МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
Ігор ШЕЛЕХОВ
(підпис)
16 грудня 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня магістр

зі спеціальності 122 - Комп'ютерних наук,
освітньо-професійної програми «Інформатика»
на тему: «Інформаційна технологія проектування сервісу для обміну файлами без ідентифікації учасників процесу»
здобувача групи ІН.м - 23 Захлєбаєва Дениса Станіславовича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

Денис ЗАХЛЄБАЄВ
(підпис)

Керівник, старший викладач
Анна БАДАЛЯН
(підпис)

Суми – 2023

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня магістра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН.м-23 Захлєбаєва Дениса Станіславовича

1. Тема роботи: «Інформаційна технологія проектування сервісу для обміну файлами без ідентифікації учасників процесу»
затверджую наказом по СумДУ від *«06» грудня 2023 року № 1412-VI*

2. Термін здачі здобувачем кваліфікаційної роботи *до 16 друдня 2023 року*

3. Вхідні дані до кваліфікаційної роботи

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Аналіз проблеми предметної області, постановка й формування завдань дослідження. 2) Огляд технологій, що використовуються для шифрування, функцій обмінів файлів. 3) Розробка сервісу для обміну файлами без ідентифікації учасників процесу. 4) Аналіз результатів.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «_06_» ____листопада_____ 20 23___ р.

Завдання прийняв до виконання _____ Керівник _____
(підпис) (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Аналіз проблеми предметної області, постановка й формування завдань дослідження*	08.12.2023	
2	*Огляд технологій, що використовуються для шифрування, функцій обмінів файлів.*	09.12.2023	
3	*Розробка сервісу для обміну файлами без ідентифікації учасників процесу*	10.12.2023	
4	*Аналіз отриманих результатів*	15.12.2023	
5	*Оформлення пояснювальної записки до кваліфікаційної роботи*	16.12.2023	

Здобувач вищої освіти _____ Керівник _____

(підпис) (підпис)

АНОТАЦІЯ

Записка: 55 стор., 13 рис., 1 додаток, 27 джерел.

Обґрунтування актуальності теми роботи: Тема кваліфікаційної роботи є актуальною, оскільки присвячена розв’язанню важливої задачі створення деперсоналізованого файлообміннику шляхом розробки відповідних методів та інформаційної технології.

Об’єкт дослідження — Процес обміну файлами без ризику порушення приватності чи компрометації даних.

Мета роботи — створення сервісу для обміну файлами без ідентифікації учасників процесу

Результати — *розроблено сервісу для обміну файлами без ідентифікації учасників процесу. Створений сервіс зручний у користуванні, дозволяє відправляти та отримувати файли. Розробка проводилась на базі мови програмування JavaScript та з використанням Node.JS, Socket.IO, Express, WebCryptoAPI . У ході тестування проблем не виявлено.*

JS, NODE.JS, ECB, CBC, CFB, OFB, CRC32, Веб-програмування

ЗМІСТ

ВСТУП

Обґрунтування вибору теми роботи

Актуальність. У сучасному цифровому світі, де обмін інформацією став необхідністю, питання безпеки та приватності даних на першому плані. За останні роки, зростання використання платформ для файлообміну та хмарного зберігання призвело до необхідності вирішення проблеми збереження конфіденційності учасників цього обміну. Існуючі рішення, вимагаючи особистої ідентифікації, стають обмеженням у бажанні зберегти анонімність при передачі даних. Тож актуальність дослідження полягає в пошуку інноваційних підходів до обміну файлами, що забезпечують не лише безпеку, але й високий рівень анонімності учасників. Прагнення створити технологію, яка поєднує у собі зручність та безпеку обміну, залишаючи при цьому особисту ідентифікацію за межами обміну інформацією, стає справжнім завданням в епоху постійного зростання цифрової активності.

Об’єкт дослідження. Процес обміну файлами без ризику порушення приватності чи компрометації даних.

Предмет дослідження. Предметом дослідження є алгоритми та методи, які лежать в основі технології, яка забезпечить безпеку передачі даних, не ризикуючи втратою особистої ідентифікації, що має потенціал перетворити підходи до обміну інформацією в цифровому світі.

Гіпотеза. Гіпотеза цієї роботи полягає в припущенні та переконанні, що розроблена інформаційна технологія забезпечить оптимальний баланс між функціональністю та захистом особистої інформації під час обміну файлами. А саме, що інноваційний підхід до обробки даних та управління ними забезпечить не лише високий рівень безпеки, але й збереже анонімність учасників обміну інформацією. Гіпотетично, така технологія стане

відповіддю на потребу сучасного світу в безпечному та конфіденційному обміні даними, відкриваючи нові можливості для ефективної цифрової взаємодії без обмежень приватності.

Новизна. Новизна цього дослідження полягає в підході до розв'язання актуальної проблеми безпеки та приватності в цифровому обміні інформацією. Робота не лише аналізує існуючі платформи та технології, але й пропонує новаторське рішення, що враховує якість обміну файлами разом із збереженням конфіденційності учасників. Створення цієї інформаційної технології відкриває нові перспективи для безпечного обміну даними без потреби в особистій ідентифікації, що робить її унікальною у сфері цифрової безпеки та приватності."

Структура. Дане робота складається зі вступу, аналітичного огляду, постановки задачі, вибір методу розв’язання поставленої задачі, опису програмного забезпечення інформаційної системи, висновків, списку використаних джерел та додатків.

В еру стрімкого розвитку інформаційних технологій, обмін файлами став найважливішою частиною повсякденного життя, проникаючи в усі сфери діяльності від особистих взаємодій до корпоративного світу. Однак, у цій епохальній гонитві за ефективністю та зручністю, питання безпеки та приватності даних зазнають серйозних викликів.

Сучасні платформи файлообмінників і хмарного зберігання даних, як-от Google Drive, Dropbox і OneDrive, стали невід'ємною частиною нашого цифрового суспільства, надаючи широкі можливості для обміну інформацією. Однак, ці рішення, що вимагають ідентифікації користувача, часто не задовольняють вимогам приватності та анонімності учасників. Це ставить під

сумнів безпеку передачі даних, особливо в разі необхідності обміну інформацією без розкриття особистої ідентифікації.

У рамках цієї роботи здійснюється глибокий аналіз різних категорій готових рішень, починаючи від популярних файлообмінників до інноваційних технологій Big Data. При цьому основна увага приділяється пошуку і створенню інформаційної технології, яка поєднує в собі зручність обміну файлами з високим рівнем анонімності та безпеки. Такий підхід відкриває нові горизонти для обміну інформацією, виключаючи необхідність особистої ідентифікації, зберігаючи водночас безпеку та конфіденційність даних.

Метою роботи є створення сервісу для обміну файлами без ідентифікації учасників процесу.

Для досягнення мети були поставлено наступні завдання:

- Зробити огляд схожих за функціоналом сервісів
- Провести аналіз сервісів для обміну файлами та знайти шляхи для створення більш зручного та ефективного аналогу
- Розглянути технічні рішення які не можуть підійти для розробки такого сервісу

1 ОГЛЯД СХОЖИХ ЗА ФУНКЦІОНАЛОМ СЕРВІСІВ

Перш ніж розробити власний сервіс для обміну файлами, потрібно ознайомитися з самим контекстом приватності а потім перейти до аналогів сервісів.

1.1 Контекст приватності

Приватність у сфері цифрових даних стала одним із найактуальніших питань нашого часу. У нашу технологічно насичену та інформаційну епоху, коли дані відіграють ключову роль у повсякденному житті, забезпечення приватного та конфіденційного опрацювання інформації стало невід'ємною частиною обговорень громадської політики, законодавства та поведінки споживачів.

Однак, виникають серйозні виклики з приводу захисту приватності. Відсутність чітких правил і нормативних рамок часом призводить до ситуації, коли приватні дані стають вразливими перед збором, використанням і навіть незаконною передачею. Люди, за своєю суттю, стикаються з дилемою, де потреба в послугах і технологіях, що використовують особисті дані, часто суперечить потребі в збереженні приватності та контролі над своєю інформацією.

Саме в цьому контексті стає важливим проаналізувати будь-які конкретні аспекти, що стосуються приватності, чи то дії великих технологічних компаній, чи то державне регулювання, чи то заходи безпеки у зв'язку з терористичними погрозами, чи то навіть приклади компаній, що зливають дані за запитом. Оцінка цих аспектів допоможе краще зрозуміти, як сучасні дії та тенденції впливають на приватність користувачів і суспільство загалом.

1.1.2 BigTech компанії та їх продукти

Великі технологічні компанії, включно з гігантами на кшталт Google (з їх Google Drive), Facebook, Amazon та інших, зіткнулися з низкою проблем у сфері приватності даних. [1]

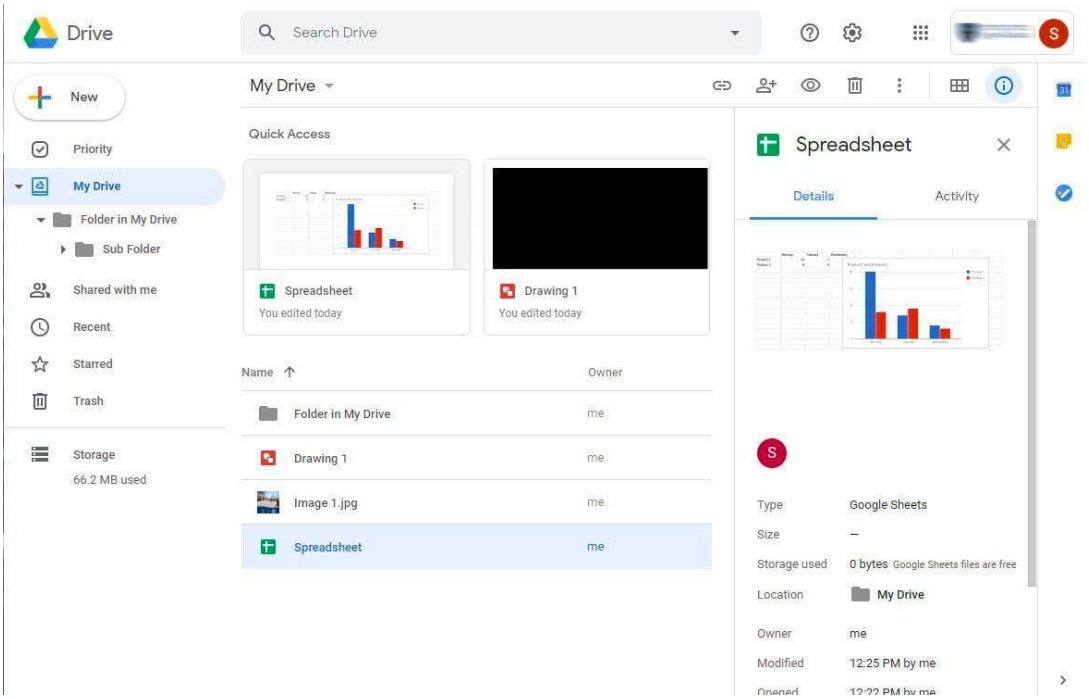

Рисунок 1.1 - Зовнішній вигляд сервісу для обміну файлами Google Drive

Ці організації часто стикаються зі звинуваченнями в недостатньому контролі за особистою інформацією користувачів і недозволеному використанні даних у комерційних цілях. Збір і аналіз великих обсягів даних став звичайною практикою для Bigtech, що викликає питання про те, наскільки ці компанії захищають конфіденційність користувачів. Відсутність прозорості у використанні даних і можливість непрозорих алгоритмів аналізу

може призвести до небажаного використання особистої інформації, що зі свого боку створює загрозу приватності користувачів.

Для багатьох людей, Bigtech стали символом проблеми у сфері приватності. Зростаюча тривога щодо витоку даних, використання особистої інформації для рекламних цілей або навіть продажу даних третім сторонам підкреслює необхідність посилення правил і нормативних вимог щодо захисту даних, а також підвищення прозорості та відповідальності з боку цих компаній.

1.1.3 Приватність у контексті тероризму

Останніми роками терористичні загрози привернули увагу до обговорення питань приватності.

Влада та правоохоронні органи часто аргументують необхідність розкриття або моніторингу особистих даних як інструменту боротьби з тероризмом та запобігання злочинам [2].

Однак, втручання в приватне життя громадян для цілей безпеки викликає дебати про баланс між забезпеченням безпеки суспільства і захистом особистої свободи. Запити на доступ до особистих даних, включно з листуванням, місцезнаходженням або іншими чутливими відомостями, часто викликають побоювання з приводу можливого зловживання цією інформацією та загрози приватності громадян. Інший аспект пов'язаний з етичними та законодавчими питаннями: де провести межу між необхідним моніторингом для безпеки та захистом приватності цивільних? Це питання залишається предметом дискусій у суспільстві, потребуючи уважного розгляду в рамках правових і нормативних рамок, з урахуванням балансу між безпекою та захистом особистих даних.

1.1.4 Приватність у контексті держрегулювання

Державні органи вводять заходи регулювання, які мають прямий вплив на дотримання приватності даних.

Такі заходи часто зобов'язують компанії та провайдерів інтернету надавати доступ до особистої інформації за запитом правоохоронних органів, що може охоплювати перехоплення та аналіз повідомлень, даних про місцезнаходження та інші особисті відомості.

Однак, такі вимоги викликають занепокоєння серед захисників приватності з приводу можливого порушення особистого життя та витоку конфіденційної інформації. Відсутність чітких правил і механізмів захисту даних у рамках таких регуляцій може створити ризик зловживання доступом до особистих даних. Системи, подібні до СОРМ, є частиною таких регуляцій і викликають питання про прозорість, контроль за використанням персональної інформації та забезпечення її захисту. Обговорення про баланс між безпекою держави та захистом особистого життя громадян залишаються важливими в контексті державного регулювання та його впливу на приватність даних.

1.1.5 TREEMA

Компанії, подібні до THREEMA [3] – які позіціонують себе як такі, комунікація всередині яких є дуже захищеною, стикаються з викликами, пов'язаними із забезпеченням приватності користувацьких даних.

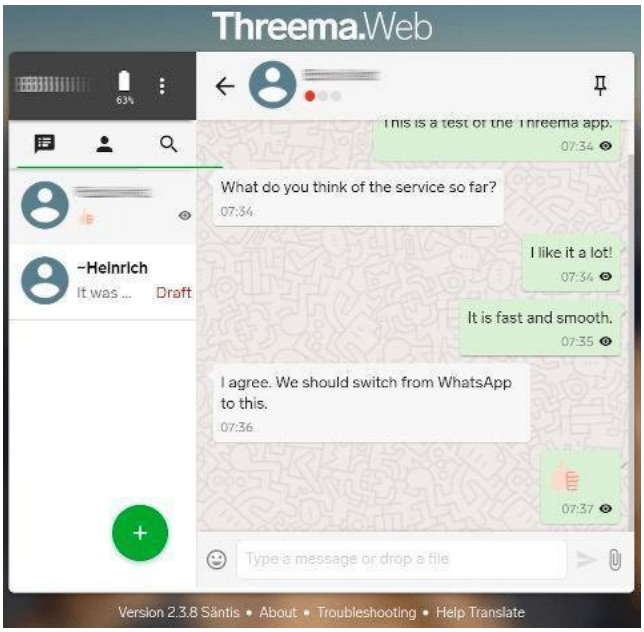

Рисунок 1.2 - Зовнішній вигляд месенджеру THREEMA

Їхня політика конфіденційності та звіти про прозорість стають ключовими інструментами для демонстрації їхньої турботи про захист особистої інформації. Однак, коли йдеться про надання даних за запитом правоохоронних органів (наприклад, хеш номера телефону), це викликає побоювання з приводу можливого витоку конфіденційних відомостей користувачів.

Вимога надання даних за запитом може мати потенційні наслідки для приватності, оскільки навіть хеш номера телефону може бути використаний для ідентифікації особи, особливо в контексті великих баз даних. Це підкреслює важливість встановлення суворих механізмів захисту даних, забезпечення узгодженості із законодавством про захист даних і прозорості в поводженні з особистою інформацією.

1.2. Файлообмінники

Файлообмінники являють собою важливу частину ринку інформаційних технологій, однак, не всі з них відповідають вимогам щодо безпеки та комфорту користувача.

Proton Drive та інші рішення на ринку, такі як We Transfer, Dropbox, SendGB, Firefox Send, надають можливість обміну файлами без ідентифікації учасників. Вони часто забезпечують конфіденційність і шифрування даних, що важливо для безпеки. Однак, деякі з цих рішень мають обмеження, які можуть знижувати комфорт користувача.

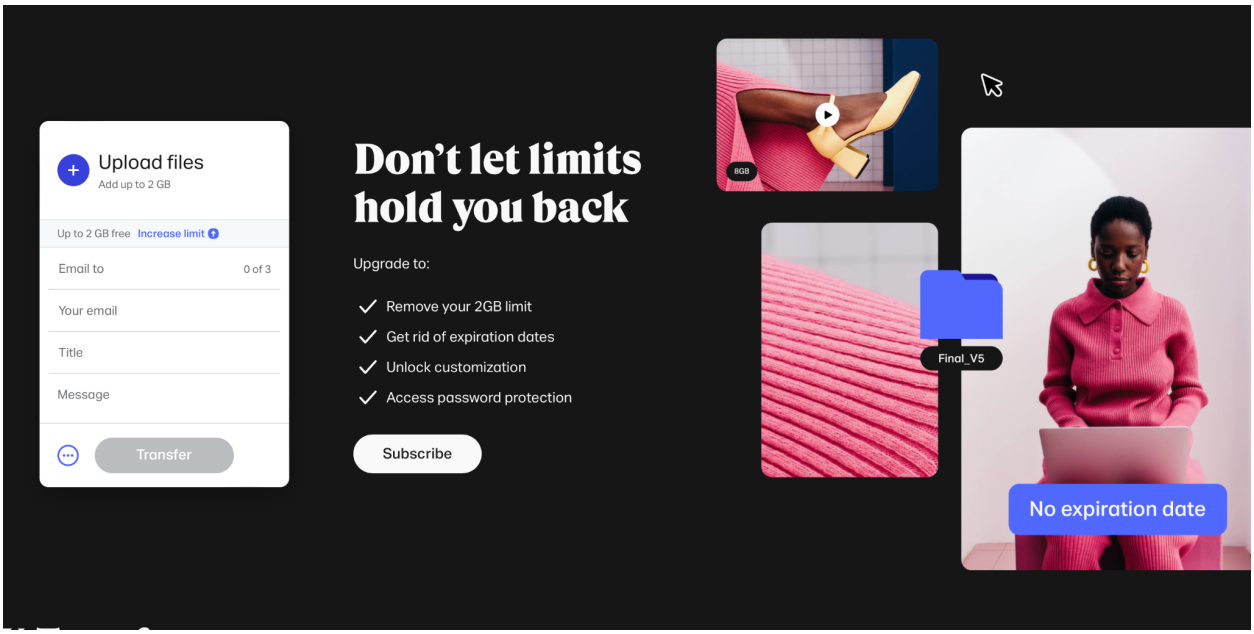

Рисунок 1.3 - Зовнішній вигляд сервісу для обміну файлами WeTransfer

Наприклад, WeTransfer [4], що є популярною платформою для обміну файлами, що дає змогу користувачам надсилати файли до 2 ГБ безкоштовно. Це забезпечує відносно простий і швидкий спосіб передачі даних без необхідності реєстрації. Однак обмеження за розміром файлу може виявитися недостатнім для користувачів, яким потрібна передача великих обсягів даних.

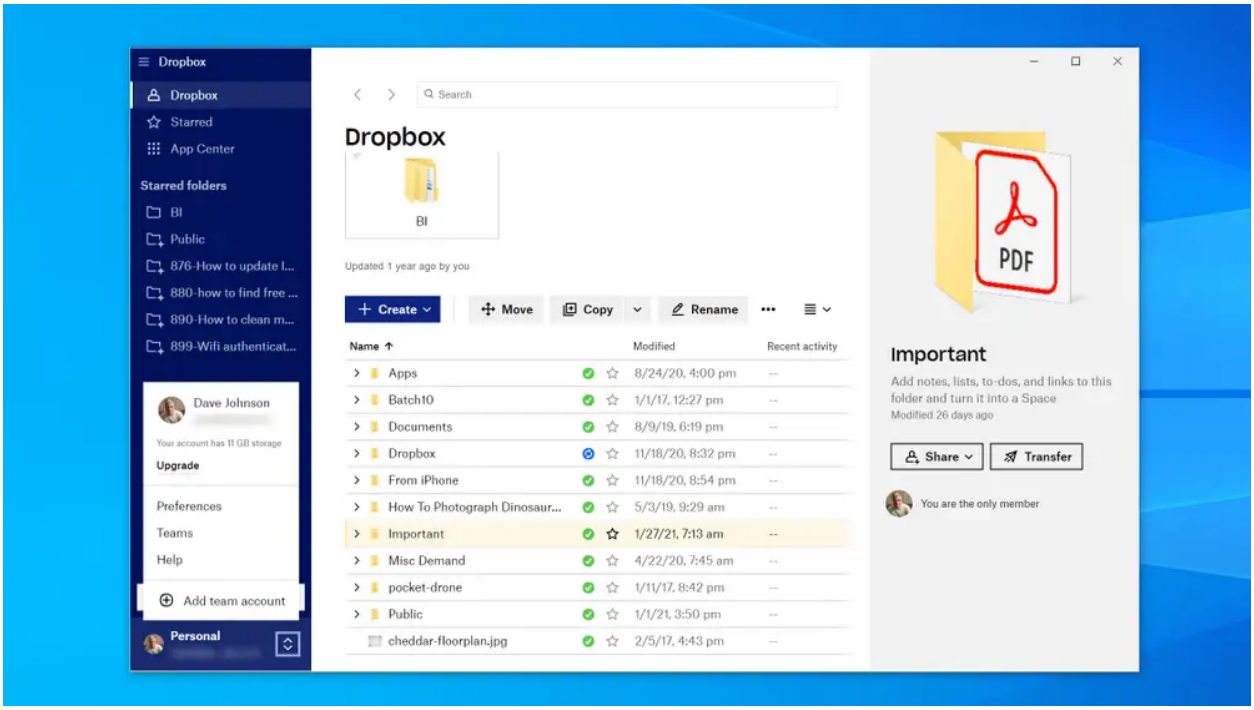

Рисунок 1.4 - Зовнішній вигляд сервісу для обміну файлами Dropbox

Також можемо розглянути сервіс Dropbox [5], який надає можливість обмінюватися файлами до 100 ГБ для користувачів із платною підпискою. Однак для отримання файлу одержувач повинен мати обліковий запис Dropbox або вводити свою електронну пошту для скачування. Це може створювати незручності для користувачів, які не знайомі з платформою Dropbox або не бажають реєструватися.

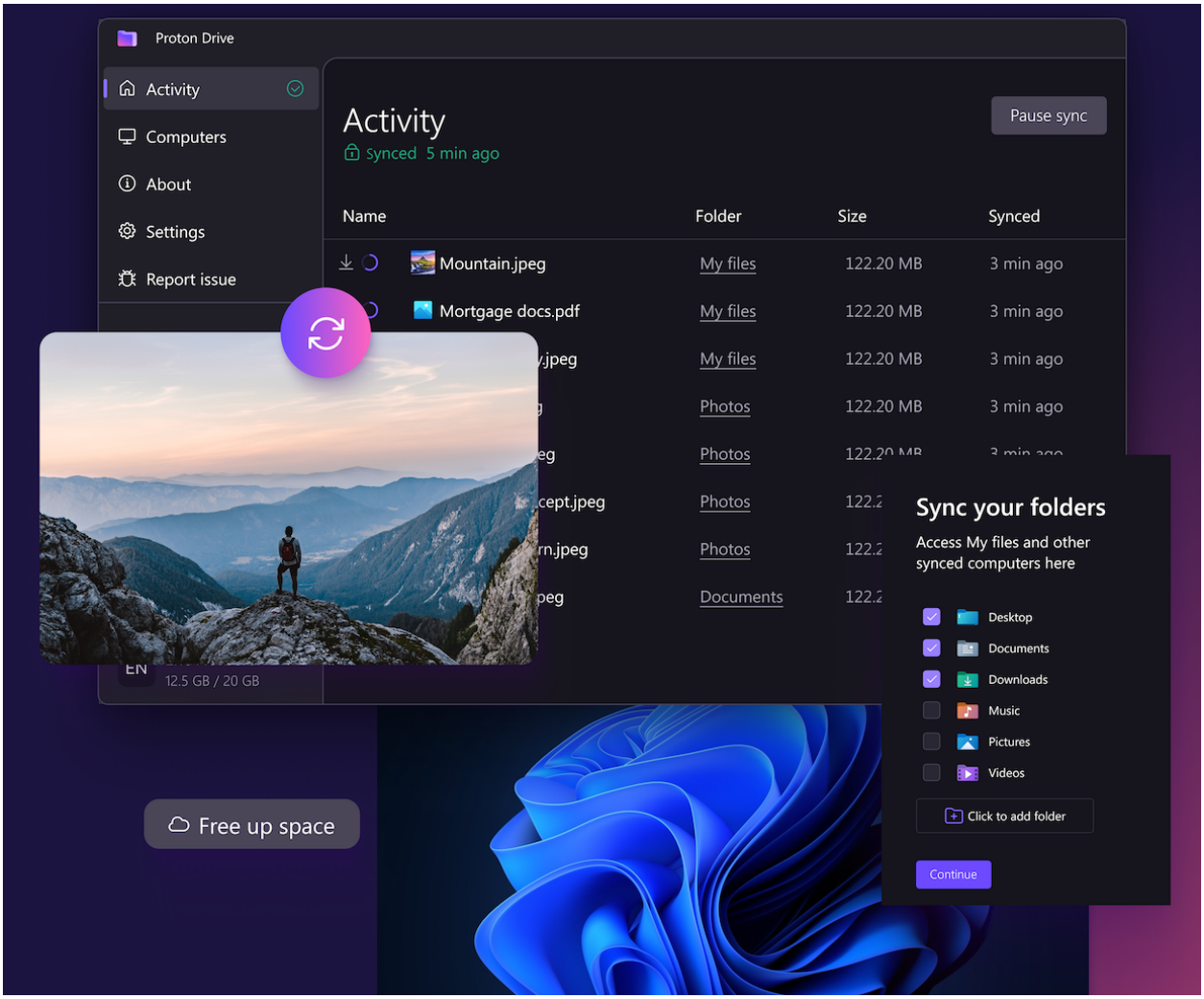

Рисунок 1.5 - Зовнішній вигляд сервісу для обміну файлами ProtonDrive

ProtonDrive [6] відомий своїм підвищеним рівнем конфіденційності та захищеністю даних. Він надає більш високий рівень шифрування, забезпечуючи більш безпечний обмін файлами без ідентифікації учасників. Проте деякі обмеження за розміром файлів і зручністю використання можуть бути проблематичними для користувачів, які обмінюються великими обсягами даних або віддають перевагу більш зручним і широко використовуваним платформам.

Таким чином, хоча є рішення, надають певні рівні безпеки, вони можуть не бути ідеальним вибором для контексту безпеки або комфорту користувача через обмеження у використанні, а також потенційні вразливості в забезпеченні анонімності та конфіденційності.

1.3 Месенджери

Месенджери стали невід'ємною частиною нашої повсякденної комунікації, забезпечуючи швидке та зручне передавання повідомлень і файлів. У контексті обміну інформацією без ідентифікації учасників, деякі месенджери пропонують функціонал, який забезпечує підвищений рівень конфіденційності та безпеки. Однак, під час розгляду таких платформ важливо враховувати не тільки їхню захищеність, а й обмеження, які можуть впливати на комфорт використання. Давайте розглянемо кілька популярних месенджерів та особливості, які визначають їхній рівень безпеки та зручності для користувачів.

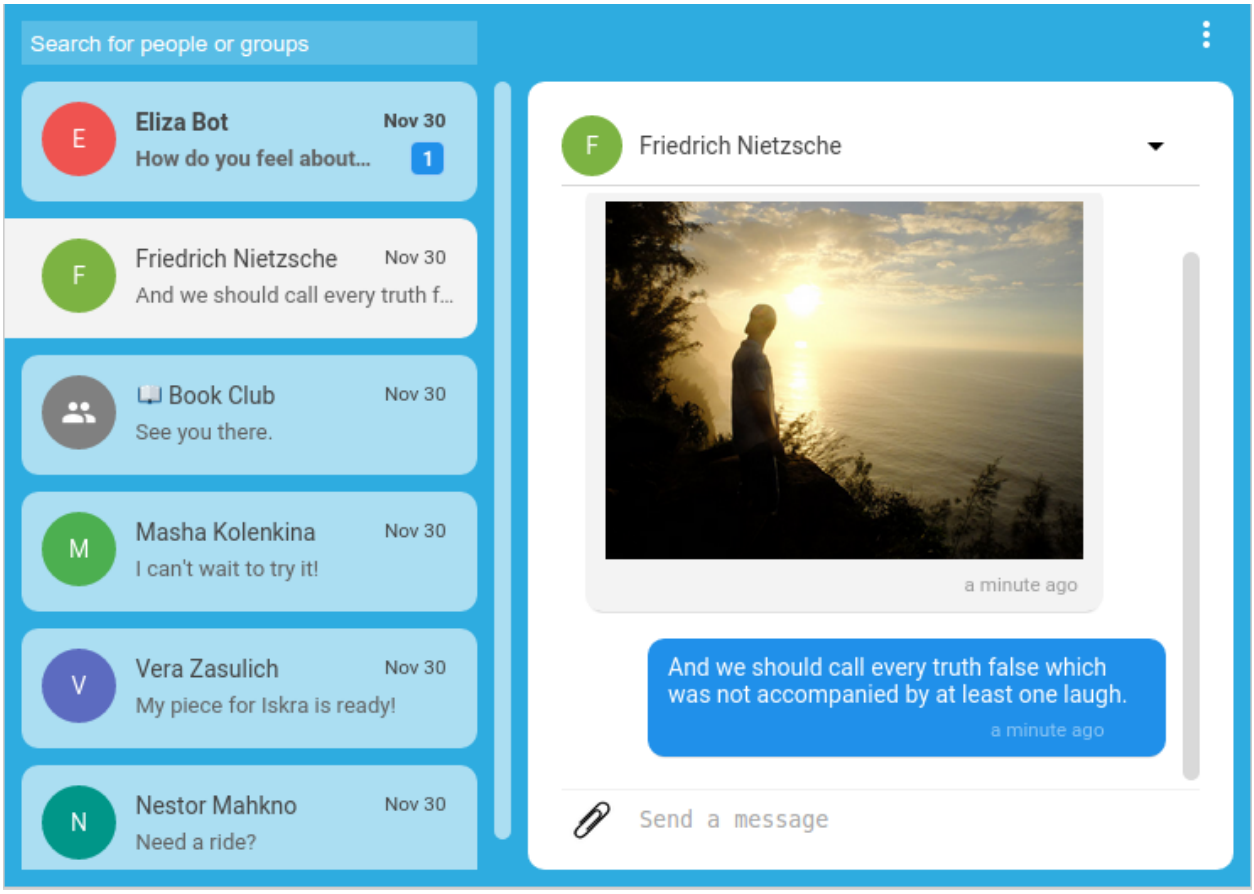

Рисунок 1.6 - Зовнішній вигляд месенджеру Signal

Signal [7] відомий своїм високим рівнем конфіденційності та безпеки. Він надає шифрування end-to-end для повідомлень і файлів, забезпечуючи високий рівень захисту даних. Однак, процедура реєстрації з використанням номера телефону може створювати незручності для користувачів, які бажають абсолютної анонімності.

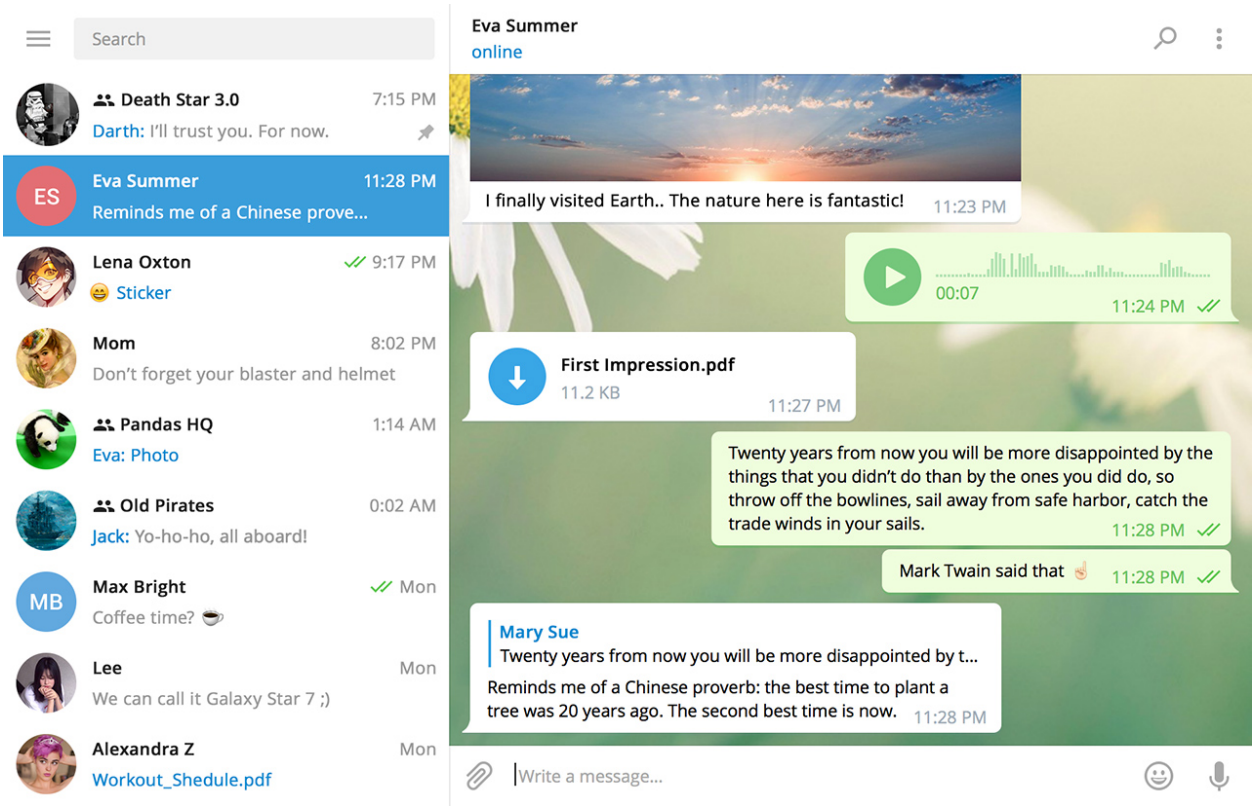

Рисунок 1.7 - Зовнішній вигляд месенджеру Telegram

Telegram [8] пропонує функцію "Секретних чатів" із шифруванням і автодеструкцією повідомлень, забезпечуючи підвищений рівень безпеки. Однак, стандартні чати Telegram не забезпечують такого ж рівня конфіденційності, що може бути проблемою для користувачів, які потребують підвищеного захисту.

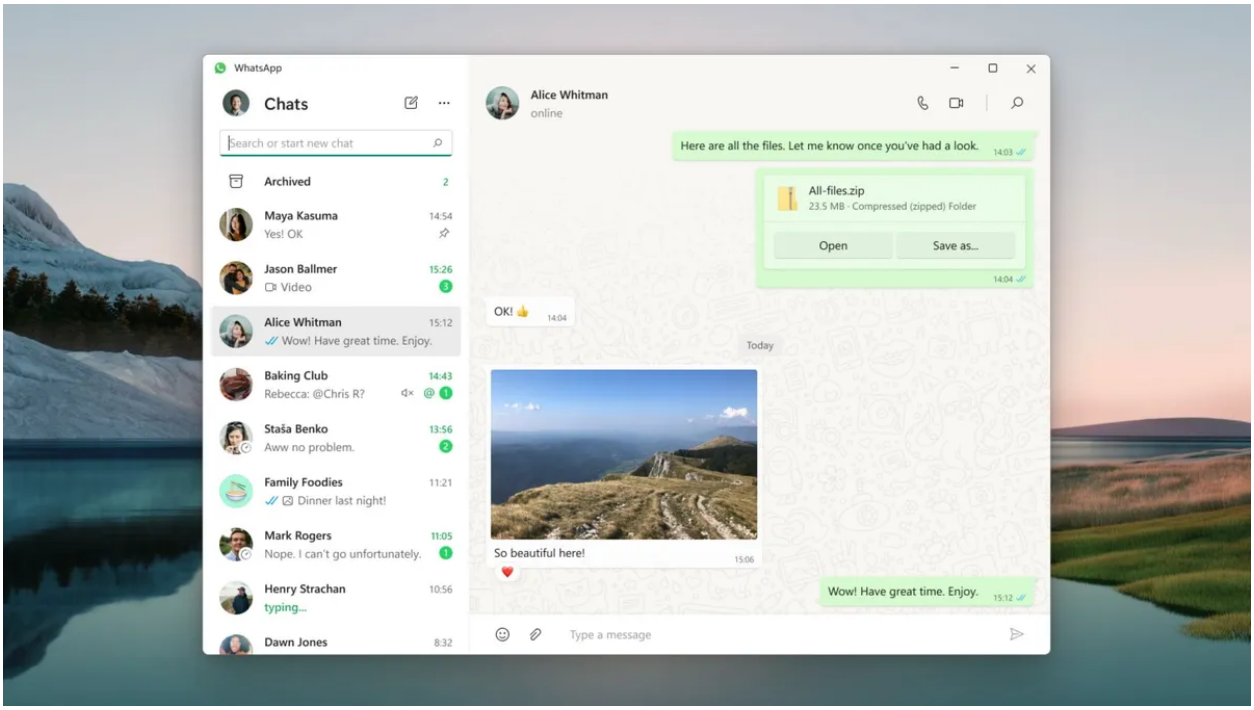

Рисунок 1.8 - Зовнішній вигляд месенджеру WhatsApp

WhatsApp [9] також забезпечує шифрування повідомлень, але вимагає номер телефону для реєстрації, що може створювати певні обмеження в анонімності користувачів. Крім того, він має обмеження за розміром переданих файлів, що може бути незручним для обміну великими обсягами даних.

Таким чином, кожен месенджер пропонує певні рівні безпеки та комфорту під час обміну повідомленнями та файлами без ідентифікації учасників. Однак, багато хто з них має певні обмеження, такі як вимога номера телефону для реєстрації або обмеження за розміром файлів, що може ускладнювати використання або створювати незручності для певних категорій користувачів.

1.4 P2P платформи

Платформи P2P (peer-to-peer) обміну файлами, як-от BitTorrent, LimeWire і eDonkey, тривалий час надавали можливість користувачам обмінюватися контентом безпосередньо, оминаючи централізовані сервери. Ці системи засновані на розподіленій мережі, де кожен користувач може одночасно виступати як клієнт і сервер, викачуючи і завантажуючи файли іншим учасникам.

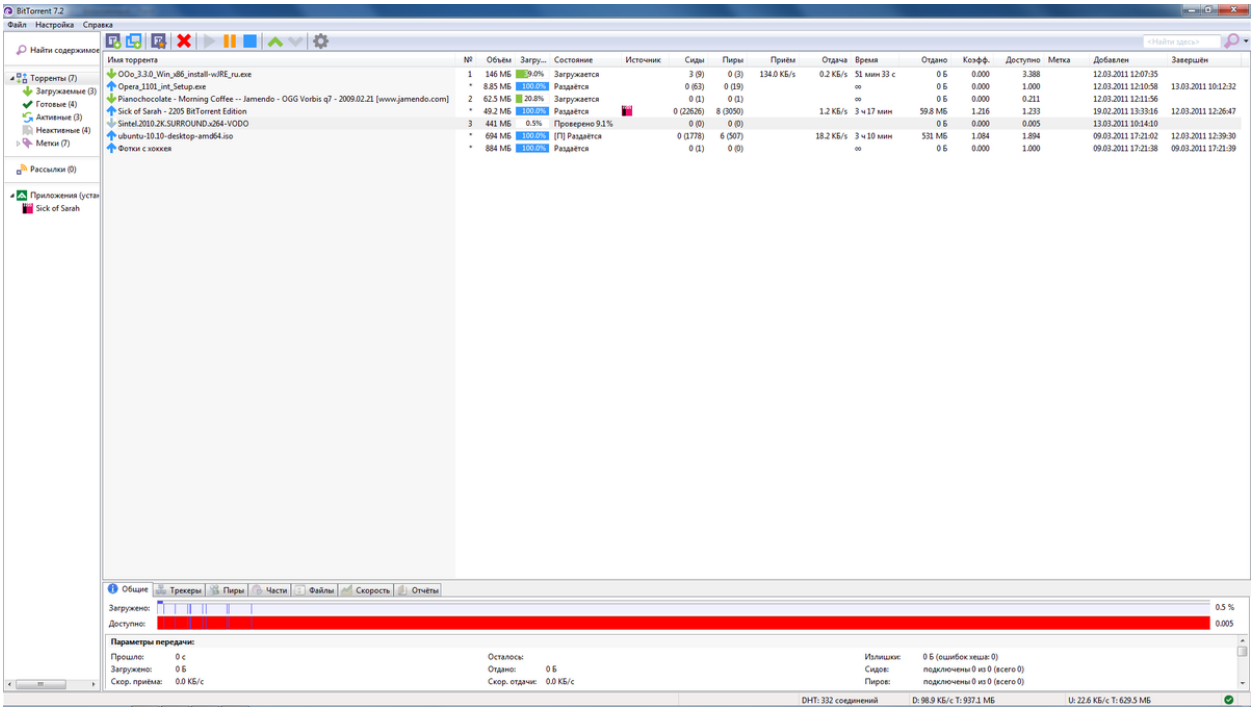

Рисунок 1.9 - Зовнішній вигляд P2P платформи BitTorrent

BitTorrent [10] дає змогу обмінюватися файлами, розбиваючи їх на невеликі частини, що підвищує швидкість завантаження за рахунок використання ресурсів усіх учасників мережі. Такі системи спочатку створювалися для ефективного обміну файлами, але у них є серйозні проблеми з точки зору безпеки та приватності. BitTorrent, хоча й дає змогу обмінюватися файлами ефективно, не надає гарантій конфіденційності даних. Його політика конфіденційності дозволяє збирати інформацію про

користувачів та їхню активність, що створює ризик витоку особистої інформації.

Крім того, багато платформ P2P мають обмежений рівень шифрування або його відсутність [11], що робить передачу даних вразливою до прослуховування і витоків. Деякі з них також можуть мати недостатні політики приватності, дозволяючи збір даних про користувачів та їхню активність без належного захисту.

Таким чином, незважаючи на їхню ефективність в обміні файлами, системи P2P стикаються з серйозними проблемами безпеки, несанкціонованим доступом до вмісту та вразливостями приватності, що робить їх невідповідними для сценаріїв, які потребують високого рівня конфіденційності та безпеки даних.

1.5 Висновок

Реалізація інформаційної технології, що забезпечує обмін файлами без ідентифікації учасників, видається важливою для забезпечення приватності та безпеки користувачів.

Вона відповідає основним принципам, які я вважаю необхідними у сфері обміну інформацією - це анонімність і захист особистих даних. Використання такої технології відповідає моїй філософії забезпечення конфіденційності інформації та поваги до особистого життя користувачів. Це важливий крок у напрямку забезпечення приватності та підтримки довіри користувачів до сервісу обміну файлами, що для мене є пріоритетом.

1.6 Постановка задачі

Для досягнення поставленої мети необхідно вирішити наступні завдання:

- Аналіз наявних бібліотек та технологій для шифрування та виявлення їхніх недоліків.
- Розробка концепції та архітектури сервісу.
- Програмна реалізація додатку з використанням сучасних обраних технологій розробки сервісів для обміну файлами.
- Тестування розробленого сервісу.

Результатом роботи має стати система обміну файлами без ідентифікації користувачів.

2 ПІДГОТОВКА ДО РОЗРОБКИ ВЛАСНОГО СЕРВІСУ

Розглянувши аналоги, потрібно підготуватися до розробки сервісу. Перш за все, нам потрібна технологія симетричного шифрування. Симетричне шифрування, одне з фундаментальних понять криптографії, відіграє важливу роль у забезпеченні безпеки передачі інформації.

В роботі будуть розглянуті принципи та застосування симетричного шифрування в контексті обміну файлами без ідентифікації учасників процесу. Цей метод шифрування ґрунтується на використанні одного й того самого ключа для шифрування та дешифрування даних, що забезпечує їхню конфіденційність і захищеність.

Буде здійснено детальний огляд різних методів симетричного шифрування, таких як AES256 GCM (Advanced Encryption Standard з довжиною ключа 256 біт у режимі галочкового коду аутентифікації), а також порівняння типів шифрування, таких як ECB, CBC, CFB, OFB та інших. Буде проаналізована їх ефективність, стійкість до атак, а також застосовність у контексті створення безпечного середовища для обміну файлами без ідентифікації учасників процесу.

Перший режим шифрування це ECB (Electronic Codebook) - є одним із базових методів симетричного шифрування, який може бути застосований у різних сценаріях обміну файлами [12]. У межах моєї роботи цей режим шифрування використовується для захисту даних під час передачі файлів між учасниками, забезпечуючи конфіденційність інформації.

Основний принцип роботи ECB полягає в розбитті відкритого тексту на блоки фіксованого розміру і шифруванні кожного блоку незалежно один від

одного з використанням обраного алгоритму і ключа. Такий підхід забезпечує простоту і швидкість шифрування, що важливо в контексті обробки великих обсягів даних, коли швидкість роботи алгоритму відіграє значну роль.

Однак, важливо враховувати, що використання ECB може мати свої недоліки. Одним із головних є можливість ідентифікації шаблонів у зашифрованих даних, особливо коли однакові блоки інформації шифруються в однакові шифртексти. Це може стати джерелом уразливості, особливо під час роботи з даними, що містять повторювані або однотипні блоки інформації.

У роботі, застосування ECB є усвідомленим і ґрунтується на специфіці даних, що передаються, з огляду на можливі вразливості та вимагаючи додаткових заходів для забезпечення безпеки обміну файлами.

Наступний режим це CBC (Cipher Block Chaining) [13]. Він є важливою складовою безпечного обміну файлами без ідентифікації учасників. Цей метод шифрування застосовується для забезпечення конфіденційності переданої інформації шляхом комбінування відкритих даних із попередніми зашифрованими блоками.

Використовуючи режим CBC, кожен блок відкритого тексту комбінується з попереднім зашифрованим блоком за допомогою операції XOR перед застосуванням алгоритму шифрування з ключем. Цей метод надає додатковий рівень безпеки порівняно з ECB, оскільки він забезпечує дифузію змін у відкритому тексті та запобігає простій ідентифікації шаблонів у зашифрованих даних.

Однак, важливо зазначити, що для коректної роботи режиму CBC необхідно правильно керувати ініціалізаційним вектором (IV) - випадковим

початковим значенням. Повторне використання одного і того ж IV може призвести до вразливостей у безпеці. Також, режим CBC має особливість, що вимагає дешифрування кожного блоку зашифрованого тексту перед дешифруванням наступного, що може обмежити паралельне опрацювання даних і впливати на продуктивність.

У рамках роботи, застосування режиму CBC здійснюється з усвідомленням його переваг і вразливостей, а також із вжиттям відповідних заходів для забезпечення безпеки обміну файлами без ідентифікації учасників.

Наступний режим шифрування, який буде використаний це CFB (Cipher Fiiedback) - являє собою метод криптографії, який може використовуватися для забезпечення безпеки обміну файлами без прив'язки до конкретних учасників процесу [14]. Цей режим шифрування належить до групи режимів роботи блочного шифру, де кожен блок шифрується з використанням попереднього зашифрованого блоку.

Процес шифрування в режимі CFB починається з передачі блоку даних через функцію зворотного зв'язку шифру, результат якої комбінується з відкритим текстом за допомогою операції XOR, створюючи шифртекст. Цей отриманий шифртекст стає входом для наступної операції шифрування, забезпечуючи подальший ланцюжок зашифрованих блоків даних.

Однією із значних переваг режиму CFB є його здатність працювати з різними розмірами блоків даних, що робить його гнучким і застосовним для шифрування файлів різного розміру і структури. Крім того, цей режим дає змогу уникнути прямої відповідності між блоками відкритого тексту і шифртексту, підвищуючи рівень безпеки переданих даних.

Однак, важливо враховувати, що режим CFB вимагає точної синхронізації між відправником і одержувачем, оскільки будь-яка зміна в блоці відкритого тексту позначатиметься на наступних зашифрованих блоках. Це може зробити цей режим менш стійким до помилок передачі даних. У роботі використання режиму CFB здійснюється з урахуванням його переваг і особливостей з метою забезпечення безпечного обміну файлами без прив'язки до конкретних учасників.

Ще одним режимом шифрування є OFB (Output Feedback) – що є методом шифрування блочного типу, використовується для забезпечення конфіденційності під час передачі даних, не вимагаючи прив'язки до певних учасників у процесі обміну файлами [15].

Цей метод починає процес шифрування зі створення потоку псевдовипадкових бітів, використовуваного для перетворення відкритого тексту за допомогою операції XOR. Початковим значенням для формування цього потоку слугує початковий вектор (IV), який піддається шифруванню, і результат перетворюється на перший блок для подальшого використання в цьому ж ланцюжку шифрування.

Однією з ключових переваг OFB є його здатність створювати псевдовипадкові біти, які не залежать від самого відкритого тексту. Це дає змогу забезпечити безпеку даних без створення прямого зв'язку між блоками вихідної інформації та шифрованим текстом, що збільшує рівень захисту під час передавання файлів.

Однак, важливо зазначити, що помилка в передачі даних в одному блоці відкритого тексту може призвести тільки до помилки у відповідному блоці шифрованого тексту, не зачіпаючи наступні дані. У своїй роботі я враховую ці

особливості OFB і використовую цей режим з розумом для забезпечення безпечного обміну файлами без необхідності ідентифікації учасників.

Останнім режимом шифрування який я буду використовувати це AES256 GCM (Advanced Encryption Standard 256 - Galois/Counter Mode) – який є одним із просунутих методів шифрування, який використовується для забезпечення безпеки під час обміну файлами без ідентифікації учасників [16].

У роботі застосування AES256 GCM є важливою складовою, оскільки цей метод шифрування має високий ступінь захисту та ефективності. Він комбінує алгоритм AES256 для шифрування даних та автентифікацію за алгоритмом GCM, що забезпечує конфіденційність, цілісність та автентичність переданих повідомлень.

Одна з головних переваг AES256 GCM полягає в його здатності забезпечувати автентифікацію даних, що дає змогу перевірити цілісність повідомлень і виявити будь-які спроби зміни інформації. Крім того, цей режим шифрування забезпечує високу швидкість обробки даних, що важливо під час передавання файлів.

Розглянемо основні складові роботи. Технології вебу були обрані задля реалізації інтерфейсу. Основним компонентом є Javascript. Використання JavaScript у роботі, дозволило скористатися широким вибором бібліотек, що забезпечує гнучкість і розширює можливості даного проекту. Це дало змогу розробити універсальне рішення, яке може легко інтегруватися і працювати на різних платформах і пристроях.

Однією з головних особливостей JavaScript, яку активно використано в проєкті, є асинхронні функції [17]. Цей аспект мови програмування дає змогу

ефективно керувати виконанням операцій, особливо під час роботи з обробкою файлів і мережевими запитами. Асинхронні функції забезпечують високу чуйність програми, що вкрай важливо для процесу обміну файлами в режимі реального часу.

Застосування JavaScript і його можливостей асинхронного програмування не тільки покращує продуктивність, а й забезпечує безпеку та ефективність обміну файлами в рамках даного проєкту. Це ключовий фактор, який визначає успіх розробленої універсальної бібліотеки для обміну файлами без ідентифікації учасників.

Окрім Javascript, також буде використовуватися Node.js [18] - середовище виконання JavaScript на сервері, яке відіграє важливу роль у роботі над створенням універсальної бібліотеки для обміну файлами без ідентифікації учасників. Використання Node.js забезпечує мені можливість створення масштабованих і ефективних серверних додатків на базі JavaScript.

За допомогою Node.js створено серверну частину бібліотеки, забезпечено обробку запитів, управління даними та безпеку інформації. Його асинхронна природа дозволяє мені ефективно обробляти безліч запитів без блокування основного потоку виконання.

Також будуть використані HTML та CSS для створення розмітки веб сторінки сервісу.

Для забезпечення та контролю цілісності файлів використано метод перевірки CRC32. Цей метод не є криптографічним підписом, але забезпечує перевірку цілісності даних шляхом обчислення контрольної суми файлу.

CRC32 [19] - це алгоритм циклічного надлишкового коду, який обчислює контрольну суму для файлу. Це дає змогу швидко визначити, чи змінився

файл, ґрунтуючись на його контрольній сумі. Цей метод широко використовується для виявлення випадкових змін у файлах, таких як помилки під час передачі даних або пошкодження.

Використовуючи перевірку CRC32, можно швидко й ефективно перевірити цілісність файлів під час обміну ними без ідентифікації учасників. Це дає змогу виявляти випадкові зміни у файлах, забезпечуючи додатковий рівень безпеки та цілісності в моїй бібліотеці обміну файлами.

Також у роботі буде використано низку бібліотек. Зокрема, Socket.IO, Express та WebCryptoAPI.

Socket.IO [20] використовується для забезпечення двосторонньої комунікації між клієнтом і сервером. Socket.IO надає зручний інтерфейс для роботи з веб-сокетами і забезпечує механізми реального часу для обміну даними. Основна перевага Socket.IO полягає в його розширеній функціональності та надійності порівняно зі звичайними веб-сокетами.

Express [21] - це веб-фреймворк для Node.js, який буде інтегровано у даний проєкт для забезпечення створення серверних додатків і API. Він надає потужні інструменти для спрощення розробки серверної частини додатка, забезпечуючи маршрутизацію, управління запитами та відповідями, а також підтримку шаблонів для створення веб-сторінок. Однією з основних причин використання Express є його легкість і гнучкість. Фреймворк дає змогу створювати масштабовані та продуктивні веб-додатки, забезпечуючи гнучкі налаштування маршрутів і обробників запитів.

WebCryptoAPI [22] в JavaScript являє собою інтегрований інтерфейс для криптографічних операцій у браузері. Незважаючи на свою корисність, він має аспекти, на які варто звернути увагу. Асинхронна природа певних

операцій іноді може сповільнити виконання складних завдань, впливаючи на продуктивність. Але API забезпечує високий рівень безпеки та надійності, завдяки використанню вбудованих криптографічних реалізацій браузера, захищаючи дані від атак на хеш-функції або шифри. Працює він у безпечному середовищі, доступ до якого обмежений через HTTPS-протокол або локальне середовище, що забезпечує захист від перехоплення або зміни даних.

Також вважаю за необхідне розповісти про інші наявні технологічні рішення для розробки, однак відхилині через низку причин.

По-перше, це Bcrypt [23] - адаптивна криптографічна хеш-функція формування ключа, що використовується для захищеного зберігання паролів. Незважаючи на широке поширення і застосування Bcrypt у контексті захисту паролів, рішення про його виключення зумовлене необхідністю забезпечення максимального рівня довіри і перевіреності у виборі криптографічних методів.

Виключивши Bcrypt з розгляду, я використовую інші альтернативи, що пропонують високий рівень перевіреності та безпеки в контексті хешування паролів. Це дає змогу забезпечити надійний захист конфіденційних даних користувачів і знизити потенційні ризики, пов'язані із застосуванням неперевірених криптографічних методів.

По-друге, це Асиметричне шифрування, включно з алгоритмом Шора та RSA.

Алгоритм Шора [24], хоч і становить теоретичну загрозу для методів, заснованих на складності факторизації, на практиці потребує квантових комп'ютерів для ефективної роботи. Наразі такі комп'ютери перебувають у

стадії досліджень і розробок, тому реальне застосування алгоритму Шора поки що залишається теоретичним.

RSA [25], широко використовуваний асиметричний алгоритм, заснований на складності факторизації великих чисел. Однак розвиток квантових комп'ютерів створює потенційну загрозу для RSA і подібних методів шифрування, оскільки квантові обчислення можуть легко зламувати алгоритми, що базуються на факторизації.

У контексті файлообмінника, асиметричне шифрування для обміну ключами є зайвим. Оскільки сам процес обміну файлами вже передбачає використання симетричних ключів для шифрування та дешифрування, додаткова реалізація асиметричного шифрування для обміну ключами є недоцільною.

По-третє, це IPSEC [26] - набір протоколів і методів, призначених для забезпечення безпеки та захисту даних у мережах IP. Його зазвичай застосовують для створення віртуальних приватних мереж (VPN), забезпечуючи шифрування та автентифікацію трафіку між пристроями.

У моєму веб додатку буде активно використовуватися HTTPS для забезпечення безпеки в інтернеті. HTTPS (HTTP Secure) - це комбінація протоколу HTTP і шифрування SSL/TLS, що застосовується для захисту комунікації між клієнтами та серверами в мережі. Вона забезпечує захищене передавання даних, запобігає перехопленню і забезпечує цілісність інформації.

Я буду інтегрувати SSL/TLS сертифікати для шифрування даних, забезпечуючи безпечне передавання чутливої інформації, такої як особисті дані та фінансова інформація.

IPsec, хоча й ефективний для забезпечення безпеки на рівні мережі, вимагає додаткової конфігурації та інтеграції на рівні операційної системи або мережевого обладнання. Оскільки HTTPS вже широко застосовується і забезпечує високий рівень захисту, впровадження IPsec є надлишковим і складним для реалізації, особливо на рівні веб-додатку.

Наостанок, повідомлю про невикористання P2P. Відмова від використання P2P у моєму проєкті зумовлена кількома факторами. Насамперед, у процесі дослідження було виявлено, що управління з'єднаннями безпосередньо між пристроями, характерне для P2P, може становити значні труднощі для користувачів, особливо для тих, хто не володіє високим рівнем технічної підготовки. Це може створювати проблеми і внести складнощі в користувацький досвід.

Крім того, аспект безпеки також був важливим у прийнятті рішення. P2P може становити вразливість для атак типу "Man-in-the-Middle" [27], де зловмисники можуть перехоплювати або змінювати передані дані між пристроями. З огляду на важливість безпеки для проєкту, рішення відмовитися від P2P на користь більш надійних і безпечних альтернатив було виправданим.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Розробка веб-застосунку

Отже розробку почнемо з написання клієнтської частини веб додатку. Як уже зазначалося, вона буде написана на JavaScript.

Спершу створено центральний клас для зберігання вхідних даних, інформації про файли а також змінні для виводу результатів. Задля зберігання вхідних даних (файли, пароль, ідентифікатор файлу, CRC32 таблиці) зазначимо змінну inputs. Змінну яка буде зберігати оброблені дані, включно з метаданими файлу, індексом чанків, самими чанками та публічним індексом назвемо chanked. Об'єкт, що містить інформацію про виведення процесу, як-от ім'я файлу, метадані файлу, ключ і серверний валідатор називатиметься output.

У центральний клас також необхідно додати різноманітні методи для програмної обробки. Використано наступні: reconscrtuct() - метод для переініціалізації змінних inputs, chanked і output, що обнуляє їхні значення, start() - метод для запуску обробки файлу, що визначає, чи є він відправником або одержувачем, і, залежно від цього, запускає процеси, _sendStart() - Основний метод для обробки файлу, що надсилається. Тут відбувається поділ файлу на чанки, їх шифрування, обчислення CRC32 і створення індексів. Також _sendStartRunner() - метод, що запускає процес надсилання чанків файлу, _NextToSend() - метод, що відповідає за надсилання наступного чанка файлу, _GetLink() - метод для отримання інформації про файл після його завантаження або обробки, FirstReq() - метод, що відправляє перший запит на сервер для отримання файлу за ідентифікатором, _GetReqDown() - метод, що обробляє запит на скачування файлу, Downloadreq() - метод, що запитує чанк

файлу для скачування,_Downloadhandler() - метод, що обробляє отримані чанки та їхнє дешифрування, DecryptONDown() - метод для дешифрування отриманих даних файлу після їх скачування, _processAndDownloadFile() - метод для обробки і скачування зібраного файлу.

Також були використані методи для шифрування та дешифрування файлів, а саме _generateEncryptionKey() - генерація ключа для шифрування, _encryptData(data, key) - шифрування даних за допомогою ключа,_decryptData(encryptedBuffer, key) - дешифрування даних за допомогою ключа, _decryptIndex(combined, hexKey) - дешифрування індексу файлу на основі отриманого ключа.

Були застосовані другорядні методи для створення та обчислення CRC32 таблиці _makeCRCTable(): та _crc32(data).

Використані методі для відправки та обробки повідомлень через сокети з використанням socket.emit() і обробників подій для різних типів повідомлень, а саме SendFileReq для ініціювання передачі файлу на сервер, ChankRunner для надсилання інформації про чанк файлу на сервер, PrimaryRunner що надсилає оброблений чанк файлу на сервер, FirstReq - використовується для запиту файлу із зазначеним ідентифікатором на сервері, Downloadreq який надсилає запит на сервер для отримання певного чанка файлу.

У межах класу було реалізовано кілька допоміжних функцій, що забезпечують різні аспекти роботи з даними. Серед них функції для перетворення даних між форматами, як-от _arrayBufferToHexString і _hexStringToArrayBuffer, а також методи для імпорту та використання ключів шифрування даних, на кшталт _importKey, _encryptData, _decryptData і _generateEncryptionKey. Також присутні функції для обчислення контрольної

суми CRC32 (_crc32), перевірки типу об'єкта (isPlainObject) і конкатенації буферів даних (concatenateBuffers). Ці функції забезпечують важливі кроки по роботі з даними, шифруванню та забезпеченню цілісності інформації всередині класу.

Усередині класу передбачено функції для роботи з DOM. Наприклад, функція _sendStart, яка оновлює інтерфейс користувача, відображаючи інформацію про обраний файл: змінює вміст текстових елементів (наприклад, за допомогою document.getElementById) для відображення імені файлу, його розміру та поточного стану обробки. Аналогічно, функція _GetLink встановлює значення елементів DOM для відображення інформації про згенеровані дані файлу, такі як ID файлу та ключ. Ці функції забезпечують динамічне оновлення інтерфейсу користувача залежно від стану виконання операцій із файлом.

Враховуючи вищезазначене, серверна реалізація та клас виглядатиме наступним чином:

```
class Runner {
    constructor() {
        this.inputs = {
            isSender: false,
            selectedFile: null,
            password: "",
            fileId: "",
            crc32Table: [],
```

```
    };
    this.chanked = {
        meta: "",
        index: {},
        chanked: {},
        indexpublic: {}
    };
    this.output = {
        File:"",
        FileMeta:"",//finishLater
        Key:"",
        ServerValidator:""
    };
    this._makeCRCTable();
}
reconscrtuct(){
    this.inputs = {
        isSender: false,
        selectedFile: null,
        password: "",
```

```
        fileId: "",
        crc32Table: [],
    };
    this.chanked = {
        meta: "",
        index: {},
        chanked: {},
        indexpublic: {}
    };
    this.output = {
        File:"",
        FileMeta:"",//finishLater
        Key:"",
        ServerValidator:""
    };

    this._makeCRCTable();
}
start(isSender, file = null, password = "", fileId = "123") {
    this.inputs.isSender = isSender;
```

```
    this.inputs.selectedFile = file;
    this.inputs.password = password;
    this.inputs.fileId = fileId;
    if (this.inputs.isSender) {
        console.log("Selected file:", this.inputs.selectedFile.name);
        socket.emit("SendFileReq", true);
    } else if (this.inputs.password && this.inputs.fileId) {
        this.FirstReq();
        console.log("Requesting file with ID:", this.inputs.fileId);
    } else {
        throw new Error('ErrCode: 001, Not all data was given');
    }
}
async _sendStart(FileID) {
    // (...)
}
_sendStartRunner(test){
    // (...)
}
_NextToSend(message){
```

```
    // (...)
}
async sendChanckRunner(id){
    // (...)
}
_GetLink(msg){
    // (...)
}
async _encryptIndex(key) {
    // (...)
}
FirstReq(){
    // (...)
}
async _GetReqDown(msg) {
    // (...)
}
Downloadreq(chank){
    // (...)
}
```

```
_Downloadhandler(msg){
    // (...)
}
async DecryptONDown(){
    // (...)
}
async _processAndDownloadFile(assembledFile) {
    // (...)
}
_makeCRCTable() {
    // (...)
}
_crc32(data) {
    // (...)
}
async _generateEncryptionKey() {
    // (...)
}
_arrayBufferToHexString(buffer) {
    // (...)
```

```
}
async _decryptIndex(combined, hexKey) {
    // (...)
}
_hexStringToArrayBuffer(hexString) {
    // (...)
}
isPlainObject(obj) {
    // (...)
}
concatenateBuffers(...buffers) {
    // (...)
}
async _importKey(rawKey) {
    // (...)
}
async _encryptData(data, key) {
    // (...)
}
async _decryptData(encryptedBuffer, key) {
```

```
        // (...)
    }
    async testEncryptionDecryption() {
        // (...)
    }
}
```

Повний код серверної частини можна подивитися у додатку.

Тепер перейдемо до клієнтської частини веб-додатку. У цій частині я також створюю клас Runner, проте з відмінностями від серверної частини.

Цей клас відіграє роль керуючого елемента для обробки файлів. Він містить властивості inputs (зберігання інформації про файл і передачу), chanked (для поділу файлу на чанки та управління ними) і output (для збереження даних перед надсиланням). Метод start у класі Runner призначений для початку обробки файлу залежно від режиму роботи (isSender) і наявності необхідних даних. Якщо екземпляр класу є відправником (isSender === true), обраний файл, його пароль та ідентифікатор файлу встановлюються у властивості класу, після чого ініціюється передача файлу на сервер. В іншому випадку, якщо встановлено пароль та ідентифікатор файлу, викликається метод FirstReq, що відправляє запит на сервер для отримання файлу із зазначеним ідентифікатором. Якщо не всі необхідні дані задано, викидається помилка з кодом 001.

Метод _sendStart у класі Runner відповідає за обробку файлу, що надсилається. Він розбиває обраний файл на чанки певного розміру, шифрує кожен чанк, зберігає інформацію про чанки (оригінальний CRC32, ключ,

зашифрований CRC32) і відправляє їх на сервер. Після обробки всіх чанків формується зашифрований індекс і надсилається на сервер. Додатково генерується ключ шифрування, який також відправляється на сервер. Нарешті, метод готує дані для надсилання, встановлює стан файлу та ініціює надсилання індексів на сервер.

Методи _sendStartRunner, _NextToSend і sendChanckRunner у класі Runner послідовно обробляють передавання чанків файлу на сервер. _sendStartRunner перевіряє коректність конфігурації файлу і починає надсилання чанків, починаючи з індексу 0. _NextToSend обробляє повідомлення від сервера, продовжуючи передачу в разі можливої повторної спроби. sendChanckRunner відправляє вказаний чанк файлу на сервер. Ці методи забезпечують управління передачею даних, обробку результатів і послідовне надсилання чанків до завершення процесу.

Методи `_GetLink` і `_GetReqDown` у класі `Runner` відіграють ключову роль в обробці даних під час отримання посилання на файл і запиту на скачування із сервера. `_GetLink` обробляє повідомлення після передавання файлу, формує посилання для скачування на основі отриманих даних, оновлює інтерфейс і готує клас до нового процесу. А `_GetReqDown` реагує на запит із сервера на скачування файлу, обробляє отримані дані, розшифровує їх, оновлює інтерфейс і готує клас до скачування та подальших дій. Обидва методи надають ключову функціональність для завершення передачі файлу, забезпечуючи обробку даних і підготовку до завершальних етапів процесу.

Тепер зупинюся окремо на методах що забезпечують шифрування і дешифрування даних з використанням алгоритму AES-GCM (Advanced Encryption Standard - Galois/Counter Mode). Зокрема, це метод

_generateEncryptionKey що генерує новий ключ шифрування розміром 256 біт, який використовується для шифрування даних. Він створює ключ, експортує його та повертає для подальшого використання. Також, це методи які використовуються для шифрування та дешифрування даних: _encryptData та _decryptData. _encryptData приймає дані, ключ шифрування і використовує алгоритм AES-GCM для їхнього шифрування, а _decryptData використовує той самий алгоритм для дешифрування даних із використанням заданого ключа. Також створені методи для шифрування та дешифрування індексу файлової структури, це методи _encryptIndex та _decryptIndex, де . _encryptIndex приймає структуру даних, алгоритмічно їх шифрує, створюючи зашифрований індекс, який можна зберегти і передати безпечно, а _decryptIndex використовує ключ для дешифрування зашифрованого індексу і відновлює оригінальні дані.

Враховуючи вищезазначене, наведу короткий уривок з клієнтської частини з класом Runner:

```
const socket = io();

class Runner {

    constructor() {

        this.inputs = {

            isSender: false,

            selectedFile: null,

            password: "",

            fileId: "",
```

```
            crc32Table: [],
        };
        this.chanked = { meta: "", index: {}, chanked: {}, indexpublic: {} };
        this.output = { File: "", FileMeta: "", Key: "", ServerValidator: "" };
        this._makeCRCTable();
    }

    reconscrtuct(){
            this.inputs = { isSender: false, selectedFile: null, password: "", fileId: "",
crc32Table: [] };
        this.chanked = { meta: "", index: {}, chanked: {}, indexpublic: {} };
        this.output = { File: "", FileMeta: "", Key: "", ServerValidator: "" };
        this._makeCRCTable();
    }

    start(isSender, file = null, password = "", fileId = "123") {
        // Початок обробки файлу..
    }

    async _sendStart(FileID) {
```

```
        // Початок обробки та шифрування файлу..
    }

    // Other methods...

    _makeCRCTable() {
        // Створення CRC32 таблиці..
    }

    _crc32(data) {
        // Обчислення CRC32 даних...
    }

    // Інші методи для шифрування

    async testEncryptionDecryption() {
        // Тестування шифрування та дешифрування.
    }
}
```

Повний код клієнтської частини можна подивитися у додатку.

Тепер перейдемо до створення інтерфейсу веб додатку. Я створив просту веб-сторінку для надсилання та отримання файлів з використанням сокетів. Вона містить елементи керування, як-от кнопки для надсилання та отримання файлів, поля для введення пароля та ідентифікатора файлу, а також місця для відображення інформації про файл.

Функції handleAction(isSender) викликаються під час натискання на кнопки "Send File" і "Get File" для ініціювання надсилання або отримання файлів. Вони витягують інформацію про файл, пароль та ідентифікатор файлу з відповідних елементів керування і запускають метод start об'єкта runner (екземпляра класу Runner), передаючи необхідні дані для обробки.

Події сокетів, як-от 'StarterPackId', 'StartRunner', 'NextPlease', 'doneHere', 'GetReqDown' і 'Downloadhandler', викликають відповідні методи екземпляра runner для опрацювання відповідей на надсилання та отримання файлів. Наприклад, подія 'StarterPackId' ініціює метод _sendStart об'єкта runner, який починає процес надсилання файлу.

Сторінка використовує сокети для взаємодії з сервером і об'єкт runner, щоб ініціювати процеси надсилання та отримання файлів, а також для оновлення користувацького інтерфейсу відповідно до операцій, що відбуваються.

Код сторінки виглядає наступним чином:

```
<!DOCTYPE html>
<html lang="en" style="background-color: darkslategrey;">
<head>
    <meta charset="UTF-8">
```

```
                    <meta name="viewport" content="width=device-width,
initial-scale=1.0">
        <title>Send File</title>
    </head>
    <body>
        <h1>Send a File</h1>
        <input type="file" id="fileInput">

        <button onclick="handleAction(true)">Send File</button><br><br>

        <input type="password" id="passwordInput" placeholder="Password">
        <input type="text" id="fileIdInput" placeholder="File ID">
        <button onclick="handleAction(false)">Get File</button>

        <p id="FileName"></p>
        <p id="FileSize"></p>
        <p id="FileState"></p>
        <p id="FileID"></p>
        <p id="FileKEY"></p>
```

```
<script src="/socket.io/socket.io.js"></script>
<script src="main.js"></script>
<script>
    var runner = new Runner();  // Инициализация runner здесь

    function handleAction(isSender) {
        const fileInput = document.getElementById('fileInput');
        const passwordInput = document.getElementById('passwordInput');
        const fileIdInput = document.getElementById('fileIdInput');
        const selectedFile = isSender ? fileInput.files[0] : null;

                    runner.start(isSender,  selectedFile,  passwordInput.value,
fileIdInput.value);
        //runner.start(isSender, selectedFile, "", "");
    }

    //up list
    socket.on('StarterPackId', (File) => runner._sendStart(File));
    socket.on('StartRunner', (File) => runner._sendStartRunner(File));
    socket.on('NextPlease', (File) => runner._NextToSend(File));
```

```
            socket.on('doneHere', (File) => runner._GetLink(File));


            //down list
            socket.on('GetReqDown', (File) => runner._GetReqDown(File))
                                        socket.on('Downloadhandler', (File) =>
runner._Downloadhandler(File))
        </script>
        </body>
        </html>
```

3.2 Тестування проекту

Після розробки сервісу, вихідний код якого можна буде знайти у додатку, та його розгортання на сервері, ми можемо перейти до його тестування. Переходячи на сторінку, нас зустрічає форма для відправки та отримання файлів. Якщо жодного файлу для відправки не обрано, про це повідомляється.

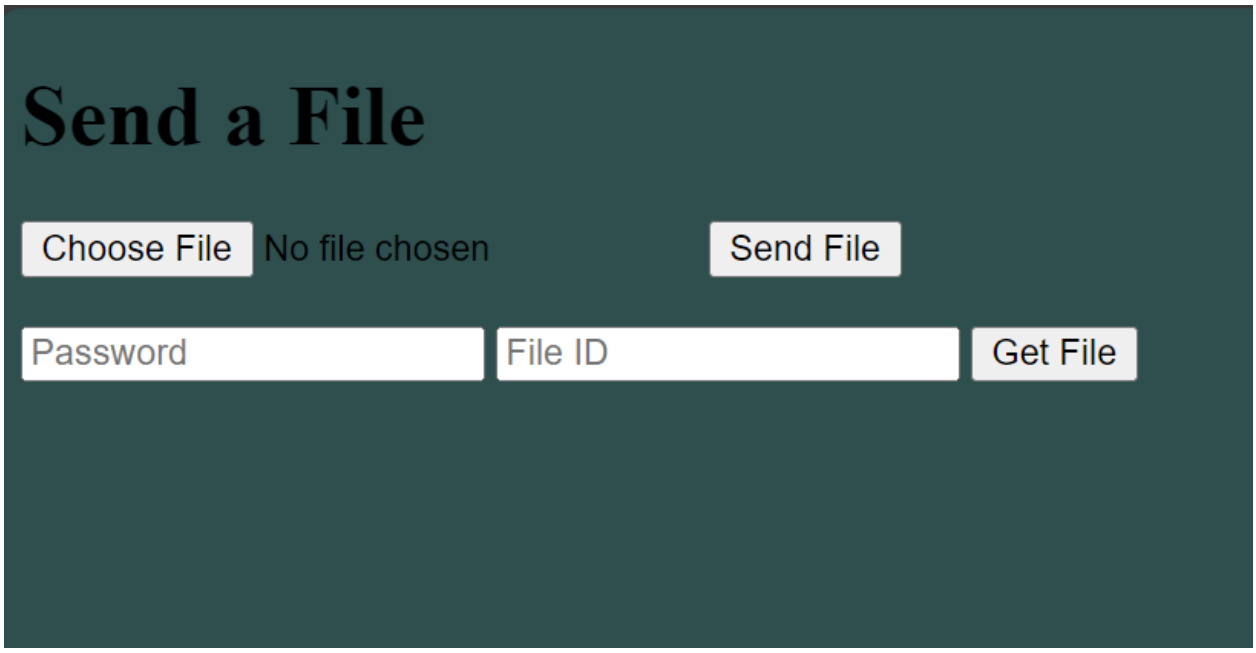

Рисунок 3.1 - Сторінка сервісу для відправки та отримання файлів

Для того щоб відправити файл, потрібно спочатку обрати файл натиснувши кнопку “Choose File” та вибрати файл з комп’ютера.

Рисунок 3.2 - Сторінка сервісу після обрання файлу

Сторінка покаже нам назву обраного файла. Для відправки натискаємо Send File.

Рисунок 3.3 - Сторінка сервісу після відправки файлу

Після успішної відправки файлу, сторінка покаже нам наступні дані: назву файлу, його розмір в байтах, текстовий статус - підтвердження віправки, ID файлу та ключ до файлу для отримання.

Для того щоб отримати відправлений файл, нам необхідно заповнити поля File ID, в нашому випадку це 309, та Password, значення якого вказано було у FileKEY:

Рисунок 3.4 - Сторінка сервісу після отримання файлу

Система вкаже назву файлу, розмір в бітах, та текстове повідомлення про статус запиту. Для наявності я також додав на скрін підтвердження того що загрузка відбулась.

Виходячи з тестування, робимо висновок що сервіс працює справно, без зайвих проблем і ми можемо визначити перспективи та плани його розвитку.

3.3 Визначення планів та перспектив на майбутнє

Отже, основним планом та перспективою на майбутнє для файлообміннику є інтеграція сервісу у повноцінне рішення для широкого обміну інформацією між користувачами, зокрема текстом, з підвищеною приватністю. Це покликано перетворитись на привабливу комерційну пропозицію з усіма шансами заняття місця на ринку та стати повноцінною альтернативою популярним месенджерам та файлообмінникам.

ВИСНОВКИ

В ході виконання роботи були вирішені наступні задачі:

- розглянуто та проаналізовано контекст приватності у різних плоскостях;
- розглянуті та проаналізовані аналоги сервісів для обміну файлами;
- проведено аналіз видів та методів шифрування, та допоміжних інструментів;
- розроблено сервіс для обміну файлами;
- протестовано сторонніми користувачами сервіс для обміну файлами.

Результатом виконання роботи є сервіс для обміну файлами без ідентифікації учасників процесу, що буде гарним помічником для тих, кому потрібен функціонал обміну файлами без додаткових ідентифікацій користувачів.

СПИСОК ЛІТЕРАТУРИ

1. The Problems of Internet Privacy and Big Tech Companies [Електронний ресурс] – Режим доступу: https://thesciencesurvey.com/
2. Scoping the Issue: Terrorism, Privacy, and Technology [Електронний ресурс] – Режим доступу: https://nap.nationalacademies.org
3. Threema [Електронний ресурс] – Режим доступу: https://threema.ch/en
4. WeTransfer [Електронний ресурс] – Режим доступу: https://wetransfer.com/
5. Dropbox [Електронний ресурс] – Режим доступу: https://www.dropbox.com/home
6. ProtonDrive [Електронний ресурс] – Режим доступу: https://proton.me/drive
7. Signal [Електронний ресурс] – Режим доступу: https://signal.org/
8. Telegram [Електронний ресурс] – Режим доступу: https://telegram.org/
9. WhatsApp [Електронний ресурс] – Режим доступу: https://www.whatsapp.com/
10. Bittorrent [Електронний ресурс] – Режим доступу: https://www.bittorrent.com/
11. Is P2P encryption safe [Електронний ресурс] – Режим доступу: https://www.caplinked.com/
12. Electronic Code Book (ECB) [Електронний ресурс] – Режим доступу: https://www.techtarget.com/
13. Block chaining of operations [Електронний ресурс] – Режим доступу: https://www.geeksforgeeks.org/

14. What is CFB? [Електронний ресурс] – Режим доступу: https://www.educative.io
15. What is OFB? [Електронний ресурс] – Режим доступу: https://www.educative.io
16. Advanced Encryption Standard with Galois Counter Mode using Field Programmable Gate Array [Електронний ресурс] – Режим доступу: https://iopscience.iop.org
17. Introducing asynchronous JavaScript [Електронний ресурс] – Режим доступу: https://developer.mozilla.org
18. About Node.js [Електронний ресурс] – Режим доступу: https://nodejs.org/en/about
19. CRC32 [Електронний ресурс] – Режим доступу: https://fuchsia.googlesource.com/
20. Socket.IO [Електронний ресурс] – Режим доступу: https://socket.io/
21. Express [Електронний ресурс] – Режим доступу: https://expressjs.com/
22. WebCryptoAPI [Електронний ресурс] – Режим доступу: https://nodejs.org/api/webcrypto.html#web-crypto-api
23. Bcrypt [Електронний ресурс] – Режим доступу: https://www.npmjs.com/package/bcrypt
24. Shor's Factorization Algorithm [Електронний ресурс] – Режим доступу: https://www.geeksforgeeks.org/
25. RSA algorithm [Електронний ресурс] – Режим доступу: https://www.simplilearn.com
26. What is IPSEC [Електронний ресурс] – Режим доступу: https://www.cloudflare.com/
27. MITM attack [Електронний ресурс] – Режим доступу: https://www.imperva.com/

ДОДАТОК

Вихідний код сервісу

server.js

```
const express = require('express');
const http = require('http');
const socketIo = require('socket.io');
const fs = require('fs');
const path = require('path');
const zlib = require('zlib');
const readline = require('readline');

const app = express();
const server = http.createServer(app);
//const io = socketIo(server);
const io = require('socket.io')(server, {
   maxHttpBufferSize: 1e7 // Пример: 10 МБ
});

const directoryPath = path.join(__dirname, '../bin');
app.use(express.static('public'));

//config
const listSize = 1000
```

```
// CRCCalculator Class
class CRCCalculator {
    constructor() {
        this.crc32Table = [];
        this._makeCRCTable();
    }

    _makeCRCTable() {
        let c;
        for (let n = 0; n < 256; n++) {
            c = n;
            for (let k = 0; k < 8; k++) {
                c = ((c & 1) ? (0xEDB88320 ^ (c >>> 1)) : (c >>> 1));
            }
            this.crc32Table[n] = c;
        }
    }

    _crc32(data) {
        let crc = 0 ^ (-1);
        for (let i = 0; i < data.length; i++) {
            crc = (crc >>> 8) ^ this.crc32Table[(crc ^ data[i]) & 0xFF];
        }
        return (crc ^ (-1)) >>> 0;
    }
}
```

```
function createLineReader(filePath) {
    const stream = fs.createReadStream(filePath, 'utf8');
    const reader = readline.createInterface({ input: stream });
    let lines = [];

    reader.on('line', (line) => {
        lines.push(line);
    });

    return {
        readLine: function (lineNumber, callback) {
            if (lineNumber <= lines.length) {
                callback(null, lines[lineNumber - 1]);
            } else {
                reader.once('line', () => {
                    callback(null, lines[lineNumber - 1]);
                });
            }
        },
        close: function () {
            reader.close();
        }
    };
}
```

```
const crcCalculator = new CRCCalculator();

io.on('connection', (socket) => {
   console.log('New client connected');

   const validator = {}

   socket.on('SendFileReq', () => {
      console.log('Client asked for connect');
      //check for existing file
      if (!fs.existsSync(directoryPath)) {
         fs.mkdirSync(directoryPath, { recursive: true });
      }
      const uniqueFileName = generateUniqueFileName(directoryPath);
      fs.writeFileSync(path.join(directoryPath, uniqueFileName), ''); // создает
пустой файл в директории ../bin
      console.log(`Created new file: ${path.join(directoryPath,
uniqueFileName)}`);
      let filename = parseInt(uniqueFileName.slice(0, -4))
      validator[filename] = Math.floor(Math.random() * listSize)
      let filerunner = {
         file:filename,
         v:validator[filename]
      }
      socket.emit('StarterPackId',filerunner)
   });
```

```
    let fileExpector = {};
    let LocalIndexHolder = {}

    socket.on('ChankRunner', (message) => {

console.log(isNaN(message.v),isNaN(message.file),!checkIndexPublic(message
))
        if(isNaN(message.v) || isNaN(message.file) ||
!checkIndexPublic(message)){
            socket.emit("StartRunner",false)
            return;
        }
        console.log(Object.keys(message.indexpublic).length)
        socket.emit("StartRunner",true)
        fileExpector[message.v] = new
Array(Object.keys(message.indexpublic).length);
        LocalIndexHolder[message.v] = message.indexpublic;
        return
    })

    socket.on('PrimaryRunner', (message) => {
        console.log(message)
        if(!fileExpector[message.v]){
            socket.emit('NextPlease', { success: false, message: "File error", again:
false });
            console.log("File fileExpector error")
            return
```

```
        }
        //Запустить тут проверку crc32
        const crc32Result = crcCalculator._crc32(Buffer.from(message.chanck));
        console.log(crc32Result + " = " +
LocalIndexHolder[message.v][message.id])

        if(crc32Result !== LocalIndexHolder[message.v][message.id]){
            socket.emit('NextPlease', { success: false, message: "Signature failed",
again: false, id:message.id });//again: true, id:message.id });
            return
        }

        let filePath = path.join(directoryPath, message.f + ".txt");
        appendChunkToFile(filePath, message.chanck.toString('base64'), (err)
=> {
//        appendCompressedChunkToFile(filePath, message.chanck, (err) => {
            if (err) {
                socket.emit('NextPlease', { success: false, message: err.message,
again: false });
            } else {
                socket.emit('NextPlease', { success: true, again: false, id: message.id
});
                if(message.id == (Object.keys(LocalIndexHolder[message.v]).length -
1)){
                    //console.log("Вщту")
                    //let link = "Wherever it gets: fileID: " + message.f + " and your
own pass";
```

```
                socket.emit('doneHere', {f:true,m:message.f});
                return //socket.disconnect();
            }
          }
        });

    })


    //downloader
    socket.on('FirstReq', (msg) => {
console.log(msg + ' YOU BUSTERD');
let pather = directoryPath + "\\" + msg + ".txt";

checkFileExists(pather)
  .then(exists => {
    if(exists){
        console.log("File Was read");
        readSpecificLineFromFile(pather, 1, (err, line) => {
          if (!err) {
            let lineReader = createLineReader(pather);

            //console.log(`Прочитана строка: ${line}`);
//            socket.emit('GetReqDown', "yea!")
            socket.emit('GetReqDown',Buffer.from(line,'base64'));
               //// NEXT CHANK TO REQ
               socket.on('Downloadreq', (msg) => {
```

```
                console.time("yourFunctionTimer");
                console.log(msg)
                let number = msg+1;
//                readSpecificLineFromFile(pather, number, (err, line) => {
                lineReader.readLine(number, (err, line) => {
                  ////////////////////////////////////////////////////////////////Если загрузка
завершена - то удалить подключение через close

                  console.timeEnd("yourFunctionTimer");
                  if(!err){
                    socket.emit('Downloadhandler',Buffer.from(line,'base64'))
                  }
                })
              });
        } else {
          console.error('Произошла ошибка:', err.message);
          socket.emit('GetReqDown', false)
        }
    });
  }else{
    socket.emit('GetReqDown', false)
  }})
  .catch(err => socket.emit('GetReqDown', false));
    });


    socket.on('disconnect', () => {
```

```
        console.log('Client disconnected');
        //socket.connect();
    });
});

const PORT = 3000;
server.listen(PORT, () => {
    console.log(`Server is running on http://localhost:${PORT}`);
});

function generateUniqueFileName(dirPath) {
    let fileName;
    do {
        fileName = `${Math.floor(Math.random() * listSize)}.txt`;
    } while (fs.existsSync(path.join(dirPath, fileName)));

    return fileName;
}

function checkIndexPublic(obj) {
    for (let key in obj.indexpublic) {
        // Проверка, что ключ - это строка, представляющая цифру
        if (isNaN(parseInt(key))) {
            return false;
        }
        // Проверка, что значение - это валидное CRC32
        if (isNaN(obj.indexpublic[key])) {
```

```
            return false;
        }
    }
    return true;
}

////Функция для записи чанка в файл
function appendChunkToFile(filePath, chunk, callback) {
    // Открытие файла для добавления данных
    fs.open(filePath, 'a', (err, fd) => {
        if (err) {
            callback(err);
            return;
        }

        // Запись чанка в конец файла
        fs.write(fd, chunk + '\n', (err) => {
            fs.close(fd, (errClose) => {
                if (err || errClose) {
                    callback(err || errClose);
                    return;
                }
                callback(null);
            });
        });
    });
}
```

```
function checkFileExists(filePath) {
   return new Promise((resolve, reject) => {
     fs.access(filePath, fs.constants.F_OK, (err) => {
       if (err) {
         console.error(`${filePath} does not exist`);
         return resolve(false);
       }
       console.log(`${filePath} exists`);
       resolve(true);
     });
   });
 }

  function readSpecificLineFromFile(filePath, lineNumber, callback) {
   let lineCount = 0;
   let specificLine = null;

   const stream = fs.createReadStream(filePath, 'utf8');
   const reader = readline.createInterface({ input: stream });

   reader.on('line', (line) => {
       lineCount++;
       if (lineCount === lineNumber) {
           specificLine = line;
           reader.close();
       }
```

```
    });

    reader.on('close', () => {
        if (!specificLine) {
            return callback(new Error('Invalid line number'));
        }
        callback(null, specificLine);
    });

    stream.on('error', (err) => {
        callback(err);
    });
}
```

main.js

```
const socket = io();

class Runner {
    constructor() {
        this.inputs = {
            isSender: false,
            selectedFile: null,
```

```
    password: "",
    fileId: "",
    crc32Table: [],
};

this.chanked = {
    meta: "",
    index: {},
    chanked: {},
    indexpublic: {}
};

this.output = {
    File:"",
    FileMeta:"",//finishLater
    Key:"",
    ServerValidator:""
}

// Инициализация CRC32 таблицы
```

```
    this._makeCRCTable();
}

reconscrtuct(){
    this.inputs = {
        isSender: false,
        selectedFile: null,
        password: "",
        fileId: "",
        crc32Table: [],
    };

    this.chanked = {
        meta: "",
        index: {},
        chanked: {},
        indexpublic: {}
    };

    this.output = {
```

```
            File:"",
            FileMeta:"",//finishLater
            Key:"",
            ServerValidator:""
        }

        this._makeCRCTable();
    }

    // Запуск обработки файла
    start(isSender, file = null, password = "", fileId = "123") {
        this.inputs.isSender = isSender;
        this.inputs.selectedFile = file;
        this.inputs.password = password;
        this.inputs.fileId = fileId;

//                  if (this.inputs.isSender && this.inputs.selectedFile &&
this.inputs.password && this.inputs.fileId) {
        if (this.inputs.isSender) {
            console.log("Selected file:", this.inputs.selectedFile.name);
```

```
            socket.emit("SendFileReq", true);
        } else if (this.inputs.password && this.inputs.fileId) {
            this.FirstReq();
            console.log("Requesting file with ID:", this.inputs.fileId);
        } else {
            throw new Error('ErrCode: 001, Not all data was given');
        }
    }

    // Основная функция для обработки отправляемого файла
    async _sendStart(FileID) {
        console.log(FileID)
        const meta = {
            name: this.inputs.selectedFile.name,
            size: this.inputs.selectedFile.size,
            type: this.inputs.selectedFile.type
        };
                                document.getElementById('FileName').textContent =
this.inputs.selectedFile.name;
                                 document.getElementById('FileSize').textContent =
this.inputs.selectedFile.size;
```

```
document.getElementById('FileState').textContent = 'Encrypting file';
this.chanked.meta = meta;

const CHUNK_SIZE = 128 * 1024; // Размер чанка в байтах (128KB)
let chunkIndex = 1; // Индекс чанка начинается с 1

for (let i = 0; i < this.inputs.selectedFile.size; i += CHUNK_SIZE) {
                const currentChunk = this.inputs.selectedFile.slice(i, i + CHUNK_SIZE);
    const chunkArrayBuffer = await currentChunk.arrayBuffer();

                                const originalCrc32 = this._crc32(new Uint8Array(chunkArrayBuffer));
            const randomKeyArray = window.crypto.getRandomValues(new Uint8Array(32)); // 256-битный ключ
            const randomKey = await this._importKey(randomKeyArray); // Преобразование в CryptoKey

        const encryptedChunk = await this._encryptData(chunkArrayBuffer, randomKey); // Шифрование чанка
                                const encryptedCrc32 = this._crc32(new Uint8Array(encryptedChunk));
```

```
        this.chanked.index[chunkIndex] = {
            O: originalCrc32,   // Original CRC32
            K: randomKeyArray,  // Случайный ключ (необходимо сохранять его в необработанном виде для дешифрования)
            E: encryptedCrc32   // Encrypted CRC32
        };

        this.chanked.indexpublic[chunkIndex] = encryptedCrc32; // Только зашифрованные CRC32
        this.chanked.chanked[chunkIndex] = encryptedChunk;

        chunkIndex++;
    }

    console.log("File processing complete", this.chanked);

    // Генерация ключа шифрования
    const encryptionKey = await this._generateEncryptionKey();

    // Шифрование индекса
```

```
                const encryptedIndexArrayBuffer = await this._encryptIndex(encryptionKey.key);
        this.chanked.chanked[0] = encryptedIndexArrayBuffer;  // Сохранение зашифрованного индекса как ArrayBuffer

    // Вычисление CRC32 зашифрованного индекса
                const encryptedIndexCrc32 = this._crc32(new Uint8Array(encryptedIndexArrayBuffer));
        this.chanked.indexpublic[0] = encryptedIndexCrc32; // Сохранение CRC32 зашифрованного индекса

    console.log("File processing complete", this.chanked);

    // Очистка выбранного файла из памяти
    this.inputs.selectedFile = null;

        let exportedKey = await window.crypto.subtle.exportKey("raw", encryptionKey.key);
    let keyString = this._arrayBufferToHexString(exportedKey);

    this.output.File = FileID.file
```

```
    this.output.ServerValidator = FileID.v
    this.output.Key = keyString
    this.output.FileMeta = this.chanked.meta

    //отправка индексов
    document.getElementById('FileState').textContent = 'Sending file';
    socket.emit("ChankRunner", {
      indexpublic:this.chanked.indexpublic,
      v:FileID.v,
      file:FileID.file
   })
 }

_sendStartRunner(test){
   if(!test){
      throw new Error('ErrCode: 002, File is missconfigured, try again!');
   }

   this.sendChanckRunner(0)
   return
```

```
    }


    _NextToSend(message){
        console.log(message)
        if(!message.success && !message.again){
            throw new Error('ErrCode: 003, File is missconfigured, try again!');
            return
        }
        let id = message.id + 1;
        if(message.again){
            id--
        }
        console.log(id)
        if(id == Object.keys(this.chanked.indexpublic).length){
            console.log("We are done!"); // delete file from memory here))
            //load constructer
                    document.getElementById('FileState').textContent = 'File was
uploaded))';
            return 1;
        }
```

```
    this.sendChanckRunner(id);
}

async sendChanckRunner(id){
    let obj = {
        id:id,
        v:this.output.ServerValidator,
        f:this.output.File,
        chanck:this.chanked.chanked[id]
    }
    socket.emit("PrimaryRunner",obj)
    return
}

_GetLink(msg){
    //console.log(msg)
    if(!msg.f){
        console.log("Shit happends")
    }
```

```
    let output = JSON.stringify(this.output)
    console.log(output);
            document.getElementById('FileID').textContent = 'FileID: '+ this.output["File"];
          document.getElementById('FileKEY').textContent = 'FileKEY: '+ this.output["Key"];
        //"key=" + this.output.Key + "&id=" + this.output.ServerValidator + "&name=" + this.chanked.meta + "";
    return this.reconscrtuct(); ///
  }

  async _encryptIndex(key) {
    const iv = window.crypto.getRandomValues(new Uint8Array(12));
                                                const indexString = JSON.stringify({index:this.chanked.index,meta:this.chanked.meta});
    const encrypted = await window.crypto.subtle.encrypt(
                                { name: "AES-GCM", iv: iv }, key, new TextEncoder().encode(indexString)
    );

     // Конвертируем IV и зашифрованные данные в формат ArrayBuffer и объединяем их
```

```
    const combined = new Uint8Array(iv.length + encrypted.byteLength);
    combined.set(new Uint8Array(iv), 0);
    combined.set(new Uint8Array(encrypted), iv.length);

    return combined.buffer;
}

FirstReq(){
    socket.emit("FirstReq",this.inputs.fileId);
    return
}

async _GetReqDown(msg) {
    console.log(msg);
    try {
        let inputobj = await this._decryptIndex(msg, this.inputs.password);
        console.log("Расшифрованные данные:", inputobj);
        //fix xss just in case
        if(this.isPlainObject(inputobj)){
```

```
                    document.getElementById('FileName').textContent =
inputobj.meta.name;
                    document.getElementById('FileSize').textContent =
inputobj.meta.size;
          document.getElementById('FileState').textContent = 'Downloading
the file';
          this.chanked.counter = 1;
          this.chanked.index = inputobj.index;
          this.chanked.meta = inputobj.meta;
          return this.Downloadreq(this.chanked.counter)
        }else{
                  document.getElementById('FileState').textContent = 'File
isdamaged';
          console.log("File is broken");
          return
        }
      } catch (error) {
          document.getElementById('FileState').textContent = 'Key does not
match :-( Most probably))';
        //console.error("Ключ не подходит к обьекту((")
        console.error("Ошибка при расшифровке:", error);
```

```
            // Здесь можно добавить дополнительные действия, например, показать сообщение пользователю
        }
    }


    Downloadreq(chank){
        socket.emit("Downloadreq",chank);
        return 1;
    }


    _Downloadhandler(msg){
        console.time("yourFunctionTimer");
        //console.log(msg)
        let encryptedCrc32 = this._crc32(new Uint8Array(msg));
                                                    //console.log(encryptedCrc32, this.chanked.index[this.chanked.counter].E)


        if(encryptedCrc32 != this.chanked.index[this.chanked.counter].E){
            console.error("File is damaged!");
            return
```

```
        }
        this.chanked.chanked[this.chanked.counter] = msg

console.log(Object.keys(this.chanked.index).length,Object.keys(this.chanked.chanked).length)
                                    if(Object.keys(this.chanked.index).length == Object.keys(this.chanked.chanked).length){
            console.log("We downloaded it")
                document.getElementById('FileState').textContent = 'Downloaded, Decryptoing the file';
            return this.DecryptONDown();
        }
        this.chanked.counter++;
        socket.emit("Downloadreq",this.chanked.counter);
        console.timeEnd("yourFunctionTimer");
    }

    async DecryptONDown(){

let assembledFile = new Blob([], { type: "application/octet-stream" });
let chunkIndex = 1; // Индекс чанка начинается с 1
```

```
while (this.chanked.chanked[chunkIndex]) {
   try {
      const encryptedChunk = this.chanked.chanked[chunkIndex];
                                                const rawKey = new Uint8Array(Object.values(this.chanked.index[chunkIndex].K));
      const key = await this._importKey(rawKey);

      const decryptedChunk = await this._decryptData(encryptedChunk, key);
      const decryptedCrc32 = this._crc32(new Uint8Array(decryptedChunk));

      if (decryptedCrc32 !== this.chanked.index[chunkIndex].O) {
            throw new Error(`CRC check failed for chunk ${chunkIndex}: File may be damaged or tampered`);
      }

          assembledFile = new Blob([assembledFile, decryptedChunk], { type: "application/octet-stream" });
      chunkIndex++;
   } catch (error) {
      console.error('Ошибка при обработке файла:', error);
```

```
        return;
    }
}

console.log("File assembly complete");
return this._processAndDownloadFile(assembledFile);
    }

    async _processAndDownloadFile(assembledFile) {
        // Проверка размера файла
        if (assembledFile.size !== this.chanked.meta.size) {
                throw new Error("File size mismatch: the assembled file may be corrupted.");
        }

        // Установка метаданных файла
            const file = new File([assembledFile], this.chanked.meta.name, { type: this.chanked.meta.type });

        // Инициирование скачивания файла
        const url = URL.createObjectURL(file);
```

```
const a = document.createElement("a");
a.href = url;
a.download = this.chanked.meta.name;
document.body.appendChild(a);
a.click();

// Очистка ссылки
window.URL.revokeObjectURL(url);
a.remove();

console.log("File download initiated");

// Здесь можете добавить код для завершения обработки класса
                // Например, очистка используемых ресурсов, вызов callback-функции и т.д.
// ...
console.log("We are done!"); // delete file from memory here))
//load constructer
            document.getElementById('FileState').textContent = 'File was downloaded))';
return this.reconscrtuct(); ///!!!!!!!!!!!!!!! check if it works
```

```
    }


//.....
//.....
//.....
//.....
//.....
//.....
//.....
//.....
//.....
//.....
//.....
    // Создание таблицы для вычисления CRC32
    _makeCRCTable() {
        let c;
        for (let n = 0; n < 256; n++) {
            c = n;
            for (let k = 0; k < 8; k++) {
```

```
            c = ((c & 1) ? (0xEDB88320 ^ (c >>> 1)) : (c >>> 1));
        }
        this.inputs.crc32Table[n] = c;
    }
}

// Вычисление CRC32 для данных
_crc32(data) {
    let crc = 0 ^ (-1);
    for (let i = 0; i < data.length; i++) {
        crc = (crc >>> 8) ^ this.inputs.crc32Table[(crc ^ data[i]) & 0xFF];
    }
    return (crc ^ (-1)) >>> 0;
}



async _generateEncryptionKey() {
    const key = await window.crypto.subtle.generateKey(
        { name: "AES-GCM", length: 256 }, true, ["encrypt", "decrypt"]
    );
```

```
        // Экспорт ключа не обязателен, если вы будете использовать его только в этом контексте
        // Но если вы хотите его сохранить или передать, вам нужно будет его экспортировать
        const exportedKey = await window.crypto.subtle.exportKey("raw", key);
        const keyString = this._arrayBufferToHexString(exportedKey);
        console.log("Сгенерированный ключ (hex):", keyString);

        return {key:key}; // Возвращает только ключ
    }


    _arrayBufferToHexString(buffer) {
        return Array.prototype.map.call(new Uint8Array(buffer), byte => {
            return ('0' + (byte & 0xFF).toString(16)).slice(-2);
        }).join('');
    }

    async _decryptIndex(combined, hexKey) {
        try {
            // Преобразуем hex-ключ в ArrayBuffer
```

```
    const keyBuffer = this._hexStringToArrayBuffer(hexKey);

    // Импортируем ключ
    const key = await window.crypto.subtle.importKey(
        "raw", keyBuffer, { name: "AES-GCM" }, false, ["decrypt"]
    );

    // Разделяем IV и зашифрованные данные
    const iv = combined.slice(0, 12);
    const encryptedData = combined.slice(12);

    // Расшифровываем данные
    const decrypted = await window.crypto.subtle.decrypt(
        { name: "AES-GCM", iv: iv }, key, encryptedData
    );
    const indexString = new TextDecoder().decode(decrypted);
    return JSON.parse(indexString);
} catch (e) {
    console.error("Ошибка при расшифровке:", e);
    throw e; // или обработайте ошибку по своему усмотрению
```

```
    }
  }

    // Вспомогательная функция для преобразования hex-строки в ArrayBuffer
  _hexStringToArrayBuffer(hexString) {
    const bytes = new Uint8Array(Math.ceil(hexString.length / 2));
    for (let i = 0, j = 0; i < hexString.length; i += 2, j++) {
      bytes[j] = parseInt(hexString.substr(i, 2), 16);
    }
    return bytes.buffer;
  }

  isPlainObject(obj) {
    return obj && typeof obj === 'object' && !Array.isArray(obj) && obj !== null;
  }


concatenateBuffers(...buffers) {
  let totalLength = buffers.reduce((acc, value) => acc + value.byteLength, 0);
```

```
    let result = new Uint8Array(totalLength);

    let offset = 0;
    buffers.forEach(buffer => {
        result.set(new Uint8Array(buffer), offset);
        offset += buffer.byteLength;
    });

    return result.buffer;
}

// Функция для импорта сырого ключа
async _importKey(rawKey) {
    return window.crypto.subtle.importKey(
        "raw",
        rawKey,
        "AES-GCM",
        false,
        ["encrypt", "decrypt"]
    );
```

```
}

// Функция шифрования
async _encryptData(data, key) {
        const iv = window.crypto.getRandomValues(new Uint8Array(12)); // Вектор инициализации

    const encryptedData = await window.crypto.subtle.encrypt(
        {
            name: "AES-GCM",
            iv: iv
        },
        key,
        data
    );

    return this.concatenateBuffers(iv, encryptedData);
}

// Функция дешифрования
```

```
async _decryptData(encryptedBuffer, key) {
    const ivSize = 12; // Размер IV
    const iv = encryptedBuffer.slice(0, ivSize);
    const encryptedData = encryptedBuffer.slice(ivSize);

    return window.crypto.subtle.decrypt(
        {
            name: "AES-GCM",
            iv: iv
        },
        key,
        encryptedData
    );
}

// Тестирование функций шифрования и дешифрования
async testEncryptionDecryption() {
    const testData = new TextEncoder().encode("Тестовые данные");

    try {
```

```
        const rawKey = window.crypto.getRandomValues(new Uint8Array(32)); // 256-битный ключ
        const key = await this._importKey(rawKey);

        const encryptedData = await this._encryptData(testData, key);
        console.log('Зашифрованные данные:', encryptedData);

        const decryptedData = await this._decryptData(encryptedData, key);
                                console.log('Дешифрованные данные:', new TextDecoder().decode(decryptedData));
    } catch (error) {
        console.error('Ошибка:', error);
    }
}
}
```

index.html

```
<!DOCTYPE html>
<html lang="en" style="background-color: darkslategrey;">
<head>
```

```
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Send File</title>
</head>
<body>
    <h1>Send a File</h1>
    <input type="file" id="fileInput">

    <button onclick="handleAction(true)">Send File</button><br><br>

    <input type="password" id="passwordInput" placeholder="Password">
    <input type="text" id="fileIdInput" placeholder="File ID">
    <button onclick="handleAction(false)">Get File</button>

    <p id="FileName"></p>
    <p id="FileSize"></p>
    <p id="FileState"></p>
    <p id="FileID"></p>
    <p id="FileKEY"></p>
```

```
<script src="/socket.io/socket.io.js"></script>
<script src="main.js"></script>
<script>
    var runner = new Runner();  // Инициализация runner здесь

    function handleAction(isSender) {
        const fileInput = document.getElementById('fileInput');
        const passwordInput = document.getElementById('passwordInput');
        const fileIdInput = document.getElementById('fileIdInput');
        const selectedFile = isSender ? fileInput.files[0] : null;

                    runner.start(isSender, selectedFile, passwordInput.value, fileIdInput.value);
        //runner.start(isSender, selectedFile, "", "");
    }

    //up list
    socket.on('StarterPackId', (File) => runner._sendStart(File));
    socket.on('StartRunner', (File) => runner._sendStartRunner(File));
    socket.on('NextPlease', (File) => runner._NextToSend(File));
```

```
        socket.on('doneHere', (File) => runner._GetLink(File));


        //down list
        socket.on('GetReqDown', (File) => runner._GetReqDown(File))
        socket.on('Downloadhandler', (File) => runner._Downloadhandler(File))
</script>
</body>
</html>
```