НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ЮРИДИЧНА АКАДЕМІЯ»
ФАКУЛЬТЕТ КІБЕРБЕЗПЕКИ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

Кафедра кібербезпеки

КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«МЕТОДИ ЗАХИСТУ ПЕРСОНАЛЬНИХ ДАНИХ В ОНЛАЙН-БАНКІНГУ: ПОРІВНЯЛЬНИЙ АНАЛІЗ ТЕХНОЛОГІЙ ШИФРУВАННЯ»

Виконав: здобувач 4 курсу
Рівень бакалавр
Галузь знань 12 «Інформаційні технології»,
спеціальність 125 «Кібербезпека та захист інформації»
Величко Софія Владиславівна

Керівник: д.т.н., професор
Соколов Артем Вікторович

Рецензент: к.т.н., доцент
Кухаренко Сергій Вікторович

Одеса – 2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ЮРИДИЧНА АКАДЕМІЯ»
Факультет **кібербезпеки та інформаційних технологій**
Кафедра **кібербезпеки**
Галузь знань: **12 Інформаційні технології**
Спеціальність: **125 Кібербезпека та захист інформації**
Назва освітньої програми: **Кібербезпека**

ЗАТВЕРДЖУЮ
Завідувач кафедри кібербезпеки
професор С.А. Горбаченко
_____ «____ »_____ 20__ року

З А В Д А Н Н Я
на кваліфікаційну (бакалаврську) роботу
здобувачу Величко Софії Владиславівні

1. Тема роботи: «Методи захисту персональних даних в онлайн-банкінгу: порівняльний аналіз технологій шифрування»
Керівник роботи: д.т.н, професор Соколов Артем Вікторович
затверджені наказом НУ ОЮА від «18» березня 2025 року № 548-6
2. Строк подання здобувачем роботи «19» травня 2025 року
3. Дата видачі завдання «16» вересня 2024 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка

Здобувач _____
(підпис) (прізвище та ініціали)

Керівник роботи _____
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Кваліфікаційна робота на тему «Методи захисту персональних даних в онлайн-банкінгу: порівняльний аналіз технологій шифрування» на здобуття першого (бакалаврського) рівня вищої освіти за спеціальністю 125 Кібербезпека та захист інформації, освітньою програмою: Кібербезпека, містить 30 рисунків, 10 таблиць, 2 додатки, 35 літературних джерел за переліком посилань. Робота виконана на 90 сторінках загального тексту і 68 сторінках основного тексту.

Метою роботи є аналіз сучасних методів захисту персональних даних в онлайн-банкінгу та проведення порівняльного аналізу технологій шифрування з подальшим впровадженням найбільш ефективних засобів у вигляді захищеного вебзастосунку.

У результаті виконання кваліфікаційної роботи було досліджено актуальні загрози інформаційній безпеці в онлайн-банкінгу та проаналізовано ефективність сучасних криптографічних технологій, що використовуються для захисту персональних даних. В рамках практичної частини було реалізовано вебзастосунок Secure Bank — прототип «сайту-фортеці», що імітує функціонал сучасного банківського онлайн-сервісу з підвищеним рівнем безпеки та проведено пентестинг.

Результати роботи можуть бути використані для подальших досліджень у галузі кібербезпеки, удосконалення практичних рішень у сфері фінансових онлайн-сервісів, а також як база для впровадження нових механізмів захисту в реальних інформаційних системах.

Можливими напрямками подальших досліджень є вивчення постквантових алгоритмів шифрування для підвищення стійкості до загроз; розробка інтелектуальних систем для виявлення аномальної активності користувачів у режимі реального часу; застосування поведінкової біометрії як інструменту адаптивної автентифікації; а також дослідження ефективності впровадження Zero Trust-архітектури в інформаційних системах фінансових установ

ІНФОРМАЦІЙНА БЕЗПЕКА, ШИФРУВАННЯ ДАНИХ, БЕЗПЕКА ПРОТОКОЛІВ, ОНЛАЙН-БАНКІНГ, АТАКИ НА КОНФІДЕНЦІЙНІ ДАНІ

ABSTRACT

The qualification work on the topic « Methods for securing personal data in online banking: contemporary analysis of encryption technologies» for obtaining the first (bachelor's) level of higher education in the specialty 125 Cybersecurity and information protection, educational program: Cybersecurity, contains 30 figures, 10 tables, 2 appendices, 35 literary sources according to the list of references. The work is performed on 90 pages of general text and 68 pages of the main text.

The purpose of the work is to analyze modern methods of protecting personal data in online banking and conduct a comparative analysis of encryption technologies with the subsequent implementation of the most effective means in the form of a secure web application.

As a result of the qualification work, current threats to information security in online banking were investigated and the effectiveness of modern cryptographic technologies used to protect personal data was analyzed. As part of the practical part, the Secure Bank web application was implemented - a prototype of a "fortress site" that simulates the functionality of a modern online banking service with an increased level of security, and pentesting was conducted.

The results of the work can be used for further research in the field of cybersecurity, improving practical solutions in the field of online financial services, as well as as a basis for implementing new protection mechanisms in real information systems.

Possible areas of further research include studying post-quantum encryption algorithms to increase resilience to threats; developing intelligent systems to detect anomalous user activity in real time; using behavioral biometrics as an adaptive authentication tool; and investigating the effectiveness of implementing Zero Trust architecture in information systems of financial institutions.

INFORMATION SECURITY, DATA ENCRYPTION, PROTOCOL SECURITY, ONLINE BANKING, ATTACKS ON CONFIDENTIAL DATA

ЗМІСТ

ВСТУП 6

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ЗАХИСТУ ПЕРСОНАЛЬНИХ ДАНИХ В ОНЛАЙН-БАНКІНГУ 9

1.1. Загальні принципи безпеки в онлайн-банкінгу 9

1.2. Нормативно-правова база захисту персональних даних 10

2. АНАЛІЗ ЗАГРОЗ ТА ВРАЗЛИВОСТЕЙ В ОНЛАЙН-БАНКІНГУ ... 19

2.1. Методи атак на системи онлайн-банкінгу 19

2.2. Вразливості веб-сайтів фінансових установ 24

2.3. Аналіз фінансової безпеки сайтів банків та маркетплейсу 27

2.4. Основні алгоритми шифрування 40

2.5. Протоколи захисту передавання даних 44

2.6. Оцінка ефективності технологій шифрування в банківському секторі 47

3 ПРАКТИЧНЕ ДОСЛІДЖЕННЯ: СТВОРЕННЯ ТА АНАЛІЗ "САЙТУ-ФОРТЕЦІ" 50

3.1.Розробка "сайту-фортеці" 50

3.2. Проведення тестування на проникнення (пентестинг) 56

3.3.Користувацький інтерфейс 60

3.4.Виявлені вразливості та рекомендації щодо їх усунення 64

ВИСНОВКИ 67

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 71

ДОДАТОК А. Графічне представлення робочого процесу recaptcha 74

ДОДАТОК Б. Повний код реалізованого застосунку 75

ВСТУП

1. Актуальність теми

У сучасному цифровому середовищі онлайн-банкінг є не лише зручним засобом доступу до фінансових послуг, а й потенційним об’єктом для кіберзагроз. Згідно з дослідженнями, зростання цифровізації та активне використання Інтернету призводять до збільшення ризиків витоку персональних даних, що може мати серйозні наслідки для приватності та безпеки громадян [24]. Постійне зростання обсягів персональних даних, що обробляються фінансовими установами, зумовлює підвищення вимог до захисту інформації. Зломи облікових записів, викрадення паролів, атаки типу «людина посередині» (MITM), фішинг, SQL-ін’єкції, а також використання вразливих сторонніх сервісів — це лише частина загроз, з якими стикаються користувачі та розробники банківських систем.

Онлайн-банкінг став невід'ємною частиною фінансових послуг, надаючи користувачам зручний доступ до управління своїми фінансами. Проте, зростання популярності таких сервісів супроводжується підвищеними вимогами до безпеки, оскільки фінансові транзакції та особисті дані клієнтів стають мішенню для кіберзлочинців. Забезпечення надійного захисту інформації в онлайн-банкінгу є критично важливим для підтримки довіри користувачів та стабільності фінансової системи

Кіберзагрози стають все більш складними та витонченими, що створює серйозні виклики для захисту персональних даних. Згідно зі Звітом про цифровий захист Microsoft 2024 року, щодня фіксується понад 600 мільйонів кібератак, включаючи фішинг, програми-вимагачі та інші форми злочинної діяльності [23]. У цьому контексті, ефективне шифрування даних є ключовим інструментом для забезпечення конфіденційності та цілісності інформації, особливо в сфері онлайн-банкінгу, де обробляються чутливі фінансові дані.

Наукові дослідження та практичні розробки в цій галузі зосереджені на вдосконаленні технологій шифрування, автентифікації користувачів та захисту каналів зв’язку. Проте в реальних умовах захищеність залежить не лише від

вибору конкретного криптографічного алгоритму, а й від комплексності підходу, що поєднує кілька рівнів безпеки. Саме тому проведення порівняльного аналізу існуючих технологій захисту, виявлення вразливостей і демонстрація практичного застосування сучасних методів є вкрай актуальними.

2. Мета і завдання дослідження

Метою даної роботи є аналіз сучасних методів захисту персональних даних в онлайн-банкінгу та проведення порівняльного аналізу технологій шифрування з подальшим впровадженням найбільш ефективних засобів у вигляді захищеного вебзастосунку.

Для досягнення цієї мети були поставлені наступні завдання:

1. провести аналіз типових загроз і вразливостей в онлайн-банкінгу;
2. дослідити сучасні алгоритми шифрування та протоколи захисту даних;
3. оцінити ефективність застосовуваних криптографічних технологій;
4. розробити прототип «сайту-фортеці» із вбудованими механізмами захисту;
5. провести тестування реалізованого програмного забезпечення на наявність вразливостей;
6. сформувати рекомендації щодо підвищення рівня захисту персональних даних у вебзастосунках.

Об’єктом дослідження є процес захисту персональних даних в онлайн-банкінгу.

Предметом дослідження виступають технології шифрування, автентифікації та методи захисту інформації, які застосовуються у вебзастосунках фінансового призначення.

3. Практичне значення отриманих результатів

Практична частина роботи реалізована у вигляді вебзастосунку Secure Bank, який інтегрує сучасні технології безпеки: автентифікацію через пароль і Passkey (WebAuthn), використання TLS, механізми обмеження кількості спроб входу, захист через reCAPTCHA v3, хешування паролів, перевірку активності

сесій, CS-захист, а також аналіз вразливостей. Отримані результати можуть бути використані як приклад реалізації комплексного захисту в вебзастосунках банківської сфери, а також як основа для подальших досліджень і вдосконалень у галузі кібербезпеки.

4. Публікації та апробація кваліфікаційної роботи

Основні результати кваліфікаційної роботи були апробовані через наукову публікацію, підготовлену на основі результатів проведеного дослідження. Зокрема, результати аналізу вразливостей персональних даних на фінансових вебсайтах, а також підходів до їх усунення, були представлені у науковій статті Fedorenko O.H., Velychko S.V., Kaidan Y.V. Investigating vulnerabilities of personal data on financial websites. — Матеріали конференції, індексується у базі даних Scopus.

У публікації викладено основні теоретичні положення дослідження, підтверджено актуальність проблеми захисту персональних даних в онлайн-банкінгу, проаналізовано типові загрози та запропоновано шляхи їх мінімізації за допомогою сучасних криптографічних технологій і комплексних захисних рішень.

Стаття була опублікована в рамках 7-го семінару для молодих учених з комп’ютерних наук та інженерії програмного забезпечення (CS&SE@SW 2024), що відбувся 27 грудня 2024 року у місті Кривий Ріг, Україна. Вона була опублікована в CEUR Workshop Proceedings, том 3917, сторінки 451–458, і доступна у відкритому доступі за посиланням:

https://ceur-ws.org/Vol-3917/paper64.pdf

РОЗДІЛ 1. ТЕОРЕТИЧНІ ОСНОВИ ЗАХИСТУ ПЕРСОНАЛЬНИХ ДАНИХ В ОНЛАЙН-БАНКІНГУ

1.1. Загальні принципи безпеки в онлайн-банкінгу

Онлайн-банкінг є одним із найважливіших елементів сучасної фінансової інфраструктури. Зручність віддаленого доступу до банківських послуг супроводжується значними ризиками, пов'язаними з кіберзагрозами.

Безпека онлайн-банкінгу є важливим аспектом проведення фінансових ранзакцій через інтернет. Вона охоплює різні заходи, політики, технології та процеси, що впроваджуються для забезпечення конфіденційності, цілісності та доступності фінансової інформації [1].

Основні принципи безпеки в онлайн-банкінгу:

Конфіденційність – забезпечення захисту персональних даних та фінансової інформації користувачів від несанкціонованого доступу. Для цього використовуються сучасні криптографічні алгоритми, такі як AES та RSA, а також протоколи захищеного зв’язку, наприклад, SSL/TLS.

Цілісність даних – гарантія того, що передані та збережені дані не були змінені або скомпрометовані сторонніми особами. Цього досягають за допомогою цифрових підписів, хешування (SHA-256, SHA-3) та контролю версійності даних.

Доступність – безперебійна робота банківських сервісів для клієнтів при дотриманні високих стандартів безпеки. Це передбачає використання системи резервування серверів, балансування навантаження та захисту від DDoS-атак.

Автентифікація та ідентифікація користувачів – підтвердження особи клієнта через паролі, біометричні дані, токени, двофакторну (2FA) або багатофакторну (MFA) аутентифікацію. Наприклад, OTP (одноразові паролі) або апаратні ключі (U2F, FIDO2).

Контроль доступу та авторизація – надання користувачам прав доступу відповідно до їхнього рівня повноважень. Використання політик контролю доступу, таких як RBAC (рольова модель доступу) або ABAC (атрибутна

модель доступу), допомагає запобігти несанкціонованому використанню даних.

Захист під час передавання даних – шифрування інформації при її передачі між клієнтом і сервером за допомогою протоколів HTTPS, TLS 1.3 або VPN.

Моніторинг та аудит безпеки – постійний аналіз трафіку, поведінки користувачів і виявлення аномальних активностей. Для цього використовують SIEM-системи (Security Information and Event Management), що дозволяють швидко реагувати на потенційні загрози.

Захист від шкідливого програмного забезпечення – використання антивірусного захисту, фаєрволів (WAF), механізмів виявлення аномальної поведінки (IDS/IPS) та регулярне оновлення ПЗ для усунення вразливостей.

Освіта та обізнаність користувачів – підвищення рівня знань клієнтів про потенційні загрози, методи фішингу, соціальної інженерії та безпечного користування онлайн-банкінгом. Регулярне навчання співробітників банку також є невід'ємною складовою системи безпеки.

Резервне копіювання та план відновлення – створення резервних копій даних і розробка стратегій відновлення роботи у випадку кібератаки або технічного збою.

Дотримання цих принципів дозволяє значно знизити ризики компрометації персональних даних і фінансових операцій у системах онлайн-банкінгу. У подальших розділах буде розглянуто конкретні методи атак, вразливості та заходи для їх усунення.

1.2. Нормативно-правова база захисту персональних даних

Захист персональних даних є ключовим аспектом кібербезпеки, особливо в онлайн-банкінгу, де обробляються конфіденційні фінансові відомості клієнтів. Законодавче регулювання цього питання відбувається на міжнародному та національному рівнях, що дозволяє забезпечити

відповідність фінансових установ стандартам безпеки та захисту прав користувачів.

Міжнародні нормативно-правові акти

1. Загальний регламент захисту даних (GDPR)

General Data Protection Regulation (GDPR) – основний документ Європейського Союзу щодо захисту персональних даних, чинний із 2018 року. Він регламентує збирання, обробку, збереження та передачу особистої інформації користувачів.

Стаття 1 GDPR. Предмет і цілі

1.Цей Регламент установлює норми щодо захисту фізичних осіб у зв'язку з опрацюванням персональних даних і норми про вільний рух персональних даних.

2.Цей Регламент захищає фундаментальні права і свободи фізичних осіб, зокрема їхнє право на захист персональних даних.

3.Вільний рух персональних даних у всьому Союзі не повинно бути обмежено чи заборонено із причин, пов'язаних із захистом фізичних осіб у зв'язку з опрацюванням персональних даних [2].

Основні принципи GDPR:

- Прозорість обробки даних – користувачі мають право знати, як використовуються їхні дані.
- Мінімізація збору даних – обробляються лише ті дані, які необхідні для надання послуг.
- Шифрування та псевдонімізація – для зменшення ризиків витоку інформації.
- Право на видалення даних ("право бути забутим").
- Обов'язкове повідомлення про витік даних – організації зобов'язані інформувати контролюючі органи та постраждалих клієнтів у разі порушення безпеки.

2. Стандарти безпеки даних платіжних карток (PCI DSS)

Стандарт безпеки даних індустрії платіжних карток (PCI DSS) — це загальноприйнятий набір політик і процедур, призначених для оптимізацій безпек та операцій з кредитними, дебетовими та готівковими картками та захисту власників карток від неправомірного використання їх особистої інформації. PCI DSS розроблено для запобігання порушенням кібербезпеки конфіденційних даних і зниження ризику шахрайства для організацій, які обробляють інформацію про платіжні картки.

Рисунок 1.1 - Вимоги PCI DSS

Основною метою PCI DSS є захист та оптимізація безпеки конфіденційних даних власників карток, як-от номери кредитних карток, терміни дії та коди безпеки. Засоби безпеки стандарту допомагають підприємствам мінімізувати ризики витоку даних, шахрайства та крадіжки особистих даних (рис. 1.1).

Відповідність стандартам PCI DSS також гарантує, що компанії дотримуються найкращих галузевих практик під час обробки, зберігання та передачі даних кредитних карток. Так само відповідність вимогам PCI DSS зміцнює довіру між клієнтами та зацікавленими сторонами.

Рада зі стандартів безпеки PCI (PCI SSC) створила шість основних цілей для PCI DSS:

1. Створюйте та підтримуйте безпечну мережу та системи. Операції з кредитною карткою мають здійснюватися в захищеній мережі.

2. Захистіть дані власника картки. Організації, які дотримуються PCI DSS, повинні захищати інформацію про власника картки, де б вона не зберігалася.

3. Підтримуйте програму керування вразливістю. Організації, що обслуговують картки, повинні запровадити програми оцінки ризиків і управління вразливістю, які захищають їхні системи від дій зловмисних хакерів, таких як шпигунське та шкідливе ПЗ.

4. Впровадити жорсткі заходи контролю доступу. Доступ до системної інформації та операцій повинен бути обмежений і контрольований.

5. Регулярно перевіряйте та тестуйте мережі. Мережі необхідно регулярно контролювати та тестувати, щоб переконатися, що заходи безпеки діють, функціонують належним чином і оновлюються.

6. Підтримувати політику інформаційної безпеки. Офіційна політика інформаційної безпеки повинна бути визначена, підтримувана та дотримана всіма учасниками [3].

В Україні нормативно-правовою базою захисту персональних даних є ряд законів та Постанов НБУ.

1. Закон України "Про захист персональних даних"

Цей Закон регулює правові відносини, пов’язані із захистом і обробкою персональних даних, і спрямований на захист основоположних прав і свобод людини і громадянина, зокрема права на невтручання в особисте життя, у зв’язку з обробкою персональних даних.

Цей Закон поширюється на діяльність з обробки персональних даних, яка здійснюється повністю або частково із застосуванням автоматизованих засобів, а також на обробку персональних даних, що містяться у картотеці чи призначені до внесення до картотеки, із застосуванням неавтоматизованих засобів.

Стаття 3. Законодавство про захист персональних даних

Законодавство про захист персональних даних складають Конституція України, цей Закон, інші закони та підзаконні нормативно-правові акти, міжнародні договори України, згода на обов'язковість яких надана Верховною Радою України.

Стаття 22. Контроль за додержанням законодавства про захист персональних даних

1. Контроль за додержанням законодавства про захист персональних даних у межах повноважень, передбачених законом, здійснюють такі органи:

1) Уповноважений;

2) суди [4].

2. ЗАКОН УКРАЇНИ «Про електронну ідентифікацію та електронні довірчі послуги»

Цей Закон визначає правові та організаційні засади електронної ідентифікації та надання електронних довірчих послуг, права та обов’язки суб’єктів відносин у сферах електронної ідентифікації та електронних довірчих послуг, порядок здійснення державного контролю за дотриманням вимог законодавства у сферах електронної ідентифікації та електронних довірчих послуг.

Стаття 2. Сфера дії Закону

1.Цей Закон регулює відносини, що виникають між юридичними, фізичними особами, суб’єктами владних повноважень у процесі надання і отримання послуг електронної ідентифікації, гаманців з цифровою ідентифікацією та електронних довірчих послуг, процедури надання таких

послуг, нагляду і контролю за дотриманням вимог законодавства у сферах електронної ідентифікації та електронних довірчих послуг.

Стаття 3. Законодавство у сферах електронної ідентифікації та електронних довірчих послуг

1. Відносини, пов’язані з електронною ідентифікацією та наданням електронних довірчих послуг, регулюються Конституцією України, Цивільним кодексом України, законами України "Про інформацію", "Про захист інформації в інформаційно-комунікаційних системах", "Про електронні документи та електронний документообіг", "Про захист персональних даних", цим Законом, а також іншими нормативно-правовими актами.

Стаття 5. Система органів, що здійснюють державне регулювання у сферах електронної ідентифікації та електронних довірчих послуг

1. Державне регулювання у сферах електронної ідентифікації та електронних довірчих послуг здійснюють:

Кабінет Міністрів України;

центральний орган виконавчої влади, що забезпечує формування та реалізує державну політику у сферах електронної ідентифікації та електронних довірчих послуг;

спеціально уповноважений центральний орган виконавчої влади з питань організації спеціального зв’язку та захисту інформації;

Національний банк України [5].

3. Закон України "Про платіжні послуги"

Цей Закон визначає поняття та загальний порядок виконання платіжних операцій в Україні, встановлює виключний перелік платіжних послуг та порядок їх надання, категорії надавачів платіжних послуг та умови авторизації їх діяльності, визначає загальні засади функціонування платіжних систем в Україні, загальні засади випуску та використання в Україні електронних грошей та цифрових грошей Національного банку України, установлює права, обов’язки та відповідальність учасників платіжного ринку України, визначає загальний порядок здійснення нагляду за діяльністю надавачів платіжних

послуг, надавачів обмежених платіжних послуг, порядок здійснення оверсайта платіжної інфраструктури.

Стаття 2. Законодавство, що регулює діяльність на платіжному ринку

1. Загальні засади функціонування платіжного ринку, відносини у сфері надання платіжних послуг користувачам регулюються Конституцією України, цим Законом, іншими законами України та прийнятими на їх основі нормативно-правовими актами.

2. Під час надання споживачам послуг на платіжному ринку надавачі платіжних послуг зобов’язані дотримуватися вимог законів України "Про фінансові послуги та фінансові компанії", "Про захист прав споживачів" та інших нормативно-правових актів у сфері захисту прав споживачів [6].

Цей закон, що набув чинності в 2022 році, імплементує європейські стандарти безпеки платежів, зокрема Директиву ЄС PSD2. Основні положення:

Введення відкритого банкінгу (Open Banking).

Обов’язкове використання багатофакторної автентифікації для платежів.

Посилений контроль за фінансовими операціями для протидії шахрайству.

4. Постанова НБУ №95 "Про організацію заходів із забезпечення інформаційної безпеки в банківській системі України"

Це Положення встановлює:

1) обов’язкові мінімальні вимоги щодо організації заходів із забезпечення інформаційної безпеки та кіберзахисту;

2) принципи управління інформаційною безпекою;

3) вимоги до інформаційних систем банку, що взаємодіють з інформаційними системами Національного банку України (далі – Національний банк), з урахуванням напрямів розвитку криптографічного захисту інформації в інформаційних системах Національного банку [7].

Національний банк України встановлює вимоги до банків щодо кібербезпеки та захисту персональних даних, зокрема:

Обов’язкове впровадження сучасних методів шифрування.

Використання систем моніторингу підозрілої активності.

Регулярне проведення аудиту та тестування на проникнення.

5. Закон України "Про основні засади забезпечення кібербезпеки України"

Цей Закон визначає правові та організаційні основи забезпечення захисту життєво важливих інтересів людини і громадянина, суспільства та держави, національних інтересів України у кіберпросторі, основні цілі, напрями та принципи державної політики у сфері кібербезпеки, повноваження державних органів, підприємств, установ, організацій, осіб та громадян у цій сфері, основні засади координації їхньої діяльності із забезпечення кібербезпеки [8].

В Україні контроль за виконанням законів щодо захисту персональних даних здійснюють такі органи:

Уповноважений Верховної Ради України з прав людини – слідкує за дотриманням законодавства у сфері персональних даних.

Національний банк України (НБУ) – регулює безпеку фінансових установ, зокрема у сфері онлайн-банкінгу.

Кіберполіція України – займається розслідуванням порушень у сфері захисту інформації та персональних даних.

Залежно від порушення передбачено адміністративну та кримінальну відповідальність. Наприклад: штрафи за незаконне збирання, зберігання чи поширення персональних даних; кримінальна відповідальність за несанкціонований доступ до баз даних (ст. 361 Кримінального кодексу України).

Нормативно-правова база захисту персональних даних в онлайн-банкінгу в Україні розвивається відповідно до міжнародних стандартів. Впровадження норм GDPR, PSD2, PCI DSS у національне законодавство сприяє підвищенню рівня безпеки в банківському секторі. Банки зобов’язані використовувати сучасні технології кіберзахисту, шифрування та

багатофакторної автентифікації, а також дотримуватися вимог українських законів для захисту персональних даних клієнтів.

2. АНАЛІЗ ЗАГРОЗ ТА ВРАЗЛИВОСТЕЙ В ОНЛАЙН-БАНКІНГУ

2.1. Методи атак на системи онлайн-банкінгу

З розвитком цифрових технологій системи онлайн-банкінгу стають мішенню для різноманітних кібератак. Нападники використовують соціальну інженерію, вразливості мережевих протоколів, шкідливе ПЗ та методи перебору паролів, щоб отримати доступ до фінансових даних користувачів.

Фішинг – один із найпоширеніших методів атак, заснований на соціальній інженерії. Атакуючі обманом змушують жертв ввести конфіденційні дані (логіни, паролі, номери карток) на підроблених сайтах або у фальшивих мобільних застосунках. У таблиці 1 наведено основні види фішингових атак, а також їх опис та приклад.

Таблиця 1 - Основні види фішингових атак

Вид фішингу	Опис	Приклад
Класичний фішинг	Користувач отримує підроблений лист від "банку" з проханням оновити дані.	Лист із посиланням на сайт, схожий на справжній банківський портал.
Смшинг (SMS-фішинг)	Атака через SMS з фальшивим посиланням або проханням повідомити код з картки.	"Ваша картка заблокована. Для розблокування введіть код за цим посиланням."
Вішинг (голосовий фішинг)	Шахраї телефонують, представляючись співробітниками банку.	Дзвінок: "Вашу картку зламали, повідомте код підтвердження для безпеки."

Наприклад, банківський сайт realbank.com може бути підроблений як realbank-security.com, де жертва вводить свої дані, які одразу потрапляють до хакерів.

Схема фішингової атаки (рис.2.1):

1. Хакер створює сайт-клон банку.
2. Користувач отримує фальшивий email/SMS із посиланням.
3. Вводить свої дані на підробленому сайті.
4. Дані потрапляють до зловмисника, який використовує їх для викрадення коштів.

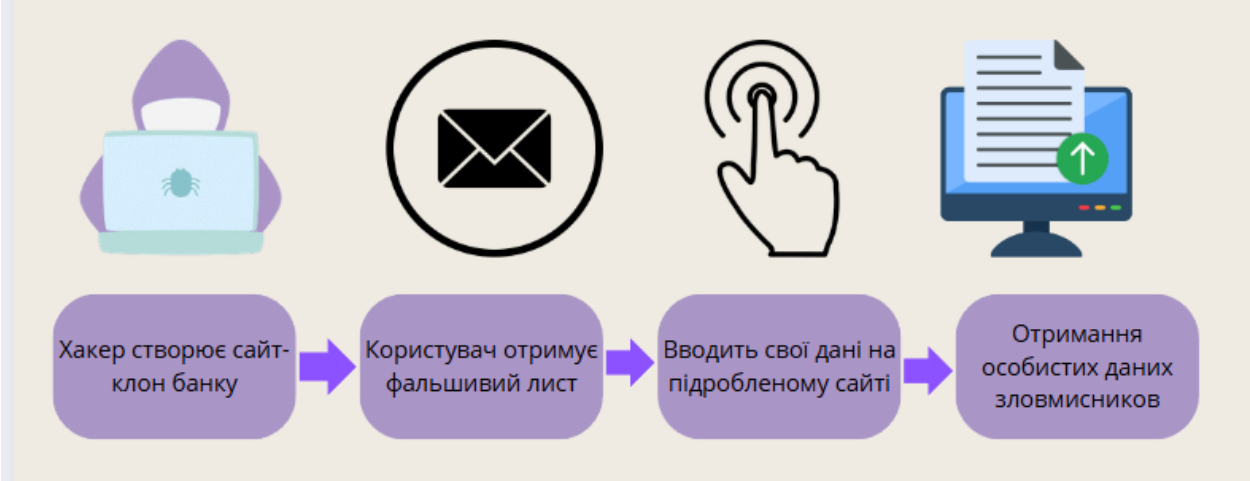

Рисунок 2.1 - Схема фішингової атаки

MITM (Man-in-the-Middle) – атака, при якій зловмисник перехоплює та змінює дані між клієнтом і банком, не викликаючи підозри. У таблиці 2 наведено основні атаки типу «людина посередині».

Таблиця 2 - Основні методи MITM

Вид атаки	Опис	Приклад
Підміна HTTPS (SSL Stripping)	Хакер змушує користувача підключитися через HTTP замість HTTPS.	Користувач бачить "http://bank.com" замість "https://bank.com".
Атака на публічний Wi-Fi	Хакер створює фальшиву точку Wi-Fi з назвою "Free Airport Wi-Fi" і перехоплює трафік.	Користувач вводить пароль від банку, а зловмисник його читає.

Приклад атаки MITM через Wi-Fi:

1. Користувач підключається до безкоштовного Wi-Fi.
2. Хакер перенаправляє його трафік через свій сервер.
3. Всі дані (логіни, паролі) передаються через хакера.
4. Зловмисник отримує доступ до банківського рахунку.

Шкідливе програмне забезпечення (малваре) встановлюється на пристрій користувача та викрадає конфіденційні дані. Існують декілька видів шкідливого програмного забезпечення, а саме: кейлогер, троян, та банківський троян(таблиця 3).

Таблиця 3 - Основні види шкідливого ПЗ

Вид	Опис	Приклад атаки
Кейлогер	Записує всі натискання клавіш.	Користувач вводить пароль, і він надсилається хакеру.
Троян (Trojan)	Маскується під корисний додаток, але викрадає дані.	Фальшивий банківський застосунок.
Банківський троян	Відстежує банківські транзакції та перенаправляє кошти хакеру.	Zeus, Dridex, Emotet.

Схема атаки з кейлогером:

- Користувач завантажує заражену програму.
- Вірус записує всі натискання клавіш.
- Дані (логіни, паролі) надсилаються хакеру.

Слабкі паролі легко вгадати або підібрати, тому хакери використовують спеціальні атаки, спрямовані на підбір паролей(таблиця 4)

Таблиця 4 - Основні методи атак

Метод	Опис	Приклад
Брутфорс (Brute-force)	Повний перебір можливих паролів.	"123456" ламається за секунди.
Атака словником	Використання списку популярних паролів.	Перевірка "qwerty", "password".
Атака з викраденими базами даних	Використання злитих паролів з інших сервісів.	Перевірка пароля з бази зламаних акаунтів.

Як захиститися:

- Використовувати довгі та унікальні паролі.

- Використовувати менеджери паролів.
- Увімкнути двофакторну автентифікацію (2FA).

Банківські системи використовують різні мережеві протоколи для передачі даних, і хакери атакують їхні слабкі місця.

Таблиця 5 - Основні атаки

Атака	Вразливий протокол	Опис
Атака на TLS (TLS Downgrade)	TLS 1.0, SSL 3.0	Примушує сервер використовувати застаріле шифрування.
DNS Spoofing	DNS	Підробка записів DNS, щоб спрямувати користувачів на підроблений сайт.
ARP Spoofing	ARP	Підміна MAC-адрес для перехоплення трафіку.

Приклад атаки DNS Spoofing:

1. Користувач вводить "bank.com".
2. Хакер підміняє DNS і спрямовує його на фальшивий сайт.
3. Користувач вводить логін та пароль.
4. Дані надсилаються хакеру.

Захист від атак на протоколи:

- Використання нових версій протоколів (TLS 1.3).
- Використання DNSSEC для захисту DNS-запитів.
- Використання VPN для шифрування з’єднання.

Онлайн-банкінг є мішенню для різних атак: від соціальної інженерії (фішинг) до складних технічних атак (MITM, шкідливе ПЗ, атаки на протоколи). Найефективніший захист – комплексний підхід, що включає

використання надійного шифрування, багатофакторної автентифікації, оновлення ПЗ та підвищення обізнаності користувачів.

2.2. Вразливості веб-сайтів фінансових установ

Безпека веб-сайтів фінансових установ є критично важливою, оскільки їх злам може призвести до крадіжки персональних даних, фінансових втрат і репутаційних ризиків. Основні вразливості включають неправильні конфігурації серверів, ін’єкційні атаки та витік даних через сторонні сервіси.

Відкриті порти можуть бути використані хакерами для атак на сервер. Деякі сервіси, якщо вони не захищені належним чином, можуть дозволяти несанкціонований доступ (таблиця 6).

Таблиця 6 - Основні загрози відкритих портів

Порт	Протокол	Потенційна загроза
21	FTP	Можливість підбору паролів та крадіжки файлів.
22	SSH	Брутфорс-атаки на доступ адміністратора.
25	SMTP	Використання для спаму або фішингу.
80	HTTP	Відсутність шифрування трафіку (небезпека MITM).
3306	MySQL	Атаки на базу даних через слабкі паролі.

Приклад атаки на відкритий порт:

1. Хакер сканує відкриті порти за допомогою nmap:

```
css
КопироватьРедактировать
nmap -p 21,22,80,443,3306 example.com
```

2. Виявляє незахищений MySQL-сервер на порту 3306.
3. Використовує стандартний логін/пароль для підключення.
4. Отримує доступ до банківської бази даних.

Захист:

- Закрити непотрібні порти в файрволі.

- Використовувати VPN або обмежити доступ за IP.
- Регулярно оновлювати серверне програмне забезпечення.

SQL-ін’єкції (SQLi) дозволяють хакеру виконати несанкціоновані запити до бази даних.

Приклад SQL-ін’єкції:

Користувач вводить у поле логіну замість імені наступний рядок:

```
sql
КопироватьРедактировать
' OR '1'='1' --
```

SQL-запит виглядає так:

```
sql
КопироватьРедактировать
SELECT * FROM users WHERE username='' OR '1'='1' --' AND password='xxx';
```

Це призводить до того, що база даних повертає всі записи користувачів, даючи хакеру доступ до всіх акаунтів.

Захист:

- Використовувати підготовлені запити (Prepared Statements).
- Валідувати вхідні дані.
- Вимкнути відображення помилок бази даних.

XSS (Cross-Site Scripting) дозволяє вставляти шкідливий JavaScript-код у веб-сторінки.

Приклад XSS: Користувач вводить у форму такий код:

```
html
КопироватьРедактировать
<script>alert('Ваш пароль зламано!');</script>
```

Якщо сайт не фільтрує введені дані, цей код буде виконано у браузері жертви.

Захист:

- Використовувати фільтрацію та екранування символів (<, >, ").
- Встановлювати Content Security Policy (CSP).
- Забороняти введення небезпечного HTML-коду.

Фінансові установи використовують сторонні сервіси (аналітика, чати, рекламу), які можуть призводити до витоку даних. У таблиці 7 наведено потенційні проблеми відповідно до джерел загроз.

Таблиця 7 - Приклади загроз

Джерело загрози	Потенційна проблема
Сторонні скрипти (Google Analytics, Facebook Pixel)	Можливий витік даних клієнтів через API.
Плагіни та бібліотеки	Використання застарілих бібліотек з вразливостями.
Небезпечні API-запити	Недостатній контроль доступу до API.

Приклад витоку через API:

1. Вебсайт має відкритий API-запит:

```
bash
КопироватьРедактировать
https://bank.com/api/getUserData?id=123
```

2. Хакер змінює `id=123` на `id=124` та отримує чужі дані.
3. Дані клієнтів витікають у відкритий доступ.

Захист:

- Обмежити доступ до API через токени авторизації.
- Використовувати контроль прав доступу (RBAC, OAuth).
- Перевіряти безпеку сторонніх бібліотек.

Основні вразливості веб-сайтів фінансових установ включають відкриті порти, неправильні конфігурації серверів, ін’єкційні атаки та витік даних через сторонні сервіси. Захист від цих загроз передбачає комплексні заходи: правильне налаштування серверів, використання надійних механізмів аутентифікації та перевірку всіх вхідних даних на сайті.

2.3. Аналіз фінансової безпеки сайтів банків та маркетплейсу

Онлайн-банкінг став необхідною складовою сучасного фінансового світу, що надає користувачам зручність та доступність у керуванні своїми фінансами. Серед численних платформ онлайн-банкінгу, Приват24, як продукт ПриватБанку, відіграє значну роль на ринку фінансових послуг в Україні.

На сайті є вкладки «Про персональні дані» (http://surl.li/srpnk) та «безпека» (http://surl.li/srpnz).

На вкладці «Про персональні дані» викладені правила та регламент збирання та обробки персональних даних. Там користувач може ознайомитись з усіма політиками безпеки збирання та передавання персональних даних. У процесі надання послуг та проведення господарської діяльності банком обробляються різні категорії персональних даних. Дані можуть отримуватися в процесі встановлення та реалізації ділових відносин із Банком, проведення платіжних та інших фінансових операцій, проведення фінансового моніторингу та з інших джерел.

Одним із ключових аспектів безпеки є наявність чіткої політики конфіденційності. Приват24 має ретельно розроблену політику, яка визначає правила збору, використання та захисту персональних даних користувачів. Приват24 застосовує сучасні методи шифрування трафіку за допомогою протоколу HTTPS, що забезпечує конфіденційність інформації, що передається між користувачем та сервером. Платформа має ефективні механізми управління доступом та аутентифікації, включаючи введення пароля, використання одноразових кодів та біометричних даних для підтвердження особи користувача. Приват24 використовує системи моніторингу для виявлення підозрілої активності та потенційних загроз безпеці. Швидке виявлення та реагування на порушення допомагає забезпечити безпеку та надійність платформи. Приват24 готовий швидко сповістити користувачів про будь-які порушення безпеки та надати відповідні

рекомендації щодо подальших дій, співпрацюючи з ними у вирішенні проблем.

Розглянемо зміст вкладки «безпека»(рис.2.2).

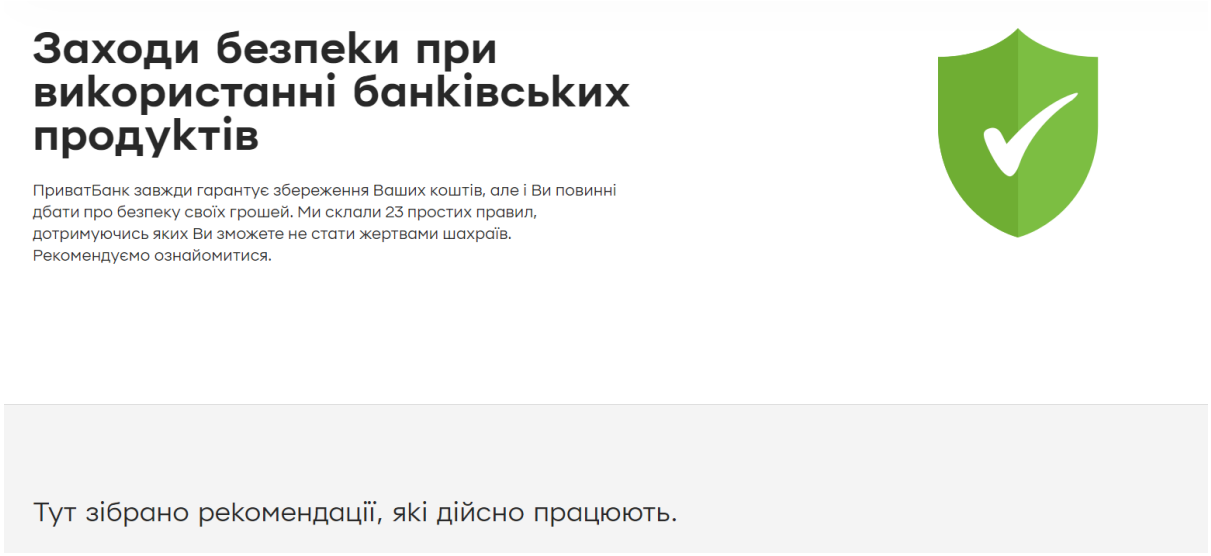

Рисунок 2.2 - Вкладка «Безпека»

У даному розділі зібрано рекомендації щодо:

- Методів протидії шахраям;
- Заходів безпеки під час користування смартфоном;
- Безпеки під час купівель в інтернеті;
- Заходів безпеки під час користування банківськими картками;
- Страхування від шахраїв.

ПриватБанк активно бореться із шахрайством, заохочуючи співробітників повідомляти про махінації. Як всередині банку, так і за його межами банк дотримується політики нульової терпимості до шахрайства та використовує весь арсенал заходів щодо попередження крадіжок коштів клієнтів. Користувач може допомогти банку, повідомляючи про випадки шахрайства.

Також, на сайті розташовано Положення "Про запобігання шахрайству" (http://surl.li/srpou).

Проведено сканування пакетів, при вході до облікового запису privat24(рис.2.3).

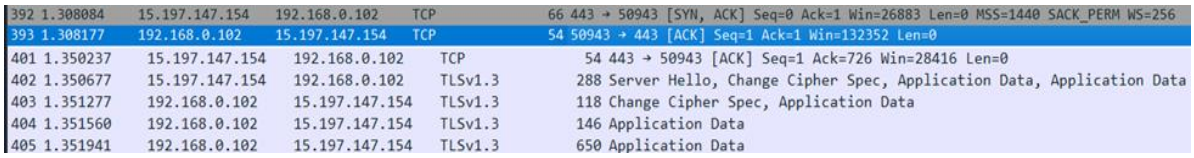

Рисунок 2.3 - Перехоплені пакети у середовищі wireshark

При аналізі пакетів з’ясовано, що:

IP-адреса, на яку надсилається запит та з якої приходить відповідь: 15.197.147.154

IP-адреса пристрою, з якого проведено сканування: 192.168.0.102

Source port: 15.197.147.154/192.168.0.102 (в залежності від виду запит або відповідь)

Destination port: 15.197.147.154/192.168.0.102 (в залежності від виду запит або відповідь)

Protocol: TCP, TLSv1.3

Port сервера: 443

Port користувача: 50943

Рисунок 2.4 - Частина змісту перехопленого пакету

З’єднання захищено за протоколом TLSv1.3. TLSv1.3 - це остання версія протоколу TLS (Transport Layer Security), який використовується для забезпечення безпечного обміну даними в інтернеті. Порт 443(див. рис. 2.4) вказує на те, що з’єднання здійснюється за протоколом HTTPS, що є безпечним. Текст запиту та відповіді є зашифрованими. Протокол TCP один з основних протоколів мережевого рівня (рівень транспорту) в моделі OSI (Open

Systems Interconnection). Він забезпечує надійний, упорядкований та стриманий з'єдноконтроль передачі даних між пристроями у мережі.

Було виконано дослідження процесу авторизації web-банкінгу Privat24. У результаті проведеного аналізу можна зробити висновок, що платформа Приват24 відповідає високим стандартам безпеки та захисту персональних даних, забезпечуючи своїм користувачам безпечне та надійне середовище для здійснення фінансових операцій онлайн.

Для дослідження було виконано авторизацію на сайті https://next.privat24.ua/. Паралельно з цим виконали захват пакетів використовуючи програмне забезпечення Wireshark. Авторизацію на сайті було виконано двома різними способами. За допомогою пароля та QR-коду. Прослідковано як саме відбувається обмін ключами з сервером. Загальний алгоритм наступний: клієнт (ваш браузер) надсилає запит на веб-сайт за допомогою HTTPS. Сервер відповідає, представляючи свій сертифікат безпеки, який містить його публічний ключ. Після чого клієнт перевіряє цей сертифікат і, якщо все в порядку, генерує секретний ключ, який буде використовуватися для шифрування даних. Далі клієнт шифрує цей секретний ключ за допомогою публічного ключа сервера та надсилає його серверу. Після чого сервер дешифрує отриманий секретний ключ за допомогою свого приватного ключа. Тепер як браузер так і сервер можуть використовувати цей секретний ключ для шифрування та розшифрування даних, якими обмінюються під час сесії. З аналізу перехоплених пакетів зроблено висновки.

1. Клієнт відправляє свій відкритий ключ, що можна побачити у пакеті 6214, зміст якого наведено на рисунку 2.5.

Відкритий ключ клієнта:
78d57bb3b95b10eae82bf1d55fe02d438bd3924105aefc3c8bb5e78273872d46

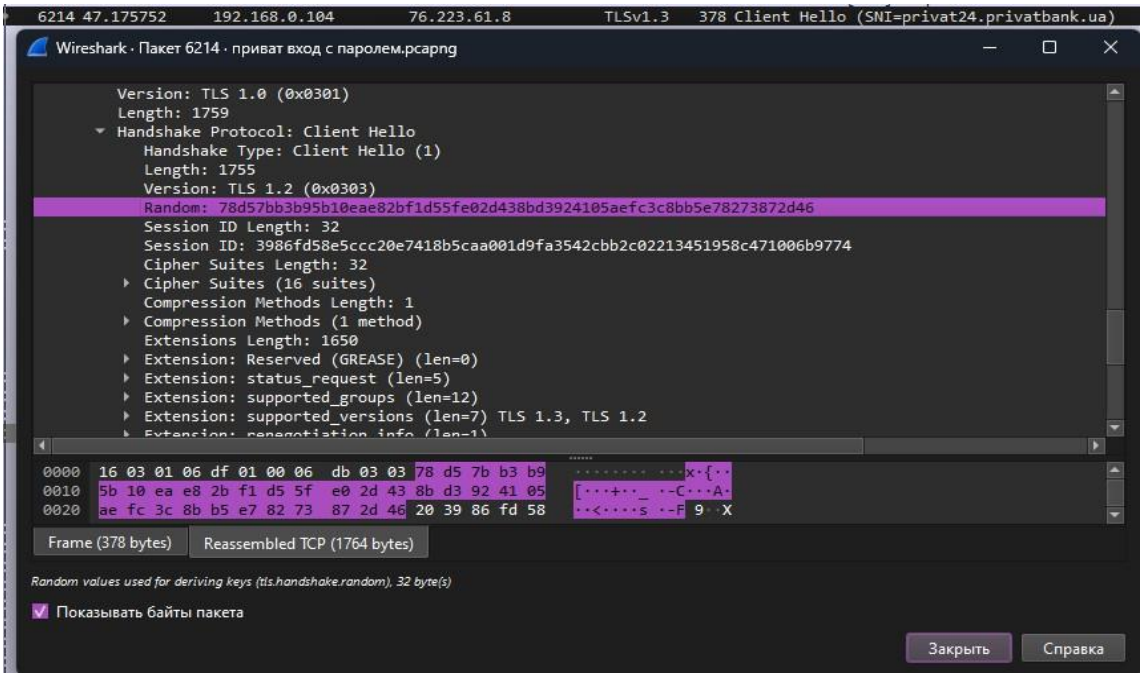

Рисунок 2.5 - Зміст пакету 6214

Далі на сервері відбувається обмін відкритими ключами. Клієнт та сервер мають по два ключі: відкритий та закритий.

2. Відкритий ключ сервера також можна відслідкувати. Пакет, у якому був отриманий відкритий ключ від сервера має номер 6216 та його зміст наведено на рисунку 2.6.

Відкритий ключ сервера:
9892668ad8b1092997bf1ce8130486dfa2a212315c8f53f7953d5c660e3df874

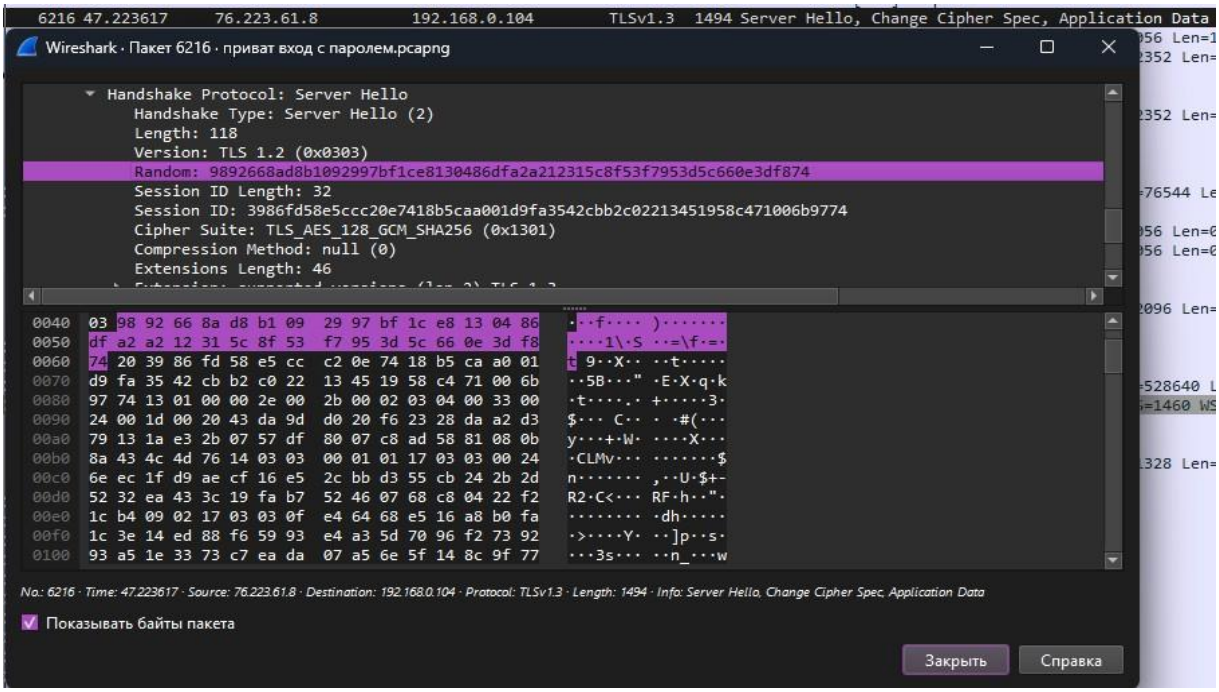

Рисунок 2.6 - Зміст пакету 6216

3. Визначити у якому саме пакеті відбувається передача пароля неможливо. Це відбувається у одному з пакетів наведених на рисунку 2.7, що виділені рожевим.

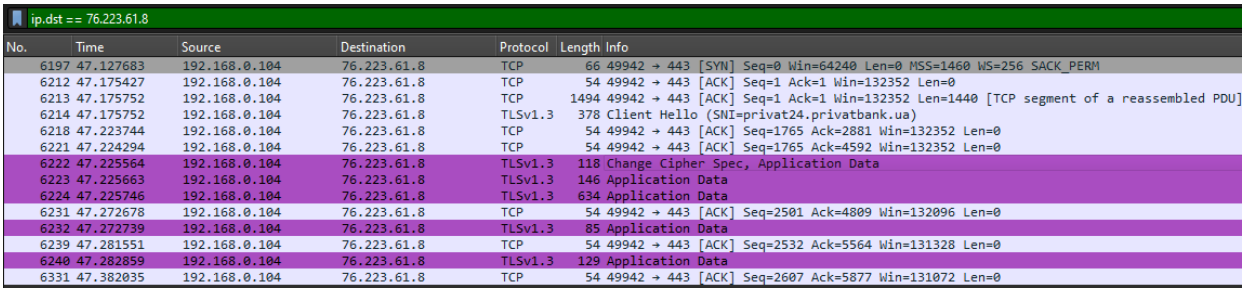

Рисунок 2.7 - Пакети у яких може вудбуватись передача пароля

Можемо припустити, що це відбувається у пакеті з найбільшим розміром(6224), проте, це лише припущення.

4. Під час авторизації за допомогою QR-коду відсутній обмін ключами, оскільки залишились ключі з попередньої сесії (так як авторизація цим методом була виконана одразу після авторизації за допомогою пароля).

5. При авторизації даним методом аналогічно неможливо визначити у якому саме пакеті відбулась передача пароля. Пароль був переданий у одному з чотирьох пакетів, що виділені рожевим на рисунку 2.8

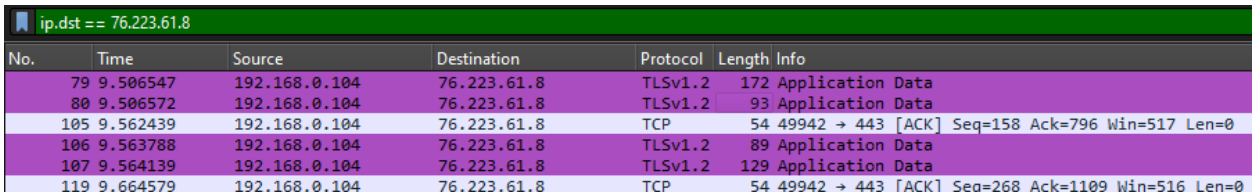

Рисунок 2.8 - Пакети у яких може вудбуватись передача пароля

Аналогічно можемо припустити, що передача пароля відбулась у пакеті з найбільшим розміром(79), проте це лише припущення.

Аналізуючи перехоплені пакети під час авторизації на сайті https://next.privat24.ua/, ми виявили деякі цікаві аспекти. Спочатку була проведена авторизація за допомогою пароля, а потім за допомогою qr-коду. Під час першої авторизації відбувався обмін відкритими ключами між клієнтом і сервером. Ми виявили відкриті ключі як клієнта, так і сервера. Проте точно визначити, у якому саме пакеті відбувається передача пароля, виявилось неможливо, хоча можна припустити, що це сталося в пакеті з найбільшим розміром.

У разі авторизації за допомогою qr-коду не відбувається обмін ключами через те, що залишаються ключі з попередньої сесії. Так само, аналогічно до авторизації за допомогою пароля, точно визначити, у якому саме пакеті відбулась передача пароля, складно, хоча можна припустити, що це також сталося в пакеті з найбільшим розміром.

Отже, аналіз перехоплених пакетів дозволяє зрозуміти процес авторизації та безпекові аспекти, проте точність визначення моменту передачі пароля залишається засекреченим.

Проте питання перехоплення паролю залишалось відкритим, тому було пророблено наступну роботу.

У середовищі linux був увімкнутий проксі-сервер.

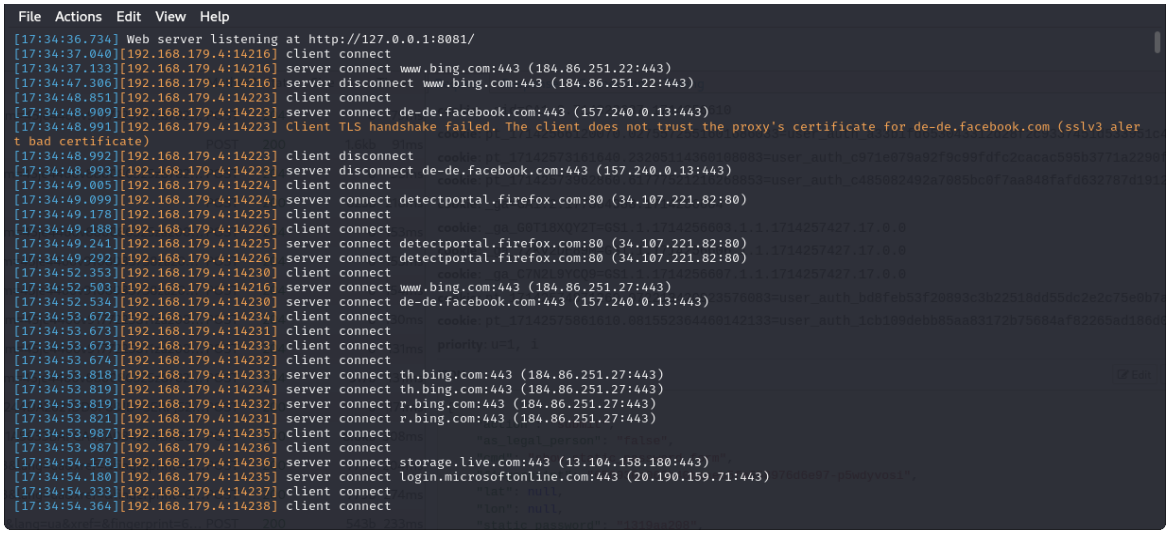

Рисунок 2.9 - Увімскнення проксі-сервера у середовищі linux

У середовищі windows приєднуємось до проксі-сервера (рис.2.10).

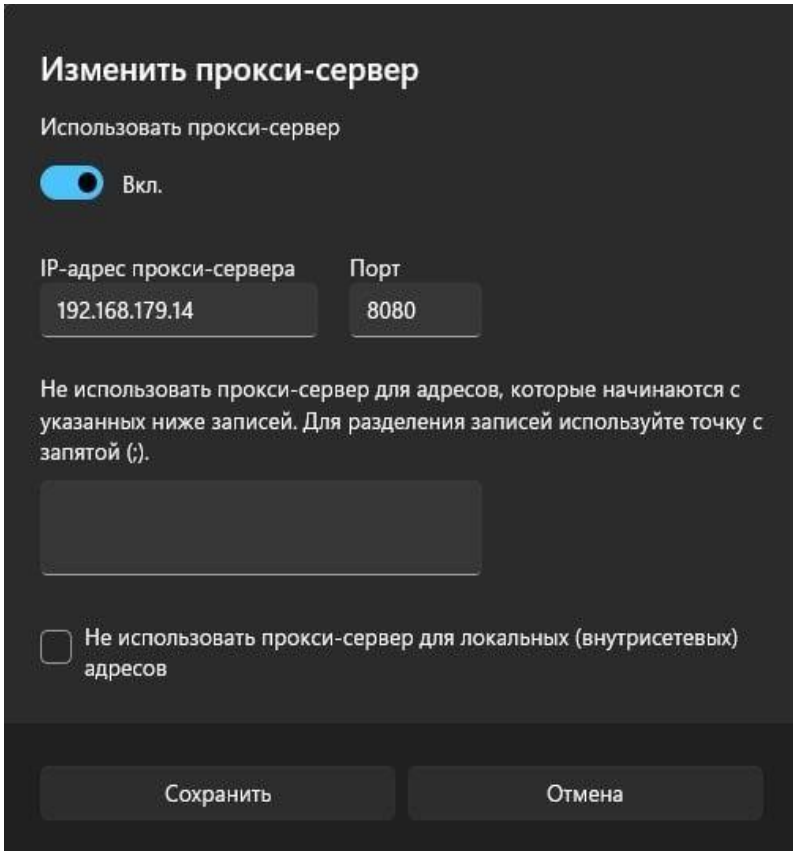

Рисунок 2.10 - Приєднання до проксі-сервера

Після підключення проксі-сервера заходимо http://mitm.it/#Windows , де завантажуємо та встановлюємо сертифікат безпеки (рис.2.11).

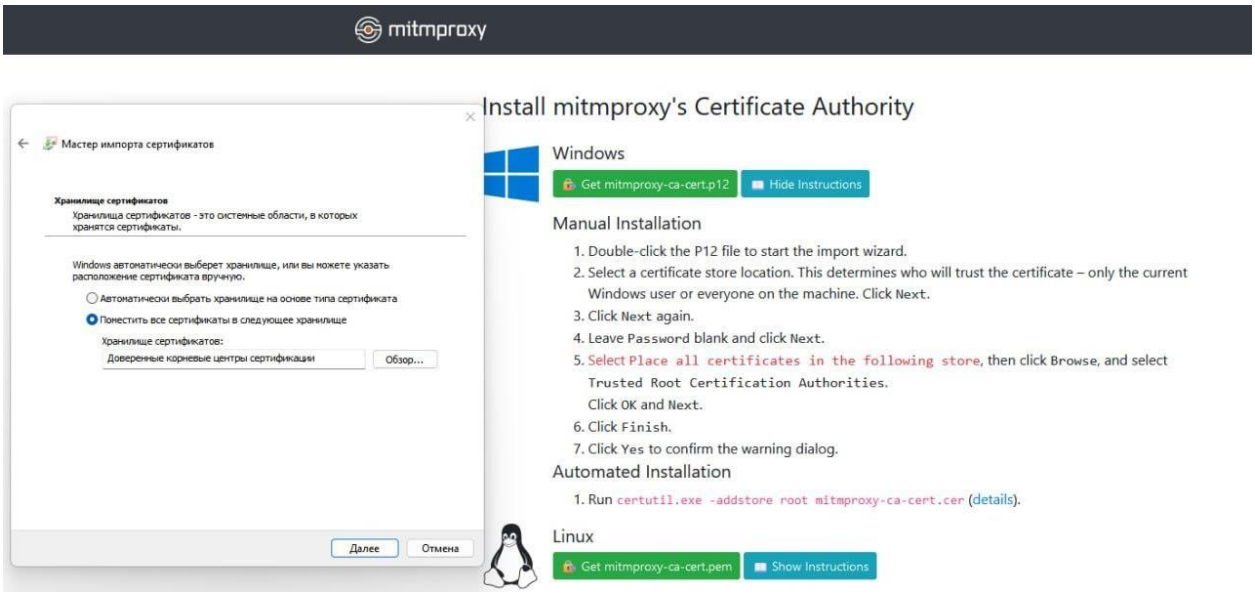

Рисунок 2.11 - Завантаження та встановлення сертифікату безпеки

Якщо перейти за посиланням не під’єднавшись, то отримаємо таке сповіщення (рис.2.12)

If you can see this, traffic is not passing through mitmproxy.

Visit the Documentation

Рисунок 2.12 - Сповіщення, отримане в результаті непід’єднання

Якщо перейти після під’єднання, то отримаємо дане вікно (рис.2.13)

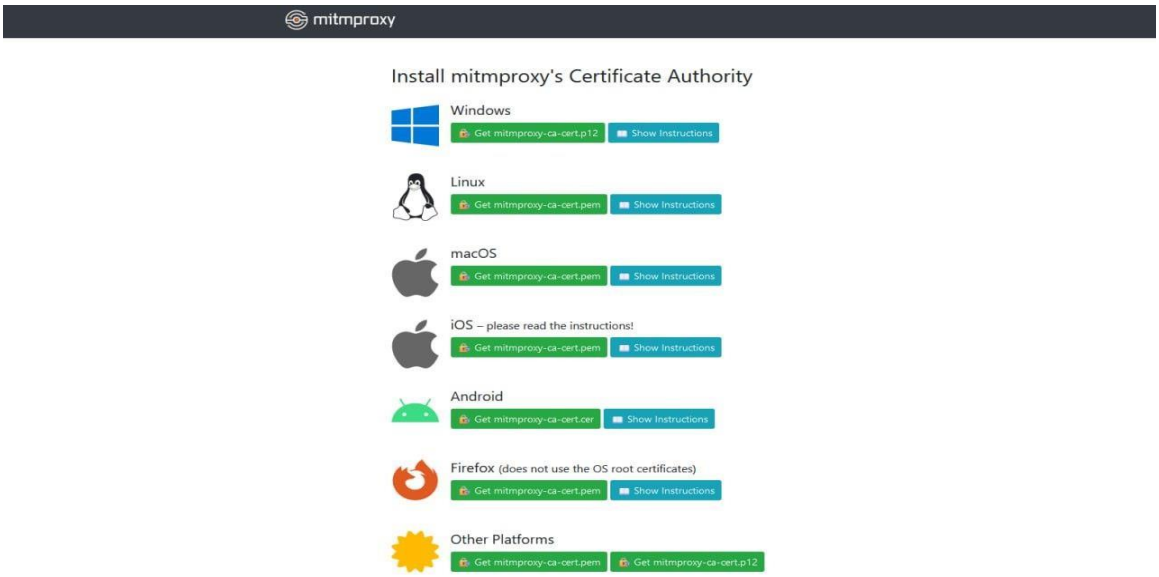

Рисунок 2.13 - Вікно, отримане у результаті успішного приєднання

Переходимо на сторінку приват24, та вводимо логін після чого шукаємо пакет, що відправляє дане повідомлення, та бачимо його зміст. У даному пакеті можна побачити логін у відкритому вигляді (рис.2.14).

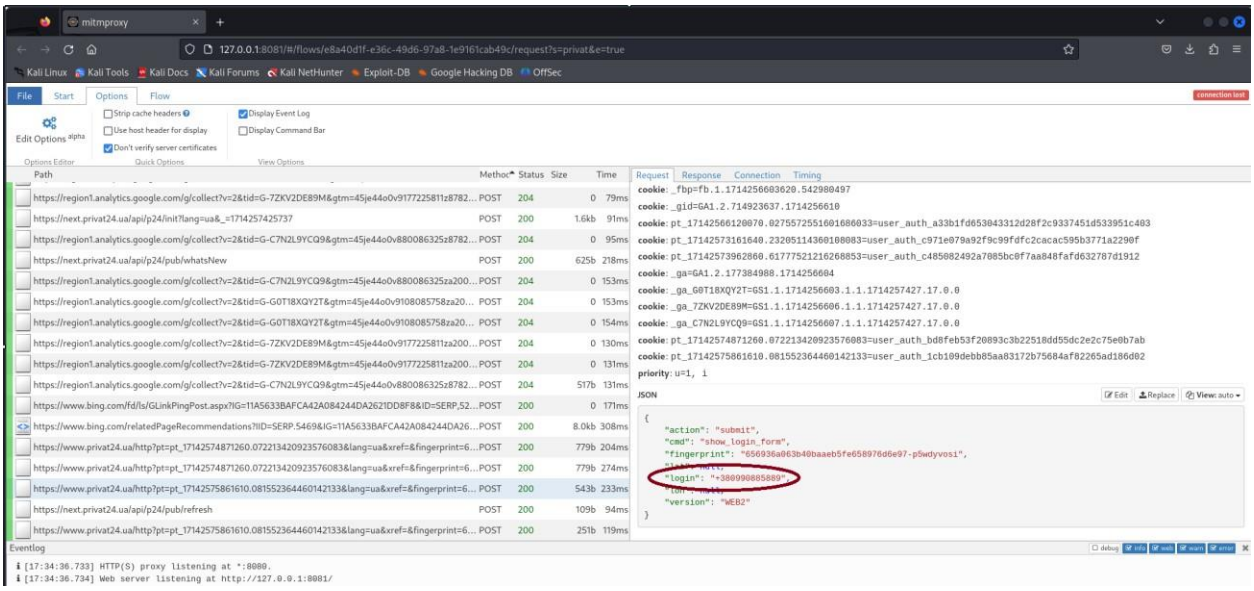

Рисунок 2.14 - Пакет, що містить логін у відкритому вигляді

Далі вводимо пароль та шукаємо відповідний пакет, у якому у відкритому вигляді також можна побачити пароль(пароль приховано через конфіденційність персональних даних) (Рис.2.15).

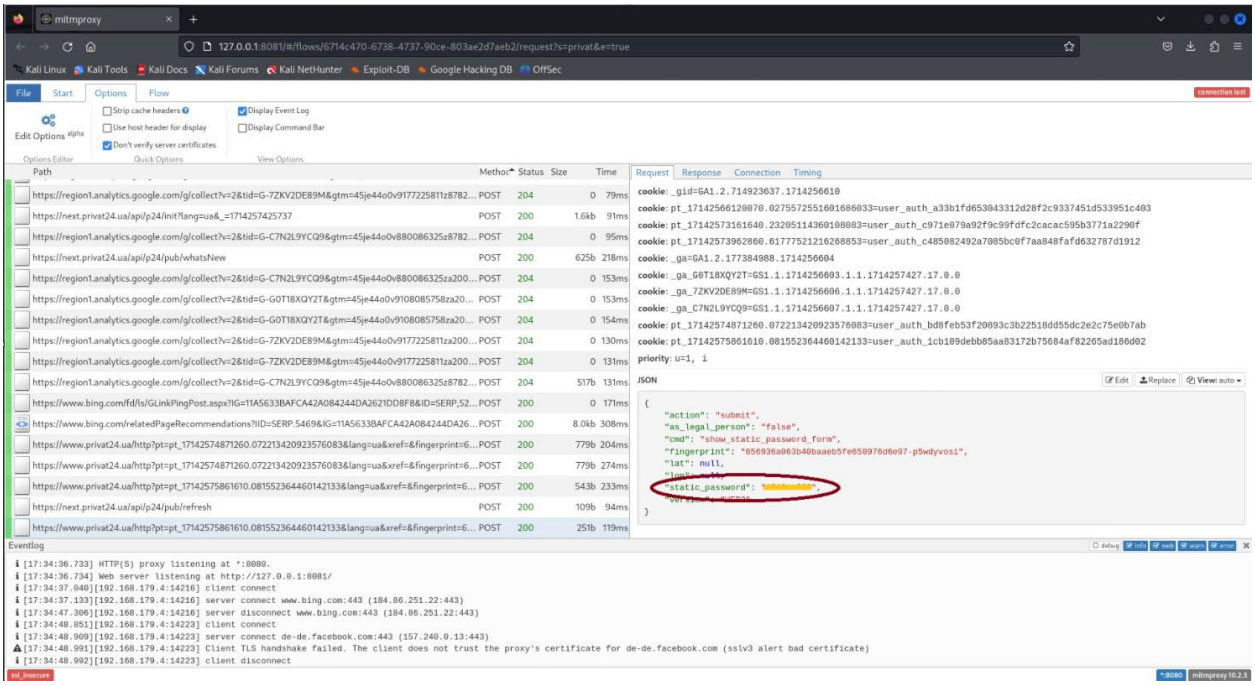

Рисунок 2.15 - Пакет, що містить пароль у відкритому вигляді

Рядок, що виділено у наступному рисунку вказує на те, що не була підтверджена дія (тому що встановлена двофакторна аутентифікація) (рис.2.16).

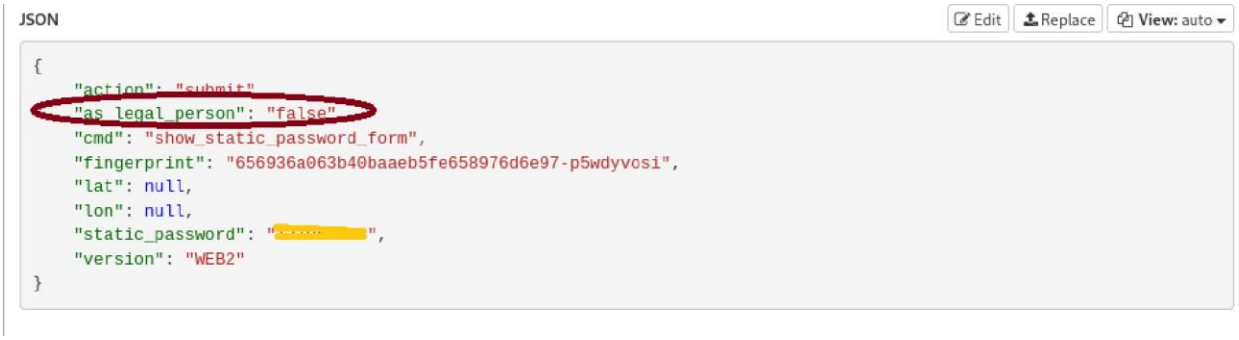

Рисунок 2.16 - Непідтвердження входу через двохфакторну автентифікацію

Дане перехоплення можливо лише якщо у браузері відсутні додаткові методи безпеки. Для отримання результату, що описано вище, був використаний MsEdge. Крім того, дане перехоплення можливо лише у разі підключення до проксі-сервера.

Захист персональних даних на фінансових веб-сайтах є критично важливим питанням у сучасному цифровому середовищі. Оскільки залежність від онлайн-платформ для фінансових транзакцій та послуг продовжує зростати, зростають і ризики, пов'язані з порушеннями даних, несанкціонованим доступом та шахрайською діяльністю.

Аналіз поширених вразливостей та методів атак підкреслив важливість усунення недоліків у безпеці веб-додатків, таких як недоліки ін'єкцій, міжсайтовий скриптинг та пошкоджена автентифікація. Також наголошується на необхідності того, щоб користувачі були пильними щодо тактик соціальної інженерії, таких як фішинг та шкідливе програмне забезпечення, які можуть поставити під загрозу їхню особисту інформацію.

Аналіз маркетплейсу “OLX.ua”. На сайті маркетплейсу розташовані Політика конфіденційності (http://surl.li/srpmp) та розділ «безпека» (https://blog.olx.ua/bezpeka-uk/). Розпорядником Бази персональних даних користувачів OLX.ua є компанія OLX Global BV, яка створена і діє за законодавством Королівства Нідерландів (ця країна є учасником Конвенції про захист осіб у зв'язку з автоматизованою обробкою персональних даних, та членом Європейського економічного простору). Персональні дані користувачів OLX.ua обробляються на захищених серверах, розташованих як у межах, так і поза межами Європейського економічного простору. Ця Політика конфіденційності описує встановлений Компанією порядок обробки персональних даних, зібраних за допомогою Сайту www.olx.ua (далі - Сайт), мобільного додатку OLX (далі - мобільний додаток) і пов'язаних з ними послуг і інструментів, що дозволяють користувачам реєструватися на Сайті, публікувати або переглядати в режимі реального часу вже опубліковані оголошення, використовувати інший, пов'язаний з вищенаведеним сервіс OLX.

У всіх зазначених випадках Компанія обробляє персональні дані користувачів виключно в рамках вимог Закону України «Про захист персональних даних» та Конвенції про захист осіб у зв'язку з автоматизованою обробкою персональних даних. Політика конфіденційності розроблена

відповідно до положень цих документів. Компанія не збирає і не обробляє персональні дані про расове або етнічне походження, політичні, релігійні або світоглядні переконання, членство в політичних партіях та професійних спілках та подібну інформацію. Вся інформація, яку збирає маркетплейс захищена технічними засобами і процедурами забезпечення безпеки з метою запобігання несанкціонованому доступу або використання даних. Афілійовані з Компанією особи, надійні партнери і незалежні постачальники послуг зобов'язуються використовувати отриману від Компанії інформацію відповідно до вимог до безпеки і Політики Конфіденційності.

Розглянемо вкладку «безпека».(рис.2.17)

Даний розділ надзвичайно зручний та важливий для безпеки користувачів. У ньому є форма, у яку користувач може додати посилання на оголошення, перевіривши чи є це посилання на справжній сайт olx.ua, що не дає можливості шахраям ввести користувача в оману для отримання його персональних даних та фінансових активів.

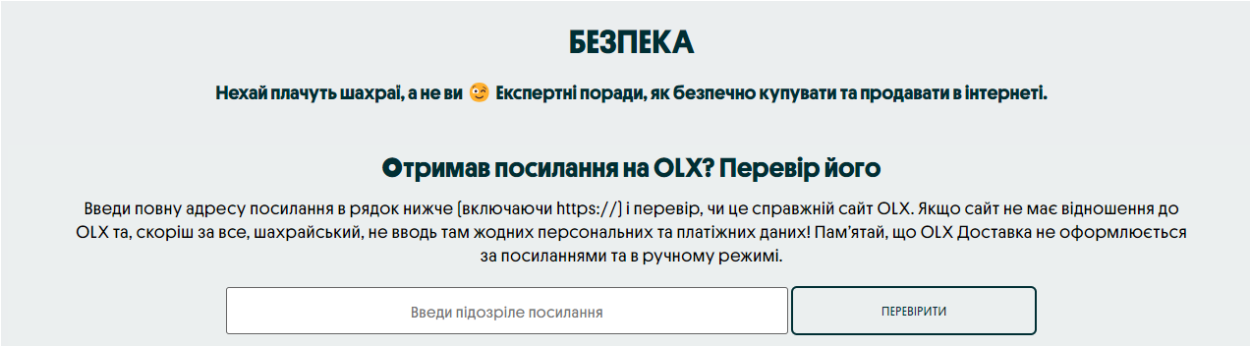

Рисунок 2.17 - Вміть вкладки «безпека»

Одним із ключових аспектів захисту даних є шифрування. OLX використовує шифрування для захисту передачі даних між користувачем і сервером, забезпечуючи конфіденційність і цілісність інформації. OLX має механізми для виявлення та реагування на порушення безпеки даних, включаючи систему повідомлень про підозрілу активність та контакт з користувачем у разі необхідності.

Виходячи з аналізу маркетплейсу доходимо висновку, що сайт дотримується всіх вимог безпеки та надає користувачам можливість для перевірки посилань, що робить процес купівлі/продажу безпечним.

Оцінки безпеки OLX.ua та Privat24 надали практичне розуміння впровадження заходів захисту персональних даних на популярних фінансових платформах. Хоча обидві платформи продемонстрували різні засоби контролю безпеки та зусилля з навчання користувачів, все ще існували потенційні області для вдосконалення, такі як більш детальний контроль доступу, біометрична автентифікація та інтерактивне навчання з питань безпеки [21].

2.4. Основні алгоритми шифрування

Шифрування є основним методом захисту персональних даних в онлайн-банкінгу. Всі алгоритми шифрування поділяються на симетричні, асиметричні та гібридні методи.

Симетричне шифрування використовує один і той самий ключ для зашифрування та розшифрування даних. Це швидкий та ефективний метод, проте вимагає безпечної передачі ключа між сторонами. Характеристики та варіанти використання симетричного шифрування наведено в таблиці 8

Таблиця 8 - Основні алгоритми симетричного шифрування

Алгоритм	Довжина ключа	Безпека	Використання
AES (Advanced Encryption Standard)	128, 192, 256 біт	Дуже висока	Банки, VPN, військові системи
3DES (Triple Data Encryption Standard)	112, 168 біт	Середня (застарілий)	Старі банківські системи, ATM-термінали

Приклад роботи AES

1. Користувач вводить PIN-код у банкомат.
2. PIN шифрується за допомогою AES-256.
3. Передається у банк у зашифрованому вигляді.
4. Банк розшифровує та перевіряє правильність.

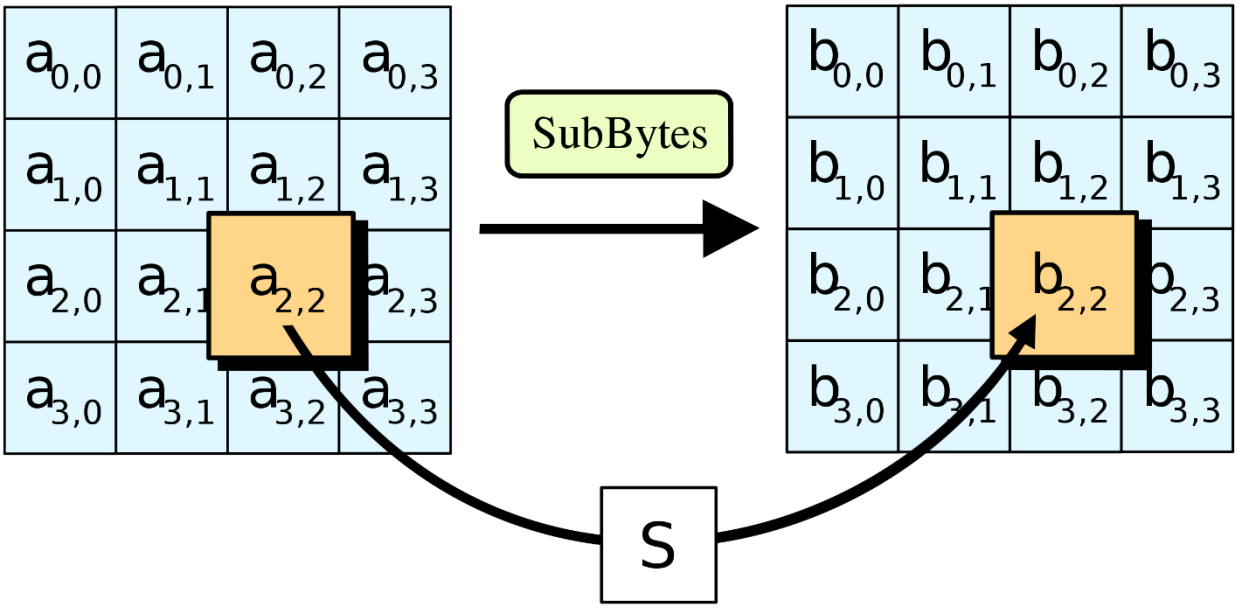

Рисунок 2.18 - Схема роботи AES [9]

Перевагами симетричного шифрування є висока швидкість обробки даних та мінімальне використання обчислювальних ресурсів.

Проте, є і недоліки, а саме: проблема передачі ключа між сторонами та вразливість до атак при втраті ключа.

Асиметричне шифрування використовує пару ключів: відкритий ключ для шифрування та закритий ключ для розшифрування. У таблиці 9 наведено основні алгоритми асиметричного шифрування.

Таблиця 9 – Основні алгоритми асиметричного шифрування

Алгоритм	Довжина ключа	Безпека	Використання
RSA (Rivest-Shamir-Adleman)	2048, 4096 біт	Висока	SSL/TLS, цифрові підписи
ECC (Elliptic Curve Cryptography)	256, 384, 521 біт	Вища, ніж у RSA при меншій довжині ключа	Мобільні платежі, криптовалюти

Приклад роботи RSA:

1. Банк надсилає клієнту відкритий ключ RSA.
2. Клієнт шифрує свій пароль відкритим ключем.

Лише банк може розшифрувати повідомлення за допомогою свого закритого ключа

Рисунок 2.19 - Схема RSA [10]

Переваги асиметричного шифрування:

- Не потрібно передавати закритий ключ.
- Висока безпека навіть при відкритому доступі до публічного ключа.

Недоліки асиметричного шифрування:

- Нижча швидкість роботи у порівнянні з симетричним шифруванням.
- Великі ключі потребують більше ресурсів для обчислень.

Оскільки асиметричне шифрування повільне, а симетричне має проблему передачі ключів, банки використовують гібридні методи. Суть гібридного методу полягає у використанні RSA або ECC для безпечної передачі симетричного ключа. Далі всі дані шифруються швидким AES-256.

Приклад у банкінгу (SSL/TLS)

Банк надсилає клієнту відкритий ключ RSA.

1. Клієнт створює **симетричний ключ AES** і шифрує його RSA.
2. Банк отримує зашифрований ключ і розшифровує його.
3. Далі всі дані передаються через швидке **AES-шифрування**.

Перевагами гібридного методу є висока безпека завдяки асиметричному шифруванню та висока швидкість роботи завдяки симетричному шифруванню. Основним недоліком гібридного методу є ускладнена реалізація у порівнянні з окремими методами.

Для наочності у таблиці 10 наведено порівняльну характеристику алгоритмів шифрування.

Таблиця 10 - Порівняльна таблиця алгоритмів

Метод	Приклад	Ключ	Безпека	Швидкість	Використання
Симетричне	AES	128-256 біт	Висока	Висока	Шифрування даних у банку
Асиметричне	RSA, ECC	2048+ біт	Дуже висока	Повільна	Аутентифікація, цифрові підписи
Гібридне	TLS (RSA + AES)	256+ біт	Дуже висока	Висока	Банківські операції онлайн

У банківській сфері гібридні методи є найкращим рішенням, оскільки поєднують надійність асиметричного шифрування для передачі ключів та швидкість симетричного шифрування для обробки транзакцій. Основними стандартами є AES-256, RSA-4096 та ECC-256, які забезпечують високий рівень безпеки для фінансових установ.

2.5. Протоколи захисту передавання даних

Захист передавання даних є ключовим аспектом безпеки в онлайн-банкінгу та інших системах, що працюють з конфіденційною інформацією. Існує декілька основних протоколів, що забезпечують безпечний обмін даними між клієнтом і сервером, серед яких SSL/TLS, HTTPS, VPN і TOR.

Протоколи Secure Sockets Layer (SSL) і Transport Layer Security (TLS) забезпечують шифрування переданих даних, автентифікацію сторін та збереження цілісності інформації. TLS є більш сучасною версією SSL і використовується майже у всіх сучасних веб-застосунках. Під час встановлення з'єднання відбувається процес «рукостискання» (handshake), під час якого сервер і клієнт узгоджують параметри шифрування та обмінюються

сертифікатами. TLS підтримує різні криптографічні алгоритми, включаючи AES та RSA, що забезпечують високу безпеку комунікацій [11].

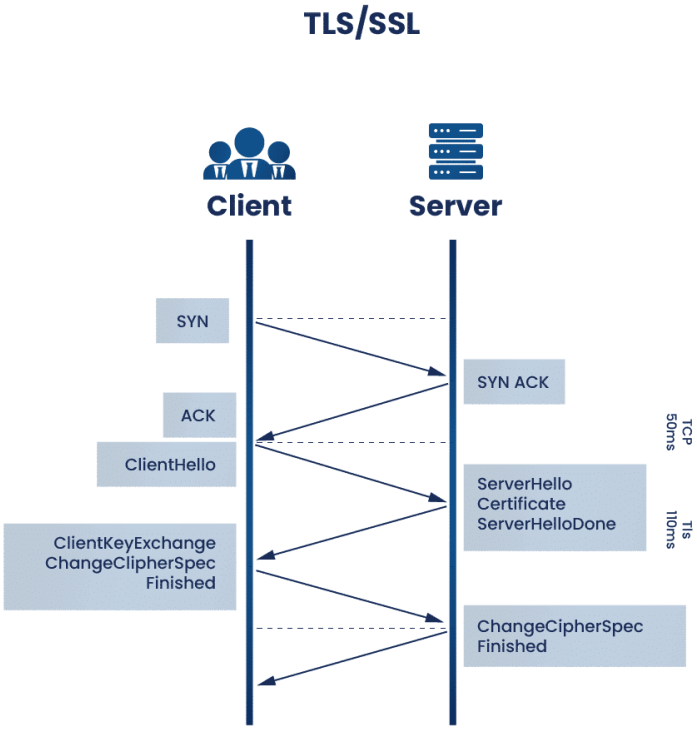

Рисунок 2.20 - Схема рооботи TLS/SSL [12]

HyperText Transfer Protocol Secure (HTTPS) – це розширена версія HTTP, що використовує протокол TLS для захисту передаваних даних. Основними перевагами HTTPS є:

шифрування даних, що унеможливлює їх перехоплення;

автентифікація веб-сайту за допомогою цифрових сертифікатів;

забезпечення цілісності переданих повідомлень.

HTTPS є стандартом для всіх фінансових сервісів, онлайн-банкінгу та інших критично важливих застосунків, що обробляють персональні дані користувачів [13].

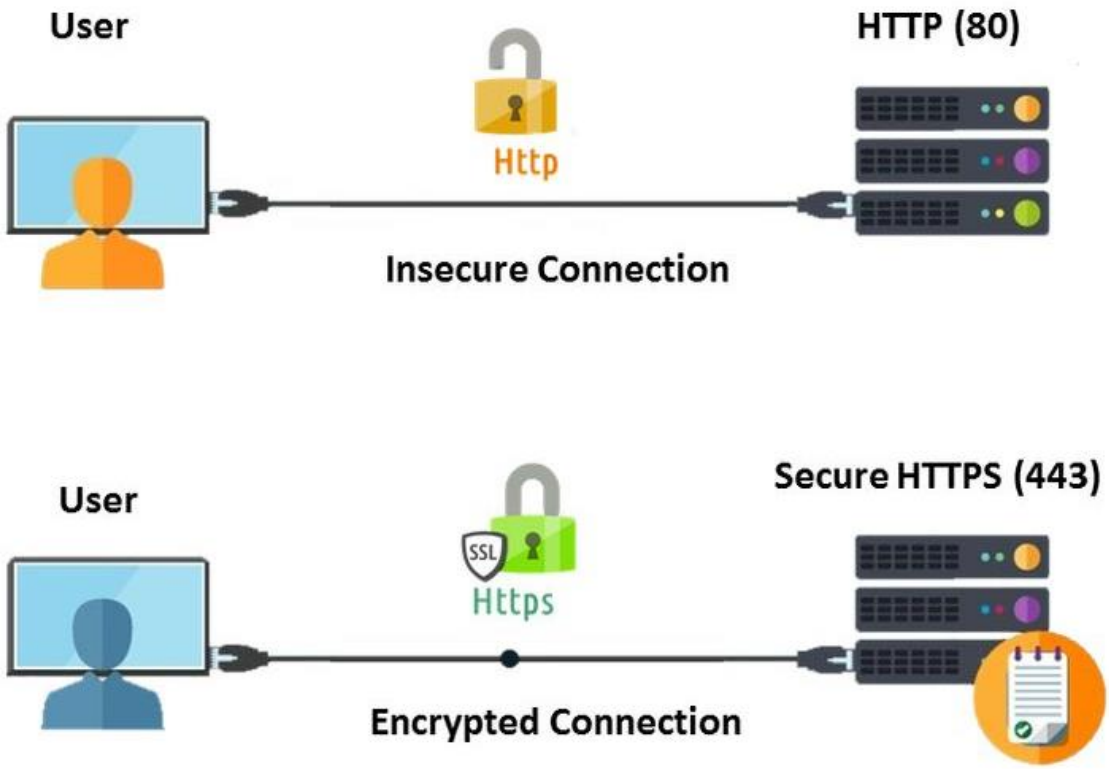

Рисунок 2.21 - Різниця між HTTP та HTTPS

Virtual Private Network (VPN) створює зашифрований тунель між користувачем і сервером, дозволяючи безпечно передавати дані через незахищені мережі. VPN-шлюзи використовують різні протоколи шифрування, такі як OpenVPN, IPSec, WireGuard, що забезпечують конфіденційність і захист від атак типу Man-in-the-Middle. У контексті онлайн-банкінгу VPN дозволяє безпечно підключатися до банківських сервісів навіть у публічних Wi-Fi-мережах [14].

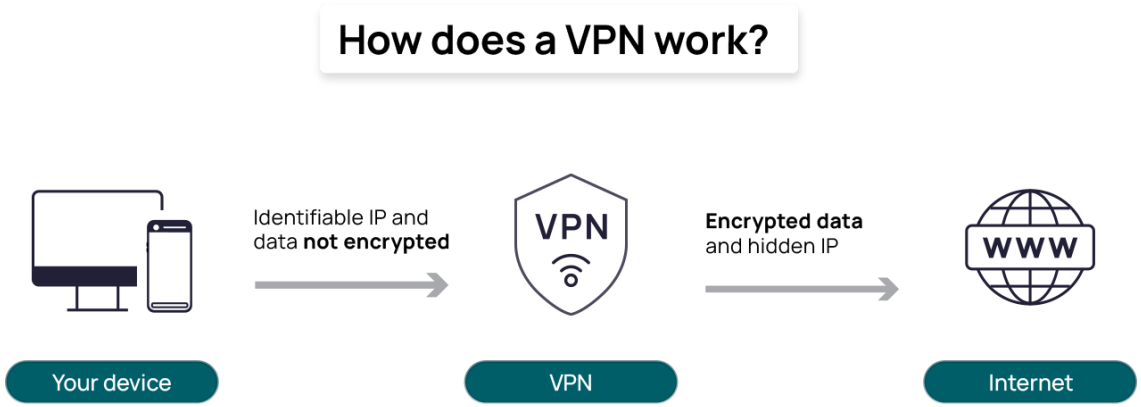

Рисунок 2.22 - Схема роботи VPN

The Onion Router (TOR) – це мережа анонімізації, яка маршрутизує трафік через кілька вузлів, забезпечуючи конфіденційність користувача. На відміну від VPN, TOR не просто шифрує трафік, а також приховує IP-адресу користувача, що робить його ефективним засобом захисту від стеження. Проте використання TOR у банківській сфері є обмеженим через потенційні ризики шахрайства та уповільнення швидкості передавання даних [15].

Таким чином, для забезпечення безпеки передавання даних онлайн-банкінги активно застосовують TLS/HTTPS, а також можуть використовувати VPN для додаткового рівня захисту.

2.6. Оцінка ефективності технологій шифрування в банківському секторі

Захист персональних даних у банківському секторі критично залежить від ефективності технологій шифрування. Банки застосовують сучасні криптографічні методи для забезпечення конфіденційності, цілісності та автентифікації даних клієнтів. Оцінка ефективності цих технологій базується на стійкості до атак, продуктивності, відповідності регуляторним вимогам та можливості масштабування.

Основні критерії оцінки ефективності

Стійкість до атак. Надійні алгоритми шифрування повинні протистояти сучасним загрозам, включаючи атаки грубої сили, квантові атаки та криптоаналіз. Наприклад, AES-256 вважається безпечним навіть у довгостроковій перспективі [16].

Продуктивність. Висока швидкість шифрування та дешифрування важлива для забезпечення швидкої обробки банківських транзакцій. Алгоритми, такі як ChaCha20, можуть бути ефективнішими за AES у мобільних застосунках завдяки меншому споживанню ресурсів [17].

Відповідність стандартам. Банки повинні відповідати вимогам міжнародних стандартів, таких як PCI DSS (Payment Card Industry Data Security Standard) та ISO/IEC 27001. Використання TLS 1.3 у банківських сервісах гарантує відповідність сучасним стандартам безпеки [18].

Масштабованість та інтеграція. Шифрувальні технології мають легко інтегруватися в існуючу інфраструктуру банку, підтримувати багатофакторну автентифікацію та бути сумісними з мобільними пристроями.

Технологія	Стійкість до атак	Продуктивність	Використання в банківському секторі
AES-256	Висока	Висока	Використовується для зберігання та передавання даних
RSA-2048	Середня (можливі квантові атаки)	Низька	Використовується для автентифікації
ECC (Elliptic Curve Cryptography)	Висока	Висока	Альтернатива RSA для цифрових підписів
ChaCha20	Висока	Дуже висока	Використовується у мобільних та веб-системах
CRYSTALS-Kyber	Висока (постквантова)	Висока	Перспективна технологія для майбутнього застосування

Рисунок 2.23 - Порівняння основних технологій шифрування

З появою квантових комп’ютерів деякі класичні алгоритми шифрування можуть стати вразливими. Наприклад, RSA-2048 та ECC можуть бути зламані алгоритмом Шора [19]. У зв’язку з цим банки досліджують постквантові

криптографічні алгоритми, такі як CRYSTALS-Kyber та SPHINCS+, рекомендовані Національним інститутом стандартів і технологій США (NIST) [20].

Використання технологій шифрування у банківській сфері. Банки впроваджують шифрування в різні аспекти своєї діяльності:

Захист клієнтських даних: збереження паролів, номерів карток, особистих даних у зашифрованому вигляді.

Безпечні онлайн-транзакції: використання TLS 1.3 та HTTPS для захисту передавання даних між клієнтом і сервером.

Автентифікація користувачів: застосування цифрових сертифікатів (PKI), двофакторної автентифікації (2FA) та біометричних методів.

Шифрування резервних копій: захист резервних даних за допомогою AES-256 та інших алгоритмів.

Таким чином, ефективність технологій шифрування у банківському секторі залежить від їхньої стійкості до атак, продуктивності та відповідності стандартам безпеки. Використання передових алгоритмів, таких як AES-256, ECC, ChaCha20, у поєднанні з впровадженням постквантових рішень, забезпечує надійний захист фінансових операцій та персональних даних клієнтів.

3 ПРАКТИЧНЕ ДОСЛІДЖЕННЯ: СТВОРЕННЯ ТА АНАЛІЗ "САЙТУ-ФОРТЕЦІ"

3.1.Розробка "сайту-фортеці"

У межах практичної частини кваліфікаційної роботи було розроблено безпечний вебзастосунок під назвою Secure Bank. Його мета — демонстрація інтеграції сучасних методів захисту персональних даних, зокрема у сфері онлайн-банкінгу. Сайт реалізує функціонал автентифікації з використанням паролів та passkey (WebAuthn), а також включає сесійну перевірку, шифрування даних, контроль часу активності користувача, використання TLS-сертифікатів, обмеження у кількості спроб за певний час, перевірку через сервіс reCAPTCHA v3 та захист від несанкціонованого доступу.

Розробка сайту відбувалась із використанням декількох технологій. Flask (фреймворк) для створення серверної частини Python-застосунку. SQLite легка реляційна база даних для зберігання облікових записів користувачів. React + TypeScript + Vite для створення фронтенду з високою продуктивністю. dotenv для безпечного зберігання змінних середовища (API-ключі, конфігурації). werkzeug.security для хешування паролів. JWT для декодування токенів під час входу через passkey. requests для взаємодії з зовнішнім WebAuthn-сервісом Hanko. Обмеження у кількості спроб входу за певний час для захисту від атак методом брутфорс та від DoS-атак(Denial of Service). reCAPTCHA v3 для захисту від спаму та зловживань, а також для виявлення інших типів шахрайських дій, як-от введення облікових даних, захоплення облікового запису (ATO) і автоматичне створення облікового запису.

Обмеження кількості спроб за певний час (за хвилину) на сайті реалізується для забезпечення безпеки та захисту від різних типів атак. Захист від атак методом брутфорс забезпечується зменшенням ймовірність успіху брутфорс-атаки та вчасно виявити проблему, в результаті чого заблокувати або затримати зловмисників, які намагаються вгадати паролі або виконувати інші небажані дії. Також обмеження спроб допомагає захистити сервер від

надмірної кількості запитів та протистояти атакам з метою виснаження ресурсів серверу, що виключає можливість DoS-атаки.

reCAPTCHA пропонує розширене виявлення з більш детальними оцінками, кодами причин для ризикованих подій, SDK для мобільних додатків, виявленням злому або витоку паролів, багатофакторною автентифікацією (MFA) і можливістю налаштувати модель сайту для захисту корпоративного бізнесу. Коли reCAPTCHA розгортається у середовищі, він взаємодіє з серверною частиною та клієнтом (веб-сторінками).

Коли кінцевий користувач відвідує веб-сторінку або використовує мобільну програму, послідовно запускаються такі події:

1. Клієнт завантажує веб-сторінку з серверної частини або запускає мобільний додаток.
2. Веб-сторінка або мобільний додаток ініціалізує reCAPTCHA JavaScript API або мобільний SDK, який починає збирати сигнали.
3. Коли кінцевий користувач запускає дію, захищену reCAPTCHA, наприклад вхід, API reCAPTCHA JavaScript або мобільний SDK у клієнті запитує вердикт від reCAPTCHA.
4. reCAPTCHA повертає зашифрований маркер reCAPTCHA клієнту для подальшого використання.
5. Клієнт надсилає зашифрований токен reCAPTCHA на сервер для оцінки.
6. Сервер надсилає assessments.createзапит на створення оцінки ( ) і зашифрований маркер reCAPTCHA до reCAPTCHA.
7. Після оцінки reCAPTCHA повертає вердикт (оцінки від 0,0 до 1,0 і код причини) на основі ризику, оціненого для цього запиту, до серверної частини.
8. Залежно від вердикту ви (як розробник) можете визначити наступні кроки для конкретного запиту або дії користувача [**Ошибка! Источник ссылки не найден.**].

На діаграмі послідовності показано графічне представлення робочого процесу reCAPTCHA(див. додаток А).

У рамках реалізації безпарольної автентифікації в проєкті Secure Bank використовується Hanko API - платформа, що надає зручні інтерфейси для впровадження сучасних методів входу в систему без паролів. Вона підтримує аутентифікацію за допомогою Passkey (WebAuthn) та дозволяє використовувати біометричні методи (Face ID, Touch ID), апаратні токени (YubiKey тощо) та інші засоби, що забезпечують високий рівень безпеки [**Ошибка! Источник ссылки не найден.**].

Основними етапами інтеграції Hanko API є чотири етапи: реєстрація Passkey для користувача, завершення реєстрації, логування користувача через Passkey, отримання JWT (JSON Web Token). На першому етапі користувач ініціює процес реєстрації, у якому генерується PublicKeyCredentialCreationOptions. Для цього здійснюється POST-запит до Hanko API, що повертає необхідні параметри для створення ключа на фронтенді. Після того, як користувач створив ключ на своєму пристрої, фронтенд надсилає результат назад на сервер для завершення реєстрації, використовуючи відповідний POST-запит. Коли користувач хоче увійти в систему, сервер ініціює логін через Hanko API. Сервер надсилає запит на перевірку дійсності ключа, що був попередньо зареєстрований, і отримує підтвердження чи відмову. Після успішного входу Hanko API надає токен (JWT), який містить ідентифікатор користувача (user_id). Цей токен використовується для авторизації у подальших запитах до сервера.

Інтеграція Hanko API дозволяє значно спростити процес впровадження безпарольної автентифікації, звільняючи розробників від необхідності самостійно реалізовувати складні криптографічні операції та обробку підписів. Платформа забезпечує відповідність сучасним стандартам безпеки, таким як WebAuthn та FIDO2, і дозволяє користувачам користуватися біометрією та апаратними ключами для доступу до їх облікових записів. Це

підвищує рівень захисту та забезпечує зручність користування без потреби запам’ятовувати паролі.

Перевагами використання Hanko API в проєкті є зручність і безпека завдяки безпарольній автентифікації, зниження ризику фішингових атак та викрадення паролів, підвищення довіри користувачів завдяки використанню біометричних даних або апаратних ключів для доступу.

Проєкт складається з двох основних частин: бекенду та фронтенду. Структуровану карту файлів наведено на рисунку 3.1.

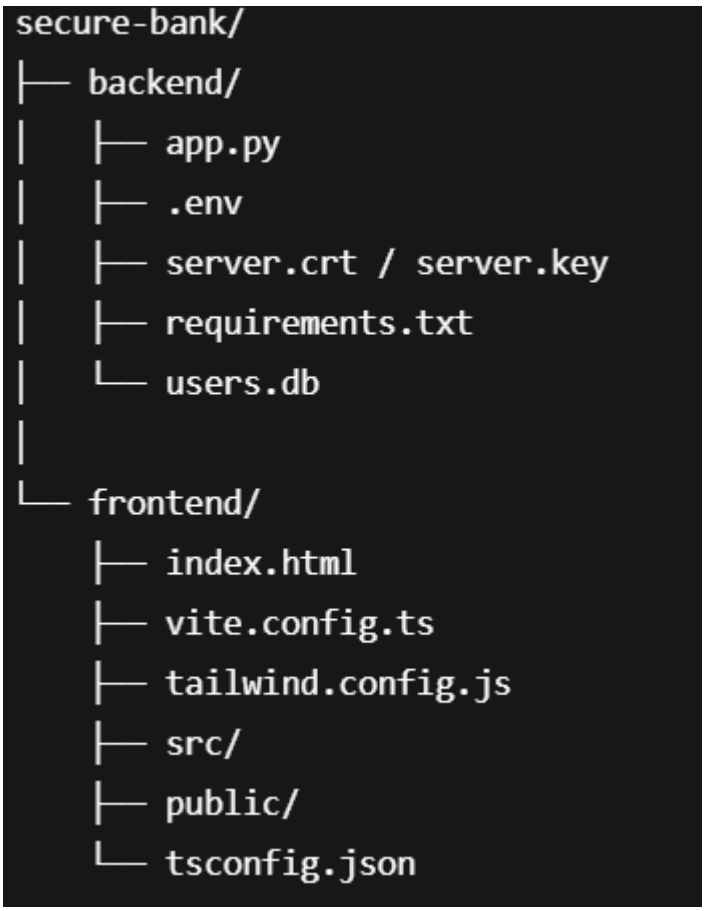

Рисунок 3.1 - Структурована карта файлів проєкту.

- app.py — реалізує маршрутизацію, логіку автентифікації, створення сесій, взаємодію з Hanko API.
- .env — містить конфіденційні дані (ключі доступу до WebAuthn-платформи).

- users.db — база даних, у якій зберігаються зареєстровані користувачі та їх ключі.
- server.crt / server.key — використовуються для запуску сервера через HTTPS (TLS/SSL).
- requirements.txt — описує усі бібліотеки Python, необхідні для запуску застосунку.
- src/ — основна логіка фронтенду, компоненти, маршрути, виклики до API.
- vite.config.ts — визначає середовище запуску, шляхи, HTTPS-підключення.
- tailwind.config.js — стилізація через утилітарний CSS-фреймворк.
- package.json — містить перелік залежностей, скриптів та конфігурацій.

У рамках розробки сайту Secure Bank для забезпечення безпеки користувачів були реалізовані різноманітні механізми захисту. Зокрема, це хешування паролів, керування сесіями, використання Passkey (WebAuthn) для безпарольної автентифікації та захист через HTTPS.

Для захисту паролів користувачів використовувалась бібліотека werkzeug.security, яка дозволяє хешувати паролі перед збереженням у базі даних. Це гарантує, що навіть у випадку компрометації бази даних паролі користувачів не будуть доступні в текстовому вигляді.

```
from werkzeug.security import generate_password_hash, check_password_hash

# Генерація хешу пароля при реєстрації
hashed_password = generate_password_hash(password)
# Перевірка пароля під час логіну
if check_password_hash(user[2], password):
```

Для управління сесіями та забезпечення безпеки користувачів застосовано використання сесій через бібліотеку Flask. Сесії дозволяють зберігати інформацію про користувача протягом його сесії на сайті та захищати

від несанкціонованого доступу. Для цього використовується сесійний ідентифікатор, який генерується при кожному успішному вході.

```
import uuid

# Генерація унікального ID сесії
session_id = str(uuid.uuid4())
session["user_id"] = user[0]
session["email"] = user[1]
session["session_id"] = session_id
```

Крім того, реалізовано перевірку активних сесій:

```
from datetime import datetime, timedelta

# Перевірка часу останньої активності сесії
if datetime.now() - active_sessions[user_id]['last_activity'] > timedelta(minutes=30):
    del active_sessions[user_id]
    session.clear()
```

Для забезпечення безпарольної автентифікації в проекті використовується інтеграція з Hanko API для підтримки WebAuthn. Це дозволяє користувачам реєструвати та використовувати біометричні дані або апаратні токени (наприклад, YubiKey) для входу в систему без введення пароля.

Після реєстрації користувача через Passkey, наступний код ініціює процес входу:

```
import requests

@app.route("/passkey/start-login", methods=["POST"])
def start_login():
    response = requests.post(f"{baseUrl}/login/initialize", headers=headers)
    login_options = response.json()
    return jsonify(login_options)
```

Після того, як користувач успішно автентифікується за допомогою Passkey, сервер генерує токен, який потім використовується для авторизації:

```
import jwt

token = data.get('token')
decoded_payload = jwt.decode(token.encode(), options={"verify_signature": False})
user_id = decoded_payload.get('sub')
```

Для забезпечення безпеки передачі даних між клієнтом та сервером використовується протокол HTTPS. Сервер налаштований на підтримку SSL/TLS з використанням сертифікатів, що гарантує захищене з'єднання та захист даних від перехоплення.

```
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=443, ssl_context=('server.crt', 'server.key'))
```

Також, для безпеки сесій та захисту від атак типу "man-in-the-middle", сервер налаштовується на роботу з HTTPS-з'єднаннями. Повний код наведено у додатку Б.

3.2. Проведення тестування на проникнення (пентестинг)

Пентестинг, або тестування на проникнення, є ключовим етапом в оцінці безпеки веб-застосунків та інфраструктури. Метою цього процесу є виявлення потенційних вразливостей, які можуть бути використані зловмисниками для несанкціонованого доступу до системи або конфіденційної інформації. Оскільки безпека веб-застосунків є критично важливою, проведення пентестингу дає змогу підтвердити надійність системи та забезпечити належний рівень захисту.

Для оцінки рівня захищеності створеного сайту проведено тестування з урахуванням різних аспектів безпеки, таких як аутентифікація, управління сесіями, захист від атак на веб-додатки, а також використання криптографічних методів для захисту даних. Під час тестування перевірились всі основні функціональні компоненти, а також можливі точки доступу для зловмисників.

Першим етапом перевірки є захист від брутфорсу та Dos-атак шляхом перебору паролів. При підборі паролів до акаунту існуючого користувача отримали повідомлення про те, що можливість введення пароля відсутня на 1 хвилину, через велику кількість введення неправильного паролю (5) (рис.).

Захист від підбору паролю реалізовано шляхом обмеження кількості спроб входу з одної IP-адреси (максимум 5 спроб протягом 60 секунд) (рис.3.2).

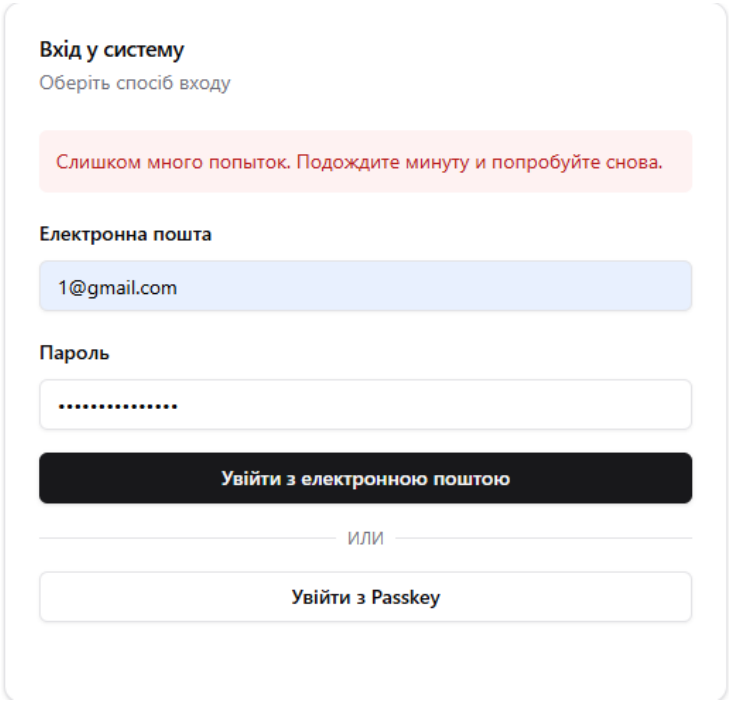

Рисунок 3.2 - Обмеження кількості спроб

Другим етапом перевірки є прехват паролю у базі даних, що також є неефективною, через те, що використовується werkzeug.security для хешування паролей за допомогою generate_password_hash та перевірки через check_password_hash. Навіть при витоці інформації з бази даних атакуючий не отримає реальний пароль, оскільки він зберігається у форматі хешу (рис. 3.3).

	id	email	password
	Фильтр	Фильтр	Фильтр
1	a2ea98e9-2b1f-41ce-be6b-d9e1ce959b65	sofiavelichko33@gmail.com	scrypt:32768:8:1$dyXDNcWZfGnKXCgt$1e0295562c75df67c3248834b7e92f56b8264031fb7419f9cb1575e6d2e
2	41c6f5f2-3efa-4025-902c-193fbc389d35	1@gmail.com	scrypt:32768:8:1$09nqT62aKtpKpcwo$e0ec0ee34ea74cafa2d0f9d98de7c36bada5901a44673d3fbd286cf594c
3	5c798ba8-e2ab-4db4-bbd5-5d73a13d2ce5	Lexa@gmail.com	scrypt:32768:8:1$luh2j70gbD8eHoE8$5ab9f3360c19b4e925f279308c374e260724be308e9ab5fc88fa2482e13
4	44ce8c40-18e9-4b92-adee-6d61c0e40dbb	000@gmail.com	scrypt:32768:8:1$CH0OFq0WyWIg16Sj$6e85dc6d7043b12eda0c893b91603fdcea236350109763e21d7b9fba2ec

Рисунок 3.3 - База даних з хешованим паролем

Наступним етапом є перевірка сожливості атаки з перехватом сесії та з підміною сесії. Використання Flask із сесіями для зберігання даних про користувача, а також перевірка цілісності сесій з використанням унікальних ID

сесій, зберігання сесій у словнику active_sessions та терміну дії сесії. Якщо сесія закінчується (після 30 хвилин бездіяльності), користувач буде змушений знову увійти до системи. Збереження унікальних ID сесій запобігає атакам з фальсифікацією сесій.

```
# Генеруємо унікальний ID сесії для нового користувача
        session_id = str(uuid.uuid4())
        session["user_id"] = user_id
        session["email"] = email
        session["session_id"] = session_id
```

Крім того, для забезпечення безпеки сесійних даних використовується параметр SESSION_COOKIE_SECURE, який дозволяє передавати cookies тільки через захищений канал (HTTPS), а також параметр SESSION_COOKIE_SAMESITE, щоб запобігти атакам типу CSRF.

```
app.config.update(
    SESSION_COOKIE_SECURE=True,          # Куки передаються лише по HTTPS
    SESSION_COOKIE_SAMESITE="None"       # Дозволити кросс-доменні запити
)
```

Також для запобігання зловживанням сесіями сесії автоматично очищаються від застарілих записів, що допомагає захиститися від витоків даних. Старі дані видаляються під час роботи з кожним новим запитом.

```
def cleanup_old_attempts(ip):
    now = time.time()
    login_attempts[ip] = [t for t in login_attempts.get(ip, []) if now - t < BLOCK_WINDOW]
```

reCAPTCHA у проекті виконує кілька важливих функцій захисту від автоматизованих атак, таких як Brute Force і Fake Registrations. Цей захист забезпечується перевіркою, чи є запит від реального користувача, а чи не від автоматичного бота. Валідація відбувається при вході користувача, реєстрації та аутентифікації через Passkeys, забезпечуючи безпеку та запобігаючи зловживанням.

```
# Функція для перевірки reCAPTCHA токена
def verify_recaptcha(token):
    if not token:
        return False, "reCAPTCHA token is missing"

    response = requests.post(
        "https://www.google.com/recaptcha/api/siteverify",
```

```
            data={
                "secret": RECAPTCHA_SECRET_KEY,
                "response": token,
            },
        )
        result = response.json()

        print("reCAPTCHA verification result:", result)    # Отладка

        # перевіряємо успіх
        if not result.get("success"):
            error_codes = result.get("error-codes", [])
            return False, f"reCAPTCHA verification failed: {error_codes}"
        # перевіряємо score
        if result.get("score", 0) < 0.3:  # Сниженный порог для тестов
            return False, f"reCAPTCHA score too low: {result.get('score')}"

        return True, None
```

Від атак типу Man-in-the-Middle (MITM) використано захист завдяки використанню HTTPS та безпечних cookies. Навіть у випадку, коли атакуючий спробує перехопити трафік, то не зможе розшифрувати дані, так як весь трафік зашифровано з використанням HTTPS.

Також неможливою є атака XSS, коли зловмисник може впровадити шкідливий скрипт на сторінку. Захист від такої атаки реалізовано завдяки контентній безпековій політиці (CSP), що обмежує джерела контенту, з яких може завантажуватися скрипт або дані.

```
    # Додаємо CSP-заголовок
    @app.after_request
    def add_security_headers(response):
        csp = (
            "default-src 'self'; "
            "script-src      'self'      https://www.google.com https://www.gstatic.com; "
            "frame-src https://www.google.com; "
            "frame-ancestors https://www.google.com; "
            "connect-src      'self'      https://www.google.com https://www.gstatic.com; "
            "img-src        'self'        https://www.google.com https://www.gstatic.com; "
            "style-src       'self'       https://www.google.com https://www.gstatic.com;"
        )
```

```
    response.headers['Content-Security-Policy'] = csp
    return response
```

Для усунення можливості фішингової атаки реалізовано використання passkeys. Passkeys є більш безпечною, ніж традиційні паролі, оскільки вона використовує криптографічні ключі, а не паролі.

```
# Passkeys
tenant_id = os.getenv("PASSKEY_TENANT_ID")
api_key = os.getenv("PASSKEY_API_KEY")

baseUrl = f"https://passkeys.hanko.io/{tenant_id}"
headers = {
    "apikey": api_key,
    "Content-Type": "application/json",
}
```

3.3.Користувацький інтерфейс

Користувацький інтерфейс (UI) вебзастосунку Secure Bank був розроблений із урахуванням принципів мінімалізму, ергономічності та безпеки. Основна мета — забезпечити просту, інтуїтивно зрозумілу взаємодію з системою навіть для недосвідчених користувачів, водночас інтегруючи критично важливі функції інформаційного захисту.

Інтерфейс реалізований за допомогою технологій React (TypeScript). Вся логіка комунікації з бекендом здійснюється через HTTP-запити до API сервера, реалізованого на Flask. Це дозволяє чітко розділити логіку клієнтської та серверної частин застосунку.

Головна сторінка — home.tsx. Ця сторінка є першою точкою взаємодії користувача з системою. Вона виконує роль привітального екрана, який містить:

- назву застосунку;
- логотип;
- кнопку «Увійти або зареєструватися», яка перенаправляє користувача на форму автентифікації.

Рисунок 3.4 - Головна сторінка вебзастосунку Secure Bank

Сторінка входу/реєстрації — login.tsx. Це центральний елемент взаємодії користувача із захищеними функціями системи. Залежно від того, чи існує email у базі даних, система автоматично:

- проводить автентифікацію (якщо користувач уже зареєстрований),
- або створює новий обліковий запис (реєстрація).
- Основні елементи сторінки:
- поля для введення email і пароля;
- кнопка «Увійти» або «Зареєструватися» — в залежності від наявності користувача;
- кнопка «Увійти з Passkey» — реалізація WebAuthn-автентифікації;
- візуальне повідомлення про правила до пароля: "Password must be >8 characters, with upper and lower case letters";
- перевірка введеного reCAPTCHA токена;
- сповіщення про помилки, такі як:
    - «Email and password are required»;
    - «Неправильний пароль»;
    - «Забагато спроб. Зачекайте хвилину та спробуйте ще раз.»;

- «Session expired - please login again» тощо.

Інтерфейс адаптований для мобільних пристроїв, усі повідомлення виводяться в зрозумілій формі з урахуванням досвіду взаємодії користувача.

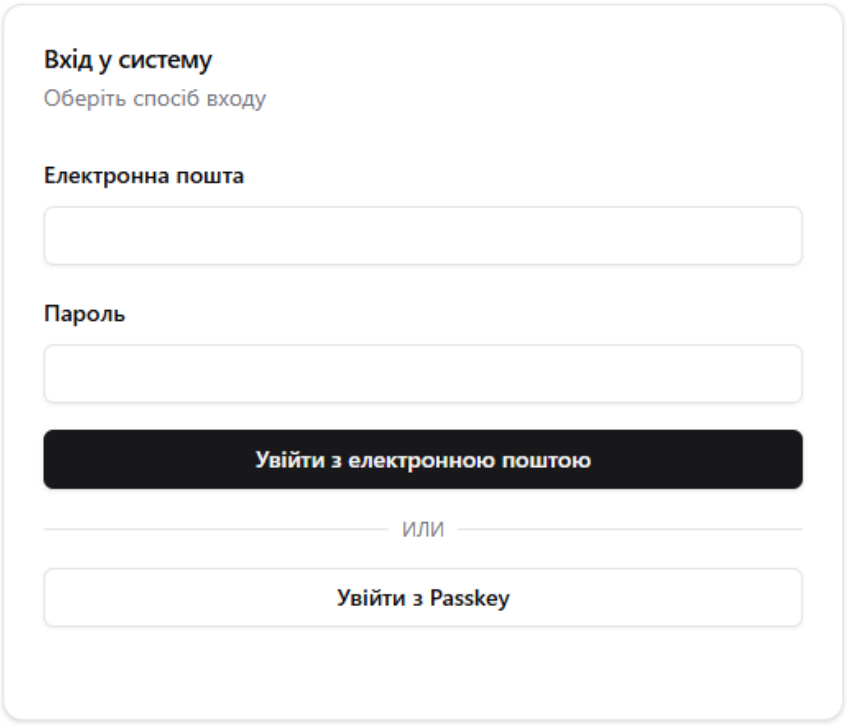

Рисунок 3.5 - Сторінка входу та реєстрації з повідомленнями та кнопкою WebAuthn

Особистий кабінет — dashboard.tsx. Це сторінка, доступна лише після успішного входу до системи. Вона містить:

- логотип застосунку;
- вітання: «Вітаємо, [ім’я користувача]»;
- кнопку «Вийти», яка завершує сесію та повертає користувача на головну сторінку;
- кнопку «Зареєструвати новий Passkey», що дозволяє додати ще один автентифікаційний метод на основі WebAuthn.

У цій частині реалізована перевірка сесії з контролем часу її дії та відповідності IP-адреси. У випадку виявлення активної, але некоректної або завершеної сесії, користувачу пропонується повторно ввійти до системи.

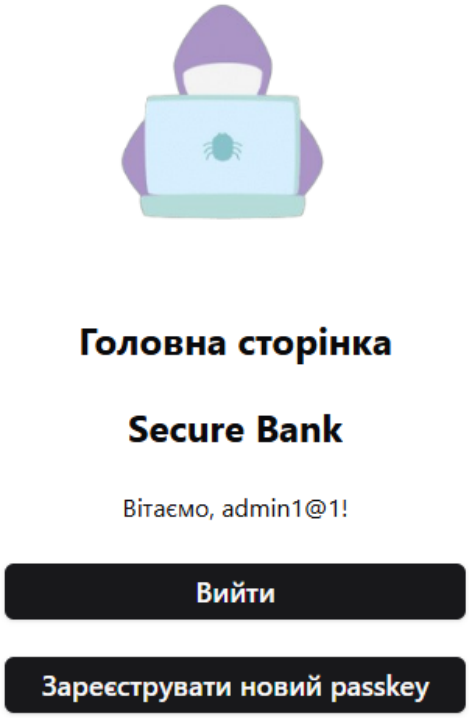

Рисунок 3.6 - Особистий кабінет користувача з функцією управління Passkey

Особливості дизайну з точки зору безпеки

Дизайн також враховує основні принципи "secure by design":

1. інтегровані рекапіча-перевірки;
2. обмеження на кількість спроб входу (з повідомленням);
3. виведення попереджень про причини відмови у доступі;
4. підтримка WebAuthn, що візуально маркується окремою кнопкою.

Ці елементи не лише посилюють кіберзахист, а й підвищують довіру користувачів до системи.

Користувацький інтерфейс Secure Bank є сучасним, мінімалістичним та достатньо гнучким для реалізації як традиційних, так і інноваційних механізмів автентифікації. Його побудовано відповідно до принципів безпеки,

адаптивності та прозорості, що дозволяє ефективно використовувати систему в навчальних, тестових або дослідницьких умовах.

3.4.Виявлені вразливості та рекомендації щодо їх усунення

У процесі тестування вебзастосунку було проведено аналіз безпеки реалізованих механізмів автентифікації, авторизації та захисту даних.

У проекті є механізм авторизації через електронну пошту та пароль, або з використанням Passkeys. Для забезпечення захисту від атак методом перебору паролей наявна перевірка на кількість спроб неправильного введення паролю а також додано використання reCAPTCHA. Для найбільш безпечного входу додано Passkeys, що значно підвищує безпеку, оскільки вони захищені від фішингу та підбору паролів. Навіть якщо зловмисник вкраде дані про passkey, він не зможе використовувати його без доступу до пристрою користувача. Усі ці заходи унеможливлюють атаки типу Brute Force.

Незважаючи на впровадження низки сучасних технологій безпеки, таких як WebAuthn (Passkey), reCAPTCHA, обмеження кількості спроб входу, Content Security Policy (CSP) та захищені сесії, було виявлено кілька потенційних вразливостей, а також можливості для вдосконалення існуючих рішень.

У проєкті список активних сесій реалізовано через звичайний словник active_sessions, який існує лише під час роботи одного інстансу сервера. У разі масштабування застосунку або перезапуску сервера всі активні сесії буде втрачено. Рекомендовано дація: впровадити централізоване сховище для сесій, наприклад Redis або іншу систему збереження стану. Це забезпечить стабільність роботи застосунку в розподілених середовищах.

Хоча сесійна автентифікація використовується разом із SESSION_COOKIE_SAMESITE="None" для підтримки крос-доменних запитів, у застосунку відсутня перевірка CSRF-токенів для захисту POST-запитів від підробки. Рекомендовано додати реалізацію механізму CSRF-

захисту, наприклад, за допомогою токенів, що генеруються сервером і перевіряються на клієнті під час кожного запиту.

У функції verify_recaptcha() перевіряється лише токен користувача та його оцінка (score), однак не здійснюється звірка з IP-адресою, з якої відправлено запит. Рекомендовано у разі використання reCAPTCHA v3 перевіряти також поле hostname або action, щоб виключити можливість обходу перевірки сторонніми сервісами.

Наразі нові облікові записи автоматично створюються у випадку, якщо користувача не знайдено в базі, незалежно від кількості попередніх реєстрацій з певної IP-адреси. Це може призвести до масової реєстрації фейкових облікових записів (атака типу DoS). Рекомендовано запровадити обмеження кількості реєстрацій з одного IP за певний період, з подальшим відображенням CAPTCHA або повідомлення про перевищення ліміту.

Паролі користувачів хешуються за допомогою функції generate_password_hash(), однак поля, що зберігають публічні ключі WebAuthn (public_key), не шифруються. У разі компрометації бази даних ці дані можуть бути використані зловмисниками. Рекомендовано реалізувати шифрування публічних ключів із використанням симетричного ключа, збереженого в захищеному середовищі (наприклад, у .env-файлі або у менеджері секретів).

Функція is_valid_password() перевіряє лише наявність великих і малих літер та мінімальну довжину, але не враховує спеціальні символи, цифри чи заборонені шаблони (наприклад, email як пароль). Рекомендовано удосконалити політику складності паролів, додавши вимоги до наявності цифр, спецсимволів та виключення очевидних шаблонів.

У реалізації входу через Passkey токен, що повертається API Hanko, не перевіряється на наявність поля exp (expiration time). Це створює ризик використання токена повторно у разі його перехоплення. Рекомендовано під час розбору JWT-токена варто перевіряти час його дії, а також верифікувати підпис, якщо конфігурація безпеки дозволяє доступ до публічного ключа Hanko.

Попри впровадження сучасних технологій захисту, програмний застосунок має кілька векторів покращення, пов’язаних із збереженням сесій, захистом від CSRF, посиленням обмежень доступу та шифруванням чутливих даних. Усунення зазначених вразливостей дозволить істотно підвищити рівень інформаційної безпеки вебзастосунку.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи було досліджено актуальні загрози інформаційній безпеці в онлайн-банкінгу та проаналізовано ефективність сучасних криптографічних технологій, що використовуються для захисту персональних даних. Огляд загроз підтвердив, що фінансові платформи є об’єктами численних атак, зокрема фішингу, атак «людина посередині», SQL-ін’єкцій, XSS, перебору паролів тощо. Для успішного протистояння цим загрозам необхідна комплексна система захисту, що поєднує як програмні, так і організаційні заходи.

В рамках практичної частини було реалізовано вебзастосунок Secure Bank — прототип «сайту-фортеці», що імітує функціонал сучасного банківського онлайн-сервісу з підвищеним рівнем безпеки. Застосунок включає такі механізми захисту:

- автентифікація за допомогою пароля та технології WebAuthn (Passkey),
- хешування паролів за допомогою werkzeug.security,
- захист від атак перебору шляхом обмеження кількості спроб,
- перевірка reCAPTCHA v3,
- реалізація захищених сесій з контрольованим часом життя,
- використання TLS-сертифікатів для захисту трафіку,
- впровадження політики CSP (Content Security Policy),

Під час тестування було виявлено низку потенційних слабких місць, серед яких — збереження сесій у пам’яті сервера без використання централізованого сховища, відсутність CSRF-захисту, недостатня перевірка структури reCAPTCHA-токена, а також мінімальні вимоги до складності пароля. За результатами аналізу надано відповідні рекомендації щодо усунення вразливостей.

У результаті проведеного дослідження сформульовано такі висновки:

Досліджено актуальні загрози інформаційній безпеці в онлайн-банкінгу.

Було вивчено основні вектори атак, такі як фішинг, MITM, SQL-ін’єкції, XSS, перебір паролів, атаки на протоколи.

На основі критичного аналізу сформовано уявлення про типові вразливості банківських систем.

Це дозволило окреслити ключові напрями, в яких необхідне посилення захисту персональних даних.

Проведено порівняльний аналіз сучасних криптографічних технологій.

Проаналізовано принципи роботи та рівень стійкості симетричних (AES), асиметричних (RSA, ECC) і гібридних методів шифрування, а також протоколів TLS, HTTPS, VPN.

Здійснено порівняння їх ефективності в умовах реального банківського середовища.

Це підтвердило доцільність використання гібридного шифрування для оптимального балансу безпеки й продуктивності.

Розроблено захищений вебзастосунок Secure Bank — прототип «сайту-фортеці».

Реалізовано повноцінну автентифікацію за паролем і Passkey (WebAuthn), обмеження кількості спроб входу, перевірку через reCAPTCHA v3, хешування паролів, TLS, CSP, захищені сесії з контролем часу та IP-адреси.

Було враховано сучасні практики кіберзахисту та створено систему, наближену до реальних банківських сервісів.

Це дало змогу апробувати інтеграцію комплексних механізмів безпеки на прикладі повноцінного вебзастосунку.

Проведено тестування та аналіз вразливостей розробленої системи.

Здійснено мережевий аналіз за допомогою спеціалізованих інструментів (Wireshark, MITM-проксі), перевірено обробку сесій, трафік, захист від ботів і несанкціонованого доступу.

Виявлено низку слабких місць: збереження сесій у пам’яті сервера, відсутність CSRF-захисту, мінімальні вимоги до пароля, неповна перевірка reCAPTCHA.

Це дозволило розробити обґрунтовані рекомендації щодо підвищення рівня безпеки.

Сформульовано практичні рекомендації з удосконалення систем захисту персональних даних.

Запропоновано впровадження Redis для централізованого збереження сесій, розширення політик паролів, валідацію JWT-токенів, CSRF-захист, контроль IP-адрес для reCAPTCHA.

Ці заходи спрямовані на зниження ризиків соціальної інженерії, автоматизованого зламу та витоку інформації.

Проведено порівняння із наявними банківськими рішеннями.

Здійснено аналіз безпекової архітектури системи Приват24 як прикладу промислового рішення.

Це дало змогу оцінити відповідність створеного прототипу сучасним стандартам банківської кібербезпеки.

Порівняння із наявними банківськими рішеннями (на прикладі Приват24).

На відміну від Приват24, прототип Secure Bank побудований на Flask із використанням SQLite, що спрощує розгортання, але не забезпечує промислового рівня масштабування.

Secure Bank впроваджує WebAuthn (Passkey) — інноваційний метод автентифікації, який наразі не поширений у вітчизняних банках.

Прототип реалізує безпечне управління сесіями, обмеження на кількість входів, CSP, TLS — тобто використовує прозорі й наглядні механізми захисту, тоді як Приват24 має комплексну, але закриту архітектуру.

На відміну від Приват24, Secure Bank не має інтеграції з реальними платіжними системами та банківськими API, але надає дослідницьку платформу для апробації рішень у сфері безпеки.

Рекомендації щодо подальших досліджень

- Провести поглиблене вивчення постквантових алгоритмів шифрування з метою забезпечення стійкості до майбутніх загроз з боку квантових обчислень.
- Розробити та інтегрувати модулі штучного інтелекту для виявлення аномальної поведінки користувачів у режимі реального часу.
- Дослідити застосування поведінкової біометрії як інструменту адаптивної автентифікації в банківських вебсистемах.
- Вивчити ефективність використання Zero Trust-архітектури в інформаційних системах фінансових установ.

Таким чином, поставлену мету дослідження досягнуто: було здійснено порівняльний аналіз технологій шифрування, досліджено вразливості, реалізовано та протестовано захищений вебзастосунок, що наочно демонструє застосування сучасних методів захисту персональних даних.

Результати роботи можуть бути використані для подальших досліджень у галузі кібербезпеки, удосконалення практичних рішень у сфері фінансових онлайн-сервісів, а також як база для впровадження нових механізмів захисту в реальних інформаційних системах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. VPN Unlimited. Cybersecurity: Online banking security. URL: https://www.vpnunlimited.com/ua/help/cybersecurity/online-banking-security

2. General Data Protection Regulation (GDPR). Article 5. URL: https://gdpr-text.com/uk/read/article-5/

3. Що таке PCI DSS – стандарт безпеки даних індустрії платіжних карток. Юридична Газета. URL: https://yur-gazeta.com/dumka-eksperta/shcho-take-pci-dss-standart-bezpeki-danih-industriyi-platizhnih-kartok.html

4. Закон України «Про захист персональних даних» №2297-VI від 01.06.2010. URL: https://zakon.rada.gov.ua/laws/show/2297-17#Text

5. Закон України «Про платіжні послуги» №2155-IX від 30.09.2021. URL: https://zakon.rada.gov.ua/laws/show/2155-19#Text

6. Закон України «Про електронні комунікації» №1591-IX від 16.09.2021. URL: https://zakon.rada.gov.ua/laws/show/1591-20#Text

7. Національний банк України. Постанова №95 від 28.09.2017. URL: https://bank.gov.ua/ua/legislation/Resolution_28092017_95

8. Закон України «Про основні засади забезпечення кібербезпеки України» №2163-VIII від 05.10.2017. URL: https://zakon.rada.gov.ua/laws/show/2163-19#Text

9. Advanced Encryption Standard. Матеріал з Вікіпедії – вільної енциклопедії. URL: https://uk.wikipedia.org/wiki/Advanced_Encryption_Standard

10. RSA. Матеріал з Вікіпедії – вільної енциклопедії. URL: https://uk.wikipedia.org/wiki/RSA

11. Rescorla E. RFC 5246: The Transport Layer Security (TLS) Protocol Version 1.2. Internet Engineering Task Force (IETF), 2008.

12. Encryption Consulting. What is TLS/SSL? URL: https://www.encryptionconsulting.com/education-center/what-is-tls-ssl/

13. Mozilla Foundation. HTTP over TLS. MDN Web Docs, 2022. URL: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

14. OpenVPN Inc. OpenVPN Protocol Overview, 2023. URL: https://openvpn.net

15. The Tor Project. Tor: Overview, 2023. URL: https://www.torproject.org

16. National Institute of Standards and Technology (NIST). Advanced Encryption Standard (AES). FIPS 197, 2001.

17. Bernstein D. J. ChaCha, a variant of Salsa20, 2008. URL: http://cr.yp.to/chacha.html

18. Internet Engineering Task Force (IETF). RFC 8446: The Transport Layer Security (TLS) Protocol Version 1.3, 2018.

19. Shor P. W. Algorithms for quantum computation: Discrete logarithms and factoring. Proceedings of the 35th Annual Symposium on Foundations of Computer Science, 1994.

20. National Institute of Standards and Technology (NIST). Post-Quantum Cryptography Standardization, 2023. URL: https://csrc.nist.gov/projects/post-quantum-cryptography

21. Young Scientists in Computer Science & Software Engineering 2024 Investigating vulnerabilities of personal data on financial Websites Olena H. Fedorenko , Sofiia V. Velychko and Yelyzaveta V. Kaidan

22. Google Cloud. reCAPTCHA Overview. URL: https://cloud.google.com/recaptcha/docs/overview

23. Microsoft Digital Defense Report 2024. URL: https://www.microsoft.com/uk-ua/security/security-insider/intelligence-reports/microsoft-digital-defense-report-2024

24. Аніщук В.В. Проблема захисту персональних даних в кіберпросторі. Науковий вісник Ужгородського національного університету. Серія: Право. 2024. Вип. 84, ч. 3. С. 253-254.

25. Державна служба спеціального зв'язку та захисту інформації України. URL: https://cip.gov.ua/

26. ISO/IEC 27001:2022 – Information security, cybersecurity and privacy protection. URL: https://www.iso.org/isoiec-27001-information-security.html

27. OWASP (Open Web Application Security Project). URL: https://owasp.org/

28. Schneier B. Applied Cryptography: Protocols, Algorithms, and Source Code in C. Wiley, 2020.

29. Stallings W. Cryptography and Network Security: Principles and Practice. Pearson, 2022.

30. Ukrainian Cybersecurity Cluster Reports. URL: https://cyberdefense.in.ua/

31. Крюков Є.І., Якубовська Н.М. Інформаційна безпека в цифрову епоху: аспекти захисту персональних даних. Наукові праці НАУ. URL: https://jrnl.nau.edu.ua/

32. Національна стратегія кібербезпеки України (2021). URL: https://www.president.gov.ua/documents/4472021-40033

33. НБУ. Стратегія кібербезпеки банківської системи України – https://bank.gov.ua/

34. NIST SP 800-57: Recommendation for Key Management. URL: https://csrc.nist.gov/publications/detail/sp/800-57-part-1/rev-5/final

35. Hanko. URL: https://www.hanko.io/

ДОДАТОК А. Графічне представлення робочого процесу recaptcha

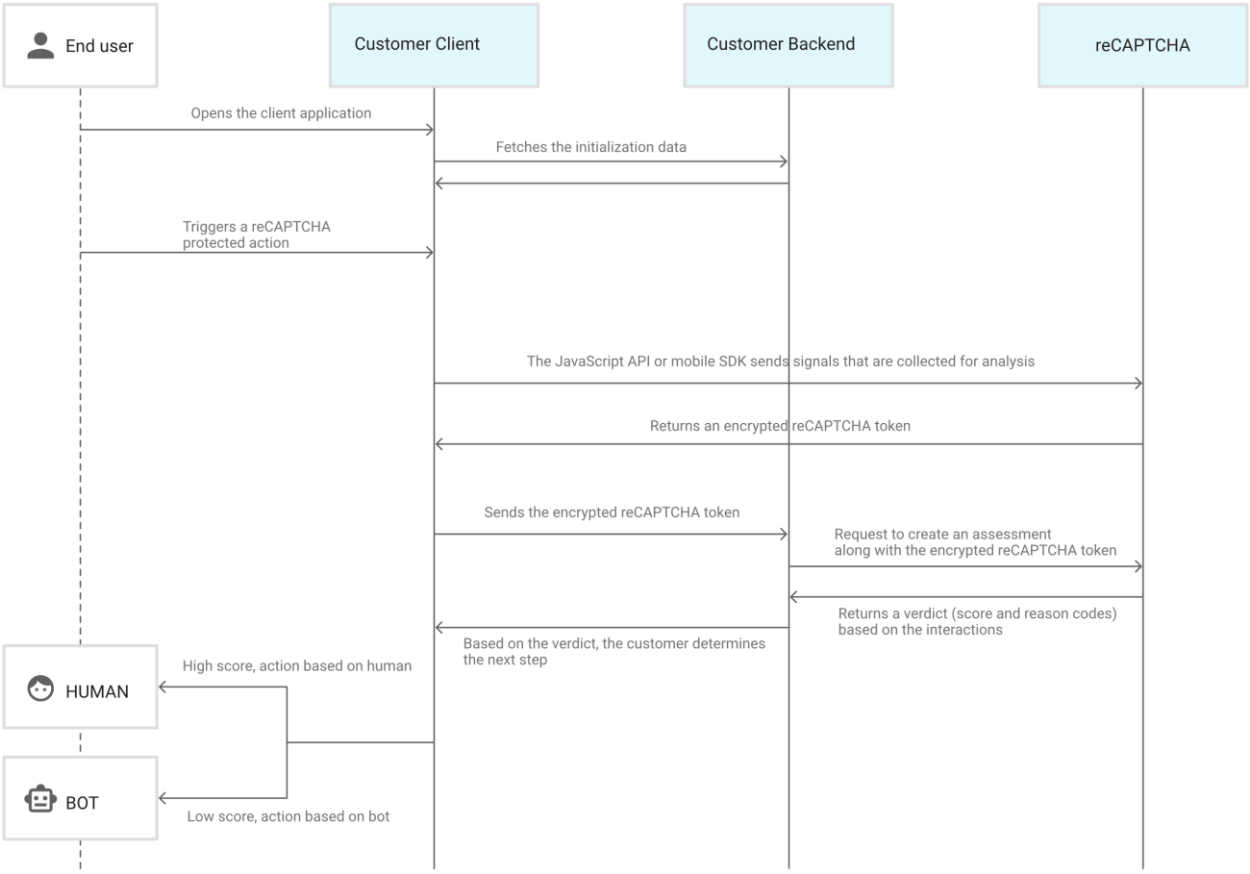

Рисунок А.1 - Графічне представлення робочого процесу reCAPTCHA

ДОДАТОК Б. Повний код реалізованого застосунку

Повний код app.py

```
from flask import Flask, request, jsonify, session
from flask_cors import CORS
from dotenv import load_dotenv
import os
import requests
import jwt
import sqlite3
import uuid
import secrets
from werkzeug.security import generate_password_hash, check_password_hash
from functools import wraps
from datetime import datetime, timedelta
import time

load_dotenv()

app = Flask(__name__)
CORS(app, supports_credentials=True, origins=["https://localhost", "http://localhost:3000", "https://localhost:3000"])
app.secret_key = secrets.token_hex(16)  # Безпечний секретний ключ

app.config.update(
    SESSION_COOKIE_SECURE=True,         # Кука буде передаватись лише по HTTPS
    SESSION_COOKIE_SAMESITE="None"    # Дозволити кросс-доменні запити
)

# Додаємо CSP-заголовок
@app.after_request
def add_security_headers(response):
    csp = (
        "default-src 'self'; "
        "script-src 'self' https://www.google.com https://www.gstatic.com; "
        "frame-src https://www.google.com; "
        "frame-ancestors https://www.google.com; "
        "connect-src 'self' https://www.google.com https://www.gstatic.com; "
        "img-src 'self' https://www.google.com https://www.gstatic.com; "
        "style-src 'self' https://www.google.com https://www.gstatic.com;"
    )
    response.headers['Content-Security-Policy'] = csp
    return response
```

```
# Путь до бази даних SQLite
DB_PATH = "users.db"

# Словар для відслідковування активних сесій
active_sessions = {}

login_attempts = {}  # {ip: [timestamp1, timestamp2, ...]}
MAX_ATTEMPTS = 5
BLOCK_WINDOW = 60  # 60 секунд

# reCAPTCHA Secret Key з .env
RECAPTCHA_SECRET_KEY = os.getenv("RECAPTCHA_SECRET_KEY")

# Функція для перевірки reCAPTCHA токена
def verify_recaptcha(token):
    if not token:
        return False, "reCAPTCHA token is missing"

    response = requests.post(
        "https://www.google.com/recaptcha/api/siteverify",
        data={
            "secret": RECAPTCHA_SECRET_KEY,
            "response": token,
        },
    )
    result = response.json()

    print("reCAPTCHA verification result:", result)  # Отладка

    # Перевіряємо успіх
    if not result.get("success"):
        error_codes = result.get("error-codes", [])
        return False, f"reCAPTCHA verification failed: {error_codes}"
    # Перевіряємо score
    if result.get("score", 0) < 0.3:
        return False, f"reCAPTCHA score too low: {result.get('score')}"

    return True, None

def init_db():
    with sqlite3.connect(DB_PATH) as conn:
        c = conn.cursor()
        # Створюємо таблицю users, якщо вона існує
        c.execute('''CREATE TABLE IF NOT EXISTS users (
            id TEXT PRIMARY KEY,
            email TEXT UNIQUE,
            password TEXT
        )''')
        # Створюємо таблицю для зберігання інформації про passkeys,
якщо потрібно
        c.execute('''CREATE TABLE IF NOT EXISTS user_passkeys (
            id TEXT PRIMARY KEY,
```

```
            user_id TEXT,
            credential_id TEXT UNIQUE,
            public_key TEXT,
            FOREIGN KEY (user_id) REFERENCES users (id)
        ''')
        conn.commit()
    print("Database initialized successfully.")

# Перевірка вимог до паролю
def is_valid_password(password):
    return (len(password) > 8 and
            any(c.isupper() for c in password) and
            any(c.islower() for c in password))

def cleanup_old_attempts(ip):
    now = time.time()
    login_attempts[ip] = [t for t in login_attempts.get(ip, []) if now
- t < BLOCK_WINDOW]

def is_ip_blocked(ip):
    cleanup_old_attempts(ip)
    return len(login_attempts.get(ip, [])) >= MAX_ATTEMPTS

def record_failed_attempt(ip):
    login_attempts.setdefault(ip, []).append(time.time())

@app.route('/')
def hello_world():
    return 'Hello, World!'

@app.route("/login", methods=["POST"])
def login_user():
    data = request.json
    email = data.get("email")
    password = data.get("password")
    recaptcha_token = data.get("recaptchaToken")

    ip = request.remote_addr
    if is_ip_blocked(ip):
        return jsonify({"message": "Забагато спроб. Зачекайте хвилину
та спробуйте ще раз."}), 429


    # Перевірка reCAPTCHA токен
    recaptcha_valid,                recaptcha_error                =
verify_recaptcha(recaptcha_token)
    if not recaptcha_valid:
        return jsonify({"message": recaptcha_error}), 403

    if not email or not password:
        return jsonify({"message": "Email and password are required"}),
400
```

```
    if not is_valid_password(password):
        return jsonify({"message": "Password must be >8 characters,
with upper and lower case letters"}), 400

    conn = sqlite3.connect(DB_PATH)
    c = conn.cursor()
    c.execute("SELECT * FROM users WHERE email = ?", (email,))
    user = c.fetchone()

    if user:
        if check_password_hash(user[2], password):
            # Генеруємо унікальний ID сесії
            session_id = str(uuid.uuid4())
            session["user_id"] = user[0]
            session["email"] = user[1]
            session["session_id"] = session_id  # Зберігаємо ID в сесії

            active_sessions[user[0]] = {
                "email": user[1],
                "last_activity": datetime.now(),
                "session_id":    session_id,        #    Використовуємо
згенерований ID
                "ip_address": request.remote_addr
            }
            user_info = {"id": user[0], "email": user[1]}
            conn.close()
            return  jsonify({"message":  "Login  successful",  "user":
user_info}), 200
        else:
            record_failed_attempt(ip)
            conn.close()
            return jsonify({"message": "Неправильний пароль"}), 401
    else:
        user_id = str(uuid.uuid4())
        hashed_password = generate_password_hash(password)
        c.execute("INSERT INTO users (id, email, password) VALUES (?,
?, ?)",
                  (user_id, email, hashed_password))
        conn.commit()

        # Генеруємо унікальний ID сесії для нового користувача
        session_id = str(uuid.uuid4())
        session["user_id"] = user_id
        session["email"] = email
        session["session_id"] = session_id

        active_sessions[user_id] = {
            "email": email,
            "last_activity": datetime.now(),
            "session_id": session_id,
            "ip_address": request.remote_addr
```

```
        }
        user_info = {"id": user_id, "email": email}
        conn.close()
        return jsonify({"message": "Registration and login successful",
"user": user_info}), 201

@app.route('/logout')
def logout_user():
    user_id = session.get('user_id')
    if user_id in active_sessions:
        # Видаляємо користувача з активних сесій
        del active_sessions[user_id]
    session.clear()
    return jsonify({"message": "You have been logged out"}), 200

# Оновлюємо декоратор для перевірки активних сесій
def login_required(f):
    @wraps(f)
    def decorated_function(*args, **kwargs):
        user_id = session.get('user_id')
        session_id = session.get('session_id')

        # Перевіряємо наявність користувача в активних сесіях
        if not user_id or user_id not in active_sessions:
            return jsonify({"message": "Unauthorized - please login"}),
401

        # Перевіряємо відповідність ID сесії
        if active_sessions[user_id]['session_id'] != session_id:
            return jsonify({"message": "Session mismatch - please login
again"}), 401

        # Перевіряємо час останньої активності
        if datetime.now() - active_sessions[user_id]['last_activity']
> timedelta(minutes=30):
            del active_sessions[user_id]
            session.clear()
            return jsonify({"message": "Session expired - please login
again"}), 401

        # Оновлюємо час останньої активності
        active_sessions[user_id]['last_activity'] = datetime.now()

        return f(*args, **kwargs)
    return decorated_function

#### Passkeys
tenant_id = os.getenv("PASSKEY_TENANT_ID")
api_key = os.getenv("PASSKEY_API_KEY")

baseUrl = f"https://passkeys.hanko.io/{tenant_id}"
headers = {
```

```
    "apikey": api_key,
    "Content-Type": "application/json",
}

@app.route('/passkey/start-registration', methods=["POST"])
def start_registration():
    if 'user_id' not in session:
        return jsonify({"error": "User must be logged in to register a
passkey"}), 401

    user_id = session['user_id']
    user_email = session['email']

    payload = {"user_id": user_id, "username": user_email}
    response   =   requests.post(f"{baseUrl}/registration/initialize",
headers=headers, json=payload)
    creationOptions = response.json()
    return jsonify(creationOptions)

@app.route("/passkey/finalize-registration", methods=["POST"])
def finalize_registration():
    data = request.json
    response    =    requests.post(f"{baseUrl}/registration/finalize",
headers=headers, json=data)

    if response.status_code != 200:
        return  jsonify({"message":  "Passkey  registration  failed",
"error": response.json()}), response.status_code

    user_id = session.get("user_id")
    email = session.get("email")

    if not user_id or not email:
        return  jsonify({"message":  "Session  expired  -  please  login
again"}), 401

    # Оновлюємо сесію без створення нової (зберігаємо поточний
session_id)
    active_sessions[user_id].update({
        "last_activity": datetime.now(),
        "auth_method": "passkey"
    })

    return jsonify({
        "message": "Passkey registered successfully",
        "user": {
            "id": user_id,
            "email": email
        }
    }), 200

@app.route("/passkey/start-login", methods=["POST"])
```

```
def start_login():
    data = request.json
    recaptcha_token = data.get("recaptchaToken")

    # Перевіряємо reCAPTCHA токен
    recaptcha_valid,                recaptcha_error                =
verify_recaptcha(recaptcha_token)
    if not recaptcha_valid:
        return jsonify({"message": recaptcha_error}), 403

    response      =      requests.post(f"{baseUrl}/login/initialize",
headers=headers)
    login_options = response.json()
    return jsonify(login_options)

@app.route("/passkey/finalize-login", methods=["POST"])
def finalize_login():
    data = request.json
    recaptcha_token = data.get("recaptchaToken")

    # Перевіряємо reCAPTCHA токен
    recaptcha_valid,                recaptcha_error                =
verify_recaptcha(recaptcha_token)
    if not recaptcha_valid:
        return jsonify({"message": recaptcha_error}), 403

    response       =       requests.post(f"{baseUrl}/login/finalize",
headers=headers, json=data)
    data = response.json()

    if response.status_code != 200:
        return jsonify({"message": "Failed to finalize login", "error":
data}), response.status_code

    token = data.get('token')
    if not token:
        return jsonify({"message": "Token not found in response",
"error": data}), 400

    decoded_payload           =           jwt.decode(token.encode(),
options={"verify_signature": False})
    user_id = decoded_payload.get('sub')

    conn = sqlite3.connect(DB_PATH)
    c = conn.cursor()
    c.execute("SELECT * FROM users WHERE id = ?", (user_id,))
    user = c.fetchone()
    conn.close()

    if user:
        # Генеруємо новий ID сесії
        session_id = str(uuid.uuid4())
```

```
        session["user_id"] = user[0]
        session["email"] = user[1]
        session["session_id"] = session_id

        active_sessions[user[0]] = {
            "email": user[1],
            "last_activity": datetime.now(),
            "session_id": session_id,
            "ip_address": request.remote_addr,
            "auth_method": "passkey"
        }
        user_info = {"id": user[0], "email": user[1]}
        return jsonify({"message": "Login successful", "user":
user_info}), 200
    else:
        return jsonify({"message": "Invalid credentials"}), 401

@app.route('/dashboard')
@login_required
def dashboard():
    return jsonify({
        "message": "Welcome to dashboard",
        "user": {
            "id": session["user_id"],
            "email": session["email"]
        }
    })

if __name__ == '__main__':
    init_db()
    app.run(host='0.0.0.0', port=443, ssl_context=('server.crt',
'server.key'))
```

Повний код dashboard.tsx

```
import { useEffect, useState } from "react";
import { useNavigate } from "react-router-dom";
import { create, type CredentialCreationOptionsJSON } from
"@github/webauthn-json";
import { Button } from "@/components/ui/button";
import { toast } from "sonner";

const Dashboard = () => {
    const navigate = useNavigate();
    const [isLoading, setIsLoading] = useState(true);
    const [userEmail, setUserEmail] = useState(""); // ← ця строка
обов'язкова!

    // Перевіряємо аутентифікацію при завантаженні компонента
    useEffect(() => {
        const checkAuth = async () => {
            try {
```

```
                const response = await
fetch('http://localhost:8000/dashboard', {
                    method: 'GET',
                    credentials: 'include'
                });

                if (!response.ok) {
                    throw new Error('Не автентифіковано');
                }

                const data = await response.json();
                setUserEmail(data.user.email);  // <- тут зберігаємо
email
                setIsLoading(false);

                //    Якщо    відповідь    позитивна,    користувача
автентифіковано
                setIsLoading(false);
            } catch (error) {
                toast.error('Будь ласка, авторизуйтесь, щоб отримати
доступ до Secure Bank');
                navigate('/login');
            }
        };

        checkAuth();
    }, [navigate]);

    const handleLogout = async () => {
        try {
            const response = await
fetch('http://localhost:8000/logout', {
                credentials: 'include',
            });

            if (response.ok) {
                navigate('/login');
                toast.success('Вихід успішно виконано');
            } else {
                const data = await response.json();
                toast.error(data.message || 'Вихід не вдалось
виконати');
            }
        } catch (error) {
            toast.error('Помилка виходу');
            console.error('Logout error:', error);
        }
    };

    async function registerPasskey() {
        try {
            const createOptionsResponse = await fetch(
```

```
                "http://localhost:8000/passkey/start-registration",
                {
                    method: "POST",
                    headers: { "Content-Type": "application/json" },
                    credentials: 'include',
                }
            );

            if (!createOptionsResponse.ok) {
                throw new Error('Не вдалося розпочати реєстрацію
passkey');
            }

            const createOptions = await createOptionsResponse.json();
            const credential = await create(createOptions as
CredentialCreationOptionsJSON);

            const response = await fetch(
                "http://localhost:8000/passkey/finalize-
registration",
                {
                    method: "POST",
                    headers: { "Content-Type": "application/json" },
                    credentials: "include",
                    body: JSON.stringify(credential),
                }
            );

            if (response.ok) {
                toast.success("Passkey успішно зареєстровано!");
            } else {
                throw new Error('Не вдалось зареєструвати passkey');
            }
        } catch (error) {
            toast.error("Не вдалось зареєструвати passkey");
            console.error("Помилка реєстрації Passkey :", error);
        }
    }

    if (isLoading) {
        return <div className="p-4">Loading...</div>;
    }

    return (
        <div className="p-4 flex justify-center items-center min-h-
screen">
            <div className="text-center space-y-6">

                <img
                src="/icon.png"
                alt="Icon"
```

```
                
                <h1 className="text-2xl font-bold mb-4">Головна сторінка</h1>
                <h2 className="text-2xl font-bold mb-4">Secure Bank</h2>
                <p className="mb-6">Вітаємо, {userEmail}!</p>
                <Button
                    onClick={handleLogout}
                    className="px-6 py-3 text-lg w-full max-w-sm mx-auto"
                >
                    Вийти
                </Button>
                <div>
                    <Button
                        onClick={registerPasskey}
                        className="px-6 py-3 text-lg w-full max-w-sm mx-auto"
                    >
                        Зареєструвати новий passkey
                    </Button>
                </div>
            </div>
        </div>
    );
};

export default Dashboard;
```

Повний код home.tsx

```
import { Link } from 'react-router-dom';

const Home = () => {
    return (
        <div className="flex flex-col items-center justify-center min-h-screen">
            
            <h1 className="text-2xl font-bold mb-4">Secure Bank</h1>
            <Link to="/login" className="px-4 py-2 bg-black text-white rounded-md transition duration-150 ease-in-out">Увійти або зареєструватись</Link>
        </div>
    );
}
```

```
export default Home;
```

Повний код login.tsx

```
import { Button } from "@/components/ui/button";
import { Card, CardContent, CardDescription, CardHeader, CardTitle }
from "@/components/ui/card";
import { Input } from "@/components/ui/input";
import { Label } from "@/components/ui/label";
import useLogin from "@/hooks/useLogin";
import { useState, useRef } from "react";
import { useNavigate } from "react-router-dom";
import { get } from "@github/webauthn-json";
import ReCAPTCHA from "react-google-recaptcha";

const Login = () => {
    const navigate = useNavigate();
    const { loginUser, isLoading } = useLogin();
    const [email, setEmail] = useState('');
    const [password, setPassword] = useState('');
    const [error, setError] = useState('');
    const [passkeyError, setPasskeyError] = useState('');
    const [recaptchaError, setRecaptchaError] = useState('');
    const recaptchaRef = useRef<ReCAPTCHA>(null);

    const executeRecaptcha = async () => {
        setRecaptchaError('');
        try {
            const token = await recaptchaRef.current?.executeAsync();

            if (!token) {
                setRecaptchaError('Не    вдалось    отримати    токен
reCAPTCHA');
                return null;
            }
            return token;
        } catch (err) {
            console.error("reCAPTCHA error:", err);
            setRecaptchaError('Помилка зававнтаження reCAPTCHA');
            return null;
        }
    };

    const         handleSubmit         =         async         (event:
React.FormEvent<HTMLFormElement>) => {
        event.preventDefault();
        setError('');
        setRecaptchaError('');

        // Запрашуємо токен reCAPTCHA
        const token = await executeRecaptcha();
```

```
        if (!token) {
            setRecaptchaError('Будь ласка, підтвердьте, що ви не
робот');
            return;
        }

        const payload = { email, password, recaptchaToken: token };


        try {
            const success = await loginUser(email, password, token);
            if (success) {
                navigate('/dashboard');
            }
        } catch (err) {
            setError(err instanceof Error ? err.message : 'Помилка при
вході');
        } finally {
            // Збрасуємо reCAPTCHA
            recaptchaRef.current?.reset();
        }
    };

    async function signInWithPasskey() {
        setPasskeyError('');
        setRecaptchaError('');

        // Запрашуємо токен для start-login
        const startToken = await executeRecaptcha();
        if (!startToken) {
            setRecaptchaError('Будь ласка, підтвердьте, що ви не
робот');
            return;
        }

        const startPayload = { start: true, finish: false, credential:
null, recaptchaToken: startToken };


        try {
            const createOptionsResponse = await
fetch("http://localhost:8000/passkey/start-login", {
                method: "POST",
                headers: { "Content-Type": "application/json" },
                credentials: 'include',
                body: JSON.stringify(startPayload),
            });

            if (!createOptionsResponse.ok) {
                const errorData = await createOptionsResponse.json();
                throw new Error(errorData.message || 'Не вдалось почати
вхід з Passkey');
```

```
            }

            const loginOptions = await createOptionsResponse.json();
            const options = await get(loginOptions as any);

            // Запрашуємо новий токен для finalize-login
            const finalizeToken = await executeRecaptcha();
            if (!finalizeToken) {
                setRecaptchaError('Будь ласка, підтвердьте, що ви не
робот');
                return;
            }

            const finalizePayload = { ...options, recaptchaToken:
finalizeToken };


            const response = await
fetch("http://localhost:8000/passkey/finalize-login", {
                method: "POST",
                headers: { "Content-Type": "application/json" },
                credentials: 'include',
                body: JSON.stringify(finalizePayload),
            });

            if (!response.ok) {
                const errorData = await response.json();
                throw new Error(errorData.message || 'Помилка при
завершенні входу з Passkey');
            }

            navigate("/dashboard");
        } catch (err) {
            setPasskeyError(err instanceof Error ? err.message :
'Помилка при вході з Passkey');
        } finally {
            // Збрасуємо reCAPTCHA
            recaptchaRef.current?.reset();
        }
    }

    return (
        <div className="w-screen h-screen flex items-center justify-
center">
            <Card className="w-full max-w-lg">
                <CardHeader>
                    <CardTitle>Вхід у систему</CardTitle>
                    <CardDescription>Оберіть спосіб
входу</CardDescription>
                </CardHeader>
                <CardContent>
                    <div className="flex flex-col gap-4">
```

```
                        {/* Блок для помилок email/password */}
                        {error && (
                            <div className="p-3 bg-red-50 text-red-700
rounded-md text-sm">
                                {error}
                            </div>
                        )}

                        {/* Блок для помилок reCAPTCHA */}
                        {recaptchaError && (
                            <div className="p-3 bg-red-50 text-red-700
rounded-md text-sm">
                                {recaptchaError}
                            </div>
                        )}

                        <form                  onSubmit={handleSubmit}
className="space-y-4">
                            <div className="space-y-2">
                                <Label>Електронна пошта</Label>
                                <Input
                                    id="email"
                                    required
                                    name="email"
                                    type="email"
                                    value={email}
                                    onChange={(e)                    =>
setEmail(e.target.value)}
                                    autoComplete="email"
                                    disabled={isLoading}
                                />
                            </div>
                            <div className="space-y-2">
                                <Label>Пароль</Label>
                                <Input
                                    id="password"
                                    name="password"
                                    type="password"
                                    value={password}
                                    onChange={(e)                    =>
setPassword(e.target.value)}
                                    autoComplete="current-password"
                                    disabled={isLoading}
                                />
                            </div>
                            <Button  type="submit"  className="w-full"
disabled={isLoading}>
                                {isLoading   ?   'Завантаження...'   :
'Увійти з електронною поштою'}
                            </Button>
                        </form>
```

```
                        <div className="relative">
                            <div  className="absolute  inset-0  flex
items-center">
                                <span className="w-full border-t" />
                            </div>
                            <div  className="relative  flex  justify-
center text-xs uppercase">
                                <span  className="bg-background  px-2
text-muted-foreground">
                                    Или
                                </span>
                            </div>
                        </div>

                        {/* Блок для помилок passkey */}
                        {passkeyError && (
                            <div className="p-3 bg-red-50 text-red-700
rounded-md text-sm">
                                {passkeyError}
                            </div>
                        )}

                        <Button
                            variant="outline"
                            className="w-full"
                            onClick={signInWithPasskey}
                            disabled={isLoading}
                        >
                            Увійти з Passkey
                        </Button>

                        {/* reCAPTCHA v3 (невидима) */}
                        <div className="mt-4">
                            <ReCAPTCHA
                                ref={recaptchaRef}

sitekey="6LdC5xUrAAAAAL_3RRGFOs41gLlmSTL_8m5p0aVT" // Ваш ключ
                                size="invisible"
                                onChange={() => {}} // Порожній колбек
                            />
                        </div>
                    </div>
                </CardContent>
            </Card>
        </div>
    );
};

export default Login;
```