $360 000 [3].

Але навіть з такими показниками, у багатьох реальних інформаційних системах ще досі відсутнє шифрування на рівні зберігання чи передавання даних, зустрічається використання застарілих криптографічних протоколів та недбале управління ключами, що створює критичні вразливості.

Дана робота присвячена впровадженню сучасних криптографічних засобів захисту даних у веб-сервіс, який зберігає персональні та фінансові дані. В ході дослідження буде виконано повноцінний аналіз рівня захищеності системи, виявлено її вразливості, проаналізовано ризики методологією оцінки ризиків та розроблено прототип обраного веб-сервісу, в якому буде враховані виявлені ризики та впроваджені відповідні методи захисту. Також буде проведено експериментальне порівняння криптографічних алгоритмів шифрування на основі даних, що відповідають структурі реального веб-сервісу, щоб визначити найбільш ефективний алгоритм з точки зору ефективності, безпеки та доцільності використання у веб-застосунку, який обробляє чутливу інформацію.

Мета: Підвищення рівня безпеки веб-сервісу для управління персоналом шляхом застосування сучасних криптографічних методів, впровадження механізму управління ключами та усунення критичних загроз, пов’язаних із зберіганням чутливих даних.

Об’єктом дослідження є процеси забезпечення інформаційної безпеки веб-сервісів, призначених для зберігання та обробки чутливих персональних даних працівників організації.

Предметом дослідження є методи забезпечення конфіденційності, цілісності та недоступності даних за допомогою криптографічних алгоритмів (AES, ChaCha20 тощо).

Результатом дослідження стане розроблений прототип веб-сервісу - HR система для роботи з даними працівників організації, в якому реалізовано підвищений рівень захищеності за рахунок впровадження оптимального криптографічного алгоритму, інтегрування механізму управління ключами та інших методів захисту.

Кваліфікаційна робота складається з вступу, трьох розділів та списку використаних джерел.

У першому розділі роботи обґрунтовується поняття чутливих даних і криптографічних методів їх захисту, окреслюються критерії оцінки алгоритмів шифрування та можливі способи їх криптоаналізу, окреслюються найкращі практики керування криптографічними ключами, а також формується поняття рівня захищеності веб-сервісу.

У другому розділі аналізується веб-сервіс управління персоналом на наявність вразливостей методологією оцінки ризиків та пропонуються заходи безпеки спрямовані на підвищення рівня захищеності веб-сервісу для забезпечення цілісності та конфіденційності даних. Крім того, виконуєтсья дослідження обраних алгоритмів шифрування з позиції оцінки їх безпекової стійкості та формується завдання кваліфікаційної роботи.

У третьому розділі визначаються функціональні вимоги до веб-сервісу, проектується клієнтська та серверна архітектура та описуються використані інструменти під час розробки. Крім того, проводиться тестування обраних алгоритмів шифрування з погляду продуктивності, і на основі отриманих результатів обирається найоптимальніше рішення з урахування безпекової стійкості та продуктивності. Окремо описуються інші впровадження заходи безпеки: механізм управління ключами, автентифікація з OAuth 2.0, забезпечення захищеного каналу передачі даних, а також оцінка стійкості обраного криптографічного алгоритму .

1.ТЕОРЕТИЧНІ ВІДОМОСТІ

1.1 Класифікація даних

Чутливі дані – це дані, втрата, модифікація, неправильне використання чи потрапляння до сторонніх осіб яких може мати негативні репутаційні, фінансові, правові чи операційні наслідки для організації.

Класифікація даних за рівнем чутливості – це процес присвоєння даним певного рівня чутливості. Для визначення рівня чутливості аналізуються такі питання: кому належать дані, хто має мати до них доступ, якими будуть наслідки при потраплянні цих даних до рук неавторизованих осіб.

Основні рівні класифікації даних:

1. Висока чутливість – інформація, яка підлягає захисту та постійного моніторингу. Зазвичай така інформація регулюється нормативно-правовими актами як така, що вимагає суворого контролю доступу та спеціальних заходів захисту. В більшості випадків саме високочутливі дані стають головною ціллю зловмисників.
2. Середня чутливість – дані, які не призначені для публічного розголошення, однак їх витік не приведе до критичних наслідків для компанії.
3. Низька чутливість – загальнодоступна інформація, яка не потребує спеціального захисту.

Такі дані зазвичай групуються в кілька основних категорій (список не є вичерпним):

1. PII (Personally identifiable information) – інформація, яка дає змогу прямо або опосередковано ідентифікувати особу. До неї належить ім’я, дата народження, податковий номер, паспортні дані, номер телефону, електронна адреса, фінансові та біометричні дані.

2. PHI (Protected health information) – захищена медична інформація, пов’язана зі станом здоров’я, медичною історією лікування. До цієї категорії належать медичні записи, результати діагностики, рецепти, дані медичного страхування. У багатьох країнах така інформація регулюється спеціальними законами і вимагає підвищених заходів захисту.
3. PCI (Payment Card Information) – дані платіжних карток, що включають номер картки, код безпеки, ім’я власника, термін дії картки, дані з магнітної стрічки або чипа, PIN - код.

1.2 Криптографічні методи захисту інформації

Криптографічні методи захисту інформації - це математичні алгоритми та процедури, які використовуються для збереження конфіденційності та цілісності чутливих даних.

До криптографічних методів захисту інформації належать:

1. шифрування даних - перетворення відкритої інформації у зашифрований вигляд для збереження її конфіденційності при передачі чи зберігання;
2. хешування даних - функції генерування “хеш значення” на основі даних для забезпечення їх цілісності;
3. цифровий підпис - комбінація асиметричного шифрування та хешування, що забезпечує цілісність даних та верифікацію їх авторства.

1.3 Поняття шифрування даних

Шифрування даних – це криптографічна техніка захисту, яка перетворює дані у зашифрований (недоступний для читання) формат за допомогою

алгоритму шифрування та криптографічних ключів. Такий підхід забезпечує конфіденційність даних під час їх передачі чи зберігання.

1.3.1 Симетричне шифрування

Симетричне шифрування - це тип шифрування, який передбачає використання одного ключа для зашифровування та розшифровування даних. Цей ключ знають обидві сторони і одна сторона ним зашифровує дані, а інша - розшифровує.

Симетричні алгоритми шифрування класифікують за типом: блоковий (block cipher) чи потоковий (keystream cipher).

1.3.1.1 Блокові алгоритми шифрування

При блоковому шифруванні вхідний текст розбивається на послідовні блоки фіксованої довжини. Кожен блок шифрується шляхом виконання кількох раундів. Кожен раунд включає підстановки, перестановки, XOR з ключем або частиною ключа та інші операції залежно від алгоритму. До блокових алгоритмів шифрування належать DES, 3DES, AES та Blowfish.

Головна перевага симетричних алгоритмів шифрування – швидкість. Вони обробляють дані набагато швидше, ніж асиметричні алгоритми тому що працюють з простішими математичними операціями. Саме тому вони краще підходять для великих обсягів даних.

DES (Data Encryption Standard) – ранній стандарт симетричного блокового шифрування, затверджений NIST у 1977 році. Він використовує ключ довжиною 56 біт і шифрує дані блоками по 64 біти. Алгоритм належить до класу шифрів

Фейстеля - процес шифрування відбувається у 16 раундів, на кожному з яких дані поділяються на дві частини та проходять серію перестановок, замін та побітних операцій XOR з ключем раунду. Через довжину ключа алгоритм вважається недостатньо стійким до атак методом повного перебору, що робить його непридатним для використання в сучасних інформаційних системах.

3DES – покращений варіант алгоритму DES, який використовує механізм потрійного шифрування даних, в якому для шифрування застосовується три 56-бітні ключі (сумарно 168 бітів) та виконує послідовне шифрування одного і того ж блоку цими ключами. Хоча він є стійкішим до атак, через блоки 64 біти він не є достатньо ефективним порівняно з сучасними алгоритмами шифрування.

Blowfish – симетричний блоковий алгоритм шифрування, що шифрує інформацію блоками по 64 біти, та дає можливість використовувати ключі різної довжини (від 32 до 448 біт). Як і DES, Blowfish належить до класу шифрів Фейстеля та використовує серії перестановок, замін та побітних операцій XOR з ключем раунду.

AES[4] (Advanced Encryption Standard) – алгоритм шифрування, розроблений Національним Інститутом Стандартів та Технологій (NIST) у 2001 році. Це блочний алгоритм шифрування, який шифрує дату блоками в 128 бітів (16 байтів). Для шифрування можуть використовуватись ключі розмірів 128, 192, 256 біт.

На вхід алгоритму отримуємо початковий незашифрований текст, який проходить через певну кількість раундів шифрування, і в кінцевому результаті отримується зашифрований текст того самого розміру, як і початкові дані.

Залежно від розміру ключа алгоритм буде мати різну кількість раундів – 10 раундів для ключа 128 біт, 12 раундів для ключа 192 біт та 14 раундів для ключа 256 біт. Головний ключ використовується для створення допоміжних ключ для кожного раунду.

В кожному раунді алгоритм працює з даними як з матрицею 4x4 байти і виконує такі 4 кроки:

1. SubBytes – заміна кожного байту іншим байтом відповідно до таблиці S-box.
2. ShiftRows – зміщення рядків матриці. Перший рядок не зміщується, другий рядок зміщується на один байт вліво, третій рядок зміщується на два байти вліво, четвертий – на три байти вліво.
3. Mix Columns – операція, де кожен стовпчик матриці множиться на фіксовану матрицю MDS.
4. Add Round Key – обчислення результату, шляхом застосування операції XOR між даними з попереднього кроку та ключем цього раунду.

Розшифрування полягає в застосуванні інверсних функцій, які скасовують попередні перетворення і в результаті повертають дані в початковий розшифрований стан. Відбувається та сама кількість раундів і кожен з них складається з таких кроків:

1. Add Round Key - застосування операції XOR між зашифрованими даними та ключем раунду.
2. Inverse Mix Columns – виконання зворотної матричної операції над кожним стовбцем матриці, шляхом множення їх на константну матрицю, визначену в стандарті.
3. Inverse ShiftRows – виконання зсуву рядків блоку вправо з метою відновлення початкових позицій байтів.
4. Inverse SubBytes – операція заміни кожного байта іншим відповідно до зворотної таблиці S-box.

Також для обробки тесту, який більший за один блок заданого розміру, можуть використовуватись різні режими шифрування:

1. ECB (Electronic Codebook) – режим шифрування, у якому кожен блок тексту шифрується незалежно.

2. CBC (Cipher Block Chaining) – режим шифрування, у якому для кожного блоку даних перед шифруванням виконується XOR операція з попереднім вже зашифрованим блоком даних. Для початкового блоку виконується XOR операція з спеціальним випадковим вектор ініціалізації (IV). Такий підхід створює зв’язок між блоками, унаслідок чого кожен наступний блок залежить від попереднього, підвищуючи стійкість шифру.
3. CTR (Counter mode) - режим шифрування, який використовує спеціальний лічильник. Для кожного наступного блоку лічильник обчислюється як IV (вектор ініціалізації) + n (номер блоку). Алгоритм шифрує цей лічильник AES алгоритмом, а зашифрований текст – результат операції XOR між зашифрованим лічильником та блоком початкового тексту. Такий підхід дає можливість паралельного шифрування, адже блоки не залежать один від одного.
4. GCM (Galois/Counter) – режим шифрування, який поєднує CTR та створення тегу автентифікації. На рисунку 1.1 окреслено головні кроки створення тегу автентифікації алгоритмом GCM.

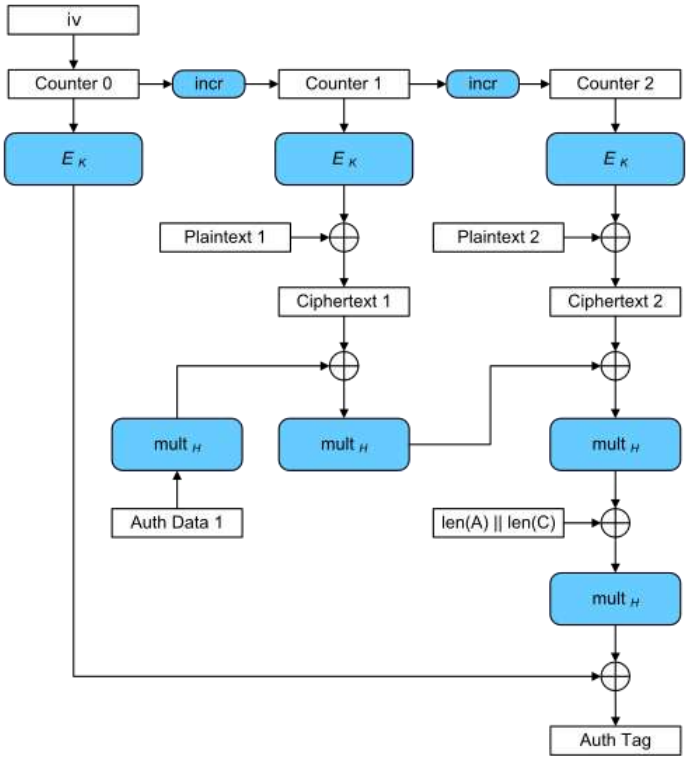


Рис. 1.1. Схема роботи алгоритму AES-GCM[5]

Першим кроком створення тегу автентифікації шифрується блок з 128 нулів заданим криптографічним ключем, утворюючи підключ H. На кожному етапі відбувається поліноміальна обробка: ADD як перший блок та зашифровані AES алгоритмом наступні блоки (Y_i) використовуються для рекурсивного обчислення X (GHASH значення).

$$X_0 = 0, X_i = (X_{i-1} \oplus Y_i) * H$$

Після обробки всіх блоків додається заключний блок, який містить довжини ADD та шифрованих даних – він обробляється, використовуючи попередню формулу.

Наступним кроком обчислюється J_0 шляхом додавання до тегу автентифікації 4-байтового рядка 0x00000001, утворюючи блок 128 бітів та його

шифрування AES алгоритмом. Останнім кроком є створення тега автентифікації шляхом операції XOR між J_0 та X.

Поєднання шифрування та створення тегу автентифікації забезпечує ефективне та безпечне шифрування та цілісність інформації.

1.3.1.2 Потокові алгоритми шифрування

Потокове шифрування шифрує дані послідовно, біт за бітом або байт за байтом, шляхом побітної операції XOR між даними та псевдовипадковою шифрувальною послідовністю (keystream), що генерується на основі криптографічного ключа та інших додаткових параметрів. Завдяки цьому потокові алгоритми зазвичай швидші за блокові, адже ефективніше працюють з даними невідомої чи змінної довжини. До потокових алгоритмів шифрування належить RC4, Salsa20, ChaCha20.

RC4 – потоковий шифр, який спочатку ініціалізує масив S довжиною 256 байтів на основі заданого криптографічного ключа, після чого змінює порядок байтів цього масиву та генерує шифрувальну послідовність зі значень S. Кожен байт відкритого тексту шифрується через операцію XOR із відповідним байтом послідовності. Хоча алгоритм є швидким в програмній реалізації та потребує мінімальних обчислювальних витрат, він не є рекомендованим зараз через незбалансованість потоку шифрування та вразливості ключа.

Salsa20 - потоковий шифр, який створює внутрішній стан з 16 слів (кожне 32 байти) – 4 є фіксованими константами, 8 слів – криптографічний 256-бітний ключ, одне слово – 32-бітний лічильник та останні 3 слова – nonce. Після цього виконується 20 раундів AXR операцій: додавання по модулю 2^{32}, лівий циклічний ключ та XOR, де спочатку обробляються колонки матриці стану, а потім її діагоналі. Після завершення раундів отриманий стан модульно додається

до початкового, утворюючи keystream. Для відкритого тексту та потоку обчислюється операція XOR.

Алгоритм ChaCha20 алгоритм шифрування був розроблений американо-німецьким математиком Даніелем Бернштейном у 2008 році. Це потоковий симетричний шифр, який виконує 20 раундів перетворення даних – звідси пішла його назва “ChaCha20”.

ChaCha20 часто поєднується з Poly1305 – алгоритмом для забезпечення перевірки цілісності та автентичності даних. Разом вони утворюють алгоритм ChaCha20-Poly1305, що є повноцінним AEAD-рішенням (Authenticated Encryption with Associated Data), який одночасно забезпечує конфіденційність та цілісність даних без потреби застосування окремих алгоритмів. ChaCha20-Poly1305 був включений до алгоритмів для TLS1.3.

На рисунку 1.2 зображено головні кроки шифруванням даних алгоритмом ChaCha20- Poly1305.

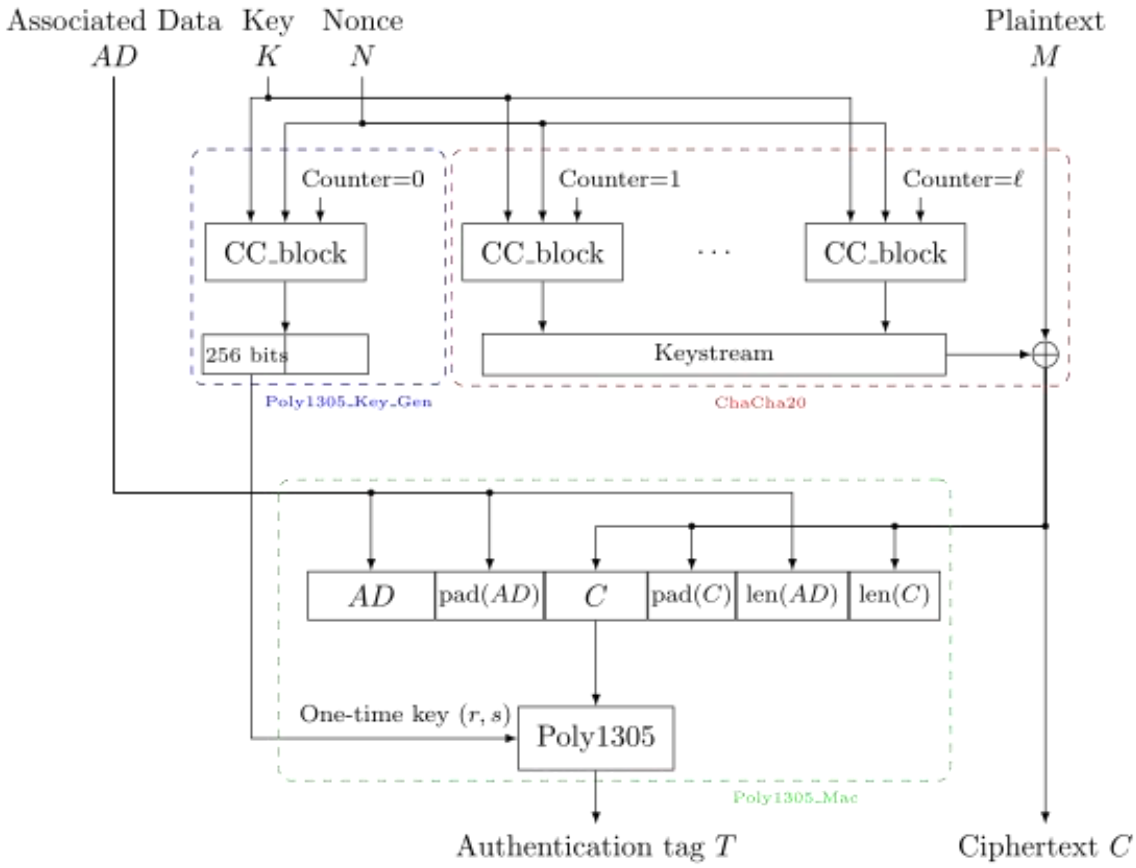


Рис. 1.2. Алгоритм ChaCha20-Poly1305[6]

На вхід алгоритм приймає 256-бітний симетричний ключ, 96-бітний вектор ініціалізації, 32-бітний лічильник, ADD (Additional Associated Data) та відкритий текст, який потрібно зашифрувати.

Використовуючи ключ та вектор ініціалізації та лічильник (який спочатку дорівнює нулю), генерується перші 64 байти потоку шифрування. Перші 32 байти використовуються як одноразовий ключ для алгоритму Poly1305.

Наступний кроком є шифрування відкритого тексту: за допомогою того самого ключа, вектора ініціалізації і лічильника, ChaCha20 створює новий шифрувальну послідовність. Лічильник інкрементується після генерації 64 байтів keystream, що забезпечує унікальність потоку шифрування. Після цього

виконується операція XOR для відкритого тексту та потоку, результатом якого є зашифрований текст.

Після завершення шифрування формується MAC (тег автентифікації) – короткий блок даних, який використовується для перевірки автентичності. Poly1305 генерує тег, маючи ADD, зашифрований текст, довжину ADD та зашифрованого тексту і ключ, згенерований на першому етапі.

Дешифрування ChaCha20-Poly1305 включає два етапи – обов'язкову перевірку тегу автентифікації та розшифрування поданих зашифрованих даних. Для дешифрування на вхід алгоритму подається криптографічний ключ, вектор ініціалізації, лічильник (встановлений на нуль), зашифрований текст, тег автентичності та ADD. Алгоритмом ChaCha20 генеруються 64 байти потоку шифрування, та перші 32 байти беруться як одноразовий ключ для алгоритму Poly1305. Далі алгоритм Poly1305 обчислює очікуваний MAC. Лише після успішної перевірки тегу відбувається розшифрування: ChaCha20 створює новий потік і виконується операція XOR між ним та зашифрованими даними.

1.3.2 Асиметричне шифрування

Асиметричне шифрування – це тип шифрування, в якому використовуються два ключі – публічний для шифрування та приватний для розшифрування даних: будь-хто може зашифрувати дані публічним ключем, але тільки сторона, яка має приватний ключ матиме змогу розшифрувати дані.

Через високу обчислювальну складність та здатність безпечно обмінюватись ключами, асиметричне шифрування часто використовується для цифрових сертифікатів, автентифікації та встановлення захищеного з'єднання. До алгоритмів асиметричного шифрування належать RSA та ECIES.

Алгоритм RSA відбувається в два кроки – генерація ключа та шифрування. Для генерації ключа обираються 2 великі прості числа p та q та обчислюється

n= p× q та $\varphi(n) = (p-1)(q-1)$. Обирається відкрита компонента e так, що gcd (e, $\varphi(n)$) = 1. Знаходиться d, так що d × e ≡ 1 $\left(mod_{\varphi(n)}\right)$. Формується відкритий ключ – пара (n, e) та закритий ключ (n, d). Для повідомлення розміром m обчислюють $c = m^e \bmod n$. Розшифрування обчислюється $m = c^d \bmod n$, де c – шифрований текст.

ECIES (Elliptic Curve Integrated Encryption Scheme) - це гібридна криптографічна схема, яка поєднує переваги асиметричного шифрування на основі еліптичних кривих та симетричного шифрування.

На рисунку 1.3 зображено головні кроки шифруванням даних алгоритмом ECIES.

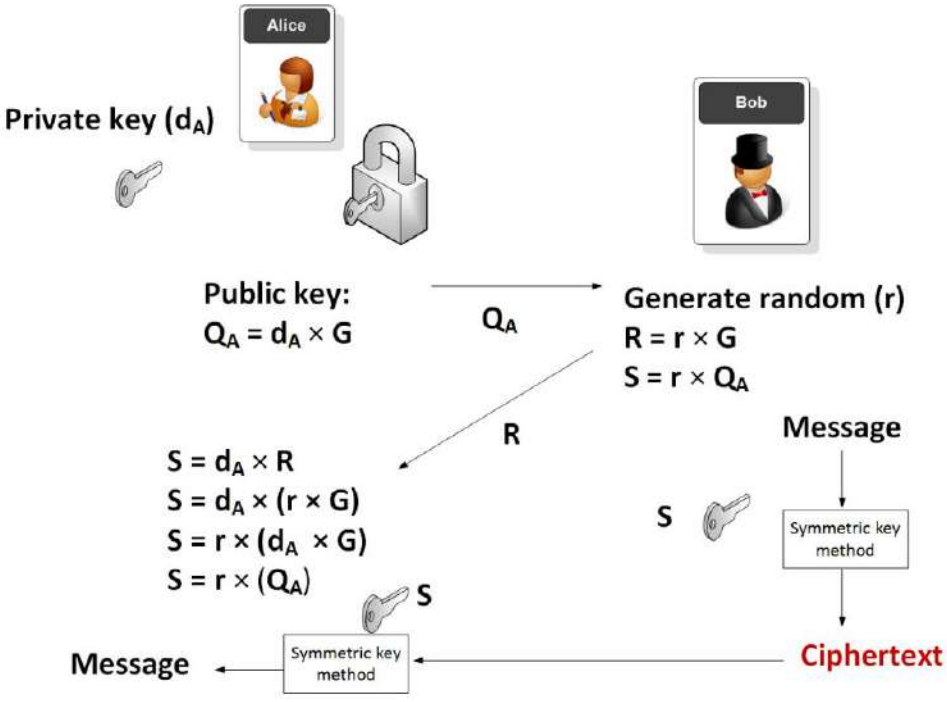


Рис. 1.3. Алгоритм ECIES[7]

Головний крок роботи ECIES полягає в обчисленні спільного секрету між відправником та отримувачем без прямої передачі ключа. Отримувач (на схемі Alice) має власний набір приватного (d_A) та публічного ключа (Q_A), при чому останній обчислюється шляхом множення приватного ключа на базову точку еліптичної прямої (G). Відправник (Bob) генерує тимчасову пару ключів та, маючи публічний ключ отримувача, обчислює спільний секрет S. Завдяки властивостям еліптичних ключів, отримувач, маючи свій приватний ключ та тимчасовий публічний ключ відправника, може отримати той самий спільний секрет.

Спільний секрет використовується як вхід до функції виведення ключа, з якої обидві сторони отримують симетричний ключ, який використовується для шифрування/розшифрування повідомлення.

1.4 Оцінка алгоритмів шифрування

Оцінка криптографічних алгоритмів включає аналіз їх продуктивності та безпеки. З погляду продуктивності зазвичай розглядають два взаємопов'язані критерії: execution time – час виконання шифрування/розшифрування фіксованого обсягу даних та пропускну здатність алгоритму (throughput), який вимірюється за формулою $\frac{D}{T}$, де D – обсяг даних та T – час обробки даних.

Безпекова складова оцінки алгоритму базується на різних підходах, одними з яких є avalanche ефект, візуальна стійкість (visual assessment) та безпекова стійкість (security strength).

Avalanche ефект вимірює чутливість зашифрованого тексту до змін у відкритому тексті. Формально avalanche ефект можна обрахувати як

$$A = \frac{\text{кількість змінених біт у зашифрованому тексті при зміні}}{\text{загальна кількість бітів у зашифрованому тексті}}$$

Strict Avalanche Criterion (SAC) є одним з критеріїв оцінки avalanche ефекту, за яким алгоритм можна вважати безпечним, якщо $A \geq 0.5$, тобто коли при зміні одного біта у відкритому тексті змінюється більше половини бітів у шифрованому тексті.

Visual assessment передбачає перевірку відсутності очевидних структур чи патернів у шифр тексті, які можуть дати зловмиснику можливість отримати будь-яку корисну інформацію за допомогою візуального аналізу зашифрованого тексту.

Безпекова стійкість алгоритму (security strength) оцінює складність найкращої відомої атаки на алгоритм (зазвичай виражається в бітах n) [8, с. 14]. Цей показник може вимірюватись двома критеріями:

1. Attack Steps Metric [9, с. 7] – скільки операцій потрібно, щоб зламати алгоритм. Для симетричних алгоритмів вона практично дорівнює довжині ключа, оскільки найефективніша атака – повний перебір потребує 2^n операцій для n-бітного ключа. В асиметричних алгоритмах, зокрема RSA, величину n обчислюють спираючись на атаку General Number Field Sieve (GNFS), оскільки вона перевищує швидкість повного перебору. Підрахунки показують [8, с. 53], що для забезпечення безпекової стійкості 112 біт необхідна довжина ключа для алгоритму RSA становить 2048 біт. Атака Полларда на алгоритми з еліптичними кривими є найкращою відомою зараз, що зводить оцінку параметра n до обчислення $\sqrt{q}$, де q – порядок кривої. Таким чином довжина ключа x забезпечує приблизний рівень безпеки = x/2.

2. Attack Time Metric [9, с. 7] – скільки часу потрібно для успішного зламу алгоритму. Кількісне оцінка полягає в діленні кількісті необхідних операцій на річну продуктивність системи на якій буде проводитись атака, отже $T = \frac{Attack\ Step\ metric}{\text{Mtops/рік}}$. Згідно з NIST [9, с. 15], пілотний показник становить 3.83 x 10^{16} операцій на рік. Сучасні системи, такі як суперкомп'ютер El Capilot, демонструють продуктивність 1.742 × 10^9 Mtops операцій за секунду, що приблизно дорівнює 5,49 × 10^{16} операцій за рік [10].

1.5 Key management

Правильне шифрування даних є надійним способом запобігти отриманню інформації зловмиснику. Навіть у разі отримання несанкціонованого доступу до даних, без відповідного ключа зловмисник отримає набір нечитабельних зашифрованих символів. Проте ключ – засіб розшифрувати дані, тому критично важливо забезпечити належне управління ключами.

Управління ключами – це процес, який визначає як організація створює, зберігає, захищає та використовує криптографічні ключі, щоб забезпечити їхню цілісність, конфіденційність і контрольований доступ. Належне управління ключами є ключовим елементом будь-якої системи захисту бази даних.

До найкращих практик управління ключами належать:

- Використання централізованого репозиторію для управління ключами. Такий репозиторій дозволяє автоматизувати управління ключами та надати інструменти для захисту ключів.
- Ротація ключів – регулярне оновлення ключів для зменшення ризику їх компрометації.

- Контроль доступу – політики обмеження доступу до централізованого репозиторію, а також визначення чітких ролей доступу до оновлення, видалення чи перегляду ключів, з використанням авторизації та автентифікації.
- Логування та моніторинг – ведення записів будь-яких операцій з ключами, включаючи перегляд, використання, оновлення та видалення для виявлення можливих аномалій та спроб несанкціонованого доступу.

Реалізація таких практик дозволяє знизити ризики пов’язані з витоком чи неправильним використанням криптографічних ключів та забезпечити стійкість організації до інцидентів інформаційної безпеки, пов’язаних з неправомірним доступом до зашифрованої інформації.

1.6 Криптографічні атаки

Криптографічні атаки (криптоаналіз) – атаки, в якій зловмисники намагаються обійти криптографічний захист систем, знайшовши слабкі місця в шифрі, схемі управління ключами чи коді.

Поширеними криптографічними атаками є:

1. Brute-force атака, методом якої є підбирання ключа шифрування всіма можливими способами; отримуючи ключ – є можливість розшифрувати зашифрований текст. Наприклад, якщо ключ, який використовувався в криптографічному алгоритмі має довжину 8 біт, є 256 всіх можливих ключів, які могли бути використаними. Для проведення успішної brute-force атаки, зловмисник також повинен знати яким алгоритмом шифрувались відкриті дані та мати доступ до зашифрованих даних.

2. Chosen-plaintext attacks – криптоаналіз, в якому зловмисник може обрати довільний відкритий текст і надіслати його системі для шифрування. Після отримання відповідного зашифрованого тексту для різних тестових вхідних даних зловмисник намагається проаналізувати закономірності між відкритим та зашифрованим текстом для того, щоб потенційно визначити алгоритм шифрування, вразливості реалізації та відтворити або підробити зашифрований текст для нових повідомлень без знання ключа.
3. Known-plaintext attack – атака, в якій зловмисник вже знає частину відкритих даних, а також відповідний зашифрований текст. Ця інформація може бути використана для визначення криптографічного ключа шляхом аналізу закономірностей та дешифрування інших повідомлень, які були зашифровані тим самим ключем.
4. Key reuse attack (атака через повторне використання ключа) – це криптографічна атака, яка виникає коли один і той самий ключ використовується для кількох повідомлень. Це відкриває можливість зловмиснику виявити закономірності між зашифрованими текстами та частково або повністю відновити незашифровані дані.
5. Downgrade attack – атака на криптографічні протоколи, в якій зловмисник змушує клієнта та сервер перейти з безпечного криптографічного протоколу на старіший, в якому присутні відомі вразливості, які можуть бути використані для розшифрування даних, модифікації повідомлень, тощо.

1.7 Визначення рівня захищеності веб-сервісу

Рівень захищеності веб-сервісу – це міра, яка відображає наскільки ефективно веб-сервіс захищений від відомих вразливостей та атак. Він відіграє критичну роль для будь-якої організації, адже від нього залежить конфіденційність, цілісність та доступність даних, безперервність бізнес процесів та дотримання регуляторних вимог. Недостатня захищеність веб-сервісу може привести до фінансовий втрат, негативного репутаційного впливу та юридичної відповідальності.

Методологія оцінки ризиків (risk assessment) – інструмент, який використовується, щоб кількісно і якісно оцінити безпекову позицію, ідентифікувати слабкі місця системи та визначити засоби захисту.

Основні етапи методології включають:

1. ідентифікацію всіх ключових компонентів системи;
2. ідентифікацію вразливостей – пошук технічних недоліків системи, які можуть використовуватись зловмисниками;
3. оцінка ризику – якісна оцінка рівня ризику в результаті потенційної експлуатації вразливості. Ризик обчислюється на основі оцінки наслідків реалізації загрози зловмисником, ймовірності її виникнення та наявних засобів захисту, які знижують ймовірність загрози.

Для оцінки ризику використовується матриця оцінки ризиків (див. Рис. 1.4.) - підхід, який дозволяє здійснити якісну класифікацію загрози, надаючи їм відповідний рівень (низький, середній, високий чи критичний) та визначити пріоритети для подальшого впровадження заходів безпеки.

Рівень ризику		Рівень впливу				
		Дуже низький	Низький	Середній	Високий	Критичний
Ймовірність	Дуже висока	Низький	Середній	Високий	Високий	Високий
	Висока	Низький	Середній	Середній	Високий	Високий
	Середня	Низький	Низький	Середній	Середній	Високий
	Низька	Низький	Низький	Середній	Середній	Середній
	Дуже низька	Низький	Низький	Низький	Середній	Середній

Рис. 1.4. Матриця оцінки рівня ризику

4. Визначення засобів захисту – формування заходів, які знижують або усувають ідентифіковані ризики.

Висновок до розділу 1

У першому розділі роботи розкрито поняття чутливих даних, виділено групи даних високої чутливості, а також розглянуто основні криптографічні методи їх захисту. Досліджено симетричні (DES, 3DES, Blowfish, AES, ChaCha20, Salsa20, RC4) та асиметричні (RSA та ECIES) алгоритми шифрування. Визначено критерії оцінки алгоритмів шифрування з погляду безпекової стійкості метриками Attack steps metric, Attack time metric, візуальна стійкість та avalanche ефект та продуктивності метриками execution time та пропускна здатність. Розглянуто можливі способи криптоаналізу алгоритмів, зокрема атаки методом перебору криптографічних ключів, атаки chosen-plaintext, known-plaintext, downgrade та атака через повторне використання ключа. Крім того, досліджено підходи до визначення рівня захищеності веб-сервісу шляхом методології оцінки ризиків та описано найкращі практики управління криптографічними ключами, які включають використання централізованого репозиторію для зберігання криптографічних ключів, їх ротацію, контролі доступу та моніторинг всіх операцій з ключами.

2. АНАЛІЗ ПОТОЧНОГО РІВНЯ ЗАХИЩЕНОСТІ ОБРАНОГО ВЕБ-СЕРВІСУ ОРГАНІЗАЦІЇ ТА ПОЧАТКОВИЙ АНАЛІЗ АЛГОРИТМІВ ШИФРУВАННЯ

2.1 Аналіз поточного рівня захищеності обраного веб-сервісу організації

Організація, для якої проводився аналіз, є фінансовою установою, що надає послуги у сфері управління активами та консультативного супроводу інвестиційної діяльності.

В її структурі функціонує автоматизована інформаційна система, яка інтегрує механізми авторизації та автентифікації, зберігання та обробку даних про працівників та управлінням ключовими HR-процесами.

З метою збереження конфіденційності, назва організації та веб-сервісу не вказується.

Аналіз було проведено на основі наявної документації, загальних відомостях про архітектуру та функціонал системи без безпосередньої можливості ручного тестування.

Веб-сервіс розгорнуто у межах внутрішньої мережі організації, проте це не виключає необхідності аналізу його рівня захищеності, адже існує низка ризиків доступу до нього. Неправильні конфігурації мережевих компонентів може надати доступ до сервісу ззовні або дозволити обхід обмежень через тунелювання; успішні компроментація будь-якої іншої системи інфраструктури може надати привілеї для подальшого проникнення до веб-сервісу управління персоналом. Також не варто забувати про внутрішні загрози: зловмисні інсайдери можуть навмисно відкрити вразливості в системі або ж працівники з вищими привілеями можуть ненавмисно змінити важливі конфігурації, що створюють вразливості.

Для аналізу вразливостей сервісу використовувалися рекомендації OWASP, зокрема OWASP Top 10 Web Application Security Risks 2021[11]. Цей перелік є загальновизнаним у галузі і може бути орієнтиром для виявлення та усунення найпоширеніших вразливостей у веб сервісах. Ризики, визначені в списку охоплюють такі критичні напрями як автентифікація, контроль доступу, захист даних, автентифікація, конфігурації безпеки, цілісність та своєчасне оновлення програмного забезпечення, логування та моніторинг.

Результати аналізу веб-системи відповідно до зазначених напрямів:

Автентифікація: у системі використовується централізований механізм входу через організаційного провайдера ідентифікації (SSO). Це значно знижує ризики, по’вязані з обробкою та зберіганням облікових даних локально.

Контролі доступу: у системі реалізовано рольову модель контролю доступу (RBCA), яка дозволяє обмежити доступ до функціоналу та даних залежно від ролі користувача.

Безпека даних: система обробляє та зберігає високочутливі дані, включаючи персональну інформацію, яка може бути використана для ідентифікації особи (PII) та фінансову інформацію (PCI).

Дані передаються в зашифрованому вигляді, використовується TLS 1.0 протокол. Дані в базі даних зберігаються у відкритому вигляді. У дата центрі впроваджено фізичні заходи безпеки, що забезпечують обмеження доступу до інфраструктури.

Актуальність компонентів програмного забезпечення: було виявлено використання застарілих версій серверного програмного забезпечення, зокрема середовища виконання Java. Відомі вразливості цих компонентів залишаються відкритими через відсутність оновлень.

Логування та моніторинг: У системі реалізовано центральне логування подій безпеки з інтеграцією в інструменти моніторингу та аналітики, що дозволяє вчасно виявити та реагувати на потенційні інциденти.

2.2 Аналіз ризиків обраного веб-сервісу

На основі аналізу було ідентифіковані ключові ризики, які становлять загрозу безпеці системи.

Найбільш критичним ризиком є ***відсутність шифрування даних, які зберігаються у базі даних***, що може привести компрометації конфіденційної інформації у разі несанкціонованого доступу до сховища. Ймовірність реалізації даної загрози оцінюється як середня, оскільки в системі впроваджені заходи безпеки, що обмежують несанкціонований доступ до бази даних ззовні; водночас слід враховувати ймовірність внутрішньої загрози, коли користувач з наданими повноваженнями може зловживати доступом у шкідливих цілях. Вплив є критичним – найвищим по матриці оцінки ризиків. Тому рівень ризику – “критичний”.

Заходом зниження ризику є впровадження криптографічного захисту чутливої інформації шляхом використання сучасних алгоритмів шифрування. Крім того, реалізувати механізм зберігання ключів відповідно до принципів безпечного управління ключами.

Ще одним ризиком є ***використання застарілої версії середовища виконання Java***. Ризик експлуатації вразливостей присутній у випадках компрометації внутрішньої мережі або через вразливості інших сервісів. Ймовірність виникнення такої загрози була оцінена як середня, вплив – високий, що відповідає рівню ризику «високий» згідно з матрицею оцінки ризиків.

Заходом зниження ризику є оновлення до актуальної та підтримуваної версії середовища виконання Java.

Хоча дані передаються в зашифрованому вигляді, ***використовується TLS 1.0 протокол, який є застарілим*** згідно стандарту NIST SP 800-52 Rev. 2[12], адже він вразливий до атак таких як POODLE.

Згідно з матрицею оцінки ризиків, ймовірність експлуатації цих вразливостей оцінюється як середня, а потенційний вплив — як високий, що призводить до загального рівня ризику «високий». Рекомендованим засобом зниження цього ризику є оновлення конфігурацій – відмова від TLS 1.0 на користь TLS версій 1.2 чи 1.3.

2.2 Оцінка криптографічних алгоритмів за критерієм безпеки

Було відібрано низку симетричних та асиметричних алгоритмів шифрування та проаналізовано їхню безпекову стійкість за критеріями Attack steps metric та Attack time metric (у роках) (див. Таблицю 2.1), з урахуванням обчислювальною потужності суперкомп’ютера El Capitan (≈ 5,49 * 10^16 операцій/рік).

Таблиця. 2.1. Результати безпекової стійкості вибраних алгоритмів

Алгоритм	Розмір ключа(у бітах)	Безпекова стійкість (у бітах)	Attack Steps Metric/Метрика кількості операцій атаки	Attack Time Metric/Метрика часу атаки (у роках)
BLOWFISH-128	128	32	2^32	2,5 (секунд)
BLOWFISH-256	256	32	2^32	2,5 (секунд)
BLOWFISH-448	448	32	2^32	2,5 (секунд)
DES	56	56	2^56	1,31
3DES-112	112	112	2^112	9.45*10^16
3DES-168	168	112	2^112	9.45*10^16
AES-128	128	128	2^128	6,19*10^21
ECIES-256	128	128	2^128	6,19*10^21
RSA-2048	2048	128	2^128	6,19*10^21
AES-192	192	192	2^192	1.14*10^41
ECIES-384	192	192	2^192	1.14*10^41
CHACHA20	256	256	2^256	2.11*10^60
AES-256	256	256	2^256	2.11*10^60
AES-CBC	256	256	2^256	2.11*10^60
AES-GCM	256	256	2^256	2.11*10^60
ECIES-521	256	256	2^256	2.11*10^60

Для сімейства алгоритмів Blowfish з будь-якою довжиною ключа домінує birthday атака (sweet 32), яка через шифрування блоків по 64 біти для великих даних зменшує безпекову стійкість до ≈ 32 біт, що відповідає 2,5 секунди враховуючи обчислювальну потужність суперкомп'ютера.

DES алгоритм з ключем 56 біт може бути зламаний за 2^56 операцій повним перебором (≈1,31 рік), що в порівнянні з іншими алгоритмами є надзвичайно швидким. Через низький рівень стійкості стандарт NIST виключив DES з списку схвалених алгоритмів для федерального використання, що робить його використання у сучасних системах небезпечним та нерекомендованим [13].

У 3DES-168 з трьома ключами по 56 біт в алгоритмі, атаки класу meet-in-the-middle-attack знижують безпекову стійкість до 112 біт [14].

Всі інші досліджені алгоритми потребують астрономічного часу для їх зламу, що значно перевищує вік Всесвіту (1,38*10^10 років)[15].

2.3 Формування завдання

Основним завданням є аналіз і впровадження криптографічних алгоритмів в процес розробки веб-сервісу управління персоналу, що моделює інформаційну систему, подібну за характеристиками до тієї, яка була попередньо проаналізована. Метою розробки є підвищення рівня безпеки веб-сервісу шляхом усунення ризиків з “критичним” та “високим” рівнем, пов’язаних з відсутністю шифрування чутливих даних та використанням застарілого серверного програмного забезпечення.

Завдання роботи також включає:

- визначення найбільш ефективного методу шифрування даних з урахуванням типу та обсягу інформації;
- впровадження механізму управління криптографічними ключами з дотриманням принципів їх безпечного управління;
- використання актуального середовища виконання з метою усунення вразливостей пов’язаних з використанням застарілих компонентів програмного забезпечення;
- перевірку впроваджених засобів безпеки на відповідність очікуваному рівню захищеності.

Висновок до розділу 2

У другому розділі проаналізовано веб-сервіс управління персоналом у обраній організації, зокрема компоненти автентифікації та авторизації, контролів доступу, безпеки чутливих даних, моніторингу та актуальності компонентів програмного забезпечення. Було виявлено такі вразливості веб-сервісу:

відсутність шифрування даних у базі даних, включаючи високочутливі дані, використання застарілої версії середовища виконання Java та використання застарілого TLS протоколу. Оцінено ризики виявлених вразливостей за допомогою матриці оцінки ризиків та запропоновано такі безпекові заходи для зниження ризику: забезпечення захисту інформації шляхом впровадження ефективного алгоритму шифрування, використання актуального та підтримуваного програмного забезпечення для розробки веб-сервісу та застосування актуальної версії 1.3 протоколу TLS при передачі даних. У другій частині розділу було досліджено обрані алгоритми шифрування з позиції оцінки їх безпекової стійкості, аналізуючи Attack steps metric та Attack time metric для кожного з них.

3. РЕАЛІЗАЦІЯ ПРАКТИЧНОЇ ЧАСТИНИ

3.1 Опис функціональності веб-сервісу

Буде розроблено веб-сервіс інформаційної системи для роботи з інформацією про працівників організації.

До функцій веб-сервісу входитимуть:

- створення профіля працівника організації з персональною, робочою та контрактною інформацією;
- редагування профілю працівника;
- перегляд статусу працівника, зокрема його позиції, дати початку/завершення роботи;
- завантаження та зберігання супровідних документів у форматі PDF (паспортні дані, контракт та страхування);
- управління фінансовими реквізитами працівника, зокрема даними банківського рахунку;
- ведення аналітики, зокрема статистичних даних щодо зарплат працівників, кількості працівників у відділах, кількість нових працівників кожного місяця;
- перегляд даних всіх працівників;
- сортування, фільтрація та пошук працівників.

3.2 Проектування архітектури веб-сервісу управління персоналом

Для розробки веб-сервісу було обрано клієнт-серверну архітектуру, за якою завдання та відповідальність розділені між двома незалежними компонентами –

сервером, що реалізує бізнес логіку, обробляє запити та оперує даними та клієнтом, який забезпечує взаємодію з клієнтом та формування запитів.

3.2.1 Серверна частина

3.2.1.1 Використані інструменти

Серверна частина написана на мові програмування Java (Open JDK 23.0.2), яка завдяки віртуальній машині забезпечує незалежність від платформи, високу продуктивність та високий рівень безпеки через часті оновлення та її широке використання. Мова має статичну типізацію, що зменшує кількість можливих вразливостей через більшу можливість контролю, а також широкий набір вбудованих бібліотек.

У якості веб-фреймворку обрано Spring Boot, оскільки він забезпечує комплексну автоконфігурацію та модульність. Завдяки механізму стартерів, Spring Boot автоматично підбирає та конфігурує залежності для типових сценаріїв (наприклад, доступ до бази даних та безпеки), спрощуючи початкову роботу над проєктом. Веб-фреймворк дозволяє легко інтегрувати інші залежності та легко поєднується р іншими модулями екосистеми Spring.

Для організації рівня доступу використовується Spring Security, а для взаємодії з базою даних Spring data JPA , яка забезпечує абстракцію для CRUD-операцій до реляційної бази даних через інтерфейси. Криптографічні ключі централізовано зберігаються у HashiCorp Vault за допомогою Spring Vault, а інші криптографічні операції реалізовано з використанням Java Crypto API та бібліотеки Bouncy Castle для підтримки еліптичних кривих.

Для інтеграції з хмарними сервісами авторизації використано Google-бібліотеку Firebase Admin SDK, яка відповідає за обробку OAuth2.0-запитів.

3.2.1.2 Проектування архітектури серверної частини

Серверна частина системи спроєктована за багатошаровою архітектурою, тобто вона чітко розподілена на кілька взаємопов'язаних шарів, кожен з яких має конкретний функціональні обов'язки. Як зображено на рис. 3.1, у межах архітектури виділено такі шари:

1. Шар представлення – головною задачею якого є прийняття та обробку HTTP-запитів з клієнтської частини та формування відповіді у вигляді JSON. Реалізується через HTTP-контролерів з анотаціями @RestController та методи з анотаціями @GetMapping, @PostMapping тощо які прив'язуються до конкретних URL.

У реалізованому веб-сервісі представлено AuthContorller для автентифікації і авторизації та UserController для операцій з користувачами.

2. Шар бізнес-логіки – це сервісні класи з анотацією @Service, що інкапсулють бізнес логіку. Функції сервісу відповідають за перевірку бізнес-умов, агрегацію викликів репозиторіїв та реалізацію сценаріїв використання.

У веб-сервісі реалізовано AuthService для обробки сценаріїв автентифікації та UserController для управління операціями з даних працівників.

3. Шар доступу до даних реалізовано на базі String Data JPA репозиторіїв, які інкапсулюють взаємодію з реляційним сховищем та відповідають за CRUD-операції і запити до бази даних.

Також цей шар містить класи предметної області: сутності @Entity та DTO (Data Transfer Objects), які використовуються для передачі даних між шарами.

Реалізовано сутності User, Role, FinancialDetails, UserDocuments, а також UserDTO для зручного та безпечного обміну даними між шарами без прямого відображення внутрішньої структури сутностей.

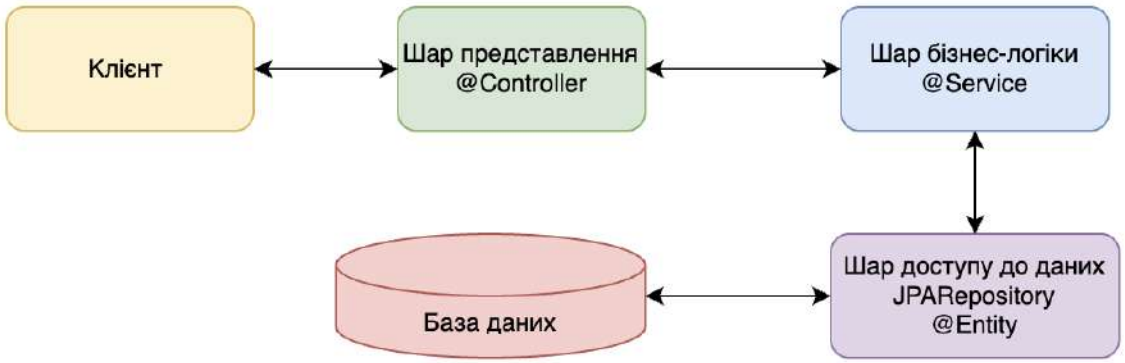


Рис. 3.1. Багатошарова архітектура сервера

3.2.2 Клієнтська частина

3.2.2.1 Використані інструменти

Для побудови клієнтської частини обрано фреймворк Angular 19, завдяки його компонентно-орієнтованій архітектурі та вбудованому RxJS, що забезпечує реактивну обробку даних і дозволяє розділяти веб-сервіс на незалежні модулі. Для реалізації аутентифікації інтегровано Firebase JavaScript SDK. Інтерфейс стилізовано за допомогою Bootstrap та Tailwind CSS. Відображення графіків та діаграм здійснюється за допомогою пакетів g2-charts і Chart.js. Усі залежності керуються через npm, що спрошує додавання та оновлення пакетів.

3.2.2.2 Проектування архітектури клієнтської частини

Клієнтська чистина сервісу побудована за принципом шарової архітектури та модульності містить, що дозволяє чітко розділити обов’язки та відповідальності.

Першим головним шаром є «Injector», який містить сервіси та охоронці маршрутів, що використовуються у іншим компонентах веб-сервісу. Реалізовано два сервіси: AuthService, який відповідає за автентифікацію та авторизацію, та UserService, який обробляє запити з сервера для CRUD-операцій. Охоронці маршрутів (AdminGuard, HrManagerGuard та EmployeeGuard) блокують доступ до компонентів залежно від роді користувача.

Другим шаром є “Feature Modules”, що ізолюють функціональні області, кожна з яких забезпечена захистом та роутингом. До Feature Modules належить:

- Auth Module – реалізує сторінку входу, доступну без автентифікації;
- Admin Module - містить компоненти, які доступні адміністратору і захищений AdminGuard;
- HrManager Module - містить компоненти, які доступні менеджеру з управління персоналу і захищений HrManagerGuard; та
- Employee Module - містить компоненти, які доступні працівнику організації і захищений EmployeeGuard.

Також, для скорочення дублювання коду та уніфікації інтерфейсних елементів створено Shared Module, який містить повторювані елементи – графіки використовувані для статистики (BarChartComponent, PieChartComponent) та елементи управління інтерфейсом (ButtonComponent, TopBarComponent).

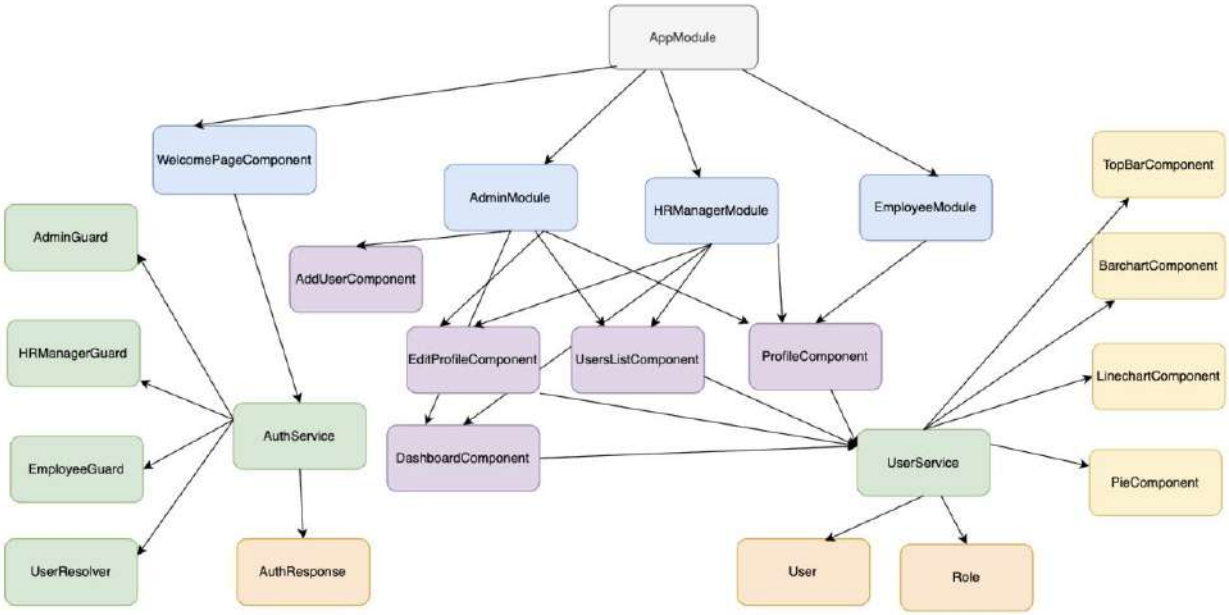


Рис. 3.2. Архітектура клієнтської частини

3.2.3 База даних

Для зберігання даних було обрано реляційну СУБД MySQL, а візуалізацію структури таблиць та зав'язків здійснено в середовищі MySQL Workbench.

У центрі моделі знаходиться таблиця user, що зберігає інформацію про працівників: ім'я, номер телефону, робочу та персональну електронну пошту, посаду, відділ, дати початку і кінця роботи, роль користувача в системі тощо.

Для відображення ієрархії підпорядкування у межах компанії реалізовано рекурсивне поле manager_id, яке посилається на emp_id іншого запису в цій таблиці.

Таблиця role містить перелік ролей і має зв'язок з таблицею user через зовнішній ключ role_id.

Для зберігання фінансових даних створено таблицю financial_details, реалізовано зв'язок один-до-одного з таблицею user через поле user_id із

каскадним видаленням. У таблиці міститься інформація про банківські реквізити та податкова інформація.

Для зберігання документів створено таблицю user_documents, яка містить паспорт, контракт та страховий поліс. Зв’язок один-до-одного з таблицею user реалізовано через user_id із каскадним видаленням.

Розділення даних у різні таблиці забезпечує нормалізацію бази даних, а також дає змогу виконувати цільові запити до конкретних наборів полів без завантаження всього запису, що підвищує продуктивність сервісу.

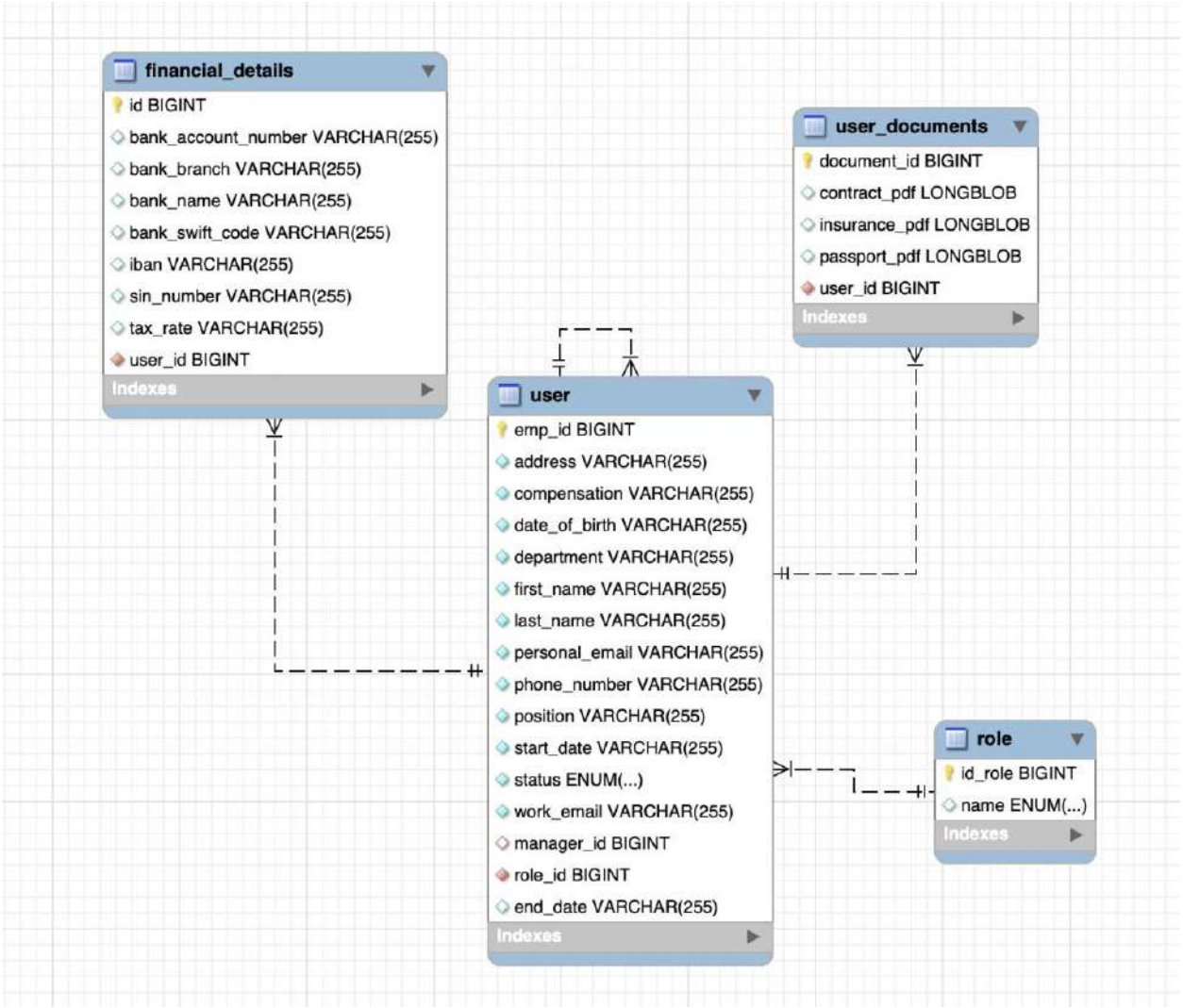


Рис. 3.3. Реляційна модель розробленого веб-сервісу управління персоналу

3.3 Тестування криптографічних алгоритмів

Для вибору найкращого криптографічного алгоритму захисту чутливих даних було проведено тестування низки симетричних та асиметричних алгоритмів з метою оцінки швидкості шифрування/розшифрування, як один з критеріїв оцінки алгоритмів шифрування. Зокрема, було протестовано такі симетричні алгоритми: AES (128, 192, 256 біт), а також AES-GCM та AES-CBC, ChaCha20, (128, 256 та 448 біт), DES, 3DES (112 та 168 біт) . Щодо асиметричних алгоритмів, тестування охоплювало ECIES з кривими 256, 384 та 521 біт.

Для реалізації та порівняння алгоритмів було використано патерн Strategy, що належить до поведінкових патернів проєктування та передбачає інкапсуляцію родини алгоритмів за єдиним інтерфейсом.

Учасниками патерну є:

1. інтерфейс стратегії, який оголошує методи, які мають бути імплементовані у конкретних алгоритмах;
2. алгоритми, які реалізовують інтерфейс;
3. контекст – клас, який має залежність від інтерфейсу стратегії і тим самим - доступ до різних алгоритмів потрібної операції.

Така архітектура забезпечує можливість динамічної заміни алгоритмів у контексті та значно спрощує додавання нових криптографічних алгоритмів без необхідності модифікувати клієнтський код. На рис. 3.4 зображено реалізацію патерну Strategy: Encryption Strategy – інтерфейс стратегії, який містить два методи – encrypt та decrypt, SymmetricEncryption та AsymmetricEncryption, а також наслідувані від них класи – конкретні реалізації операцій шифрування/розшифрування та Encryption Service, який виступає контекстом, який динамічно застосовує вибрану стратегію.

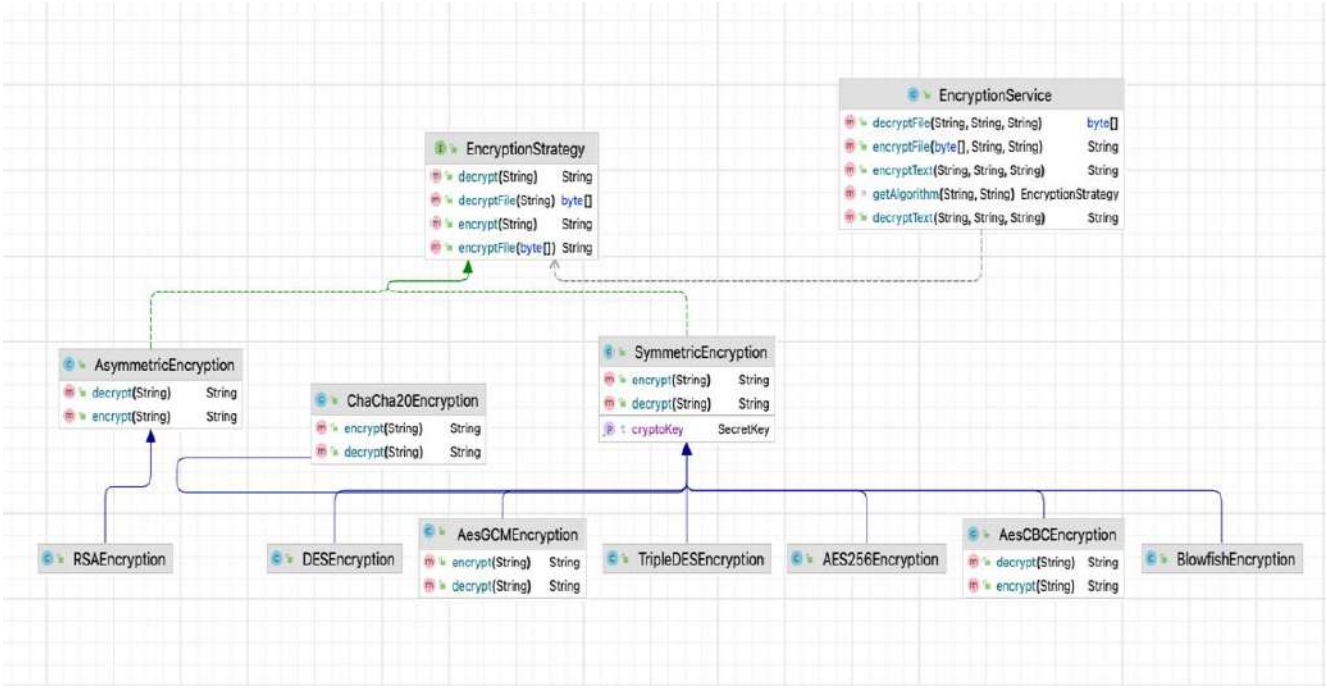


Рис. 3.4. Реалізація патерну "Strategy" для тестування продуктивності алгоритмів

Компонент EncryptionService реалізує фабричний підхід до динамічного вибору криптографічного методу на основі переданого рядкового ідентифікатора.

Для оцінки продуктивності алгоритмів було здійснено серію експериментальних вимірювань часу шифрування та розшифрування даних, що моделюють реальний обсяг інформації користувацьких профілів.

В якості тестових сценаріїв було обрано дев’ять масштабів навантаження: 10, 50, 100, 200, 500, 1000, 2500, 4300, 5000 профілів, де кожен профіль може займати різну кількість байтів. Результати тестування наведено в Додатку 1. Для кожного сценарію збирались результати про величину даних у байтах, час, необхідний для шифрування та розшифрування у мілісекундах. Як головна метрика порівняння використовувалось середнє арифметичне часу шифрування.

На рис. 3.5-3.8 та в таблицях 3.1, 3.2, 3.3 наведено результати вимірювання середнього часу шифрування при трьох обсягах даних, що відповідають 50, 2500 та 5000 профілям користувачів.

Таблиця 3.1 наводить результати вимірювання середнього часу шифрування десяти найшвидших алгоритмів для обсягу даних 30942 байтів (≈ 31 КБ).

Таблиця 3.1. Результати 10-и найшвидших за середнім часом алгоритмів для обсягу даних відповідному профілям 50 працівників

Алоритм	Обсяг даних (байти)	Час шифрування (мс)	Час розшифрування (мс)	Середній час шифрування (мс)
DES	30942	26,7	25,398	26,049
3DES-168	30942	29,113	28,222	28,6675
3DES-112	30942	29,887	29,013	29,45
BLOWFISH-448	30942	32,107	29,092	30,5995
BLOWFISH-256	30942	40,368	37,125	38,7465
BLOWFISH-128	30942	52,948	52,527	52,7375
AES-GCM	30942	63,517	78,334	70,9255
AES-CBC	30942	82,128	76,152	79,14
ChaCha20	30942	93,105	65,45	79,2775
AES-192	30942	86,449	80,248	83,3485

Найкращий результат продемонстрував DES (≈ 26 мс), слідом йдуть 3DES (112 та 168 бітів) (≈ 28 − 29 мс) та три варіанти Blowfish (з ключами 128, 256,448 біт). AES-GCM та AES-CBC, ChaCha20 демонструють час виконання ≈ 70 − 79 мс. Алгоритми ECIES (≈242 мс) не включено до таблиці через значну повільність. Натомість всі протестовані алгоритми зображені на гістограмі (рис. 3.5), що дозволяє наочно їх порівняти.

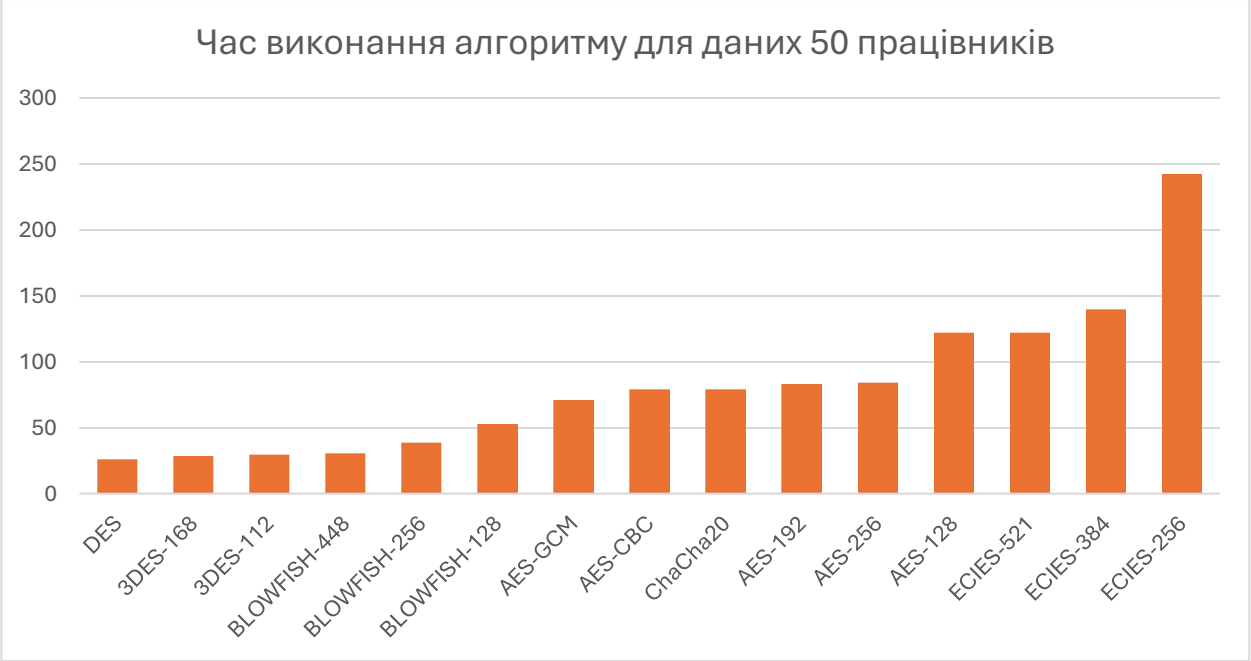


Рис. 3.5. Час виконання алгоритмів для обсягу даних відповідному профілям 50 працівників

В таблиці 3.2 представлено аналогічні вимірювання для обсягу даних 2887104 байт (≈ 2,89 МБ), що відповідає даним 2500 працівників. Лідером при заданому обсягу даних став AES-GCM (≈1182 мс), а наступні найкращі результати були продемонстровані алгоритмами AES-CBC (≈1199 мс) та ChaCha20 (≈ 1228 мс). При збільшені даних, стає візуально видно різницю між результатами симетричних та асиметричних алгоритмів (рис. 3.6). Для алгоритмів ECIES в цьому сценарії час виконання склав ≈ 7500 – 7950 мс, що в приблизно 6 раз повільніше ніж AES-GCM.

Таблиця. 3.2. Результати 10-и найшвидших за середнім часом алгоритмів для обсягу даних відповідному профілям 2500 працівників

Алоритм	Обсяг даних (байти)	Час шифрування (мс)	Час розшифрування (мс)	Середній час шифрування (мс)
AES-GCM	2887104	1195,758	1168,452	1182,105
AES-CBC	2887104	1216,636	1183,113	1199,8745
ChaCha20	2887104	1265,651	1190,352	1228,0015
AES-256	2887104	1286,81	1276,304	1281,557
BLOWFISH-256	2887104	1399,292	1396,832	1398,062
BLOWFISH-448	2887104	1429,407	1411,167	1420,287
AES-192	2887104	1505,382	1506,173	1505,7775
3DES-168	2887104	2613,055	1502,631	2057,843
DES	2887104	2772,218	1355,916	2064,067
3DES-112	2887104	2649,564	1548,738	2099,151

Рис. 3.6. Час виконання алгоритмів для обсягу даних відповідному профілям 2500 працівників

В таблиці 3.3 представлено аналогічні вимірювання для обсягу даних 4474669 байт (≈ 4,47 МБ), що відповідає 5000 профілям працівників. Найкращі

результати продемонстрував AES-256 (≈1687 мс), DES (≈1766 мс) та AES-GCM (≈1798 мс). Найповільніше спрацював алгоритм ECIES-256, час якого склав ≈ 124111 мс.

Таблиця 3.3. Результати 10-и найшвидших за середнім часом алгоритмів для обсягу даних відповідному профілям 5000 працівників

Алоритм	Обсяг даних (байти)	Час шифрування (мс)	Час розшифрування (мс)	Середній час шифрування (мс)
AES-256	4474669	1694,353	1679,8	1687,0765
DES	4474669	1767,469	1765,032	1766,2505
AES-GCM	4474669	1810,892	1786,908	1798,9
AES-CBC	4474669	1916,4	1858	1887,2
AES-192	4474669	1897,954	1884,292	1891,123
BLOWFISH-128	4474669	2145,104	2150,197	2147,6505
3DES-112	4474669	2201,053	2165,074	2183,0635
3DES-168	4474669	2217,135	2221,769	2219,452
BLOWFISH-256	4474669	2204,697	2243,652	2224,1745
BLOWFISH-448	4474669	2450,226	2495,091	2472,6585

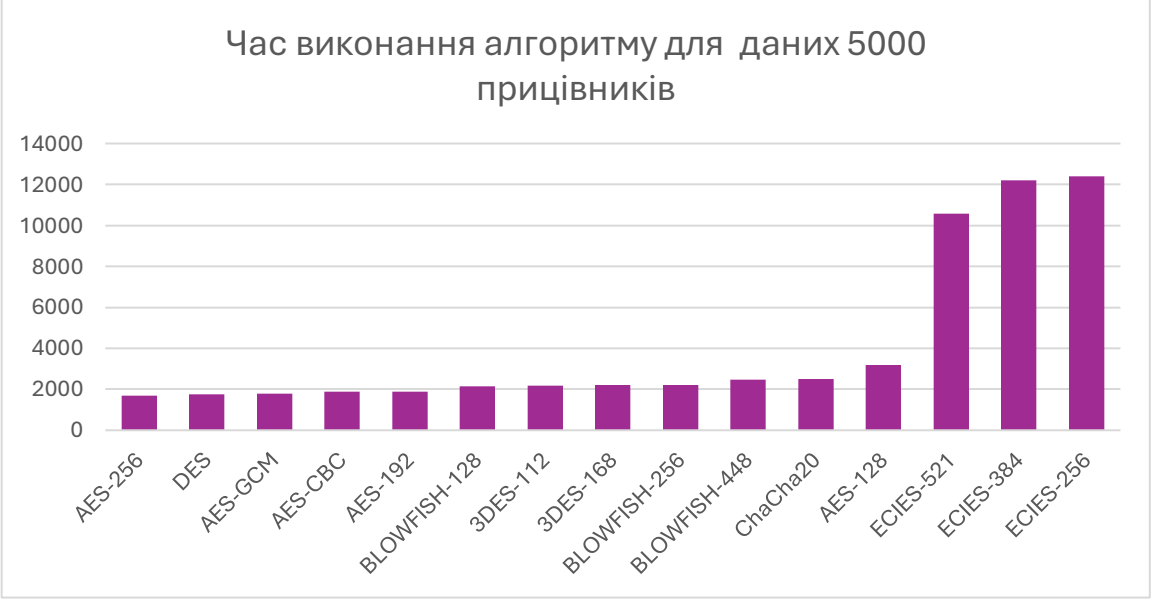


Рис. 3.7. Час виконання алгоритмів для обсягу даних відповідному профілям 5000 працівників

Для забезпечення узагальненого порівняння продуктивності було розраховано середній час виконання кожного алгоритму на основі всіх дев’яти сценаріїв із різними обсягами даних. Це було зроблено для усунення флуктуацій окремих обсягів даних, які не стосуються продуктивності алгоритму; та з метою вибору рішення, яке може підійти для різного навантаження системи з урахуванням як малих, так і великих обсягів даних.

Найкращий результат продемонстрував алгоритм AES-GCM з середнім показником ≈ 720 мс, наступні – алгоритми AES-256 та AES-CBC ($\approx 725 - 740$ мс . Хороші результати також були отримані з використанням алгоритмів DES та ChaCha20 ($\approx 780 - 806$ мс).

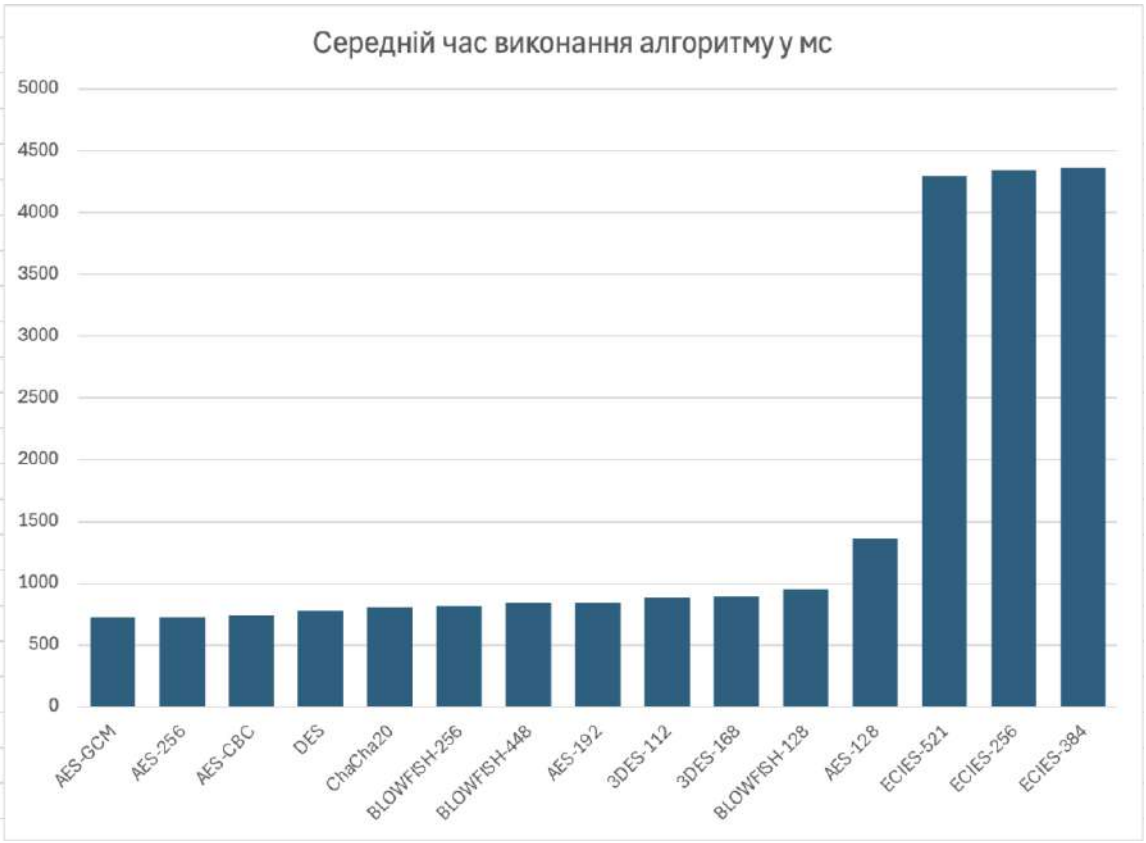


Рис. 3.8. Середній час виконання алгоритмів для всіх протестованих обсягів даних

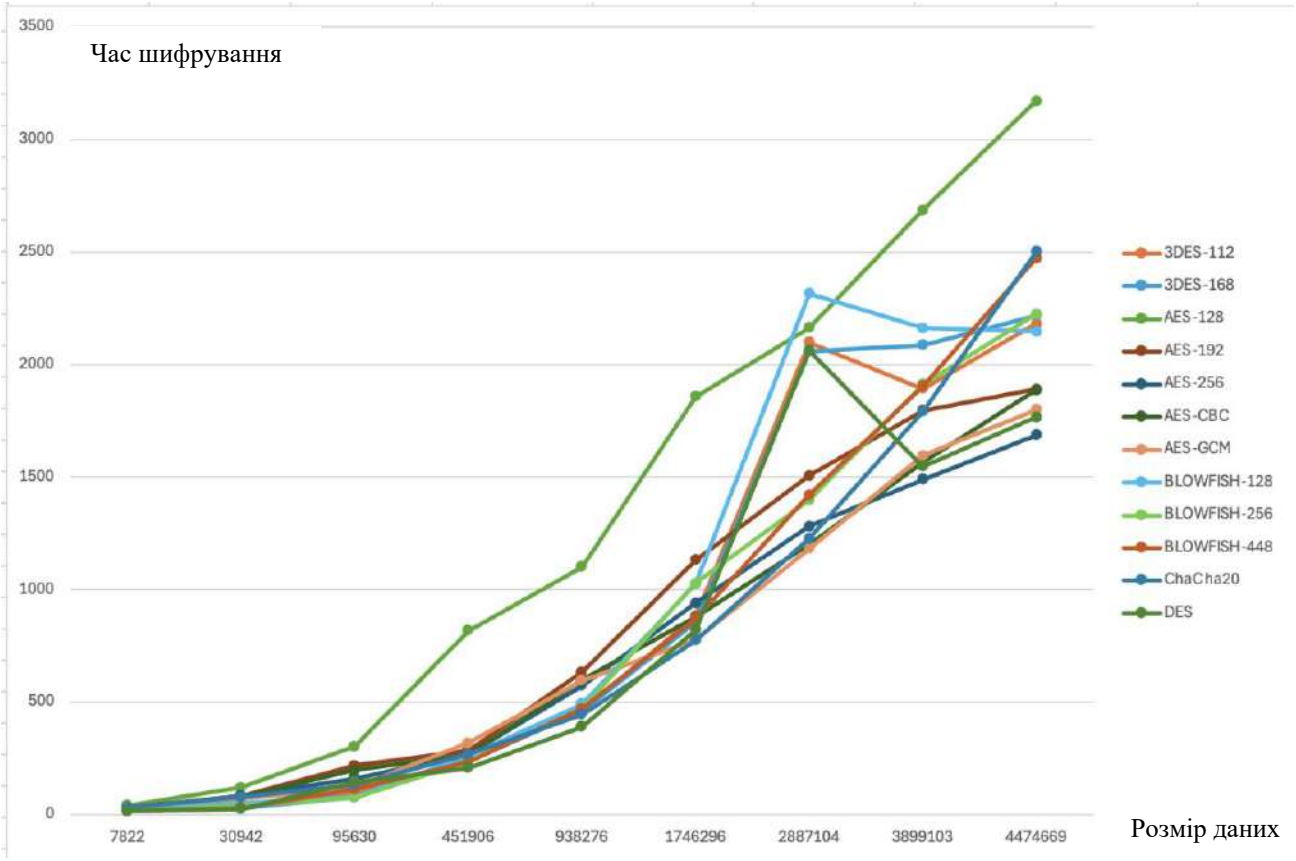


Рис. 3.9. Залежність часу шифрування алгоритму від обсягу даних (вісь x – розмір даних, y – час шифрування)

Для симетричних алгоритмів було побудовано графік їхньої пропускної здатності (throughput) – скільки байтів даних обробляє алгоритм за 1 мс.

Кут нахилу відображає, як швидко падає пропускна здатність зі збільшенням кількості даних: чим крутіша лінія алгоритму, тим сильніше знижується ефективність алгоритму на більших обсягах даних. Алгоритм DES має найкрутіший підйом, що означає, що його пропускна здатність найбільше зменшується з збільшенням кількості даних. Алгоритми AES-GCM та ChaCha20 зростають повільніше – вони найкраще масштабуються при збільшенні обсягів даних.

Зважена середня пропускна здатність була розрахована діленням загального обсягу зашифрованих даних на сумарний час, необхідний для їх обробки.

Таблиця 3.4. Результати обрахунку зваженої пропускної здатності алгоритмів

Алгоритм	Пропускна здатність (байти/мс)
AES-128	1185
BLOWFISH-128	1698
3DES-168	1807
3DES-112	1838
AES-192	1917
BLOWFISH-448	1928
BLOWFISH-256	1963
ChaCha20	2002
DES	2079
AES-CBC	2168
AES-256	2232
AES-GCM	2239

Найкращу пропускну здатність показав алгоритм AES-GCM з результатом $\approx 2079\ \frac{bytes}{ms}$.

Отримані результати тестування підтвердили, що симетричні алгоритми значно ефективніші за асиметричні чи гібридні при обробці великих обсягів даних. Середні показники трьох найкращих симетричних алгоритмів виявились приблизно у шість разів меншими, ніж у алгоритмів ECIES.

Крім того, помітний вплив апаратного прискорення AES-NI – набору інструкцій в сучасних CPU, призначені для прискорення операцій блочного шифрування AES. Таким чином алгоритми AES демонструють кращі результати порівняно з програмними алгоритмами (3DES, Blowfish). Це також пояснює перевагу AES-GCM і AES-CBC над ChaCha20, який не використовує AES-NI.

3.4 Реалізація обраного криптографічного алгоритму

Для шифрування даних в розробленій системі було обрано AES-GCM з огляду на наступні характеристики:

1. Безпекова стійкість алгоритму складає 256 біт, що відповідає 2^{256} операціям повного перебору та робить злам алгоритму неможливим в теперішній час.
2. Продуктивність алгоритму має одні з найкращих показників серед усіх тестованих алгоритмів.
3. Алгоритм забезпечує як конфіденційність, так і цілісність через використання тегу автентифікації, що усуває потребу в додаткових обчисленнях.
4. Завдяки апаратному прискоренню AES-NI, використання AES алгоритму є більш ефективним.

AES-GCM реалізований у класі AesGCMEncryption, що наслідує клас SymmetricEncryption. Для шифрування використовується трансформація “AES/GCM/NoPadding”. Довжина вектору ініціалізації становить 12 байт, а довжина тегу автентифікації становить 128 біт. При кожному виклику методу encrypt() генерується випадковий вектор ініціалізації за допомогою SecureRandom(). Шифр ініціалізується викликом cipher.init() з тегом, вектором та ключем, отриманого з Vault. Операція cipher.doFinal() виконує шифрування та одночасно генерує тег автентифікації, який додається в кінець масиву зашифрованих даних. Останньою операцією є додавання вектору на початок шифрованих даних для зручності передачі та подальшого розшифрування.

```
@Override
public String encrypt(String data) {
    try {
        Cipher cipher = Cipher.getInstance(TRANSFORMATION);
        byte[] ivector = new byte[IV_SIZE];
        new SecureRandom().nextBytes(ivector);

        GCMParameterSpec spec = new GCMParameterSpec(TAG_LENGTH, ivector);
        cipher.init(Cipher.ENCRYPT_MODE, getCryptoKey(), spec);

        byte[] encrypted = cipher.doFinal(data.getBytes());
        byte[] merged = new byte[ivector.length + encrypted.length];
        System.arraycopy(ivector, srcPos: 0, merged, destPos: 0, ivector.length);
        System.arraycopy(encrypted, srcPos: 0, merged, ivector.length, encrypted.length);

        return Base64.getEncoder().encodeToString(merged);
    } catch (Exception e) {
        throw new RuntimeException("AES-GCM encryption error", e);
    }
}
```

Рис. 3.10. Реалізація методу шифрування з використанням алгоритму AES-GCM

При розшифруванні ключ ініціалізації та зашифровані дані розділяються, після чого створюється новий екземпляр шифру та ініціалізується у режимі DECRYPT_MODE з криптографічним ключем, тегом автентифікації та вектором. Виклик cipher.doFinal() спочатку перевіряє тег автентифікації, і якщо він успішно валідується - розшифровує дані.

```
@Override
public String decrypt(String encryptedData) {
    try {

        byte[] decoded = Base64.getDecoder().decode(encryptedData);
        byte[] iv = new byte[IV_SIZE];
        byte[] encrypted = new byte[decoded.length - IV_SIZE];

        System.arraycopy(decoded, srcPos: 0, iv, destPos: 0, IV_SIZE);
        System.arraycopy(decoded, IV_SIZE, encrypted, destPos: 0, encrypted.length);

        Cipher cipher = Cipher.getInstance(TRANSFORMATION);
        GCMParameterSpec spec = new GCMParameterSpec(TAG_LENGTH, iv);
        cipher.init(Cipher.DECRYPT_MODE, getCryptoKey(), spec);

        byte[] decrypted = cipher.doFinal(encrypted);
        return new String(decrypted);
    } catch (Exception e) {
        throw new RuntimeException("AES-GCM decryption error", e);
    }
}
```

Рис. 3.11. Реалізація методу розшифрування з використанням алгоритму AES-GCM

В архітектурі кожен запис користувача шифрується з використанням унікального криптографічного ключа, прив’язаного до його ідентифікатора (employee ID). Ці ключі зберігаються у HarshiCorp Vault у відповідних директоріях за ID, що додає зручність та гарантує ізоляцію даних.

Для коректного вибору та використання ключа під час операцій необхідно знати поточний ID користувача. Цю задачу виконує реалізований клас EncryptionContext, який використовує ThreadLocal<Long> для зберігання

ідентифікатора в межах потоку для обробки запиту. Перед початком операції компонент встановлює значення, а після її завершення – видаляє.

```
public class EncryptionContext {
    3 usages
    private static  final ThreadLocal<Long> localThreadEmplId = new ThreadLocal<>();
    6 usages   innaaa1
    public static void setEmplId(Long emplId) {
        localThreadEmplId.set(emplId);
    }
    8 usages   innaaa1
    public static Long getEmplId(){
      return  localThreadEmplId.get();
    }
    6 usages   innaaa1
    public static void remove(){
        localThreadEmplId.remove();
    }
}
```

Рис. 3.12. Реалізація класу EncryptionContext для зберігання id працівника

Шифрування даних у базі даних було реалізовано через механізм JPA AttributeConverter. Кожне поле, яке має бути зашифроване позначається анотацією @Convert(converter = AESGCMStringEncryptor.class)(або відповідним конвертером для дат та чисел).

```
@Column(nullable = false)
@Convert(converter = AESGCMStringEncryptor.class)
private String phoneNumber;

@Column(nullable = false)
@Convert(converter = AESGCMStringEncryptor.class)
private String address;

@Column(nullable = false)
@Convert(converter = AESGCMDateEncryptor.class)
private LocalDate dateOfBirth;
```

Рис. 3.13. Використання AttributeConverter для автоматизованого шифрування/розшифрування даних

Кожен конвертер реалізує інтерфейс AttributeConverter<X, String> та перевизначає методи convertToDatabaseColumn та convertToEntityAttribute, для шифрування та розшифрування даних.

```
@Override
public String convertToDatabaseColumn(String s) {
    String id = String.valueOf(EncryptionContext.getEmplId());
    try {
        return encryptionService.encryptText(s, encryption, id);
    } catch (NoSuchAlgorithmException | InvalidKeySpecException e) {
        throw new RuntimeException(e);
    }
}
no usages   innaaa1
@Override
public String convertToEntityAttribute(String s) {
    String id = String.valueOf(EncryptionContext.getEmplId());
    try {
        return encryptionService.decryptText(s, encryption, id);
    } catch (NoSuchAlgorithmException | InvalidKeySpecException e) {
        throw new RuntimeException(e);
    }
}
```

Рис. 3.14. Реалізація методів шифрування/розшифрування у класі нащадку AttributeConverter

3.5 Механізми автентифікації та авторизації

У веб-сервісі реалізовано OAuth 2.0, інтегрований за допомогою сервісу Firebase Authentication.

OAuth 2.0 – протокол авторизації , який дозволяє веб-сервісу отримати доступ до обмежених ресурсів користувача (наприклад, імейл) через зовнішній сервіс (Google, Facebook). Під час входу у веб-сервіс, користувач перенаправляється на сторінку авторизації Google, де він погоджується надати обмежений доступ до певних даних облікового запису, після чого авторизаційний сервер Google надає веб-сервісу токен доступу, який підтверджує згоду користувача на доступ до даних. Цей токен не містить пароля і не передає пароль веб-сервісу; він також є тимчасовим.

На клієнтській стороні (Angular) використовується Firebase JS SDK, яка ініціалізована в app.config.ts та містить дані про середовище та API ключ, необхідний для доступу до проєкту в Firebase.

```
export const appConfig: ApplicationConfig = {
  providers: [
    provideFirebaseApp(() =>
      initializeApp(environment.firebase)
    ),

    provideAuth(() => getAuth()),
    provideFirestore(() => getFirestore()),
```

Рис. 3.15. Конфігурація Firebase JS SDK у Angular-клієнті

На серверній стороні реалізований метод initialize(), який зчитує облікові дані сервісного акаунту з JSON файлу та ініціалізується FirebaseApp з опціями FirebaseOptions, до яких входить GoogleCredentials.

При автентифікації, коли користовувач натискає кнопку “Google Sign-in” викликається метод signInWithPopup з провайдером GoogleAuthProvider; успішна авторизація повертає об’єкт user, який реактивно передається методу loginWithFirebase в сервісі authService, де токен юзера передається у POST запит для подальшої верифікації токену на сервері.

```
googleAuthProvider = new GoogleAuthProvider();
auth = inject(Auth);

onSignInWithGoogle() {
  signInWithPopup(this.auth, this.googleAuthProvider)
  .then(async (response) => {
    const user = response.user
    this.authService.loginWithFirebase(user).subscribe({
      next: () => this.redirectToDashboard(),
      error: () =>  this.errorMessage = 'Authorization failed'
    })

  })
    .catch( () => {
      this.errorMessage = 'Something went wrong';
    });
}
```

Рис. 3.16. Реалізація автентифікації через Google

Для верифікації реалізовано фільтр автентифікації FirebaseAuthenticationFilter, який за допомогою методу FirebaseAuth.getInstance().verifyIdToken(token) перевіряє чи отриманий токен дійсний, чи він був виданий Firebase та підтверджує підпис токена.

З токена отримується імейл і за ним здійснюється пошук користувача в базі даних. Якщо користувач знайдений, створюється об'єкт UsernamePasswordAuthenticationToken з роллю користувача та передається в контекст безпеки SecurityContextHolder, що дозволяє Spring Security далі обробляти запит як автентифікований.

```
UsernamePasswordAuthenticationToken authentication = new UsernamePasswordAuthenticationToken(
        email,
        credentials: null,
        List.of(new SimpleGrantedAuthority( role: "ROLE_" + user.getRole().getName().name()))
);
SecurityContextHolder.getContext().setAuthentication(authentication);
```

Рис. 3.17. Ініцілізація об'єкту UsernamePathwordAuthentificationToken із призначенням ролі для його встановлення у SecurityContextHolder

У веб-сервісі реалізований role-based access, який обмежує чи надає доступ до певного функціоналу залежно від ролі. В предметній області визначено три ролі – адміністратор системи, менеджер з управління персоналом та працівник організації.

На серверній стороні реалізоване обмеження доступу до REST- ендпоінтів в класі конфігурації безпеки за допомогою методу authrizeHttpRequests(). Також перед будь-яким запитом виконується фільтр FirebaseAuthenticationFilter, який перевіряє валідність токена.

```
.authorizeHttpRequests(auth -> auth
        .requestMatchers("/api/auth/login").permitAll()
        .requestMatchers(
                "/userslist",
                "/role/**", "/user/edit"
        ).hasAnyRole( ...roles: "ADMIN", "HR_MANAGER")
        .requestMatchers(
                "/user/add").hasRole("ADMIN")
        .anyRequest().authenticated()
)
```

Рис. 3.18. Правила доступу в Spring Security для різних ролей

Додатково, для захисту мартшрутизованих сторінок на клієнтській стороні реалізовано Angular Guards (adminGuard, hrmanagerGuard та employeeGuard), які підключаються до маршрутизатора та перевіряють роль поточного користувача, збереженого в AuthService. Якщо роль не дозволяє доступ до відповідної сторінки – користувач перенаправляється на головну сторінку.

3.6 Інтегрування механізму управління ключами

Vault – централізована система для управління критичною датою та секретами, зокрема криптографічними ключами. Вона забезпечує контрольоване сховище для чутливих даних, шифрує секретну інформацію, дає можливості керуваннями контролями доступу, ротації та видалення ключів.

Для виконання цієї роботи, було обрано HashiCorp Vault як сховище для зберігання симетричних та асиметричних ключів з можливістю управління ключами для кожного користувача.

Після ініціалізації HashiCorp Vault створюється майстер ключ, який використовується для шифрування даних в базі. Він розбивається на декілька

ключів з використанням Shamir’s Secret Sharing. Ініціатор може обрати на скільки ключів він хоче розбити ключ та скільки ключів потрібно надати, щоб розпечатати (unseal) вольт. Таких підхід запобігає повного контролю однією людиною, тим самим зменшуючи ризики несанкціонованого доступу до майстер ключа.

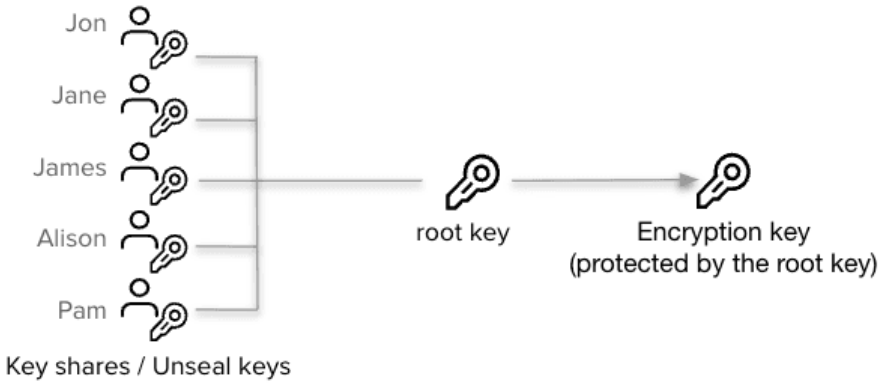


Рис. 3.19. Механізм unseal у Vault[16]

За замовчуванням Vault запускається у стані “sealed” (запечатаний), тобто навіть, якщо він був запущений, доступу до репозиторію немає. Після введення частин майстер ключа, а також токену ініціалізації, який надається після частин майстер ключа, отримується доступ до самого сховища.

Для зберігання ключів налаштовано secret engine типу KV v2, який забезпечує зберігання пар ключ-значення з підтримкою версій та метаданих.
На рис. 3.20 зображений приклад збереження ключа для користувача з id 4284 для алгоритму AES-GCM.

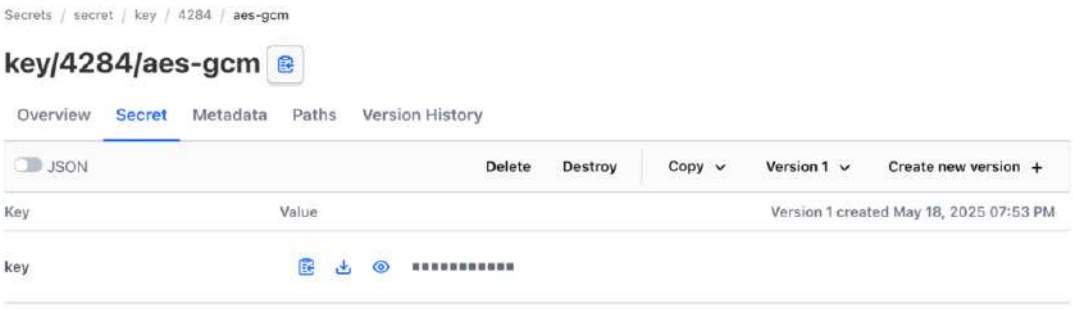


Рис. 3.20. Збереження ключа у HarshiCorp Vault

Кожен запис має унікальний шлях, який використовується в запитах до вольта. Робота з HashiCorp Vault реалізована з використанням Spring Vault, зокрема компонента Vault Template. Для отримання конкретного ключа викликається метод vaultTemplate.read(«шлях до потрібного ключа»), та повертаються значення ключа та метадані.

Компонент VaultKeyProvider використовується для керування криптографічними ключами. У ньому реалізовані функції генерації та збереження у вольт ключів для симетричних та асиметричних алгоритмів, отримання ключів з сховища, а також перевірку існування ключів для заданого алгоритму та юзера (рис. 3.21).

```
public boolean hasSymmetricKey(String userID, String algorithm) {
    String path = "secret/data/key/" + userID + "/" + algorithm.toLowerCase();
    VaultResponse response = vaultTemplate.read(path);
    return ((Map<?, ?>) response.getData().get("data")).get("key") != null;
}
```

Рис. 3.21. Метод перевірки наявності ключа для працівника у Vault

```
    public void writeSymmetricKey(String userID, String algorithm, int keySize) throws
        String path = "secret/data/key/" + userID + "/" + algorithm.toLowerCase();

            String keyAlgorithm = algorithm.split( regex: "-")[0].toUpperCase();
            KeyGenerator keyGen = KeyGenerator.getInstance(keyAlgorithm);
            keyGen.init(keySize);
             byte[] rawKey = keyGen.generateKey().getEncoded();
        String encodedKey = Base64.getEncoder().encodeToString(rawKey);

        vaultTemplate.write(path, Map.of( k1: "data", Map.of( k1: "key", encodedKey)));
    }
}
```

Рис. 3.22. Реалізація методу генерації криптографічного ключа та збереження його у Vault

Генерація та збереження ключів реалізовані методами writeSymmetricKey та writeAsymmetricKey для симетричних та асиметричних алгоритмів відповідно. У випадку симетричного алгоритму шифрування метод формує шлях до вольта для конкретного користувача та алгоритму, за допомогою KeyGenerator.getInstance(), створює ключ потрібного розміру для відповідного алгоритму та кодує його в Base64 і записує за зазначеним шляхом.

Для асиметричних алгоритмів процедура ідентична, проте використовується KeyPairGenerator для створення пари ключів (публічного та приватного), кожен з яких зберігається у визначеному шляху.

Для генерації ключів були використані Java Cryptography Architecture– Java API для криптографічних операцій та Bouncy Castle Provider – провайдер з підтримкою EC – алгоритмів.

3.7 Налаштування захищеного каналу зв’язку (TLS)

Для реалізації захищеного каналу зв’язку між клієнтом та сервером згенеровано самопідписаний сертифікат за допомогою інструмента mkcert, який автоматично створює локальний центр сертифікації (CA), інсталює його в довірене сховище системи та видає сертифікат, якому надається локальна довіра. Після цього цей сертифікат був перетворений у формат PKCS12, який підтримується у Spring Boot. У конфігурації application.properties було додано налаштування для приймання запитів на порті 8443, знаходження сертифікату, спосіб його відкрити (пароль) та набір дозволених версій TLS, виключаючи старі вразливі і запобігаючи атаки POODLE.

```
server.port=8443
server.ssl.enabled=true
server.ssl.key-store=classpath:localhost.p12
server.ssl.key-store-password=
server.ssl.key-store-type=PKCS12
server.ssl.key-alias=localhost
server.ssl.protocol=TLS
server.ssl.enabled-protocols=TLSv1.3,TLSv1.2
```

Рис. 3.23. Конфігурація TLS у Spring Boot

На стороні Angular-клієнта було налаштовано завантаження сертифікату у локальне сховище браузера, що гарантує коректну валідацію та встановлення з’єднання без попередження про ненадійний сертифікат.

На рисунку 3.24 зображено успішний TLS-handshake за версією 1.3, що забезпечує конфіденційність трафіку, цілісність повідомлень та підтвердження автентичності сторін.

```
* Host localhost:4200 was resolved.
* IPv6: ::1
* IPv4: 127.0.0.1
*   Trying [::1]:4200...
* Connected to localhost (::1) port 4200
* ALPN: curl offers h2,http/1.1
* (304) (OUT), TLS handshake, Client hello (1):
*  CAfile: /etc/ssl/cert.pem
*  CApath: none
* (304) (IN), TLS handshake, Server hello (2):
* (304) (IN), TLS handshake, Unknown (8):
* (304) (IN), TLS handshake, Certificate (11):
* (304) (IN), TLS handshake, CERT verify (15):
* (304) (IN), TLS handshake, Finished (20):
* (304) (OUT), TLS handshake, Finished (20):
* SSL connection using TLSv1.3 / AEAD-AES256-GCM-SHA384 / [blank] / UNDEF
* ALPN: server accepted http/1.1
```

Рис. 3.24. Успішне встановлення TLS-з'єднання

3.8 Огляд інтерфейсу програми

Всі ці функції веб-сервісу були реалізовані через зручний інтерфейс з чітко розмежованим доступом відповідно до ролі користувача.

Після відкриття веб-сервісу користувач потрапляє на початковій сторінці авторизації, де йому надається можливість ввести електронну адресу та пароль або здійснити вхід через Google для входу до системи. У разі вибору автентифікації через Google необхідно пройти процедуру авторизації, тим самим надаючи згоду на передачу інформації свого облікового запису веб-сервісу.

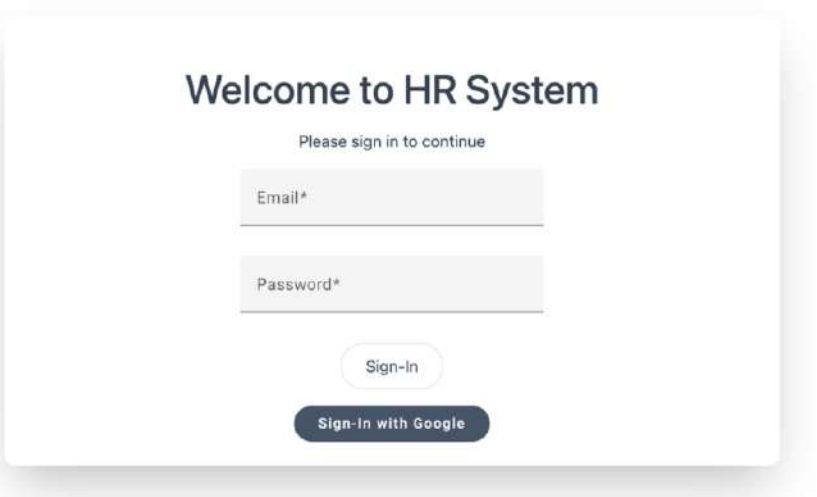


Рис. 3.25. Інтерфейс авторизації користувача в системі управління персоналом

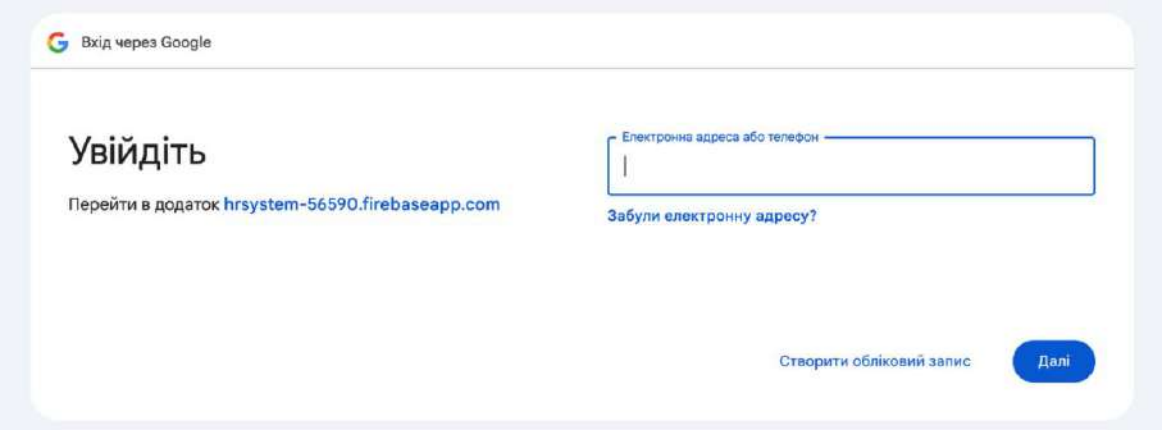


Рис. 3.26. Вікно авторизації через Google

Залежно від ролі користувача, веб-сервіс перенаправляє його на відповідний інтерфейс. В системі присутні три ролі – адміністратор, менеджер з керування персоналу та працівник.

Користувач з роллю «адміністратор» отримує доступ до панелі статистики, яка містить інформацію про кількість працівників в організації, кількість працівників за відділами, кількість працівників за роками роботи, кількість працівників, які почали працювати в компанії останній рік, середню заробітну плату за відділом тощо.

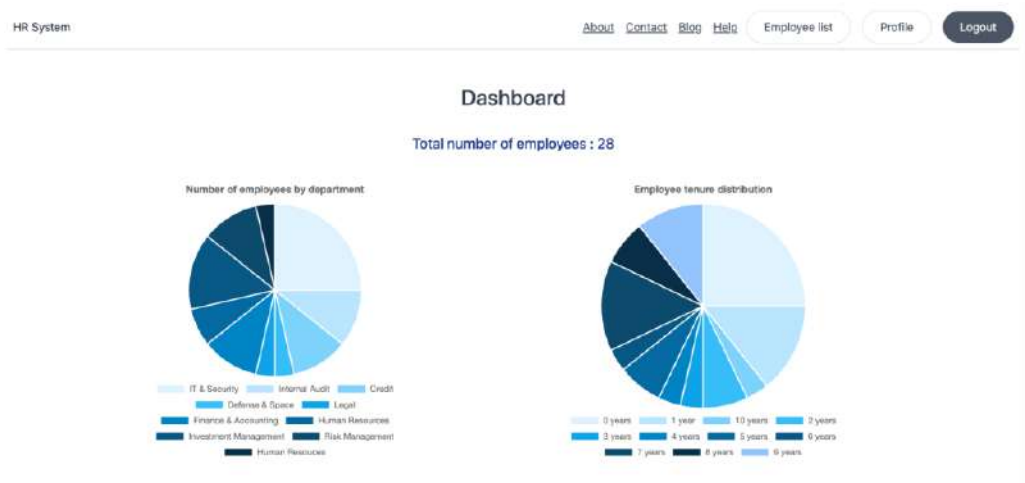


Рис.3.27. Панель статистики на головній сторінці веб-сервісу з діаграмами розподілу працівників та їх стажу

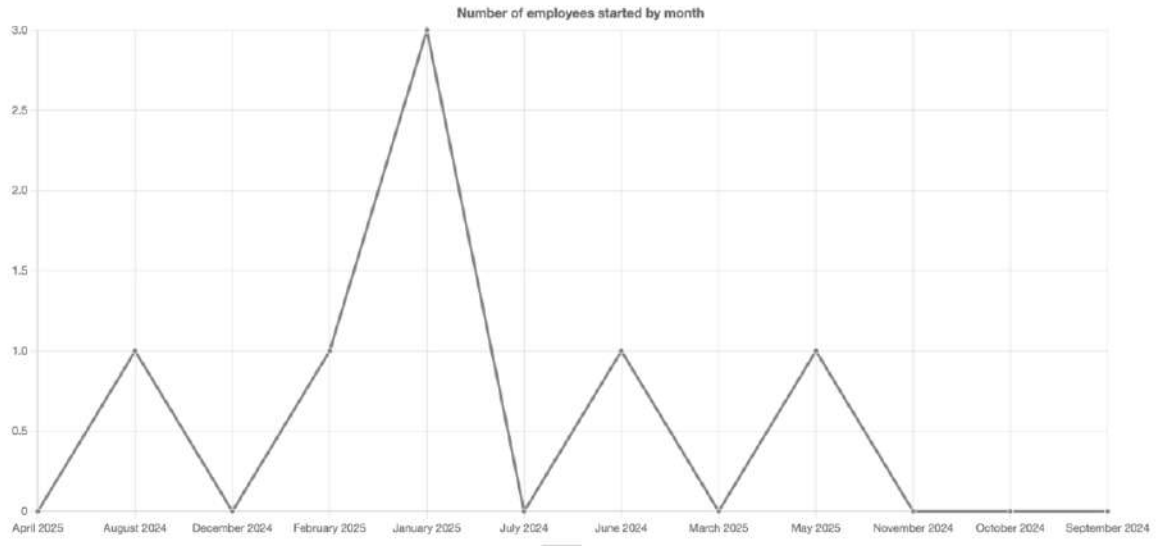


Рис.3.28. Динаміка кількості працівників, які почали роботу в певний місяць

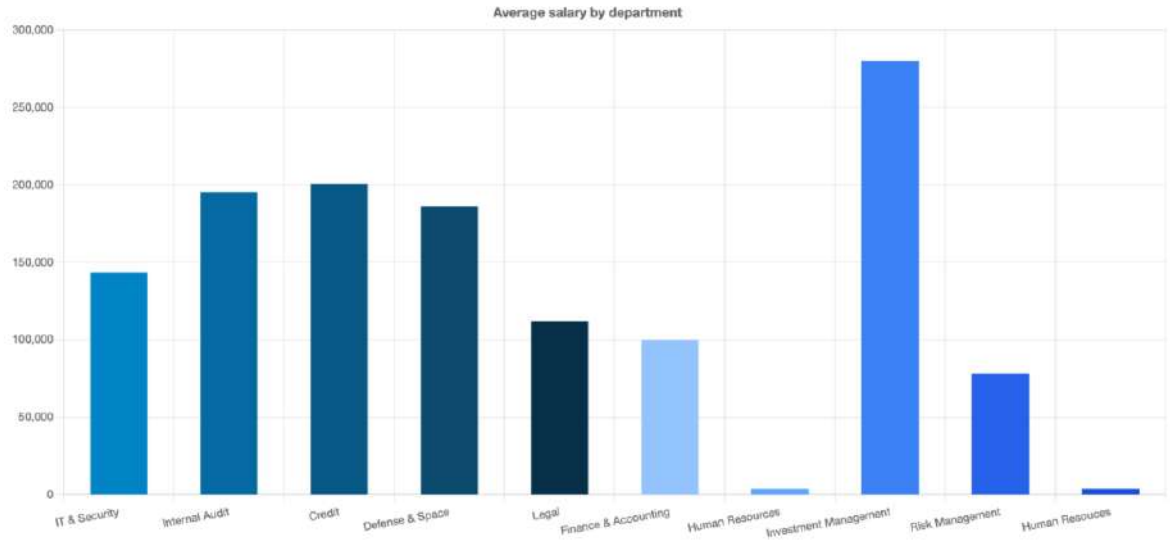


Рис.3.29. Діаграма з статистикою середньої заробітної плати працівників за відділами

При натисканні на кнопку “Employee list” система відображає перелік усіх працівників компанії з можливістю фільтрування даних, пошуку необхідного користувача та додавати нових користувачів.

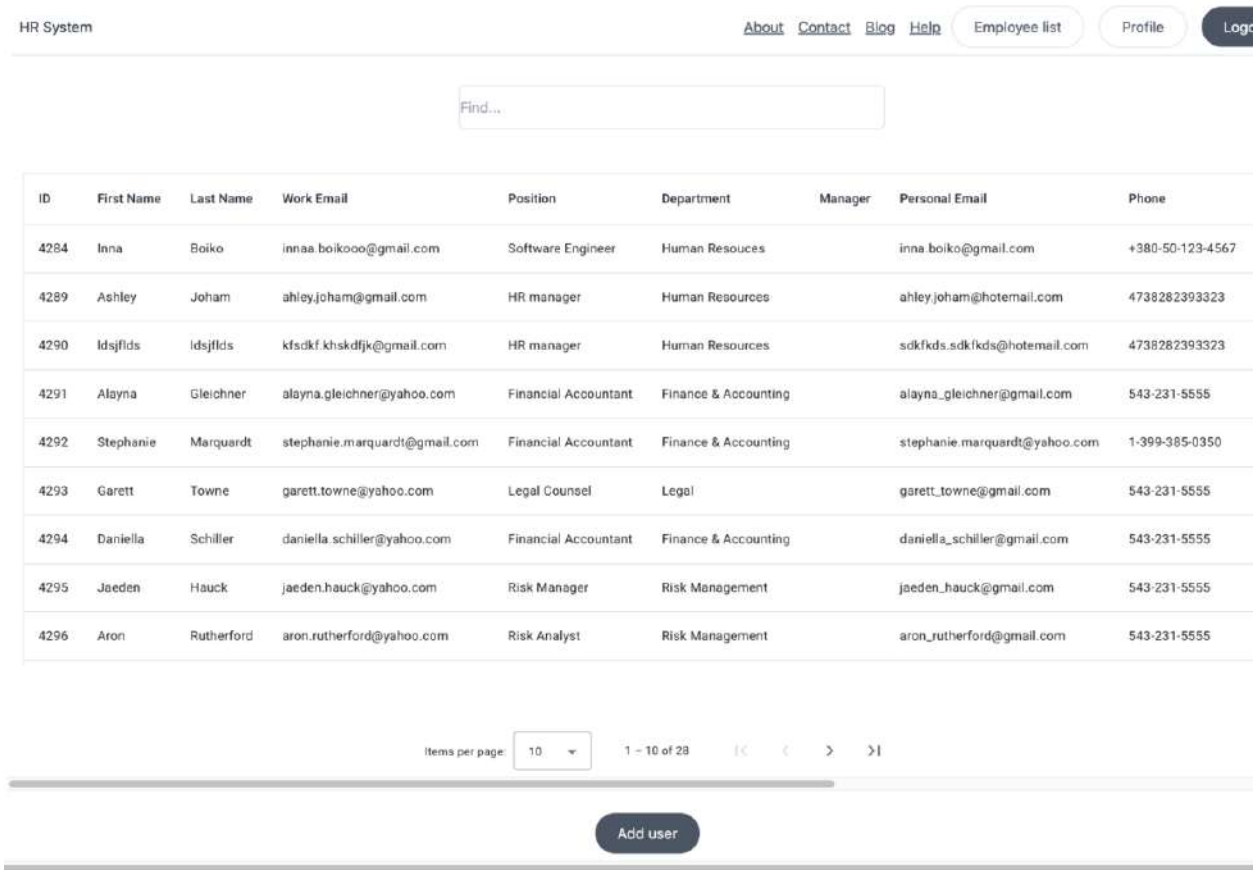


Рис. 3.30. Сторінка "employee list" з таблицею даних працівників

Обрання конкретного користувача надає доступ на перегляд та редагування його даних, які включать:

1. персональні дані - ім’я, персональний імейл, день народження, адреса;
2. службові відомості – посада, відділ, дату початку, дату кінця (якщо є), статус зайнятості (повний чи неповний робочий день),
3. фінансові дані - банківські реквізити, податковий номер;
4. документи - паспорт працівника, контракт з компанією та страховий поліс. Кожен з цих документів може бути завантажений на пристрій.

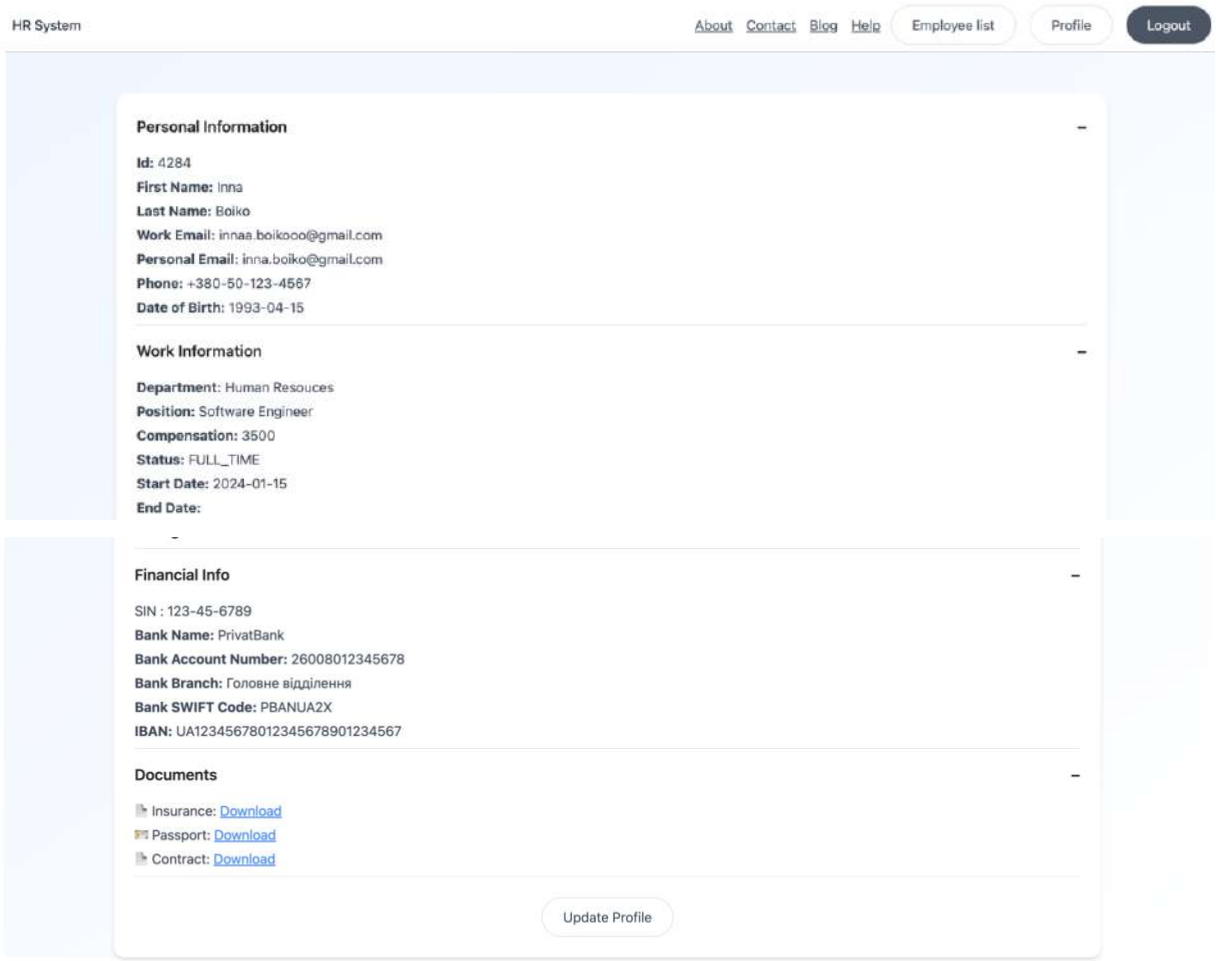
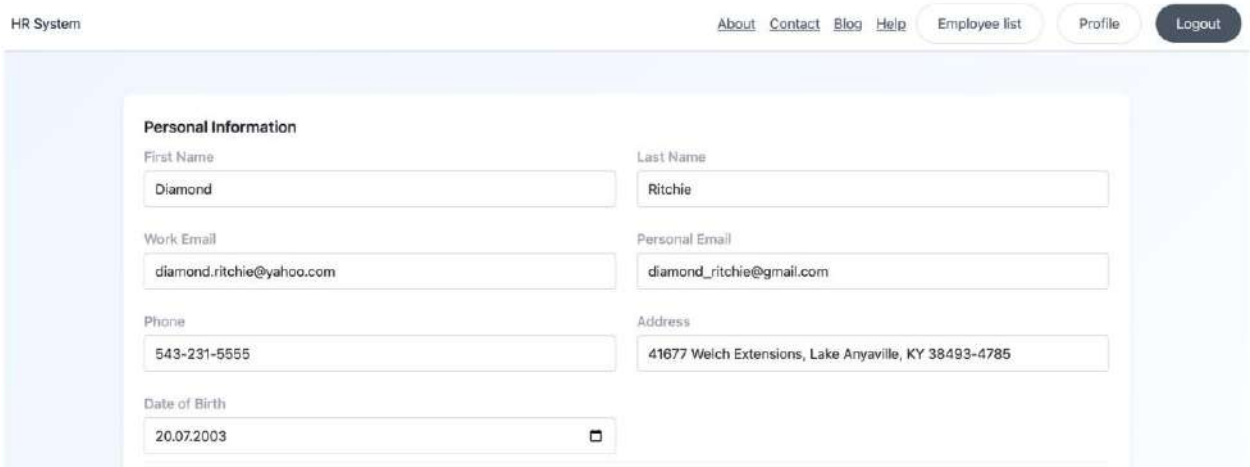


Рис.3.31. Сторінка перегляду детальної інформації про працівника

Для оновлення даних адміністратор натискає кнопку “update profile”, що відкриває форму з передзаповненими полями для зручного редагування.



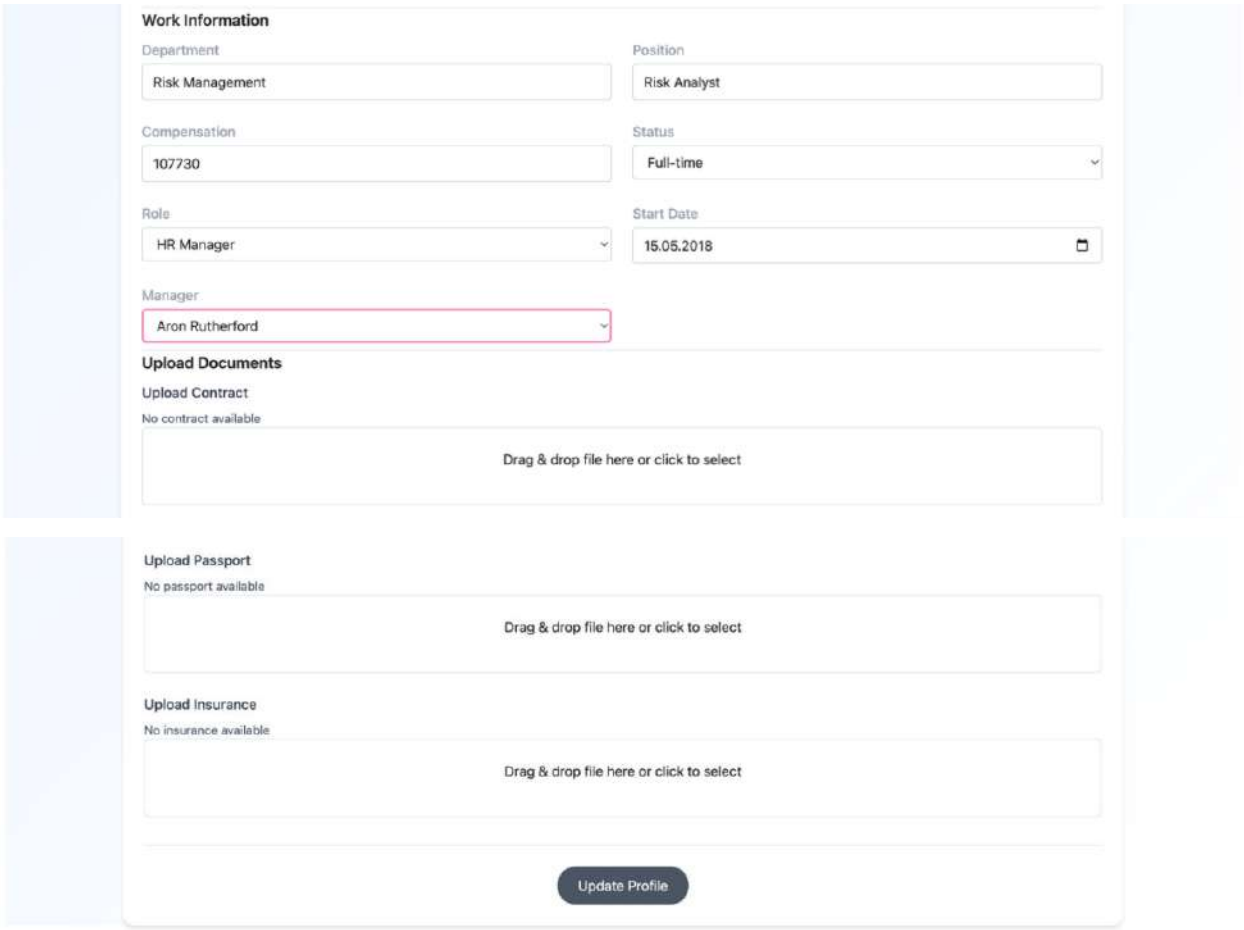


Рис.3.32. Сторінка оновлення даних працівника

Інтерфейс менеджера з управління персоналом відрізняється від інтерфейсу адміністратора лише незначними обмеженнями. Зокрема, він не може додавати нових працівників, бачити фінансову дані працівників, у блоці «Documents» може переглядати чи завантажувати лише контракт працівника.

Працівник після авторизації має змогу переглядати та редагувати виключно власний профіль.

3.9 Тестування захисту чутливих даних

Для оцінки стійкості механізму захисту чутливих даних була змодельована атака “known-plaintext”. Припущено, що атакуючий знає застосований алгоритм (AES-GCM) та факт використання одного ключа для всіх даних одного користувача; також атакуючий має доступ до бази даних із зашифрованими записами та знає відкритий текст, який відповідає певному шифру. Використовуючи фрагмент відомого відкритого тексту (P_1) та відповідний шифротекст (C_1), атакуючий обчислює keystream (L) як XOR операція між C_1 та P_1. За умови використання одного вектора ініціалізації та одного ключа, однаковий keystream застосувався б до іншого шифру (C_2), що дає можливість відновити інший текст за формулою, $P_2 = C_2$ XOR L.

У представленому тесті передбачалась спроба розшифрувати інші поля таблиці саме таким чином. Однак завдяки коректному використанню вектора ініціалізації для кожної операції шифрування розшифрувати P_2 не вдалося(див. рис. 3.33), тож атака виявилась неуспішною.

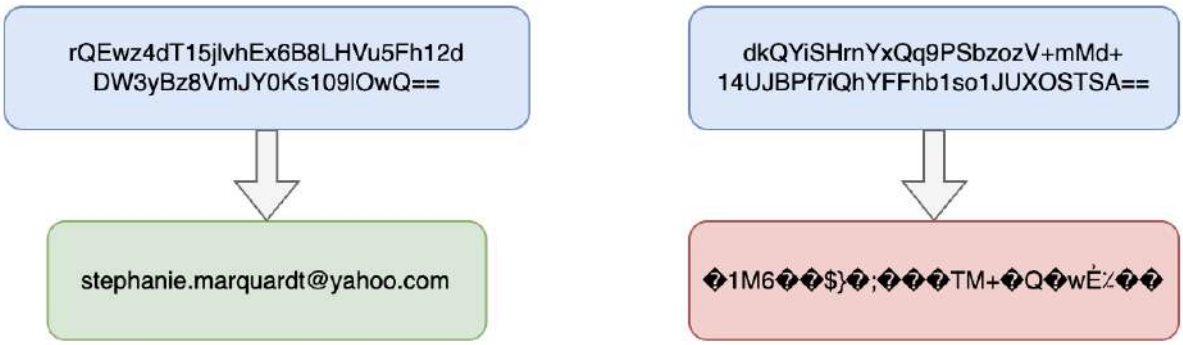


Рис. 3.33. Моделювання "known-plaintext" атаки на AES-GCM

Таким чином, змодельована атака продемонструвала, що алгоритм був впроваджений правильно і це підтверджує стійкість захисту чутливих даних.

Висновок до розділу 3

У третьому розділі сформовано функціональні вимоги до веб-сервісу, а саме додавання та оновлення працівників, робота з даними всіма працівниками та ведення аналітики тощо; спроектовано клієнт-серверну архітектуру та описано використані інструменти під час розробки. Проведено тестування обраних алгоритмів шифрування (AES (128, 192, 256 біт), AES-GCM та AES-CBC, ChaCha20, (128, 256 та 448 біт), DES, 3DES (112 та 168 біт), ECIES (256, 384 та 521 біт)) за дев'ятьма рівнями навантаження (за кількістю оброблених профілів працівників) для оцінки продуктивності; проаналізовано пропускну здатність симетричних алгоритмів.

На основі отриманих результатів було обрано алгоритм AES-GCM як оптимальний, через його безпекову стійкість 256 біт, одні з найкращих показників execution time в тестуванні, найкращу пропускну здатність, а також забезпечення конфіденційності та цілісності даних; деталізовано його інтеграцію у веб-сервіс використовуючи AttributeConverter для автоматизації відповідних операцій. Додатково було змодельовано криптографічну атаку "known-plaintext" для перевірки захищеності реалізації алгоритму шифрування.

Крім того, у систему введено централізоване управління ключами з допомогою HarshiCorp Vault, реалізовано OAuth 2.0-автентифікацію через Google та інтеграцію сервісу Firebase для оптимізації процесу автентифікації, а також забезпечено конфіденційність каналів передачі даних шляхом додавання TLS 1.3.

ВИСНОВОК

У цій кваліфікаційній роботі було розкрито поняття чутливих даних, виділено групи даних високої чутливості, а також розглянуто основні криптографічні методи їх захисту. Були висвітлені поняття симетричних та асиметричних алгоритмів шифрування та досліджено алгоритми DES, 3DES, Blowfish, AES, ChaCha20, та ECIES. Визначено критерії оцінки алгоритмів шифрування з погляду безпекової стійкості та продуктивності. Висвітлились метрики Attack steps metric, Attack time metric, візуальна стійкість, avalanche ефект, execution time та пропускна здатність, способи їх вимірювання та використання. Розглянуто можливі способи криптоаналізу алгоритмів, включаючи атаки методом перебору криптографічних ключів, атаки chosen-plaintext, known-plaintext, downgrade та атака через повторне використання ключа. Крім того, досліджено підходи до визначення рівня захищеності веб-сервісу ґрунтуючись на методології оцінки ризиків та описано найкращі практики управління криптографічними ключами, які включають використання централізованого репозиторію для зберігання криптографічних ключів, їх ротацію, контролі доступу та моніторинг всіх операцій з ключами.

У роботі проаналізовано веб-сервіс управління персоналом у обраній організації за напрями автентифікації та авторизації, контролів доступу, безпеки чутливих даних, моніторингу та актуальності компонентів програмного забезпечення. Було виявлено вразливості веб-сервісу до яких належали відсутність шифрування даних в базі даних, застарілий протокол захисту каналів передачі даних та використання застарілого програмного забезпечення. З допомогою матриці оцінки ризиків проводилась оцінка ризиків цих вразливостей та запропоновано такі безпекові заходи для зниження ризику: впровадження криптографічного захисту інформації шляхом впровадження ефективного

алгоритму шифрування, використання актуального та підтримуваного програмного забезпечення для розробки веб-сервісу та застосування актуальної версії 1.3 протоколу TLS при передачі даних.

Практична частина роботи полягала у розробці прототипу обраного веб-сервісу управління персоналом та підвищення рівня його безпеки шляхом застосування сучасних криптографічних методів шифрування для захисту чутливих даних, впровадження механізму управління ключами та усунення загроз, пов'язаних із використанням неактуальних компонентів програмного забезпечення.

Для розробки веб-сервісу було спроєктовано клієнт-серверну архітектуру із багатошаровим поділом обох частин. Серверна частина реалізована на Java з використанням Spring Boot та розділена на три основні шари: шар представлення, шар бізнес логіки та шар доступу до бази даних. Клієнтська частина розроблена з використанням Angular з компонентною та модульною архітектурою: додано Injectors, функціональні модулі, які розділяють доступи за ролями та компоненти, які відповідають за відображення інтерфейсу. Спроєктовано базу даних з таблицями user, user_documents, financial_details та role. Для її реалізації обрано MySQL.

Для знаходження оптимального алгоритму шифрування було досліджено алгоритми шифрування (AES (128, 192, 256 біт), AES-GCM та AES-CBC, ChaCha20, (128, 256 та 448 біт), DES, 3DES (112 та 168 біт), ECIES (256, 384 та 521 біт)) з позиції оцінки їх безпекової стійкості та продуктивності спираючись на потрібну структуру даних. Для кожного алгоритму обраховуватись Attack steps metric та Attack time metric, execution time та пропускну здатність шляхом тестування за дев'ятьма рівнями навантаження з метою вибору оптимального алгоритму для різних сценаріїв використання.

На основі отриманих результатів було обрано алгоритм AES-GCM як оптимальний, через його безпекову стійкість 256 біт, одні з найкращих показників часу виконання шифрування, найкращу пропускну здатність, а також забезпечення конфіденційності та цілісності даних; деталізовано його інтеграцію у веб-сервіс використовуючи AttributeConverter для автоматизації відповідних операцій і змодельовано криптографічну атаку “known-plaintext” для перевірки захищеності реалізації алгоритму шифрування.

Результатом роботи став розроблений веб-сервіс керування персоналом з підвищеним рівнем безпеки за допомогою таких рішень:

1. імплементації алгоритму шифрування AES-GCM для зберігання чутливих даних;
2. забезпечення конфіденційності каналів передачі даних шляхом додавання TLS 1.3;
3. введення у сервіс централізованого управління ключами використовуючи HarshiCorp Vault;
4. реалізації OAuth 2.0-автентифікації через Google;
5. інтеграції сервісу Firebase для оптимізації процесу автентифікації;
6. впровадження ролей для контролю доступу;
7. використання актуального програмного забезпечення.

Розроблений веб-сервіс забезпечує виконання критичних операцій з управління персоналом, такі як додавання нових працівників, пошук та відображення детальної інформації про конкретного працівника, додання та завантаження та зберігання необхідних документів, перегляд та оновлення власних даних, та побудову статистичних звітів у вигляді графіків для аналізу ключових показників і оптимізації процесів кадрового менеджменту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Cost of a data breach 2024 | IBM. IBM - United States. URL: https://www.ibm.com/reports/data-breach
2. Lack of encryption the primary reason for sensitive data loss. *Security Magazine | The business magazine for security executives*. URL: https://www.securitymagazine.com/articles/100227-lack-of-encryption-the-primary-reason-for-sensitive-data-loss
3. Data breach costs rising, financial impact felt for years, study - Cloud Boardroom. *Cloud Boardroom*. URL: https://cloudboardroom.eu/2019/08/01/data-breach-costs-rising-financial-impact-felt-for-years-study/
4. GeeksforGeeks. Advanced encryption standard (AES) - geeksforgeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/advanced-encryption-standard-aes/?ref=rp
5. Galois/Counter mode - wikipedia. *Wikipedia*. URL: https://en.wikipedia.org/wiki/Galois/Counter_Mode (дата звернення: 29.05.2025).
6. ChaCha20-Poly1305 - wikipedia. *Wikipedia, the free encyclopedia*. URL: https://en.wikipedia.org/wiki/ChaCha20-Poly1305
7. Elliptic curve integrated encryption scheme (ECIES). *Security and So Many Things*. URL: https://asecuritysite.com/ecies/index
8. NIST SP 800-57 Part 1 Revision 4. Recommendation for key management, part 1: general. На заміну NIST SP 800-57 Part 1 Rev. 4. Вид. офіц. Gaithersburg, MD : U.S. Department of Commerce, NIST, 2016. 161 с. URL: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r4.pdf.
9. Jorstad N. D. Cryptographic algorithm metrics. Institute for Defense Analyses, 1997. 38 с. URL:

https://csrc.nist.gov/files/pubs/conference/1997/10/10/proceedings-of-the-20th-nissc-1997/final/docs/128.pdf.

10. Highlights - november 2024 | TOP500. *Home | TOP500*. URL: https://www.top500.org/lists/top500/2024/11/highs/
11. OWASP Top Ten | OWASP Foundation. *OWASP Foundation, the Open Source Foundation for Application Security | OWASP Foundation*. URL: https://owasp.org/www-project-top-ten/
12. NIST SP 800-52r2. Guidelines for the selection, configuration, and use of transport layer security (TLS) implementations. На заміну NIST SP 800-52r1 ; чинний від 2019-08-16. Вид. офіц. Gaithersburg, MD : National Institute of Standards and Technology, 2019. 72 с. URL: https://doi.org/10.6028/NIST.SP.800-52r2.
13. NIST withdraws outdated data encryption standard. *NIST*. URL: https://www.nist.gov/news-events/news/2005/06/nist-withdraws-outdated-data-encryption-standard
14. Cobb M. What is Triple DES and why is it being disallowed? | TechTarget. *Search Security*. URL: https://www.techtarget.com/searchsecurity/tip/Expert-advice-Encryption-101-Triple-DES-explained
15. Cooper K. How old is the universe?. *Space*. URL: https://www.space.com/24054-how-old-is-the-universe.html
16. Seal/Unseal | vault | hashicorp developer. *Seal/Unseal | Vault | HashiCorp Developer*. URL: https://developer.hashicorp.com/vault/docs/concepts/seal

ДОДАТОК 1

Результати тестування алгоритмів шифрування на різних об’ємах даних.

Алоритм	Обсяг даних (байти)	Час шифрування (мс)	Час розшифрування (мс)	Середній час шифрування (мс)
Algorithm	**Size Bytes**	**Encrypt Time Ms**	**Decrypt Time Ms**	**Average Time Ms**
BLOWFISH-256	7822	15,775	15,829	15,802
DES	7822	17,935	16,659	17,297
BLOWFISH-448	7822	18,854	18,293	18,5735
3DES-112	7822	19,505	17,647	18,576
3DES-168	7822	19,129	18,233	18,681
BLOWFISH-128	7822	20,703	18,8	19,7515
AES-256	7822	25,121	26,155	25,638
AES-GCM	7822	32,084	20,597	26,3405
AES-CBC	7822	28,862	25,34	27,101
ChaCha20	7822	43,062	19,801	31,4315
AES-192	7822	33,819	31,076	32,4475
ECIES-384	7822	42,684	36,658	39,671
AES-128	7822	43,295	39,822	41,5585
ECIES-521	7822	46,328	37,539	41,9335
ECIES-256	7822	244,873	44,407	144,64
DES	30942	26,7	25,398	26,049
3DES-168	30942	29,113	28,222	28,6675
3DES-112	30942	29,887	29,013	29,45
BLOWFISH-448	30942	32,107	29,092	30,5995
BLOWFISH-256	30942	40,368	37,125	38,7465
BLOWFISH-128	30942	52,948	52,527	52,7375
AES-GCM	30942	63,517	78,334	70,9255
AES-CBC	30942	82,128	76,152	79,14
ChaCha20	30942	93,105	65,45	79,2775
AES-192	30942	86,449	80,248	83,3485
AES-256	30942	81,747	86,615	84,181
AES-128	30942	130,421	113,619	122,02
ECIES-521	30942	132,484	112,02	122,252
ECIES-384	30942	142,208	137,544	139,876
ECIES-256	30942	348,271	136,165	242,218
BLOWFISH-256	95630	75,742	76,247	75,9945

3DES-168	95630	79,027	76,173	77,6
BLOWFISH-128	95630	81,818	78,94	80,379
3DES-112	95630	102,233	94,898	98,5655
BLOWFISH-448	95630	118,035	109,319	113,677
AES-GCM	95630	127,129	117,079	122,104
ChaCha20	95630	158,003	112,623	135,313
DES	95630	157,224	134,268	145,746
AES-256	95630	167,281	158,033	162,657
AES-CBC	95630	202,793	195,682	199,2375
AES-192	95630	234,718	199,878	217,298
ECIES-384	95630	279,567	218,516	249,0415
ECIES-521	95630	289,501	232,682	261,0915
AES-128	95630	314,639	292,466	303,5525
ECIES-256	95630	625,178	335,756	480,467
ChaCha20	1746296	785,546	764,372	774,959
AES-GCM	1746296	784,804	773,096	778,95
DES	1746296	828,393	815,517	821,955
3DES-168	1746296	853,341	857,43	855,3855
BLOWFISH-448	1746296	859,209	883,82	871,5145
AES-CBC	1746296	906,72	848,428	877,574
3DES-112	1746296	869,053	899,359	884,206
AES-256	1746296	940,983	937,85	939,4165
BLOWFISH-128	1746296	1043,239	1007,913	1025,576
BLOWFISH-256	1746296	1016,882	1039,674	1028,278
AES-192	1746296	1141,145	1123,998	1132,5715
AES-128	1746296	1886,192	1830,875	1858,5335
ECIES-256	1746296	4789,353	3665,982	4227,6675
ECIES-384	1746296	5063,219	4047,274	4555,2465
ECIES-521	1746296	5390,358	4263,785	4827,0715
DES	451906	215,579	201,741	208,66
BLOWFISH-448	451906	235,267	232,359	233,813
BLOWFISH-256	451906	240,012	233,079	236,5455
3DES-168	451906	237,108	238,306	237,707
3DES-112	451906	240,15	240,157	240,1535
BLOWFISH-128	451906	264,352	261,182	262,767
AES-CBC	451906	274,356	259,807	267,0815
AES-256	451906	267,572	267,539	267,5555
ChaCha20	451906	288,585	249,259	268,922
AES-192	451906	281,753	289,193	285,473

AES-GCM	451906	329,879	310,209	320,044
AES-128	451906	849,702	784,88	817,291
ECIES-384	451906	1190,476	974,888	1082,682
ECIES-521	451906	1310,693	1020,209	1165,451
ECIES-256	451906	1589,997	1088,836	1339,4165
DES	938276	391,149	390,564	390,8565
ChaCha20	938276	466,204	424,147	445,1755
3DES-168	938276	457,344	463,623	460,4835
3DES-112	938276	464,275	459,88	462,0775
BLOWFISH-448	938276	470,847	471,814	471,3305
BLOWFISH-256	938276	472,526	472,097	472,3115
BLOWFISH-128	938276	491,643	492,284	491,9635
AES-256	938276	571,683	574,506	573,0945
AES-GCM	938276	597,854	594,205	596,0295
AES-CBC	938276	617,568	580,973	599,2705
AES-192	938276	634,703	634,058	634,3805
AES-128	938276	1109,466	1091,797	1100,6315
ECIES-256	938276	2611,652	1911,158	2261,405
ECIES-384	938276	2771,963	2172,18	2472,0715
ECIES-521	938276	2808,77	2278,964	2543,867
AES-GCM	2887104	1195,758	1168,452	1182,105
AES-CBC	2887104	1216,636	1183,113	1199,8745
ChaCha20	2887104	1265,651	1190,352	1228,0015
AES-256	2887104	1286,81	1276,304	1281,557
BLOWFISH-256	2887104	1399,292	1396,832	1398,062
BLOWFISH-448	2887104	1429,407	1411,167	1420,287
AES-192	2887104	1505,382	1506,173	1505,7775
3DES-168	2887104	2613,055	1502,631	2057,843
DES	2887104	2772,218	1355,916	2064,067
3DES-112	2887104	2649,564	1548,738	2099,151
AES-128	2887104	2189,829	2138,694	2164,2615
BLOWFISH-128	2887104	3093,487	1538	2315,7435
ECIES-256	2887104	8489,259	6641,44	7565,3495
ECIES-384	2887104	8775,578	6738,992	7757,285
ECIES-521	2887104	9035,593	6869,122	7952,3575
AES-256	3899103	1495,256	1484,597	1489,9265
DES	3899103	1547,289	1548,447	1547,868
AES-CBC	3899103	1582,982	1548,089	1565,5355
AES-GCM	3899103	1609,785	1581,543	1595,664

ChaCha20	3899103	1821,251	1762,627	1791,939
AES-192	3899103	1817,353	1775,813	1796,583
3DES-112	3899103	1893,832	1891,326	1892,579
BLOWFISH-448	3899103	1904,925	1904,315	1904,62
BLOWFISH-256	3899103	1917,307	1906,973	1912,14
3DES-168	3899103	2088,9	2084,186	2086,543
BLOWFISH-128	3899103	2144,88	2181,14	2163,01
AES-128	3899103	2735,293	2637,033	2686,163
ECIES-256	3899103	11795,22	9026,455	10410,8375
ECIES-384	3899103	12323,698	9267,463	10795,5805
ECIES-521	3899103	12714,996	9613,466	11164,231
AES-256	4474669	1694,353	1679,8	1687,0765
DES	4474669	1767,469	1765,032	1766,2505
AES-GCM	4474669	1810,892	1786,908	1798,9
AES-CBC	4474669	1916,4	1858	1887,2
AES-192	4474669	1897,954	1884,292	1891,123
BLOWFISH-128	4474669	2145,104	2150,197	2147,6505
3DES-112	4474669	2201,053	2165,074	2183,0635
3DES-168	4474669	2217,135	2221,769	2219,452
BLOWFISH-256	4474669	2204,697	2243,652	2224,1745
BLOWFISH-448	4474669	2450,226	2495,091	2472,6585
ChaCha20	4474669	2565,758	2439,232	2502,495
AES-128	4474669	3201,531	3143,146	3172,3385
ECIES-521	4474669	11676,934	9496,665	10586,7995
ECIES-384	4474669	13719,636	10695,199	12207,4175
ECIES-256	4474669	13930,633	10891,595	12411,114