Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Бакалаврська кваліфікаційна робота

на тему: «Розробка вебзастосунку для криптографічного захисту файлів»

Виконав: студент 4 курсу
групи 1ПІ-21б
спеціальності
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)
Гиренко В.В.
(прізвище та ініціали)
Керівник: к.т.н., доц. каф. ПЗ Майданюк В.П.
(прізвище та ініціали)
Рецензент: к.т.н., доц. каф. ОТ Обертюх М. Р.
(прізвище та ініціали)

Допущено до захисту
Завідувач кафедри ПЗ
д.т.н., проф. Романюк О.Н.
«9» _____ 2025 р.

ВНТУ – 2025

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Рівень вищої освіти перший бакалаврський
Галузь знань 12 – Інформаційні технології
Спеціальність 121 – Інженерія програмного забезпечення
Освітньо-професійна програма – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н._____
«21» березня 2025 р.

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

Гиренку Вадиму Віталійовичу

1. Тема роботи – «Розробка вебзастосунку для криптографічного захисту файлів»

Керівник роботи: Майданюк Володимир Павлович, к.т.н., доц. кафедри ПЗ, затверджені наказом вищого навчального закладу від 20 березня 2025 р. № 97.

2. Строк подання студентом роботи 2 червня 2025 р.

3. Вихідні дані до роботи: шифрування файлі, операційна система – Windows, середовище розробки – Visual Studio Code, мова програмування – JavaScript

4. Зміст розрахунково-пояснювальної записки: вступ; аналіз стану розвитку для криптографічного захисту файлів; розробка методів та алгоритмів програмного застосунку; розробка застосунку; тестування застосунку; висновки; список використаних джерел; додатки.

5. Перелік графічного матеріалу: титульний слайд; актуальність теми; мета, об'єкт та предмет дослідження; задачі дослідження, наукова новизна, практична цінність одержаних результатів; порівняльний аналіз аналогів.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Майданюк В.П., к.т.н., доцент кафедри ПЗ	24.03.25	02.06.25

7. Дата видачі завдання 24 березня 2025 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз стану розвитку застосунків для для криптографічного захисту файлів	25.03.25 – 01.04.25	вик.
2	Розробка методів та алгоритмів вебзастосунку	02.04.25- 12.04.25	вик.
3	Варіантний аналіз та вибір мови програмування розробки	13.04.25- 27.04.25	вик.
4	Розробка застосунку	28.04.25- 09.05.25	вик
5	Тестування застосунку	10.05.25- 20.05.25.	вик
6	Оформлення матеріалів до захисту БКР	21.05.25- 30.05.25	вик

Студент _____ Гиренко В. В.
(підпис) (прізвище та ініціали)

Керівник бакалаврської кваліфікаційної роботи _____ Майданюк В. П.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Бакалаврська дипломна робота складається з 61 сторінок формату А4, на яких є 24 рисунків та 2 таблиць, список використаних джерел містить 30 найменувань.

Метою роботи є підвищення зручності процесу шифрування файлів за рахунок розробки веб-застосунку, що дозволяє виконувати операції шифрування у браузері без встановлення додаткових програм.

У роботі проведено аналіз сучасних підходів до криптографічного захисту даних. Для реалізації обрано шифрування методом гамування з використанням генератора псевдовипадкових чисел (ПВЧ) для створення гами шифру. Розробка виконана в середовищі Microsoft Visual Studio Code мовою програмування JavaScript.

Розроблений веб-застосунок який може використовуватись як у практичних цілях для демонстрації роботи простого симетричного шифрування, так і в освітньому процесі для вивчення основ криптографії та генерації псевдовипадкових чисел.

Ключові слова: Visual Studio Code, JavaScript, симетричне шифрування, веб застосунок.

ABSTRACT

The bachelor's thesis consists of 61 A4 pages, including 24 figures and 2 tables. The list of references contains 30 sources.

The aim of the thesis is to improve the convenience of the file encryption process by developing a web application that enables encryption operations directly in the browser without the need to install additional software.

The thesis presents an analysis of modern approaches to cryptographic data protection. The chosen method for implementation is stream cipher encryption using a pseudorandom number generator (PRNG) to create the keystream. The development was carried out in Microsoft Visual Studio Code using the JavaScript programming language.

The developed web application can be used both for practical purposes—to demonstrate the operation of a simple symmetric encryption method—and in the educational process to study the basics of cryptography and pseudorandom number generation.

Keywords: Visual Studio Code, JavaScript, symmetric encryption, web application.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Сучасний етап розвитку інформаційних технологій характеризується стрімким зростанням обсягів цифрової інформації, яка передається, обробляється та зберігається в глобальних комп'ютерних мережах. З появою нових форм зберігання і обробки даних, таких як хмарні технології, веб-застосунки та великі дані, виникає безліч проблем, пов'язаних з безпекою цієї інформації. Зокрема, однією з основних загроз є несанкціонований доступ до інформації, її зміна або крадіжка. В умовах постійного зростання цифрових даних і все більш активного використання хмарних сервісів для зберігання та передачі файлів необхідно забезпечити високий рівень захисту цієї інформації. Веб-сервіси, що надають доступ до файлів, можуть бути піддані різноманітним загрозам, що ставить питання про необхідність реалізації ефективних методів захисту даних, таких як криптографія.

Криптографія — це наука про методи захисту інформації від несанкціонованого доступу, змін і крадіжки через її перетворення в зашифровану форму. Цей процес дозволяє забезпечити конфіденційність, цілісність, автентичність та невідмовність переданої інформації. Розвиток криптографічних алгоритмів надає можливість створювати високонадійні системи захисту даних, що мають важливе значення для збереження безпеки в умовах зростаючої кількості кіберзагроз.

До недавнього часу криптографічний захист даних в основному використовувався великими компаніями та урядовими структурами, і доступ до подібних рішень був обмежений. Однак з розвитком веб-технологій, зокрема в напрямку надання користувачам можливості шифрування та дешифрування даних без необхідності встановлення додаткового програмного забезпечення, виникла потреба в створенні нових інтуїтивно зрозумілих та доступних криптографічних веб-сервісів. Це дозволяє простим користувачам швидко і безпечно виконувати операції з даними, захищаючи їх від несанкціонованого доступу або змін.

Загрозливі тенденції щодо безпеки даних вимагають високого рівня захисту

навіть для звичайних користувачів. Поява таких технологій, як хмарні сервіси, призводить до нових викликів для забезпечення конфіденційності. Більшість традиційних методів захисту, зокрема паролі або обмежений доступ до серверів, є недостатньо ефективними для захисту файлів у таких середовищах. Тому все більшу значущість набувають рішення, що забезпечують криптографічний захист, дозволяючи користувачам самостійно шифрувати і дешифрувати свої дані [1].

Актуальність цієї теми також підкреслюється зростанням популярності використання веб-застосунків для виконання різноманітних задач, що включають зберігання, обробку та передачу даних. Веб-технології забезпечують високу доступність сервісів, але з цим виникає проблема забезпечення належного рівня захисту. Тому існує необхідність у створенні зручних і безпечних веб-застосунків, які дозволяють здійснювати криптографічний захист даних без потреби в додатковому програмному забезпеченні та забезпечують доступ до функцій шифрування для широкого кола користувачів, що свідчить про актуальність теми роботи.

Зв'язок роботи з науковими програмами, планами, темами. Робота виконувалася згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Мета та завдання дослідження. Метою роботи є підвищення зручності процесу шифрування файлів за рахунок розробки веб-застосунку, що дозволяє виконувати операції шифрування у браузері без встановлення додаткових програм.

Основними задачами дослідження є:

- розробити архітектуру вебсистеми;
- розробити користувацький інтерфейс;
- розробити модулі шифрування та дешифрування;
- розробити генерацію, збереження та перевірку криптографічних ключів;
- розробити алгоритми роботи застосунку;
- розробити функціонал застосунку;
- провести тестування розробленого застосунку.

Об'єкт дослідження – процес розробки вебзастосунку для

криптографічного захисту файлів.

Предмет дослідження – способи і програмні засоби реалізації криптографічного захисту файлів у вебсередовищі.

Методи дослідження. У процесі досліджень використовувались: теорія алгоритмів, теорія криптографії, методи побудови вебзастосунків, способи тестування для розробки вебзастосунку криптографічного захисту файлів; комп'ютерне моделювання для аналізу та перевірки отриманих теоретичних положень.

Новизна отриманих результатів.

1. Подальшого розвитку отримав метод побудови застосунків для криптографічного захисту даних, у якому, на відміну від існуючих, використано клієнт-серверну модель вебзастосунку, що підвищує зручність процесу шифрування за рахунок можливості виконувати операції у браузері без встановлення додаткових програм.

2. Подальшого розвитку отримав метод генерації гами шифру, у якому, на відміну від існуючих, використано конгруентний генератор псевдовипадкових чисел, що підвищує криптостійкість за рахунок можливості генерації ключа розміром, який перевищує розмір даних, що шифруються.

Практична цінність отриманих результатів. Практична цінність одержаних результатів полягає в тому, що на основі проведених у бакалаврській дипломній роботі досліджень було розроблено зручний і функціональний вебзастосунок для криптографічного захисту файлів, який може бути використаний як інструмент для забезпечення інформаційної безпеки при роботі з конфіденційними даними.

Особистий внесок здобувача. Усі наукові результати, викладені у бакалаврській кваліфікаційній роботі, отримані автором особисто. Автором самостійно розроблено архітектуру вебзастосунку, реалізовано клієнтську та серверну частину програмного забезпечення, інтегровано криптографічні алгоритми, а також проведено тестування функціональності системи [2].

Апробація результатів роботи. Описані у бакалаврській роботі положення доповідались на конференції «Молодь в науці: дослідження, проблеми, перспективи (МН-2025)» [3].

1 АНАЛІЗ СТАНУ РОЗВИТКУ ВЕБСИСТЕМ ДЛЯ КРИПТОГРАФІЧНОГО ЗАХИСТУ ФАЙЛІВ

1.1 Аналіз стану питання вебзастосунків для криптографічного захисту файл

Розробка вебзастосунку для криптографічного захисту файлів [4] представляє собою комбінацію класичних математичних Методів і сучасних обчислювальних технологій. Криптографія, завдяки своїй здатності забезпечувати конфіденційність, цілісність і автентичність даних, отримала широке застосування в інформаційній безпеці, банківських системах, електронній комерції та цифровій комунікації.

Сучасні алгоритми шифрування базуються на класичних підходах, таких як симетричне шифрування (AES, DES), а також гібридних схемах, які поєднують обидва способи для досягнення балансу між швидкодією та надійністю [5]. Завдяки інтеграції криптографічних бібліотек у серверну та клієнтську частини вебзастосунку, відкривається можливість забезпечення надійного шифрування файлів без потреби в сторонніх додатках. Це дозволяє створювати інтерактивні рішення, які надають користувачам можливість шифрувати та дешифрувати файли безпосередньо у веббраузері.

У багатьох сучасних вебзастосунках використовуються гібридні способи, які поєднують криптографічні алгоритми із сучасними технологіями автентифікації, такими як токени доступу (JWT), цифрові сертифікати та біометричні дані. Такий підхід дозволяє адаптувати рівень захисту під конкретні умови, наприклад, враховуючи рівень довіри до користувача або рівень чутливості переданих даних, що сприяє досягненню більшої гнучкості системи безпеки. Ринок програмного забезпечення для криптографічного захисту файлів включає як open-source проєкти, так і комерційні рішення, розроблені на різних мовах програмування, включаючи Python, JavaScript та Go. Сучасні вебзастосунки часто мають зручні інтерфейси користувача, які дозволяють завантажувати файли, обирати тип шифрування, генерувати ключі та зберігати результати, а також легко

інтегруються в більші інформаційні системи, наприклад, у хмарні сервіси чи корпоративні платформи.

Важливо відзначити, що зростаюча роль вебзастосунків у повсякденному житті користувачів формує нові виклики у сфері захисту інформації. Оскільки велика кількість конфіденційних даних передається через Інтернет, зокрема через вебформи, API-запити або файлообмінні інтерфейси, виникає необхідність забезпечення криптографічного захисту на всіх етапах обробки інформації. Це вимагає від розробників не лише реалізації сучасних алгоритмів шифрування, а й дотримання кращих практик зберігання ключів, автентифікації та захисту комунікаційних каналів.

Крім того, важливим аспектом при створенні подібних систем є забезпечення зручності користувача без компромісів у безпеці. Наприклад, необхідно мінімізувати кількість кроків для шифрування чи дешифрування файлу, при цьому впроваджуючи автоматичне попередження у випадках неправильного вибору файлу або помилки автентифікації. Додатково слід впроваджувати механізми перевірки цілісності даних за допомогою хеш-функцій, таких як SHA-256, що дозволяє гарантувати, що дані не були змінені під час передачі чи зберігання.

Одним із ключових напрямів удосконалення сучасних вебзастосунків є забезпечення конфіденційності без шкоди для зручності користувача. Це досягається за допомогою асинхронної обробки даних, використання криптографії на стороні клієнта та розподілених систем управління ключами. Усе частіше використовуються технології WebAssembly [6] та Web Crypto API, які дозволяють реалізовувати складні криптографічні операції безпосередньо у браузері з високою швидкістю. Це не тільки зменшує навантаження на сервер, але й мінімізує ризики витоку даних під час передачі.

Завдяки впровадженню WebAssembly з'явилася можливість використовувати оптимізовані криптографічні бібліотеки, розроблені на C або C++, без необхідності переписувати код на JavaScript. Це значно підвищує продуктивність операцій шифрування та розшифрування у браузері, що особливо важливо для

обробки великих файлів. Крім того, підтримка багатопоточності у сучасних браузерах дозволяє виконувати обробку даних у фоновому режимі без помітного зниження швидкості роботи інтерфейсу користувача.

Попри перераховані переваги, розробникам доводиться вирішувати проблеми, пов’язані з забезпеченням сумісності між браузерами, безпечним збереженням ключів шифрування та захистом від атак типу “людина посередині”.

Поєднання криптографічних способи із сучасними вебтехнологіями, а також розробка більш оптимізованих та адаптивних алгоритмів захисту відкривають нові можливості, але при цьому й створюють додаткові вимоги до масштабованості, продуктивності та стійкості до атак вебзастосунків.

1.2 Порівняльний аналіз аналогів

Для дослідження вимог до вебзастосунків для криптографічного захисту файлів та визначення функціоналу власної розробки, необхідно провести аналіз аналогів. Наразі існують три провідних вебзастосунки для криптографічного захисту файлів Cryptomator, VeraCrypt, BitLocker**.**

Cryptomator [7] – це безкоштовний і відкритий вебзастосунок для криптографічного захисту файлів, що забезпечує зручне і надійне шифрування даних. Програма підтримує шифрування на рівні файлів, що дозволяє створювати захищені зашифровані сховища. Cryptomator використовує алгоритм AES-256 для забезпечення високого рівня захисту, що робить її популярною серед користувачів, які потребують високої безпеки з низьким рівнем складності використання.

Програма підтримує кросплатформеність і доступна для Windows, macOS, Linux, а також для мобільних платформ iOS та Android, що робить її зручною для користувачів на різних пристроях. Інтерфейс Cryptomator простий та інтуїтивно зрозумілий, дозволяючи навіть новачкам швидко освоїти шифрування файлів та працювати з ними в безпечному середовищі.

Cryptomator дозволяє створювати зашифровані "контейнери" (директорії), які є зашифрованими файлами з розширенням .cryptomator. Програма використовує файлову систему FUSE (Filesystem in Userspace) для інтеграції з

системами, що дозволяє користувачам працювати з зашифрованими файлами, як з нормальними файлами. Це дозволяє створювати шифровані файли і взаємодіяти з ними без необхідності вручну їх дешифрувати або переглядати через сторонні інструменти. Cryptomator також підтримує автоматичне шифрування та дешифрування на основі пароля, забезпечуючи безперервний доступ до зашифрованих файлів.

Програма підтримує синхронізацію з хмарними сховищами та надає інтуїтивно зрозумілий інтерфейс, що дозволяє швидко шифрувати і дешифрувати файли. Активна спільнота користувачів допомагає підтримувати актуальність програми, регулярно випускаючи оновлення та патчі.

Інтерфейс Cryptomator наведено на рисунку 1.1.

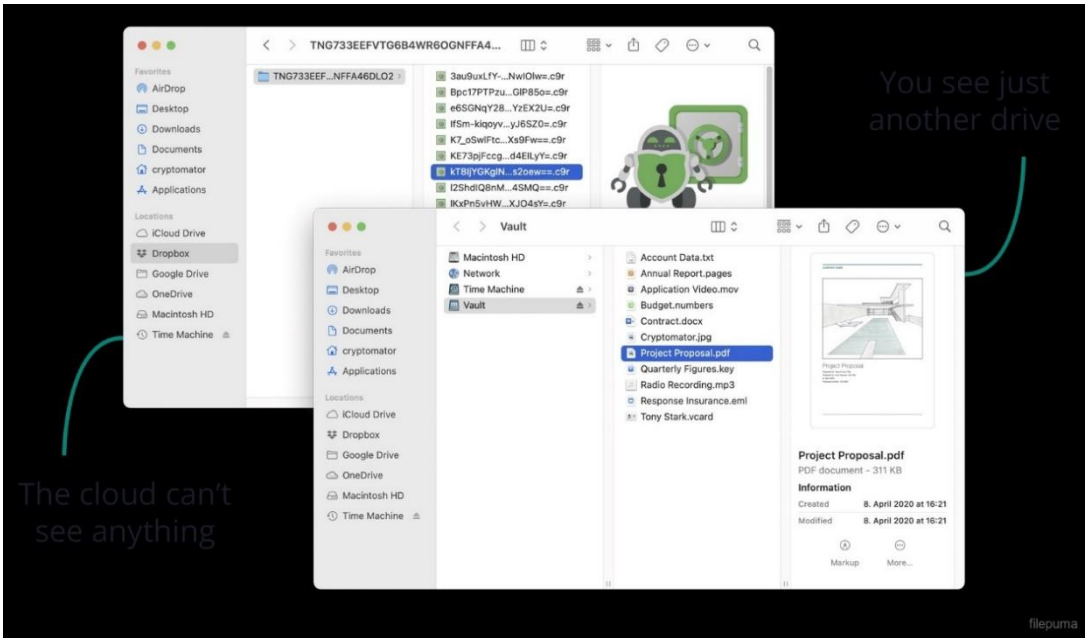


Рисунок 1.1 — Інтерфейс Cryptomator

VeraCrypt [8] – це популярний інструмент для створення зашифрованих томів та шифрування дисків. VeraCrypt підтримує шифрування на рівні диска і дозволяє створювати зашифровані контейнери, що ідеально підходять для зберігання великих обсягів даних. Одна з головних переваг VeraCrypt полягає в тому, що він пропонує підтримку декількох алгоритмів шифрування, зокрема AES, Serpent, Twofish.

Особливість VeraCrypt полягає в її здатності створювати приховані томи (внутрішні шифровані томи в середині іншого тома), що дозволяє створити додатковий рівень захисту в разі примусового доступу до шифрованих даних. Для забезпечення додаткового захисту від атак на паролі, VeraCrypt підтримує захист від атак грубою силою, використовуючи довгі паролі і подвійне шифрування. Програма дозволяє використовувати USB-ключі для автентифікації, що є додатковим рівнем безпеки, а також може працювати на апаратному рівні з TPM.

Програма також включає підтримку автоматичного стиснення даних, що дозволяє зменшити обсяг зашифрованих томів, тим самим знижуючи навантаження на систему під час доступу до зашифрованих файлів. VeraCrypt підтримує багатозадачність і багатоядерні процесори, що дозволяє ефективно працювати з великими обсягами даних.

Інтерфейс VeraCrypt є досить простим, але функціональним. Користувач може швидко створювати нові зашифровані томи, шифрувати диски або змінювати параметри існуючих томів через основне вікно програми. Більш складні параметри та настройки доступні через розширені налаштування.

Цей застосунок забезпечує високу безпеку завдяки використанню комбінацій різних алгоритмів і надає користувачам можливість настроювати параметри шифрування в залежності від вимог.

Інтерфейс VeraCrypt наведено на рисунку 1.2.

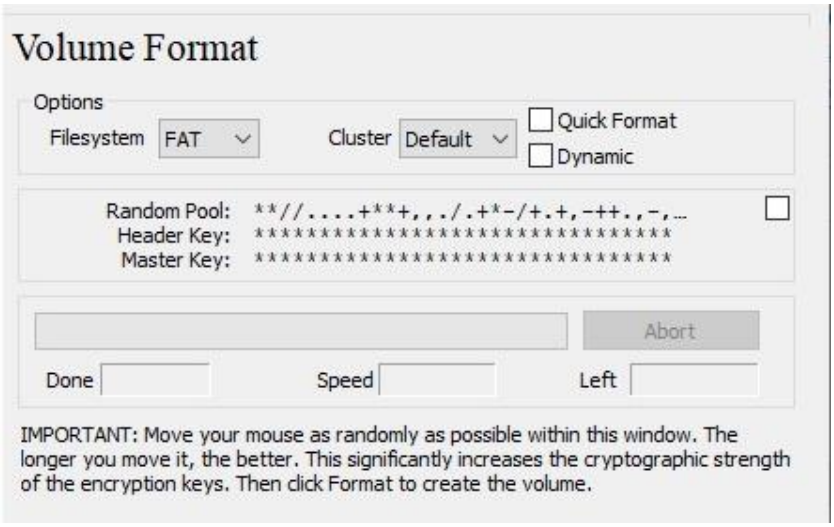


Рисунок 1.2 — Інтерфейс VeraCrypt

BitLocker [9] – це інтегрований інструмент для шифрування дисків, який надається в операційних системах Windows. BitLocker дозволяє шифрувати як системні, так і зовнішні диски, забезпечуючи надійний захист даних через шифрування на рівні апаратного забезпечення (TPM). Основною перевагою BitLocker є інтеграція в операційну систему, що дозволяє здійснювати шифрування без необхідності в додатковому програмному забезпеченні.

Програма підтримує два режими роботи: підтримка пароля для доступу до зашифрованого диска та підтримка USB-ключа (так званий "ключ відбитку"), що дозволяє додатково захистити доступ до даних. BitLocker також включає підтримку передових параметрів відновлення, таких як зашифровані копії паролів або PIN-кодів, які можуть бути використані для відновлення доступу до даних у разі втрати ключа.

BitLocker забезпечує безперервний захист даних навіть у разі викрадення або фізичної крадіжки пристрою. Для цього весь диск шифрується, включаючи операційну систему, що забезпечує максимально захищений доступ до даних. В разі втрати пристрою, дані на диску залишаються недоступними без коректного пароля або ключа.

Крім того, BitLocker орієнтований переважно на корпоративне використання, оскільки найефективніше функціонує у поєднанні з інфраструктурою Active Directory та груповими політиками, що дозволяє централізовано керувати ключами шифрування та налаштуваннями безпеки. Це значно спрощує адміністрування в організаціях із великою кількістю пристроїв. Водночас, для звичайних користувачів, які не мають доступу до подібної інфраструктури, конфігурація BitLocker може виявитися менш гнучкою або вимагати додаткових технічних знань.

На відміну від BitLocker, сучасні вебзастосунки для шифрування файлів надають більшу мобільність, оскільки не потребують встановлення програмного забезпечення та можуть бути використані на будь-якому пристрої з доступом до Інтернету. Вони забезпечують можливість захисту окремих файлів або архівів, що є зручним для щоденного обміну інформацією. Таким чином, хоча BitLocker

залишається потужним рішенням для повнодискового шифрування, його застосування у сфері захисту окремих файлів є обмеженим, що підкреслює актуальність розвитку веб-орієнтованих засобів криптографічного захисту.

Однак цей інструмент обмежений в функціональності порівняно з іншими рішеннями, такими як VeraCrypt, та підходить здебільшого для шифрування цілих дисків, а не для файлів.

Інтерфейс BitLocker наведено на рисунку 1.3.

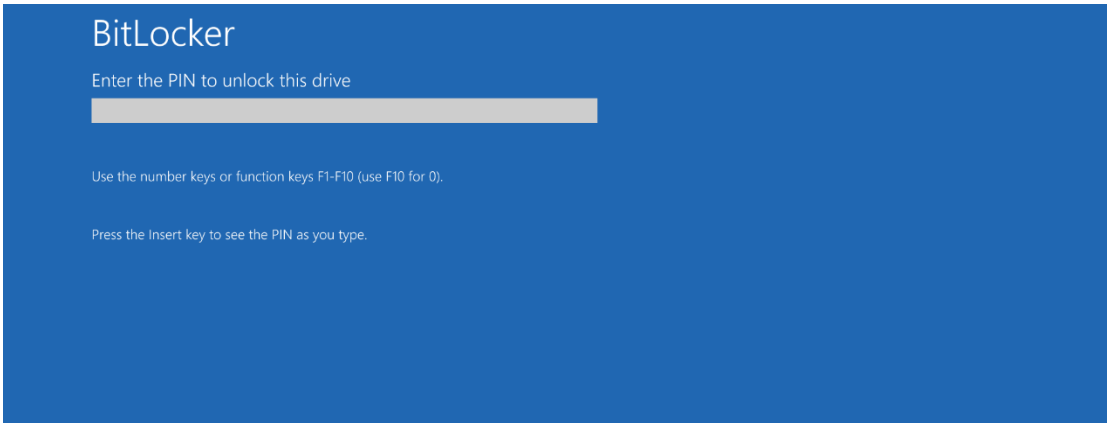


Рисунок 1.3 — Інтерфейс BitLocker

Для порівняння можливостей та функціоналу аналогів між собою, та в власною розробкою, було розроблено таблицю 1.1.

Таблиця 1.1 — Порівняльний аналіз аналогів

Характеристика	Cryptomator	VeraCrypt	BitLocker	Власна розробка
Підтримка шифрування файлів	+	+	-	+
Підтримка шифрування дисків	-	+	+	+
Гнучкість налаштувань	+	+	-	+
Інтерактивність	+	-	-	+

Характеристика	Cryptomator	VeraCrypt	BitLocker	Власна розробка
Підтримка багатокористувацького доступу	+	-	-	+
Легкість у використанні	+	-	+	+
Загальний бал	5	4	3	6

Таким чином, власна розробка має вищий загальний бал порівняно з аналогами на 20% (100-5/6*100% = 20%). Отже, власна розробка випереджає аналоги за можливостями та функціоналом, що робить її актуальною для застосування.

Крім того, розроблений вебзастосунок надає більшу адаптивність, оскільки не потребує встановлення додаткового програмного забезпечення та може працювати у будь-якому сучасному браузері. Це дозволяє інтегрувати його в різноманітні системи захисту даних, що особливо актуально для малих підприємств, віддалених команд та індивідуальних користувачів.

1.3 Аналіз методів розв’язання задачі

Класичні криптографічні алгоритми, такі як AES, RSA, та алгоритми хешування (SHA-2, SHA-3), є фундаментом сучасного криптографічного захисту даних [10]. Ці способи забезпечують високий рівень надійності завдяки формальній математичній базі, перевіреній десятиліттями досліджень і практичного застосування. Вони дозволяють реалізувати як симетричне, так і асиметричне шифрування, кожне з яких має свої особливості, переваги та обмеження. Наприклад, AES є дуже ефективним для шифрування великих обсягів

даних завдяки високій швидкості роботи, тоді як RSA підходить для захищеної передачі ключів або невеликих повідомлень завдяки своїй стійкості до атак на основі факторизації великих чисел.

У процесі вибору криптографічних засобів для веб-застосунку важливим є також питання відповідності обраного методу нормативним вимогам та міжнародним стандартам [11], таким як FIPS 140-2, ISO/IEC 27001, NIST SP 800-57 . Це особливо актуально при створенні рішень для комерційного використання, де безпека має бути не лише технічно ефективною, а й офіційно сертифікованою. Застосування сертифікованих алгоритмів, таких як AES-256 у режимі GCM або хеш-функцій SHA-3, дозволяє уникнути проблем із відповідністю вимогам безпеки та підвищує довіру до програмного забезпечення з боку користувачів і регуляторних органів.

Ще одним аспектом, який необхідно враховувати при розробці криптографічного захисту, є стійкість системи до побічних атак, зокрема до атак по сторонньому каналу (side-channel attacks), таймінгових атак (timing attacks) або атак, пов'язаних із витоками з пам'яті. При реалізації алгоритмів на стороні клієнта в JavaScript або WebAssembly варто обирати бібліотеки, які зменшують ризик таких атак завдяки використанню захищених реалізацій криптографічних примітивів. У випадку шифрування великих файлів слід звертати увагу на структуру потокової обробки даних, щоби уникнути завантаження всього обсягу в пам’ять, що може призвести до витоків або зависань.

Крім того, при розробці вебсистеми важливо передбачити можливість гнучкого управління політиками безпеки, зокрема вибору рівня шифрування, типу автентифікації та режимів обміну ключами. Це дозволяє адаптувати рішення під різні сценарії використання — від персонального захисту документів до корпоративного обміну файлами в зашифрованому вигляді. Такі функції, як генерація одноразових сесійних ключів, обмеження часу дії посилань для завантаження зашифрованих файлів, автоматичне видалення даних після обробки — сприяють підвищенню безпеки системи та зменшенню ризику витоку інформації навіть у разі компрометації облікового запису користувача.

Перевагою даних алгоритмів є їх стійкість до крипто аналітичних атак, що досягається завдяки складним математичним перетворенням, багаторівневій обробці даних та використанню криптографічно стійких ключів. Однак їх використання у вебсистемах, зокрема для шифрування великих файлів, може спричинити зниження продуктивності, особливо у випадку, якщо обробка відбувається на стороні сервера. Це може призвести до підвищеного навантаження на серверні ресурси та збільшення часу очікування для користувачів.

Другий підхід — використання гібридної криптографії — поєднує переваги симетричних і асиметричних Методів шифрування. У типовому сценарії асиметричний алгоритм (RSA або ECC) використовується для безпечного обміну симетричними ключами, після чого для шифрування файлів застосовується AES або інший симетричний алгоритм [12]. Цей підхід дозволяє одночасно забезпечити високий рівень безпеки та ефективність обробки великих обсягів інформації. Особливо актуальним він є для вебзастосунків, де важливо мінімізувати час обробки запитів і забезпечити безпечну взаємодію між користувачами та сервером.

Однак впровадження гібридної криптографії потребує додаткової уваги до управління ключами, їхнього захищеного зберігання та обміну. Неправильна реалізація може призвести до компрометації всієї системи безпеки. Саме тому важливо застосовувати стандартизовані підходи та бібліотеки, що пройшли аудит безпеки.

Сучасні вебтехнології, зокрема WebCrypto API, дозволяють реалізовувати криптографічні операції безпосередньо в браузері користувача. Це відкриває нові можливості для створення клієнт-орієнтованих вебсистем, у яких основна частина шифрування відбувається локально, без необхідності передавати нешифровані дані до сервера. Такий підхід знижує ризики перехоплення інформації під час передачі та дозволяє зменшити серверне навантаження [13].

WebCrypto API підтримує основні криптографічні алгоритми, зокрема AES-GCM , RSA-OAEP та SHA-256 , забезпечуючи достатній рівень безпеки для більшості прикладних задач. Його перевагою є висока продуктивність завдяки

використанню нативних реалізацій криптографічних операцій у браузерах, що, своєю чергою, можуть використовувати апаратне прискорення через графічні процесори або спеціалізовані криптомодулі пристрою.

З метою додаткового підвищення безпеки доцільно впроваджувати цифрові підписи та перевірку цілісності файлів за допомогою алгоритмів хешування. Наприклад, хеш-функція SHA-256 дозволяє створити унікальний цифровий відбиток файлу, який можна використовувати для виявлення змін або спроб підміни даних. Цифрові підписи на основі асиметричної криптографії дозволяють підтвердити автентичність автора та гарантувати, що дані не були змінені після підписання.

Новітні підходи до захисту даних також включають використання штучного інтелекту для виявлення підозрілої активності та аномалій у поведінці користувачів. Системи на основі машинного навчання можуть аналізувати вхідні запити, визначати типові шаблони взаємодії та оперативно виявляти потенційні загрози, такі як спроби брутфорсу, SQL-ін’єкції [14] або несанкціонований доступ. Інтеграція таких рішень у криптографічну вебсистему дозволяє забезпечити не лише захист даних, а й виявлення загроз у режимі реального часу.

Втім, застосування технологій ШІ вимагає додаткових ресурсів, а також відповідної інфраструктури для зберігання, обробки та аналізу великих обсягів даних. У процесі проєктування таких систем необхідно дотримуватись принципів «безпеки за замовчуванням» та «мінімального привілею», а також впроваджувати багаторівневу систему аутентифікації й авторизації для доступу до ключових функцій.

Отже, розробка вебсистеми для криптографічного захисту файлів є комплексною задачею, що вимагає поєднання класичних криптографічних Методів, оптимізованих архітектур вебзастосунків, інтеграції сучасних API та застосування інноваційних підходів на основі машинного навчання.

Це дозволяє забезпечити не лише високий рівень захисту, але й гнучкість, масштабованість і зручність використання вебсистеми для кінцевих користувачів. Такий підхід також сприяє підвищенню продуктивності системи, зменшенню

затримок та покращенню досвіду взаємодії з платформою, що є критично важливим фактором для сучасних вебсервісів із високими вимогами до безпеки та доступності.

1.4 Постановка задач дослідження

Провівши аналіз Методів розв’язання поставленої задачі, аналіз сучасного стану вебзастосунків для криптографічного захисту файлів, а також порівняння існуючих аналогів, було сформульовано такі основні задачі дослідження:

- розробити архітектуру вебсистеми;
- розробити користувацький інтерфейс;
- розробити модулі шифрування та дешифрування;
- розробити генерацію, збереження та перевірку криптографічних ключів;
- розробити алгоритми роботи застосунку;
- розробити функціонал застосунку;
- провести тестування розробленого застосунку.

Технічне завдання на розробку наведено в додатку А.

1.5 Висновки

У цьому розділі було проведено аналіз сучасних методів криптографічного захисту файлів у вебсистемах, здійснено огляд існуючих рішень та їх особливостей.

Визначено функціональні вимоги до власного застосунку та обґрунтовано актуальність його розробки. На основі аналізу Методів шифрування обрано гібридний підхід як оптимальний для реалізації безпечного, зручного та ефективного інструменту захисту файлів. Сформульовано перелік задач, необхідних для створення програмних модулів вебсистеми.

2 РОЗРОБКА МЕТОДІВ ТА АЛГОРИТМІВ ВЕБЗАСТОСУНКУ

2.1 Аналіз інформаційного забезпечення

Інформаційна архітектура застосунку для криптографічного захисту файлів у вебсистемі побудована на основі трьох основних потоків даних. Вхідні дані включають файли, які користувач завантажує для шифрування або дешифрування, налаштування алгоритмів (тип шифрування, ключі, параметри безпеки), а також інформацію про формат експорту захищених даних. Проміжні дані містять тимчасові версії файлів, зашифровані блоки, хеші перевірки цілісності, журнали дій користувача та метадані про процес шифрування. Вихідними даними є зашифровані файли, цифрові підписи, результати перевірки цілісності, а також збережені конфігурації проєктів. Така організація дозволяє забезпечити цілісний, керований і надійний процес обробки файлів.

З метою забезпечення повної відповідності вимогам безпеки, інформаційне забезпечення веб-застосунку включає механізми автентифікації, авторизації та контролю доступу. Користувачі проходять перевірку за допомогою пароля або криптографічного токена, після чого їм призначається рівень доступу до функцій застосунку. Для захисту авторизаційних даних використовується двофакторна автентифікація та алгоритми хешування з сіллю (наприклад, SHA-512 + PBKDF2).

Всі сесії взаємодії із системою мають обмежений час дії та автоматично анулюються при виявленні підозрілої активності.

Інтеграція з хмарними сервісами зберігання дозволяє користувачу обирати серед декількох Методів збереження зашифрованих файлів — локально, у веб-сховищі браузера або в облікових записах таких сервісів, як Google Drive або Dropbox. При цьому зашифровані файли завжди зберігаються у вигляді, недоступному для сторонніх осіб, оскільки ключі ніколи не передаються на сторонні сервери. Передбачено також підтримку захищеної синхронізації конфігурацій користувача між пристроями з використанням персональних майстер-ключів.

Для покращення аналітики та контролю ефективності функціонування системи, у застосунку реалізовано модуль моніторингу, який відстежує час

виконання криптографічних операцій, кількість оброблених файлів, частоту використання окремих алгоритмів та випадки помилок. Зібрана інформація зберігається у захищеному вигляді та може використовуватись для автоматичної оптимізації параметрів шифрування або для формування звітів про безпеку. Таким чином, інформаційне забезпечення поєднує у собі як прикладний рівень зручності, так і системний рівень надійності та масштабованості.

Для ефективної роботи з криптографічними даними застосунок підтримує роботу з низкою форматів. Внутрішні формати представлені у вигляді структурованих об’єктів з інформацією про блоки шифрування, ключі та цифрові підписи. Такі об’єкти можуть зберігатися в оперативній пам’яті або серіалізуватись у JSON або XML для тимчасового зберігання. Зовнішні формати охоплюють загальні типи файлів (PDF, DOCX, JPG, PNG, TXT, ZIP), які можуть бути зашифровані, та спеціальні формати на виході, такі як .enc або .sig для збереження зашифрованих файлів і підписів відповідно. Для налаштувань використовується JSON [15] — як простий, розширюваний формат для збереження профілів користувача та конфігурацій безпеки.

Локальне зберігання даних реалізовано за допомогою таких компонентів:

1. Сховище ключів, яке зберігає відкриті та закриті ключі користувача у захищеному вигляді з використанням майстер-ключа та додаткового шифрування.

2. Журнал активності, який містить записи про всі важливі дії користувача, включаючи зміну налаштувань, завантаження файлів, виконання шифрування/дешифрування та спроби доступу.

3. Кеш налаштувань, що зберігає останні використані параметри алгоритмів для прискорення повторних операцій.

Оптимізація використання ресурсів виконується за рахунок спеціалізованого керування пам’яттю, що дозволяє зберігати великі обсяги тимчасових даних, не перевантажуючи систему. Менеджер потоків відповідає за асинхронне виконання операцій шифрування, дешифрування та генерації ключів, мінімізуючи затримки у взаємодії з користувачем. Багаторівнева система кешування зберігає повторно використовувані криптографічні об’єкти, наприклад,

таблиці підстановки для алгоритму AES або матриці перетворень для RSA. Застосунок використовує також сучасні способи стиснення та інкрементального зберігання, аби зменшити обсяг пам’яті, зайнятої тимчасовими файлами, зокрема:

1. Адаптивне стиснення даних, що не є критичними до точності, наприклад, логів або службової інформації.

2. Використання chunked-структур для великих файлів, що дозволяє обробляти їх частинами, не завантажуючи повністю в пам’ять.

3. Оптимізація обробки блоків даних при шифруванні, що забезпечує зменшення кількості операцій читання/запису на диск.

Для підвищення надійності передбачені механізми захисту інформації та збереження результатів роботи користувача:

1. Автоматичне збереження проміжних результатів з можливістю відновлення після збоїв.

2. Система контрольних сум, яка перевіряє цілісність кожного зашифрованого файлу при збереженні та відкритті.

3. Підтримка цифрового підпису для автентифікації даних.

4. Історія змін з можливістю повернення до попередніх версій шифрування та налаштувань.

Інформаційне забезпечення розробленого застосунку для криптографічного захисту файлів являє собою збалансовану систему, що об’єднує високий рівень безпеки, гнучке налаштування та ефективне управління даними. Поєднання структурованих форматів, адаптивного кешування, захищеного зберігання ключів та оптимізованого керування потоками забезпечує стабільну роботу застосунку у реальному часі, навіть при обробці великої кількості файлів одночасно. Така архітектура дозволяє реалізувати сучасний підхід до криптографічного захисту у вебсередовищі з урахуванням потреб як кінцевих користувачів, так і вимог до безпеки та продуктивності.

2.2 Розробка методу симетричного шифрування файлів

Симетричне шифрування є одним з основних методів криптографії, де для шифрування та дешифрування використовується один і той самий ключ. Це простий та ефективний підхід, що застосовується для захисту даних від несанкціонованого доступу. У випадку шифрування файлів, дані трансформуються у зашифрований формат, який можна відновити тільки за допомогою відповідного ключа. У даній роботі розглядається розробка методу симетричного шифрування файлів на основі операцій XOR [16], з використанням генератора псевдовипадкових чисел (ПВЧ), що забезпечує високу ефективність і безпеку шифрування.

Такий підхід дозволяє реалізувати компактні та продуктивні алгоритми, придатні для веб-застосунків, де ресурси можуть бути обмеженими, а швидкість обробки даних — критичною. Генератор псевдовипадкових чисел у цьому контексті виступає як джерело ключового потоку, який комбінується з вмістом файлу за допомогою побітової операції XOR. Безпека алгоритму значною мірою залежить від якості генератора та секретності початкових параметрів (ініціалізаційного вектора або пароля).

У межах реалізації застосунку було впроваджено механізм ініціалізації ПВЧ на основі хешованого пароля користувача. Це дозволяє створити унікальний потік ключів для кожної сесії шифрування, уникаючи повторюваних шаблонів та забезпечуючи криптостійкість. Крім того, задіяна перевірка контрольної суми для кожного зашифрованого блоку, що дає змогу виявляти спроби модифікації або пошкодження даних при зберіганні чи передачі.

Завдяки своїй простоті та ефективності, симетричне шифрування на основі XOR з ПВЧ є доцільним для використання у веб-середовищі. Воно забезпечує високу швидкість обробки навіть при роботі з великими файлами, не створюючи значного навантаження на систему. При цьому правильна реалізація алгоритму, а також захищене управління ключами дозволяє гарантувати надійність захисту інформації від несанкціонованого доступу та зловмисних атак.

Симетричне шифрування ґрунтується на використанні одного секретного

ключа для обох операцій — шифрування та дешифрування. Відмінною рисою цього методу є його швидкість, оскільки алгоритм потребує мінімальних обчислювальних ресурсів. Основним компонентом в алгоритмах симетричного шифрування є операція XOR, яка дає можливість ефективно змінювати значення даних за допомогою певного ключа.

Процес симетричного шифрування можна представити наступною формулою [17]:

$$C = P \oplus K \tag{2.1}$$

де:

- C — зашифрований текст (ciphertext),
- P — відкритий текст (plaintext),
- K — ключ шифрування,
- $\oplus$ — операція XOR (виключне або).

Під час дешифрування використовується та сама операція [18]:

$$P = C \oplus K \tag{2.2}$$

оскільки XOR є симетричною операцією. Це означає, що застосування XOR до зашифрованих даних з тим самим ключем відновлює початкові (відкриті) дані.

У симетричних методах шифрування важливим етапом є генерація гами шифрування, яка є послідовністю псевдовипадкових чисел, що використовуються для модифікації вихідних даних.

Для генерації гами можна використовувати лінійний конгруентний генератор (LCG) , який визначає нове значення в послідовності за такою рекурентною формулою [19]:

$$T(i+1) = (A \cdot T(i) + C) \bmod M \tag{2.3}$$

де:

- T(i) — значення на і-тій кроці генерації,
- A — множник,
- C — зсув,
- M — модуль, що визначає максимальний період генератора.

Генератор ПВЧ створює послідовність чисел, яка може бути використана як гама для шифрування.

Основні етапи шифрування файлу на основі методу гамування включають такі кроки:

Вибір ключа та параметрів генератора: Ключ шифрування обирається випадковим чином, наприклад, 128-бітний або 256-бітний ключ, що забезпечує достатній рівень безпеки. Параметри генератора ПВЧ A, C, та M повинні бути вибрані таким чином, щоб період генерації чисел був максимальним, а послідовність не мала явних закономірностей.

Ініціалізація генератора: Початкове значення генератора T0 можна ініціалізувати на основі ключа шифрування. Наприклад, можна використати перші біти ключа як початкове значення для генератора. Це дозволяє створити унікальну послідовність для кожного файлу чи сеансу шифрування.

Генерація гами шифрування: Після ініціалізації генератора починається генерація псевдовипадкових чисел, які будуть використовуватися для шифрування. Для кожного байта даних генерується псевдовипадкове число з гами, яке комбінується з відкритими даними за допомогою операції XOR [20]:

$$C(i) = P(i) \oplus T(i) \tag{2.4}$$

де:

- C(i) — зашифрований байт,
- P(i) — байт відкритих даних,
- T(i) — байт з псевдовипадкової гами.
- Запис зашифрованих даних. Після шифрування кожного байта дані записуються у новий файл, який містить зашифровану версію оригінального файлу.

Для забезпечення безпеки шифрування необхідно, щоб генератор ПВЧ мав великий період і був важким для передбачення. Псевдовипадкові числа, що генеруються таким чином, повинні бути настільки ж випадковими, щоб зловмисники не могли вгадати значення гами, навіть маючи доступ до зашифрованого файлу.

Загальні властивості такого підходу до шифрування:

Стійкість до криптографічних атакі ,підвищення періоду генератора зменшує можливість криптоаналізу.

Ефективність, алгоритм шифрування має високу швидкість виконання завдяки простоті операції XOR.

Безпека, використання довгих випадкових послідовностей, згенерованих ПВЧ, підвищує рівень безпеки, але вимагає безпечного зберігання ключа.

2.3 Розробка методу генерації та використання ключів

Розробка методу генерації та використання ключів є основою для забезпечення безпеки в криптографічних системах. Ключі — це секретні параметри, які використовуються для виконання операцій шифрування та дешифрування даних. Від їх безпеки залежить успішність захисту інформації від несанкціонованого доступу.

Однак цей тип генераторів має свої обмеження, зокрема, можна передбачити послідовності чисел, якщо зловмисник знає значення A, C і M. Для підвищення безпеки можна комбінувати декілька генераторів або використовувати додаткові способи, такі як хеш-функції.

Хеш-функції використовуються для створення ключів з даних будь-якої довжини, перетворюючи їх на фіксовану довжину. Наприклад, хеш-функція SHA-256 генерує 256-бітні хеші, що є достатньо надійними для забезпечення конфіденційності. Для обчислення хешу використовується така формула [21]:

$$H = SHA256\ (D) \tag{2.5}$$

- де H — результат хешування,

- D — вхідні дані, які можуть бути будь-якої довжини.

Оскільки хеш-функції є односторонніми, неможливо відновити вихідні дані на основі хешу. Це робить хеш-функції дуже корисними при створенні ключів для криптографічних операцій.

Для забезпечення більш високого рівня випадковості та безпеки можна використовувати фізичні джерела випадковості, такі як шум у електронних

компонентах або інші випадкові процеси. Використання фізичних джерел дозволяє отримувати істинно випадкові числа, що робить їх набагато більш надійними порівняно з псевдовипадковими генераторами.

Способи генерації ключів на основі ПВГ або хеш-функцій можуть бути вразливими до атак, що базуються на аналізі цих послідовностей. Тому важливо використовувати додаткові техніки для підвищення безпеки. Наприклад, можна використовувати протоколи обміну ключами, такі як протокол Діффі-Геллмана, який дозволяє двом сторонам безпечно обмінюватися ключами через незахищені канали. Протокол ґрунтується на складних математичних операціях, що робить неможливим для зловмисника відновити приватний ключ.

Вибір розміру ключа є важливим аспектом при розробці криптографічних систем. Чим більший розмір ключа, тим важче здійснити перебір усіх можливих варіантів. Наприклад, для 128-бітного ключа кількість можливих варіантів обчислюється за формулою [22]:

$$N=2^k \quad (2.6)$$

де

- N — кількість можливих варіантів,

- k — довжина ключа в бітах.

Для 128-бітного ключа це буде 2^128, що становить величезну кількість можливих варіантів, що робить атаку методом перебору практично неможливою. Проте з розвитком обчислювальних потужностей, криптографи рекомендують використовувати ключі більшої довжини, такі як 256 біт.

Важливим аспектом є також регулярне оновлення ключів. Це дозволяє забезпечити безпеку навіть у разі компрометації одного з ключів. Процес оновлення ключів може здійснюватися на основі певних алгоритмів або може бути здійснено вручну користувачами, в залежності від вимог безпеки.

Для зберігання ключів існують різні способи, включаючи використання апаратних засобів безпеки, таких як смарт-карти або криптографічні токени. Ці пристрої дозволяють зберігати ключі в захищеному вигляді і забезпечують доступ до них тільки в разі авторизації користувача. Крім того, існують способи

шифрування самих ключів, щоб забезпечити їх конфіденційність навіть у разі їх втрати.

Зберігання та передача ключів є важливою частиною системи безпеки. Використання протоколів, таких як протокол Діффі-Геллмана [23], дозволяє здійснити безпечний обмін ключами через незахищений канал зв'язку. У цьому випадку, навіть якщо зловмисник перехопить передану інформацію, він не зможе відновити ключі без знання додаткових даних. Це дозволяє забезпечити високу стійкість криптосистеми до атак.

У майбутньому, з розвитком квантових обчислень, традиційні способи генерації та використання ключів можуть стати уразливими. Тому вже зараз доцільно почати використовувати алгоритми, які будуть стійкими до атак за допомогою квантових комп'ютерів. Одним з таких підходів є використання криптографічних алгоритмів на основі решеток, які, як очікується, будуть стійкими до квантових атак.

Таким чином, розробка надійного методу генерації, зберігання та використання ключів є критично важливою складовою будь-якої криптографічної системи. Застосування псевдовипадкових генераторів, хеш-функцій, протоколів обміну та сучасних апаратних засобів дозволяє досягти високого рівня безпеки при збереженні продуктивності. У поєднанні з регулярною ротацією ключів та перспективними алгоритмами, стійкими до квантових атак, такі способи формують надійний фундамент захисту інформації в умовах зростаючих кіберзагроз.

2.4 Розробка алгоритмів роботи вебзастосунку

Алгоритм гамування оптимізує використання обчислювальних ресурсів при шифруванні даних шляхом диференційованого розподілу довжини та складності гамми залежно від характеристик вхідного повідомлення. Замість того, щоб застосовувати фіксовану гамму до всього повідомлення, метод адаптивно керує генерацією псевдовипадкової послідовності на основі трьох ключових факторів важливості блоку даних, ентропії та повторюваності змісту.

Для кожного блоку даних обчислюється композитний показник криптографічної складності, який визначає оптимальну довжину та оновлення гамми, сформованої за допомогою генератора псевдовипадкових чисел (ПВЧ). Блоки з вищою ймовірністю атак (наприклад, часто повторювані шаблони або ключові структурні елементи) обробляються з використанням посиленої гамми, що ускладнює криптоаналіз. Водночас, менш критичні блоки шифруються стандартною гамою, що зменшує навантаження на систему без значної втрати безпеки.

Алгоритм також застосовує динамічні криптографічні перетворення, які забезпечують узгодженість між блоками з різною інтенсивністю шифрування, зберігаючи цілісність структури даних і ускладнюючи їхнє відновлення сторонніми особами. Важливі елементи даних додатково зміщуються у псевдовипадковому порядку, що унеможливлює просте визначення структури навіть після часткового дешифрування.

При шифруванні – дешифруванні файлів повинна виконуватись така послідовність дій:

- вводиться ім’я файлу (len).
- M вибираємо рівним найближче M=2n > довжини файлу.
- вводиться 160-бітний ключ шифру.
- обчислюється вихідне значення генератора ПВЧ у відповідності з виразом:

$$T1 = (1001\ T0 + 1333) \bmod M \qquad (2.7)$$

- зчитується перший символ з файлу.
- знаходимо значення зашифрованого символу як сума по модулю “2”

Обчислюється наступне вихідне значення генератора ПВЧ у відповідності з виразом:

$$Tn = (1001\ T(n-1) + 1333) \bmod M \qquad (2.8)$$

- зчитується наступний символ з файлу.
- знаходимо значення зашифрованого символу як сума по модулю “2” Tn і введеного символу:

$$ЗС = СИМВОЛ + Tn \qquad (2.9)$$

і записуємо його в вихідний файл.

- повторюємо 2 попередніх пункти до тих пір поки не будуть зашифровані всі символи файлу.

Блок-схема алгоритму адаптивного наведена на рисунку 2.1.

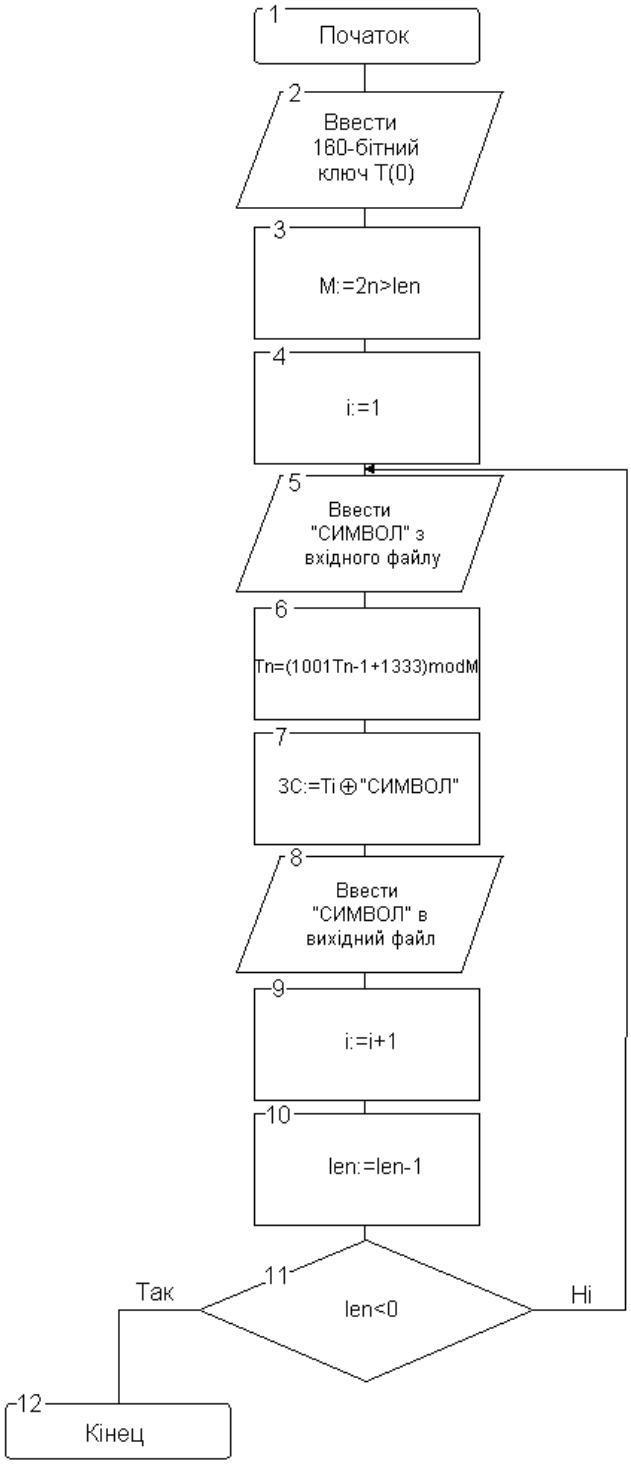


Рисунок 2.1 — Блок-схема алгоритму гамування

Для кожного блоку визначаються статистичні характеристики: частота повторення символів, рівень ентропії, наявність структурованих шаблонів. Ці характеристики є вхідними параметрами для функції оцінки важливості, яка формує вагу блоку.

Залежно від цієї ваги визначається параметр складності для генерації гамми: наприклад, кількість ітерацій генерації ПВЧ, зміна початкового значення (seed), або частота оновлення гамми протягом шифрування блоку. Таким чином, більш «вразливі» блоки отримують складніший і менш передбачуваний потік, що істотно підвищує стійкість до атак зі статистичним чи лінійним аналізом.

Для ще точнішої адаптації складності, в алгоритмі реалізована система зворотного зв’язку: під час шифрування кожного блоку відстежуються ключові метрики, такі як середній рівень зміни бітів, розподіл ентропії в часі та наявність збігів з попередніми блоками. У випадку виявлення потенційного зниження ентропії система автоматично збільшує складність генерації гамми для наступних блоків, запобігаючи формуванню передбачуваних патернів.

Крім того, застосовується контекстно-залежна ініціалізація генератора: для кожного файлу або сеансу шифрування початкове значення (seed) формується на основі хешу ключа користувача, дати створення файлу, а також випадкової домішки. Це ускладнює використання атак повторного шифрування та підвищує унікальність кожної сесії навіть при використанні однакового паролю.

Для додаткового посилення криптостійкості реалізовано багаторівневу генерацію гамми. Основна послідовність ПВЧ проходить через кілька фаз перетворення, де кожен рівень включає окрему функцію мішання або фільтрації, зокрема побітову ротацію, інверсію та нелінійне зміщення. Такий підхід створює складну та нестабільну структуру гамми, яка унеможливлює ефективний криптоаналіз навіть при частковому доступі до відкритого тексту.

Особливу увагу в алгоритмі приділено вибору ПВЧ. У реалізації застосовано комбінований генератор, що базується на з'єднанні лінійного конгруентного генератора з криптографічно стійким алгоритмом зворотного зв’язку. Це дозволяє уникнути коротких періодів, підвищує ентропію вихідної послідовності та знижує

ймовірність успішного відновлення гамми.

Шифрування реалізується за допомогою операції XOR між байтами вхідного повідомлення та згенерованої гамми. Завдяки цьому забезпечується симетричність шифру: розшифрування виконується повторним застосуванням тієї ж гамми до зашифрованого повідомлення. Такий підхід дозволяє реалізувати швидкий і легкий у реалізації алгоритм, придатний для вбудованих систем, мобільних застосунків або веб-клієнтів.

Перевага адаптивного гамування полягає в балансі між безпекою і продуктивністю. У реальних сценаріях, де ресурси системи обмежені (наприклад, у мобільних пристроях або браузерах), повне шифрування з максимальною складністю може бути надлишковим. Адаптивний підхід дозволяє сконцентрувати ресурси на найбільш критичних ділянках даних, зменшуючи загальні витрати без компромісу щодо безпеки.

Крім того, адаптивна схема є більш стійкою до атак аналізу, оскільки структура шифру постійно змінюється в залежності від змісту повідомлення. Такий алгоритм важче формалізувати, змоделювати або зламати стандартними методами криптоаналізу.

2.5 Висновки

У цьому розділі було розроблено адаптивний алгоритм гамування з використанням генератора псевдовипадкових чисел, який забезпечує ефективне та гнучке шифрування даних. Застосування адаптивного підходу дозволяє оптимізувати використання обчислювальних ресурсів та підвищити криптостійкість системи за рахунок диференційованої генерації гамми. Розроблений алгоритм легко масштабується, має високу продуктивність і може бути використаний у різноманітних програмних рішеннях..

3 РОЗРОБКА ЗАСТОСУНКУ

3.1 Варіантний аналіз та вибір мови програмування розробки

Для реалізації криптографічного застосунку, що використовує гамування з використанням псевдовипадкових чисел, важливим кроком є вибір мови програмування. Основні критерії вибору включають: підтримку сучасних криптографічних бібліотек, простоту реалізації алгоритмів, продуктивність, кросплатформеність та можливість інтеграції з веб-інтерфейсом.

Розглянемо кілька мов програмування, які теоретично могли б бути використані: Python, C++, Java, а також обґрунтуємо вибір JavaScript як основної мови розробки.

JavaScript [24]— динамічна, інтерпретована мова програмування, яка на сьогодні є стандартом для розробки веб-застосунків. Основною перевагою JavaScript у контексті криптографічного застосунку є його глибока інтеграція з браузером та можливість реалізовувати весь функціонал без встановлення стороннього ПЗ. Крім того, JS підтримує криптографічні API (наприклад, Web Crypto API), дозволяючи реалізовувати алгоритми шифрування, генерацію випадкових чисел, хешування тощо безпосередньо на стороні клієнта. Завдяки великій екосистемі бібліотек і сумісності з HTML/CSS для створення інтерфейсу, JavaScript є ідеальним вибором для створення сучасного, зручного й безпечного криптографічного застосунку.

Python [25] — високорівнева мова програмування з динамічною типізацією, яка добре підходить для реалізації криптографічних алгоритмів завдяки таким бібліотекам, як pycryptodome, cryptography та інші. Однак Python більше орієнтований на серверний бік або на створення CLI-додатків. У випадку розробки веборієнтованого застосунку використання Python вимагає створення бекенду та обслуговування серверної частини, що не відповідає завданню максимальної автономності та клієнтської реалізації.

C++ [26] — мова програмування з високою продуктивністю, що забезпечує гнучке управління пам’яттю та низькорівневий доступ до системних ресурсів. Її

часто використовують для реалізації криптографічних бібліотек на низькому рівні. Проте складність розробки, менша зручність у створенні інтерфейсу користувача та відсутність прямої інтеграції з веб-середовищем робить її менш придатною для даного типу застосунку.

Java [27] — компільована, об’єктно-орієнтована мова, яка добре підходить для розробки безпечних і платформонезалежних застосунків. Вона має розвинені бібліотеки для криптографії (javax.crypto) та багатопотоковості. Водночас Java вимагає встановлення JVM на клієнтській машині, а її інтеграція з веб-інтерфейсом значно складніша, ніж у JavaScript.

Зведене порівняння характеристик мов представлено в таблиці 3.1.

Таблиця 3.1 — Порівняльний аналіз мов програмування для криптографічного застосунку

Характеристика	JavaScript	Python	++	ava
Підтримка криптобібліотек	+	+		
Робота в браузері	+	-		
Простота реалізації алгоритмів	+	+		
Кросплатформеність	+	+		
Інтеграція з інтерфейсом користувача	+	±		
Не потребує встановлення	+	-		
Сумарний бал	6	4		

На підставі проведеного аналізу можна зробити висновок, що JavaScript є найбільш доцільною мовою програмування для реалізації клієнтського криптографічного застосунку, що використовує метод гамування. Вона забезпечує всі необхідні функції, дозволяє уникнути зайвого навантаження на сервер, гарантує просту та швидку взаємодію з користувачем через веб-інтерфейс, а також має підтримку сучасних засобів безпеки безпосередньо в браузері.

3.2 Розробка головного модуля застосунку

Головний модуль застосунку являє собою вікно, в якому розміщено інтерфейс для вибору файлу, введення пароля, а також виконання операцій шифрування та дешифрування. Він забезпечує зручну взаємодію з користувачем і не потребує спеціальних навичок для керування.

Крім того, вікно містить індикатори статусу, які відображають поточний стан операції, забезпечуючи користувача інформацією про хід процесу шифрування або дешифрування, що робить використання застосунку ще більш інтуїтивно зрозумілим і зручним.

Також у головному вікні реалізовано можливість попереднього перегляду вибраного файлу, що дозволяє користувачу переконатися в правильності вибору перед початком операцій шифрування чи дешифрування. Додано функцію автоматичної перевірки відповідності формату файлу та попередження у разі невідповідності. Для зручності було впроваджено підтримку перетягування файлів у робочу область, що робить процес взаємодії ще швидшим і комфортнішим для користувача.

Розроблено структурну схему вікна головної сторінки вебсайту, яка наведена на рисунку 3.1.

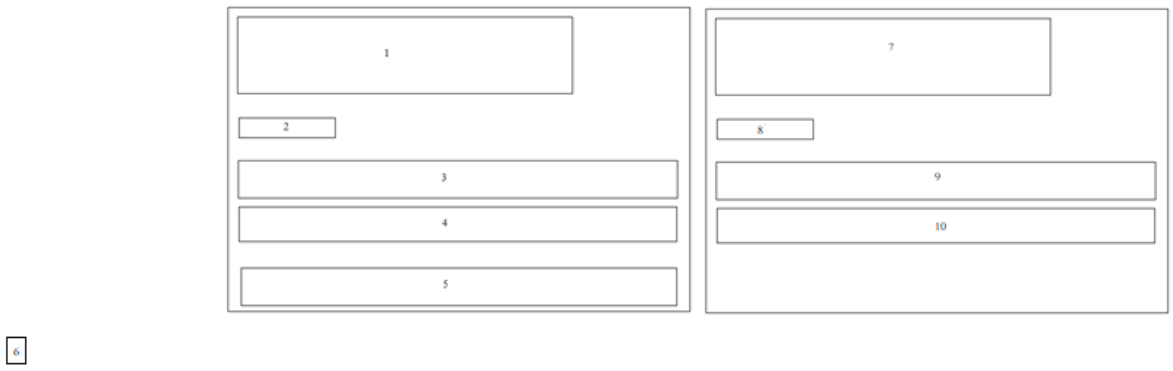


Рисунок 3.1 — Структурна схема вікна головного модуля застосунку

Складові елементи інтерфейсу вебсайту:

1. Підпис.
2. Кнопка вибору файлів.
3. Кнопка “Шифрувати файл”

4. Кнопка “Дешифрувати файл”

5. Поле виводу десяткового коду.

6. Іконка “Запитання”.

7. Підпис.

8. Кнопка вибору жорсткого диску.

9. Кнопка “Шифрувати диск”

10. Кнопка “Дешифрувати диск”

Розроблено структурну схему кнопки “Допомога” (рис 3.2), яка є частиною інтерфейсу користувача програми для шифрування файлів. Дана кнопка дозволяє користувачу отримати базову інформацію про програму, а також контактні дані розробника у випадку виникнення труднощів або запитань. Інтерфейс реалізовано таким чином, щоб користування програмою було інтуїтивно зрозумілим навіть для недосвідченого користувача.

У вікні, що відкривається при натисканні кнопки “Допомога”, міститься контактна інформація, за якою користувач може звернутися за консультацією або технічною підтримкою. Також передбачена кнопка для закриття цього вікна, що дозволяє швидко повернутися до основного функціоналу застосунку. Вікно допомоги не блокує інші дії в інтерфейсі і працює в окремому режимі, забезпечуючи зручність користування.

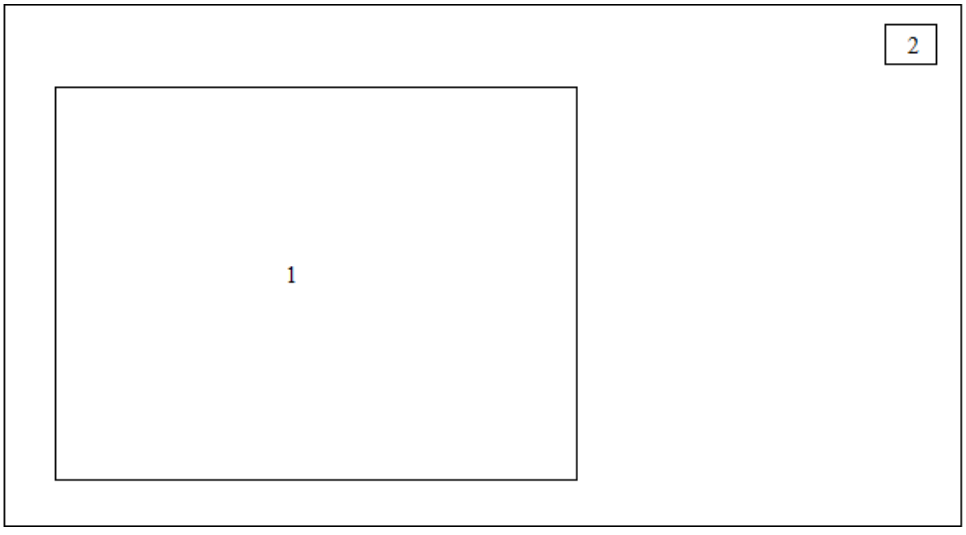


Рисунок 3.2 — Структурна схему кнопки “Допомога”

Складові елементу “Допомога”:

1. Контактна інформація.

2. Кнопка закрити вікно.

Блок-схема алгоритму роботи головного модуля вебсайту наведена на рисунку 3.3. Вона демонструє послідовність дій користувача під час вибору файлу, введення паролю, виконання шифрування або дешифрування з використанням алгоритму XOR, а також обробки результатів. Така схема дозволяє краще зрозуміти внутрішню логіку застосунку та взаємозв’язок його програмних компонентів.

Рисунок 3.3 – Блок-схема алгоритму роботи головного модуля вебсайту

Блок-схема також враховує можливість обробки помилок, таких як неправильне введення пароля або спроба обробити файл без вказаного шляху. На

кожному етапі реалізовані перевірки, що забезпечують стійкість роботи системи та мінімізують ймовірність збоїв під час виконання основних операцій. Це дозволяє користувачеві своєчасно отримати зворотний зв’язок і скоригувати дії без втрати даних.

Крім того, у схемі відображено логіку збереження результатів після завершення шифрування або дешифрування. Застосунок пропонує користувачеві вибрати шлях для збереження обробленого файлу або автоматично створює файл у відповідній директорії з інформативною назвою. Такий підхід підвищує зручність користування та дозволяє впорядкувати результати роботи із зашифрованими файлами

3.3 Розробка клієнтської частини та інтерфейсу користувача

Розробка клієнтської частини криптографічного застосунку є одним із ключових етапів у реалізації всієї системи, оскільки саме цей компонент відповідає за взаємодію з кінцевим користувачем. Його завданням є не лише забезпечити функціональний доступ до операцій шифрування та дешифрування, але й гарантувати простоту, надійність і безпечність використання. З огляду на це, до проєктування клієнтської частини висуваються як функціональні, так і нефункціональні вимоги.

Клієнтська частина реалізована з використанням мови JavaScript, що забезпечує високу гнучкість та інтеграцію з HTML/CSS для створення інтуїтивно зрозумілого інтерфейсу користувача. Основна перевага такого підходу — можливість повністю автономної роботи застосунку в браузері без необхідності встановлення додаткового програмного забезпечення чи підключення до серверної частини. Це також дозволяє досягти високого рівня безпеки, оскільки всі криптографічні операції відбуваються на стороні клієнта, без передавання незашифрованих даних у мережу.

У початковій частині скрипту відбувається перевірка, чи було завантажено файл користувачем. Для цього використовується перевірка наявності ключа 'file' у глобальному масиві $_FILES. Якщо файл не надійшов, сервер повертає помилку з

кодом 400 (некоректний запит) та повідомлення у форматі JSON, після чого припиняє подальше виконання скрипту. Це забезпечує контроль цілісності вхідних даних і унеможливлює обробку запитів без файлу (рис. 3.4).

```php
<?php
// Перевіряємо, чи прийшов файл
if (!isset($_FILES['file'])) {
    http_response_code(400);
    echo json_encode(['error' => 'Файл не завантажено']);
    exit;
}
```

Рисунок 3.4 – Блок перевірки файлу

Далі з POST-запиту приймаються додаткові параметри, такі як пароль (password), параметр disk та числові значення A, C, M, T, які можуть використовуватися для налаштувань алгоритму шифрування або генерації псевдовипадкових чисел. Якщо якийсь параметр відсутній у запиті, скрипт використовує значення за замовчуванням — порожній рядок для текстових і нуль для числових параметрів. Числові значення перетворюються у тип integer за допомогою функції intval(), що дозволяє запобігти передачі некоректних типів даних (рис. 3.5).

```php
// Приймаємо параметри з POST
$password = $_POST['password'] ?? '';
$disk = $_POST['disk'] ?? '';
$A = intval($_POST['A'] ?? 0);
$C = intval($_POST['C'] ?? 0);
$M = intval($_POST['M'] ?? 0);
$T = intval($_POST['T'] ?? 0);
```

Рисунок 3.5 – Блок параметри з POST-запиту

Перед збереженням файлу перевіряється наявність директорії uploads/, куди будуть поміщені завантажені файли. Якщо така папка відсутня, вона створюється з правами доступу 0755, що дозволяє власнику записувати файли, а іншим користувачам лише читати і виконувати (рис. 3.6). Це гарантує, що операція

збереження файлу пройде успішно, оскільки PHP не може перемістити файл у неіснуючу директорію.

```php
// Зберігаємо файл у тимчасову папку (для подальшої обробки)
$uploadDir = 'uploads/';
if (!is_dir($uploadDir)) mkdir($uploadDir, 0755, true);
```

Рисунок 3.6 – Блок директорія для збереження файлів

Для збереження файлу використовується функція move_uploaded_file(), яка безпечно переносить файл із тимчасового каталогу в папку uploads/. Ім’я файлу витягується за допомогою basename(), що запобігає потенційним атакам через маніпуляції з шляхом. Якщо файл успішно переміщено, скрипт переходить до наступних кроків, інакше повертає помилку сервера 500 та повідомлення про неможливість збереження файлу (рис. 3.7).

```php
$fileTmpPath = $_FILES['file']['tmp_name'];
$fileName = basename($_FILES['file']['name']);
$destination = $uploadDir . $fileName;

if (move_uploaded_file($fileTmpPath, $destination)) {
    // Логування отриманих даних
    $logLine = date('Y-m-d H:i:s') . " | Файл: $fileName | Пароль: $password | Диск: $disk | A:$A C:$C M:$M T:$T\n";
    file_put_contents('logs.txt', $logLine, FILE_APPEND);

    // Тут можна додати обробку файлу – зашифрувати його, викликати інші скрипти тощо.

    echo json_encode([
        'status' => 'success',
        'message' => 'Файл завантажено і параметри прийнято',
        'file' => $fileName
    ]);
} else {
    http_response_code(500);
    echo json_encode(['error' => 'Помилка збереження файлу']);
}
```

Рисунок 3.7 – Блок збереження файлу

Після успішного збереження здійснюється логування операції в файл logs.txt. У нього записується дата і час операції, ім’я файлу, пароль, параметр disk та числові налаштування A, C, M, T. Запис дописується в кінець файлу, що дозволяє вести хронологію дій користувачів та полегшує налагодження та аудит системи (рис. 3.8).

```
echo json_encode([
    'status' => 'success',
    'message' => 'Файл завантажено і параметри прийнято',
    'file' => $fileName
]);
```

Рисунок 3.8 – Блок відправлення клієнту відповіді у форматі JSON

На завершення скрипт повертає у форматі JSON повідомлення про успішне завантаження файлу та прийняття параметрів. Ця відповідь може бути оброблена на клієнтській стороні для відображення користувачу підтвердження та подальшої роботи з файлом, наприклад, запуску алгоритмів шифрування або дешифрування (рис. 3.9).

```
echo json_encode([
    'status' => 'success',
    'message' => 'Файл завантажено і параметри прийнято',
    'file' => $fileName
]);
```

Рисунок 3.9 – Блок успішного виконання

Якщо файл не вдається зберегти, скрипт повертає помилку сервера 500 і відповідний JSON-відповідь, що дає змогу клієнту правильно обробити ситуацію (рис 3.10).

```
} else {
    http_response_code(500);
    echo json_encode(['error' => 'Помилка збереження файлу']);
}
?>
```

Рисунок 3.10 – Блок помилка

Інтерфейс користувача побудовано на основі адаптивної верстки, що дозволяє забезпечити коректне відображення та зручність користування як на десктопних пристроях, так і на мобільних. Основні елементи інтерфейсу включають форму вибору файлу, поле введення пароля, кнопки для запуску

процесу шифрування та дешифрування, а також інформативні повідомлення про успішне завершення операцій або виникнення помилок. Усі дії супроводжуються візуальним зворотним зв’язком, що допомагає користувачеві зрозуміти поточний стан системи.

Особливу увагу приділено реалізації зручного механізму вибору файлів, який забезпечує підтримку як текстових, так і бінарних форматів. Це реалізовано за допомогою об’єкта FileReader API, який дозволяє зчитувати вміст обраного файлу та перетворювати його у формат, придатний для обробки криптографічним алгоритмом. Крім того, інтерфейс дозволяє переглянути ім’я вибраного файлу, його розмір та тип, що допомагає уникнути помилок при виборі.

При натисканні на кнопку «Зашифрувати» або «Розшифрувати», запускається основна логіка програми, що включає генерацію псевдовипадкової послідовності (на основі введеного пароля та внутрішніх параметрів) і виконання побітової операції XOR з вмістом файлу. Результат операції пропонується користувачу для збереження у вигляді нового файлу, який можна завантажити одним кліком. Для цього використовується Blob API, що дозволяє створити об’єкт файлу в пам’яті браузера та викликати завантаження без залучення серверної частини.

Важливою частиною інтерфейсу є реалізація повідомлень про успішність чи помилки під час шифрування. Для цього використовуються модальні вікна або блоки повідомлень, які з’являються після завершення операції. Наприклад, у випадку помилки пароля або спроби розшифрувати файл, який не був зашифрований цим застосунком, користувач отримає зрозуміле повідомлення із рекомендаціями щодо подальших дій. Це підвищує загальний рівень зручності та зменшує кількість помилок при використанні системи.

Інтерфейс також враховує нефункціональні вимоги, такі як доступність та інтуїтивність. Кнопки та поля мають логічне розташування, кольори контрастні й добре читаються. Для осіб з вадами зору передбачено можливість масштабування шрифтів та керування за допомогою клавіатури. Це відповідає сучасним стандартам доступності в розробці веб-застосунків.

З технічної точки зору, структура клієнтського коду поділена на логічні блоки: обробка подій, логіка шифрування, генерація гамми, обробка файлів та оновлення інтерфейсу. Такий підхід дозволяє спростити обслуговування проєкту, а також полегшує майбутнє масштабування або зміну окремих компонентів. Всі функції задокументовані й забезпечені перевіркою типів вхідних параметрів, що знижує ймовірність появи помилок у процесі виконання.

Окремо варто зазначити важливість продуктивності клієнтської частини. У випадках обробки великих файлів або складної логіки генерації гамми можливі затримки у виконанні. Для уникнення проблем із «заморожуванням» інтерфейсу реалізовано асинхронне виконання основних обчислювальних процесів. Наприклад, при запуску шифрування основна обробка відбувається у фоновому потоці, а на екрані з’являється індикатор прогресу, який дозволяє користувачу оцінити приблизний час виконання.

Крім основних функціональних елементів, клієнтська частина має систему налаштувань, де користувач може змінити параметри шифрування, такі як тип генератора ПВЧ, довжина ключа, метод адаптації до типу даних, що шифруються. Це робить застосунок гнучким і придатним як для звичайного користування, так і для експериментів у навчальному або дослідницькому середовищі.

У майбутньому планується додавання темної теми інтерфейсу, підтримки перетягування файлів у вікно застосунку (drag-and-drop), а також інтеграція з браузерним сховищем для тимчасового збереження ключів (із підтвердженням користувача). Такі оновлення зроблять клієнтську частину ще зручнішою та функціональнішою, не знижуючи при цьому рівень безпеки.

Таким чином, клієнтська частина криптографічного застосунку є не лише інтерфейсом, але й повноцінним функціональним ядром системи, що забезпечує повний цикл обробки даних — від завантаження вхідного файлу до отримання зашифрованого або розшифрованого результату. Вона є прикладом поєднання сучасних технологій фронтенду з криптографічними принципами безпеки.

3.4 Висновки

У цьому розділі було розроблено застосунок для шифрування та дешифрування файлів із використанням симетричного алгоритму [28]. Для реалізації програми було обрано мову програмування JavaScript, яка дозволила створити зручний графічний інтерфейс із доступом до файлової системи.

Розроблено головний модуль, що забезпечує взаємодію користувача з додатком: вибір файлів, введення пароля та виконання основних криптографічних операцій.

4 ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Аналіз методів тестування

Тестування застосунку є завершальним, але надзвичайно важливим етапом життєвого циклу розробки будь-якого програмного продукту. Саме на цьому етапі визначається відповідність роботи застосунку початковим вимогам технічного завдання, виявляються можливі помилки або недоліки, перевіряється правильність реакцій системи на дії користувача, а також оцінюється коректність виконання основних алгоритмів, які лежать в основі роботи застосунку. Особливої важливості тестування набуває у випадках розробки систем, що працюють з обробкою критичних або чутливих даних, зокрема криптографічних систем. Будь-яка помилка в алгоритмах або логіці роботи таких систем може призвести до серйозних порушень безпеки, втрати даних або компрометації конфіденційної інформації.

У контексті розробки вебзастосунку для криптографічного захисту файлів, що використовує гамування на основі псевдовипадкових чисел, тестування має кілька критично важливих цілей: перевірка коректності роботи основних криптографічних механізмів, забезпечення безпеки обробки даних, перевірка відповідності стандартам криптографічної стійкості, а також гарантування зручності та безпечності взаємодії користувача із системою.

Модульне тестування [29] є одним з основних підходів для забезпечення якості програмного забезпечення, що використовує алгоритми для обробки даних, і передбачає ізольовану перевірку функціональності окремих компонентів системи.

Для кожного модуля розробляються окремі юніт-тести, які перевіряють правильність виконання базових операцій: генерацію однаково відтворюваних послідовностей для однакових ініціалізаційних параметрів, відповідність шифрованих вихідних даних очікуваним результатам, а також правильність дешифрування без втрати даних. Також важливо перевірити обробку неочікуваних ситуацій, наприклад, введення порожніх даних, неправильних ключів або

некоректних форматів файлів.

Для криптографічного застосунку найбільше уваги потребують модулі, що безпосередньо займаються шифруванням, генерацією ключів, а також реалізацією алгоритмів гамування. Тестування таких модулів дозволяє перевірити правильність математичних обчислень, що виконуються під час шифрування та дешифрування, а також відповідність програмних реалізацій теоретичним моделям. Важливою частиною є перевірка граничних випадків, наприклад, коли використовуються дуже короткі чи довгі повідомлення, чи коли застосовуються спеціальні символи, які можуть вплинути на точність шифрування. Це дає змогу виявити помилки округлення чи неточності, які можуть мати серйозний вплив на безпеку і коректність роботи системи.

Інтеграційне тестування дозволяє перевірити правильність взаємодії між різними компонентами застосунку. У випадку криптографічного застосунку це включає перевірку правильної передачі даних між компонентами системи, що займаються шифруванням та дешифруванням, а також взаємодії між інтерфейсом користувача і алгоритмами обробки даних. Основною метою є забезпечити коректну взаємодію всіх компонентів системи, а також уникнути проблем з передачею даних, наприклад, у разі паралельної обробки на сервері або при використанні складних алгоритмів шифрування. Тестування також охоплює перевірку коректності збереження та відновлення зашифрованих даних, а також перевірку, чи не відбуваються витоки даних або їх спотворення під час передачі між різними компонентами застосунку.

Тестування графічного інтерфейсу користувача є важливою частиною верифікації застосунку, особливо якщо він передбачає активну взаємодію з користувачем. Для криптографічного застосунку це включає перевірку коректності відображення елементів керування, таких як кнопки для шифрування та дешифрування, поля для введення ключів і повідомлень, а також коректність відображення результатів після виконання операцій. Тестування інтерфейсу може здійснюватися як вручну, так і автоматизовано за допомогою спеціальних інструментів для перевірки відображення елементів керування та взаємодії з

ними.

При цьому важливо перевірити інтуїтивність інтерфейсу, а також його здатність правильно реагувати на некоректні або неповні дані, що вводяться користувачем. Враховуючи, що криптографічні операції можуть вимагати деякого часу на виконання, важливо перевірити стабільність інтерфейсу при тривалих обчисленнях, а також наявність індикаторів прогресу, які спрощують користувачеві взаємодію із застосунком.

Додатково слід проводити тестування безпеки (security testing), що передбачає перевірку захищеності даних під час їх обробки та зберігання, тестування правильності роботи механізмів автентифікації та авторизації, а також перевірку застосування кращих практик безпечної обробки введених користувачем даних (наприклад, запобігання ін’єкційним атакам, міжсайтовому скриптінгу XSS, CSRF-атакам тощо).

Тестування продуктивності (performance testing) допомагає оцінити швидкодію застосунку при різних обсягах оброблюваних даних, що є критично важливим для систем шифрування великих файлів. Під час тестування проводиться вимірювання часу шифрування і дешифрування залежно від розміру файлу, кількості паралельних запитів, обчислювальної потужності клієнтських пристроїв, а також тестується стабільність роботи при тривалому безперервному навантаженні.

Функціональне тестування має за мету перевірити правильність реалізації всіх ключових можливостей системи. У випадку криптографічного застосунку це передбачає перевірку роботи основних функцій шифрування та дешифрування, генерації ключів, а також перевірку коректності результатів при різних налаштуваннях параметрів. Особливу увагу слід приділяти перевірці граничних випадків, таких як обробка надзвичайно великих або малих за розміром повідомлень, а також тестуванню алгоритмів при роботі з різними типами даних (текстові файли, зображення, тощо). Крім того, слід перевірити правильність збереження та відновлення шифрованих даних, а також стабільність і надійність при роботі з великими обсягами інформації. Для кожної функціональної

можливості доцільно розробити набір тестових сценаріїв, що охоплюють різні можливі комбінації вхідних даних, щоб переконатися в коректності реалізації алгоритмів шифрування в усіх умовах.

4.2 Тестування застосунку

Представлена програма призначена для шифрування та дешифрування даних за допомогою криптографічних алгоритмів, зокрема алгоритму на основі операції XOR. Вона забезпечує простий і зручний інтерфейс для користувача, що не потребує спеціальних навичок для керування.

Програма орієнтована на широку аудиторію користувачів, зокрема на студентів, дослідників і фахівців, які працюють із захистом інформації. Завдяки інтуїтивному інтерфейсу та швидкому доступу до основних функцій застосунок підходить як для початківців, так і для досвідчених користувачів. Використання операції XOR для шифрування дозволяє досягти високої швидкості обробки файлів без значного навантаження на системні ресурси.

Програма підтримує базову валідацію вхідних даних, що запобігає спробам обробки некоректних або пошкоджених файлів. У разі виявлення помилки система відображає відповідне повідомлення, що підвищує зручність використання та запобігає збоїм у роботі. Такий підхід дозволяє підвищити загальну надійність застосунку і забезпечити стабільну взаємодію користувача з інтерфейсом.

Щоб запустити вебсайт на виконання, потрібно вибрати файл index.html, який знаходиться у папці Kursova. Після запуску програми відкриється сайт, де користувач може вибрати будь-який файл для подальших операцій. Головна сторінка (рис. 4.1) дозволяє вибирати файли через стандартний файловий діалог. Для цього достатньо двічі клікнути на файл.

Додатково головна сторінка надає користувачеві короткі інструкції щодо подальших дій. Інтерфейс виконаний у мінімалістичному стилі, що дозволяє зосередити увагу саме на виконанні основних операцій без зайвих відволікаючих елементів. Дизайн сторінки оптимізовано для коректного відображення на різних типах пристроїв: комп’ютерах, планшетах та мобільних телефонах.

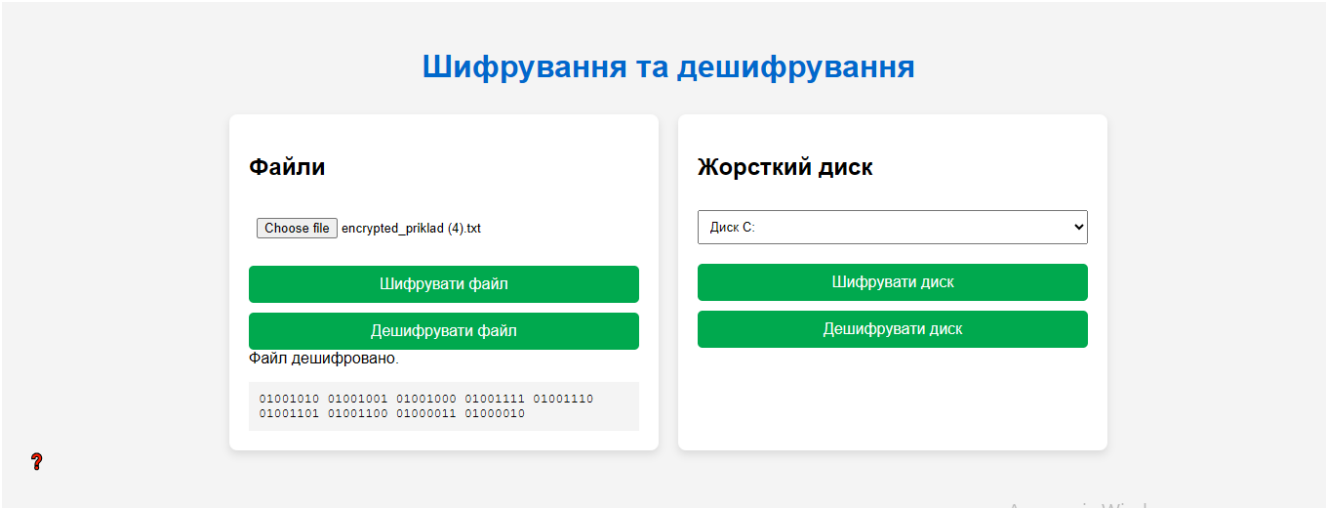


Рисунок 4.1 – Головна сторінка вебсайту

Щоб зашифрувати файл, необхідно вибрати потрібний файл . Для цього достатньо вибрати файл і натиснути кнопку "Шифрувати файл" на вкладці "Файл" вибрати команду "Шифрувати файл" (рис. 4.2).

Після активації шифрування відбувається обробка вмісту обраного файлу з використанням введеного пароля як основи для побудови послідовності бітів, що накладаються на вихідний файл. Це гарантує, що навіть у разі однакового вихідного файлу різні паролі забезпечуватимуть різні результати шифрування. Такий підхід підвищує стійкість системи до атак перебору.

У процесі шифрування застосовується алгоритм XOR, який поєднує байти вмісту файлу з байтами згенерованої ключової послідовності. Завдяки цьому забезпечується двосторонність алгоритму — тобто та сама операція може бути використана як для шифрування, так і для дешифрування. Це спрощує реалізацію системи та зменшує ризик помилок при обробці даних, забезпечуючи при цьому високий рівень безпеки.

Після завершення процесу шифрування користувач отримує повідомлення про успішне збереження зашифрованого файлу, який можна завантажити або передати іншим способом. У випадку дешифрування система перевіряє правильність пароля, і якщо він вірний — відновлює початковий вміст файлу.

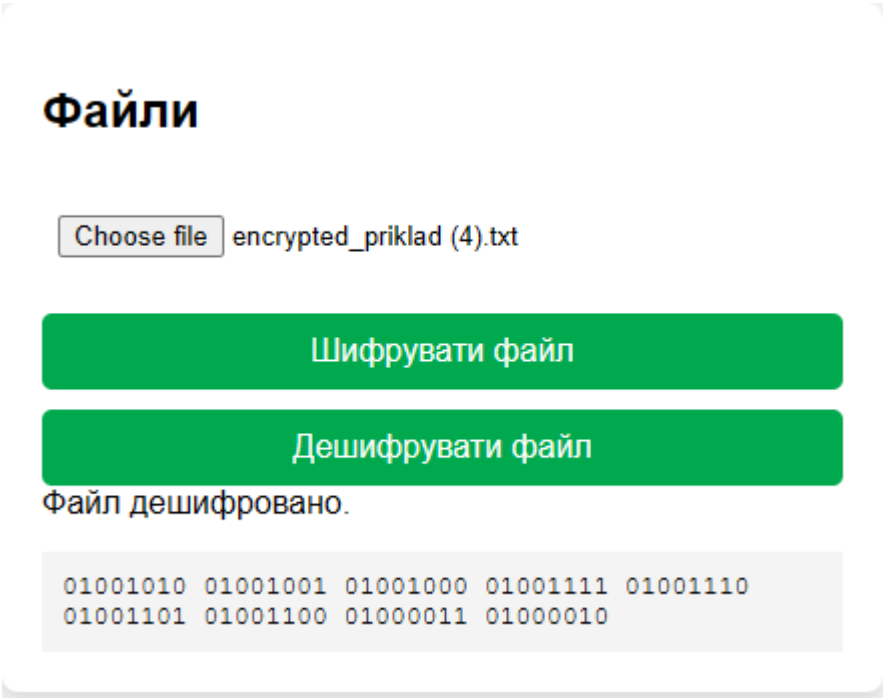


Рисунок 4.2 – Шифрування вибраного файлу

Після цього завантажиться зашифрований файл з закінченням .enc , знизу під кнопками буде виводитись десятковий код зашифрованого файлу(рис 4.3).

Десятковий код, що генерується після шифрування, служить додатковою інформацією для користувача. Він дозволяє впевнитися в тому, що дані були оброблені та зашифровані належним чином. Цей код може також використовуватися для базової верифікації результатів шифрування, якщо потрібно перевірити правильність обробки без розшифрування файлу.

Рисунок 4.3 – Результат шифрування

Для того, щоб дешифрувати файл, необхідно вибрати зашифрований файл. Після вибору файлу потрібно натискати кнопку "Дешифрувати файл", після цього завантажиться файл котрий буде мати початковий стан (рис 4.4).

Рисунок 4.4 – Результат дешифрування

Щоб зашифрувати диск, необхідно ввести правильний ключ-шифр (рис 4.5) доступу до системи. Це запобігає несанкціонованому доступу до функціоналу шифрування накопичувача.

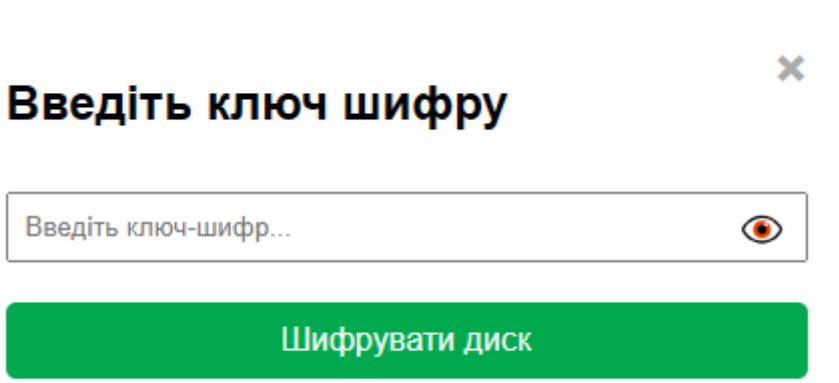


Рисунок 4.5 – Введення ключ-шифру

Після успішної автентифікації потрібно натиснути кнопку "Шифрувати диск" на вкладці "Диск", після чого система розпочне процес шифрування усіх даних на віртуальному диску (рис. 4.6).

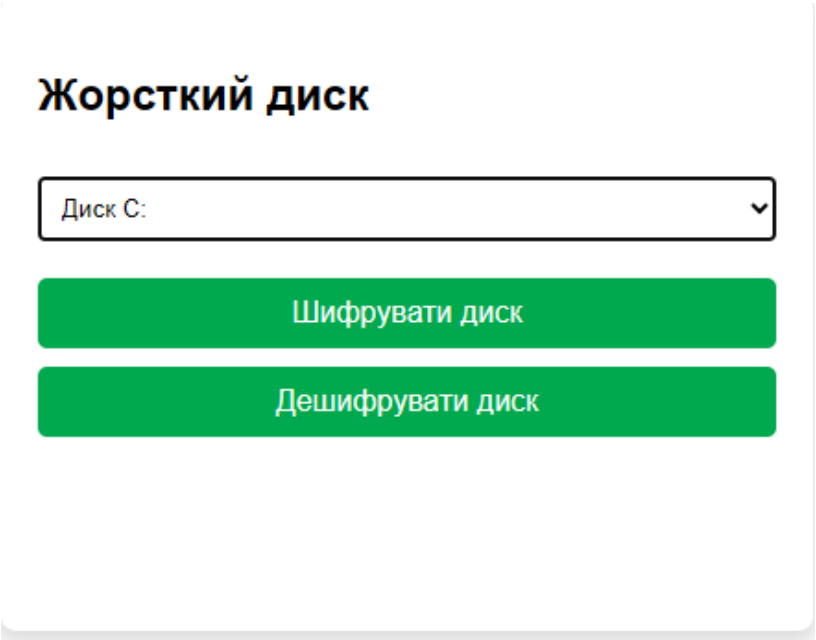


Рисунок 4.6 – Процес шифрування диску

Шифрування реалізовано за допомогою побітової операції XOR, де кожен байт даних обробляється з використанням ключа, згенерованого на основі введеного користувачем пароля. Це забезпечує базову конфіденційність при мінімальних витратах ресурсів, що підходить для вебреалізації.

Після завершення шифрування результат відображається безпосередньо в інтерфейсі у вигляді двійкового коду зашифрованого диску (рис. 4.7). Користувач може скопіювати цей код вручну для подальшого збереження або перевірки.

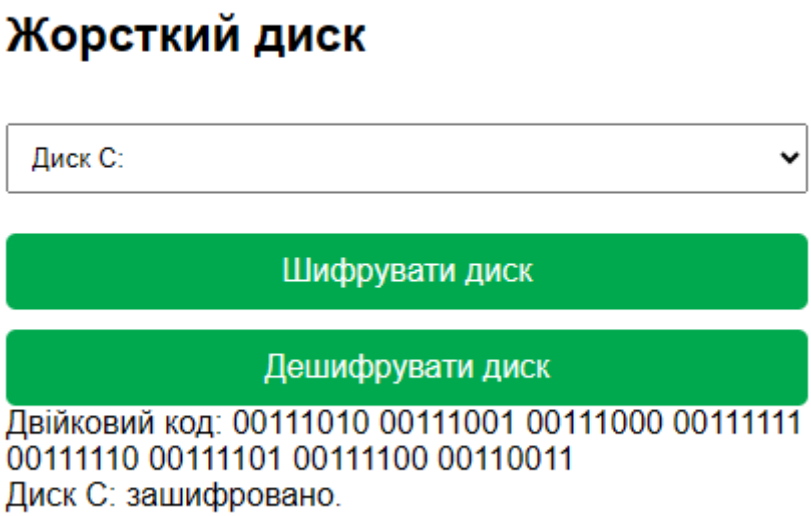


Рисунок 4.7 – Контрольний код після шифрування диску

На поточному етапі система не передбачає автоматичне збереження зашифрованого образу диску у вигляді окремого файлу. Цей функціонал може бути реалізовано на подальших етапах розробки, наприклад, через створення файлу Blob на клієнті або передачу зашифрованих даних на сервер для збереження.

Для дешифрування користувач повинен натиснути кнопку "Дешифрувати диск". У результаті на екран виводиться відновлений вміст диску та його двійковий код(рис. 4.8).

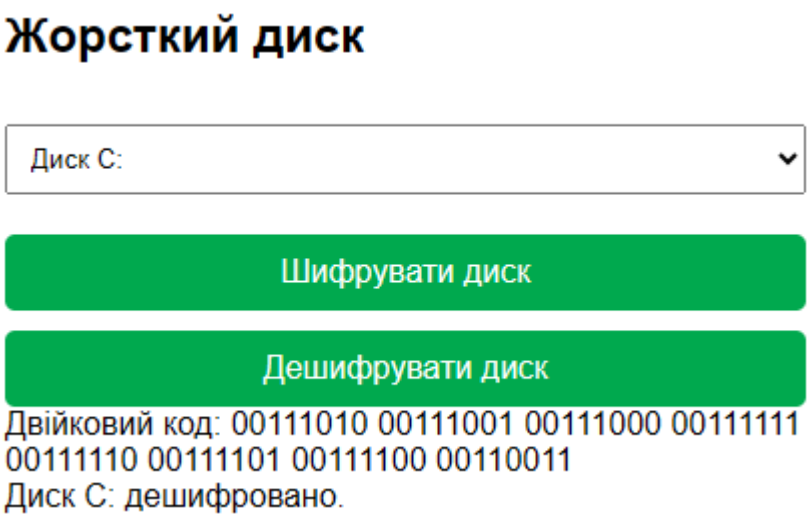


Рисунок 4.8 – Результат дешифрування диску

У даній системі для шифрування файлів та емульованого образу диску використовується генератор псевдовипадкових чисел (ГПВЧ). Алгоритм формує ключову псевдовипадкову послідовність, яка використовується для XOR-шифрування вхідних байтів даних. Кожне значення згенерованої послідовності XOR-иться з відповідним байтом даних, забезпечуючи шифрування (рис. 4.9).

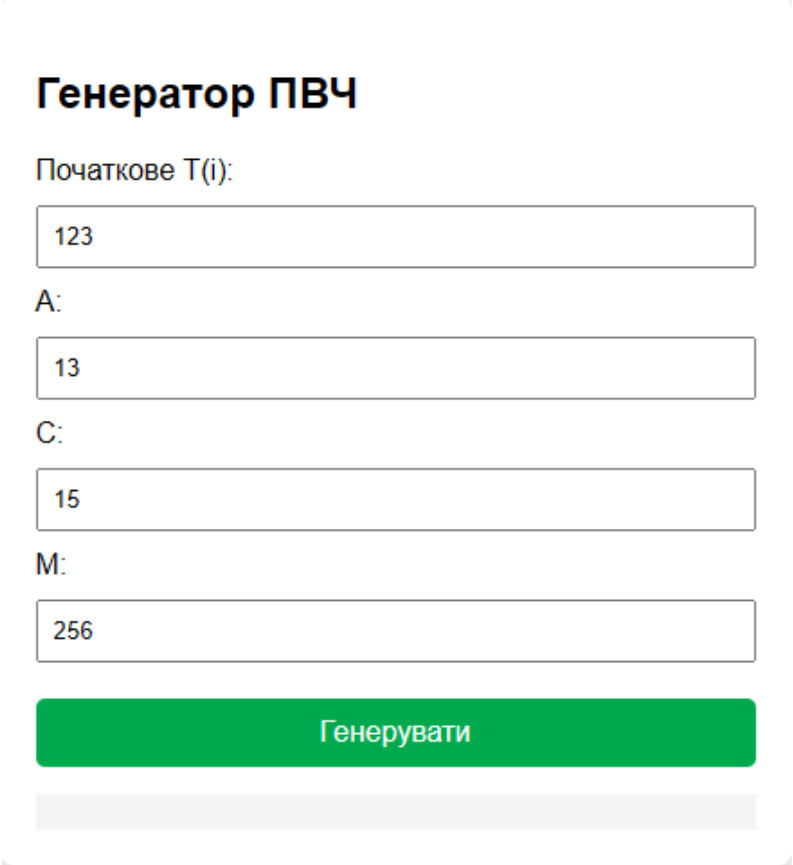


Рисунок 4.9 – Генератор псевдо випадкових чисел

Таким чином, ГПВЧ виступає центральним елементом симетричної криптосистеми, де стійкість до злому залежить від складності відгадування початкових параметрів генерації (пароля та констант A, C, M), а також коректної реалізації XOR-алгоритму.

При натисканні кнопки «Генерувати ГПВЧ» система виконує обчислення псевдовипадкової послідовності чисел за лінійно-конгруентним методом, використовуючи введені користувачем параметри:

- початкове значення T(0),
- множник A,
- доданок C,
- модуль M.

На основі отриманої послідовності T(i) виконується побітова операція XOR з фіксованим масивом байтів P, що імітує вхідні дані. В результаті формується нова послідовність C(i), яка відображається на екрані у вигляді списку чисел (рис 4.10):

C(i): 123, 87, 44, ...

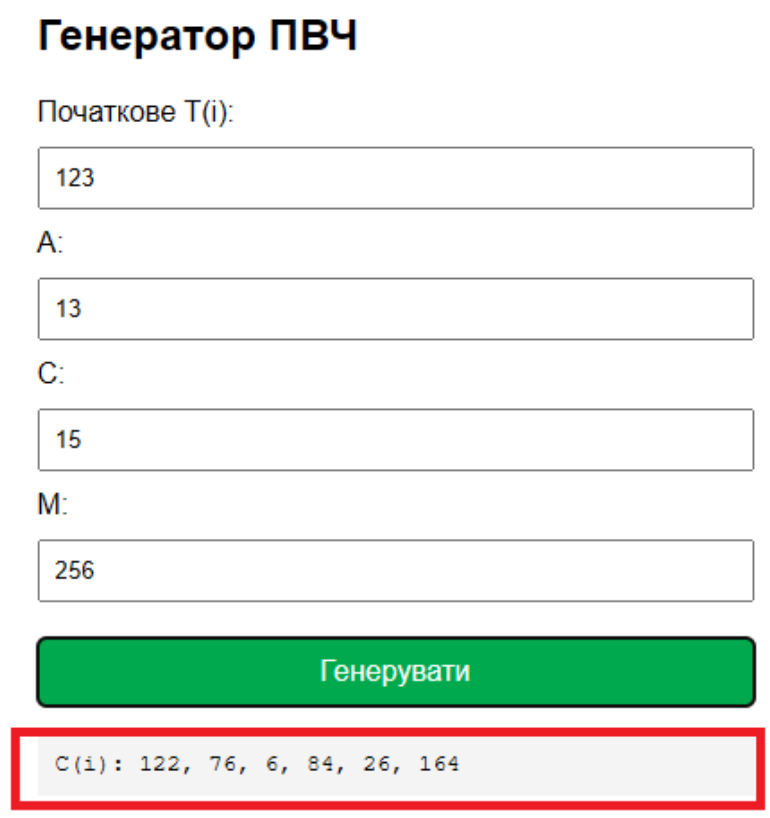


Рисунок 4.10 – Результат генерації ПВЧ

Ці значення є зашифрованими результатами XOR-операції між початковими байтами та псевдовипадковою послідовністю. Вони можуть бути використані як ключова послідовність для подальшого шифрування або дешифрування даних. Таким чином, користувач отримує наочне уявлення про те, як працює генератор псевдовипадкових чисел та як формується шифрувальний потік.

При натисканні на кнопку "Допомога", з’являється діалогове вікно, в котрому написанні усі данні автора та як з ним можна зв’язатись (рис 4.11).

Наявність функції "Допомога" значно покращує користувацький досвід, оскільки дозволяє швидко отримати контактну інформацію для консультацій або повідомлень про можливі помилки у роботі застосунку. Додатково це підвищує рівень довіри до програми, оскільки користувач бачить відкритість розробника та можливість комунікації у разі потреби.

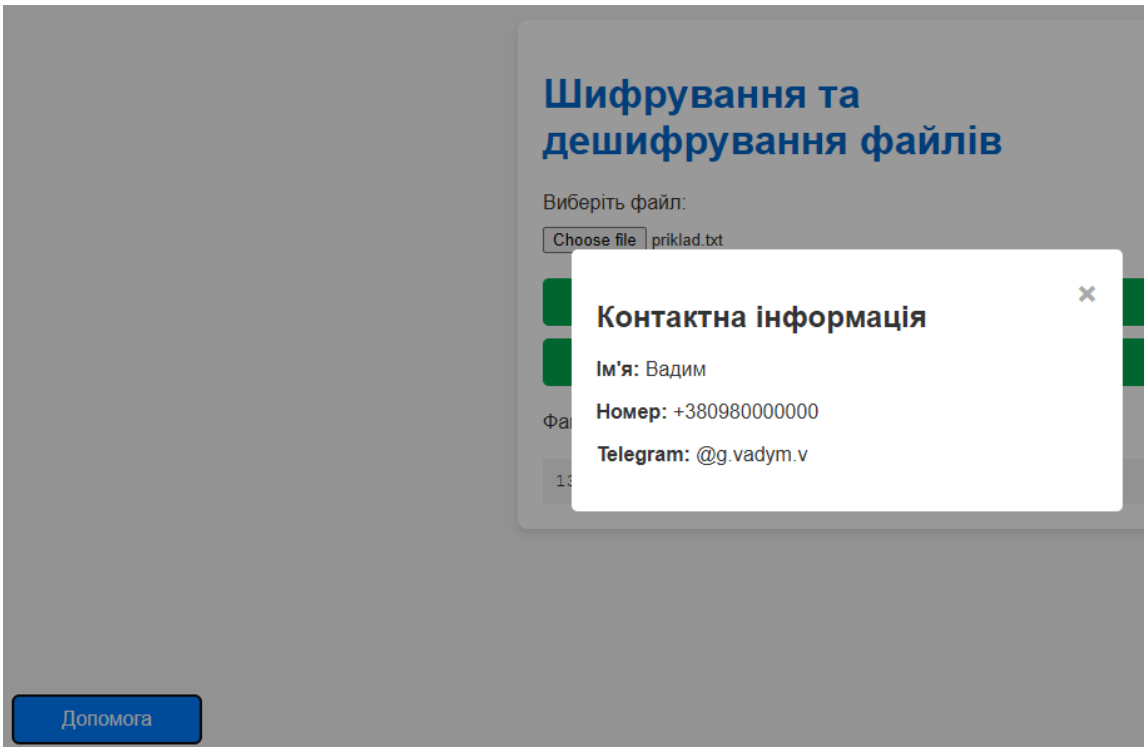


Рисунок 4.11 – Діалогове вікно "Допомога"

Діалогове вікно "Допомога" має просту і зрозумілу структуру: воно містить короткий опис призначення застосунку, контактні дані розробника та інструкції щодо повідомлення про знайдені помилки. Завдяки цьому користувачі можуть

оперативно отримати необхідну інформацію без зайвих зусиль

4.3 Висновки

У цьому розділі було проведено аналіз Методів тестування застосунку для шифрування на основі гамування з використанням ПВЧ. Тестування включало в себе перевірку коректності роботи розробленого алгоритму шифрування та дешифрування, а також аналіз його криптостійкості. Для цього були проведені модульні та інтеграційні тести, що забезпечили перевірку всіх функціональних можливостей застосунку, зокрема процесу шифрування тадешифрування даних [30].

ВИСНОВКИ

У результаті виконання бакалаврської роботи було розроблено вебзастосунок для захисту інформації шляхом шифрування з використанням методу гамування на основі генератора псевдовипадкових чисел. Розроблений застосунок дозволяє ефективно здійснювати процеси шифрування та дешифрування даних, використовуючи генератори ПВЧ для формування ключової послідовності, забезпечуючи при цьому достатній рівень криптостійкості. Система дозволяє працювати з різними типами вхідних даних, налаштовувати ключі, зберігати зашифровані результати в окремі файли для подальшого використання або передачі, а також проводити зворотне дешифрування для перевірки цілісності даних.

Для розробки було використано мову програмування JavaScript а також бібліотеки, що забезпечують генерацію випадкових чисел, побудову графічного інтерфейсу користувача, обробку файлів і реалізацію алгоритмів шифрування.

Зокрема, для реалізації серверної логіки та обробки запитів було використано Node.js, а для графічної частини — HTML, CSS та JavaScript, що забезпечили зручний інтерфейс взаємодії користувача із системою.

Окрім того, реалізовано систему перевірки ключових послідовностей на предмет повторення та циклічності, що критично важливо при побудові надійної криптосистеми.

У першому розділі було проведено аналіз сучасного стану проблеми криптографічного захисту інформації, зокрема Методів симетричного шифрування та застосування гамування в сучасних ІТ-системах. Розглянуто існуючі підходи до побудови криптографічних алгоритмів, які базуються на використанні псевдовипадкових генераторів, наведено приклади систем-аналоги. Проведено порівняльний аналіз і визначено переваги застосування власної розробки, що базується на використанні простих, але надійних підходів до побудови ключових послідовностей.

У другому розділі було проведено аналіз інформаційного забезпечення

криптографічної системи, обґрунтовано вибір методу гамування як ефективного способу шифрування з використанням ПВЧ. Описано побудову генератора псевдовипадкових чисел, алгоритм формування ключової послідовності, а також механізм XOR-перетворення вхідних даних для забезпечення шифрування і дешифрування. Окремо розглянуто особливості зберігання ключів та файлів з шифрованою інформацією.

У третьому розділі було проведено порівняльний аналіз мов програмування та обрано мову JavaScript з огляду на її широкі можливості для реалізації криптографічних алгоритмів, доступність спеціалізованих бібліотек, простоту в обробці файлів і побудові зручного інтерфейсу користувача. Розроблено основні модулі програмного застосунку, включно з модулем генерації псевдовипадкової послідовності, шифрування/дешифрування та модулем візуального представлення ключів і даних. Продемонстровано структуру інтерфейсу, наведено схеми роботи окремих елементів застосунку.

У четвертому розділі було проведено аналіз методів тестування застосунку, зокрема модульного та функціонального тестування криптографічних процедур.

Проведено перевірку коректності роботи алгоритмів шифрування і дешифрування, тестування працездатності генератора ПВЧ, а також перевірку адекватності реакції системи на некоректні вхідні дані або помилкові ключі.

Окрему увагу приділено тестуванню нових функцій, зокрема візуалізації побітових перетворень, відображення бінарного вигляду ключової послідовності, перевірки пароля перед шифруванням файлів та дисків, а також коректності виводу повідомлень при спробі доступу без авторизації. Доведено, що застосунок повністю виконує всі задачі, поставлені на початку розробки, працює стабільно, надійно обробляє дані, демонструє передбачувану поведінку в нестандартних ситуаціях і забезпечує достатній рівень захисту при шифруванні.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Безпека даних при захисті файлів. [Електронний ресурс]. URL: https://it-solutions.ua/blog/bezpeka-v-epohu-haosu-informatsijna-oborona-dlya-kompanij/

2. Майданюк В. П. Обробка сигналів. Навчальний посібник. - Вінниця: ВНТУ. Режим доступу: https://iq.vntu.edu.ua/method/getfile.php?fname 63704.pdf&x=1&card_id=7303&id=63704

3. В. П. Майданюк , В.В. Гиренко. Веб-застосунку для криптографічного захисту файлів. у IV Міжнародній науково-практичнійконференції «Інформаційні технології в освіті та науці». [Електронний ресурс]. URL: https://mdpu.org.ua/nauka/konferentsiyi-ta-seminary/

4. Криптографія шифрування. [Електронний ресурс]. URL: https://iit.com.ua/n

5. Сучасні алгоритми шифрування. [Електронний ресурс]. URL: https://www.kingston.com/ua/blog/data-security/what-is-encryption

6. WebAssembly . [Електронний ресурс]. URL: https://webassembly.org/

7. Cryptomator . [Електронний ресурс]. URL: https://cryptomator.org/

8. VeraCrypt. [Електронний ресурс]. URL: https://www.veracrypt.fr/code/VeraCrypt/

9. BitLocker. [Електронний ресурс]. URL: https://support.microsoft.com/uk-ua/windows

10. Класичні криптографічні алгоритми. [Електронний ресурс]. URL: https://hostpro.ua/wiki/ua/security/encryption-types-algorithms

11. Нормативи вимогам. [Електронний ресурс]. URL: https://www.iaasb.org/publications/2018-0

12. Симетричний алгоритм шифрування. [Електронний ресурс]. URL: https://sites.google.com/view/blog-ua

13. Що таке серверне навантаження. [Електронний ресурс]. URL: https://www.ukraine.com.ua/wiki/hosting/overload/web-server/

14. SQL-ін’єкції. [Електронний ресурс]. URL: https://foxminded.ua/sql-

iniektsii/

15. JSON. [Електронний ресурс]. URL: https://www.ibm.com/docs/en/db2-for-zos/12.0.0?topic=sql-json-val

16. Що таке XOR. [Електронний ресурс]. URL: https://hostpro.ua/wiki/ua/security/encryption-types-algorithms/

17. Формула симетричного шифрування. [Електронний ресурс]. URL: https://scribd.com/document/

18. Формула генерації XOR. [Електронний ресурс]. URL: https://studfile.net/preview/8951240/page:18/

19. Формула генерація гами . [Електронний ресурс]. URL: https://support.microsoft.com/uk-ua/office/xor

20. Формула методів генерації ключі . [Електронний ресурс]. URL: https://dou.ua/forums/topic/43026/

21. Формула обчислення хешу. [Електронний ресурс]. URL: https://habr.com/ru/articles/534596/

22. Формула обчислення 128-бітного . [Електронний ресурс]. URL: https://www.vpnunlimited.com/ua/help/cybersecurity/128-bit-encryption

23. Протокол Диффі-Геллман . [Електронний ресурс]. URL: https://conf.ztu.edu.ua/wp-content/uploads/2024/01/120-1.pdf

24. Аттіла Ковач. Modern Software Testing Techniques: A Practical Guide for Developers and Testers. Нью-Йорк: Apress, 2024. 266с.

25. Brian Okken. Python Testing with pytest: Simple, Rapid, Effective, and Scalable (2nd Edition). Нью-Йорк: The Pragmatic Bookshelf, 2022. 274с.

26. .Курт Гунтерот. Optimized C++: Proven Techniques for Heightened Performance. Нью-Йорк. 2020. 388с.

27. Christian Ullenboom. Java: The Comprehensive Guide to Java Programming for Professionals. Дортмунді. 2022. 1180с.

28. Симетричний алгоритм. [Електронний ресурс]. URL: https://medium.com/@karlooo/

29. Модульне тестування. [Електронний ресурс]. URL: https://conf.ztu.edu.ua/wp-content/uploads/2024/01/120-1.pdf

30. Методичні вказівки до виконання бакалаврських кваліфікаційних робіт для студентів спеціальності 121 "Інженерія програмного забезпечення" / Уклад. О. Н. Романюк, Г. О. Черноволик – Вінниця: ВНТУ, 2022. – 52с.

ДОДАТКИ

ДОДАТОК А (обов'язковий). Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н.
«21» 03 2025 р.

Технічне завдання
на бакалаврську кваліфікаційну роботу «Розробка вебзастосунку для криптографічного захисту файлів»
за спеціальністю
121 – Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:
к.т.н., доц. Майданюк В.П.
«21» 03 2025 року.

Виконав:
студент гр. 1ПІ-21б Гиренко В.В.
«21» 03 2025 року.

м. Вінниця – 2025 рік

1. Найменування та галузь застосування

Бакалаврська кваліфікаційна робота: «Розробка вебзастосунку для криптографічного захисту файлів».

Галузь застосування – криптографічний захист даних, навчальний процес.

2. Підстава для розробки.

Підставою для виконання бакалаврської дипломної роботи (БДР) є індивідуальне завдання на БДР та наказ №97 від «20» березня 2025 р. ректора ВНТУ про закріплення тем БДР.

3. Мета та призначення розробки.

Метою роботи є підвищення зручності процесу шифрування файлів за рахунок розробки веб-застосунку, що дозволяє виконувати операції шифрування у браузері без встановлення додаткових програм.

Основними задачами дослідження є:

– розробити архітектуру вебсистеми;
– розробити користувацький інтерфейс;
– розробити модулі шифрування та дешифрування;
– розробити генерацію, збереження та перевірку криптографічних ключів;
– розробити алгоритми роботи застосунку;
– розробити функціонал застосунку;
– провести тестування розробленого застосунку.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись БКР:

1. Криптографія шифрування . [Електронний ресурс] URL: https://iit.com.ua/n

2. Сучасні алгоритми шифрування . [Електронний ресурс] URL: https://www.kingston.com/ua/blog/data-security/what-is-encryption

3. .What is Software Development? . [Електронний ресурс] URL: https://www.ibm.com/think/topics/software-development

4. Аттіла Ковач. Modern Software Testing Techniques: A Practical Guide for Developers and Testers. Нью-Йорк: Apress, 2024. 266с.

5. Гиренко В. В., Майданюк В. П. Шифрування файлів за допомогою веб застосунку. [Електронний ресурс] URL: https://mdpu.org.ua/nauka/konferentsiyi-ta-seminary/

5. Технічні вимоги

- середовище розробки – Microsoft Visual Studio Code;

– мова програмування – JavaScript;

– методи шифрування – симетричне шифрування;

– довжина ключа – 64 біти.

6. Конструктивні вимоги.

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

–пояснювальна записка до бакалаврської кваліфікаційної роботи;

–технічне завдання;

–лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки

№ з/п	Назва етапів бакалаврської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз стану розвитку застосунків для для криптографічного захисту файлів	25.03.25 – 01.04.25
2	Розробка методів та алгоритмів вебзастосунку	02.04.25- 12.04.25
3	Варіантний аналіз та вибір мови програмування розробки	13.04.25- 27.04.25
4	Розробка застосунку	28.04.25- 09.05.25
5	Тестування застосунку	10.05.25- 20.05.25.
6	Оформлення матеріалів до захисту БКР	21.05.25- 30.05.25

10. Порядок контролю та прийняття.

Порядок контролю і приймання роботи регламентується відповідними документами ВНТУ і державними стандартами.

ДОДАТОК А (обов'язковий). Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
Завідувач кафедри ПЗ
Романюк О.Н.
«21» 03 2025 р.

Технічне завдання
на бакалаврську кваліфікаційну роботу «Розробка вебзастосунку для криптографічного захисту файлів»
за спеціальністю
121 – Інженерія програмного забезпечення

Керівник бакалаврської кваліфікаційної роботи:
к.т.н., доц. Майданюк В.П.
«21» 03 2025 року.

Виконав:
студент гр. 1ПІ-21б Гиренко В.В.
«21» 03 2025 року.

м. Вінниця – 2025 рік

ДОДАТОК В (довідниковий). Лістинг програми

```
<!DOCTYPE html>
<html lang="uk">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
  <title>Шифрування</title>
  <link rel="stylesheet" href="css/styles.css" />
</head>
<body>
  <h1>Шифрування та дешифрування</h1>

  <div class="main-container">
    <div class="column">
      <h2>Файли</h2>
      <input type="file" id="fileInput" />
      <button onclick="encryptFile()">Шифрувати файл</button>
      <button onclick="decryptFile()">Дешифрувати файл</button>
      <div id="fileResult"></div>
      <div id="binaryCode"></div>

    </div>


    <div class="column">
      <h2>Жорсткий диск</h2>
      <select id="diskSelector">
        <option value="C">Диск C:</option>
        <option value="D">Диск D:</option>
      </select>
      <button onclick="openModal()">Шифрувати диск</button>
      <button onclick="decryptDisk()">Дешифрувати диск</button>
      <div id="diskBinaryOutput"></div>
      <div id="diskResult"></div>
      <div class="error" id="diskError"></div>
    </div>
  </div>

  <div class="column center-block">
    <h2>Генератор ПВЧ</h2>
    <label>Початкове T(i):</label>
    <input type="number" id="initialT" value="123" />
    <label>A:</label>
```

```
    <input type="number" id="constA" value="13" />
    <label>C:</label>
    <input type="number" id="constC" value="15" />
    <label>M:</label>
    <input type="number" id="constM" value="256" />
    <button onclick="generatePRNG()">Генерувати</button>
    <div id="prngOutput"></div>
  </div>

  <div id="myModal" class="modal">
    <div class="modal-content">
      <span class="close" onclick="closeModal()">&times;</span>
      <h2>Введіть ключ шифру</h2>
      <div style="position: relative;">
        <input type="password" id="diskPassword" placeholder="Введіть ключ-шифр..." />
        <span onclick="togglePassword()" class="eye-icon">👁</span>
      </div>
      <button onclick="encryptDisk()">Шифрувати диск</button>
      <div id="passwordError" class="error"></div>
    </div>
  </div>

  <div id="helpIcon" onclick="showHelp()">❓</div>

  <div id="helpModal" class="modal">
    <div class="modal-content">
      <span class="close" onclick="closeHelp()">&times;</span>
      <h2>Контактна інформація</h2>
      <p><strong>Ім'я:</strong> Вадим</p>
      <p><strong>Номер:</strong> +380980000000</p>
      <p><strong>Telegram:</strong> @g.vadym.v</p>
    </div>
  </div>

  <script src="js/script.js"></script>
</body>
</html>

function stringToSeed(password) {
  let seed = 0;
  for (let i = 0; i < password.length; i++) {
    seed = (seed * 31 + password.charCodeAt(i)) >>> 0;
  }
  return seed;
}
```

```
function generateKeyStream(seed, length) {
  const A = 1664525;
  const C = 1013904223;
  const M = 2 ** 32;
  let T = seed;
  const stream = new Uint8Array(length);

  for (let i = 0; i < length; i++) {
    T = (A * T + C) % M;
    stream[i] = T & 0xFF;
  }

  return stream;
}

function xorEncrypt(data, password = "default") {
  const seed = stringToSeed(password);
  const keyStream = generateKeyStream(seed, data.length);
  const result = new Uint8Array(data.length);
  for (let i = 0; i < data.length; i++) {
    result[i] = data[i] ^ keyStream[i];
  }
  return result;
}

function encryptFile() {
  const fileInput = document.getElementById("fileInput");
  const binaryCode = document.getElementById("binaryCode");
  const result = document.getElementById("fileResult");
  const password = document.getElementById("passwordInput")?.value || "default";

  if (!fileInput.files.length) return;

  const file = fileInput.files[0];
  const reader = new FileReader();
  reader.onload = function (e) {
    const buffer = new Uint8Array(e.target.result);
    const encrypted = xorEncrypt(buffer, password);
    const blob = new Blob([encrypted]);
    const url = URL.createObjectURL(blob);
    const a = document.createElement("a");
    a.href = url;
    a.download = "encrypted_" + file.name;
    a.click();
    binaryCode.textContent = Array.from(encrypted)
      .map(b => b.toString(2).padStart(8, "0"))
      .join(" ");
```

```
    result.textContent = "Файл зашифровано.";
  };
  reader.readAsArrayBuffer(file);
}

function decryptFile() {
  const fileInput = document.getElementById("fileInput");
  const result = document.getElementById("fileResult");
  const password = document.getElementById("passwordInput")?.value || "default";

  if (!fileInput.files.length) return;

  const file = fileInput.files[0];
  const reader = new FileReader();
  reader.onload = function (e) {
    const buffer = new Uint8Array(e.target.result);
    const decrypted = xorEncrypt(buffer, password); // reversible
    const blob = new Blob([decrypted]);
    const url = URL.createObjectURL(blob);
    const a = document.createElement("a");
    a.href = url;
    a.download = "decrypted_" + file.name;
    a.click();
    result.textContent = "Файл дешифровано.";
  };
  reader.readAsArrayBuffer(file);
}

function openModal() {
  document.getElementById("myModal").style.display = "block";
}

function closeModal() {
  document.getElementById("myModal").style.display = "none";
  document.getElementById("passwordError").textContent = "";
}

function togglePassword() {
  const field = document.getElementById("diskPassword");
  field.type = field.type === "password" ? "text" : "password";
}

function encryptDisk() {
  const password = document.getElementById("diskPassword").value;
  const errorDiv = document.getElementById("diskError");
  const passwordError = document.getElementById("passwordError");
  const diskResult = document.getElementById("diskResult");
```

```
  const diskBinaryOutput = document.getElementById("diskBinaryOutput");

  if (password !== "52805280") {
    passwordError.textContent = "Невірний ключ!";
    diskResult.textContent = "";
    diskBinaryOutput.textContent = "";
    return;
  }

  errorDiv.textContent = "";
  passwordError.textContent = "";

  const fakeDiskData = new Uint8Array([65, 66, 67, 68, 69, 70]);
  const encrypted = xorEncrypt(fakeDiskData, password);

  diskResult.textContent = `Диск ${document.getElementById("diskSelector").value}: зашифровано.`;
  diskBinaryOutput.textContent = Array.from(encrypted)
    .map(b => b.toString(2).padStart(8, "0"))
    .join(" ");
  closeModal();
}

function decryptDisk() {
  const selectedDisk = document.getElementById("diskSelector").value;
  document.getElementById("diskResult").textContent = `Диск ${selectedDisk}: дешифровано.`;
}

function showHelp() {
  document.getElementById("helpModal").style.display = "block";
}

function closeHelp() {
  document.getElementById("helpModal").style.display = "none";
}

function generatePRNG() {
  const T = [parseInt(document.getElementById("initialT").value)];
  const A = parseInt(document.getElementById("constA").value);
  const C = parseInt(document.getElementById("constC").value);
  const M = parseInt(document.getElementById("constM").value);
  const P = [1, 2, 3, 4, 5, 6];
  let C_array = [];
  for (let i = 0; i < P.length; i++) {
    T[i + 1] = (A * T[i] + C) % M;
    C_array.push(P[i] ^ (T[i] & 0xFF));
  }
  document.getElementById("prngOutput").textContent = "C(i): " + C_array.join(", ");
```

ДОДАТОК Г (обов’язковий). Ілюстративна частина

ІЛЮСТРАТИВНА ЧАСТИНА

РОЗРОБКА ВЕБ-ЗАСТОСУНКУ ДЛЯ КРИПТОГРАФІЧНОГО ЗАХИСТУ ФАЙЛІВ

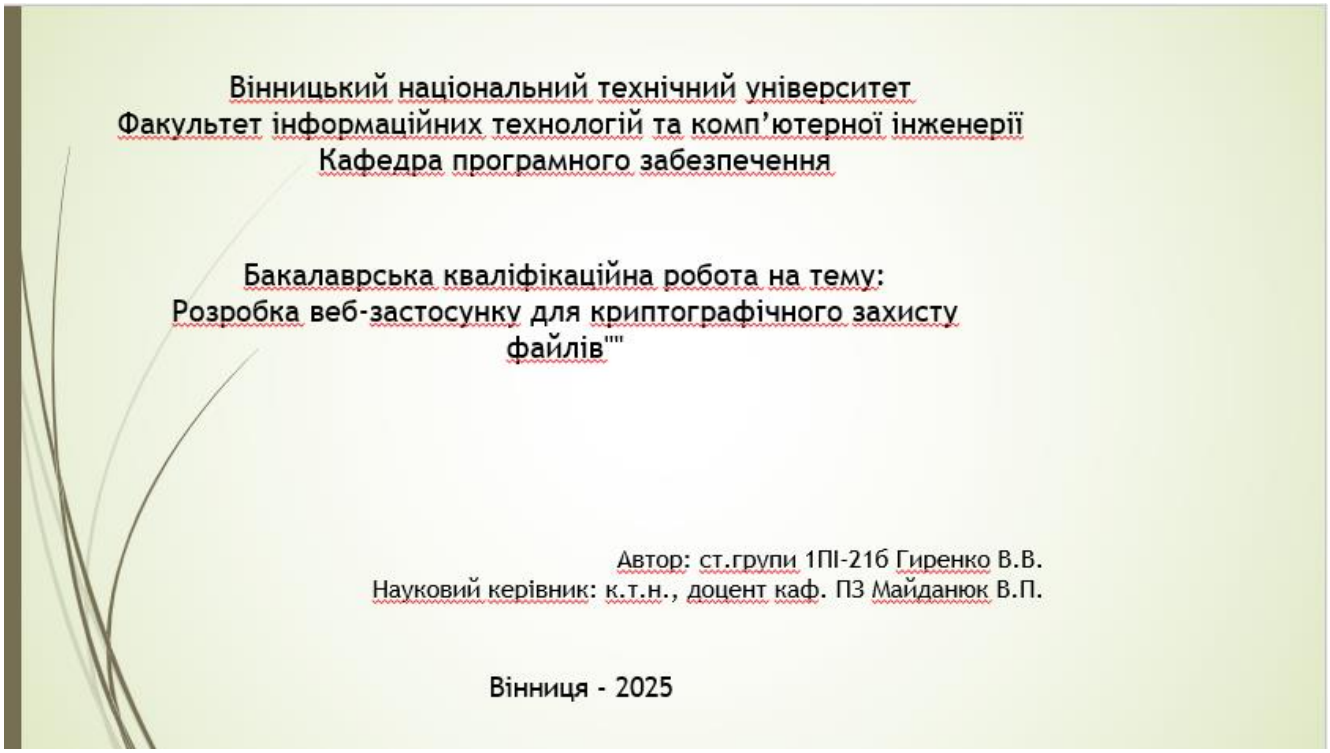


Рисунок Г.1 – Титульна сторінка

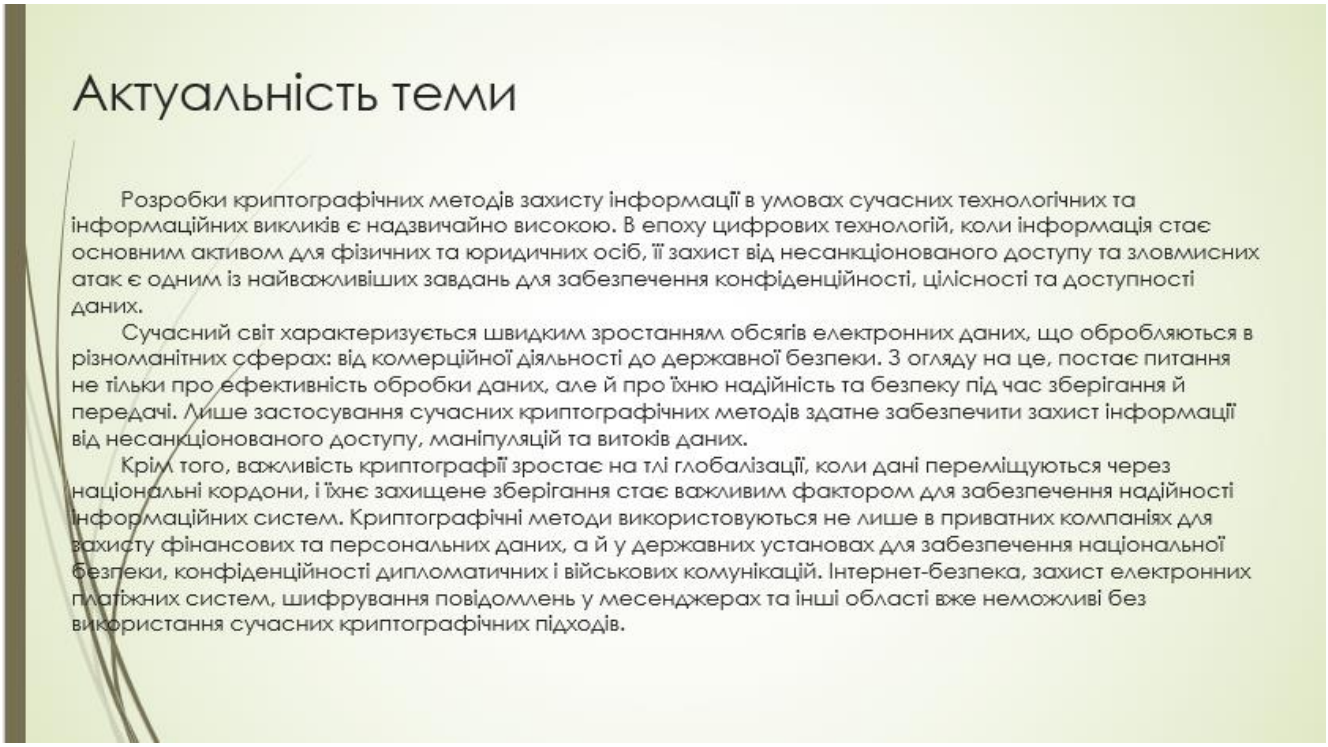


Рисунок Г.2 – Актуальність теми

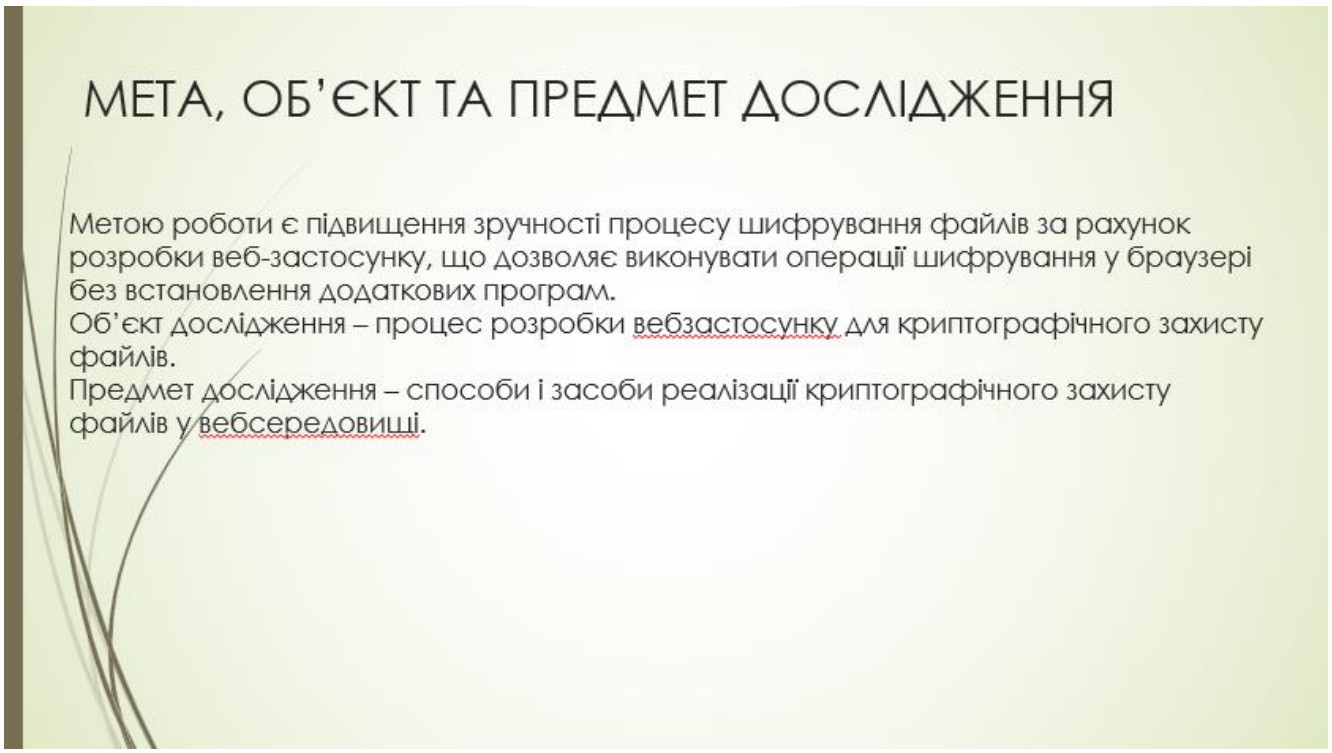


Рисунок Г.3 – Мета , об'єкт та предмет дослідження

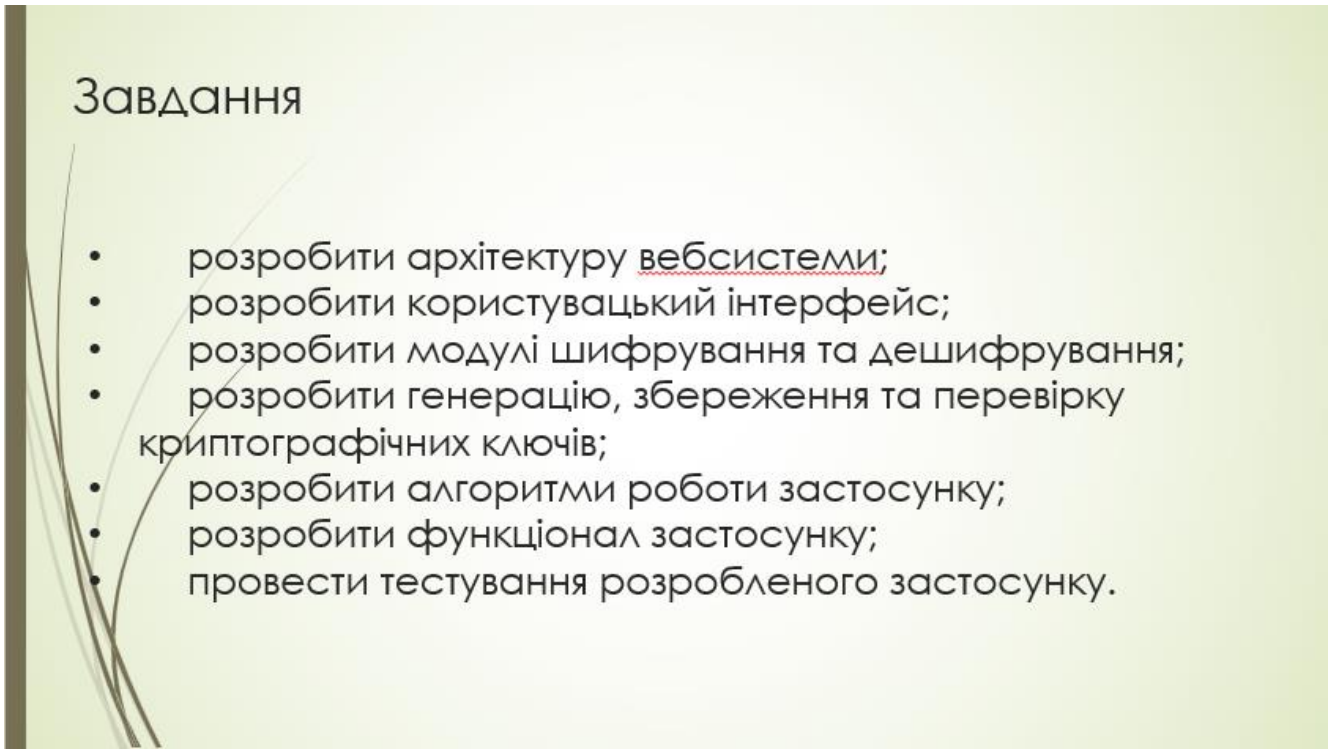


Рисунок Г.4 – Завдання

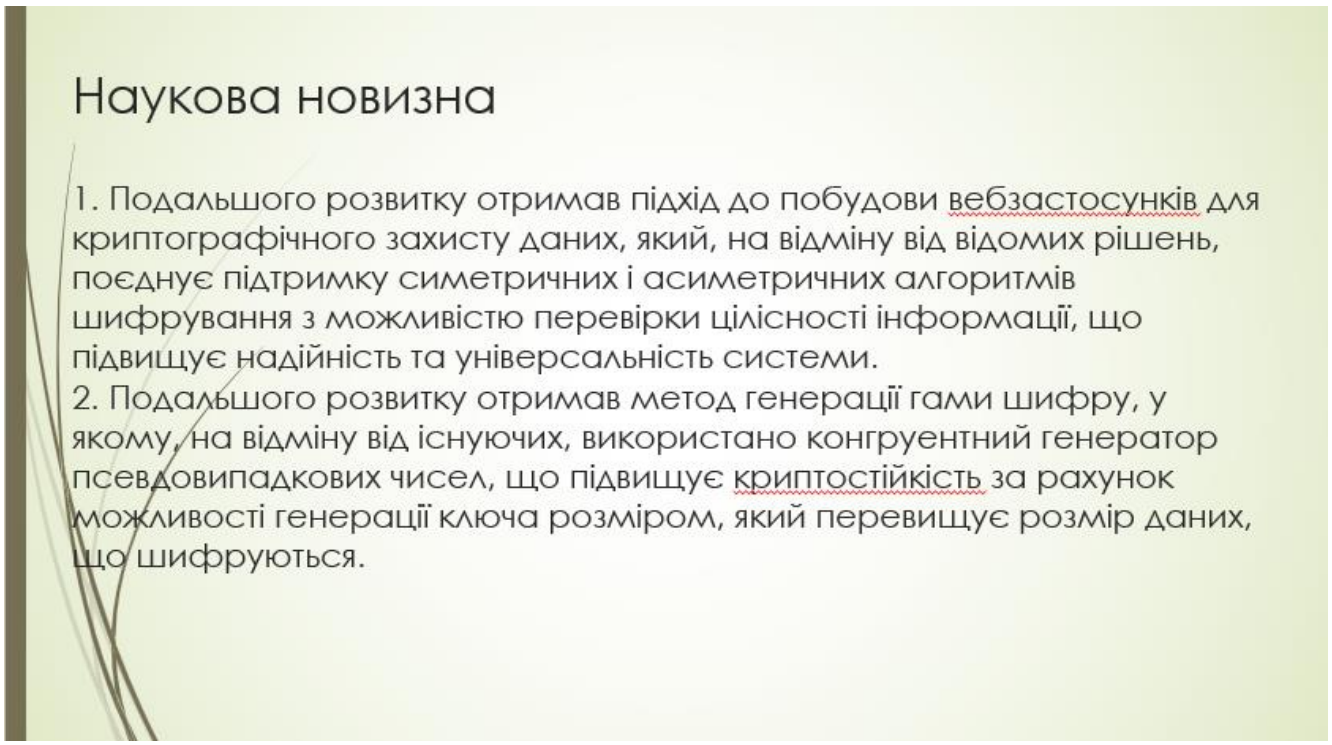


Рисунок Г.5 – Новизна

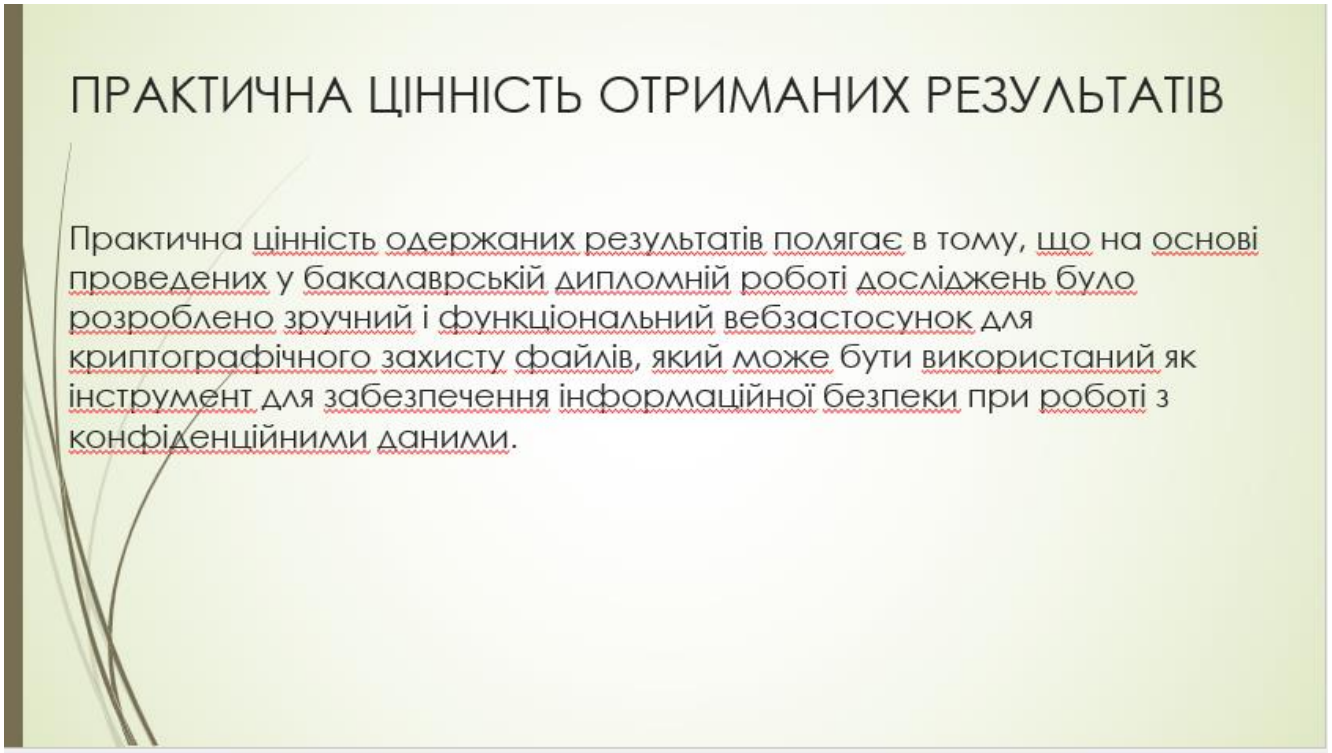


Рисунок Г.6 – Практична цінність

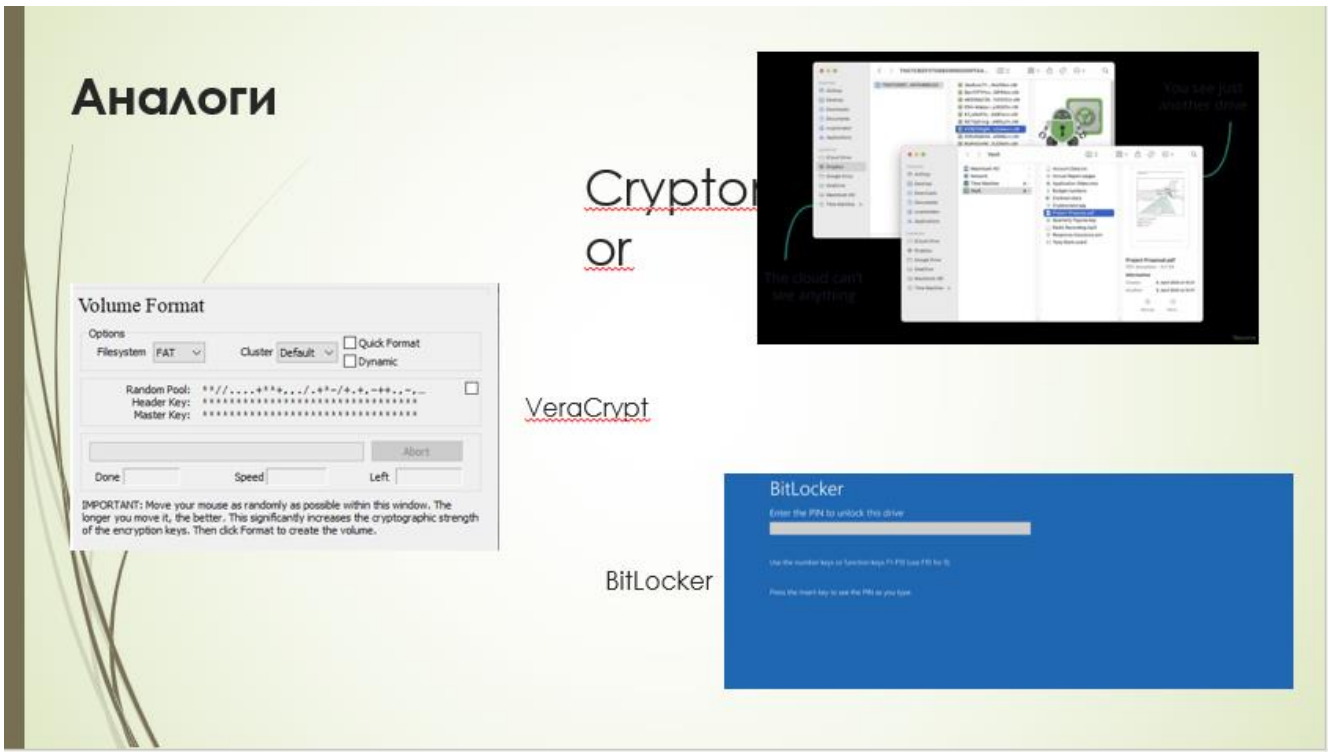
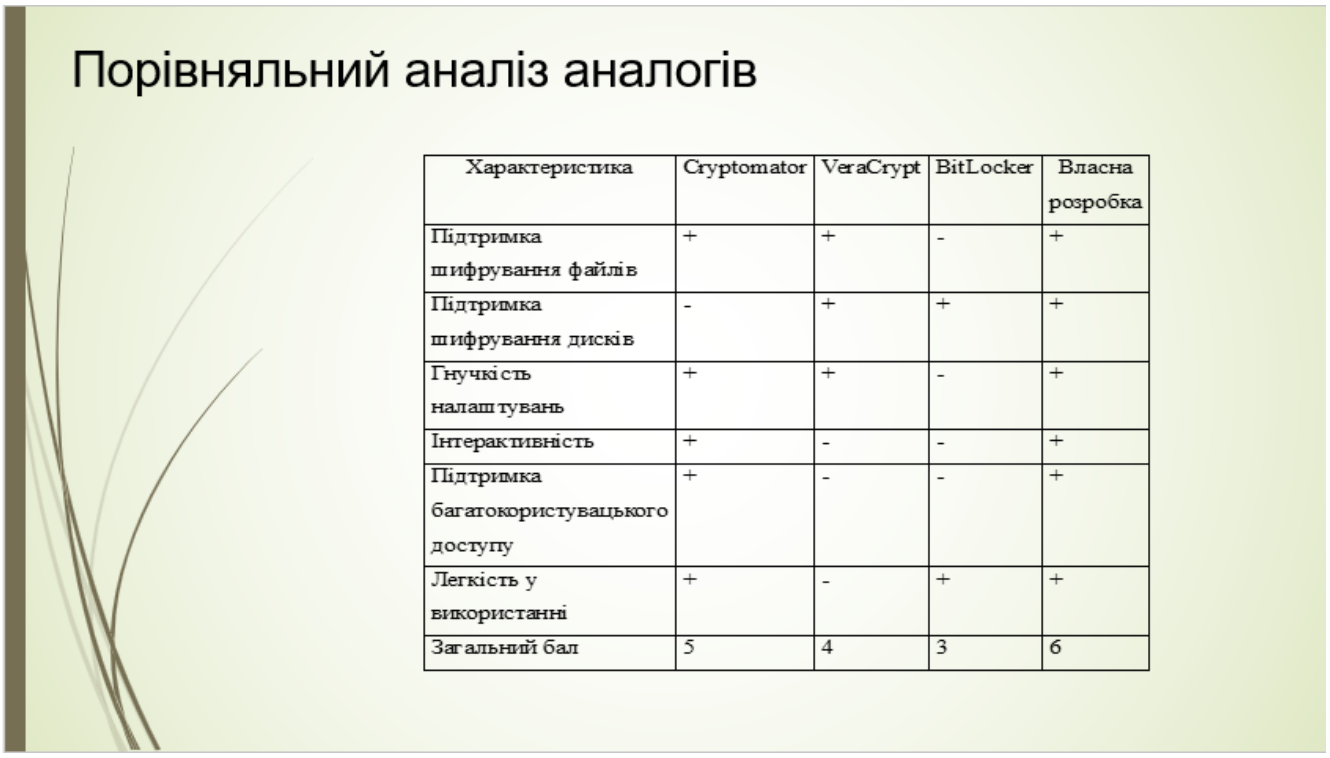


Рисунок Г.7 – Аналоги



Характеристика	Cryptomator	VeraCrypt	BitLocker	Власна розробка
Підтримка шифрування файлів	+	+	-	+
Підтримка шифрування дисків	-	+	+	+
Гнучкість налаштувань	+	+	-	+
Інтерактивність	+	-	-	+
Підтримка багатокористувацького доступу	+	-	-	+
Легкість у використанні	+	-	+	+
Загальний бал	5	4	3	6

Рисунок Г.8 – Порывняльний аналіз

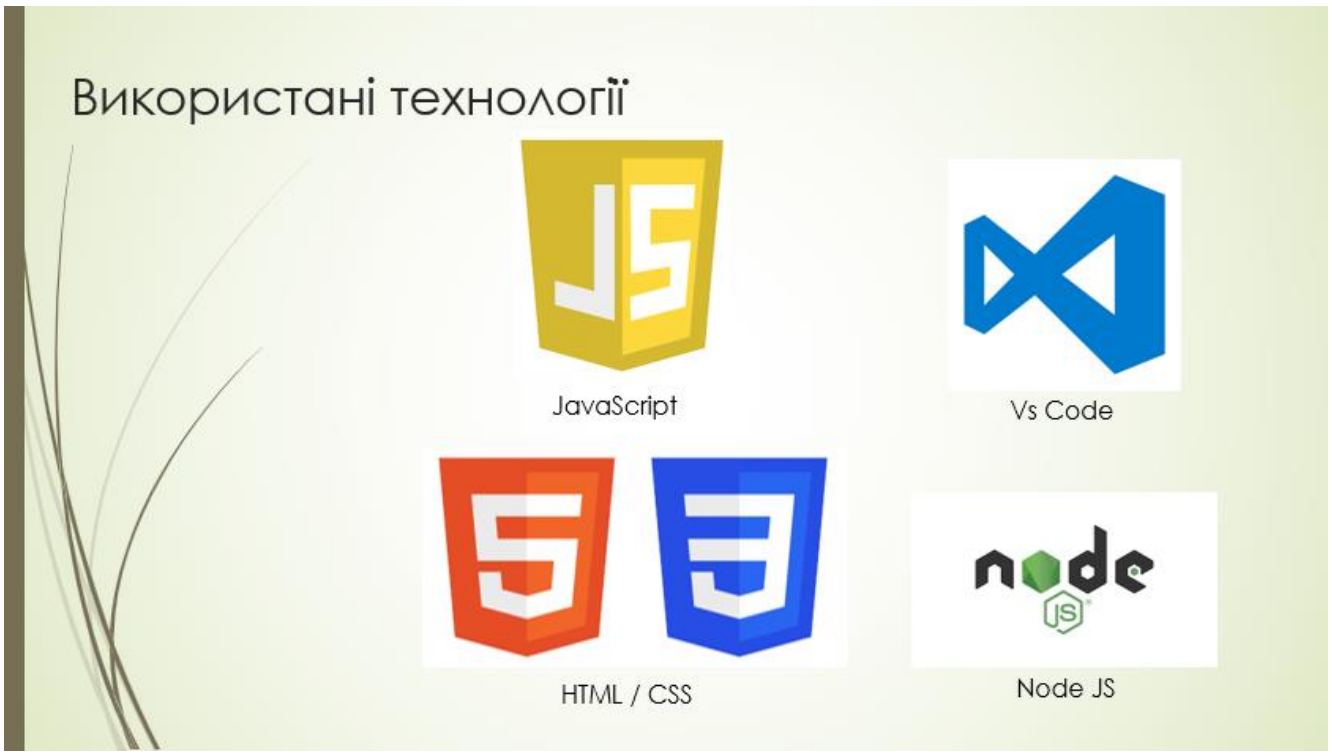


Рисунок Г.9 – Використані технології

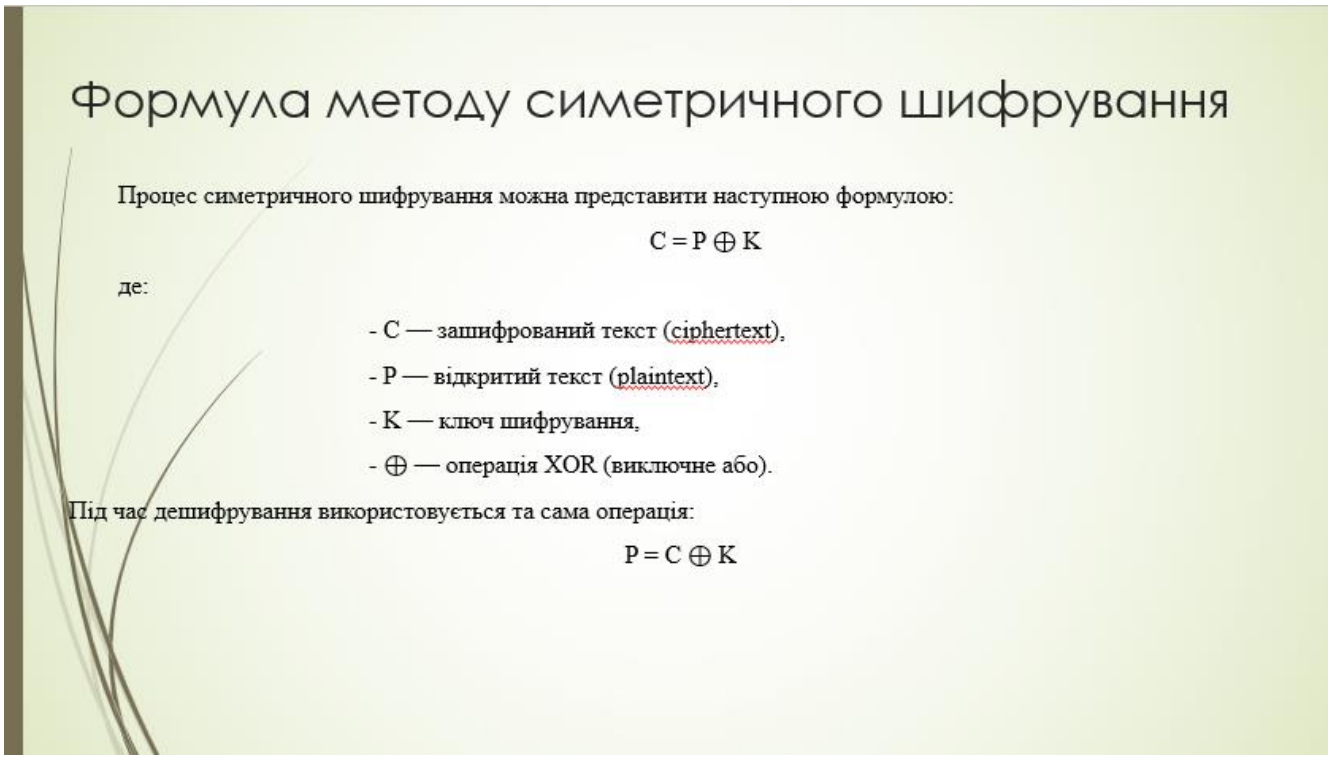


Рисунок Г.10 – Формула симетричного шифрування

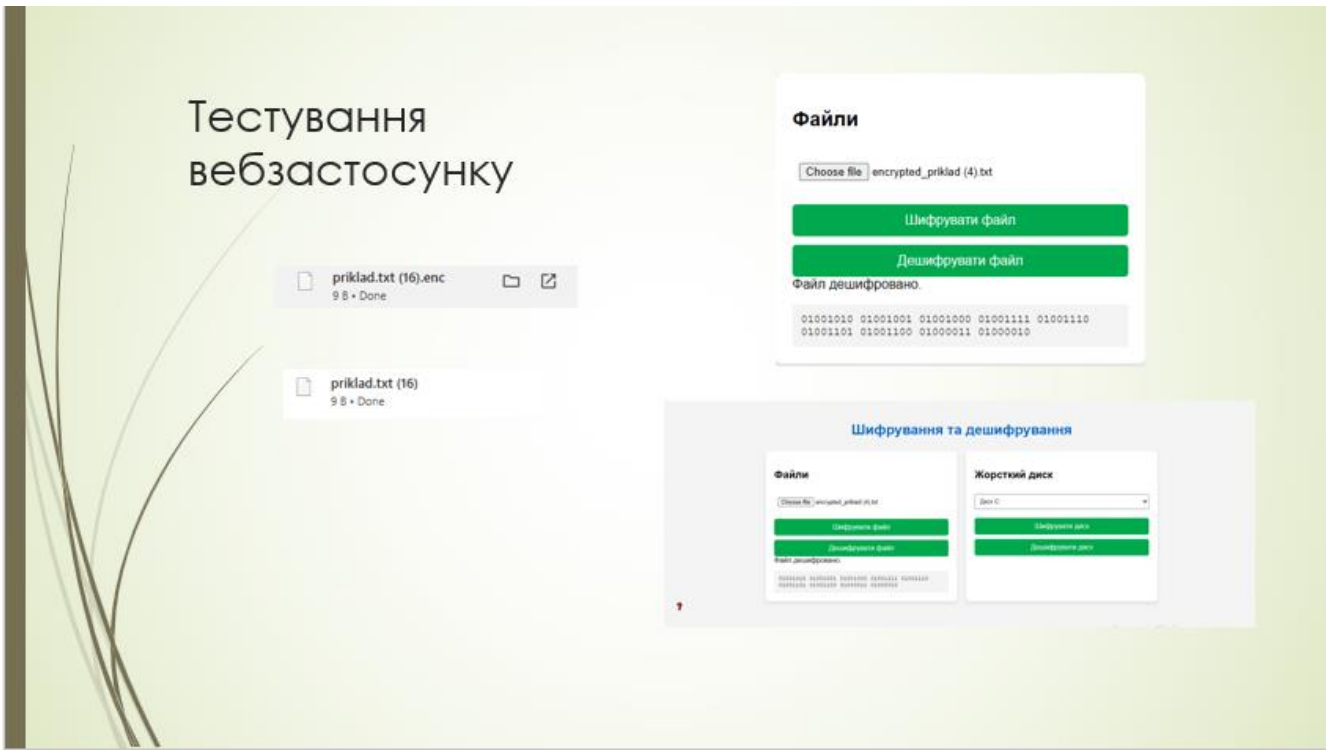


Рисунок Г.11 – Тестування застосунку

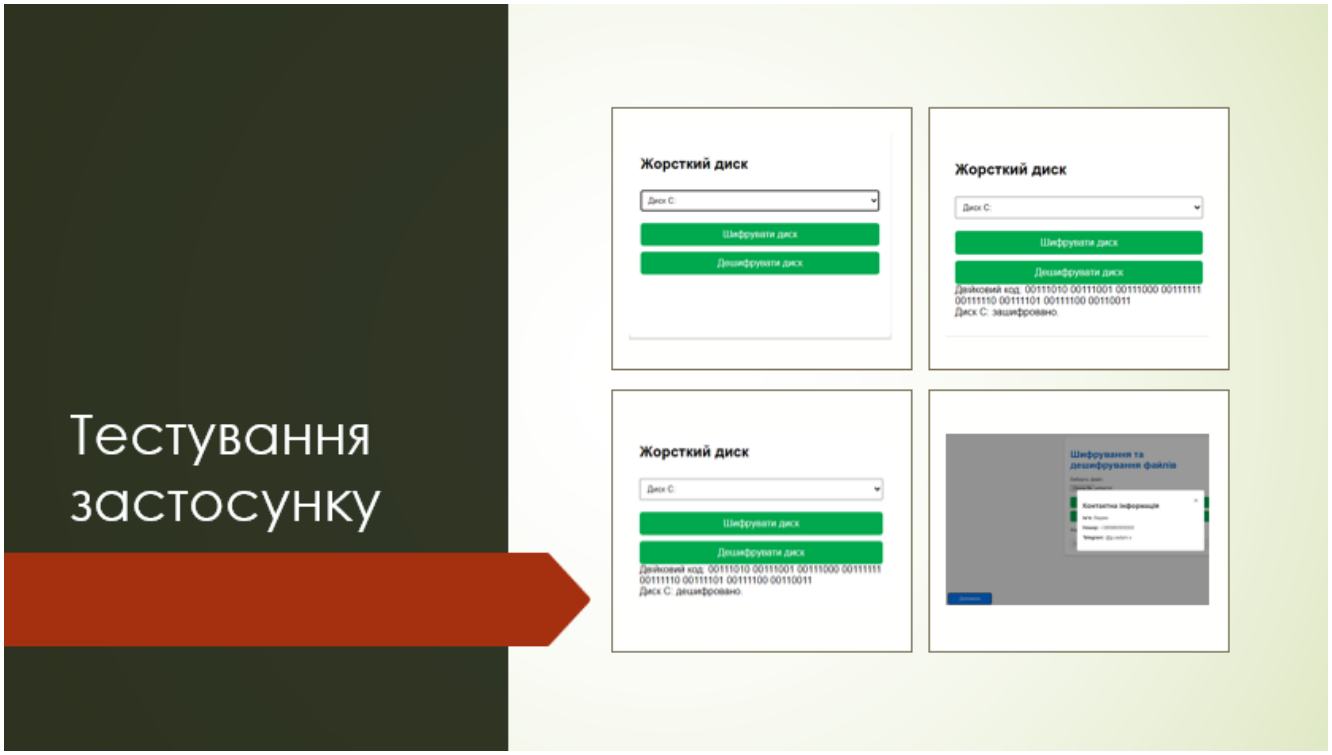


Рисунок Г.12 – Тестування застосунку

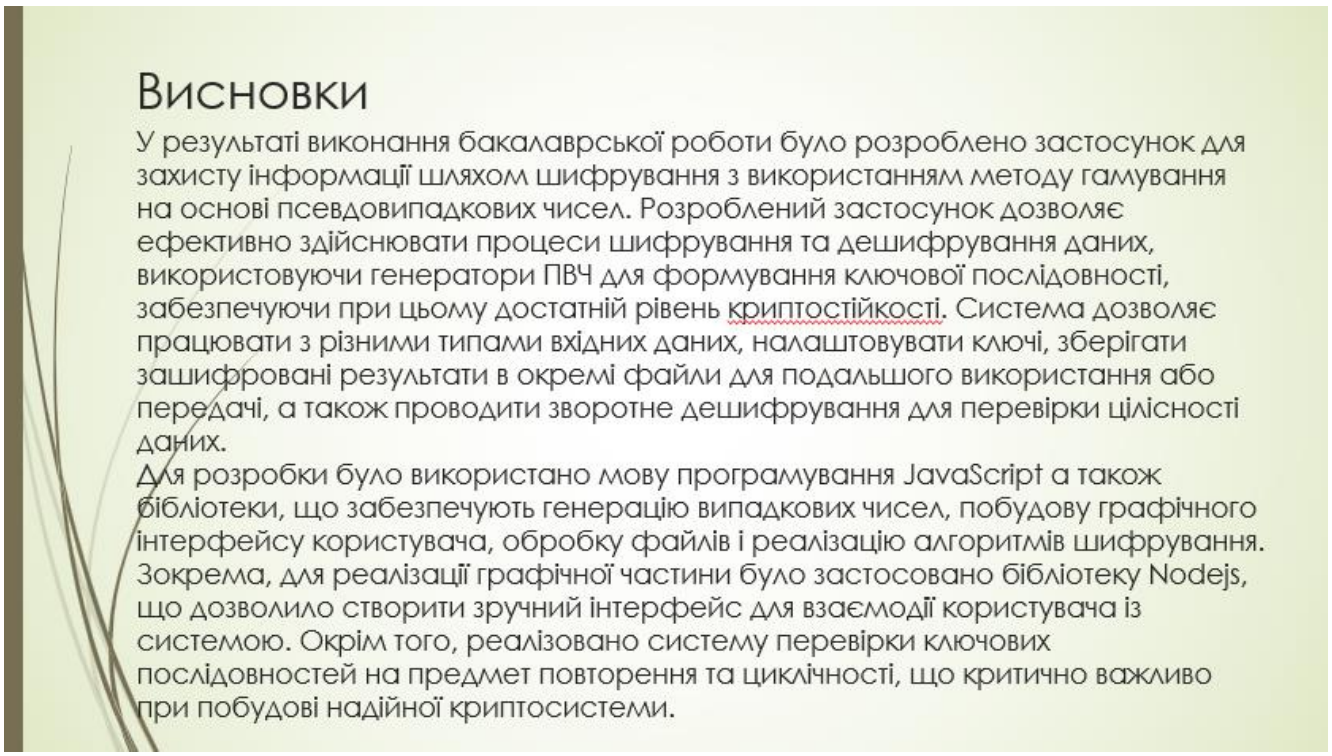


Рисунок Г.13 – Висновок