Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«МЕТОД ЗАХИЩЕНОГО ПЕРЕДАВАННЯ ФАЙЛІВ»

Виконав: студент групи 1БС-23м
спеціальності 125 Кібербезпека та захист інформації
_____ Сергій ГИРЖЕУ

Керівник: д.т.н., професор, зав. кафедри ЗІ
_____ Володимир ЛУЖЕЦЬКИЙ
«13» 12 2024 р.

Опонент: д.т.н., професор, зав. кафедри ПЗ
_____ Олександр РОМАНЮК
«13» 12 2024 р.

Допущено до захисту
Завідувач кафедри ЗІ
д. т. н., проф.
_____ Володимир ЛУЖЕЦЬКИЙ
«13» 12 2024 р.

Вінниця ВНТУ – 2024 року

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра захисту інформації
Ступінь вищої – магістр
Галузь знань – 12 Інформаційні технології
Спеціальність – 125 Кібербезпека та захист інформації
Освітньо-професійна програма – Безпека інформаційних і комунікаційних систем

ЗАТВЕРДЖУЮ
Завідувач кафедри ЗІ,
д. т. н., проф.
_____ Володимир ЛУЖЕЦЬКИЙ
«18» 09 2024 року

З А В Д А Н Н Я
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Гиржеу Сергію Дмитровичу

1. Тема роботи: «Метод захищеного передавання файлів» керівник роботи: Володимир Андрійович Лужецький, д.т.н., проф., завідувач кафедри ЗІ, затверджені наказом ректора ВНТУ від 17.09.2024р. №310

2. Строк подання студентом роботи 13 грудня 2024 р.

3. Вихідні дані до роботи:
   – канали для передавання файлів: Telegram, Signal, Gmail;
   – обсяг файлу – не більше 2^{24} байт;
   – кількість сегментів, на які розбивається файл – 3;
   – довжина секретного ключа – 64 біт.

4. Зміст текстової частини: Вступ. 1. Аналіз відомих рішень. 2. Метод захищеного передавання файлів. 3. Розробка програмного засобу для захищеного передавання даних та тестування. 4. Економічна частина. Висновки. Перелік використаних джерел. Додатки.

5. Перелік ілюстративного матеріалу: Узагальнений алгоритм сегментації; Узагальнений алгоритм відновлення файлу; Алгоритм генерації секретного ключа; Алгоритм ініціалізації масивів та розподілу байтів файлу; Алгоритм формування ущільненого та відновленого файлу; Алгоритм формування HMAC; Алгоритм перевірки HMAC.

6. Консультанти розділів роботи

Роз-діл	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Володимир ЛУЖЕЦЬКИЙ, д.т.н., проф., зав. кафедри ЗІ		
2	Володимир ЛУЖЕЦЬКИЙ, д.т.н., проф., зав. кафедри ЗІ		
3	Володимир ЛУЖЕЦЬКИЙ, д.т.н., проф., зав. кафедри ЗІ		
4	Ольга РАТУШНЯК, к.т.н., доцент, доцент каф. ЕПВМ		

7. Дата видачі завдання 18 вересня 2024 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської дипломної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз завдання. Вступ	18.09.2024 – 21.09.2024	
2	Аналіз літературних джерел за напрямком магістерської кваліфікаційної роботи	22.09.2024 – 27.09.2024	
3	Науково-технічне обґрунтування	28.09.2024 – .09.2024	
4	Розробка технічного завдання	23.09.2024 – 29.09.2024	
5	Розробка рішень	30.09.2024 – 12.10.2024	
6	Практична реалізація, моделювання, експериментування, результати	14.10.2024 – 10.11.2024	
7	Розробка розділу економічного обґрунтування доцільного розробки	11.11.2024 – 17.11.2024	
8	Аналіз виконання ТЗ, висновки	18.11.2024 – 24.11.2024	
9	Оформлення пояснювальної записки	25.11.2024 – 30.11.2024	
10	Попередній захист та доопрацювання МКР	28.11.2024 – 30.11.2024	
11	Перевірка магістерської роботи на наявність плагіату	02.12.2024 – 10.12.2024	
12	Представлення МКР до захисту	11.12.2024 – 14.12.2024	
13	Захист МКР	14.12.2024 – 21.12.2024	

Студент _____ Сергій ГИРЖЕУ

Керівник роботи _____ Володимир ЛУЖЕЦЬКИЙ

ЗМІСТ

АНОТАЦІЯ

Магістерська кваліфікаційна робота складається з 84 сторінок формату А4, на яких є 19 рисунків, 11 таблиць, 41 формул, список використаних джерел містить 37 найменувань.

Магістерська кваліфікаційна робота присвячена розробці методу захищеного передавання файлів. У роботі проведено аналіз існуючих рішень, виділено їхні переваги та недоліки. Описано новий метод сегментації файлів, який поєднує криптографічні алгоритми з механізмами перевірки цілісності на основі HMAC. Розроблено програмний засіб, що реалізує запропонований метод, включаючи алгоритми шифрування, розшифрування, перевірки HMAC та механізми логування. Проведено тестування, яке підтвердило коректність реалізації та відповідність вимогам безпеки. Запропонований засіб підвищує стійкість до атак на цілісність даних і може бути використаний у реальних умовах для забезпечення захисту конфіденційної інформації.

Ключові слова: шифрування, сегментація даних, HMAC, перевірка цілісності, криптографія, алгоритм, кібербезпека, передавання файлів, ключ шифрування, відновлення файлів, аутентифікація, логування, тестування, канали передачі, захист даних.

ABSTRACTS

The master's thesis consists of 84 pages of A4 format, which includes 19 figures, 11 tables, 41 formulas, and a list of references containing 37 titles.

The master's thesis is devoted to the development of a method for secure file transfer. The research includes an analysis of existing solutions, highlighting their advantages and disadvantages. A novel file segmentation method is proposed, integrating cryptographic algorithms with integrity verification mechanisms based on HMAC. A software tool implementing the proposed method was developed, encompassing encryption, decryption, HMAC verification, and logging mechanisms. Testing demonstrated the correctness of implementation and compliance with security requirements. The proposed solution enhances resistance to integrity attacks and can be utilized in real-world scenarios to protect confidential information.

Keywords: encryption, data segmentation, HMAC, integrity verification, cryptography, algorithm, cybersecurity, file transfer, encryption key, file recovery, authentication, logging, testing, transmission channels, data protect.

ВСТУП

Сучасний етап розвитку інформаційних технологій супроводжується значним зростанням обсягів передавання інформації, що, у свою чергу, ускладнює загрози для її безпеки. Особливо це стосується конфіденційних даних, що передаються через глобальні інформаційні мережі, де ризики перехоплення, модифікації або витоку інформації можуть мати серйозні наслідки. Забезпечення безпеки під час передавання даних є однією з головних проблем у сфері кібербезпеки, яка стосується захисту інформаційних систем на рівні окремих користувачів, підприємств, організацій та державних установ. [1].

Передавання даних через мережі завжди супроводжується низкою загроз, таких як перехоплення, несанкціонований доступ або модифікація файлів. Більшість сучасних підходів, включно із традиційними криптографічними методами, демонструють високу ефективність у певних умовах. Проте в умовах зростання обсягів переданої інформації, збільшення обчислювальних потужностей зловмисників та постійної еволюції атак, існуючі рішення часто не забезпечують належного рівня безпеки. Виникає необхідність розробки нових методів, які б поєднували криптографічні технології та інноваційні підходи до організації процесу передавання даних [2].

Виникає необхідність розробки нового методу для захищеного передавання файлів, який дозволить мінімізувати ризики несанкціонованого доступу до переданої інформації та її модифікації в процесі передачі. Завдання полягає у створенні такої системи захисту, що поєднуватиме криптографічні технології та методи перевірки цілісності файлів. Зважаючи на постійне зростання обсягів переданої через мережу інформації, забезпечення її захисту є необхідною умовою для розвитку та функціонування багатьох сучасних інформаційних систем. Вивчення існуючих підходів до захисту передавання даних дозволяє визначити основні вразливості і запропонувати нові рішення для їх усунення.

Метою дослідження є підвищення рівня захисту даних, що передаються мережею шляхом розробки методу захищеного передавання даних та засобу, який його реалізує.

Для досягнення цієї мети необхідно вирішити такі задачі:

- аналіз існуючих методів захисту інформації при передаванні файлів;
- розробка методів зашифрування та розшифрування файлу;
- розробка методу контролю цілісності файлу;
- розробка засобу для захищеного передавання файлів та його тестування.

Об'єктом дослідження є процес захищеного передавання файлів.

Предметом дослідження методи та засоби захищеного передавання даних.

Наукова новизна роботи полягає в запропонованому комплексному підході до захищеного передавання файлів, який включає в себе не тільки використання криптографічних методів шифрування, але й застосування додаткових методів перевірки цілісності файлів. Такий підхід дозволяє створити більш надійний механізм захисту інформації, який здатний ефективно протистояти як відомим, так і новим типам кіберзагроз.

Практична цінність роботи полягає у розробці методу захищеного передавання файлів, який може бути впроваджений у реальні інформаційні системи для підвищення рівня безпеки при обміні даними. Це має особливе значення для таких сфер, як фінансові послуги, електронна комерція, державні органи, а також у приватних компаніях, які працюють з конфіденційною інформацією. Запропонований метод може бути використаний як основа для розробки програмного забезпечення для захищеного передавання даних у різних галузях.

У процесі виконання дослідження будуть застосовані методи математичного моделювання та комп'ютерного тестування, що дозволить не лише розробити метод, а й провести його експериментальну перевірку в умовах реальних інформаційних систем.

1 АНАЛІЗ ВІДОМИХ РІШЕНЬ

1.1 Аналіз методів захищеного передавання файлів

Сучасні інформаційні системи надають користувачам широкий вибір інструментів для передавання файлів, зокрема, популярні месенджери, хмарні сховища, електронну пошту та інші спеціалізовані сервіси. Вибір оптимального методу для захищеного передавання файлів залежить від вимог до безпеки, швидкості передачі, зручності використання, підтримки різних платформ та додаткових засобів захисту [3].

Криптографічні методи шифрування є основою забезпечення захисту даних під час їх передавання. Вони використовуються для конфіденційного обміну інформацією, перевірки її цілісності та автентифікації учасників обміну. Серед найбільш поширених методів, які застосовуються для захищеного передавання файлів, - симетричне шифрування, асиметричне шифрування та гешування. Кожен із цих методів має свої особливості, переваги та недоліки, що обумовлюють їх використання у різних сценаріях захисту [4].

Симетричне шифрування використовує один ключ для шифрування та розшифрування даних [5]. Це означає, що обидві сторони - відправник та отримувач - повинні мати доступ до цього секретного ключа, який необхідно передати безпечно перед початком обміну інформацією. Одним із найбільш популярних алгоритмів симетричного шифрування є AES (Advanced Encryption Standard). Цей алгоритм використовує блокове шифрування з ключами довжиною 128, 192 або 256 біт, що забезпечує високий рівень захисту при порівняно низьких вимогах до обчислювальних ресурсів.

Переваги симетричного шифрування:

- Висока швидкість шифрування та розшифрування, що робить його оптимальним для великих обсягів даних;

- Менша обчислювальна складність порівняно з асиметричним шифруванням, що дозволяє використовувати його на пристроях з обмеженими ресурсами;

- Простота реалізації та відносно низьке споживання пам'яті.

Недоліки симетричного шифрування:

- Проблема безпечної передачі секретного ключа між відправником та отримувачем, що створює ризик компрометації ключа;

- Необхідність зберігання окремого ключа для кожної пари користувачів або учасників обміну, що може ускладнити управління ключами в великих системах.

Асиметричне шифрування (англ. - asymmetric coding) - набір методів криптографічного шифрування, в яких використовують два ключі - таємний(приватний) і відкритий; жоден із ключів не може бути обчислений з іншого за прийнятий час. Таке шифрування ще називають шифруванням з відкритим ключем (англ. - public key coding) [6]. Відкритий ключ використовується для шифрування даних, і він може бути вільно переданий будь-якому користувачеві. Закритий ключ зберігається в секреті та використовується для розшифрування даних. Найвідомішим алгоритмом асиметричного шифрування є RSA (Rivest-Shamir-Adleman), який ґрунтується на складності факторизації великих чисел. Інший поширений алгоритм - ECC (Elliptic Curve Cryptography), який використовує еліптичні криві для створення криптографічно стійких ключів, але з меншою довжиною, що знижує обчислювальну складність.

Переваги асиметричного шифрування:

- Відсутність потреби в передачі закритого ключа між сторонами, що підвищує рівень безпеки при ініціалізації захищеного з’єднання;

- Можливість використання відкритого ключа для автентифікації та підпису даних, що забезпечує додатковий рівень захисту при обміні файлами;

- Зручність управління ключами, оскільки для кожного користувача необхідно зберігати лише одну пару ключів.

Недоліки асиметричного шифрування:

- Висока обчислювальна складність, яка потребує більших ресурсів для шифрування та розшифрування порівняно з симетричним шифруванням;

- Повільність обробки великих обсягів даних, через що асиметричне шифрування часто використовується тільки для ініціалізації сеансу обміну ключами, а для подальшого обміну використовується симетричне шифрування [7].

Гешування — це процес перетворення вхідних даних на унікальний набір символів фіксованої довжини, відомий як геш. Ця одностороння функція широко використовується для перевірки цілісності переданої інформації, автентифікації та зберігання паролів. На відміну від шифрування, яке можна розшифрувати, гешування не передбачає зворотного перетворення до початкових даних. Навіть мінімальні зміни у вхідних даних призводять до значних змін у результаті геш-функції, що дозволяє виявляти будь-які модифікації даних під час їх передавання [8].

Одним з найпопулярніших алгоритмів гешування є SHA-256 (Secure Hash Algorithm 256-bit), який забезпечує високу криптографічну стійкість і використовується для перевірки цілісності файлів та автентифікації. Інші алгоритми, такі як MD5, вважаються менш надійними через можливість колізій, тобто ситуацій, коли різні вхідні дані можуть мати однаковий геш [8].

Переваги гешування:

- Висока швидкість обчислення, що робить його ефективним для перевірки цілісності даних при передаванні;

- Можливість використання для автентифікації та підтвердження авторства даних через механізми цифрового підпису;

- Незалежність від ключів, оскільки гешування не потребує використання парних ключів для обробки даних.

Недоліки гешування: Односторонній характер гешування означає, що дані не можуть бути відновлені після їх гешування, що робить його непридатним для шифрування інформації, яку потрібно відновити.

- Наявність менш надійних алгоритмів гешування (таких як MD5), які можуть бути вразливими до атак на колізії.

- Ризик атак на довгі повідомлення, оскільки можливі спроби підібрати інший набір даних з тим самим гешем, що створює загрозу для цілісності.

У більшості сучасних систем захищеного передавання файлів ці методи використовуються в комплексі. Наприклад, під час ініціалізації захищеного з'єднання використовується асиметричне шифрування для обміну ключами, після чого дані передаються за допомогою швидшого симетричного шифрування. Гешування застосовується для перевірки цілісності файлів та автентифікації сторін. Такий підхід дозволяє досягти високого рівня захисту, ефективності та швидкості при передаванні конфіденційної інформації [9].

Розглянемо особливості популярних методів захищеного передавання файлів, зокрема в месенджерах, хмарних сховищах та через електронну пошту.

WhatsApp є одним із найбільш популярних месенджерів для особистого та групового спілкування, що забезпечує захищене передавання файлів завдяки наскрізному шифруванню (end-to-end encryption). Кожен файл, що надсилається через WhatsApp, шифрується на пристрої відправника та розшифровується тільки на пристрої отримувача [10].

Переваги:

- Високий рівень безпеки завдяки наскрізному шифруванню;

- Простота та зручність використання для обміну файлами різного типу;

- Широка підтримка платформ, що забезпечує доступ до месенджера з різних пристроїв.

Недоліки:

- Обмеження на максимальний розмір файлу (100 МБ), що ускладнює передачу великих файлів;

- Обмеження на тип файлів, що може створювати труднощі для користувачів з нестандартними форматами;

- Невеликий контроль над збереженими даними після передачі, оскільки файли зберігаються в загальному сховищі пристрою.

Telegram є месенджером, що також забезпечує високий рівень захисту під час передавання файлів, підтримуючи як клієнт-серверне шифрування для загальних чатів, так і наскрізне шифрування для "секретних чатів" [11]. У секретних чатах файли шифруються на пристрої відправника та розшифровуються тільки на пристрої отримувача, що виключає можливість доступу до них третіх осіб.

Переваги:

- Можливість передавати файли великого розміру (до 2 ГБ), що є зручним для відеофайлів, документів тощо;

- Наявність наскрізного шифрування в секретних чатах, що підвищує рівень конфіденційності;

- Підтримка широкого спектру платформ, що дозволяє використовувати Telegram на різних пристроях.

Недоліки:

- Відсутність наскрізного шифрування для загальних чатів, що може створювати ризики у випадку атак на сервери Telegram;

- Обмеження функцій захисту у випадку використання загальних чатів для передачі конфіденційних файлів;

- Залежність від серверної інфраструктури для зберігання файлів, що може призводити до втрати контролю над даними.

Signal є месенджером з відкритим кодом, який вважається одним із найзахищеніших доступних сьогодні [12]. Signal використовує наскрізне шифрування для всіх повідомлень та файлів, що надсилаються через платформу. Це означає, що тільки відправник і отримувач мають доступ до переданої інформації.

Переваги:

- Високий рівень безпеки завдяки наскрізному шифруванню для всіх типів повідомлень і файлів;

- Додаткові функції конфіденційності, такі як автоматичне видалення повідомлень та файлів через таймер;

- Відкритий вихідний код, що дозволяє аудит безпеки з боку сторонніх експертів.

Недоліки:

- Обмеження на розмір файлів для передачі, що може обмежувати зручність використання для великих файлів;

- Відсутність розширених функцій для бізнесу або великих команд;

- Обмежена підтримка багатоплатформенності порівняно з іншими месенджерами.

Google Drive та Dropbox не месенджерами, але вони часто використовуються для передавання файлів із елементами захисту [13]. Застосовують клієнт-серверне шифрування для зберігання файлів та використовують HTTPS для захисту передачі даних між сервером і клієнтом.

Переваги:

- Підтримка великих обсягів інформації, зручний інтерфейс для управління файлами.

- Можливість швидко ділитися файлами з іншими користувачами через посилання або запрошення.

- Доступ до файлів з різних платформ, що забезпечує гнучкість використання.

Недоліки:

- Відсутність наскрізного шифрування, що теоретично дозволяє адміністраторам сервісів доступ до файлів.

- Залежність від серверів компанії, що може створювати ризики для конфіденційності інформації.

- Потреба в додатковому захисті для чутливої інформації, особливо в умовах високих вимог до безпеки.

Google Drive забезпечує захист файлів за допомогою клієнт-серверного шифрування, де файли шифруються при передаванні між пристроєм користувача та сервером Google. Google використовує шифрування даних у спокої (AES-256) та шифрування під час передавання (TLS) [13].

Переваги:

- Велика гнучкість у передаванні та зберіганні файлів;
- Простота надання доступу до файлів іншим користувачам;
- Підтримка великих обсягів файлів.

Недоліки:

- Відсутність наскрізного шифрування, через що Google теоретично може отримати доступ до файлів;
- Можливість доступу до файлів адміністраторами сервісу, що може створювати ризики для конфіденційності.

Dropbox також використовує клієнт-серверне шифрування для захисту файлів під час передавання та шифрує файли, що зберігаються на сервері, за допомогою AES-256 [14].

Переваги:

- Зручний інтерфейс для організації та передавання файлів;
- Можливість керування доступом до файлів і їхнього спільного використання.

Недоліки:

- Відсутність наскрізного шифрування;
- Доступ до файлів можуть мати адміністратори Dropbox, що створює ризики для конфіденційності.

OneDrive від Microsoft також використовує TLS для шифрування під час передавання даних та AES-256 для захисту даних у спокої. OneDrive підтримує функцію "Персонального сховища", яка забезпечує додатковий рівень захисту за допомогою багатофакторної автентифікації [15].

Переваги:

- Можливість інтеграції з іншими сервісами Microsoft;
- Додатковий захист у "Персональному сховищі".

Недоліки:

- Відсутність наскрізного шифрування;
- Потенційний доступ до файлів адміністраторами Microsoft.

Gmail використовує TLS для шифрування даних під час передавання між поштовими серверами та клієнтськими пристроями. Gmail підтримує багатофакторну автентифікацію для додаткового захисту. Водночас Gmail не забезпечує наскрізного шифрування для електронної пошти, тому дані можуть бути доступними для адміністраторів Google [16].

Переваги:

- Захищене передавання даних через TLS;
- Інтеграція з іншими сервісами Google**;**
- Відсутність наскрізного шифрування;
- Потенційний доступ до даних адміністраторами Google.

Outlook від Microsoft використовує TLS для шифрування під час передавання даних, а також підтримує шифрування S/MIME для конфіденційної електронної пошти [17]. Це дає можливість відправляти зашифровані електронні листи, які можуть бути прочитані тільки отримувачем, що має відповідний ключ.

Переваги:

- Підтримка шифрування S/MIME для конфіденційного листування;
- Інтеграція з іншими продуктами Microsoft.

Недоліки:

- Потреба у налаштуванні S/MIME для кожного отримувача;
- Відсутність наскрізного шифрування за замовчуванням [18-19].

Після проведення детального аналізу різних засобів для передавання файлів, була створена таблиця порівнянь цих засобів(1.1).

Проведений аналіз демонструє, що хоча існуючі сервіси пропонують різноманітні засоби захисту даних, жоден із них не позбавлений недоліків. Зокрема, месенджери забезпечують захист під час передавання даних, але деякі з них не мають наскрізного шифрування в загальних чатах. Хмарні сховища гарантують високий рівень шифрування збережених файлів, але доступ до них може мати адміністрація сервісів. Поштові сервіси, такі як Gmail і Outlook,

захищають дані під час передачі, але не забезпечують наскрізного шифрування, що створює ризики для конфіденційності.

Таблиця 1.1 – Порівняння засобів передавання файлів

	Метод шифрування	Переваги	Недоліки
WhatsApp	AES-256, наскрізне	Високий рівень захисту під час передачі, наскрізне шифрування (E2EE).	Резервні копії без наскрізного шифрування, обмеження на розмір файлів.
Telegram	MTProto	Підтримка великих файлів (до 2 ГБ), секретні чати з E2EE.	Відсутність наскрізного шифрування у звичайних чатах.
Signal	Double Ratchet Protocol	Повне наскрізне шифрування для всіх передач, відкритий вихідний код.	Обмеження на розмір файлів, менш поширений серед користувачів.
Google Drive	AES-256, TLS	Зручність передачі та зберігання великих файлів, інтеграція з іншими сервісами.	Відсутність E2EE, можливий доступ адміністрації сервісу до даних.
Dropbox	AES-256, TLS	Швидкий доступ до файлів, синхронізація між пристроями.	Відсутність наскрізного шифрування, залежність від централізованих серверів.
Gmail	TLS	Захист передачі даних через TLS, інтеграція з іншими сервісами Google.	Відсутність наскрізного шифрування, можливий доступ адміністрації сервісу.
OneDrive	AES-256, TLS	Додатковий захист у "Персональному сховищі", інтеграція з Microsoft 365.	Відсутність наскрізного шифрування, потенційний доступ адміністрації.

Таким чином, аналіз існуючих методів захищеного передавання файлів показує, що сучасні сервіси, зокрема месенджери, хмарні сховища та поштові платформи (Gmail), застосовують різні методи шифрування, такі як AES-256, TLS та протоколи наскрізного шифрування.

Недоліки існуючих методів:

- Месенджери, попри наявність наскрізного шифрування, мають обмеження щодо розміру файлів або застосовують E2EE лише у певних режимах (наприклад, секретні чати Telegram).
- Хмарні сервіси забезпечують зручність передачі великих обсягів даних, але відсутність наскрізного шифрування дозволяє адміністраторам сервісів отримувати доступ до файлів, що створює ризики компрометації конфіденційної інформації.
- Поштові платформи, такі як Gmail, використовують TLS для захисту даних під час передачі, проте файли не шифруються наскрізно, що також залишає можливість доступу до них на сервері.

Ці недоліки, включаючи обмеження в шифруванні, ризик компрометації даних через адміністративний доступ та залежність від централізованої інфраструктури, вказують на необхідність удосконалення існуючих підходів до забезпечення конфіденційності, цілісності та доступності інформації, зокрема шляхом впровадження нових методів шифрування та перевірки даних, які враховують сучасні виклики у сфері кібербезпеки.

1.2 Аналіз загроз для засобів передавання файлів

Під час передавання файлів у відкритих мережах існує низка загроз, які можуть поставити під загрозу конфіденційність, цілісність і доступність даних. кіберзлочинці та інші зловмисники використовують різноманітні методи для несанкціонованого доступу до файлів, їхнього перехоплення або модифікації. У цьому розділі розглянемо основні типи загроз для засобів передавання файлів,

які можуть вплинути на безпеку та надійність даних під час їхнього обміну через месенджери, хмарні сховища або електронну пошту [20].

Перехоплення даних (Eavesdropping) є однією з найбільш поширених загроз для засобів передавання файлів. Зловмисники можуть здійснювати атаки "людина посередині" (MITM), перехоплюючи трафік між відправником і отримувачем. Це дозволяє їм отримати доступ до даних у процесі передавання, навіть якщо вони шифруються [21]. Перехоплення зазвичай відбувається внаслідок:

- Використання незахищених мереж, таких як відкриті Wi-Fi;
- Вразливостей у протоколах зв'язку, наприклад, у TLS, якщо налаштування виконано неправильно;
- Відсутності наскрізного шифрування, що дозволяє зловмисникам отримати доступ до даних на сервері постачальника послуг.

Наслідки:

Зловмисники можуть отримати конфіденційну інформацію, що передається, зокрема паролі, особисті дані, фінансову інформацію тощо. Це може призвести до фінансових втрат, крадіжки даних та компрометації системи.

Модифікація даних (Data Tampering) означає несанкціоновану зміну файлів або інформації під час їхнього передавання [22]. Це може відбутися через атаки MITM або за допомогою шкідливого програмного забезпечення, яке впроваджує зміни у файли на сервері або на пристрої користувача до передачі. Атаки на цілісність даних є серйозною загрозою, оскільки змінені файли можуть залишатися непоміченими.

Приклади атак:

- Впровадження шкідливих скриптів або коду у передаванні файли, наприклад, документи або зображення;
- Модифікація важливих файлів для підриву їхньої достовірності (наприклад, підміна контрактів, банківських реквізитів тощо).

Наслідки:

Модифіковані файли можуть спричинити фінансові збитки, підрив довіри або

юридичні проблеми, особливо якщо це стосується важливої комерційної документації або особистих даних.

Несанкціонований доступ (Unauthorized Access) виникає тоді, коли зловмисники отримують доступ до файлів або облікових записів без дозволу власника [23]. Це може відбутися в результаті зламу облікового запису, використання слабких паролів або вразливостей у системі автентифікації. Несанкціонований доступ є однією з головних загроз для даних, що передаються через хмарні сервіси або месенджери.

Причини:

- Використання слабких або повторно використаних паролів, які можуть бути легко вгадані або зламані;

- Відсутність двофакторної автентифікації, яка забезпечує додатковий рівень захисту;

- Невикористання сучасних методів шифрування або їх неправильне налаштування.

Наслідки:

Зловмисники можуть отримати доступ до файлів користувача, викрасти конфіденційні дані або навіть видалити інформацію. Це створює ризик втрати даних та шкоди для репутації, особливо для комерційних або державних установ.

Фішинг (Phishing) є популярним методом соціальної інженерії, який використовується для обману користувачів та отримання доступу до конфіденційної інформації, зокрема до облікових записів [24]. Зловмисники надсилають підроблені електронні листи або повідомлення, які виглядають як справжні, з метою отримання даних для входу або інших конфіденційних відомостей. Фішинг може бути особливо небезпечним при передаванні файлів через електронну пошту або месенджери.

Способи фішингових атак:

- Підроблені повідомлення з проханням надати дані для входу або завантажити шкідливий файл;

- Фальшиві веб-сайти, які імітують справжні сервіси (наприклад, Google Drive або Dropbox);

- Вкладення шкідливих файлів або посилань у повідомленнях, які при відкритті можуть завантажити шкідливе програмне забезпечення.

Наслідки:

Користувачі можуть надати свої облікові дані зловмисникам, що дозволить їм отримати доступ до файлів або інформації в хмарних сховищах, електронній пошті чи месенджерах. Це може призвести до витоку конфіденційних даних і фінансових збитків.

Атака на сервери (Server-Side Attacks), які зберігають файли або керують передаванням даних, є ключовими елементами інфраструктури будь-якого сервісу. Атаки на сервери можуть включати DDoS-атаки, злам баз даних, вразливості в програмному забезпеченні сервера та інші види атак, що можуть призвести до витоку або втрати даних [25].

Типи атак:

- DDoS-атаки, які перевантажують сервер і роблять його недоступним для користувачів;

- Вразливості в базах даних, які дозволяють зловмисникам отримати доступ до збережених даних;

- Злам облікових записів адміністраторів, що дає доступ до керування сервером і збереженими даними.

Наслідки:

Атаки на сервери можуть призвести до витоку великої кількості даних, порушення роботи сервісу та недоступності файлів для користувачів. Крім того, компрометація серверів ставить під загрозу безпеку всіх користувачів сервісу, особливо якщо їхні дані не шифруються наскрізно.

Шкідливе програмне забезпечення (Malware) може впливати на захист даних, що передаються, якщо воно інфікує пристрій користувача або сервер. Зловмисники можуть впроваджувати віруси, трояни або шпигунське програмне

забезпечення, яке перехоплює дані користувача, змінює файли або передає інформацію на сервер зловмисників [26].

Типи шкідливого ПЗ:

- Шпигунське ПЗ, яке збирає конфіденційні дані та передає їх зловмиснику;

- Трояни, що вбудовуються у файли та можуть вразити системи користувачів при завантаженні;

- Віруси, які заражають передаванні файли та поширюються через спільні мережі.

Наслідки:

Інфіковані файли можуть призвести до поширення шкідливого ПЗ у корпоративній мережі, викрадання конфіденційних даних або порушення цілісності файлів. Крім того, користувачі можуть втратити доступ до своїх даних у разі шкідливих атак на цілісність.

Загалом, засоби передавання файлів схильні до різноманітних загроз, які можуть ставити під загрозу конфіденційність, цілісність та доступність даних. Запобігання цим загрозам вимагає застосування наскрізного шифрування, надійної автентифікації, захисту серверів і підвищення обізнаності користувачів про фішинг та інші методи соціальної інженерії.

1.3 Постановка задачі

Аналіз, виконаний у попередніх розділах, показав, що існуючі засоби для захищеного передавання файлів мають ряд недоліків, зокрема обмеження у забезпеченні конфіденційності, цілісності та автентифікації даних, що передаються через відкриті мережі. Загрози, з якими стикаються користувачі при передаванні файлів, вимагають розробки більш надійного підходу, який використовує сучасні криптографічні методи. Завдання даної роботи полягає у створенні нового методу захищеного передавання файлів, який

забезпечуватиме захист від несанкціонованого доступу, перехоплення даних та їх модифікації.

Основні завдання роботи включають розробку та впровадження методології, яка забезпечуватиме конфіденційність і цілісність файлів під час їх передавання, а також мінімізуватиме ризики несанкціонованого доступу. Серед ключових компонентів роботи — створення механізму шифрування даних для захисту конфіденційності, а також реалізація методів перевірки цілісності даних для виявлення змін у файлах.

Робота також передбачає створення прототипу, який буде використовуватися для тестування і оцінки ефективності розробленого методу, включаючи захист від різних типів атак. Основними користувачами запропонованого рішення є організації та користувачі, що працюють із конфіденційними даними і потребують надійного засобу для захищеного обміну інформацією, зокрема окремі користувачі, фінансові установи, державні організації та компанії, що оперують персональними даними.

1.4 Висновки до розділу 1

Виконаний аналіз показав, що сучасні методи захищеного передавання файлів базуються на використанні різних криптографічних підходів, таких як симетричне та асиметричне шифрування, а також механізмів гешування для перевірки цілісності даних. Кожен з методів має свої переваги та обмеження, які визначають сферу їхнього застосування.

Зокрема, симетричне шифрування забезпечує високу швидкість обробки даних, але має складнощі з безпечною передачею секретних ключів. Асиметричне шифрування дозволяє вирішити цю проблему, проте є менш ефективним для великих обсягів даних. Механізми гешування широко застосовуються для перевірки цілісності файлів, але вони не забезпечують збереження конфіденційності.

Проаналізовані недоліки існуючих методів та ризики, пов'язані з їх використанням, підтвердили необхідність розробки нових підходів до захищеного передавання файлів. Новий метод з урахуванням його адаптації до сучасних загроз, має показати кращий потенціал для забезпечення вищого рівня захисту інформації в умовах відкритих мереж.

2 МЕТОД ЗАХИЩЕННОГО ПЕРЕДАВАННЯ ФАЙЛІВ

2.1 Теоретичні основи методу поділу файлів

В умовах сучасних загроз інформаційній безпеці передавання файлів у цифрових мережах потребує нових підходів, що мінімізують ризики компрометації даних. Одним із перспективних методів є поділ файлу на частини та передача кожної частини через окремий канал. Такий підхід знижує ймовірність перехоплення цілісного файлу та підвищує загальний рівень захисту [27].

Основні принципи поділу файлів:

1. Сегментація даних. Поділ файлу на частини здійснюється шляхом сегментації, яка може базуватися на фіксованій розмірності блоків або специфічних властивостях даних. Такий підхід дозволяє оптимізувати процес передачі та мінімізувати ризики компрометації. Наприклад, сегментація може враховувати чутливість окремих частин файлу, передаючи більш критичні сегменти через найбезпечніші канали [28]. Ця сегментація створює додаткові бар'єри для зловмисників, адже вони повинні зібрати всі частини файлу для відновлення його цілісності.

2. Передача через різні канали. Поділ файлу дозволяє використовувати різні канали передачі даних, такі як месенджери, електронна пошта або хмарні сервіси. Наприклад, месенджери, такі як Signal чи Telegram, забезпечують наскрізне шифрування, що значно ускладнює перехоплення даних. Хмарні сервіси дозволяють швидко передати великі обсяги інформації, але можуть бути вразливими до атак на сервери провайдера. Електронна пошта забезпечує зручність, але менш захищена від атак типу «людина посередині». Використання декількох каналів одночасно суттєво ускладнює завдання зловмисника, оскільки він має перехопити всі потоки даних [29]. Різноманітність каналів також сприяє підвищенню надійності передачі, оскільки у разі збою одного з каналів інші можуть продовжити процес

доставки.

3. Відновлення даних. На приймальній стороні відновлення файлу відбувається шляхом об’єднання частин на основі метаданих, які ідентифікують порядок та цілісність переданих сегментів. Метадані можуть включати контрольні суми для перевірки цілісності кожного блоку, що мінімізує ризики втрати або модифікації даних [30]. Цей етап є критично важливим для забезпечення надійності системи, оскільки відсутність навіть однієї частини може зробити неможливим відновлення оригінального файлу.

Переваги використання методу поділу файлів на частини:

1. Підвищена безпека. У разі компрометації одного з каналів зловмисник отримає лише частину даних, яка без інших сегментів залишається непридатною для відновлення початкового файлу. Це значно зменшує ризик несанкціонованого доступу до інформації.

2. Зменшення ризику компрометації. Використання декількох незалежних каналів передачі значно ускладнює виконання атак типу "людина посередині", оскільки зловмиснику необхідно одночасно контролювати кілька потоків даних.

3. Масштабованість та адаптивність. Метод легко адаптується до різних умов передачі: можна використовувати канали з різними рівнями захисту, враховуючи чутливість окремих частин файлу.

4. Підвищена надійність. У разі збою одного з каналів інші забезпечують доставку даних, а відсутній сегмент може бути переданий повторно без втрати цілісності всього файлу.

5. Контроль цілісності. Використання геш-функцій або контрольних сум дозволяє перевірити кожну частину файлу на цілісність після передачі, знижуючи ризики модифікації чи пошкодження даних.

6. Різноманітність каналів. Можливість використання месенджерів, електронної пошти та хмарних сервісів розширює функціональні можливості передачі файлів і дозволяє оптимізувати процес залежно від доступності ресурсів.

7. Захист чутливої інформації. Більш критичні частини файлу можуть передаватися через найбезпечніші канали, тоді як менш чутливі – через швидші, але менш захищені.

2.2 Метод сегментації даних

Пропонується сегментацію даних виконувати з урахуванням специфічних властивостей даних, а саме з урахуванням числових значень байтів. Сегментація еквівалентна процедурі зашифрування даних, яка реалізується шляхом перестановки байтів даних.

Метод сегментації (зашифрування) передбачає виконання таких дій.

1. Ущільнення файлу. Перед зашифруванням файл підлягає ущільненню, наприклад, із використанням алгоритму gzip [34], щоб зменшити його розмір і забезпечити майже однакову частоту появи значень байтів. Це зменшує обсяг даних для передавання і ускладнює аналіз інформації для потенційного зловмисника.

2. Підготовка даних. Ущільнений файл подається у вигляді послідовності байтів:

$$\mathbf{M} = \{m_1, m_2, \dots, m_n\}, \qquad m_i \in [0, 255].$$

Кожен файл у цифровому вигляді представляється набором чисел у діапазоні від 0 до 255.

Секретний ключ **K** є набором із 64 біт (8 байтів):

$$\mathbf{K} = \{k_1, k_2, \dots, k_8\}, \qquad k_i \in [0, 255].$$

Цей ключ використовується для визначення, до якого масиву **F** буде записаний кожен байт із файлу. Перші 8 байтів файлу обробляються особливим чином відповідно до цього ключа.

3. Ініціалізація процесу.

Створюються масиви $\mathbf{F}_0, \mathbf{F}_1, \dots, \mathbf{F}_{255}$. Кожен байт даних буде записуватися до відповідного масиву згідно зі значенням байту секретного ключа (попереднього байту повідомлення). Наприклад, якщо значення байту k_i або m_{i-1} дорівнює 42, то байт m_i додається до масиву $\mathbf{F}_{42}$.

4. Розподіл байтів за ключем.

Розподіл відбувається за таким правилом:

- для перших 8 байтів повідомлення **M** використовуються значення байтів ключа **K**:

$$\mathbf{F}_{k_i} := \mathbf{F}_{k_i} + m_i \text{ для } i = \{1, 2, \dots, 8\};$$

- для всіх наступних байтів (i>8):

$$\mathbf{F}_{m_{(i-1)}} := \mathbf{F}_{m_{(i-1)}} + m_i, \quad i = \{9, 10, \dots, n\}.$$

Цей розподіл дозволяє рівномірно розподілити дані між масивами, що сприяє підвищенню стійкості шифрування.

5. Розподіл по каналах.

Після розподілу байтів масиви $\mathbf{F}_0, \mathbf{F}_1, \dots, \mathbf{F}_{255}$ розподіляються між трьома каналами для передавання даних:

Інформація для першого каналу $\mathbf{I}_1$ утворюється масивами з $\mathbf{F}_0$ до $\mathbf{F}_{84}$:

$$\mathbf{I}_1 = \{L_0, L_1, \dots, L_{84}, \mathbf{F}_0, \mathbf{F}_1, \dots, \mathbf{F}_{84}\},$$

де $L_0, L_1, \dots, L_{84}$– довжина в байтах масивів $\mathbf{F}_0, \mathbf{F}_1, \dots, \mathbf{F}_{84}$, відповідно.

Інформація для другого каналу $\mathbf{I}_2$ утворюється масивами з $\mathbf{F}_{85}$ до $\mathbf{F}_{168}$:

$$\mathbf{I}_2 = \{L_{84}, L_{86}, \dots, L_{168}, \mathbf{F}_{85}, \mathbf{F}_{86}, \dots, \mathbf{F}_{168}\},$$

де $\mathrm{L}_{85}, \mathrm{L}_{86}, \dots, \mathrm{L}_{168}$ – довжина в байтах масивів $\mathbf{F}_{85}, \mathbf{F}_{86}, \dots, \mathbf{F}_{168}$, відповідно.

Інформація для третього каналу $\mathbf{I}_3$ утворюється з масивів $\mathbf{F}_{169}$ до $\mathbf{F}_{255}$:

$$\mathbf{I}_3 = \{\mathrm{L}_{169}, \mathrm{L}_{170}, \dots, \mathrm{L}_{255}, \mathbf{F}_{169}, \mathbf{F}_{170}, \dots, \mathbf{F}_{255}\},$$

де $\mathrm{L}_{169}, \mathrm{L}_{170}, \dots, \mathrm{L}_{255}$ – довжина в байтах масивів $\mathbf{F}_{169}, \mathbf{F}_{170}, \dots, \mathbf{F}_{255}$, відповідно.

Нехай файл має довжину n байт. Оскільки після ущільнення значення усіх байтів зустрічається майже однакову кількість разів, то кожен із файлів буде мати обсяг $n/256$ байт. З урахуванням цього розрядність коду, що вказує на довжину масиву дорівнює $log_2(\frac{n}{256})$.

Приклад 1. Зашифрувати файл, що містить таку послідовність байтів:

$$\mathbf{M} = \{34, 81, 93, 72, 112, 74, 43, 65, 4, 167, 202, 153, 86, 131, 19, 58, 87, 33, 68, 47\}.$$

Нехай ключ шифрування складається з таких байтів:

$$\boldsymbol{K} = \{19, 58, 87, 153, 43, 131, 202, 65\}.$$

Крок 1. Розподіл байтів по масивах $\mathbf{F}_i$.

Перші 8 байтів файлу **M** розподіляємо за ключем **K**:

$k_1 = 19$ означає, що $m_1 = 34$ записуємо в $\mathbf{F}_{19}$:

$$\mathbf{F}_{19} := \mathbf{F}_{19} + m_1 = \{34\}.$$

Аналогічно для $k_2, k_3, \dots, k_8$ маємо:

$k_2 = 58, m_2 = 81 \rightarrow \mathbf{F}_{58} := \mathbf{F}_{58} + m_2 = \{81\}$;

$k_3 = 87, m_3 = 93 \rightarrow \mathbf{F}_{87} := \mathbf{F}_{87} + m_3 = \{93\}$;

$k_4 = 153, m_4 = 72 \rightarrow \mathbf{F}_{153} := \mathbf{F}_{153} + m_4 = \{72\};$

$k_5 = 43, m_5 = 112 \rightarrow \mathbf{F}_{43} := \mathbf{F}_{43} + m_5 = \{112\};$

$k_6 = 131, m_6 = 74 \rightarrow \mathbf{F}_{131} := \mathbf{F}_{131} + m_6 = \{74\};$

$k_7 = 202, m_7 = 43 \rightarrow \mathbf{F}_{202} := \mathbf{F}_{202} + m_7 = \{43\};$

$k_8 = 65, m_8 = 65 \rightarrow \mathbf{F}_{65} := \mathbf{F}_{65} + m_8 = \{65\}.$

Решту байтів файлу розподіляємо правилом:

$$\mathbf{F}_{m_{(i-1)}} := \mathbf{F}_{m_{(i-1)}} + m_i.$$

$m_8 = 65, m_9 = 4 \rightarrow \mathbf{F}_{65} := \mathbf{F}_{65} + m_9 = \{65, 4\};$

$m_9 = 4, m_{10} = 167 \rightarrow \mathbf{F}_4 := \mathbf{F}_4 + m_{10} = \{167\};$

$m_{10} = 167, m_{11} = 202 \rightarrow \mathbf{F}_{167} := \mathbf{F}_{167} + m_{11} = \{202\};$

$m_{11} = 202, m_{12} = 153 \rightarrow \mathbf{F}_{202} := \mathbf{F}_{202} + m_{12} = \{43, 153\};$

$m_{12} = 153, m_{13} = 86 \rightarrow \mathbf{F}_{153} := \mathbf{F}_{153} + m_{13} = \{72, 86\};$

$m_{13} = 86, m_{14} = 131 \rightarrow \mathbf{F}_{86} := \mathbf{F}_{86} + m_{14} = \{131\};$

$m_{14} = 131, m_{15} = 19 \rightarrow \mathbf{F}_{131} := \mathbf{F}_{131} + m_{15} = \{74, 19\};$

$m_{15} = 19, m_{16} = 58 \rightarrow \mathbf{F}_{19} := \mathbf{F}_{19} + m_{16} = \{34, 58\};$

$m_{16} = 58, m_{17} = 87 \rightarrow \mathbf{F}_{58} := \mathbf{F}_{58} + m_{17} = \{81, 87\};$

$m_{17} = 87, m_{18} = 33 \rightarrow \mathbf{F}_{87} := \mathbf{F}_{87} + m_{18} = \{93, 33\};$

$m_{18} = 33, m_{19} = 68 \rightarrow \mathbf{F}_{33} := \mathbf{F}_{33} + m_{19} = \{68\};$

$m_{19} = 68, m_{20} = 47 \rightarrow \mathbf{F}_{68} := \mathbf{F}_{68} + m_{20} = \{47\}.$

Оскільки останній байт m_{20} розподілено, то процес розподілення завершено. Маємо такі масиви:

$\mathbf{F}_4 = \{167\}$, $\mathbf{F}_{19} = \{34, 58\}$, $\mathbf{F}_{33} = \{68\}$, $\mathbf{F}_{43} = \{112\}$, $\mathbf{F}_{58} = \{81, 87\}$, $\mathbf{F}_{65} = \{65, 4\}$, $\mathbf{F}_{68} = \{47\}$, $\mathbf{F}_{86} = \{131\}$, $\mathbf{F}_{87} = \{93, 33\}$, $\mathbf{F}_{131} = \{74, 19\}$, $\mathbf{F}_{153} = \{72, 86\}$, $\mathbf{F}_{167} = \{202\}$, $\mathbf{F}_{202} = \{43, 153\}$.

Крок 2. Розподіл по каналах.

Канал 1 (масиви з $\mathbf{F}_0$ до $\mathbf{F}_{84}$):

$\mathbf{I}_1$ = { L_4, L_{19}, L_{33}, L_{43}, L_{58}, L_{65}, L_{68}, $\mathbf{F}_4$, $\mathbf{F}_{19}$, $\mathbf{F}_{33}$, $\mathbf{F}_{43}$, $\mathbf{F}_{58}$, $\mathbf{F}_{65}$, $\mathbf{F}_{68}$} = {1, 2, 1, 1, 2, 2, 1, 167, 34, 58, 68, 112, 81, 87, 65, 4, 47}.

Канал 2 (масиви з $\mathbf{F}_{85}$ по $\mathbf{F}_{168}$):

$\mathbf{I}_2$ = {$\mathbf{L}_{86}$, $\mathbf{L}_{87}$, $\mathbf{L}_{131}$, $\mathbf{L}_{153}$, $\mathbf{L}_{167}$, $\mathbf{F}_{86}$, $\mathbf{F}_{87}$, $\mathbf{F}_{131}$, $\mathbf{F}_{153}$, $\mathbf{F}_{167}$} = {1, 2, 2, 2, 1, 131, 93, 33, 74, 19, 72, 86, 202}.

Канал 3 (масиви з $\mathbf{F}_{169}$ по $\mathbf{F}_{255}$):

$\mathbf{I}_3$ = {$\mathbf{L}_{202}$, $\mathbf{F}_{202}$} = {2 ,43, 153}.

2.3 Метод відновлення даних

Відновлення даних полягає в об’єднанні сегментів з подальшою процедурою, яка еквівалентна розшифруванню даних, що реалізується шляхом зворотної перестановки байтів.

1. Відновлення масивів $\mathbf{F}_0$, $\mathbf{F}_1$,…,$\mathbf{F}_{255}$.

Інформація $\mathbf{I}_1$ розкладається на складники L_0, L_1,…, L_{84}, $\mathbf{F}_0$, $\mathbf{F}_1$, …, $\mathbf{F}_{84}$. При цьому спочатку відокремлюються коди довжин масивів L_0, L_1,…, L_{84}, а потім відповідно до їх числових значень формуються масиви $\mathbf{F}_0$, $\mathbf{F}_1$, …, $\mathbf{F}_{84}$.

Інформація $\mathbf{I}_2$ розкладається на складники L_{85}, L_{86},…, L_{168}, $\mathbf{F}_{85}$, $\mathbf{F}_{86}$, …, $\mathbf{F}_{168}$. При цьому спочатку відокремлюються коди довжин масивів L_{85}, L_{86},…, L_{168}, а потім відповідно до їх числових значень формуються масиви $\mathbf{F}_{85}$, $\mathbf{F}_{86}$, …, $\mathbf{F}_{168}$.

Інформація $\mathbf{I}_3$ розкладається на складники L_{169}, L_{170},…, L_{255}, $\mathbf{F}_{169}$, $\mathbf{F}_{170}$, …, $\mathbf{F}_{255}$. При цьому спочатку відокремлюються коди довжин масивів L_{169}, L_{170},…, L_{255}, а потім відповідно до їх числових значень формуються масиви $\mathbf{F}_{169}$, $\mathbf{F}_{170}$, …, $\mathbf{F}_{255}$.

2. Відновлення вихідного масиву **M** відбувається за таким правилом.

Перші 8 байтів **M** відновлюються з масивів $\mathbf{F}_{k_i}$ за ключем **K**:

$$\mathbf{M} := \mathbf{M} + \mathbf{F}_{k_i}[0] \text{ для } i = \{1, 2, \dots, 8\}.$$

При цьому нульовий елемент відповідного масиву $\mathbf{F}_{k_i}$ видаляється:

$$\mathbf{F}_{k_i} := \mathbf{F}_{k_i} \setminus \{\mathbf{F}_{k_i}[0]\}.$$

Решта байтів відновлюється з масивів $\mathbf{F}_{m_i}$:

$$\mathbf{M}: = \mathbf{F}_{m_{(i-1)}}[0] \text{ для } i = \{9, 10, \dots, n\}.$$

При цьому нульовий елемент відповідного масиву $\mathbf{F}_{m_i}$ також видаляється:

$$\mathbf{F}_{m_{i-1}}: = \mathbf{F}_{m_{i-1}} \setminus \{\mathbf{F}_{m_{i-1}}[0]\}.$$

3. Розархівування за алгоритмом gzip.

Приклад 2. Розшифрувати інформацію, що отримано з трьох каналів.

Вхідні дані

$\mathbf{I}_1$ = { L_4, L_{19}, L_{33}, L_{43}, L_{58}, L_{65}, L_{68}, $\mathbf{F}_4$, $\mathbf{F}_{19}$, $\mathbf{F}_{33}$, $\mathbf{F}_{43}$, $\mathbf{F}_{58}$, $\mathbf{F}_{65}$, $\mathbf{F}_{68}$} = {1, 2, 1, 1, 2, 2, 1, 167, 34, 58, 68, 112, 81, 87, 65, 4, 47}.

$\mathbf{I}_2$ = {L_{86}, L_{87}, L_{131}, L_{153}, L_{167}, $\mathbf{F}_{86}$, $\mathbf{F}_{87}$, $\mathbf{F}_{131}$, $\mathbf{F}_{153}$, $\mathbf{F}_{167}$} = {1, 2, 2, 2, 1, 131, 93, 33, 74, 19, 72, 86, 202}.

$\mathbf{I}_3$ = {L_{202}, $\mathbf{F}_{202}$} = {2 ,43, 153}.

$\boldsymbol{K} = \{19, 58, 87, 153, 43, 131, 202, 65\}$.

Крок 1. Відновлення масивів **F**.

Канал 1. Відокремлюємо коди довжин масивів:

{L_4, L_{19}, L_{33}, L_{43}, L_{58}, L_{65}, L_{68}} = {1, 2, 1, 1, 2, 2, 1};

L_4 = 1, L_{19} = 2, L_{33} = 1, L_{43} = 1, L_{58} = 2, L_{65} = 2, L_{68} = 1.

Відокремлення числових значень елементів масивів:

$\{\mathbf{F}_4, \mathbf{F}_{19}, \mathbf{F}_{33}, \mathbf{F}_{43}, \mathbf{F}_{58}, \mathbf{F}_{65}, \mathbf{F}_{68}\} = \{167, 34, 58, 68, 112, 81, 87, 65, 4, 47\}.$

$\mathbf{F}_4=\{167\}, \mathbf{F}_{19}=\{34,58\}$ $\mathbf{F}_{33}=\{68\}, \mathbf{F}_{43}=\{112\}$, $\mathbf{F}_{58}=\{81, 87\}$, $\mathbf{F}_{65}=\{65, 4\}$, $\mathbf{F}_{68}=\{47\}$

Канал 2. Також відокремлюємо коди довжин масивів та їх числові значення.

$$\{L_{86}, L_{87}, L_{131}, L_{153}, L_{167}\}=\{1, 2, 2, 2, 1, 1\};$$

$$L_{86} = 1, L_{87} = 2, L_{131} = 2, L_{153} = 2, L_{167} = 1;$$

$$\{\mathbf{F}_{86}, \mathbf{F}_{87}, \mathbf{F}_{131}, \mathbf{F}_{153}, \mathbf{F}_{167}\}=\{131, 93, 33, 74, 19, 72, 86, 202\}.$$

$\mathbf{F}_{86} = \{131\}, \mathbf{F}_{87} = \{93, 33\}, \mathbf{F}_{131} = \{74, 19\}, \mathbf{F}_{153} = \{72, 86\}, \mathbf{F}_{167} = \{202\}.$

Канал 3. Повторюємо дії що і в попередніх каналах.

$$\{L_{202}\} = \{2\};$$

$$\{\mathbf{F}_{202}\} = \{43, 153\}.$$

Крок 2. Об’єднання масивів каналів.

$\mathbf{F}_4 =\{167\}$, $\mathbf{F}_{19} =\{34, 58\}$, $\mathbf{F}_{33} =\{68\}$, $\mathbf{F}_{43} = \{112\}$, $\mathbf{F}_{58} =\{81, 87\}$, $\mathbf{F}_{65} = \{65, 4\}$, $\mathbf{F}_{68} =\{47\}$, $\mathbf{F}_{86} =\{131\}$, $\mathbf{F}_{87}=\{93, 33\}$, $\mathbf{F}_{131} =\{74, 19\}$, $\mathbf{F}_{153} = \{72, 86\}$, $\mathbf{F}_{167} =\{202\}$, $\mathbf{F}_{202}=\{43, 153\}$.

Крок 3. Відновлення вихідного файлу **M**.

Для відновлення використовується секретний ключ:

$$\boldsymbol{K} = \{19, 58, 87, 153, 43, 131, 202, 65\}.$$

Перші 8 байтів файлу **M** розподіляємо за формулою:

$$\mathbf{M} := \mathbf{M} + \mathbf{F}_{k_i}[0].$$

Оскільки $k_1 = 19$, то $m_1 = \mathbf{F}_{19}[0] = 34$. $\mathbf{M} = \{34\}$.

Після вилучення нульового байту масиву $\mathbf{F}_{19}$ маємо:

$\mathbf{F}_{19} := \mathbf{F}_{19} \setminus \{34\} = \{58\}.$

$k_2 = 58 \rightarrow m_2 = \mathbf{F}_{58}[0] = 81$. $\mathbf{F}_{58} := \mathbf{F}_{58} \setminus \{81\} = \{87\}$.

$\mathbf{M} = \{34, 81\}$.

$k_3 = 87 \rightarrow m_3 = \mathbf{F}_{87}[0] = 93$. $\mathbf{F}_{87} := \mathbf{F}_{87} \setminus \{93\} = \{33\}$.

$\mathbf{M} = \{34, 81, 93\}$.

$k_4 = 153 \rightarrow m_4 = \mathbf{F}_{153}[0] = 72$. $\mathbf{F}_{153} := \mathbf{F}_{153} \setminus \{72\} = \{86\}$.

$\mathbf{M} = \{34, 81, 93, 72\}$.

$k_5 = 43 \rightarrow m_5 = \mathbf{F}_{43}[0] = 112$. $\mathbf{F}_{43} := \mathbf{F}_{43} \setminus \{112\} = \{\ \}$.

$\mathbf{M} = \{34, 81, 93, 72, 112\}$.

$k_6 = 131 \rightarrow m_6 = \mathbf{F}_{131}[0] = 74$. $\mathbf{F}_{131} := \mathbf{F}_{131} \setminus \{74\} = \{19\}$.

$\mathbf{M} = \{34, 81, 93, 72, 112, 74\}$.

$k_7 = 202 \rightarrow m_7 = \mathbf{F}_{202}[0] = 43$. $\mathbf{F}_{202} := \mathbf{F}_{202} \setminus \{43\} = \{153\}$.

$\mathbf{M} = \{34, 81, 93, 72, 112, 74, 43\}$.

$k_8 = 65 \rightarrow m_8 = \mathbf{F}_{65}[0] = 74$. $\mathbf{F}_{65} := \mathbf{F}_{65} \setminus \{65\} = \{4\}$.

$\mathbf{M} = \{34, 81, 93, 72, 112, 74, 43, 65\}$.

Решта байтів відновлюється з масивів $\mathbf{F}_{m_i}$ за формулою:

$$\mathbf{M}: = \mathbf{F}_{m_{(i-1)}}[0] \text{ для } i = \{9, 10, \dots, n\}.$$

При цьому нульовий елемент відповідного масиву $\mathbf{F}_{m_i}$ також видаляється:

$$\mathbf{F}_{m_{i-1}}: = \mathbf{F}_{m_{i-1}} \setminus \{\mathbf{F}_{m_{i-1}}[0]\}.$$

$m_8 = 65 \rightarrow m_9 = \mathbf{F}_{65}[0] = 4$. $\mathbf{F}_{65} := \mathbf{F}_{65} \setminus \{4\} = \{\ \}$.

$\mathbf{M} = \{34, 81, 93, 72, 112, 74, 43, 65, 4\}$.

$m_9 = 4 \rightarrow m_{10} = \mathbf{F}_4[0] = 167$. $\mathbf{F}_4 := \mathbf{F}_4 \setminus \{167\} = \{\ \}$.

$\mathbf{M} = \{34, 81, 93, 72, 112, 74, 43, 65, 4, 167\}$.

$m_{10} = 167 \rightarrow m_{11} = \mathbf{F}_{167}[0] = 202$. $\mathbf{F}_{167} := \mathbf{F}_{167} \setminus \{202\} = \{\ \}$.

$\mathbf{M} = \{34, 81, 93, 72, 112, 74, 43, 65, 4, 167, 202\}$.

$m_{11} = 202 \rightarrow m_{12} = \mathbf{F}_{202}[0] = 153.\ \mathbf{F}_{202} := \mathbf{F}_{202} \setminus \{153\} = \{\}.$

M = {34, 81, 93, 72, 112, 74, 43, 65, 4, 167, 202, 153}.

$m_{12} = 153 \rightarrow m_{13} = \mathbf{F}_{153}[0] = 86.\ \mathbf{F}_{153} := \mathbf{F}_{153} \setminus \{86\} = \{\}.$

M = {34, 81, 93, 72, 112, 74, 43, 65, 4, 167, 202, 153, 86}.

$m_{13} = 86 \rightarrow m_{14} = \mathbf{F}_{86}[0] = 131.\ \mathbf{F}_{86} := \mathbf{F}_{86} \setminus \{131\} = \{\}.$

M = {34, 81, 93, 72, 112, 74, 43, 65, 4, 167, 202, 153, 86, 131}.

$m_{14} = 131 \rightarrow m_{15} = \mathbf{F}_{131}[0] = 19.\ \mathbf{F}_{131} := \mathbf{F}_{131} \setminus \{19\} = \{\}.$

M = {34, 81, 93, 72, 112, 74, 43, 65, 4, 167, 202, 153, 86, 131, 19}.

$m_{15} = 19 \rightarrow m_{16} = \mathbf{F}_{19}[0] = 58.\ \mathbf{F}_{19} := \mathbf{F}_{19} \setminus \{58\} = \{\}.$

M = {34, 81, 93, 72, 112, 74, 43, 65, 4, 167, 202, 153, 86, 131, 19, 58}.

$m_{16} = 58 \rightarrow m_{17} = \mathbf{F}_{58}[0] = 87.\ \mathbf{F}_{58} := \mathbf{F}_{58} \setminus \{87\} = \{\}.$

M = {34, 81, 93, 72, 112, 74, 43, 65, 4, 167, 202, 153, 86, 131, 19, 58, 87}.

$m_{17} = 87 \rightarrow m_{18} = \mathbf{F}_{87}[0] = 33.\ \mathbf{F}_{87} := \mathbf{F}_{87} \setminus \{33\} = \{\}.$

M = {34, 81, 93, 72, 112, 74, 43, 65, 4, 167, 202, 153, 86, 131, 19, 58, 87, 33}.

$m_{18} = 33 \rightarrow m_{19} = \mathbf{F}_{33}[0] = 68.\ \mathbf{F}_{33} := \mathbf{F}_{33} \setminus \{68\} = \{\}.$

M = {34, 81, 93, 72, 112, 74, 43, 65, 4, 167, 202, 153, 86, 131, 19, 58, 33, 68}.

$m_{19} = 68 \rightarrow m_{20} = \mathbf{F}_{68}[0] = 47.\ \mathbf{F}_{68} := \mathbf{F}_{68} \setminus \{47\} = \{\}.$

M = {34, 81, 93, 72, 112, 74, 43, 65, 4, 167, 202, 153, 86, 131, 19, 58, 33, 68, 47}.

Оскільки останній байт m_{20} відновлено, то процес відновлення вихідного файлу **M** завершено. Маємо таке його значення:

M = {34, 81, 93, 72, 112, 74, 43, 65, 4, 167, 202, 153, 86, 131, 19, 58, 87, 33, 68, 47}.

Порівняння отриманого масиву з початковим показує, що вони однакові. Це підтверджує коректність методів зашифрування і розшифрування.

2.4 Метод контролю цілісності даних

Контроль цілісності файлів є ключовим аспектом забезпечення безпеки передавання даних у сучасних інформаційних системах. Особливо важливим це

є для розподілених методів передачі, таких як поділ файлів на сегменти. Гешування дозволяє швидко і точно перевірити, чи не було дані змінено під час передачі, зберігання чи обробки [31].

Гешування — це процес перетворення вхідного набору даних (наприклад, тексту, зображень або файлів) у фіксовану довжину строку символів, відому як геш-значення або хеш. Геш-функція забезпечує створення унікального цифрового відбитку, який дозволяє легко перевірити, чи дані залишилися незмінними після їхньої передачі чи збереження [31].

Основними властивостями функцій гешування є:

1. Детермінованість: однакові вхідні дані завжди генерують однаковий геш;

2. Односторонність: неможливо відновити вхідні дані за допомогою геш-значення;

3. Стійкість до колізій: малоймовірно, що дві різні сукупності даних матимуть однаковий геш;

4. Швидкість обчислення: алгоритми гешування мають низькі обчислювальні витрати, що дозволяє використовувати їх навіть на обмежених пристроях [31].

Принцип роботи геш-функцій полягає в тому, що він приймає вхідні дані будь-якої довжини та обчислює їх компактне представлення (геш) фіксованої довжини. В результаті вихідне значення гешу має вигляд, що не дозволяє відновити початкові дані, але слугує унікальним "підписом" цих даних.

Функція гешування може бути записана наступним чином:

$$h = \mathrm{H}(\mathrm{M}),$$

де M — вхідні дані, H — алгоритм гешування, h — результат гешування.

HMAC (Hash-based Message Authentication Code) — це криптографічний механізм, що поєднує геш-функцію та секретний ключ для перевірки цілісності

даних і забезпечення автентичності повідомлення. На відміну від звичайного гешування, яке може гарантувати лише цілісність, HMAC забезпечує також перевірку джерела даних [32].

Переваги HMAC:

1. Безпека ключа. HMAC використовує секретний ключ, який додає значну стійкість до атак на основі геш-функції, наприклад, до атак "день народження".

2. Гнучкість. Працює з різними криптографічними геш-функціями (SHA-1, SHA-256, MD5 тощо), що дозволяє адаптувати його до конкретних вимог безпеки.

3. Стійкість до атак. HMAC захищає дані навіть у разі часткового компрометації геш-функції, оскільки він додає додатковий рівень безпеки завдяки секретному ключу.

4. Цілісність даних. HMAC дозволяє виявляти навіть найменші зміни в даних під час передачі, що критично важливо для забезпечення надійності системи.

5. Сумісність. Легко інтегрується в існуючі протоколи та системи (наприклад, HTTPS, IPsec, SSL/TLS).

6. Простота реалізації. Хоча HMAC складніший, ніж просте гешування, його алгоритм легко реалізується на різних платформах.

AES (Advanced Encryption Standard), а саме версія AES-128, є одним із найпоширеніших стандартів шифрування, що забезпечує високу безпеку даних.

Завдяки високій продуктивності, стійкості до атак та енергоефективності розглянемо AES-128 як основу для побудови HMAC [32].

Обґрунтування вибору AES-128[33]:

1. Висока продуктивність. AES-128 є одним із найшвидших алгоритмів завдяки оптимізованій реалізації для сучасного апаратного забезпечення (наприклад, підтримка AES-NI у процесорах).

2. Стійкість до криптоаналітичних атак. AES-128 має високу стійкість до атак, таких як криптоаналіз на основі диференціального та лінійного аналізу.

3. Захист від колізій. Хоча AES зазвичай використовується для шифрування, його структура дозволяє отримати ефективний геш-функціонал, який захищений від пошуку колізій.

4. Надійність і стандартизація. AES затверджений як стандарт Національним інститутом стандартів і технологій США (NIST) і широко використовується в різних галузях, що підтверджує його надійність.

5. Енергоефективність. AES-128 менш вимогливий до обчислювальних ресурсів порівняно з більш складними версіями (AES-192, AES-256), що робить його ідеальним для використання на різних платформах, зокрема пристроях з обмеженими ресурсами.

Характеристики AES-128 [33]:

1. Довжина ключа: використовує ключ довжиною 128 біт, що забезпечує баланс між високою безпекою та ефективністю.
2. Структура алгоритму: ґрунтується на 10 раундах перетворень, які включають операції підстановки, перемішування, перестановки та додавання ключа.
3. Широка підтримка: AES-128 підтримується практично всіма сучасними протоколами безпеки (TLS, VPN, IPSec тощо).
4. Апаратна прискорюваність: На багатьох процесорах реалізовані спеціальні інструкції для прискорення AES, що значно покращує його швидкодію.

Переваги використання AES-128 у нашій роботі:

- Оптимальний рівень захисту для розв'язуваних завдань.
- Швидке виконання у середовищах із високим навантаженням.
- Можливість комбінування з іншими механізмами захисту, такими як HMAC, для створення комплексних рішень безпеки.

Таким чином, поєднання HMAC із використанням AES-128 як основної геш-функції забезпечує баланс між високим рівнем безпеки, швидкодією та ефективністю. Це робить запропонований метод оптимальним для вирішення завдань безпечної передачі файлів у сучасних умовах.

2.5 Аналіз безпеки та переваги методу

Метод передачі файлів базується на поділі вихідного файлу **M** на сегменти $\mathbf{I}_1, \mathbf{I}_2, \mathbf{I}_3$, обчисленні для кожного з них геш-значення **H** та шифруванні геш-значення за допомогою секретного ключа **K**. Ці дані передаються різними каналами зв'язку. Такий підхід забезпечує новий рівень безпеки порівняно з традиційними методами, унеможливлюючи доступ зловмисника до всієї інформації навіть за умови компрометації одного з каналів.

Для кожного сегмента **I** обчислюється його геш-значення **H** за допомогою шифру AES-128, який працює у режимі зчеплення блоків, з використанням секретного ключа **K**. Останній зашифрований блок і є геш-значенням **H**.

Інформація, яка передається каналами:

Канал 1 (Telegram).

$\mathbf{I}_1 \parallel \mathbf{H}_1 \parallel \mathbf{K}$

Протокол MTProto забезпечує наскрізне шифрування даних.

Канал 2 (Signal).

$\mathbf{I}_2 \parallel \mathbf{H}_2 \parallel \mathbf{K}$

Протокол Double Ratchet гарантує forward secrecy та backward secrecy.

Канал 3 (Gmail):

$\mathbf{I}_3 \parallel \mathbf{H}_3 \parallel \mathbf{K}$

Gmail використовує TLS, який захищає дані від перехоплення. Розподілена передача сегментів через різні канали ускладнює перехоплення всієї інформації зловмисником, оскільки кожен із каналів використовує наскрізне шифрування.

Кожен із каналів забезпечує певний рівень захисту завдяки використанню сучасних криптографічних протоколів.

MTProto, використовуваний у Telegram, створено для забезпечення наскрізного шифрування між клієнтами.

MTProto використовує симетричне шифрування на основі AES-256 із використанням режиму шифрування IGE (Infinite Garble Extension), що забезпечує стійкість до атак на цілісність даних.

Секретні ключі генеруються окремо для кожної комунікації, зменшуючи ризик компрометації ключів.

Стійкість до атак: Завдяки наскрізному шифруванню Telegram захищає повідомлення як під час передачі, так і під час зберігання на серверах. Крім того, сервери Telegram не зберігають ключі сесії, що унеможливлює доступ до даних навіть адміністраторам сервісу.

Обмеження: Telegram може бути вразливим до атак MITM у разі компрометації клієнтського додатка, але це ризик мінімізується завдяки постійному оновленню клієнтського програмного забезпечення.

2. Signal використовує протокол Double Ratchet для забезпечення максимальної конфіденційності повідомлень.

Основні принципи роботи:

1. Динамічне оновлення ключів після кожного повідомлення. Це забезпечує forward secrecy (секретність майбутніх повідомлень) та backward secrecy (секретність попередніх повідомлень).

2. Кожне повідомлення шифрується за допомогою унікального ключа, що значно ускладнює аналіз трафіку для зловмисників.

Стійкість до атак: запобігає атакам MITM завдяки протоколу перевірки автентичності ключів на основі QR-кодів, що дозволяє перевіряти автентичність з'єднання між сторонами.

Захист метаданих: обмежує доступ до метаданих, приховуючи імена відправників та отримувачів. Це значно підвищує рівень конфіденційності комунікацій навіть у разі перехоплення трафіку.

Gmail забезпечує захищену передачу файлів завдяки використанню протоколу TLS (Transport Layer Security).

TLS використовує:

1. RSA для початкового обміну ключами;

2. AES для симетричного шифрування переданих даних після встановлення захищеного з'єднання.

Стійкість до атак: TLS захищає від атак MITM завдяки автентифікації серверів через сертифікати SSL/TLS, що підтверджують легітимність серверів.

Обмеження: Gmail не забезпечує наскрізне шифрування на рівні користувачів, оскільки дані можуть бути доступні адміністраторам Google під час зберігання на серверах. Однак передача через TLS залишається захищеною.

2.6 Висновки до розділу 2

Розробка методів захищеного передавання файлів у другому розділі дозволила сформувати теоретичну базу для побудови ефективного алгоритму сегментації та розподілу даних. Аналіз існуючих загроз, таких як перехоплення, модифікація та аналіз файлів, допоміг визначити ключові аспекти, які повинні бути враховані під час розробки системи. Алгоритм сегментації, що передбачає поділ файлу на частини із розподілом по масивах залежно від значення ключа та попередніх байтів, забезпечує рівномірність і стійкість до криптоаналізу. Крім того, процес розподілу частин файлу через кілька каналів додає значного рівня складності для потенційного зловмисника. Виконане моделювання підтвердило, що вибрані підходи є доцільними та здатними забезпечити високий рівень безпеки під час передачі даних. Таким чином, теоретичні положення другого розділу є основою для практичної реалізації системи, представленої у третьому розділі.

3 РОЗРОБКА ПРОГРАМНОГО ЗАСОБУ ДЛЯ ЗАХИЩЕНОГО ПЕРЕДАВАННЯ ДАНИХ ТА ТЕСТУВАННЯ

3.1 Алгоритм сегментації файлу

Оскільки процес зашифрування даних є критично важливим етапом у забезпеченні безпеки передачі інформації, важливо чітко формалізувати його логіку. Побудова схеми алгоритму дозволяє візуалізувати всі кроки, забезпечуючи легкість розуміння, точність реалізації та можливість ідентифікації потенційних слабких місць у процесі. Такий підхід також сприяє стандартизації процесу, дозволяючи відтворити його у будь-якому середовищі, незалежно від використаних інструментів чи технологій.

Крім того, створення схеми надає можливість інтегрувати метод зашифрування до складних систем із мінімальними адоптаціями. Вона дозволяє чітко визначити послідовність операцій, залежності між ними та ключові точки контролю, такі як перевірка цілісності або відповідність отриманих даних до вихідних. У випадках, коли виникає необхідність вносити зміни чи оптимізувати процес, схема надає необхідний рівень деталізації, спрощуючи роботу інженерів та розробників.

Таким чином, алгоритм зашифрування даних у вигляді блок-схеми не лише забезпечує зручність для реалізації, а й виступає ключовим інструментом для подальшого аналізу та вдосконалення системи. Блок-схема узагальненого алгоритму сегментації файлу наведена на рис. 3.1.

Процес сегментації починається із завантаження вхідного файлу, який підлягає подальшій обробці. Перший етап передбачає ущільнення файлу, що виконується за допомогою алгоритмів компресії, таких як gzip. Цей етап дозволяє зменшити обсяг даних та забезпечити рівномірний розподіл значень байтів, що ускладнює їх аналіз зловмисниками.

Наступним етапом є генерація секретного ключа, який визначає логіку розподілу байтів між масивами. Цей ключ є унікальним для кожного процесу і слугує основою для подальшої обробки даних.

Рисунок 3.1 – Узагальнений алгоритм сегментації файлу

Після цього здійснюється ініціалізація масивів з $\mathbf{F}_0$ по $\mathbf{F}_{255}$, кожен з яких представляє можливе значення байтів. Ці масиви є базовими контейнерами для зберігання розподілених частин файлу.

Далі виконується розподіл байтів файлу між масивами відповідно до формул наведених в теоретичній частині.

На останньому етапі дані формуються у вигляді трьох потоків, кожен із яких відповідає певному діапазону масивів. Ці потоки передаються через окремі канали зв’язку. Такий підхід дозволяє досягти підвищеного рівня безпеки, оскільки навіть у разі перехоплення одного потоку зловмисник не зможе отримати повну інформацію, а також кожен з каналів використовує додатково наскрізне шифрування при передаванні.

Ця процедура забезпечує захищене передавання файлу, гарантуючи, що жоден канал не містить достатньо даних для відновлення початкової інформації без доступу до всіх потоків і секретного ключа.

Надійна генерація секретного ключа є одним із найважливіших етапів у забезпеченні захищеності інформаційних систем. У криптографії секретний ключ забезпечує основний рівень захисту даних, а тому його стійкість і унікальність мають бути гарантовані. Генерація ключів відповідно до сучасних стандартів кібербезпеки унеможливлює несанкціонований доступ до даних та мінімізує ризики злому. Недостатньо надійний ключ може стати слабким місцем будь-якого алгоритму безпеки. Тому критично важливо забезпечити дотримання визнаних міжнародних стандартів, таких як NIST SP 800-90A, ISO/IEC 18031, і OWASP Cryptographic Storage Cheat Sheet [34].

Алгоритм генерації секретного ключа наведено на рис 3.2.

Генерація починається із ініціалізації бібліотеки secrets, яка відповідає криптографічним стандартам безпеки. Далі визначається довжина ключа — 64 біти (8 байтів). Це оптимальний розмір для забезпечення балансу між стійкістю до атак і швидкістю обчислень. Після цього виконується генерація ключа методом token_bytes(8), який створює випадковий набір із 8 байтів.

Згенерований ключ представляється у форматі $\{k_1, k_2, ..., k_n\}$, де кожен байт поданий як число у діапазоні [0, 255]. Потім ключ зберігається для подальшого використання в процесі сегментації файлу, після якої він буде поділений на три частини та переданий через окремі канали зв’язку.

Збереження ключа у захищеному середовищі є фінальним етапом цього алгоритму.

Рисунок 3.2 – Алгоритм генерації секретного ключа

Код, що реалізує даний алгоритм:

```
import secrets
# Генерація секретного ключа K розміром 64 біти (8 байтів)
K = secrets.token_bytes(8)
# Вивід ключа у форматі {k1, k2, ..., kn}, де k_i - числове значення байта
K_list = [int(byte) for byte in K]
formatted_key = "K = {" + ", ".join(map(str, K_list)) + "}."
print("Секретний ключ у форматі:")
print(formatted_key)
```

Результат реалізації наведеного програмного коду подано на рис. 3.3.

```
Секретний ключ у форматі:
K = {188, 227, 222, 65, 179, 185, 64, 71}.


** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

Рисунок 3.3 – Результат генерації секретного ключа

Алгоритм розподілу байтів файлу між масивами є ключовим етапом у процесі підготовки даних для подальшого безпечного передавання. Він забезпечує рівномірний розподіл даних між масивами, що підвищує стійкість системи до аналізу та можливих атак. Алгоритм ініціалізації масивів та розподілу байтів файлу наведено на рис 3.4.

Алгоритм починається з ініціалізації 256 порожніх масивів $\mathbf{F}_0$, $\mathbf{F}_1$,…,$\mathbf{F}_{255}$, які будуть використовуватись для зберігання байтів файлу. Наступним кроком є введення вихідних даних і секретного ключа, які визначають правила розподілу. Далі послідовно зчитується кожен байт m_i з файлу.

Для перших 8 байтів $i \leq 8$, розподіл здійснюється за значенням ключа k_i: байт m_i додається до масиву $\mathbf{F}_{k_i}$. Для решти байтів (i > 8), розподіл виконується на основі значення попереднього байта m_{i-1} : байт m_i додається до масиву $\mathbf{F}_{m_{i-1}}$.

Процес повторюється, доки всі байти файлу не будуть розподілені. На завершальному етапі перевіряється, чи оброблено всі байти, і алгоритм завершується. Такий підхід дозволяє забезпечити детермінований і криптографічно стійкий розподіл даних, який є основою для подальшого формування сегментів.

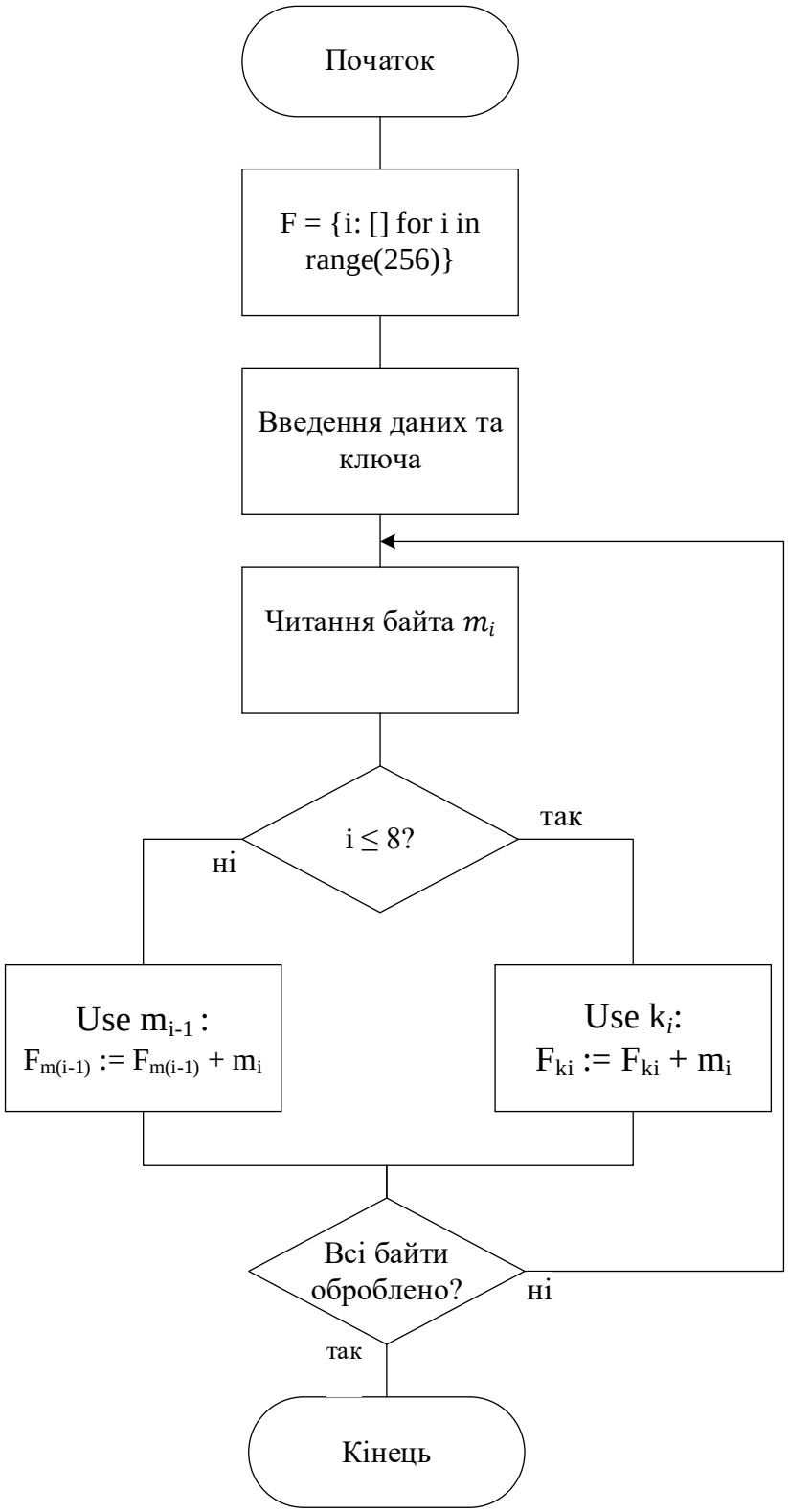


Рисунок 3.4 – Алгоритм ініціалізації масивів та розподілу байтів файлу

У програмі реалізація цього алгоритму відбувається функцією segment_file. Ось відповідна частина коду:

```
def segment_file(data, key):
    """Сегментація та шифрування даних на основі ключа."""
    # Ініціалізація масивів F_0, F_1, ..., F_255
    F = {i: [] for i in range(256)}
    # Розподіл байтів файлу між масивами
    for i, byte in enumerate(data):
        if i < 8:  # Перші 8 байтів розподіляються за ключем K
            F[key[i % len(key)]].append(byte)
        else:  # Решта байтів розподіляються за попереднім байтом
m_(i-1)
            F[data[i - 1]].append(byte)
    return F
```

Цей код безпосередньо реалізує алгоритм (див. рис. 3.4) та забезпечує розподіл байтів між масивами відповідно до правил.

3.2 Алгоритм відновлення файлу

Алгоритм відновлення файлу є ключовим етапом для вирішення поставленого завдання. Він гарантує збереження конфіденційності та цілісності інформації навіть у разі потенційних спроб перехоплення. Узагальнений алгоритм відновлення файлу наведено на рис. 3.5.

Процес відновлення файлу починається з отримання даних, переданих через три окремі канали. На цьому етапі всі сегменти інформації, що були розподілені між каналами під час процесу сегментації, збираються для подальшої обробки. Алгоритм отримання даних із трьох каналів наведено на рис. 3.6 Код отримання даних з сегментів реалізується функцією read_channel_file():

```
def read_channel_file():
    """Вибір та читання файлу сегмента."""
    root = Tk()
    root.withdraw()  # Приховує головне вікно
    root.attributes('-topmost', True)  # Робить вікно діалогу
поверх усіх інших
    channel_file = filedialog.askopenfilename(title="Оберіть файл
сегмента")
  if not channel_file:
   log("Файл не обрано.")  # Лог повідомлення, якщо файл не обрано
      return None
  with open(channel_file, 'rb') as f:
```

```
      return f.read()  # Зчитування даних файлу
# Збирання даних із трьох каналів
channels_data = []
for i in range(3):
    print(f"Виберіть сегмент {i+1}:")
    data = read_channel_file()
    if data is None:
        print(f"Дані з каналу {i+1} не вибрано.")  # Повідомлення
про відсутність даних
        return
    channels_data.append(data)  # Додаємо сегмент до списку
```

Після цього виконується відновлення сегментів, отриманих із кожного каналу. Це передбачає аналіз метаданих, зокрема кодів довжини масивів, які дозволяють визначити, як саме були розподілені байти у кожному сегменті. Цей етап є важливим для забезпечення цілісності даних і формування базових масивів.

Код для роботи із сегментами:

```
# Обробка даних кожного каналу
for idx, channel_data in enumerate(channels_data):
    if idx == 0:  # Для першого каналу
        lengths = channel_data[:85]  # Довжини масивів F_0 до F_84
        data = channel_data[85:]  # Дані масивів
    elif idx == 1:  # Для другого каналу
        lengths = channel_data[:84]  # Довжини масивів F_85 до F_168
        data = channel_data[84:]  # Дані масивів
    else:  # Для третього каналу
        lengths = channel_data[:87]  # Довжини масивів F_169 до F_255
        data = channel_data[87:]  # Дані масивів
    # Відновлення масивів із даних
   for i, length in enumerate(lengths):
       if length > 0:  # Якщо масив не порожній
           segment = data[:length]  # Зчитуємо сегмент потрібної довжини
            F[85 * idx + i if idx < 2 else 169 + i] = list(segment)
           # Відновлюємо масив F
            data = data[length:]  # Видаляємо оброблені байти
```

Далі відбувається відновлення масивів з $\mathbf{F}_0$ по $\mathbf{F}_{255}$. Використовуючи метадані та значення байтів, кожен масив реконструюється у тому вигляді, в якому він був створений на етапі сегментації. Ця процедура забезпечує точну відповідність між початковими та отриманими масивами.

Рисунок 3.5 – Узагальнений алгоритм відновлення файлу

Код для об’єднання масивів:

```
# Ініціалізація порожніх масивів
F = {i: [] for i in range(256)}  # Створення 256 масивів F_0 до F_255
# Відновлення вихідного повідомлення
reconstructed_data = []
for i, k in enumerate(key[:8]):
    if k in F and F[k]:  # Перевіряємо, чи є дані у відповідному масиві
        reconstructed_data.append(F[k][0])    # Додаємо байт до вихідного повідомлення
        F[k] = F[k][1:]  # Видаляємо оброблений байт із масиву
```

```
# Обробка решти байтів
while True:
    prev_byte = reconstructed_data[-1]  # Останній доданий байт
    if prev_byte in F and F[prev_byte]:  # Перевіряємо наявність даних у масиві
        reconstructed_data.append(F[prev_byte][0])  # Додаємо байт до вихідного повідомлення
        F[prev_byte] = F[prev_byte][1:]  # Видаляємо оброблений байт
    else:
        break  # Завершуємо, якщо всі байти оброблено
```

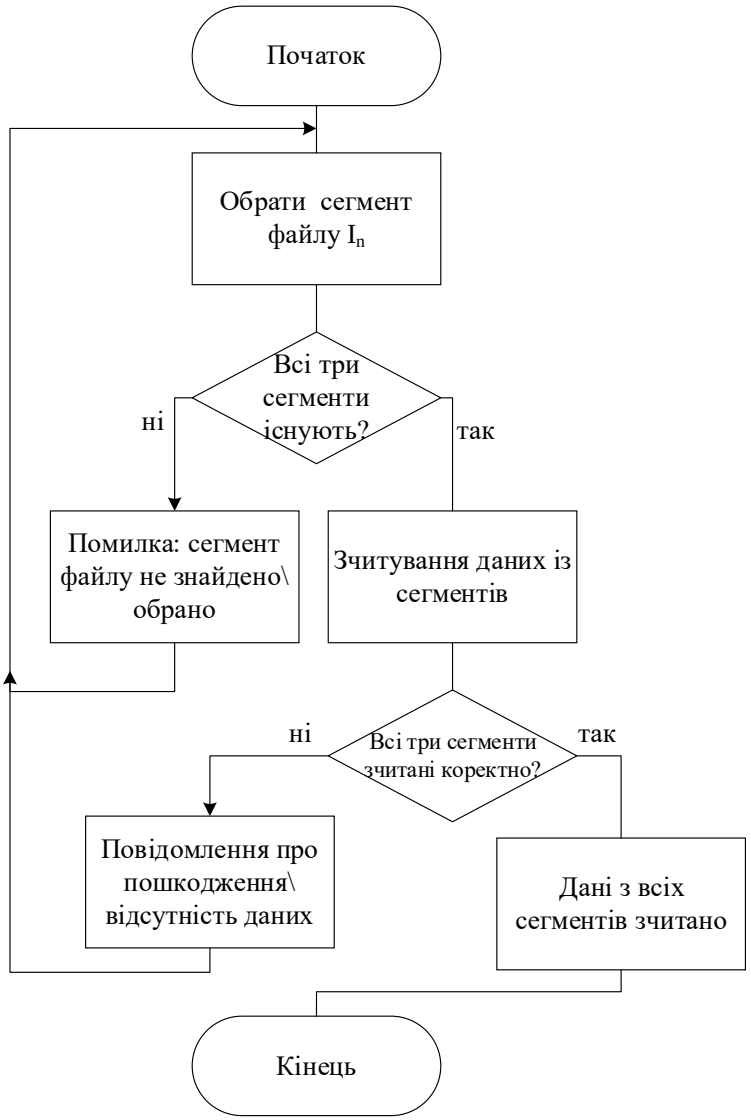


Рисунок 3.6 – Алгоритм отримання даних із трьох каналів

На наступному етапі формується ущільнений файл. Об’єднуючи всі відновлені масиви у правильній послідовності, вдається відтворити ущільнений файл, який є попередником початкового файлу. Цей етап завершує підготовку даних до остаточного відновлення.

```
# Формування відновлених даних та збереження їх у файл
decompressed_data = decompress_data(bytes(reconstructed_data))  # Декомпресія зібраних байтів
with open(save_path, 'wb') as f:
    f.write(decompressed_data)  # Збереження декомпресованих даних у файл
```

Фінальний етап – відновлення початкового файлу. На цьому етапі виконується відновлення файлу з використанням gzip, яка еквівалентна алгоритму ущільнення, що застосовувався під час сегментації. У результаті відновлюється оригінальний файл у його початковій формі, готовий для використання. Алгоритм формування ущільненого та відновленого файлу наведено на рис. 3.7.

```
try:
    decompressed_data = decompress_data(reconstructed_data)  # Спроба декомпресії
    with open(save_path, 'wb') as f:
        f.write(decompressed_data)  # Збереження початкового файлу
    print(f"Відновлений файл збережено у: {save_path}")  # Повідомлення про успіх
except gzip.BadGzipFile:
    print("Помилка: Відновлений файл не є gzip-архівом.")  # Повідомлення про помилку декомпресії
```

Кожен блок коду наведеному в пункті розділу відповідає своєму етапу роботи алгоритму. Безпосередньо використання алгоритмів спрощують розуміння логіки роботи програми та подальшого її написання.

Таким чином використавши загальні алгоритми сегментації та відновлення файлу, а також детальний огляд основних блоків стає можлива програмна реалізація.

3.3 Алгоритми формування та перевірки HMAC

Перевірка HMAC (Hash-based Message Authentication Code) є критично важливим етапом для забезпечення цілісності даних у системі. У запропонованій програмній реалізації HMAC генерується для кожного

сегменту файлу за допомогою симетричного ключа. Цей процес гарантує, що дані не були змінені під час передачі.

Рисунок 3.7 – Алгоритм формування ущільненого та відновленого файлу

HMAC генерується для кожного сегменту даних з використанням алгоритму AES у режимі ECB (Electronic Codebook). При цьому ключ, що використовується для HMAC, має довжину 64 біти, тоді як алгоритм AES працює з блоками розміром 128 біт. Для адаптації ключа до потрібного розміру застосовується дзеркальне дублювання ключа: до початкового ключа додається його перевернута копія. Алгоритм формування HMAC наведено на рис. 3.8.

Рисунок 3.8 – Алгоритм формування HMAC

Ось код, який реалізує формування HMAC:

```
from Crypto.Cipher import AES
def pad_data(data, block_size):
    padding_length = block_size - (len(data) % block_size)
    return data + bytes([padding_length] * padding_length)
def generate_hmac_aes(data, key):
    """Генерація HMAC для даних за допомогою AES-128."""
    padded_data = pad_data(data, AES.block_size)    # Доповнення
даних до розміру блоку AES
```

```
    aes = AES.new((key + key[::-1])[:16], AES.MODE_ECB)     # 
Дзеркальне дублювання ключа до 128 біт (16 байт)
    hashed = aes.encrypt(padded_data)[:16]   # Шифрування даних і 
відсічення перших 16 байт
    return hashed.hex()
```

Особливості при формуванні HMAC:

- Дзеркальне дублювання ключа: Ключ довжиною 64 біти (8 байтів) доповнюється перевернутою копією самого себе.

- Доповнення даних: Якщо довжина даних не кратна розміру блоку AES (16 байтів), дані доповнюються байтами, значення яких відповідає кількості байтів, необхідних для заповнення блоку. Це забезпечує правильну обробку даних у режимі ECB.

- Застосування режиму CBC [35]: Режим CBC дозволяє застосувати AES для шифрування даних, формуючи криптографічний підпис (HMAC).

Перевірка HMAC здійснюється під час декодування файлу. Спочатку програма зчитує оригінальний HMAC із файлу сегмента, а потім обчислює новий HMAC для відповідних даних. Якщо обчислений HMAC співпадає з оригінальним, то дані вважаються цілісними. Алгоритм перевірки HMAC наведено на рис. 3.9.

Код, що реалізує перевірку HMAC:

```
def verify_hmac():
    """Перевірка HMAC для сегментів."""
    print("Оберіть каталог, де збережено файли сегментів та 
HMAC:")
    root = Tk()
    root.withdraw()
    root.attributes('-topmost', True)
    directory = filedialog.askdirectory(title="Оберіть каталог із 
сегментами")

    if not directory:
        print("Каталог не обрано.")
        return

    key_file = os.path.join(directory, 'key.txt')
    if not os.path.exists(key_file):
        print("Файл ключа не знайдено.")
        return
```

```
    with open(key_file, 'r') as f:
        key_line = f.readlines()[1].strip()
        key = bytes(map(int, key_line.split(', ')))

    for i in range(1, 4):
        channel_file = os.path.join(directory, f'channel_{i}.bin')
        hmac_file = os.path.join(directory,
f'channel_{i}_hmac.txt')

        if not os.path.exists(channel_file) or not
os.path.exists(hmac_file):
            print(f"Файли для каналу {i} не знайдено.")
            continue

        with open(channel_file, 'rb') as f:
            channel_data = f.read()

        with open(hmac_file, 'r') as f:
            stored_hmac = f.read().strip()

        computed_hmac = generate_hmac_aes(channel_data, key)

        if computed_hmac == stored_hmac:
            print(f"HMAC для каналу {i} вірний.")
        else:
            print(f"HMAC для каналу {i} НЕВІРНИЙ!")
```

Особливості перевірки

1. Зчитування оригінального HMAC: Зчитується значення HMAC, збережене під час формування сегменту, із відповідного текстового файлу.

2. Обчислення нового HMAC: Застосовується функція generate_hmac_aes для обчислення нового HMAC на основі зчитаних даних і ключа.

3. Порівняння HMAC: Якщо обчислений і зчитаний HMAC співпадають, це підтверджує цілісність даних.

Механізм перевірки HMAC є надійним інструментом для виявлення змін даних і гарантує, що жодні сторонні зміни не були внесені під час передачі.

3.4 Алгоритм роботи з логами

Логи – це повідомлення про виконання основних операцій програми, які допомагають відстежувати її поведінку та відлагоджувати процеси. У

представленій програмі передбачено механізм ввімкнення та вимкнення логів через меню, що дозволяє користувачеві контролювати обсяг виведеної інформації.

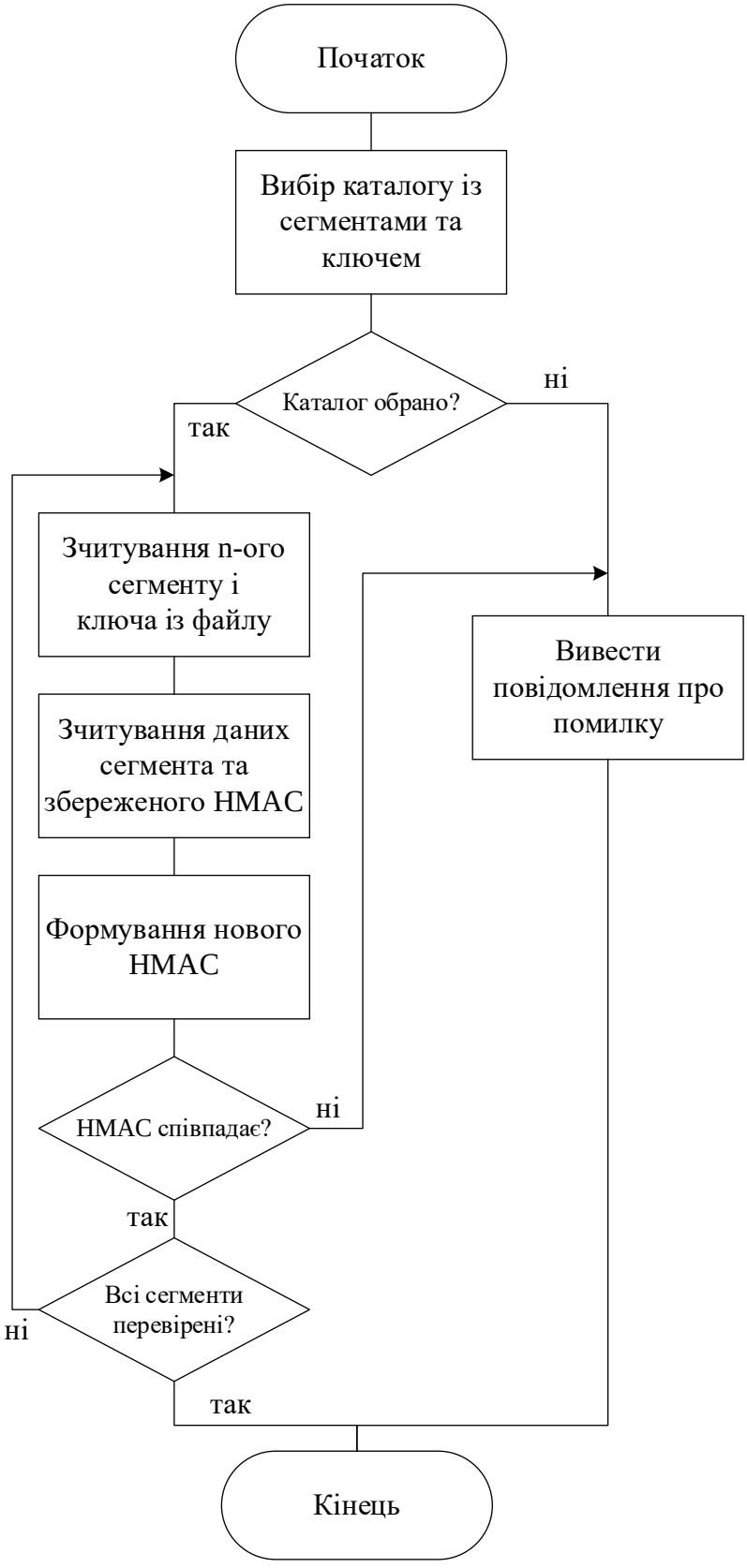


Рисунок 3.9 – Алгоритм перевірки HMAC

Основні можливості логів:

- Інформування про хід виконання операцій, таких як сегментація файлу, перевірка HMAC, відновлення даних.

- Вимкнення зайвого виводу для зручності роботи, залишаючи лише основні повідомлення.

Код, що реалізує механізм логів:

Глобальна змінна для контролю логів:

```
LOG_ENABLED = False  # Контролює стан логів
```

Функція логування:

```
def log(message):
    if LOG_ENABLED:
        print(message)
```

Перемикання логів у меню:

```
elif choice == '5':  # Encryption Program
    LOG_ENABLED = not LOG_ENABLED
    print("Логи увімкнено" if LOG_ENABLED else "Логи вимкнено")
```

Механізм логів забезпечує гнучкість використання програми, дозволяючи користувачам активувати докладний вивід або працювати з мінімальними повідомленнями, залежно від потреб.

3.5 Тестування засобу для захищеного передавання даних

Тестування програмного засобу проводилося для перевірки його працездатності, коректності виконання основних функцій та відповідності поставленим вимогам. Основними завданнями тестування були перевірка функцій сегментації (шифрування) файлу, відновлення (розшифрування) та порівняння відновлених даних із початковими для підтвердження їх цілісності. А також забезпечення перевірки цілісності та автентичності за допомогою HMAC.

Процес тестування включав наступні етапи:

1. Перевірка функції сегментації (шифрування): Вхідний файл поділяється на три сегменти за допомогою ключа. Перевіряється правильність поділу шляхом порівняння кількості отриманих сегментів із очікуваним значенням.

2. Перевірка функції відновлення (розшифрування): Виконується об'єднання сегментів для відновлення початкового файлу. Результат порівнюється з вихідним файлом для підтвердження ідентичності.

3. Перевірка роботи перемикача логів: Чи коректно вмикається відображення логів в терміналі, де детально розписаний процес сегментації та відновлення, але уповільнює роботу програми.

4. Перевірка можливості переглянути вміст сегменту після процесу зашифрування: Чи відповідає вміст сегмента після зашифрування даним що відомі з терміналу після процедури сегментації.

Тестування виконувалося на файлі test_file.txt розміром 6,91 КБ, що містив текстові дані, та на основі симетричного ключа розміром 8 байтів. Шлях до відновленого файлу був вказаний як reconstructed_file.txt.

Вхідні дані:

- Оригінальний файл: test_file.txt
- Відновлений файл: reconstructed_file.txt
- Симетричний ключ: [19, 58, 87, 153, 43, 131, 202, 65]

Нижче представлений код, який реалізує процес тестування сегментації (шифрування) та відновлення (розшифрування):

```
import os
from encryption_program import segment_file, generate_hmac_aes
from file_decryption import reconstruct_file

# Функція для порівняння файлів
def compare_files(original_path, reconstructed_path):
    with open(original_path, 'rb') as original,
open(reconstructed_path, 'rb') as reconstructed:
        return original.read() == reconstructed.read()

# Тестування сегментації (шифрування)
def test_segmentation_encryption(file_path, key):
```

```
    print("Тест сегментації (шифрування)")
    with open(file_path, 'rb') as f:
        data = f.read()
    segments = segment_file(data, key)
    assert len(segments) == 3, "Кількість сегментів не відповідає
очікуваній"
    print("Успіх: файл успішно зашифровано та сегментовано.")

# Тестування відновлення (розшифрування)
def test_reconstruction_decryption(original_path, key,
reconstructed_path):
    print("Тест відновлення (розшифрування)")
    with open(original_path, 'rb') as f:
        data = f.read()
    segments = segment_file(data, key)  # Сегментація як
шифрування
    reconstructed_data = reconstruct_file(segments, key)  #
Відновлення файлу
    with open(reconstructed_path, 'wb') as f:
        f.write(reconstructed_data)
    assert compare_files(original_path, reconstructed_path),
"Помилка: відновлений файл не відповідає оригіналу"
    print("Успіх: файл успішно відновлено.")
# Основний блок тестування
if __name__ == "__main__":
    test_file = "test_file.txt"  # Шлях до тестового файлу
    reconstructed_file = "reconstructed_file.txt"  # Шлях до
відновленого файлу
    test_key = [19, 58, 87, 153, 43, 131, 202, 65]  # Приклад
ключа
    # Запуск тестів
    test_segmentation_encryption(test_file, test_key)
    test_reconstruction_decryption(test_file, test_key,
reconstructed_file)
    print("Всі тести пройдено успішно!")
```

Результати тестування з використанням цього коду відображено на рис. 3.10 - 3.12.

```
PS C:\Users\gyrzh\OneDrive\Робочий стіл\file_encryption_and_decryption> python test_program.py
>> python test_program.py
Тест сегментації (шифрування)
Успіх: файл успішно зашифровано та сегментовано.
Тест відновлення (розшифрування)
Успіх: файл успішно відновлено.
Всі тести пройдено успішно!
```

Рисунок 3.10 – Вікно терміналу після виконання тестування

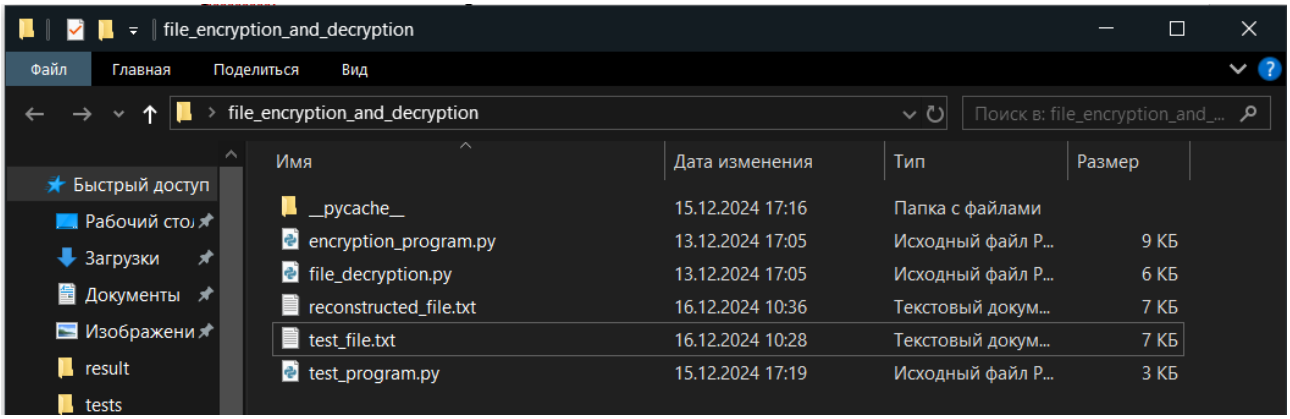


Рисунок 3.11 – Каталог з файлом оригіналом та відновленим файлом

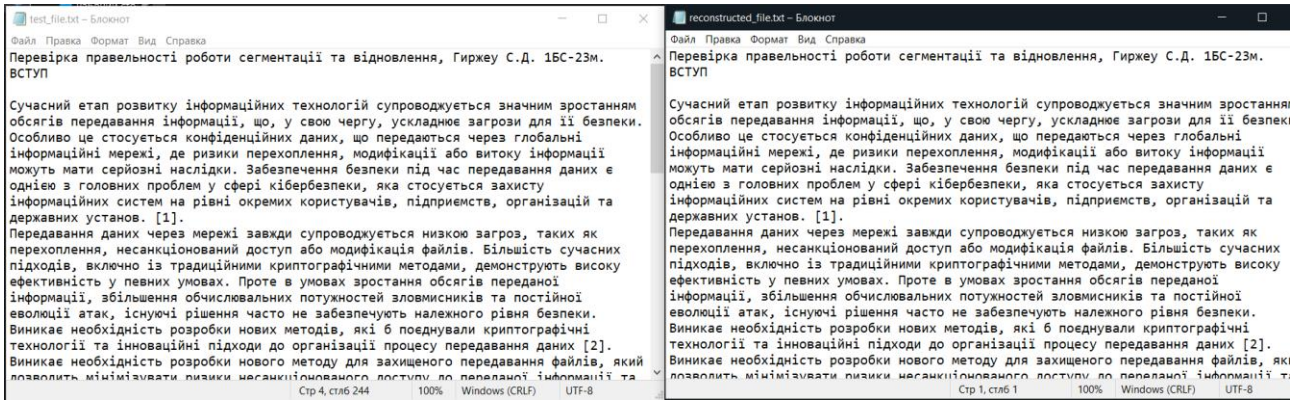


Рисунок 3.12 – Вміст оригіналу та відновленого файлу

Тестування підтвердило працездатність основних механізмів:

- Сегментація (шифрування) коректно поділяє файл на три частини відповідно до заданого алгоритму.

- Відновлення (розшифрування) успішно об'єднує сегменти у початковий файл.

- Порівняння відновленого файлу з оригінальним підтвердило їх повну ідентичність.

Результати покрокового виконання програми. Після запуску програми в терміналі з’являється головне меню(рис 3.13).

Створюється файл test1.txt, який використовується для тестування (рис 3.14).

```
PS C:\Users\gyrzh\OneDrive\Робочий стіл\file_encryption_and_decryption> python encryption_program.py
Оберіть дію:
1. Зашифрувати
2. Розшифрувати
3. Перевірити HMAC
4. Переглянути вміст сегменту
5. Увімкнути/вимкнути логи
6. Вихід
Введіть ваш вибір:
```

Рисунок 3.13 – Головне меню програми для захищеного передавання файлів

Після вибору пункту меню «1. Зашифрування». Відобразилось кілька спливаючих вікон, щоб обрати місце де знаходиться файл для сегментації, а також каталог де буде він збережений вже в вигляді сегментів, hmac та секретного ключа(рис. 3.15 – 3.16).

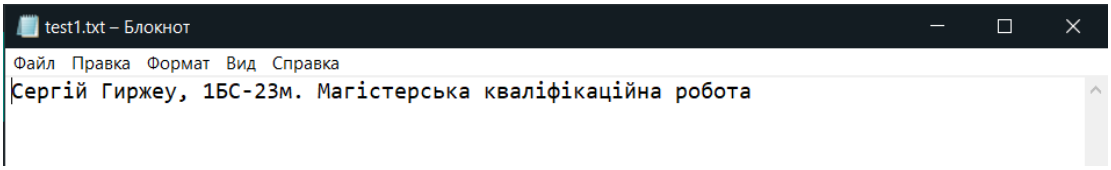


Рисунок 3.14 – Вміст файлу test1.txt

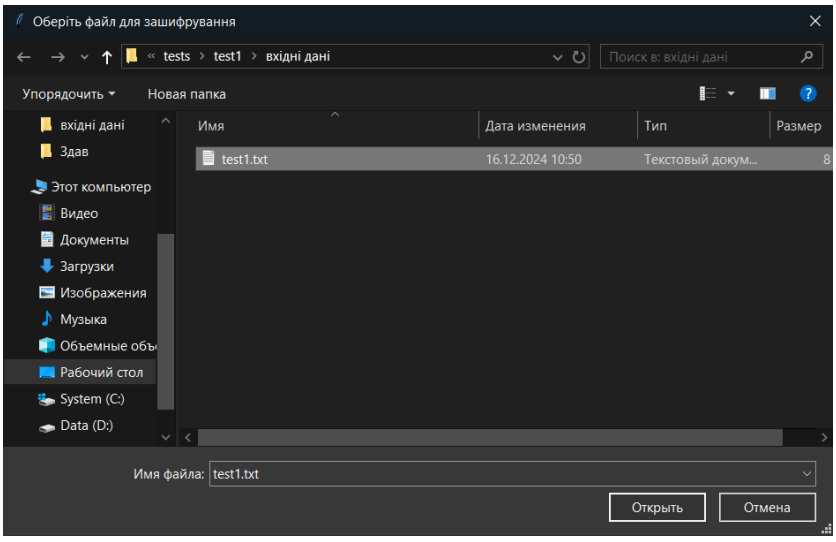


Рисунок 3.15 – Вікно вибору файлу для шифрування

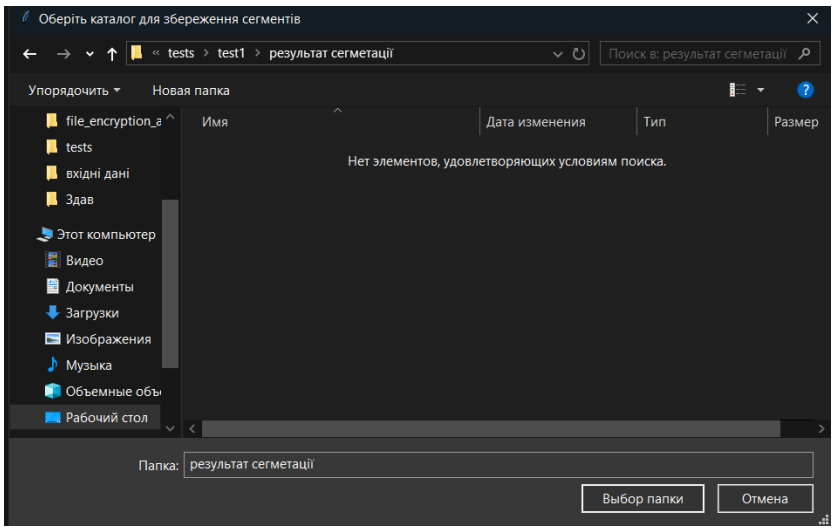


Рисунок 3.16 – Вікно вибору каталогу для збереження сегментів, hmac та секретного ключа

В результаті отримано три сегменти, hmac для кожного із них та файл з секретним ключем (рис. 3.17).

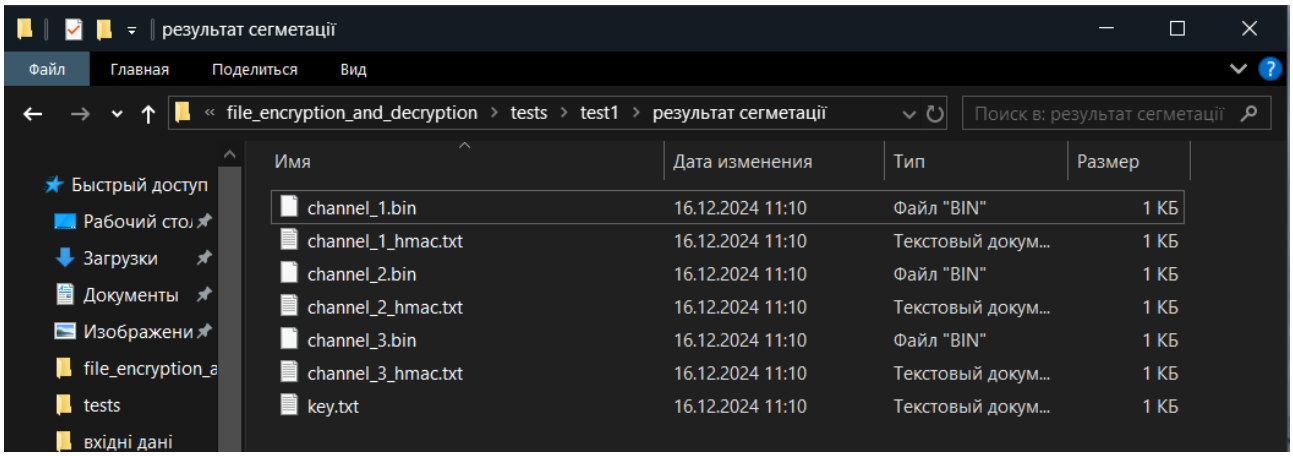


Рисунок 3.17 – Вікно каталогу після сегментації файлу

Для перевірки правильності HMAC обирається пункт «Перевірити HMAC». У спливаючому вікні обирається каталог, де збережені сегменти та їх HMAC. Результат наведено на рисунку 3.18.

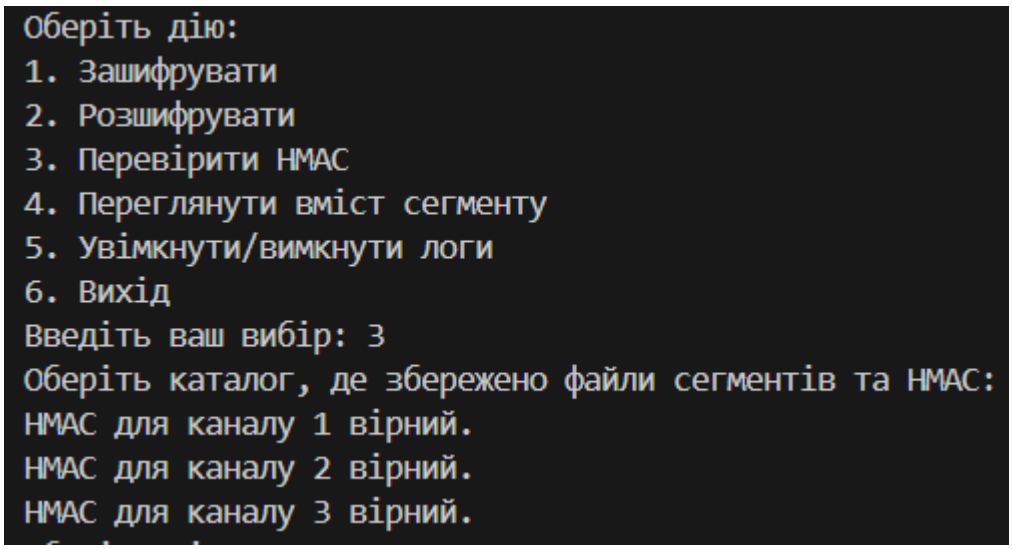


Рисунок 3.18 – Результат перевірки HMAC

При відновленні файлу необхідно використати спливаючі вікна, окремо для кожного сегменту та секретного ключа. Після вибору місця для збереження відновленого файлу можна переглянути його вміст та порівняти з оригіналом (рис 3.19).

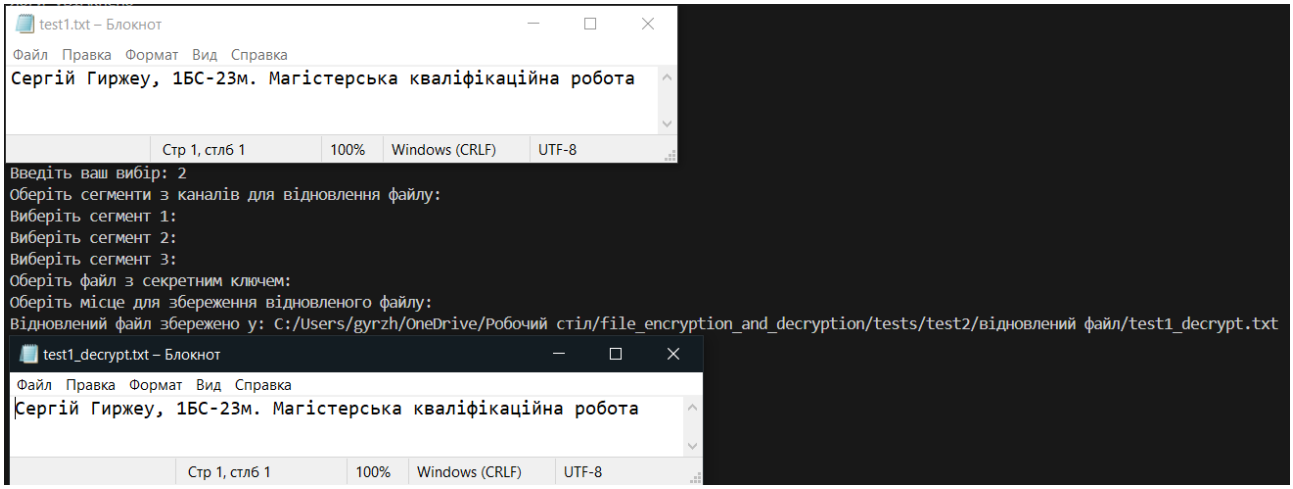


Рисунок 3.19 – Порівняння вмісту оригіналу та відновленого файлу

Проведене тестування показало, що всі основні функції програмного засобу, включно з сегментацією (шифруванням) та відновленням (розшифруванням) файлу, а також механізмом перевірки HMAC, працюють коректно. Кожна функція була перевірена за допомогою тестових сценаріїв, які підтвердили цілісність даних, надійність механізму шифрування та відсутність

помилок під час відновлення файлів. Таким чином, розроблений засіб відповідає вимогам безпеки та ефективності.

3.6 Висновки до розділу 3

У третьому розділі було розроблено та реалізовано програмний засіб для захищеного передавання файлів, який забезпечує високу надійність і відповідність вимогам інформаційної безпеки. Основною складовою програми є алгоритми сегментації та відновлення файлів, які дозволяють ефективно шифрувати дані шляхом їх розподілу між різними сегментами та подальшого збирання. Цей підхід забезпечує рівномірний розподіл інформації, що підвищує стійкість системи до криптоаналізу та ускладнює несанкціонований доступ до даних.

Особливу увагу було приділено механізму перевірки цілісності даних за допомогою HMAC. Цей механізм дозволяє виявляти спроби модифікації даних, гарантуючи їх автентичність під час передачі. Генерація HMAC здійснюється з використанням алгоритму AES у режимі CBC із застосуванням дзеркального дублювання ключа, що забезпечує надійність формування криптографічного підпису.

Крім того, у програмному засобі реалізовано функціонал логування, який дозволяє адаптувати обсяг виводу інформації на екран відповідно до потреб користувача. Це сприяє зручності використання програми, забезпечуючи детальний вивід для розробників або скорочений — для кінцевих користувачів.

Тестування підтвердило працездатність основних функцій, включно з сегментацією (шифруванням) та відновленням (розшифруванням) файлів. Усі функції були перевірені та показали коректну роботу. Механізм перевірки HMAC успішно виявляв зміни у даних, а результати тестування відновлених файлів продемонстрували їх ідентичність оригіналу. Механізм логування також функціонував відповідно до очікувань, забезпечуючи необхідний рівень інформативності.

4 ЕКОНОМІЧНА ЧАСТИНА

4.1 Оцінювання комерційного потенціалу розробки

Комерційний та технологічний аудит проводиться підвищення рівня захисту даних, що передаються мережею шляхом розробки методу захищеного передавання даних та засобу, який його реалізує.

Для проведення технологічного аудиту було залучено 3-х незалежних експертів:

– Експерт 1: Рура В.О. Засновник та керівник VipNet (ФОП, провайдер власного інтернет-зв’язку в Тульчинському р-н Вінницької обл.)

– Експерт 2: Цибрій Є. В. Фахівець з телекомунікаційних послуг, VipNet

– Експерт 3: Лужецький В.А., д.т.н., проф. зав. кафедри захисту інформації Вінницького національного технічного університету.

Для проведення технологічного аудиту було використано таблицю 4.1 [1] в якій за п’ятибальною шкалою використовуючи 12 критеріїв здійснено оцінку комерційного потенціалу.

Таблиця 4.1 – Рекомендовані критерії оцінювання комерційного потенціалу розробки та їх можлива бальна оцінка

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
ри-терій	0	1	2	3	4
Технічна здійсненність концепції:					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги (недоліки):					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів

Продовження табл. 4.1

4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування

Продовження табл. 4.1

10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Таблиця 4.2 – Рівні комерційного потенціалу розробки

Середньоарифметична сума балів СБ, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0-10	Низький
11-20	Нижче середнього
21-30	Середній
31-40	Вище середнього
41-48	Високий

В таблиці 4.3 наведено результати оцінювання експертами комерційного потенціалу розробки.

Таблиця 4.4 – Результати оцінювання комерційного потенціалу розробки

Критерії	Прізвище, ініціали, посада експерта		
	Експерт 1	Експерт 2	Експерт 3
	Бали, виставлені експертами:		
1	4	4	4
2	3	3	3
3	3	3	4
4	4	4	4
5	4	4	4
6	4	4	4
7	4	4	4
8	4	4	4
9	3	3	3
10	4	4	4
11	4	4	4
12	3	3	3
Сума балів	$СБ_1$=31	$СБ_2$=34	$СБ_3$=34
Середньоарифметична сума балів $\overline{СБ}$	$\overline{СБ} = \frac{\sum_{1}^{3} СБ_i}{3} = \frac{31+34+34}{3} = 33$		

Середньоарифметична оцінка, отримана на основі експертних висновків, становить 33 бали, і згідно з таблицею 4.3, це вказує на вище середнього рівень комерційного потенціалу результатів проведених досліджень.

Розроблений метод захищеного передавання файлів буде реалізований через створення програмного засобу, що базується на описаному в дипломній роботі підході. Цей засіб забезпечить сегментацію файлів, передачу їх частин через різні канали зв’язку (наприклад, Telegram, Signal, Пошта Gmail) із застосуванням алгоритмів шифрування, а також механізмів перевірки цілісності даних. Особливістю реалізації стане можливість об'єднання переданих сегментів на стороні отримувача для відновлення початкового файлу.

Програмний продукт може бути впроваджений у державних установах, які працюють із секретними або конфіденційними даними, у фінансових

компаніях для захисту транзакційних даних, у комерційних організаціях, які працюють із персональною чи корпоративною інформацією, а також в освітніх та наукових закладах. Завдяки своїм перевагам розробка буде корисною для будь-яких структур, які потребують надійного захисту при обміні інформацією.

Проведемо оцінку якості і конкурентоспроможності нової розробки порівняно з аналогом.

В якості аналога для розробки було обрано месенджер Signal. Signal є одним із найбільш захищених месенджерів, який використовує наскрізне шифрування для передавання повідомлень і файлів. Месенджер має відкритий вихідний код, що дозволяє проводити аудит безпеки сторонніми експертами. Signal забезпечує конфіденційність переданих даних завдяки функціям автоматичного видалення повідомлень та файлів.

Основними недоліками аналога є:

1. Обмеження на максимальний розмір файлів для передавання, що ускладнює використання для великих файлів.

2. Залежність від централізованих серверів, що створює ризики для конфіденційності даних у разі компрометації серверної інфраструктури.

Також до недоліків можна віднести відсутність механізмів багатоканальної передачі даних. Це робить систему менш стійкою до атак на єдиний канал передавання, наприклад, типу «людина посередині».

У розробці дана проблема вирішується шляхом впровадження методу сегментації файлів і передавання їх частин через три незалежні канали зв’язку. Такий підхід значно знижує ризик перехоплення чи модифікації файлів, оскільки кожен канал передає лише частину даних. Крім того, використовуються сучасні механізми перевірки цілісності даних (HMAC), які гарантують автентичність переданих файлів. Це дозволяє забезпечити передачу навіть великих файлів без обмежень на їх розмір.

Аналіз існуючих методів вирішення задачі, що розглядається, показав, що більшість сучасних сервісів використовують традиційні методи шифрування, такі як AES для симетричного шифрування та RSA для асиметричного. Ці

методи забезпечують високий рівень конфіденційності, але залишаються вразливими до атак на єдиний канал зв’язку. Методи наскрізного шифрування, як у Signal, вирішують частину проблем, але не гарантують захисту при компрометації серверів чи перехопленні частини трафіку.

Система випереджає аналог за такими параметрами, як:

– Використання трьох незалежних каналів для передачі даних, що мінімізує ризик перехоплення інформації.

– Підвищена стійкість до атак типу «людина посередині» завдяки передачі кожного сегмента файлу окремим каналом.

– Інтеграція механізмів перевірки цілісності (HMAC), що забезпечує автентичність даних і своєчасне виявлення спроб їх модифікації.

Запропонована розробка демонструє суттєве покращення безпеки в порівнянні з існуючими рішеннями, завдяки чому може бути інтегрована в системи, що потребують високого рівня захисту даних.

В таблиці 4.5 наведені основні техніко-економічні показники аналога і нової розробки.

Проведемо оцінку якості продукції, яка є найефективнішим засобом забезпечення вимог споживачів та порівняємо її з аналогом.

Таблиця 4.5 – Основні параметри нової розробки та товару-конкурента

Показник	Варіанти		Відносний показник якості	Коефіцієнт вагомості параметра
	Базовий (товар-конкурент)	Новий (інноваційне рішення)		
1	2	3	4	5
Зашифрування даних (рівень захисту), бали	9	10	1,11	25%
Максимальний розмір файлу	100	1000	10.0	20%
Час передачі файлу	60	50	1.2	20%
Стійкість до перехоплення (рівень захисту) , бали	8	10	1.25	20%
Надійність передачі (рівень захисту) , бали	8	9	1.13	15%

Визначимо відносні одиничні показники якості по кожному параметру за формулами (4.1) та (4.2) і занесемо їх у відповідну колонку табл. 4.6.

$$q_i = \frac{P_{Hi}}{P_{Бi}} \tag{4.1}$$

або

$$q_i = \frac{P_{Бi}}{P_{Hi}} \tag{4.2}$$

де $P_{Пi}$, $P_{Бi}$ – числові значення i-го параметру відповідно нового і базового виробів.

$$q_1 = \frac{10}{9} = 1{,}11;$$
$$q_2 = \frac{1000}{100} = 10;$$
$$q_3 = \frac{60}{50} = 1{,}2;$$
$$q_4 = \frac{10}{8} = 1{,}25;$$
$$q_5 = \frac{9}{8} = 1{,}13.$$

Відносний рівень якості нової розробки визначаємо за формулою:

$$К_{я.в.} = \sum_{i=1}^{n} q_i \cdot \alpha_i, \tag{4.3}$$

$$К_{я.в.} = 1{,}11 \cdot 0{,}25 + 10 \cdot 0{,}20 + 1{,}2 \cdot 0{,}2 + 1{,}25 \cdot 0{,}2 + 1{,}13 \cdot 0{,}15 = 2{,}9$$

Відносний коефіцієнт показника якості нової розробки більший одиниці, отже нова розробка якісніший базового товару-конкурента.

Наступним кроком є визначення конкурентоспроможності товару. Конкурентоспроможність товару є головною умовою конкурентоспроможності підприємства на ринку і важливою основою прибутковості його діяльності.

Однією із умов вибору товару споживачем є збіг основних ринкових характеристик виробу з умовними характеристиками конкретної потреби покупця. Такими характеристиками найчастіше вважають нормативні та технічні параметри, а також ціну придбання та вартість споживання товару.

В табл. 4.6 наведено технічні та економічні показники для розрахунку конкурентоспроможності нової розробки відносно товару-аналога, технічні дані взяті з попередніх розрахунків.

Таблиця 4.6 – Нормативні, технічні та економічні параметри нової розробки і товару-виробника

Показники	Варіанти	
	Базовий (товар-конкурент)	Новий (інноваційне рішення)
1	2	3
1. Нормативно-технічні показники		
Зашифрування даних (рівень захисту), бали	9	10
Максимальний розмір файлу	100	1000
Час передачі файлу	60	50
Стійкість до перехоплення (рівень захисту) , бали	8	10
Надійність передачі (рівень захисту) , бали	8	9
2. Економічні показники		
Ціна придбання, грн.	20000	15000

Загальний показник конкурентоспроможності інноваційного рішення (К) з урахуванням вищезазначених груп показників можна визначити за формулою:

$$K = \frac{I_{m.n.}}{I_{e.n.}}, \tag{4.4}$$

де $I_{m.n.}$ – індекс технічних параметрів; $I_{e.n.}$ – індекс економічних параметрів.

Індекс технічних параметрів є відносним рівнем якості інноваційного рішення. Індекс економічних параметрів визначається за формулою (4.5)

$$I_{e.n.} = \frac{\sum_{i=1}^{n} P_{Hei}}{\sum_{i=1}^{n} P_{Бei}}, \tag{4.5}$$

де $P_{Неі}$, $P_{Беі}$ – економічні параметри (ціна придбання та споживання товару) відповідно нового та базового товарів.

$$\text{I}_{\text{е.п.}} = \frac{20000}{15000} = 1{,}33;$$

$$\text{К} = \frac{2{,}9}{1{,}33} = 2{,}2.$$

Зважаючи на розрахунки, можна зробити висновок, що нова розробка буде біль конкурентоспроможна, ніж конкурентний товар.

4.2 Прогнозування витрат на виконання науково-дослідної роботи

Витрати, пов’язані з проведенням науково-дослідної роботи групуються за такими статтями: витрати на оплату праці, витрати на соціальні заходи, матеріали, паливо та енергія для науково-виробничих цілей, витрати на службові відрядження, програмне забезпечення для наукових робіт, інші витрати, накладні витрати.

1. Основна заробітна плата кожного із дослідників $З_О$, якщо вони працюють в наукових установах бюджетної сфери визначається за формулою:

$$\text{З}_\text{О} = \frac{\text{М}}{\text{Т}_\text{р}} * t \text{ (грн)} \qquad (4.6)$$

де $М$ – місячний посадовий оклад конкретного розробника (інженера, дослідника, науковця тощо), грн.;

$T_Р$ – число робочих днів в місяці; приблизно $T_Р \approx 21...23$ дні;

t – число робочих днів роботи дослідника.

Зведемо сумарні розрахунки до таблиця 4.7.

Таблиця 4.7 – Заробітна плата дослідника в науковій установі бюджетної сфери

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату грн.
Керівник	18000	857,1	5	4286
Програміст	22000	1047,6	21	22000
Всього				26286

2. Витрати на основну заробітну плату робітників ($З_р$) за відповідними найменуваннями робіт розраховують за формулою:

$$З_р = \sum_{i=1}^{n} C_i \cdot t_i, \quad (4.7)$$

де $С_і$ – погодинна тарифна ставка робітника відповідного розряду, за виконану відповідну роботу, грн/год;

$t_і$ – час роботи робітника на виконання певної роботи, год.

Погодинну тарифну ставку робітника відповідного розряду $С_і$ можна визначити за формулою:

$$C_i = \frac{М_м \cdot К_і \cdot К_с}{Т_р \cdot t_{зм}}, \quad (4.8)$$

де $М_М$ – розмір прожиткового мінімуму працездатної особи або мінімальної місячної заробітної плати (залежно від діючого законодавства), грн;

$К_і$ – коефіцієнт міжкваліфікаційного співвідношення для встановлення тарифної ставки робітнику відповідного розряду;

$К_с$ – мінімальний коефіцієнт співвідношень місячних тарифних ставок робітників першого розряду з нормальними умовами праці виробничих об’єднань і підприємств до законодавчо встановленого розміру мінімальної заробітної плати.

$T_р$ – середня кількість робочих днів в місяці, приблизно Тр = 21…23 дні;

$t_{зм}$ – тривалість зміни, год.

Таблиця 4.8 – Величина витрат на основну заробітну плату робітників

Найменування робіт	Тривалість роботи, год	Розряд роботи	Погодинна тарифна ставка, грн	Величина оплати на робітника, грн
1. Розробка алгоритму	2	2	52,4	104,8
2. Написання коду	3	3	64,3	192,9
3. Налагодження програми	2	5	81,0	161,9
4. Випробувальні	6	2	52,4	314,3
5. Документування системи	3	2	71,4	214,3
Всього				988,1

3. Розрахунок додаткової заробітної плати робітників

Додаткова заробітна плата $З_д$ всіх розробників та робітників, які приймали устають в розробці нового технічного рішення розраховується як 10 - 12 % від основної заробітної плати робітників.

На даному підприємстві додаткова заробітна плата начисляється в розмірі 11% від основної заробітної плати.

$$З_д = (З_о + З_р) * \frac{Н_{доп}}{100\%} \quad (4.9)$$

$$З_д = 0{,}11 * (26286 + 988{,}1) = 3000{,}12 \text{ (грн)}$$

4. Нарахування на заробітну плату $Н_{ЗП}$ дослідників та робітників, які брали участь у виконанні даного етапу роботи, розраховуються за формулою (4.10):

$$Н_{ЗП} = (З_о + З_р + З_д) * \frac{\beta}{100} \text{ (грн)} \quad (4.10)$$

де $З_о$ – основна заробітна плата розробників, грн.;

$З_д$ – додаткова заробітна плата всіх розробників та робітників, грн.;

$З_р$ – основну заробітну плату робітників, грн.;

β – ставка єдиного внеску на загальнообов'язкове державне соціальне страхування, % .

Дана діяльність відноситься до бюджетної сфери, тому ставка єдиного внеску на загальнообов'язкове державне соціальне страхування буде складати 22%, тоді:

$$Н_{ЗП} = (26286 + 988{,}1 + 3000{,}12) * \frac{22}{100} = 6660{,}26 \text{ (грн)}$$

5. Сировина та матеріали.

До статті «Сировина та матеріали» належать витрати на сировину, основні та допоміжні матеріали, інструменти, пристрої та інші засоби й предмети праці, які придбані у сторонніх підприємств, установ і організацій та витрачені на проведення досліджень за прямим призначенням згідно з нормами їх витрачання, а також витрачені придбані напівфабрикати, що підлягають монтажу або виготовленню й додатковій обробці в цій організації, чи дослідні зразки, що виготовляються виробниками за документацією наукової організації.

Витрати на матеріали (М) у вартісному вираженні розраховуються окремо для кожного виду матеріалів за формулою:

$$М = \sum_{i=1}^{n} H_j \cdot Ц_j \cdot К_j - \sum_{i=1}^{n} В_j \cdot Ц_{вj}, \qquad (4.11)$$

де Нj – норма витрат матеріалу j-го найменування, кг;

n – кількість видів матеріалів;

Цj – вартість матеріалу j-го найменування, грн/кг;

Кj – коефіцієнт транспортних витрат, (Кj = 1,1 … 1,15);

Вj – маса відходів j-го найменування, кг;

Цвj – вартість відходів j-го найменування, грн/кг.

Проведені розрахунки зведені в таблицю 4.9.

Таблиця 4.9 – Витрати на матеріали

Найменування матеріалу, марка, тип, сорт	Ціна за 1 кг, грн	Норма витрат, шт	Вартість витраченого матеріалу, грн
Папір	180	1	180
Ручка	20	1	20
Блокнот	50	1	50
Флешка	300	1	300
З врахуванням коефіцієнта транспортування			605

6. Програмне забезпечення для наукових (експериментальних) робіт

Балансову вартість програмного забезпечення розраховують за формулою:

$$B_{прг} = \sum_{i=1}^{k} Ц_{iпрг} \cdot C_{прг.i} \cdot K_i \ , \qquad (4.12)$$

де $Ц_{iпрг}$ – ціна придбання одиниці програмного засобу даного виду, грн;

$C_{прг.i}$ – кількість одиниць програмного забезпечення відповідного найменування, які придбані для проведення досліджень, шт.;

K_i – коефіцієнт, що враховує інсталяцію, налагодження програмного засобу тощо, ($K_i = 1,10…1,12$);

k – кількість найменувань програмних засобів.

Отримані результати необхідно звести до таблиці:

Таблиця 4.10 – Витрати на придбання програмних засобів по кожному виду

Найменування програмного засобу	Кількість, шт	Ціна за одиницю, грн	Вартість, грн
Ліцензійне ПЗ	1	500	500
База даних для зберігання тестів	1	1000	1000
Всього з врахуванням налагодження			1650

7. Амортизація обладнання, програмних засобів та приміщень

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо, можуть бути розраховані з використанням прямолінійного методу амортизації за формулою:

$$A_{обл} = \frac{Ц_б}{T_в} \cdot \frac{t_{вик}}{12}, \quad (4.13)$$

де $Ц_б$ – балансова вартість обладнання, програмних засобів, приміщень тощо, які використовувались для проведення досліджень, грн;

$t_{вик}$ – термін використання обладнання, програмних засобів, приміщень під час досліджень, місяців;

$T_в$ – строк корисного використання обладнання, програмних засобів, приміщень тощо, років.

Проведені розрахунки необхідно звести до таблиці 4.11.

Таблиця 4.11 – Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн	Строк корисного використання, років	Термін використання обладнання, місяців	Амортизаційні відрахування, грн
Комп'ютер	25000	2	3	1041,67
Всього				1041,67

8. До статті «Паливо та енергія для науково-виробничих цілей» відносяться витрати на всі види палива й енергії, що безпосередньо використовуються з технологічною метою на проведення досліджень.

$$В_е = \sum_{i=1}^{n} \frac{W_{yt} \cdot t_i \cdot Ц_е \cdot К_{впі}}{\eta_i} \quad (4.14)$$

де W_{yt} – встановлена потужність обладнання на певному етапі розробки, кВт;

t_i – тривалість роботи обладнання на етапі дослідження, год;

$Ц_е$ – вартість 1 кВт-години електроенергії, грн;

$К_{впі}$ – коефіцієнт, що враховує використання потужності, $К_{впі} < 1$;

$\eta_і$ – коефіцієнт корисної дії обладнання, $\eta_і < 1$.

Для написання магістерської роботи використовується персональний комп'ютер для якого розрахуємо витрати на електроенергію.

$$В_е = \frac{0{,}5 \cdot 180 \cdot 10{,}5 \cdot 0{,}5}{0{,}8} = 590{,}63$$

9. Службові відрядження.

Витрати за статтею «Службові відрядження» розраховуються як 20…25% від суми основної заробітної плати дослідників та робітників за формулою:

$$В_{св} = (З_о + З_р) * \frac{Н_{св}}{100\%}, \qquad (4.15)$$

де $Н_{св}$ – норма нарахування за статтею «Службові відрядження».

$$В_{св} = 0{,}2 * (26286 + 988{,}1) = 5454{,}76$$

10. Накладні (загальновиробничі) витрати Внзв охоплюють: витрати на управління організацією, оплата службових відряджень, витрати на утримання, ремонт та експлуатацію основних засобів, витрати на опалення, освітлення, водопостачання, охорону праці тощо. Накладні (загальновиробничі) витрати Внзв можна прийняти як (100…150)% від суми основної заробітної плати розробників та робітників, які виконували дану МКНР, тобто:

$$В_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%}, \qquad (4.16)$$

де $Н_{нзв}$ – норма нарахування за статтею «Інші витрати».

$$В_{нзв} = (26286 + 988{,}1) \cdot \frac{100}{100\%} = 27273{,}81 \text{ грн}$$

Сума всіх попередніх статей витрат дає витрати, які безпосередньо стосуються даного розділу МКНР

В = 26286 + 988,1 + 3000,12 + 6660,26 + 605 + 1650 + 1041,67 + 590,63 + 5454,76 + 27273,81 = 73550,05 грн

Прогнозування загальних втрат ЗВ на виконання та впровадження результатів виконаної МКНР здійснюється за формулою:

$$ЗВ = \frac{В}{\eta}, \quad (4.17)$$

де η – коефіцієнт, який характеризує стадію виконання даної НДР.

Оскільки, робота знаходиться на стадії науково-дослідних робіт, то коефіцієнт β = 0,7.

Звідси:

$$ЗВ = \frac{73550,05}{0,7} = 105071,51 \text{ грн.}$$

4.3 Розрахунок економічної ефективності науково-технічної розробки

У даному підрозділі кількісно спрогнозуємо, яку вигоду, зиск можна отримати у майбутньому від впровадження результатів виконаної наукової роботи. Розрахуємо збільшення чистого прибутку підприємства $\Delta П_i$, для кожного із років, протягом яких очікується отримання позитивних результатів від впровадження розробки, за формулою

$$\Delta П_i = \sum_{1}^{n} \left(\Delta Ц_o \cdot N + Ц_o \cdot \Delta N\right)_i \cdot \lambda \cdot \rho \cdot \left(1 - \frac{\upsilon}{100}\right) \quad (4.18)$$

де $\Delta Ц_0$ – покращення основного оціночного показника від впровадження результатів розробки у даному році.

N – основний кількісний показник, який визначає діяльність підприємства у даному році до впровадження результатів наукової розробки;

ΔN – покращення основного кількісного показника діяльності підприємства від впровадження результатів розробки:

$Ц_o$ – основний оціночний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки;

n – кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки:

л – коефіцієнт, який враховує сплату податку на додану вартість. Ставка податку на додану вартість дорівнює 20%, а коефіцієнт *л* = 0,8333.

р – коефіцієнт, який враховує рентабельність продукту. р = 0,25;

х – ставка податку на прибуток. У 2024 році – 18%.

Припустимо, що ціна зросте на 100 грн. Кількість одиниць реалізованої продукції також збільшиться: протягом першого року на 200 шт., протягом другого року – на 300 шт., протягом третього року на 400 шт. Реалізація продукції до впровадження розробки складала 1 шт., а її ціна до 15000 грн. Розрахуємо прибуток, яке отримає підприємство протягом трьох років.

$$\Delta П_1 = [100 \cdot 1 + (15000 + 100) \cdot 200] \cdot 0{,}833 \cdot 0{,}25 \cdot \left(1 + \frac{18}{100}\right) = 515913{,}11 \text{ грн.}$$

$$\Delta П_2 = [100 \cdot 1 + (15000 + 100) \cdot (200 + 300)] \cdot 0{,}833 \cdot 0{,}25 \cdot \left(1 + \frac{18}{100}\right) = 1289840{,}1 \text{ грн.}$$

$$\Delta П_3 = [100 \cdot 1 + (15000 + 100) \cdot (200 + 300 + 400)] \cdot 0{,}833 \cdot 0{,}25 \cdot \left(1 + \frac{18}{100}\right) = 2321632{,}1 \text{ грн.}$$

4.4 Розрахунок ефективності вкладених інвестицій та періоду їх окупності

Розрахуємо основні показники, які визначають доцільність фінансування наукової розробки певним інвестором, є абсолютна і відносна ефективність вкладених інвестицій та термін їх окупності.

Розрахуємо величину початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки.

$$PV = k_{\text{інв}} \cdot \text{ЗВ}, \tag{4.19}$$

$k_{\text{інв}}$ – коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію. Це можуть бути витрати на підготовку приміщень, розробку технологій, навчання персоналу, маркетингові заходи тощо ($k_{\text{інв}} = 2…5$).

$$PV = 2 \cdot 105071{,}51 \quad = 210143{,}02$$

Розрахуємо абсолютну ефективність вкладених інвестицій $E_{абс}$ згідно наступної формули:

$$E_{абс} = (ПП - PV) \tag{4.20}$$

де ПП – приведена вартість всіх чистих прибутків, що їх отримає підприємство від реалізації результатів наукової розробки, грн.;

$$ПП = \sum_{1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \tag{4.21}$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДЦКР, грн.;

Т – період часу, протягом якою виявляються результати впровадженої НДДКР, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні; для України цей показник знаходиться на рівні 0,2;

t – період часу (в роках).

$$\text{ПП} = \frac{515913{,}11}{(1+0{,}2)^1} + \frac{1289840{,}1}{(1+0{,}2)^2} + \frac{2321632{,}1}{(1+0{,}2)^3} = 2675435{,}99 \text{ грн.}$$

$$\text{Е}_{\text{абс}} = (2675435{,}99 - 210143{,}02) = 2465292{,}98 \text{грн.}$$

Оскільки $E_{абс} > 0$ то вкладання коштів на виконання та впровадження результатів НДДКР може бути доцільним.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $E_в$. Для цього користуються формулою:

$$E_в = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1, \tag{4.22}$$

$T_ж$ – життєвий цикл наукової розробки, роки.

$$\text{Е}_\text{в} = \sqrt[3]{1 + \frac{2465292,98}{210143,02}} - 1 = 1,90 = 190\%$$

Визначимо мінімальну ставку дисконтування, яка у загальному вигляді визначається за формулою:

$$\tau = d + f, \tag{4.23}$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2022 році в Україні d= (0,14…0,2);

f – показник, що характеризує ризикованість вкладень; зазвичай, величина f = (0,05...0,1).

$$\tau_{\min} = 0,18 + 0,05 = 0,23.$$

Так як $E_в > \tau_{\min}$ то інвестор може бути зацікавлений у фінансуванні даної наукової розробки.

Розрахуємо термін окупності вкладених у реалізацію наукового проекту інвестицій за формулою:

$$T_{ок} = \frac{1}{E_в} \tag{4.24}$$

$$\text{Т}_\text{ок} = \frac{1}{1,9} = 0,5 \text{ роки}$$

Так як $T_{ок} \leq$ 3...5-ти років, то фінансування даної наукової розробки в принципі є доцільним.

4.5 Висновки до розділу 4

Результати здійсненого технологічного аудиту вказують на вище середнього рівень комерційного потенціалу. У порівнянні з аналогічним виробом виявлено, що нова розробка вищої якості і більш конкурентоспроможна, як з технічних, так і економічних позначень.

Вкладені інвестиції в даний проект окупляться через 6 місяців. Загальні витрати складають 105071,51 грн.

ВИСНОВКИ

Проведений аналіз сучасних підходів до захищеного передавання файлів показав, що вони передбачають використання симетричного та асиметричного шифрування, а також гешування. Виявлено переваги та недоліки цих методів, їхню відповідність вимогам сучасної кібербезпеки, а також визначено основні загрози, такі як перехоплення даних, модифікація, несанкціонований доступ.

Запропонований метод захищеного передавання файлів передбачає використання поділу файлу на три сегменти і передавання їх трьома різними каналами Telegram, Signal та Gmail. Ще однією особливістю цього методу є ущільнення файлу перед його поділом на сегменти. Це забезпечує зменшення його обсягу та ускладнення аналізу зловмисником оскільки результатом ущільнення є набір символів, які мають майже однакову частоту появи, що і унеможливлює статистичний аналіз зашифрованого файлу.

При розробці програмного засобу, що реалізує запропонований метод захищеного передавання файлів, особливу увагу приділено механізму перевірки цілісності переданого файлу, який дозволяє виявляти спроби модифікації даних.

Результати тестування програмного засобу для захищеного передавання файлів показали, що відновлені файли ідентичні оригінальним даним. Механізм логування працює відповідно до очікувань, забезпечуючи необхідний рівень інформативності для моніторингу процесів.

Виконаний розрахунок економічної ефективності запропонованого рішення показав, що розроблений програмний засіб для захищеного передавання файлів є конкурентоспроможним як із технічної, так і з економічної точки зору, демонструючи високий рівень якості та надійності.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Основи кібербезпеки. *Головна*. URL: https://moz.gov.ua/uk/osnovi-kiberbezpeki-2 (дата звернення: 10.09.2024).

2. Посібник із кібергігієни та стійкості до кіберзагроз | Security Insider. *Your request has been blocked. This could be due to several reasons.* URL: https://www.microsoft.com/uk-ua/security/security-insider/practical-cyber-defense/cyber-resilience-hygiene-guide (дата звернення: 10.09.2024).

3. Як безпечно зберігати та ділитися файлами в Інтернеті - CyberSTAR. *CyberSTAR*. URL: https://cyber-star.org/ua/cs-articles/how-to-keep-and-share-files-securely-online-ua/ (дата звернення: 10.09.2024).

4. Шифрування: типи і алгоритми. Що це, чим відрізняються і де використовуються? • Hostpro Wiki. *Hostpro Wiki*. URL: https://hostpro.ua/wiki/ua/security/encryption-types-algorithms/ (дата звернення: 12.09.2024).

5. Шифрування: Типи й алгоритми. *hostkoss*. URL: https://hostkoss.com/b/uk/encryption-types-algorithms/ (дата звернення: 12.11.2024).

6. Асиметричне шифрування. *Словник з інформатики*. URL: https://іт.словник.укр/index.php/Асиметричне_шифрування (дата звернення: 12.09.2024).

7. Academy B. Симетричне та асиметричне шифрування | Binance Academy. *Binance Academy*. URL: https://academy.binance.com/uk/articles/symmetric-vs-asymmetric-encryption (дата звернення: 12.09.2024).

8. Гешування Даних: Методи та Алгоритми. *CyberSet*. URL: https://cyberset.com.ua/encryption/heshuvannya-danykh-metody/ (дата звернення: 14.09.2024).

9. Academy B. Що таке хешування? | Binance Academy. *Binance Academy*. URL: https://academy.binance.com/uk/articles/what-is-hashing (дата звернення: 17.09.2024).

10. Staying Safe on WhatsApp | WhatsApp Security. *WhatsApp.com*. URL: https://www.whatsapp.com/security/ (дата звернення: 18.09.2024).

11. Telegram FAQ. *Telegram*. URL: https://telegram.org/faq#secret-chats (дата звернення: 19.09.2024).

12. Documentation. *Signal Messenger*. URL: https://signal.org/docs/ (дата звернення: 19.09.2024).

13. Cloud-Sicherheit und DatenschutzÂ | Google Workspace. *Google Workspace*. URL: https://workspace.google.com/security/ (дата звернення: 20.09.2024).

14. Dropbox Security Features. Dropbox, Inc. URL: https://www.dropbox.com/security Дата звернення: 20.09.2024.

15. OneDrive Security and Privacy Overview. Microsoft Corporation. URL: https://www.microsoft.com/en-us/microsoft-365/onedrive/online-cloud-storage. (дата звернення: 20.09.2024).

16. Gmail Community. *Google Help*. URL: https://support.google.com/mail/community/?hl=en&gpf=#!forum/gmail (дата звернення: 20.11.2024).

17. Outlook Email Encryption and Security. Microsoft Corporation. URL: https://support.microsoft.com/en-us/outlook/email-encryption. Дата звернення: 21.09.2024.

18. Миронович В. Які месенджери безпечніші. *Speka - онлайн медіа про технології та підприємництво | SPEKA.media | SPEKA.media*. URL: https://speka.media/naskilki-vasi-cati-v-mesendzerax-v-bezpeci-p0ewlv (дата звернення: 19.09.2024).

19. Prostobank Consulting. DropBox, OneDrive, Google Drive і Box: хмарні сховища. *Рейтинги банківських послуг України для населення в 2024*. URL: https://bankchart.com.ua/finansoviy_gid/groshi_rodini/statti/porivnyannya_hm

arnih_shovisch_onedrive_dropbox_google_drive_i_box (дата звернення: 19.09.2024).

20. Проблеми інформаційної безпеки. Загрози при роботі в Інтернеті і їх уникнення. *Всеосвіта*. URL: https://vseosvita.ua/lesson/problemy-informatsiinoi-bezpeky-zahrozy-pry-roboti-v-interneti-i-ikh-unyknennia-697721.html (дата звернення: 21.09.2024).

21. Перехоплення Даних Мережі. *CyberSet*. URL: https://cyberset.com.ua/network/arp-spoofing-percept-data-mergesecurity/ (дата звернення: 22.09.2024).

22. Загрози безпеці інформації – Wiki ТНТУ. *Wiki ТНТУ*. URL: https://wiki.tntu.edu.ua/Загрози_безпеці_інформації (дата звернення: 23.09.2024).

23. Терміни та визначення у сфері кібербезпеки. *Всеосвіта*. URL: https://www.vpnunlimited.com/ua/help/cybersecurity/unauthorized-access?srsltid=AfmBOoqCO67xMGJZQHp8ZHSVQuj2aPkRIcbbf47GTV4-HUBvJtnh4GK6 (дата звернення: 24.09.2024).

24. Що таке фішинг, як його розпізнати і як із ним боротися. *Digvel*. URL: https://digvel.com.ua/blog/shho-take-fishyng-i-yak-iz-nym-borotysya/ (дата звернення: 25.09.2024).

25. Oteir N. DDoS-атака: що це таке та в чому небезпека для сервера?. *INTROSERV*. URL: https://introserv.com/ua/blog/ddos-ataka-shho-cze-take-ta-v-chomu-nebezpeka-dlya-servera/ (дата звернення: 26.09.2024).

26. Що таке шкідливе програмне забезпечення? Визначення й типи | Захисний комплекс Microsoft. *Microsoft*. URL: https://www.microsoft.com/uk-ua/security/business/security-101/what-is-malware (дата звернення: 28.09.2024).

27. Методи і технології захисту комп’ютерних мереж (фізичний та канальний рівні). *Телекомунікації та захист інформації*. URL: https://ela.kpi.ua/server/api/core/bitstreams/e5c73b24-058a-42d8-bd2d-03a17e2c7c9a/content (дата звернення: 04.12.2024).

28. Лужецький В. А., Кожухівський А. Д, Войтович О. П Основи інформаційної безпеки : навчальний посібник Вінниця : ВНТУ, 2013. 221 с.

29. Жилін А., Шаповал О., Успенський О. Технології захисту інформації в інформаційно-телекомунікаційних системах : навч. посіб. Київ, 2020. 214 с. URL: https://ela.kpi.ua/server/api/core/bitstreams/d99a0045-e907-4d17-afc1-5431f67d2444/content (дата звернення: 04.10.2024).

30. Що таке Зашифрована передача файлів - Терміни та визначення кібербезпеки. *Кібербезпека*.URL: https://www.vpnunlimited.com/ua/help/cybersecurity/encrypted-file-transfer (дата звернення: 05.10.2024).

31. Гешування. Характеристики функцій. *ScienceDirect*. URL: https://www.sciencedirect.com/science/article/pii/S0893608017302147 (дата звернення: 11.10.2024).

32. GeeksforGeeks. What is HMAC(Hash based Message Authentication Code)? - GeeksforGeeks. *GeeksforGeeks*. URL: https://www.geeksforgeeks.org/what-is-hmachash-based-message-authentication-code/ (дата звернення: 20.10.2024).

33. Відомості про AES-шифрування. *TechVizor*. URL: https://techvizor.info/zagalni-vidomosti-pro-aes-shyfruvannya/?utm_source=chatgpt.com (дата звернення: 20.10.2024).

34. gzip – Support for gzip files. *Python documentation*. URL: https://docs.python.org/uk/3/library/gzip.html (дата звернення: 10.11.2024).

35. Основи функціонування систем криптографічного захисту інформації - Бібліотека BukLib.net. *Головна - Бібліотека BukLib.net*. URL: https://buklib.net/books/28747/ (дата звернення: 22.11.2024).

36. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт / Уклад. : В. О. Козловський, О. Й. Лесько, В. В. Кавецький. – Вінниця : ВНТУ, 2021. – 42 с.

37. Методичні вказівки до виконання магістерських кваліфікаційних робіт / Уклад. : В.А. Каплун, Л. М. Куперштейн, Ю. В. Баришев, О.П. Войтович – Вінниця : ВНТУ, 2024.

ДОДАТКИ

Додаток А

Лістинг коду засобу

encryption_program.py:

```
import os
import gzip
import secrets
import hmac
from Crypto.Cipher import AES
from hashlib import sha256
from tkinter import filedialog, Tk
from file_decryption import decrypt_file

LOG_ENABLED = False  # Глобальна змінна для контролю логів

def log(message):
    if LOG_ENABLED:
        print(message)

def generate_key():
    """Генерація секретного ключа розміром 64 біти (8 байтів)."""
    K = secrets.token_bytes(8)
    K_list = [int(byte) for byte in K]
    formatted_key = "K = {" + ", ".join(map(str, K_list)) + "}."
    return K, formatted_key

def compress_data(input_file):
    """Ущільнення даних файлу за допомогою gzip."""
    with open(input_file, 'rb') as f_in:
        data = f_in.read()
    return gzip.compress(data)

def pad_data(data, block_size):
    padding_length = block_size - (len(data) % block_size)
    return data + bytes([padding_length] * padding_length)

def generate_hmac_aes(data, key):
    """Генерація HMAC для даних за допомогою AES-128."""
    padded_data = pad_data(data, AES.block_size)
    aes = AES.new(key.ljust(16, b'0'), AES.MODE_ECB)
    hashed = aes.encrypt(padded_data)[:16]
    return hashed.hex()

def segment_file(data, key):
    """Сегментація та шифрування даних на основі ключа."""
    log("Перші 100 байтів файлу після ущільнення: " +
str(list(data[:100])))

    F = {i: [] for i in range(256)}

    log("Крок 1. Розподіл байтів по масивах Fi:")
    for i, byte in enumerate(data):
        if i < 8:
            F[key[i % len(key)]].append(byte)
            log(f"k{i+1} = {key[i % len(key)]}, m{i+1} = {byte} записуємо
в F{key[i % len(key)]}: F{key[i % len(key)]} := F{key[i % len(key)]} + {byte} =
{F[key[i % len(key)]]}")
        else:
            F[data[i - 1]].append(byte)
```

```
                log(f"m{i} = {data[i - 1]}, m{i+1} = {byte} → F{data[i - 1]}
:= F{data[i - 1]} + {byte} = {F[data[i - 1]]}")

        log("Маємо такі масиви:")
        result = " ".join([f"F{k} = {v}" for k, v in F.items() if v])
        log(result)

        канали = [[], [], []]
        for i in range(256):
            if 0 <= i <= 84:
                канали[0].append(len(F[i]))
            elif 85 <= i <= 168:
                канали[1].append(len(F[i]))
            else:
                канали[2].append(len(F[i]))

        for i in range(256):
            if 0 <= i <= 84:
                канали[0].extend(F[i])
            elif 85 <= i <= 168:
                канали[1].extend(F[i])
            else:
                канали[2].extend(F[i])

        log("Крок 2. Розподіл по каналах:")
        for idx, канал in enumerate(канали):
            log(f"Канал {idx+1}: {канал}")

        return канали

    def write_segments(канали, key, output_dir):
        """Запис сегментованих даних, ключа та HMAC у файли."""
        if not os.path.exists(output_dir):
            os.makedirs(output_dir)

        for i, канал in enumerate(канали):
            channel_file = os.path.join(output_dir, f'channel_{i+1}.bin')
            hmac_file = os.path.join(output_dir, f'channel_{i+1}_hmac.txt')

            with open(channel_file, 'wb') as f:
                f.write(bytearray(канал))

            # Генерація HMAC для кожного каналу
            hmac_value = generate_hmac_aes(bytearray(канал), key)
            with open(hmac_file, 'w') as f:
                f.write(hmac_value)

        key_file = os.path.join(output_dir, 'key.txt')
        with open(key_file, 'w') as f:
            f.write("Секретний ключ:\n")
            f.write(", ".join(map(str, key)))

        return [os.path.join(output_dir, f'channel_{i+1}.bin') for i in
range(len(канали))] + [os.path.join(output_dir, f'channel_{i+1}_hmac.txt') for i
in range(len(канали))] + [key_file]

    def view_channel():
        """Перегляд вмісту обраного сегменту."""
        print("Оберіть сегмент файлу для перегляду:")
        root = Tk()
        root.withdraw()
        root.attributes('-topmost', True)
        channel_file = filedialog.askopenfilename(title="Оберіть файл каналу")
```

```
            if not channel_file:
                print("Файл не обрано.")
                return

            with open(channel_file, 'rb') as f:
                channel_data = f.read()

            print(f"Вміст обраного сегмента ({channel_file}):")
            print(list(channel_data))

        def verify_hmac():
            """Перевірка HMAC для сегментів."""
            print("Оберіть каталог, де збережено файли сегментів та HMAC:")
            root = Tk()
            root.withdraw()
            root.attributes('-topmost', True)
            directory = filedialog.askdirectory(title="Оберіть каталог із
сегментами")

            if not directory:
                print("Каталог не обрано.")
                return

            key_file = os.path.join(directory, 'key.txt')
            if not os.path.exists(key_file):
                print("Файл ключа не знайдено.")
                return

            with open(key_file, 'r') as f:
                key_line = f.readlines()[1].strip()
                key = bytes(map(int, key_line.split(', ')))

            for i in range(1, 4):
                channel_file = os.path.join(directory, f'channel_{i}.bin')
                hmac_file = os.path.join(directory, f'channel_{i}_hmac.txt')

                if not os.path.exists(channel_file) or not
os.path.exists(hmac_file):
                    print(f"Файли для каналу {i} не знайдено.")
                    continue

                with open(channel_file, 'rb') as f:
                    channel_data = f.read()

                with open(hmac_file, 'r') as f:
                    stored_hmac = f.read().strip()

                computed_hmac = generate_hmac_aes(channel_data, key)

                if computed_hmac == stored_hmac:
                    print(f"HMAC для каналу {i} вірний.")
                else:
                    print(f"HMAC для каналу {i} НЕВІРНИЙ!")

        def main():
            global LOG_ENABLED
            while True:
                print("Оберіть дію:")
                print("1. Зашифрувати")
                print("2. Розшифрувати")
                print("3. Перевірити HMAC")
                print("4. Переглянути вміст сегменту")
                print("5. Увімкнути/вимкнути логи")
                print("6. Вихід")
```

```
            choice = input("Введіть ваш вибір: ")

            if choice == '1':
                root = Tk()
                root.withdraw()
                root.attributes('-topmost', True)
                input_file = filedialog.askopenfilename(title="Оберіть файл
для зашифрування")

                if not input_file:
                    print("Файл не обрано.")
                    continue

                key, formatted_key = generate_key()
                print(f"Згенерований ключ: {formatted_key}")
                compressed_data = compress_data(input_file)
                канали = segment_file(compressed_data, key)
                output_dir = filedialog.askdirectory(title="Оберіть каталог
для збереження сегментів")

                if not output_dir:
                    print("Каталог не обрано.")
                    continue

                segment_files = write_segments(канали, key, output_dir)
                print(f"Сегменти, ключ та HMAC записано у: {segment_files}")

            elif choice == '2':
                decrypt_file()

            elif choice == '3':
                verify_hmac()

            elif choice == '4':
                view_channel()

            elif choice == '5':
                LOG_ENABLED = not LOG_ENABLED
                print("Логи увімкнено" if LOG_ENABLED else "Логи вимкнено")

            elif choice == '6':
                print("Вихід із програми.")
                break

            else:
                print("Некоректний вибір. Спробуйте ще раз.")

    if __name__ == "__main__":
        main()


    file_decryption.py:

    import os
    import gzip
    from tkinter import filedialog, Tk

    LOG_ENABLED = False  # Глобальна змінна для контролю логів

    def log(message):
        if LOG_ENABLED:
            print(message)

    def decompress_data(data):
```

```
            """Відновлення файлу за допомогою gzip."""
            return gzip.decompress(data)

        def read_channel_file():
            """Вибір та читання файлу сегмента."""
            root = Tk()
            root.withdraw()
            root.attributes('-topmost', True)
            channel_file = filedialog.askopenfilename(title="Оберіть файл
сегмента")

            if not channel_file:
                log("Файл не обрано.")
                return None

            with open(channel_file, 'rb') as f:
                return f.read()

        def reconstruct_file(channels_data, key):
            """Відновлення оригінального файлу з сегментованих даних."""
            F = {i: [] for i in range(256)}

            log(f"Секретний ключ: {key}")

            log("Вміст сегментів каналів:")
            for idx, channel_data in enumerate(channels_data):
                log(f"Канал {idx + 1}: {list(channel_data)}")

            log("Крок 1. Відновлення масивів F0, F1, …,F255:")
            for idx, channel_data in enumerate(channels_data):
                log(f"Канал {idx + 1}:")

                if idx == 0:
                    lengths = channel_data[:85]
                    data = channel_data[85:]
                elif idx == 1:
                    lengths = channel_data[:84]
                    data = channel_data[84:]
                else:
                    lengths = channel_data[:87]
                    data = channel_data[87:]

                for i, length in enumerate(lengths):
                    if length > 0:
                        log(f"L{85 * idx + i if idx < 2 else 169 + i} = {length}")

                log(f"Дані після відділення довжин для сегмента {idx + 1}:
{list(data)}")

                data_index = 0
                for i, length in enumerate(lengths):
                    if length > 0:
                        segment = data[data_index:data_index + length]
                        F[85 * idx + i if idx < 2 else 169 + i] = list(segment)
                        data_index += length

            log("Крок 2. Об’єднання масивів сегментів з каналів:")
            result = " ".join([f"F{k} = {v}" for k, v in F.items() if v])
            log(result)

            reconstructed_data = []
            log("Крок 3. Відновлення вихідного файлу:")
            for i, k in enumerate(key[:8]):
                if k in F and F[k]:
```

```
                reconstructed_data.append(F[k][0])
                log(f"k{i + 1} = {k} → m{i + 1} = {F[k][0]}")
                F[k] = F[k][1:]
            else:
                log(f"Помилка: Масив F{k} порожній або відсутній")

        while True:
            prev_byte = reconstructed_data[-1]
            if prev_byte in F and F[prev_byte]:
                reconstructed_data.append(F[prev_byte][0])
                log(f"m{len(reconstructed_data)} = F{prev_byte}[0] =
{F[prev_byte][0]}")
                F[prev_byte] = F[prev_byte][1:]
            else:
                log(f"Завершено: Масив F{prev_byte} порожній або відсутній")
                break

        return bytes(reconstructed_data)

    def decrypt_file():
        global LOG_ENABLED
        print("Оберіть сегменти з каналів для відновлення файлу:")
        channels_data = []

        for i in range(3):
            print(f"Виберіть сегмент {i+1}:")
            data = read_channel_file()
            if data is None:
                print(f"Дані з каналу {i+1} не вибрано.")
                return
            channels_data.append(data)

        print("Оберіть файл з секретним ключем:")
        root = Tk()
        root.withdraw()
        root.attributes('-topmost', True)
        key_file = filedialog.askopenfilename(title="Оберіть файл секретного
ключа")

        if not key_file:
            print("Файл ключа не обрано.")
            return

        with open(key_file, 'r') as f:
            key_line = f.readlines()[1].strip()
            key = list(map(int, key_line.split(', ')))

        print("Оберіть місце для збереження відновленого файлу:")
        save_path = filedialog.asksaveasfilename(title="Збережіть відновлений
файл", defaultextension=".bin")

        if not save_path:
            print("Місце для збереження не обрано.")
            return

        reconstructed_data = reconstruct_file(channels_data, key)
        try:
            decompressed_data = decompress_data(reconstructed_data)
            with open(save_path, 'wb') as f:
                f.write(decompressed_data)
            print(f"Відновлений файл збережено у: {save_path}")
        except gzip.BadGzipFile:
            print("Помилка: Відновлений файл не є gzip-архівом.")
```

```
def main():
    global LOG_ENABLED
    while True:
        print("Оберіть дію:")
        print("1. Розшифрувати файл")
        print("2. Увімкнути/вимкнути логи")
        print("3. Вихід")
        choice = input("Введіть ваш вибір: ")

        if choice == '1':
            decrypt_file()
        elif choice == '2':
            LOG_ENABLED = not LOG_ENABLED
            print("Логи увімкнено" if LOG_ENABLED else "Логи вимкнено")
        elif choice == '3':
            print("Вихід із програми.")
            break
        else:
            print("Некоректний вибір. Спробуйте ще раз.")

if __name__ == "__main__":
    main()
```

94

Додаток Б

ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи: Метод захищеного передавання файлів
Автор роботи: Гиржеу Сергій Дмитрович

Тип роботи: магістерська кваліфікаційна робота

Підрозділ кафедра захисту інформації ФІТКІ

Керівник Лужецький Володимир Андрійович, д.т.н., професор

Показники звіту подібності

Turnitin	
Оригінальність	98 %
Загальна схожість	2 %

Аналіз звіту подібності (відмітити потрібне):

√ 1. Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

□ 2. Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на доопрацювання.

□ 3. Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Заявляю, що ознайомлений з повним звітом подібності, який був згенерований Системою щодо роботи.

Автор роботи _____ (підпис) Сергій ГИРЖЕУ (прізвище, ініціали)

Особа, відповідальна за перевірку _____ (підпис) Валентина КАПЛУН (прізвище, ініціали)

Керівник роботи _____ (підпис) Володимир ЛУЖЕЦЬКИЙ (прізвище, ініціали)

ІЛЮСТРАТИВНА ЧАСТИНА

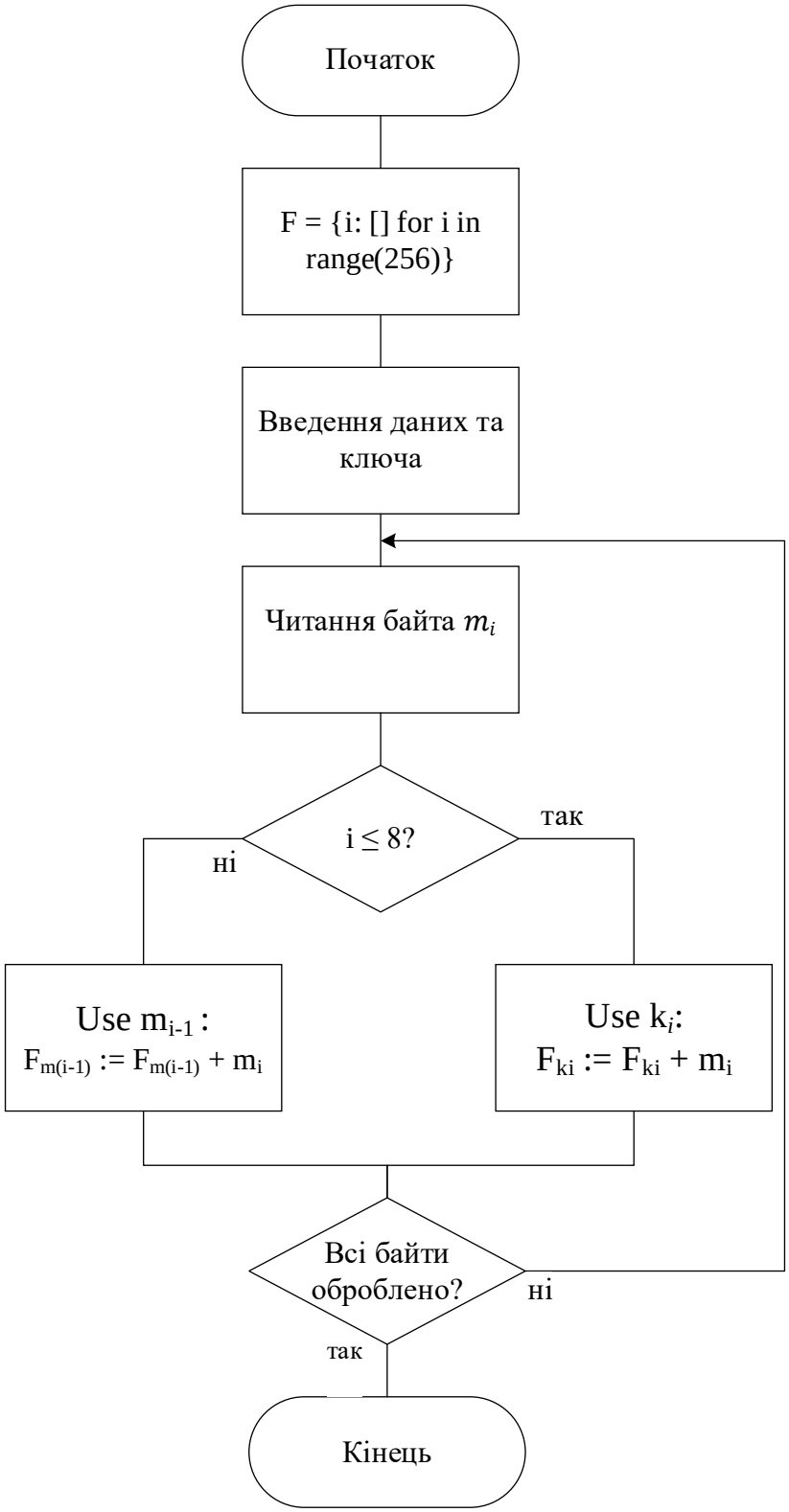
МЕТОД ЗАХИЩЕНОГО ПЕРЕДАВАННЯ ФАЙЛІВ

Узагальнений алгоритм сегментації

Узагальнений алгоритм відновлення файлу

Алгоритм генерації секретного ключа

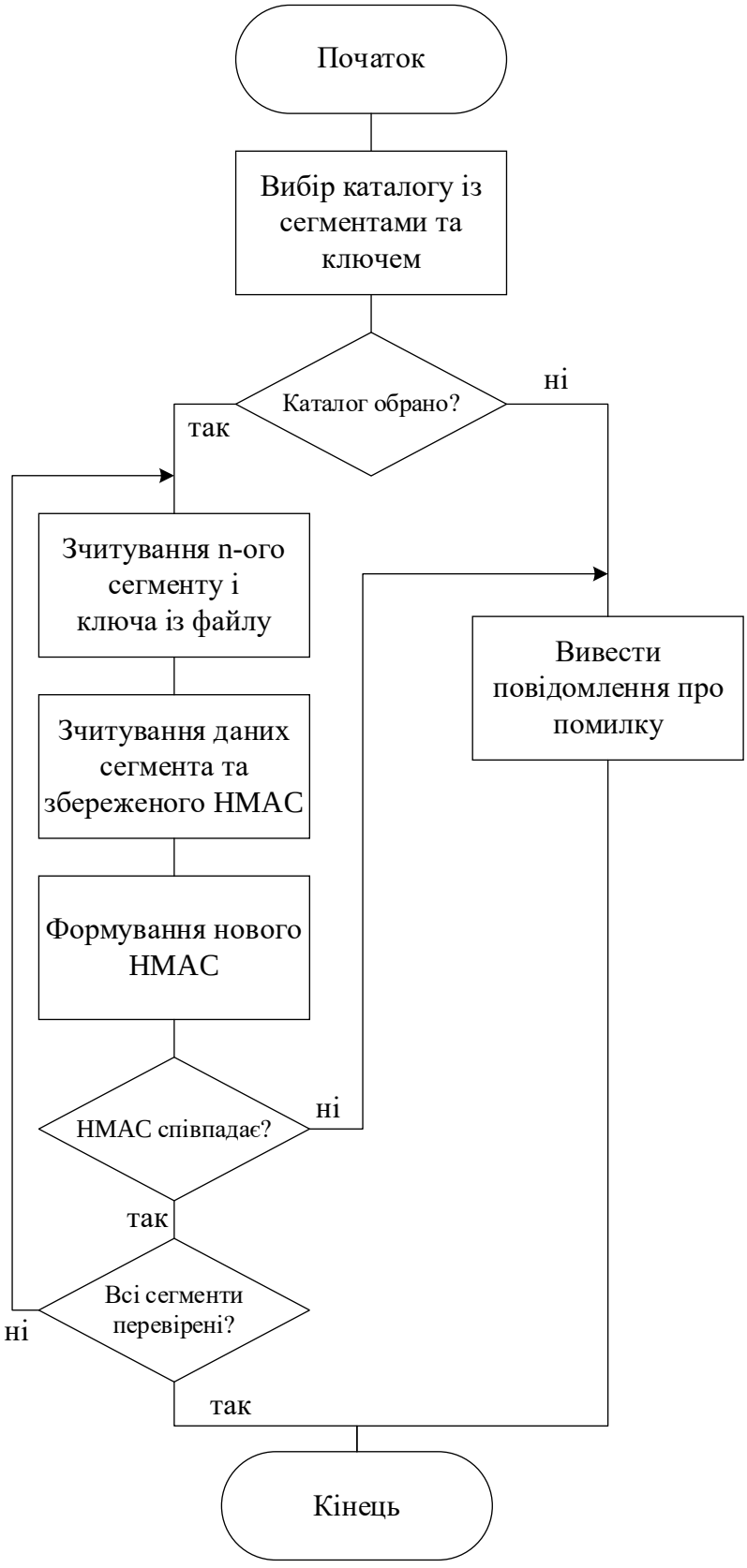
Алгоритм ініціалізації масивів та розподілу байтів файлу



Алгоритм формування ущільненого та відновленого файлу

Алгоритм формування HMAC

Алгоритм перевірки HMAC



ВІДГУК ОПОНЕНТА

на магістерську кваліфікаційну роботу

студента Гиржеу Сергія Дмитровича

на тему «Метод захищеного передавання файлів»

Питання захисту інформації у відкритих мережах стає все важливішим через значне зростання обсягів даних, що передаються, та появу більш досконалих кіберзагроз. Це й обумовлює актуальність теми магістерської кваліфікаційної роботи, яка присвячена розробці методу захищеного передавання файлів.

Автор продемонстрував вміння аналізувати сучасні методи захисту, запропонував оригінальний підхід, що базується на поділі файлів на сегменти та передачі їх різними каналами зв'язку. Особливістю методу є використання механізму HMAC для перевірки цілісності, що підвищує стійкість до атак типу «людина посередині».

Практична цінність роботи полягає у створенні програмного засобу для реалізації запропонованого методу, який був протестований за різними сценаріями виникнення помилок під час передавання даних.

Текстову та ілюстративну частини роботи виконано у повному обсязі згідно з індивідуальним завданням, їх оформлення відповідає вимогам, що висуваються до магістерських кваліфікаційних робіт.

Зауваження до змісту магістерської кваліфікаційної роботи.

1. Відсутні оцінки швидкості захищеного передавання файлів з використанням одразу трьох каналів зв’язку: Telegram, Signal, Gmail.

2. Відсутні кількісні оцінки підвищення рівня захисту інформації, що передається.

Вважаю, що магістерська кваліфікаційна робота заслуговує на оцінку С за ECTS, а її автор - Гиржеу Сергій Дмитрович заслуговує на присвоєння кваліфікації ступінь вищої освіти «магістр», галузь знань 12 «Інформаційні технології», спеціальність 125 «Кібербезпека та захист інформації», освітня програма «Безпека інформаційних і комунікаційних систем».

Опонент
зав. кафедри ПЗ
д.т.н., професор

Олександр РОМАНЮК

ВІДГУК

на магістерську кваліфікаційну роботу студента групи 1БС-23м Гиржеу Сергія Дмитровича «Метод захищеного передавання файлів»

Магістерська кваліфікаційна робота присвячена важливій проблемі забезпечення безпеки інформації в умовах сучасних кіберзагроз. Питання захищеного передавання даних є надзвичайно важливим для інформаційної безпеки в різних сферах — від приватних користувачів до державних органів та установ. Саме це й обумовлює актуальність наведених у роботі досліджень.

Науковою новизною роботи є удосконалений метод захищеного передавання файлів, що базується на сегментації файлів із використанням кількох каналів передачі та застосуванням HMAC для перевірки цілісності вмісту файлів. Теоретичні результати підтверджено тестуванням програмного засобу, що реалізує запропонований метод.

Під час роботи студент показав достатній рівень теоретичних знань і практичних навичок. Індивідуальне завдання на магістерську кваліфікаційну роботу в основному виконано. Оформлення пояснювальної записки відповідає вимогам, що висуваються до магістерських кваліфікаційних робіт.

Недоліком роботи є те, що недостатньо докладно розглянуто можливість інтеграції розробленого методу в існуючі інформаційні системи.

Вважаю, що магістерська кваліфікаційна робота заслуговує на оцінку С за ECTS, а її автор - Гиржеу Сергій Дмитрович заслуговує на присвоєння кваліфікації ступінь вищої освіти «магістр», галузь знань 12 «Інформаційні технології», спеціальність 125 «Кібербезпека та захист інформації», освітня програма «Безпека інформаційних і комунікаційних систем».

Керівник
магістерської кваліфікаційної роботи
зав. кафедри ЗІ, д.т.н., професор Володимир ЛУЖЕЦЬКИЙ