Національний університет «Полтавська політехніка імені Юрія Кондратюка»
(повне найменування вищого навчального закладу)
Навчально-науковий інститут інформаційних технологій та робототехніки
(повна назва факультету)
Кафедра комп'ютерних та інформаційних технологій і систем
(повна назва кафедри)

Пояснювальна записка
до дипломного проекту (роботи)
_____магістра_____
(освітньо-кваліфікаційний рівень)
на тему

Розробка веб-застосунку для шифрування зображень алгоритмом Triple DES

Виконав: студент 6 курсу, групи 602-ТН
спеціальності
__122 Комп'ютерні науки_____
(шифр і назва напряму)
_____Толочин М. О._____
(прізвище та ініціали)
Керівник _____Головко Г. В._____
(прізвище та ініціали)

Полтава – 2024 року

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИНАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ПОЛТАВСЬКА ПОЛІТЕХНІКА ІМЕНІ ЮРІЯ КОНДРАТЮКА»

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ТА РОБОТОТЕХНІКИ

КАФЕДРА КОМП'ЮТЕРНИХ ТА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І СИСТЕМ

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА
спеціальність 122 «Комп'ютерні науки»

на тему

«Розробка веб-застосунку для шифрування зображень алгоритмом Triple DES»

Студента групи 602-ТН Толочина Михайла Олексійовича

Керівник роботи
кандидат технічних наук,
доцент Головко Г. В.

Завідувач кафедри
кандидат технічних наук,
доцент Двірна О.А.

Полтава – 2024

РЕФЕРАТ

Кваліфікаційна робота магістра: 100 с., 60 рисунків, 1 додаток, 24 джерела.

Об’єкт дослідження: процеси шифрування та розшифрування зображень у цифрових системах з використанням криптографічних алгоритмів, втілені в програмному забезпеченні.

Мета роботи: розробка веб-застосунку для шифрування зображень з використанням алгоритму Triple DES.

Методи: аналіз існуючих рішень, веб-програмування, обробка зображень, шифрування даних, моніторинг системи.

Ключові слова: JAVASCRIPT, HTML, CSS, CRYPTOGRAPHY, IMAGE.

ANNOTATION

Qualification work of master's degree: 100 p., 60 figures, 1 application, 24 sources.

Object of study: processes of encrypting and decrypting images in digital systems using cryptographic algorithms, implemented in software.

The goal of the work: to develop a web application for image encryption using the Triple DES algorithm.

Methods: analysis of existing solutions, web programming, image processing, data encryption, system monitoring.

Keywords: JAVASCRIPT, HTML, CSS, CRYPTOGRAPHY, IMAGE.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ, СИМВОЛІВ І ТЕРМІНІВ

AES – Advanced Encryption Standard (Стандарт симетричного шифрування).

API – Application Programming Interface (Інтерфейс прикладного програмування).

BD – Big Data (Великі дані).

CC – Cryptographic Checksum (Криптографічна контрольна сума).

CPU – Central Processing Unit (Центральний процесор).

DES – Data Encryption Standard (Стандарт шифрування даних).

DFA – Differential Fault Analysis (Аналіз диференціальних помилок).

DH – Diffie-Hellman (Алгоритм Діффі-Геллмана).

ECC – Elliptic Curve Cryptography (Криптографія на еліптичних кривих).

ECB – Electronic Codebook Mode (Режим електронної кодової книги).

FIPS – Federal Information Processing Standards (Федеральні стандарти обробки інформації).

FPGA – Field Programmable Gate Array (Програмована логічна інтегральна схема).

GUI – Graphical User Interface (Графічний інтерфейс користувача).

HTTPS – Hypertext Transfer Protocol Secure (Протокол захищеної передачі гіпертексту).

IDE – Integrated Development Environment (Інтегроване середовище розробки).

IoT – Internet of Things (Інтернет речей).

JSON – JavaScript Object Notation (Нотація об'єктів JavaScript).

MAC – Message Authentication Code (Код автентифікації повідомлення).

PC – Personal Computer (Персональний комп'ютер).

RSA – Rivest-Shamir-Adleman (Алгоритм шифрування RSA).

SSL – Secure Sockets Layer (Протокол захисту мережевих з'єднань).

TCP – Transmission Control Protocol (Протокол управління передачею).

TDEA – Triple Data Encryption Algorithm (Алгоритм потрійного шифрування даних, інша назва Triple DES).

URL – Uniform Resource Locator (Уніфікований покажчик ресурсу).

UX – User Experience (Досвід користувача).

VPN – Virtual Private Network (Віртуальна приватна мережа).

XML – Extensible Markup Language (Розширювана мова розмітки).

XOR – Exclusive OR (Виключне АБО, логічна операція).

ВСТУП

Розвиток інформаційних технологій призводить до постійного зростання обсягу цифрових даних, які потребують надійного захисту від несанкціонованого доступу. Захист інформації, особливо зображень, що передаються через Інтернет або зберігаються в хмарних сховищах, є надзвичайно важливим, оскільки це може стосуватися як особистих даних, так і конфіденційної інформації підприємств. У цьому контексті особливу роль відіграють криптографічні алгоритми, які забезпечують шифрування та розшифрування даних, зокрема зображень.

Одним з найбільш надійних та перевірених методів шифрування є алгоритм Triple DES (Triple Data Encryption Standard). Цей алгоритм є модифікацією класичного DES і забезпечує більш високий рівень безпеки завдяки використанню трьох ключів для послідовного шифрування даних. Використання Triple DES дозволяє ефективно захищати інформацію від різних типів криптографічних атак, що робить його актуальним для застосування в сучасних інформаційних системах.

Метою цієї дипломної роботи є розробка веб-застосунку для шифрування зображень з використанням алгоритму Triple DES. Такий веб-застосунок дозволить користувачам зручно шифрувати та розшифровувати зображення, забезпечуючи тим самим їх безпеку при передачі або зберіганні.

Актуальність зумовлена потребою в надійних та доступних інструментах захисту даних, які могли б бути інтегровані в різні веб-системи.

Об’єкт дослідження: процеси шифрування та розшифрування зображень у цифрових системах з використанням криптографічних алгоритмів.

Предмет дослідження: алгоритм Triple DES та його застосування для шифрування зображень у веб-застосунках.

У ході дослідження буде проаналізовано сучасні методи шифрування зображень, розглянуто принцип роботи алгоритму Triple DES, а також здійснено розробку та тестування веб-застосунку для шифрування. Результати цієї роботи можуть бути корисними як для подальших наукових досліджень у сфері криптографії, так і для практичного використання в інформаційних системах.

РОЗДІЛ 1
ТЕОРЕТИЧНІ АСПЕКТИ ШИФРУВАННЯ ЗОБРАЖЕНЬ

1.1 Основні технології для створення веб-застосунків

Говорячи про структуру та стиль веб-сторінок: мова розмітки HTML використовується для створення структури веб-застосунку, а CSS — для стилізації. Це важливо для створення зрозумілого і привабливого інтерфейсу користувача. Разом із JavaScript, HTML і CSS складають основу будь-якого сайту (див. Рис. 1.1) [1].

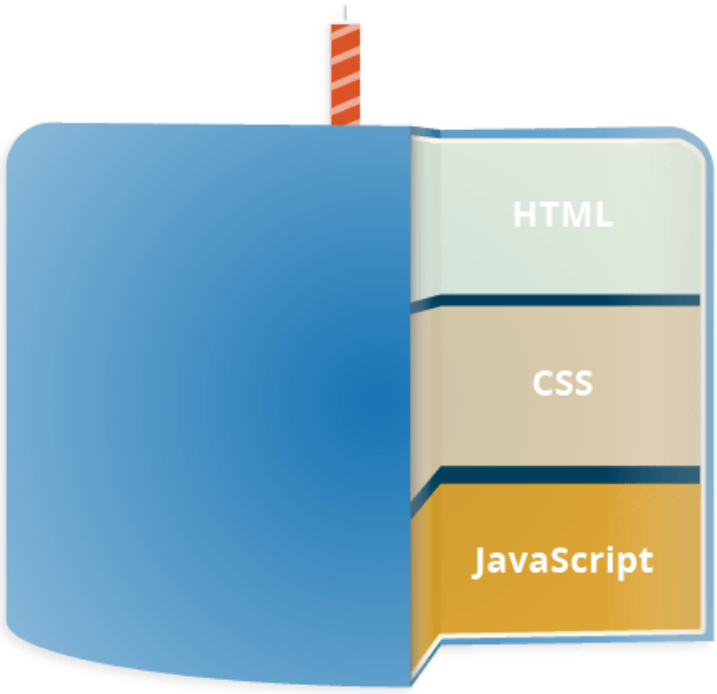

Рисунок 1.1 – Технології основи будь-якого сайту

HTML розшифровується як HyperText Markup Language (мова розмітки гіпертексту). Гіпертекст – це фактично текст у тексті, а мова розмітки – це мова, зрозуміла комп'ютерам, призначена для опису веб-сторінок і для того, щоб зробити текст більш інтерактивним [2].

Вона була винайдена в 1989 році як мова публікації в Інтернеті, і, якщо говорити дуже просто, це перший будівельний блок у створенні веб-сторінки. Коли ви вперше зіткнетеся зі світом HTML, ви, швидше за все, зіткнетеся з

цілою купою термінів і фраз, які також можуть дещо спантеличити. Три основні з них – це елементи, теги та атрибути.

HTML5 – це остання «версія» HTML, і знання в цій галузі є зростаючою тенденцією на ринку праці. За великим рахунком, він має таку ж функціональність, як і стандартний HTML, але набагато динамічніший і використовує набагато менше коду для створення чогось фантастичного [2].

HTML використовується для створення первинного контенту веб-сторінки, надаючи їй структуру. Розробник починає з написання слів, потім додає до них теги або елементи. Веб-браузер зчитує це і може зрозуміти заголовок сторінки, абзаци, а також те, де сторінка починається і де закінчується, таким чином наповнюючи вашу веб-сторінку контентом.

HTML підтримується кожним браузером і встановлений практично на кожній існуючій веб-сторінці. Вам не потрібні жодні ліцензії, вам не потрібно платити за неї, і її досить легко вивчити та кодувати. Якщо порівняти веб-сторінку з людським тілом, то HTML – це кістки тіла.

CSS – це абревіатура для каскадних таблиць стилів. Коротко кажучи, це мова стилів листів, яка є різновидом мови, яку можна використовувати для опису представлення мови розмітки – в даному випадку, для опису рухів HTML. Вона фактично визначає, як будівельні блоки, закладені HTML, будуть оформлені і представлені користувачеві [1].

CSS була вперше створена приблизно в 1996 році, щоб зробити HTML зрозумілішим і надати веб-сторінці чудового вигляду. Як і у випадку з будь-якою мовою, CSS потрібно писати, і знання того, як ми пишемо CSS, є навичкою, якою повинен володіти будь-який розробник веб-сторінок. Простий в підтримці, CSS є другою частиною двокомпонентного набору інструментів для створення веб-сторінок.

Веб-доступність дуже важлива для багатьох роботодавців і компаній в наш час, і існує жорстка конкуренція у створенні найбільш доступних і добре продуманих веб-сторінок. Вивчення деяких основ розробки програмного

забезпечення з використанням CSS може допомогти вам навчитися створювати доступні веб-сторінки.

Якщо HTML – це кістки тіла, то CSS – це шкіра, яка його покриває. Він використовується для кольору фону, стилів, макетів, меж, затінення – усіх важливих елементів дизайну, які роблять веб-сторінку витонченою та елегантною. CSS дозволяє розмежовувати презентацію і контент, змінюючи дизайн і відображення елементів HTML [3].

1.2 Принципи сучасної криптографії

Історично криптографія була більше мистецтвом, ніж наукою. Схеми розроблялися від випадку до випадку і оцінювалися на основі їхньої передбачуваної складності або розумності. Схему аналізували, щоб побачити, чи можна знайти якісь атаки; якщо так, то схему «латали», щоб запобігти атаці, і процес повторювався. Хоча, можливо, існувала згода, що деякі схеми не були безпечними (про що свідчила особливо руйнівна атака), не було узгодженого уявлення про те, яким вимогам повинна задовольняти «безпечна» схема, і не було способу надати докази того, що будь-яка конкретна схема була безпечною. За останні кілька десятиліть криптографія перетворилася на науку [1].

Схеми тепер розробляються і аналізуються більш систематично, з кінцевою метою – дати строгий доказ того, що дана конструкція є безпечною. Для того, щоб сформулювати такі докази, нам спочатку потрібні формальні визначення, які точно вказують на те, що означає «безпечний»; такі визначення корисні і цікаві самі по собі. Виявляється, що більшість криптографічних доведень покладаються на недоведені на даний момент припущення про алгоритмічну складність певних математичних задач; будь-які такі припущення повинні бути явними і точно сформульованими. Акцент на визначеннях, припущеннях і доведеннях відрізняє сучасну криптографію від класичної криптографії [1].

Принцип 1 – Формальні визначення. Одним з ключових внесків сучасної криптографії стало визнання того, що формальні визначення безпеки є важливими для правильного проектування, вивчення, оцінки та використання криптографічних примітивів.

Дійсно, набагато краще формалізувати те, що потрібно, до початку процесу розробки, ніж придумувати визначення постфактум, коли розробка вже завершена. Останній підхід може призвести до того, що фаза проектування закінчиться тоді, коли терпіння проектувальників вичерпається (а не тоді, коли мета буде досягнута), або може призвести до того, що конструкція досягне більшого, ніж потрібно, за рахунок зниження ефективності. Визначення також пропонують спосіб оцінити та проаналізувати те, що побудовано. Маючи визначення, можна вивчити запропоновану схему, щоб побачити, чи забезпечує вона бажані гарантії; в деяких випадках можна навіть довести безпеку певної конструкції, показавши, що вона відповідає визначенню [2].

Хоча ми відкладаємо формальне визначення безпечного шифрування до наступних двох розділів, тут ми неформально опишемо, що таке визначення повинно охоплювати. Загалом, визначення безпеки складається з двох компонентів: гарантії безпеки (або, з точки зору зловмисника, того, що є успішною атакою на схему) і моделі загроз.

Гарантія безпеки визначає, що схема покликана перешкодити зловмиснику, в той час як модель загроз описує можливості супротивника, тобто, які дії зловмисник, як припускається, може здійснити. Почнемо з першого з них [3].

Кілька міркувань про те, що повинна гарантувати безпечна схема шифрування:

- зловмисник не зможе відновити ключ. Раніше ми вже зазначали, що якщо зловмисник може визначити ключ, яким користуються дві сторони, використовуючи деяку схему, то ця схема не може бути безпечною. Однак, легко придумати схеми, для яких відновлення ключа є неможливим, але при

цьому схема є явно небезпечною. Розглянемо, наприклад, схему, де Enck(m) = m. Зашифрований текст не містить жодної інформації про ключ (і тому ключ неможливо відновити, якщо він достатньо довгий), але повідомлення надсилається у відкритому вигляді! Таким чином, ми бачимо, що неможливість відновити ключ не є достатньою умовою безпеки. Це має сенс: метою шифрування є захист повідомлення; ключ є засобом для досягнення цієї мети, але сам по собі він не є важливим;

- зловмисник не повинен мати можливості відновити весь відкритий текст із зашифрованого. Це визначення краще, але все ще далеке від задовільного. Зокрема, за цим визначенням схема шифрування вважається безпечною, якщо її зашифрований текст розкриває 90% відкритого тексту, доки 10% відкритого тексту залишаються складними для розгадування. Це явно неприйнятно в більшості поширених застосувань шифрування; наприклад, при шифруванні бази даних зарплат, ми були б справедливо засмучені, якби 90% зарплат співробітників було розкрито [2];
- зловмисник не повинен мати можливості відновити будь-який символ відкритого тексту із зашифрованого. Це виглядає як гарне визначення, але все ж таки недостатнє. Повертаючись до прикладу з шифруванням бази даних зарплат, ми не будемо вважати схему шифрування безпечною, якщо вона показує, чи є зарплата працівника більшою або меншою за $100,000, навіть якщо вона не розкриває жодної конкретної цифри зарплати цього працівника. Так само ми не хотіли б, щоб схема шифрування показувала, чи заробляє працівник А більше, ніж працівник Б [2].

Тепер, коли ми визначили мету безпеки, залишається визначити модель загрози. Вона визначає, яку «силу» має зловмисник, але не накладає жодних обмежень на стратегію супротивника. Це важлива відмінність: ми вказуємо, що ми припускаємо про можливості супротивника, але ми нічого не припускаємо про те, як він використовує ці можливості.

Неможливо передбачити, які стратегії можуть бути використані в атаці, і історія довела, що спроби зробити це приречені на провал.

Існує кілька вірогідних варіантів моделі загроз в контексті шифрування; стандартні з них, в порядку зростання потужності зловмисника, такі:

- атака лише на шифрований текст: це найпростіша атака, яка відноситься до сценарію, коли зловмисник просто спостерігає зашифрований текст (або декілька зашифрованих текстів) і намагається визначити інформацію про відкритий текст (або відкриті тексти), що лежить в його основі [1];
- атака з відомим відкритим текстом: тут зловмисник може дізнатися одну або декілька пар відкритий текст/шифрований текст, згенерованих за допомогою певного ключа. Мета зловмисника полягає в тому, щоб вивести інформацію про відкритий текст, що лежить в основі іншого зашифрованого тексту, створеного з використанням того ж самого ключа. Всі класичні схеми шифрування, які ми бачили, тривіально зламати за допомогою атаки з відомим відкритим текстом; ми залишаємо демонстрацію в якості вправи;
- атака з обраним відкритим текстом: У цій атаці зловмисник може отримати пари відкритий текст/шифрований текст (як описано вище) для обраних ним відкритих текстів;
- атака на обраний шифрований текст: останній тип атаки, коли зловмисник може додатково отримати (деяку інформацію про) розшифровку обраного ним шифрованого тексту, наприклад, чи розшифровка деякого шифрованого тексту, обраного зловмисником, дає дійсне англомовне повідомлення. Метою зловмисника, знову ж таки, є отримання інформації про відкритий текст, що лежить в основі іншого зашифрованого тексту (розшифрування якого зловмисник не може отримати безпосередньо) [4].

Жодна з цих моделей загроз за своєю суттю не є кращою за будь-яку іншу; правильне використання залежить від середовища, в якому розгорнуто схему шифрування. Перші два типи атак найпростіші у виконанні. При атаці тільки зашифрованого тексту все, що потрібно зробити зловмиснику, – це підслухати публічний канал зв'язку, яким надсилаються зашифровані

повідомлення. В атаці з відомим відкритим текстом передбачається, що противник якимось чином також отримує зашифровані тексти, що відповідають відомим відкритим текстам. Часто це легко зробити, оскільки не всі зашифровані повідомлення є конфіденційними, принаймні, не назавжди.

Як тривіальний приклад, дві сторони завжди можуть зашифрувати повідомлення «привіт», коли вони починають спілкуватися. Більш складний приклад – шифрування може використовуватися для збереження в таємниці квартальних звітів про прибутки до моменту їх оприлюднення; в цьому випадку будь-хто, хто підслуховує зашифрований текст, згодом отримає відповідний відкритий текст.

В останніх наведених атаках передбачається, що зловмисник може отримати зашифровані та/або розшифровані відкриті/зашифровані тексти на свій вибір [3].

Принцип 2 – Точні припущення. Більшість сучасних криптографічних конструкцій не можуть бути безумовно доведені як безпечні; такі докази вимагають вирішення питань теорії обчислювальної складності, на які на сьогоднішній день, здається, ще далеко не знайдено відповідей.

Результатом такого прикрого стану речей є те, що докази безпеки зазвичай покладаються на припущення. Сучасна криптографія вимагає, щоб будь-які такі припущення були явними і математично точними. На найпростішому рівні, це просто тому, що цього вимагають математичні докази безпеки. Але є й інші причини:

1. Перевірка припущень: за своєю природою припущення – це твердження, які не доводяться, а вважаються істинними. Для того, щоб зміцнити нашу віру в якесь припущення, необхідно, щоб це припущення було вивчене. Чим більше припущення досліджується і перевіряється, не будучи спростованим, тим більше ми впевнені в тому, що воно є істинним. Крім того, вивчення припущення може надати докази його достовірності, показавши, що воно випливає з іншого припущення, яке також є загальноприйнятим. Якщо припущення, на яке спираються, не сформульоване точно, його неможливо

вивчити і (потенційно) спростувати. Таким чином, передумовою підвищення нашої довіри до припущення є точне формулювання того, що саме ми припускаємо [4].

2. Порівняння схем: часто в криптографії нам представляють дві схеми, для яких можна довести, що вони задовольняють деякому визначенню, кожна з яких ґрунтується на різних припущеннях. За інших рівних умов, треба думати, якій схемі слід віддати перевагу. Якщо припущення, на якому базується перша схема, є слабшим за припущення, на якому базується друга схема (тобто, друге припущення випливає з першого), то перша схема є кращою, оскільки може виявитися, що друге припущення є хибним, в той час як перше є істинним. Якщо припущення, які використовуються в обох схемах, не можна порівняти, то загальне правило полягає в тому, щоб віддати перевагу схемі, яка базується на краще вивченому припущенні, в якому є більша впевненість [4].

3. Розуміння необхідних припущень: схема шифрування може базуватися на певному базовому будівельному блоці. Якщо основні припущення щодо будівельного блоку чітко визначені в рамках доведення безпеки схеми, то нам потрібно лише перевірити, чи впливають на ці припущення нові вразливості, які були знайдені.

У деяких випадках – наприклад, коли схема успішно протистоїть атакам протягом багатьох років – це може бути розумним підходом. Але такий підхід ніколи не є кращим, а при впровадженні нової схеми – просто небезпечним [3].

Ще однією перевагою покладання на «низькорівневі» припущення (а не просто припущення, що конструкція є безпечною) є те, що ці низькорівневі припущення, як правило, можуть бути використані в інших конструкціях. Нарешті, низькорівневі припущення можуть забезпечити модульність. Розглянемо схему шифрування, безпека якої залежить від певної припущеної властивості одного з її будівельних блоків. Якщо виявиться, що базовий блок не задовольняє заявленому припущенню, схему шифрування все одно можна

реалізувати за допомогою іншого компонента, який, як вважається, задовольняє необхідним вимогам.

Принцип 3 – Докази безпеки. Описані вище два принципи дозволяють нам досягти нашої мети – надати строгий доказ того, що конструкція задовольняє заданому визначенню при певних заданих припущеннях. Такі докази особливо важливі в контексті криптографії, де є зловмисник, який активно намагається «зламати» певну схему. Докази безпеки дають залізну гарантію – відносно визначення і припущень – що зловмисник не досягне успіху; це набагато краще, ніж безпринципний або евристичний підхід до проблеми. Не маючи доказів того, що жоден супротивник із зазначеними ресурсами не може зламати певну схему, ми залишаємося лише з інтуїцією, що це саме так. Досвід показує, що інтуїція в криптографії та комп'ютерній безпеці є згубною. Існує незліченна кількість прикладів недоведених схем, які були зламані, іноді відразу, а іноді через роки після розробки [2].

Підсумовуючи, суворі підходи до безпеки проти спеціальних підходів є суворим підходом до криптографії, який відрізняється від неформального підходу класичної криптографії. На жаль, безпринципні, «нестандартні» рішення все ще розробляються і застосовуються тими, хто бажає отримати швидке вирішення проблеми, або тими, хто просто не знає, що таке криптографія [4].

1.3 Симетричні алгоритми криптографії

Симетричні криптографічні схеми також називають схемами або алгоритмами з симетричним ключем, секретним ключем і одним ключем [3].

Симетричну криптографію найкраще представити на простому прикладі: є два користувачі, Аліса і Боб, які хочуть спілкуватися по незахищеному каналу (див. Рис. 1.2). Термін «канал» може звучати дещо абстрактно, але це лише загальний термін для позначення лінії зв'язку: Це може бути Інтернет, повітряний простір у випадку мобільних телефонів або

бездротової локальної мережі, або будь-який інший засіб зв'язку, який ви можете собі уявити. Фактична проблема починається з поганого хлопця, Оскара, який має доступ до каналу, наприклад, зламавши інтернет-маршрутизатор або прослуховуючи радіосигнали Wi-Fi зв'язку. Такий тип несанкціонованого прослуховування називається підслуховуванням [6].

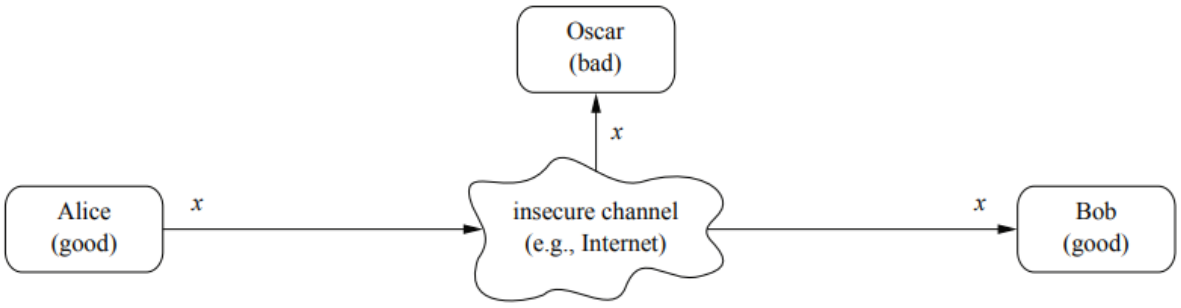

Рисунок 1.2 – Приклад ситуації для симетричної криптографії

Очевидно, що існує багато ситуацій, в яких Аліса і Боб воліли б спілкуватися без прослуховування Оскара. Наприклад, якщо Аліса і Боб представляють два офіси автовиробника і передають документи, що містять бізнес-стратегію впровадження нових моделей автомобілів у найближчі кілька років, ці документи не повинні потрапити до рук їхніх конкурентів або іноземних спецслужб, якщо вже на те пішло [5].

У цій ситуації симетрична криптографія пропонує потужне рішення: Аліса шифрує своє повідомлення x за допомогою симетричного алгоритму, отримуючи зашифрований текст y. Боб отримує зашифрований текст і розшифровує повідомлення. Розшифрування, таким чином, є зворотним процесом до шифрування (див. Рис. 1.3). У цьому випадку перевага у тому, що якщо у нас є стійкий алгоритм шифрування, то зашифрований текст буде виглядати для Оскара як випадкові біти і не міститиме жодної корисної для нього інформації [3].

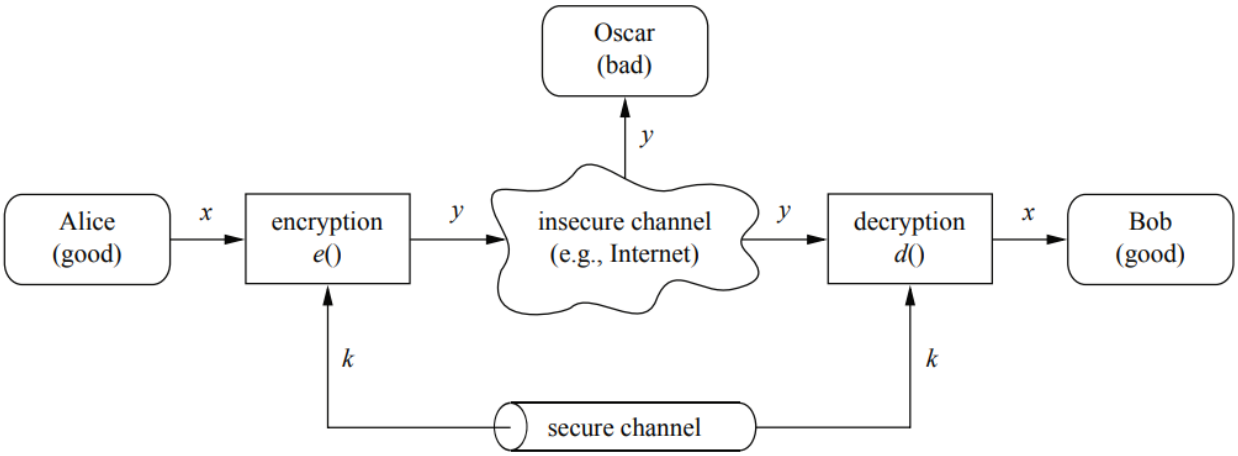

Рисунок 1.3 – Приклад ситуації із застосуванням криптографії

Змінні x, y та k на Рис. 1.2.2 є важливими у криптографії і мають спеціальні назви: x називається відкритим текстом або чистим текстом, y називається шифрованим текстом, k називається ключем, множина всіх можливих ключів називається ключовим простором.

Система потребує захищеного каналу для передачі ключа між Алісою та Бобом. Безпечним каналом, показаним на Рис. 1.2.2, може бути, наприклад, людина, яка перевозить ключ у гаманці між Алісою та Бобом. Це, звичайно, дещо громіздкий метод [7].

Прикладом, де цей метод добре працює, є попередньо розподілені ключі, що використовуються в шифруванні Wi-Fi Protected Access (WPA) в бездротових локальних мережах.

У будь-якому випадку, ключ має бути переданий лише один раз між Алісою та Бобом, а потім може бути використаний для захисту багатьох наступних комунікацій. Важливим і водночас контрінтуїтивним фактом у цій ситуації є те, що і алгоритми шифрування, і алгоритми дешифрування є загальнодоступними. Здавалося б, збереження алгоритму шифрування в таємниці має ускладнити злам всієї системи. Однак, секретні алгоритми також означають неперевірені алгоритми:

Єдиний спосіб з'ясувати, чи є метод шифрування стійким, тобто чи не може його зламати цілеспрямований зловмисник, – це оприлюднити його і

дати можливість іншим криптографам проаналізувати його. Єдине, що слід тримати у таємниці у надійній криптосистемі – це ключ [6].

Зауваження:

1. Звичайно, якщо Оскар отримає ключ, він зможе легко розшифрувати повідомлення, оскільки алгоритм є загальновідомим. Тому важливо відзначити, що проблема безпечної передачі повідомлення зводиться до проблем таємної передачі ключа і безпечного зберігання ключа.

2. У цьому сценарії ми розглядаємо лише проблему конфіденційності, тобто приховування вмісту повідомлення від підслуховування. Пізніше у цій книзі ми побачимо, що є багато інших речей, які можна зробити за допомогою криптографії, наприклад, не дати Оскару непомітно внести зміни до повідомлення (цілісність повідомлення) або гарантувати, що повідомлення дійсно походить від Аліси (автентифікація відправника) [5].

Один з найпростіших методів шифрування тексту – шифр заміни. Історично цей тип шифру використовувався багато разів, і він є гарною ілюстрацією базової криптографії. Ми будемо використовувати шифр заміни для вивчення деяких важливих фактів про довжину ключа і про різні способи злому шифрів. Метою шифру підстановки є шифрування тексту (на відміну від бітів у сучасних цифрових системах). Ідея дуже проста: Ми замінюємо кожну літеру алфавіту на іншу. Приклад методу такого шифрування показано на Рис. 1.4 [7].

A → k
B → d
C → w
...

Рисунок 1.4 – Приклад для шифрування тексту

Наприклад, поп-група ABBA буде зашифрована як kddk

Ми припускаємо, що ми вибираємо таблицю підстановок абсолютно випадковим чином, так що зловмисник не зможе її вгадати. Зверніть увагу, що таблиця підстановок є ключем цієї криптосистеми. Як завжди у симетричній

криптографії, ключ повинен бути розподілений між Алісою та Бобом у безпечний спосіб.

Здається, що текст, показаний на Рис. 1.5 не має особливого сенсу і виглядає як пристойна криптографія. Однак, шифр підстановки він зовсім не безпечний! Давайте розглянемо способи злому цього шифру [8].

```
iq ifcc vqqr fb rdq vfllcq na rdq cfjwhwz hr bnnb
hcc hwwhbsqvqbre hwq vhlq
```

Рисунок 1.5 – Зашифрований текст

Перша атака: перебір або вичерпний пошук за ключем. Атаки грубого перебору базуються на простій концепції: Оскар, зловмисник, має зашифрований текст, отриманий під час прослуховування каналу, і випадково має короткий фрагмент відкритого тексту, наприклад, заголовок файлу, який був зашифрований. Тепер Оскар просто розшифровує перший фрагмент зашифрованого тексту з усіма можливими ключами. Знову ж таки, ключем для цього шифру є таблиця підстановок. Якщо отриманий відкритий текст збігається з коротким фрагментом відкритого тексту, він знає, що знайшов правильний ключ [9].

На практиці атака грубої сили може бути складнішою, оскільки неправильні ключі можуть давати хибнопозитивні результати. Важливо зазначити, що в принципі, атака грубого перебору проти симетричних шифрів завжди можлива. Чи здійсненна вона на практиці, залежить від простору ключів, тобто від кількості можливих ключів, які існують для даного шифру. Якщо тестування всіх ключів на багатьох сучасних комп'ютерах займає занадто багато часу, тобто кілька десятиліть, то шифр є обчислювально стійким до атаки грубої сили [9].

Визначимо простір ключів шифру заміни: Вибираючи заміну для першої літери A, ми випадковим чином вибираємо одну літеру з 26 літер алфавіту (у прикладі вище ми вибрали k). Заміна для наступної літери алфавіту B вибирається випадковим чином з решти 25 літер і т.д. Таким чином, існує

наступна кількість різних таблиць підстановок: ключовий простір шифру підстановок = 26*25*…*3*2*1 = 26! ≈ 2^{88}

Навіть при наявності сотень тисяч високопродуктивних комп'ютерів такий перебір зайняв би кілька десятиліть! Таким чином, у нас виникає спокуса зробити висновок, що шифр підстановки є безпечним. Але це невірно, тому що існує інша, більш потужна атака.

Друга атака: аналіз частоти літер. Спочатку зазначимо, що атака грубої сили зверху розглядає шифр як чорний ящик, тобто ми не аналізуємо внутрішню структуру шифру. Шифр підстановки може бути легко зламаний такою аналітичною атакою [8].

Основним недоліком шифру є те, що кожному символу відкритого тексту завжди відповідає один і той самий символ зашифрованого тексту. Це означає, що статистичні властивості відкритого тексту зберігаються в зашифрованому тексті. Якщо ми повернемося до другого прикладу, то побачимо, що буква q зустрічається в тексті найчастіше. Звідси ми знаємо, що q має бути заміною однієї з частих літер англійської мови [9].

Для практичних атак можна використовувати наступні властивості мови:

1. Визначити частоту кожної літери зашифрованого тексту. Частотний розподіл, часто навіть відносно коротких фрагментів зашифрованого тексту, буде близьким до розподілу для даної мови в цілому. Зокрема, найчастіші літери часто можна легко помітити в зашифрованих текстах. Наприклад, в англійській мові E – найчастотніша літера (близько 13%), T – друга за частотою (близько 9%), A – третя за частотою (близько 8%) і так далі. У Таблиці 1.2.1 наведено розподіл частоти літер в англійській мові.

2. Описаний вище метод можна узагальнити, розглядаючи пари, трійки, четвірки і т.д. символів зашифрованого тексту. Наприклад, в англійській мові (і деяких інших європейських мовах) за літерою Q майже завжди слідує літера U. Ця поведінка може бути використана для виявлення заміни літери Q на літеру U [7].

3. Якщо припустити, що були знайдені розділювачі слів (пропуски) (що буває лише іноді), то часто можна виявити короткі слова, що часто зустрічаються, такі як THE, AND тощо. Виявивши одне з таких слів, ми одразу знаємо три літери (або довжину слова) для всього тексту. На практиці три перераховані вище методи часто комбінують для розкриття шифрів підстановки [4, 7].

Таблиця 1.1 – Розподіл частоти літер в англійській мові

Літера	**Частота**	**Літера**	**Частота**
A	0.0817	N	0.0675
B	0.0150	O	0.0751
C	0.0278	P	0.0193
D	0.0425	Q	0.0010
E	0.1270	R	0.0599
F	0.0223	S	0.0633
G	0.0202	T	0.0906
H	0.0609	U	0.0276
I	0.0697	V	0.0098
J	0.0015	W	0.0236
K	0.0077	X	0.0015
L	0.0403	Y	0.0197
M	0.0241	Z	0.0007

Тобто зашифрований текст, наведений вище, може бути розшифрований за допомогою практичних властивостей мови так, як показано на Рис. 1.6.

```
WE WILL MEET IN THE MIDDLE OF THE LIBRARY AT NOON
ALL ARRANGEMENTS ARE MADE
```

Рисунок 1.6 – Розшифрований текст

Таким чином, Хороші шифри повинні приховувати статистичні властивості зашифрованого відкритого тексту. Символи зашифрованого тексту повинні виглядати випадковими. Крім того, для надійного шифрування недостатньо лише великого ключового простору [9].

1.4 Шифрування зображень

Інформаційні та інтернет-технології стрімко розвиваються, тому в комунікації часто використовуються інтерактивні медіа, наприклад, аудіо, зображення та відео. Зображення складають значну частину мультимедіа [10].

Наприклад, агенції національної безпеки, військові та дипломатичні питання значною мірою покладаються на зображення в комунікації. Оскільки такі зображення можуть містити надзвичайно конфіденційну інформацію, користувачі повинні вживати надзвичайних заходів захисту, зберігаючи їх у ненадійних сховищах. Коли користувачі хочуть надсилати зображення через незахищену мережу, дуже важливо гарантувати повну безпеку. Коротше кажучи, зображення має бути захищене від різноманітних загроз безпеці [11].

Основною метою захисту зображень є забезпечення їхньої цілісності, конфіденційності та автентичності. Шифрування – це один із підходів, який може бути використаний для того, щоб зробити зображення більш безпечними. Шифрування можна визначити як один з процесів, який використовує ключ для перетворення зображень у зашифровані зображення [11].

Крім того, використовуючи підхід дешифрування зашифрованого зображення, який, як правило, є зворотною операцією щодо шифрування, користувачі можуть отримати оригінальне зображення. На Рис. 1.7 зображено процес шифрування, в якому (а) відображається первинне зображення; користувач використовує підхід шифрування для створення секретного зображення; (б) відображається зашифроване зображення, отримане в результаті процедури кодування. У випадку, коли одержувач отримує це приховане зображення, він/вона використовує метод розшифрування для відновлення вихідних даних, як показано на рисунку (в) [12].

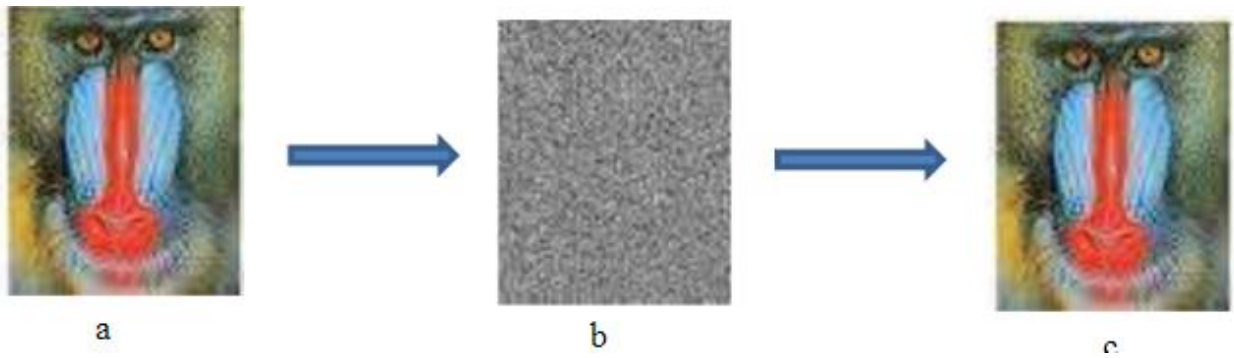

Рисунок 1.7 – Процес шифрування зображення

Існує дві основні категорії криптографії: криптографія з асиметричним і симетричним ключем. У криптографії з секретним ключем, або симетричній криптографії, одержувачі та відправники використовують один і той самий ключ для розшифрування та зашифрування, як показано на Рис. 1.8, тоді як у криптографії з відкритим ключем, або асиметричній криптографії, використовуються різні ключі для розшифрування та зашифрування повідомлень [13].

Для кодування і декодування зображень цей підхід використовує приватний і відкритий ключі відповідно. Ці два ключі відрізняються один від одного, хоча і мають математичний зв'язок, як показано на Рис. 1.9 [14].

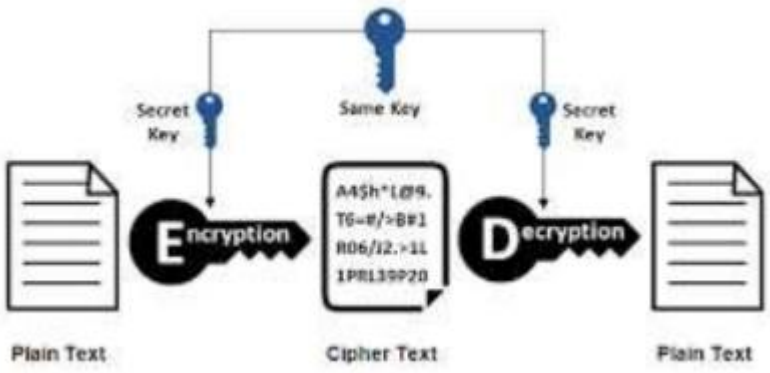

Рисунок 1.8 – Схема симетричного шифрування

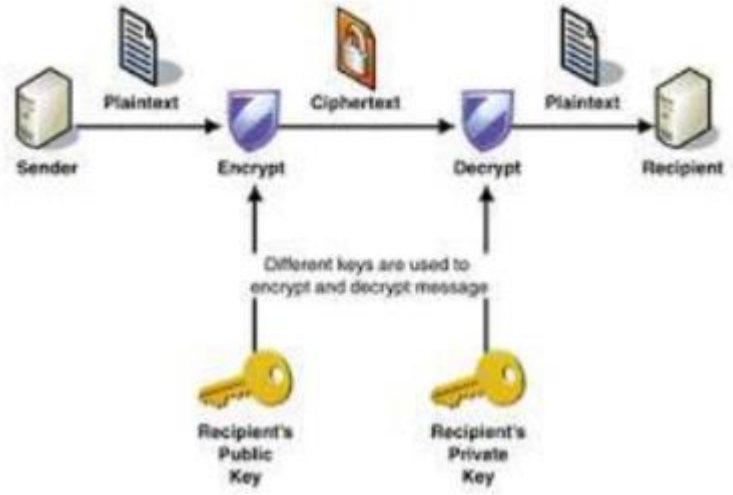

Рисунок 1.9 – Схема асиметричного шифрування

Шифрування зображення за допомогою операції XOR та афінного перетворення. Існує підхід, який використовує 64-бітний ключ у процесі шифрування. Спочатку запропонований підхід використовує афінне перетворення для розосередження пікселів за допомогою чотирьох 8-бітних додаткових ключів. Потім алгоритм розбиває зображення на блоки розміром 2*2 пікселі і виконує операцію XOR над одним з блоків, використовуючи 4 8-бітних додаткових ключа для зміни значення пікселів [13].

Для отримання перетвореного зображення ця система виконує процес перетворення над оригінальним зображенням. Після цього запропонований підхід застосовує операцію XOR до цього зміненого зображення для отримання повного зашифрованого зображення, як показано на рисунку, де показано зашифроване та оригінальне зображення після операції XOR, а також гістограми відмінностей. Результати показують, що запропонований підхід є менш успішним при мінімізації кореляції пікселів, а також має малий ключовий простір [11].

При використанні методу ключа, що використовується в процесі шифрування, такому алгоритму не вистачає достатньої складності. В результаті простої операції XOR та короткого ключа, запропонований підхід не забезпечує прийнятного рівня безпеки для зображень. Такий метод продемонстровано на Рис. 1.10, де (а) відкрите зображення; (б) гістограми

відкритого зображення; (в) зашифроване зображення після операції XOR; (г) гістограми зашифрованого зображення [14].

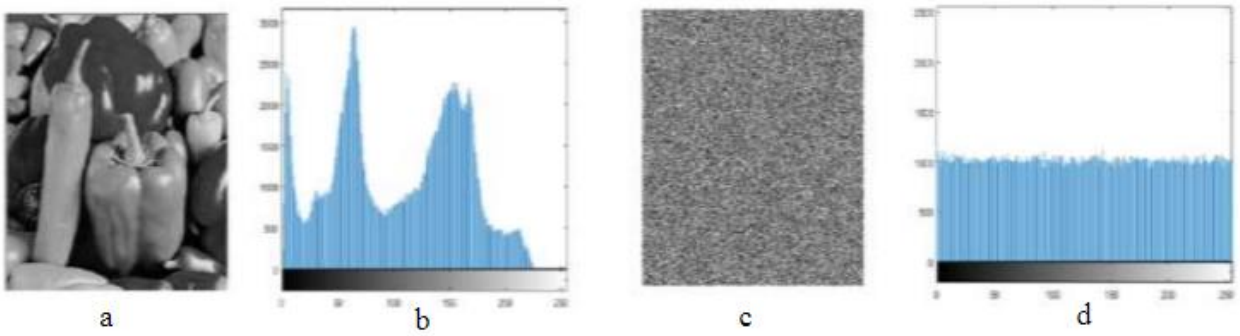

Рисунок 1.10 – Приклад використання за допомогою операції XOR та афінного перетворення

Хаотична система для шифрування зображень. Y. Zhou та L. Bao запропонували нову хаотичну систему, яка складається з 3 унікальних одновимірних хаотичних карт у цьому сценарії. Запропонований метод використовує логістичну карту як контролер для того, щоб визначити, чи потрібно генерувати випадкові послідовності за допомогою синусоїдальної карти та карти намету. Потім алгоритм використовує топологію мережі підстановок і перестановок (SPN) для набуття властивостей дифузії та плутанини [12]. Для великого ключового простору цей підхід використовує 240-бітний ключ. Цей ключ в першу чергу містить всі початкові значення та налаштування параметрів нової хаотичної системи, а також надмірну чутливість до зміни ключа при розшифруванні та зашифруванні. В результаті, запропонований метод пропонує велику захищеність від атак грубої сили, на додаток до надмірної хаотичної поведінки та чутливості ключа [13].

Підхід до шифрування зображень на основі зміщення блоків nxn з наступним міжпіксельним зміщенням атрибуту RGB пікселя. Нідхі Чандра та Амнеш Гоел у своєму дослідженні запропонували ефективний підхід для декомпозиції вихідного зображення на n*n блоків. Після цього використовується алгоритм перетворення для зменшення відстані між пікселями. Цей підхід здебільшого поділяється на два етапи. На початковому

етапі цей алгоритм проводить горизонтальне зміщення блоку перед тим, як перейти до вертикального зміщення блоку. На другому етапі виконується міжпіксельне зміщення значень RGB. Кожен з етапів має маску етапу або ключ, який використовується під час процедури [14].

На Рис. 1.11 (a) – вихідне зображення розміром 300*300 пікселів, (b) – результат застосування до нього горизонтального зсуву блоків. Крім того, після застосування вертикального зсуву блоку до горизонтально зміщеного зображення створюється зображення, яке показано за допомогою (c). Після цього створюється зображення шифру шляхом застосування міжпіксельного зсуву в значеннях RGB, як показано на (d). Результати експериментів показують, що запропонований метод генерує стійке шифроване зображення з використанням вибухового зсуву в значеннях RGB [14].

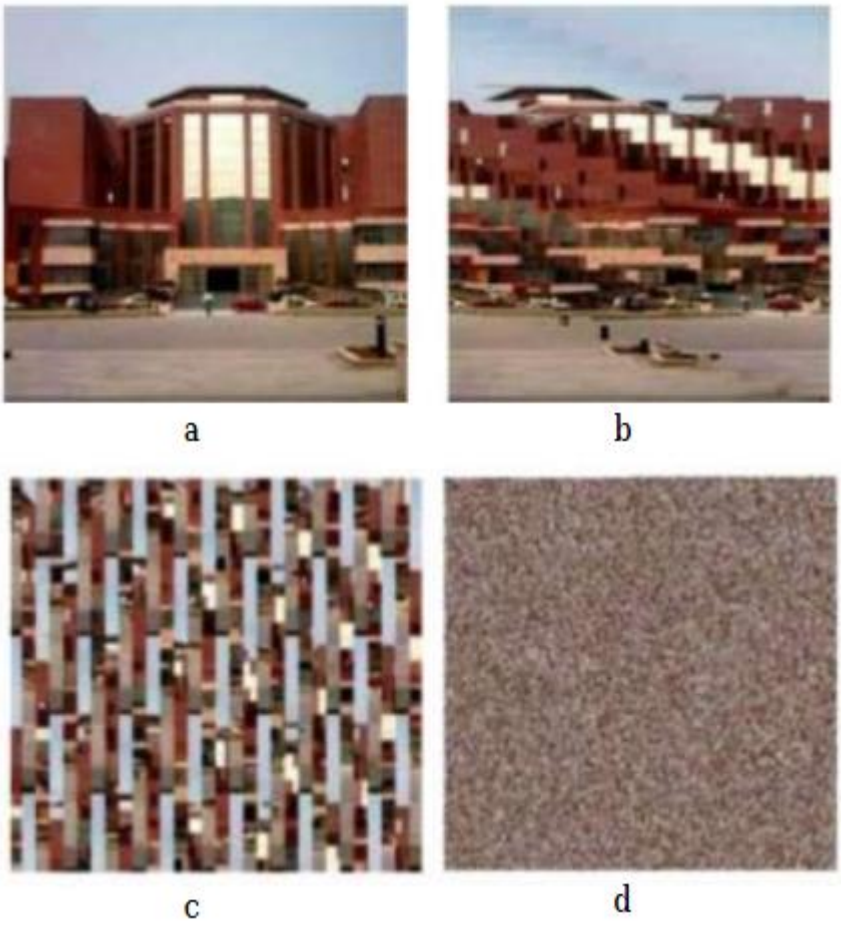

Рисунок 1.11 – Приклад шифрування зображення на основі зміщення блоків

Також було розроблено метод, заснований на перестановці бітового рівня. Щоб задовольнити функції дифузії та плутанини, ця система використовує 2 булеві операції над бітами пікселів: Обертання та XOR. Крім того, метод шифрує зображення шляхом виконання послідовних операцій XOR над усіма бітами пікселів на зображенні, після чого виконується круговий поворот вправо відносно цих бітів. Після цього підхід повторює ці 2 кроки кілька разів, щоб досягти більшої безпеки. Крім того, цей підхід шифрує і розшифровує з використанням одного і того ж секретного ключа [15].

Існує система під назвою (FCIES-Fr-HP), яка показувала і порівнювала зображення шифру з відповідною гістограмою, як показано на Рис. 1.12. Гістограма вхідного зображення, зображена на (d), має певну закономірність, яка підкреслює поняття, пов'язане зі структурою зображення. Властивості рандомізації вихідного значення можна спостерігати в гістограмі зашифрованого зображення з рівномірно і випадково розподіленими значеннями пікселів [14].

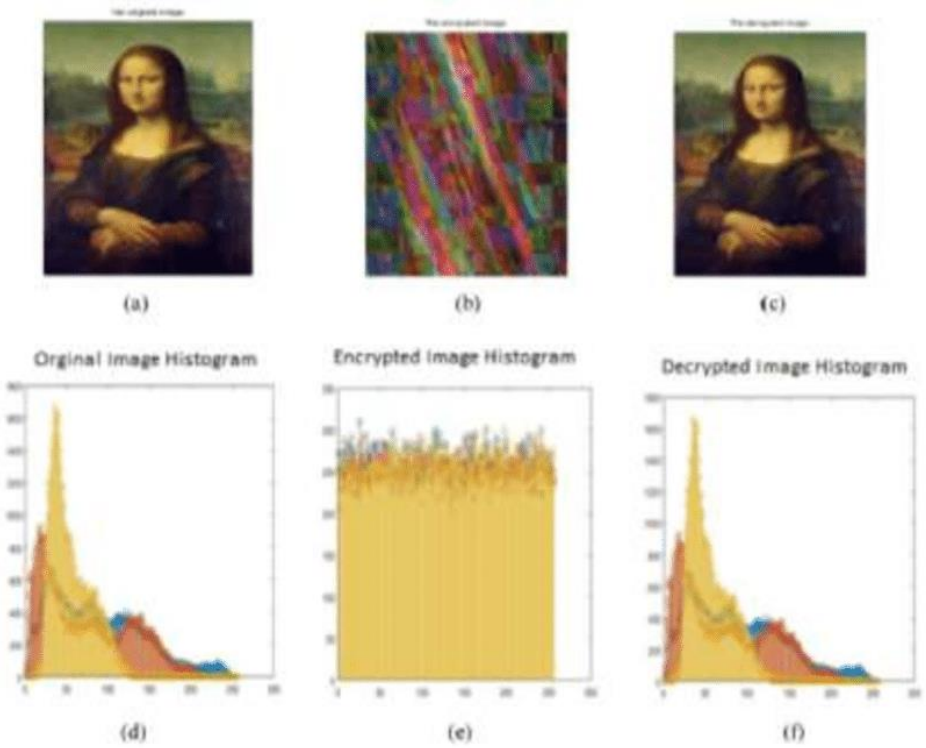

Рисунок 1.12 – Зображення з використанням системи FCIES-Fr-HP із відповідними гістограмами

Проте гістограма зашифрованого зображення, зображена на рисунку (e), демонструє рівномірно розподілені значення пікселів, що ускладнює збір інформації для зловмисника. Крім того, оцінене відхилення між абсолютними матрицями розшифрованого та відкритого зображень, наведене на рисунку (f), демонструє перевагу запропонованої схеми FCIES-Fr-HP з точки зору ефективного дешифрування та шифрування [15].

Метод шифрування зображень на основі 2-шарової хаотичної мережі. Навін Раджпал та Анчал Джайн продемонстрували метод заміщення та дифузії. Для шифрування та дешифрування запропонована система використовує двошаровий хаотичний NN. Цей метод використовує логістичну хаотичну карту для проектування зсувів і ваг NN, а також зовнішній ключ для задавання початкових умов. Загалом, запропонований підхід використовує 80-бітний ключ [16].

Проте, як при декодуванні, так і при кодуванні, хаотична послідовність генерується однаково. Підхід використовує перший шар мережі для дифузії, а другий – для заміщення, причому для розшифрування шари розташовуються у зворотному порядку. В результаті, даний підхід пропонує сильний захист від атак грубої сили, а також атак на основі вибраного або відомого відкритого тексту. Використовуючи рекурсивні клітинні автомати (RCA) та дезоксирибонуклеїнову кислоту (ДНК), Хомаюн, Абдорреза та Расул запропонували новий підхід для шифрування зображень. Шифрування зображень відбувається у два етапи [13]. Логістична карта використовується для клітинного зсуву стовпців і рядків зображення під час перестановки. Після цього RCA і ДНК використовуються для модифікації градації пікселів до нових значень під час дифузії. Цей підхід дав ідеальне значення ентропії 7,9994 і коефіцієнт кореляції 0,0001, що свідчить про хороші результати. Крім того, було показано, що запропонований підхід підвищує рівень стійкості до ряду атак шляхом шифрування зображень у бажаний спосіб [14].

Сомдіп Дей представив комбінований метод. Запропонований підхід базується на трьох криптографічних методах: Розширений шифр Хілла

(Extended Hill Cipher) (див. Рис. 1.13), який заснований на множенні матриць, обертання і реверс бітів та модифікована MSA Randomization.

Крім того, запропонована методика шифрує зображення в чотири етапи. На першому етапі алгоритм генерує відмінне число від симетричного ключа. Залежно від довжини симетричного ключа, на 2-му етапі виконується обертання та реверс бітів. На третьому етапі система використовує алгоритм розширеного шифру Хілла для шифрування. На четвертому етапі для підстановки використовується модифікований метод рандомізації MSA. Завдяки додатковій рандомізації емпіричні результати підтвердили, що підхід кодування SD-AEI є домінуючим над SD-EI [14].

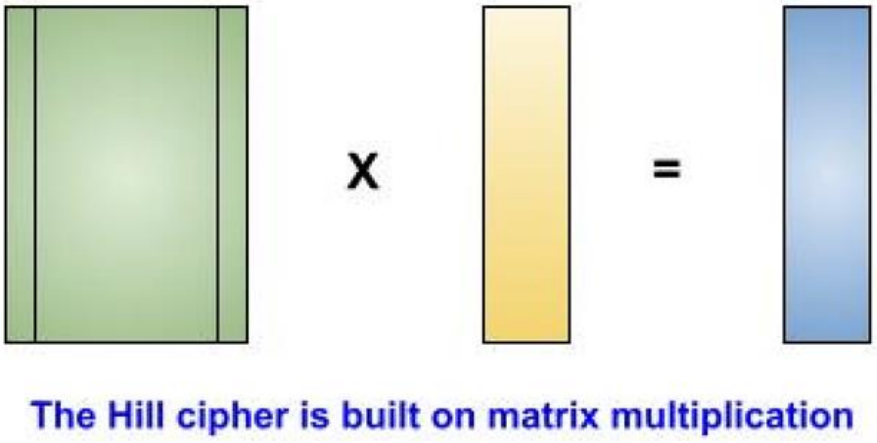

Рисунок 1.13 – Розширений шифр Хілла

Цифрове шифрування на основі багатовимірної хаотичної системи та розташування пікселів. Hazern M. Al-Najar запропонував підхід до шифрування зображень на основі багатовимірної хаотичної функції в даній роботі. Також ця система коригує значення пікселів шляхом їх розсіювання. Запропоновані підходи використовують два підходи заміни для модифікації значення пікселів та 2 операції скремблювання для розсіювання пікселів. Крім того, запропонований курс шифрує зображення наступним чином: алгоритм використовує 1-шу схему заміни на основі значення індексу стовпця, потім використовує перший підхід скремблювання на основі площин X, Y та Z формули Росслера [13].

Потім система виконує другий метод заміни, заснований на значенні індексу рядка, за яким слідує 2-й план перемішування, заснований на площинах X, Y і Z. Спостереження підтвердили, що запропонований підхід має чутливість до початкових умов, а також є невразливим до атак грубої сили та інших типів атак завдяки великому простору ключів, який становить 10^{45} [14].

З іншої ж сторони, шифрування зображень за допомогою алгоритму Triple DES із використанням бібліотеки CryptoJS є складним процесом, який передбачає кілька ключових етапів. Перед тим як застосувати шифрування, зображення необхідно перетворити у формат, що може бути представленим як текст. Для цього зазвичай використовують кодування Base64, яке перетворює двійкові дані зображення (байти) у текстовий рядок, який складається з символів ASCII (див. Рис. 1.14). Це дозволяє оцифрувати зображення у текстовий вигляд, придатний для подальшої обробки алгоритмами шифрування [12].

Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char	Decimal	Hex	Char
0	0	[NULL]	32	20	[SPACE]	64	40	@	96	60	`
1	1	[START OF HEADING]	33	21	!	65	41	A	97	61	a
2	2	[START OF TEXT]	34	22	"	66	42	B	98	62	b
3	3	[END OF TEXT]	35	23	#	67	43	C	99	63	c
4	4	[END OF TRANSMISSION]	36	24	$	68	44	D	100	64	d
5	5	[ENQUIRY]	37	25	%	69	45	E	101	65	e
6	6	[ACKNOWLEDGE]	38	26	&	70	46	F	102	66	f
7	7	[BELL]	39	27	'	71	47	G	103	67	g
8	8	[BACKSPACE]	40	28	(	72	48	H	104	68	h
9	9	[HORIZONTAL TAB]	41	29	)	73	49	I	105	69	i
10	A	[LINE FEED]	42	2A	*	74	4A	J	106	6A	j
11	B	[VERTICAL TAB]	43	2B	+	75	4B	K	107	6B	k
12	C	[FORM FEED]	44	2C	,	76	4C	L	108	6C	l
13	D	[CARRIAGE RETURN]	45	2D	-	77	4D	M	109	6D	m
14	E	[SHIFT OUT]	46	2E	.	78	4E	N	110	6E	n
15	F	[SHIFT IN]	47	2F	/	79	4F	O	111	6F	o
16	10	[DATA LINK ESCAPE]	48	30	0	80	50	P	112	70	p
17	11	[DEVICE CONTROL 1]	49	31	1	81	51	Q	113	71	q
18	12	[DEVICE CONTROL 2]	50	32	2	82	52	R	114	72	r
19	13	[DEVICE CONTROL 3]	51	33	3	83	53	S	115	73	s
20	14	[DEVICE CONTROL 4]	52	34	4	84	54	T	116	74	t
21	15	[NEGATIVE ACKNOWLEDGE]	53	35	5	85	55	U	117	75	u
22	16	[SYNCHRONOUS IDLE]	54	36	6	86	56	V	118	76	v
23	17	[END OF TRANS. BLOCK]	55	37	7	87	57	W	119	77	w
24	18	[CANCEL]	56	38	8	88	58	X	120	78	x
25	19	[END OF MEDIUM]	57	39	9	89	59	Y	121	79	y
26	1A	[SUBSTITUTE]	58	3A	:	90	5A	Z	122	7A	z
27	1B	[ESCAPE]	59	3B	;	91	5B	[	123	7B	{
28	1C	[FILE SEPARATOR]	60	3C	<	92	5C	\	124	7C	|
29	1D	[GROUP SEPARATOR]	61	3D	=	93	5D	]	125	7D	}
30	1E	[RECORD SEPARATOR]	62	3E	>	94	5E	^	126	7E	~
31	1F	[UNIT SEPARATOR]	63	3F	?	95	5F	_	127	7F	[DEL]

Рисунок 1.14 – Таблиця символів ASCII

Алгоритм Triple DES (3DES), як було описано вище, працює шляхом трикратного застосування стандартного DES для шифрування даних, що робить процес шифрування значно складнішим і безпечнішим. Спочатку зображення, представлене у вигляді тексту (наприклад, Base64), шифрується за допомогою трьох різних ключів або одного ключа, що використовується тричі в різних послідовностях. Triple DES складається з трьох основних кроків: шифрування з першим ключем, дешифрування з другим, і знову шифрування з третім ключем. Такий підхід забезпечує надійніший захист даних, оскільки навіть якщо один ключ виявиться скомпрометованим, інші етапи забезпечать додатковий рівень безпеки [4].

Після шифрування текстовий рядок, що представляє зашифроване зображення, може бути збережений або переданий у зашифрованому вигляді. Це дає можливість захистити вміст зображення від несанкціонованого доступу під час його передачі через інтернет або зберігання на серверах. Зашифроване зображення виглядає як набір символів, які не мають очевидного сенсу без наявності ключа для дешифрування [3].

Щоб розшифрувати зображення, необхідно мати доступ до того самого ключа, який використовувався для шифрування, а також застосувати зворотні операції шифрування. Спочатку здійснюється дешифрування із використанням останнього ключа, потім шифрування другим ключем, і, нарешті, знову дешифрування першим ключем. Цей процес повертає вихідний текстовий рядок, який містить закодовані дані зображення у форматі Base64. Після цього текст можна декодувати назад у двійковий формат для отримання оригінального зображення.

Використання Triple DES у для шифрування зображень дозволяє гарантувати конфіденційність візуальної інформації під час її обробки чи передачі. Оскільки зображення перетворюється у текстовий вигляд і шифрується на стороні сервера або клієнта, це забезпечує захист від перехоплення або несанкціонованого доступу. Технології шифрування, такі як Triple DES, використовуються у багатьох системах, що потребують

підвищеного рівня безпеки даних, включно із фінансовими системами та іншими додатками, де необхідний захист конфіденційної інформації [2].

Хоча цей алгоритм є значно повільнішим порівняно з сучасними методами шифрування, такими як AES, він залишається поширеним у випадках, коли важливі стандарти безпеки та сумісності з уже існуючими системами.

1.5 Постановка завдання

Метою кваліфікаційної роботи магістра є розробка веб-застосунку для шифрування та розшифрування зображень із використанням алгоритму Triple DES, який забезпечуватиме надійний захист цифрових зображень від несанкціонованого доступу.

Для досягнення поставленої мети необхідно виконати такі завдання:

- провести аналіз існуючих методів шифрування зображень та обґрунтувати вибір алгоритму triple des для реалізації;
- розробити архітектуру веб-застосунку, що дозволяє інтеграцію в існуючі системи захисту інформації;
- реалізувати алгоритм шифрування та розшифрування зображень з урахуванням сучасних підходів до веб-розробки;
- забезпечити зручний і зрозумілий інтерфейс користувача для взаємодії з веб-застосунком;
- провести тестування розробленого веб-застосунку з метою перевірки його ефективності, надійності та зручності використання.

Практичний результат роботи полягатиме у створенні інструменту, який може бути використаний як приватними користувачами, так і організаціями для захисту конфіденційних візуальних даних. Розроблений веб-застосунок має потенціал для подальших удосконалень та стане цінним інструментом для забезпечення кібербезпеки в сучасному інформаційному середовищі.

РОЗДІЛ 2
ПРОЕКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Функціональні вимоги

Функціональні вимоги для веб-застосунку для шифрування зображень алгоритмом Triple DES, який розроблятиметься в межах даної дипломної роботи, можуть включати в себе такі пункти:

1. Завантаження зображення: користувач має можливість завантажити зображення різних форматів (JPEG, PNG, BMP) на веб-застосунок для подальшого шифрування;

2. Шифрування зображення: після завантаження зображення користувач може ввести ключ шифрування довжиною 192 біти для алгоритму Triple DES.

Застосунок повинен забезпечувати процес шифрування зображення відповідно до алгоритму Triple DES. Зашифроване зображення повинно бути доступне для завантаження у зашифрованому вигляді;

3. Дешифрування зображення: користувач може завантажити зашифроване зображення і ввести відповідний ключ для його дешифрування. Застосунок повинен коректно розшифровувати зображення за допомогою алгоритму Triple DES і надавати можливість завантажити розшифроване зображення [3];

4. Перегляд результатів шифрування: після шифрування користувач може переглянути зашифрований вихідний результат. Він не повинен мати зрозумілої інформації, яка могла б бути відтворена без дешифрування;

5. Перегляд результатів дешифрування: після процесу дешифрування користувач повинен мати можливість переглянути вхідне (розшифроване) зображення та завантажити його;

6. Інтерфейс користувача: веб-застосунок повинен мати зручний та інтуїтивно зрозумілий інтерфейс, що дозволяє легко взаємодіяти з функціями шифрування зображень і дешифрування;

7. Повідомлення про помилки: у разі некоректного завантаження файлу або неправильного введення ключа, система повинна інформувати користувача через повідомлення про помилку та надавати рекомендації щодо виправлення [3].

Ці функціональні вимоги забезпечують необхідний функціонал для коректної роботи веб-застосунку з шифруванням зображень за допомогою алгоритму Triple DES, який розроблятиметься в межах даної дипломної роботи.

2.2 Нефункціональні вимоги

Нефункціональні вимоги для веб-застосунку для шифрування зображень алгоритмом Triple DES, який розроблятиметься в межах даної дипломної роботи, можуть включати в себе такі пункти:

1. Продуктивність: веб-застосунок повинен шифрувати та дешифрувати зображення розміром до 10 МБ за час не більше хвилини на середньостатистичному сервері;

2. Масштабованість: система повинна легко масштабуватись для обробки великої кількості запитів при збільшенні навантаження;

3. Надійність: застосунок повинен забезпечувати стійку роботу при різних навантаженнях та коректно обробляти помилки (наприклад, невірні файли або ключі). У разі збою користувач повинен отримати детальну інформацію про причину помилки та рекомендації для її виправлення [13];

4. Зручність використання: ынтерфейс користувача повинен бути простим та інтуїтивно зрозумілим. Всі елементи інтерфейсу повинні бути доступними та добре структурованими для легкого користування;

5. Сумісність: веб-застосунок повинен працювати коректно на основних веб-браузерах (Google Chrome, Mozilla Firefox, Microsoft Edge, Safari) і бути сумісним із різними операційними системами (Windows, macOS, Linux).

Система повинна підтримувати стандартні формати зображень (JPEG, PNG, BMP) [14];

6. Модульність: архітектура застосунку повинна бути модульною, що дозволить легко замінювати або оновлювати окремі компоненти (наприклад, бібліотеки для роботи з зображеннями або шифруванням) без значних змін у всьому коді. Код повинен бути організований так, щоб у майбутньому було легко додавати нові функції (наприклад, підтримку інших алгоритмів шифрування);

7. Обслуговуваність: код програми повинен бути добре задокументований, щоб інші розробники могли легко підтримувати та оновлювати систему;

8. Ефективність використання ресурсів: веб-застосунок повинен ефективно використовувати ресурси серверів і клієнтів (ЦП, пам'ять), мінімізуючи споживання системних ресурсів, особливо при обробці великих зображень [14].

Ці нефункціональні вимоги забезпечують надійність, безпеку, зручність використання і продуктивність веб-застосунку, а також гарантують, що система зможе масштабуватися та підтримувати сучасні стандарти розробки.

2.3 Архітектура програмного забезпечення

Веб-застосунок для шифрування та дешифрування зображень алгоритмом Triple DES побудований на основі клієнт-серверної архітектури. Основні елементи архітектури включають frontend (інтерфейс користувача), backend (обробка даних) та ресурси для шифрування/дешифрування. Оскільки в коді немає серверної частини, вся логіка обробки даних виконується на клієнтській стороні (в браузері) за допомогою JavaScript.

1. Клієнтська частина (Frontend)

Frontend відповідає за взаємодію користувача з системою, включає сторінки HTML, CSS для стилізації, та JavaScript для шифрування/дешифрування [16].

Основні файли:

- dashboard.html: початкова сторінка, що містить кнопку для переходу на сторінки шифрування (encrypt.html) та дешифрування (decrypt.html). Використовує css для візуального оформлення;
- «style_dashboard.css» — стилі для сторінки dashboard.html;
- «decrypt.html» – сторінка для завантаження зашифрованого зображення та введення секретного ключа для дешифрування;
- «style.css» — стилі для цієї сторінки;
- «crypto-js.js» — бібліотека для шифрування/дешифрування;
- «filesaver.js» — бібліотека для збереження розшифрованого зображення;
- encrypt.html – сторінка для завантаження зображення, введення секретного ключа і шифрування.

2. Логіка шифрування та дешифрування: JavaScript бібліотека «crypto-js.js» для реалізації алгоритму Triple DES. Вона використовується для шифрування зображень на сторінці «encrypt.html» та дешифрування на сторінці «decrypt.html». Шифрування: перетворює зображення у зашифрований текст за допомогою введеного ключа. Дешифрування: перетворює зашифрований текст у вихідне зображення за допомогою того ж ключа;

3. Функціонал завантаження файлів: на сторінках «encrypt.html» та «decrypt.html» реалізовано завантаження зображень через HTML елементи «<input type="file">». Після вибору файлу його зміст завантажується та обробляється через JavaScript (за допомогою «FileReader») [17];

4. Збереження файлів (FileSaver.js): «FileSaver.js» використовується для завантаження розшифрованого зображення на пристрій користувача після

дешифрування. Бібліотека створює посилання для збереження файлу і дозволяє користувачу завантажити його локально;

5. Користувацький інтерфейс (UI) оформлений за допомогою каскадних таблиць стилів CSS. CSS файли включають в себе:

- «style_dashboard.css» відповідає за стиль сторінки «dashboard.html», робить інтерфейс візуально привабливим і зручним для користувачів;
- «style.css» застосовується на сторінках шифрування та дешифрування, забезпечуючи однаковий стиль для всіх елементів, таких як кнопки, поля вводу та зона завантаження файлів.

6. Потоки даних в програмному забезпеченні складаються із функцій шифрування та дешифрування та показані за допомогою Рис. 2.1-2.2.

- шифрування: користувач завантажує зображення, вводить ключ шифрування, після чого зображення шифрується у текст за допомогою Triple DES і надається для завантаження;
- дешифрування: користувач завантажує зашифроване зображення (у вигляді тексту), вводить ключ для дешифрування, після чого зображення відновлюється у своєму початковому вигляді і може бути завантажене.

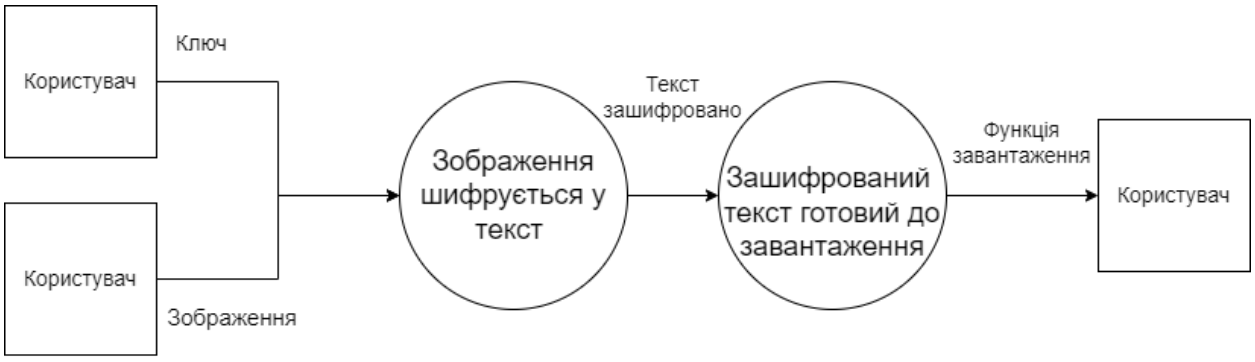

Рисунок 2.1 – Потік даних ПЗ для шифрування

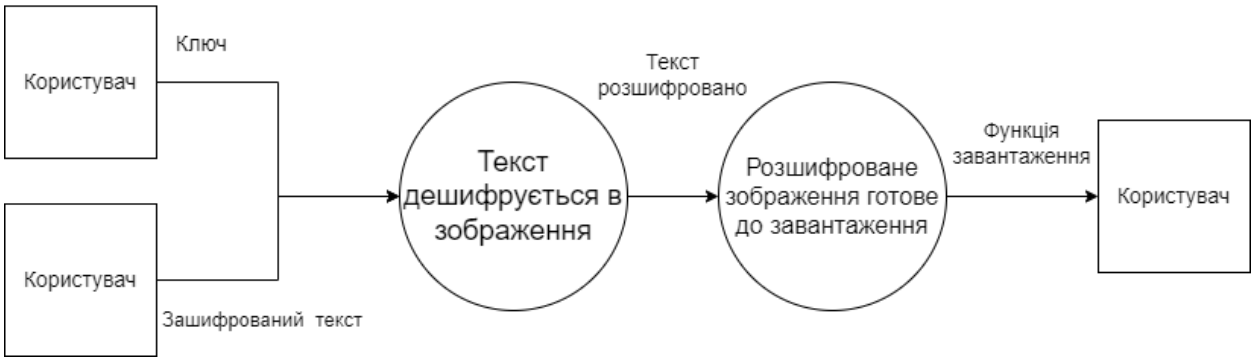

Рисунок 2.2 – Потік даних ПЗ для розшифрування

2.4 Вибір мови програмування

При розробці веб-застосунку для шифрування зображень за допомогою алгоритму Triple DES, вибір мови програмування відіграє важливу роль. Ось кілька переваг використання мови програмування JavaScript, які можуть бути корисні для цього проєкту:

- клієнтська обробка: JavaScript дозволяє виконувати шифрування і дешифрування безпосередньо в браузері, що знижує навантаження на сервер та покращує швидкість роботи програми;
- багато бібліотек: Є безліч бібліотек, таких як CryptoJS, які реалізують криптографічні алгоритми, включаючи Triple DES. Це спрощує процес імплементації шифрування;
- широка підтримка браузерами: JavaScript підтримується всіма сучасними браузерами, що робить його універсальним рішенням для веб-застосунків.

JavaScript – це легка, кросплатформенна, однопотокова та інтерпретована компільована мова програмування. Вона також відома як мова сценаріїв для веб-сторінок. Вона добре відома для розробки веб-сторінок, і багато небраузерних середовищ також використовують її [18].

JavaScript є слабкотипізованою мовою (динамічно типізованою). JavaScript можна використовувати як для клієнтських, так і для серверних розробок. JavaScript є як імперативною, так і декларативною мовою. JavaScript

містить стандартну бібліотеку об'єктів, таких як масив, дата і математика, а також основний набір мовних елементів, таких як оператори, керуючі структури та оператори.

На стороні клієнта: JavaScript надає об'єкти для керування браузером та його об'єктною моделлю документа (DOM). Наприклад, клієнтські розширення дозволяють програмі розміщувати елементи на HTML-формі та реагувати на події користувача, такі як клацання мишею, введення даних з форми та навігація сторінками. Корисними бібліотеками для клієнтської частини є AngularJS, ReactJS, VueJS та багато інших [19].

Серверна частина: JavaScript надає об'єкти, необхідні для запуску JavaScript на сервері. Розширення на стороні сервера дозволяють додатку взаємодіяти з базою даних і забезпечують безперервність інформації від одного виклику до іншого додатку або виконувати маніпуляції з файлами на сервері. Найвідомішим фреймворком, який є корисним в наші дні, є node.js.

Імперативна мова – у цьому типі мови нас більше турбує те, як це має бути зроблено. Вона просто контролює потік обчислень. Процедурний підхід до програмування, об'єктно-орієнтований підхід підпадає під це поняття, оскільки в режимі очікування асинхронного виконання ми думаємо про те, що робити далі після виклику асинхронного виконання.

Декларативне програмування – у цьому типі мови нас цікавить, як це має бути зроблено, в основному тут потрібні логічні обчислення. Її основна мета – описати бажаний результат без прямої вказівки, як його отримати, як це робить функція зі стрілкою [17].

2.5 Вибір середовища розробки

Для розробки веб-застосунку для шифрування зображень алгоритмом Triple DES обрано середовище розробки під назвою Sublime Text.

Sublime можна завантажити з офіційного сайту. Наразі доступна бета-версія 3 для OS X, Windows та Ubuntu. Існує також портативна версія, яку

можна легко запустити з флешки. Sublime Text можна завантажити безкоштовно, проте офіційно це не так. Після ознайомлювального періоду є ліцензію для подальшого використання. Однак на самій сторінці завантаження зазначено, що немає примусового обмеження часу для ознайомлення, що робить його більш добровільним. Після вибору бажаної версії завантаження відбувається швидко. Як вже згадувалося в огляді Atom, Sublime має досить невеликий розмір. Інсталяційний архів складає всього 8 МБ, а після встановлення програма займає лише 22 МБ з невеликим [12].

За загальним дизайном Atom і Sublime Text дуже схожі. Це не дивно, оскільки багато хто вважає, що Atom насправді базується на Sublime. З цієї причини також не дивно, що обидва мають автозавершення коду і підсвічування синтаксису. Хоча насправді, мабуть, жоден сучасний редактор коду не може обійтися без нього. Sublime має вбудовану підтримку декількох десятків мов програмування і відповідно кодує елементи коду. Редактор коду навіть попередить вас про помилки синтаксичного аналізу в режимі реального часу [11].

На додаток до цього, Sublime постачається з автозавершенням коду – в тому числі для створених користувачем змінних – і згортанням коду. Останнє робить великі фрагменти коду більш компактними для перегляду, приховуючи їх частини.

Як і Atom, Sublime Text можна повністю налаштувати за допомогою плагінів. Швидко інтегрований менеджер пакетів дозволяє користувачам знаходити, встановлювати, оновлювати і видаляти плагіни безпосередньо з редактора – зазвичай без перезапуску. Користувачі можуть завантажувати пакунки з Github та BitBucket, а також з PackageControl.io. Останній є власним репозиторієм Sublime, де спільнота, що стоїть за редактором коду, публікує безліч плагінів для розширення його можливостей. Загалом, там доступно понад 3800 пакунків, впорядкованих за новими, трендовими, нещодавно оновленими, популярними та іншими тегами. З їхньою допомогою можна перетворити Sublime з простого редактора коду на повноцінне середовище

розробки і створити саме те середовище, яке вам потрібно. Для Sublime Text є кілька чудових пакетів [11, 20].

Те, що швидко стає очевидним при використанні Sublime, – це швидкість роботи програми. Побудований на C/C++ та Python, редактор коду дуже легкий. Фактично, при виборі його зі стартового меню Windows, він завантажується майже миттєво. На противагу цьому, Atom займає кілька секунд, щоб з'явитися на екрані. Ця різниця також помітна при роботі з великими файлами. Sublime Text може обробляти набагато більші обсяги даних без проблем. У моїх тестах я не мав жодних проблем, а також не знайшов жодних скарг на проблеми з продуктивністю в Інтернеті. Насправді, його стабільність є однією з найпоширеніших причин, чому люди надають перевагу Sublime Text перед іншими редакторами. Зависання і збої у ньому трапляються вкрай рідко [11].

Sublime – один з найпопулярніших редакторів коду, і на те є вагома причина. Програма блискавично швидка, стабільна і зріла. Вона також має безліч дійсно корисних функцій і безліч чудових деталей. На додаток до цього, завдяки модульному підходу та розширюваності, Sublime Text може бути будь-чим для будь-кого. Редактор підходить для розробників різних рівнів кваліфікації та дисциплін. Хоча в цілому користувацький інтерфейс схожий на Atom, Sublime відчувається набагато більш впорядкованим і зрозуміло, що розробники редактора вклали в нього багато праці. Здається, що йому не завадило б використати трохи підходу WordPress/Atom і, можливо, додати ще кілька людей до своєї команди. Таким чином, прогрес був би швидшим, і Sublime міг би більше закріпитися в області редакторів коду [20].

Обираючи Sublime Text середовищем для розробки веб-застосунку для шифрування зображень за допомогою алгоритму Triple DES, можна виділити такі переваги:

1. Легкість і швидкість: Sublime Text працює дуже швидко навіть з великими проєктами. Це корисно для розробки веб-застосунків, де обробляються великі файли зображень і коду;

2. Підтримка багатьох мов програмування: Sublime Text підтримує HTML, CSS, JavaScript, що робить його зручним для роботи з файлами веб-застосунку, такими як dashboard.html, decrypt.html, encrypt.html, style.css, та скриптами для шифрування/дешифрування;

3. Плагіни та пакети: Sublime має широку підтримку плагінів. Можна легко встановити плагіни для JavaScript, шифрування, форматування коду та навіть плагіни для безпеки, що спрощує розробку та тестування криптографічних алгоритмів, таких як Triple DES [11];

4. Автодоповнення коду: Sublime Text пропонує функцію автодоповнення, яка спрощує роботу з кодом, допомагаючи швидше писати і редагувати файли, що містять шифрування та інші складні операції;

5. Можливість роботи з кількома файлами одночасно: Sublime підтримує поділ екрану і вкладки, що дозволяє працювати з кількома файлами проекту одночасно (HTML, CSS, JS), пришвидшуючи розробку;

6. Пошук і навігація: Sublime має потужні інструменти для пошуку та навігації по проєкту, що полегшує швидке знаходження необхідних фрагментів коду, особливо в великих проєктах [13];

7. Редагування кількох рядків: Можливість редагувати кілька рядків одночасно (multi-cursor editing) пришвидшує виправлення помилок або зміни в структурі коду, що є корисним при роботі з великими скриптами шифрування [20].

РОЗДІЛ 3
РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Алгоритми шифрування DES, Triple DES (3DES)

Шифр DES є варіантом базового шифру Фейстеля (див. Рис. 3.1), названого на честь Х. Фейстеля, який працював в IBM і виконав одні з перших невійськових досліджень алгоритмів шифрування. Цікавою властивістю шифру Фейстеля є те, що кругла функція є інвертованою незалежно від вибору функції у клітинці, позначеній F.

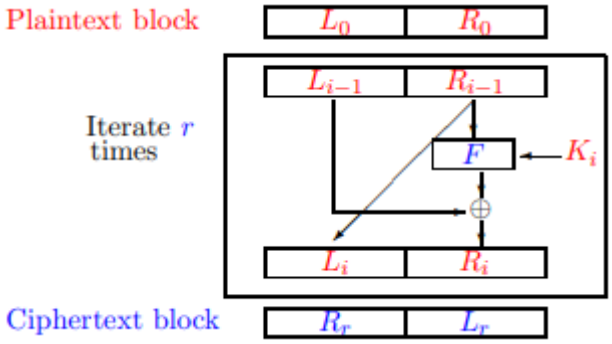

Рисунок 3.1 – Принцип роботи шифру Фейстеля

Це означає, що в шифрі Фейстеля ми дещо спростили конструкцію, оскільки – ми можемо вибрати будь-яку функцію для функції F, і ми все одно отримаємо функцію шифрування, яка може бути інвертована за допомогою секретного ключа, – той самий код/схему можна використовувати для функцій шифрування і розшифрування.

Для розшифрування потрібно лише використовувати круглі ключі у зворотному порядку. Звичайно, щоб отримати надійний шифр, ми повинні подбати про те, як генеруються раундові ключі, скільки раундів потрібно зробити, як визначається функція F. Робота над DES була розпочата на початку 1970-х років командою в IBM, до якої входив Фейстель. Спочатку він базувався на більш ранньому шифрі IBM під назвою Люцифер, але було відомо, що Агентство національної безпеки (АНБ) внесло деякі зміни в його

конструкцію. Протягом багатьох років це змушувало конспірологів вважати, що АНБ заклало лазівку в конструкцію функції F [12].

Однак зараз широко визнано, що зміни, внесені АНБ, були зроблені для того, щоб зробити шифр більш захищеним. Зокрема, зміни, внесені АНБ, зробили шифр стійким до диференціального криптоаналізу – методу, який не був відкритий у відкритому дослідницькому середовищі до 1980-х років. DES також відомий як алгоритм шифрування даних DEA в документах Американського національного інституту стандартів, ANSI.

Міжнародна організація зі стандартизації ISO посилається на DES під назвою DEA-1. Він є світовим стандартом вже більше двадцяти років і є першим загальнодоступним алгоритмом, який має «офіційний статус». Таким чином, він знаменує собою важливий крок на шляху від криптографії, яка була суто військовою сферою, до інструменту для широких мас.

Основні властивості шифру DES полягають у тому, що він є варіантом конструкції шифру Фейстеля з

- кількістю раундів r – 16;
- довжиною блоку n – 64 біти;
- довжиною ключа – 56 біт;
- довжиною раундових ключів K1,..., K16 по 48 біт кожен [12].

Зауважимо, що довжина ключа 56 біт є недостатньою для багатьох сучасних застосувань, тому часто використовують DES з трьома ключами і трьома ітераціями основного шифру. Такий варіант називається Triple DES або 3DES (див. Рис. 3.2) [12].

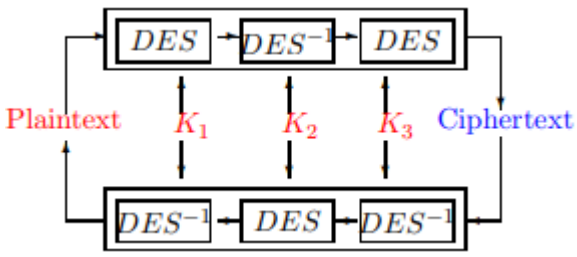

Рисунок 3.2 – Алгоритм Triple DES

У 3DES довжина ключа дорівнює 168. Існує ще один спосіб триразового використання DES, але з використанням двох ключів замість трьох, що призводить до довжини ключа 112. У цій двоключовій версії 3DES використовується базова структура 3DES, але перший і третій ключі є рівними. Однак, двоключовий 3DES не настільки безпечний, як може здатися на перший погляд [12].

По суті, DES – це шифр Фейстеля з 16 раундами (див. Рис. 3.3), за винятком того, що до і після основної ітерації Фейстеля виконується перестановка. Зверніть увагу, як два блоки міняються місцями перед тим, як пройти через фінальну зворотну перестановку. Здається, що ця перестановка не призводить до жодних змін у безпеці, і люди часто задавалися питанням, навіщо вона потрібна. Одна з відповідей, яку дав один з членів команди розробників, полягала в тому, що ця перестановка була зроблена для того, щоб оригінальну реалізацію було легше розмістити на друкованій платі [13].

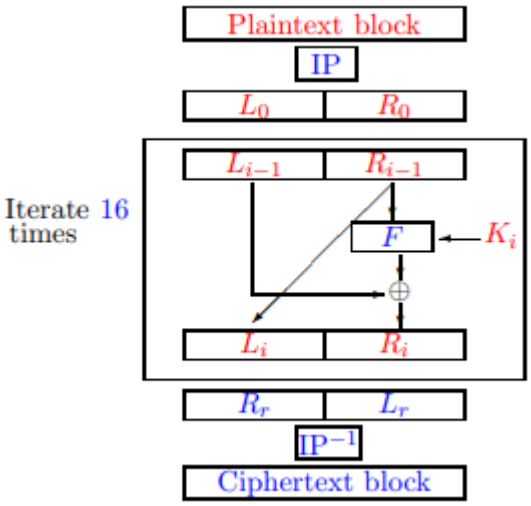

Рисунок 3.3 – DES як шифр Фейстеля

Отже, шифр DES працює з 64 бітами відкритого тексту наступним чином:

- виконати початкову перестановку;
- розділити блоки на ліву і праву половину;
- виконати 16 раундів ідентичних операцій;
- з'єднати половинки блоків назад разом;

- виконати фінальну перестановку.

Фінальна перестановка є зворотною до початкової, це дозволяє використовувати одне і те ж апаратне/програмне забезпечення для шифрування і розшифрування. Розклад ключів забезпечує 16 круглих ключів довжиною 48 біт, вибираючи 48 біт з 56-бітового основного ключа.

Опишемо роботу функції F. У кожному раунді DES вона складається з наступних етапів:

- перестановка з розширенням: права половина 32 біт розширюється і переставляється до 48 біт. Це допомагає поширенню будь-якого відношення вхідних бітів до вихідних. Перестановка розширення (яка відрізняється від початкової перестановки) була обрана таким чином, щоб один біт на вході впливав на дві заміни на виході, за допомогою S-подібних блоків нижче. Це допомагає поширити залежності і створює лавинний ефект (невелика різниця між двома відкритими текстами призведе до дуже великої різниці у відповідних зашифрованих текстах) [13];
- додавання круглого ключа: 48-бітовий результат перестановки розширення додається до круглого ключа, який також має довжину 48 біт. Зауважте, що це єдине місце, де в алгоритмі використовується круглий ключ;
- розщеплення: отримане 48-бітове значення розбивається на вісім лотів по шість біт;
- s-Box: кожне шестибітне значення передається в один з восьми різних S-Box (Substitution Box) для отримання чотирибітового результату. S-Box представляють нелінійний компонент алгоритму DES, і їх дизайн є основним фактором, що впливає на безпеку алгоритму. Кожен S-Box являє собою таблицю пошуку з чотирьох рядків і шістнадцяти стовпців. Шість вхідних бітів визначають, який рядок і стовпець використовувати. Біти 1 і 6 визначають номер рядка, а біти 2, 3, 4 і 5 – номер стовпця. Вихід кожного S-Box – це значення, що міститься в цьому елементі таблиці [13];

- p-Box: тепер у нас є вісім партій чотирибітових виходів, які потім об'єднуються у 32-бітове значення і переставляються, щоб сформувати вихід функції F [12].

Загальну структуру алгоритма DES показано на Рис. 3.4.

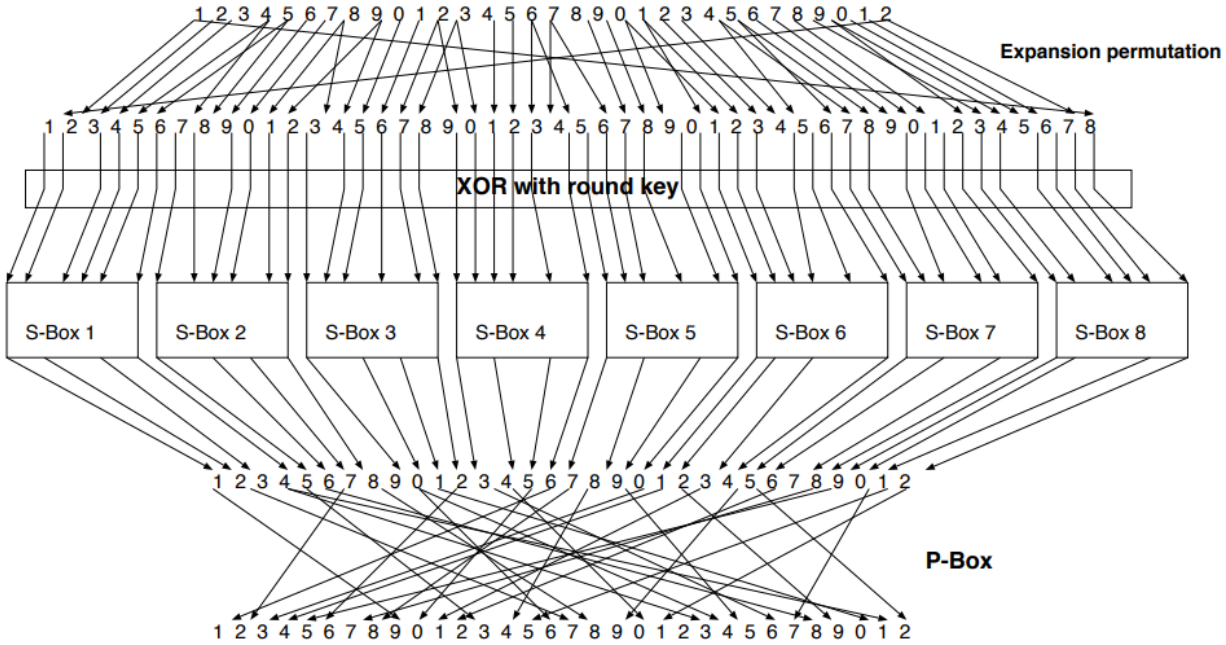

Рисунок 3.4 – Структура DES-функції F

Тепер детально розглянемо кожен з кроків, які ми ще не до кінця визначили.

Початкова перестановка, IP: Початкова перестановка DES визначена в таблиці, показаній на Рис. 3.5. Тут 58 у першій позиції означає, що перший біт на виході з IP є 58-м бітом на вході, і так далі.

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Рисунок 3.5 – Початкова перестановка DES

Зворотна перестановка подається аналогічним чином у таблиці, показаній на Рис. 3.6 [12].

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

Рисунок 3.6 – Зворотна перестановка

Перестановка розширення, E: перестановку розширення наведено у таблиці, показаній на Рис. 3.7. Кожен рядок відповідає бітам, які вводяться у відповідний S-Box на наступному етапі. Зверніть увагу, що біти, які вибирають рядок одного S-Box (перший і останній біт у кожному рядку), також також використовуються для вибору стовпчика іншого S-Box [13].

32	1	2	3	4	5
4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

Рисунок 3.7 – Перестановка розширення

S-Box: детальна інформація про вісім блоків DES S-Box наведена на Рис. 3.8. Нагадаємо, що кожен блок складається з таблиці з чотирма рядками та шістнадцятьма стовпчиками.

S-Box 1

14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
4	1	14	8	13	6	2	11	15	12	9	7	3	10	5	0
15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13

S-Box 2

15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9

S-Box 3

10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12

S-Box 4

7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14

S-Box 5

2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

S-Box 6

12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
9	14	15	5	2	8	12	3	7	0	4	10	1	13	11	6
4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13

S-Box 7

4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12

S-Box 8

13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

Рисунок 3.8 – Блоки DES S-Box

Перестановка P-Box, P: перестановка P-Box бере вісім партій чотирибітових відрізків, що виходять з S-боксів, і створює 32-бітну перестановку цих значень, як показано у таблиці, показаній на Рис. 3.9 [12].

16	7	20	21
29	12	28	17
1	15	23	26
5	18	31	10
2	8	24	14
32	27	3	9
19	13	30	6
22	11	4	25

Рисунок 3.9 – Перестановка P-Box

Для виявлення помилок DES використовує 56-бітний ключ, який фактично вводиться як бітовий рядок з 64 біт, що складається з ключа та восьми бітів парності. Ці біти парності знаходяться в бітових позиціях 8, 16,..., 64 і гарантують, що кожен байт ключа містить непарну кількість бітів.

Спочатку ми переставляємо біти ключа відповідно до наступної перестановки (яка приймає 64-бітові вхідні дані і видає 56-бітові вихідні дані, отже, відкидаючи біти парності) [14].

Результат цієї перестановки, яку в літературі називають PC-1, ділиться на 28-бітну ліву половину C_0 і 28-бітну праву половину D_0. Тепер для кожного раунду обчислимо

$C_i = C_i{-}1 \ll p_i$,

$D_i = D_i{-}1 \ll p_i$,

де x ≪ pi означає циклічний зсув x вліво на pi позицій. Якщо кругле число і дорівнює 1, 2, 9 або 16, то зсуваємось на одну позицію, інакше – на дві позиції.

Нарешті, дві частини Ci і Di знову з'єднуються разом і піддаються ще одній перестановці, яка називається PC-2, для отримання остаточного 48-бітового круглого ключа [13].

3.2 Розробка інтерфейсу користувача

Процес розробки інтерфейсу користувача для цього веб-застосунку спрямований на забезпечення зручної взаємодії користувача з функціями шифрування та дешифрування зображень. Оскільки основні дії користувача –

це завантаження файлів, введення ключа для шифрування та дешифрування, а також завантаження результату, інтерфейс має бути максимально простим і зрозумілим. Основні принципи дизайну інтерфейсу для цього проекту такі:

1. Головна сторінка (dashboard.html): це стартова сторінка, де користувач може обрати між двома основними функціями: шифруванням та дешифруванням.

Дизайн: головна сторінка містить два великі, чітко помітні кнопки – "Encrypt" і "Decrypt". Ці кнопки дозволяють користувачеві швидко вибрати потрібну дію. Центральний заголовок дає зрозуміти користувачеві основне призначення системи. Стиль простий, без зайвих елементів, що сприяє швидкій навігації;

2. Інтерфейс дешифрування (decrypt.html): на сторінці дешифрування користувач може завантажити зашифроване зображення або текст, ввести секретний ключ та отримати розшифроване зображення для завантаження.

Секція завантаження файлів передбачає можливість перетягування файлу у спеціальну зону або вибору файлу за допомогою кнопки.

Поле для введення ключа шифрування знаходиться поруч із кнопкою "Decrypt", що дозволяє зручно вводити секретний ключ і одразу запускати процес розшифрування.

Візуальні підказки: повідомлення про обмеження розміру файлу (10 MB) і помилки при введенні неправильного ключа допомагають користувачеві розуміти можливі обмеження та помилки.

Аналогічно оформлена сторінка шифрування (encrypt.html);

3. Функціональність шифрування та дешифрування. Динамічність інтерфейсу: за допомогою JavaScript реалізовано завантаження та обробку файлів прямо на сторінці. Це дозволяє користувачеві бачити результат одразу після введення ключа і натискання кнопки "Decrypt".

Зворотний зв'язок: В разі успішного дешифрування або помилок система відразу інформує користувача. Наприклад, повідомлення про помилковий ключ з’являється у вигляді спливаючого вікна, що робить інтерфейс більш

інтерактивним;

4. Мінімалістичний дизайн. Оскільки основна увага приділяється функціональності, інтерфейс побудований у мінімалістичному стилі. Використовуються базові кольорові рішення, щоб уникнути перевантаженості елементами. Користувач може швидко орієнтуватися, оскільки всі елементи мають чітко визначені функції;

5. Використання іконок та зображень

На сторінці дешифрування використовуються стандартні іконки для зон завантаження файлів (наприклад, значок хмари для перетягування файлу), що полегшує розуміння дій для користувача [21].

HTML-код dashboard.html (див. Рис. 3.10) створює головну сторінку для веб-застосунку, що надає користувачу два варіанти: шифрування або дешифрування зображень за допомогою алгоритму Triple DES. Основні елементи:

1. Заголовок сторінки: "Ласкаво просимо";

2. Стилі: підключення зовнішнього CSS для оформлення;

3. Контейнер з кнопками: заголовок: "Шифрування/Дешифрування Triple DES", кнопки: "Шифрувати" веде на сторінку `encrypt.html`. "Дешифрувати" веде на сторінку `decrypt.html`.

4. Структура: візуальний поділ на частини — верхню, нижню та центральну;

Ця сторінка слугує інтерфейсом для навігації між двома діями: шифруванням та дешифруванням файлів.

```html
<!DOCTYPE html>
<html>
<head>
  <title>Ласкаво просимо</title> <!-- Заголовок сторінки -->
  <link rel="stylesheet" href="./css/style_dashboard.css"> <!-- Підключення стилів -->
</head>
<body>
<!-- Контейнер для розміщення елементів інтерфейсу -->
<div class="container" onclick="onclick">
    <!-- Верхня частина контейнера -->
    <div class="top"></div>

    <!-- Нижня частина контейнера -->
    <div class="bottom"></div>

    <!-- Центральна частина з основним вмістом -->
    <div class="center">
      <!-- Заголовок з описом функціональності -->
      <h2 style="font-size: 30px; text-align: center;">Шифрування/Дешифрування Triple DES</h2>

      <!-- Кнопка для переходу на сторінку шифрування -->
      <button onclick="location.href='./encrypt.html'">Шифрувати</button>

      <!-- Кнопка для переходу на сторінку дешифрування -->
      <button onclick="location.href='./decrypt.html'">Дешифрувати</button>

      <h2> </h2>
    </div>
</div>
</body>
</html>

```

Рисунок 3.10 – Код розмітки для початкової сторінки

HTML-документ encrypt.html (див. Рис. 3.11) дозволяє користувачеві завантажити зображення, зашифрувати його за допомогою алгоритму Triple DES і зберегти результат. Основні елементи:

1. Заголовок: виводить текст "Зашифруйте ваше зображення";

2. Зона завантаження файлу: користувач може перетягнути або вибрати зображення для завантаження;

3. Відображення зображення: після вибору файл показується на сторінці;

4. Поле для секретного ключа: користувач вводить ключ для шифрування зображення;

5. Кнопка "Зашифрувати": викликає процес шифрування зображення.

Основний функціонал:

- зображення завантажується та конвертується у формат Base64;
- за допомогою секретного ключа та алгоритму Triple DES зображення шифрується;

- результат зберігається як текстовий файл через бібліотеку `FileSaver.js`.

```html
<!DOCTYPE html>
<html lang="uk"> <!-- Встановлення української мови -->
<head>
  <!-- Підключення бібліотек для криптографії та збереження файлів -->
  <script src="./js/crypto-js.js"></script>
  <script src="./js/FileSaver.js"></script>
  <meta charset="UTF-8">
  <title>Шифрування</title> <!-- Заголовок сторінки -->

  <link rel="stylesheet" href="./css/style.css">

</head>
<body>
  <!-- Заголовок сторінки -->
  <h1 id="title">Зашифруйте ваше зображення</h1> <!-- Заголовок сторінки -->

  <!-- Зона для завантаження файлу -->
  <div class="zone">
    <div id="dropZ">
      <i class="fa fa-cloud-upload"></i>
      <div>Перетягніть файл сюди</div> <!-- Опис дії -->
      <span>АБО</span>
      <div class="selectFile">
        <label for="file">Виберіть файл</label> <!-- Вибір файлу -->
        <input type="file" accept="image/*" name="files[]" id="file" onchange="loadFile(event)">
      </div>
      <p>Обмеження на розмір файлу: 10 МБ</p> <!-- Опис обмеження -->
    </div>
  </div>

  <!-- Зона для відображення зображення -->
  <div class="zone-image">
    <img id="output" width="400" style="margin-top: 30px; position: relative;"/>
    <label>
      <input type="text" id="secret" placeholder="Введіть свій секретний ключ"> <!-- Поле введення секретного ключа -->
      <button onclick="encryptToDES()">Зашифрувати</button> <!-- Кнопка шифрування -->
    </label>
  </div>
```

Рисунок 3.11 – Код розмітки для сторінки шифрування

HTML-документ decrypt.html (див. Рис. 3.12) дозволяє користувачеві розшифрувати зашифроване текстове зображення, яке було зашифроване за допомогою алгоритму Triple DES. Цей код забезпечує простий інтерфейс для розшифрування зашифрованих зображень. Основні елементи:

1. Заголовок: виводить текст "Розшифруйте ваше зображення";

2. Зона завантаження файлу: користувач може перетягнути або вибрати зашифрований файл для розшифрування;

3. Поле для секретного ключа: користувач вводить секретний ключ для розшифрування;

4. Кнопка "Розшифрувати": викликає процес розшифрування зображення.

Основний функціонал:

- файл, який завантажується, перетворюється на текст;

- користувач вводить секретний ключ для розшифрування зашифрованого тексту;
- алгоритм Triple DES використовується для розшифрування, після чого результат стає доступним для завантаження.

```
<!DOCTYPE html>
<html lang="uk"> <!-- Мова сторінки -->
<head>
  <!-- Підключення бібліотеки для криптографії -->
  <script src="./js/crypto-js.js"></script>
  <meta charset="UTF-8">
  <title>Дешифрування</title> <!--Заголовок сторінки -->
  <link rel="stylesheet" href="./css/style.css">
</head>
<body>
  <!-- Заголовок сторінки -->
  <h1 id="title">Дешифруйте ваше зображення</h1>

  <!-- Зона завантаження файлу -->
  <div class="zone">
    <div id="dropZ">
      <i class="fa fa-cloud-upload"></i>
      <div>Перетягніть файл сюди</div>
      <span>АБО</span>
      <div class="selectFile">
        <label for="file">Виберіть файл</label>
        <input type="file" name="files[]" id="file" onchange="previewFile()"> <!-- Поле для вибору файлу -->
      </div>
      <p>Обмеження на розмір файлу: 10 МБ</p>
    </div>
  </div>

  <!-- Зона для відображення результатів дешифрування -->
  <div class="zone-image">
    <textarea id="content" class="result" rows="20" cols="50"></textarea>
    <label>
      <input type="text" id="secret" placeholder="Введіть свій секретний ключ"> <!-- Поле для введення ключа -->
      <button onclick="decryptFromDES()">Дешифрувати</button>
    </label>
  </div>
```

Рисунок 3.12 – Код розмітки для сторінки шифрування

CSS-код style_dashboard.css (див. Рис. 3.13) налаштовує стиль для анімаційного інтерфейсу, який складається з контейнера та двох елементів (верхнього та нижнього), що змінюють свої позиції та кольори під час наведення курсора. Основні елементи:

- шрифт: використовується шрифт Roboto для тексту;
- контейнер: заповнює весь екран і містить елементи, які анімуються при наведенні;
- анімації: верхній і нижній елементи (з класами `.top` і `.bottom`) обертаються і змінюють кольори, коли курсор наведено;
- центральний елемент: відображається з затримкою, його стиль включає вирівнювання та прозорість.

```css
@import url("https://fonts.googleapis.com/css?family=Roboto:400,700"); /* Завa
*, *:before, *:after {
  box-sizing: border-box; /* Включення моделі коробки для всіх елементів */
}

body {
  min-height: 100vh; /* Мінімальна висота тіла */
  font-family: "Roboto", sans-serif; /* Використання шрифту Roboto */
  overflow: hidden; /* Сховати переповнені елементи */
}

.container {
  position: absolute;
  width: 100%;
  height: 100%;
  overflow: hidden;
}

.container:hover .top:before, .container:hover .top:after,
.container:hover .bottom:before, .container:hover .bottom:after,
.container:active .top:before, .container:active .top:after,
.container:active .bottom:before, .container:active .bottom:after {
  margin-left: 200px; /* Зсув елементів при наведенні */
  transform-origin: -200px 50%;
  transition-delay: 0s;
}

.container:hover .center, .container:active .center {
  opacity: 1; /* Зробити центр видимим */
  transition-delay: 0.2s;
}

.top:before, .top:after, .bottom:before, .bottom:after {
  content: "";
  display: block;
  position: absolute;
  width: 200vmax;
  height: 200vmax;
  top: 50%;
  left: 50%;
  margin-top: -100vmax;
  transform-origin: 0 50%;
  transition: all 0.5s cubic-bezier(0.445, 0.05, 0, 1);
  z-index: 10;
  opacity: 0.65;
  transition-delay: 0.2s;
}
```

Рисунок 3.13 – Код каскадних таблиць стилю для головної сторінки

CSS-код style.css (див. Рис. 3.14) визначає стиль для веб-контейнера з анімаційними ефектами, який містить елементи, що реагують на наведену курсор. Основні компоненти:

- шрифт: використовується шрифт Raleway для всього тексту;
- контейнер: заповнює весь екран і містить елементи, які змінюють свої позиції та прозорість при взаємодії з курсором;
- анімації: верхні (`.top`) і нижні («.bottom») елементи анімуються, обертаючись та змінюючи кольори при наведенні. Вони мають прозорість та затримку переходу;
- центральний елемент: відображається з плавною анімацією при

наведені, з елементами, вирівняними по центру.

```css
@font-face {
  font-family: 'Roboto'; /* Завантаження шрифту Roboto */
  font-style: normal;
  font-weight: 300;
  src: url('https://fonts.googleapis.com/css2?family=Roboto:wght@300&display=swap') format('truetype');
}
@font-face {
  font-family: 'Roboto';
  font-style: normal;
  font-weight: 400;
  src: url('https://fonts.googleapis.com/css2?family=Roboto:wght@400&display=swap') format('truetype');
}
@font-face {
  font-family: 'Roboto';
  font-style: normal;
  font-weight: 500;
  src: url('https://fonts.googleapis.com/css2?family=Roboto:wght@500&display=swap') format('truetype');
}
@font-face {
  font-family: 'Roboto';
  font-style: normal;
  font-weight: 700;
  src: url('https://fonts.googleapis.com/css2?family=Roboto:wght@700&display=swap') format('truetype');
}
body {
  background: #f0f4f8; /* Колір фону */
  font-family: "Roboto"; /* Використання шрифту Roboto */
}
.zone {
  margin: auto;
  position: absolute;
  top: 10%;
  left: 0;
  bottom: 0;
  right: 40%;
  background: radial-gradient(ellipse at center, #eb6a5a 0, #c9402f 100%);
  width: 40%;
  height: 50%;
  border: 5px dashed white;
  text-align: center;
  color: white;
  z-index: 20;
  transition: all 0.3s ease-out;
  box-shadow: 0 0 0 1px rgba(255, 255, 255, 0.05), inset 0 0 0.25em 0 rgba(0, 0, 0, 0.25);
}
```

Рисунок 3.14 – Код каскадних таблиць стилю для сторінок шифрування та дешифрування

Цей інтерфейс користувача забезпечує просту і зрозумілу взаємодію з веб-застосунком для шифрування та дешифрування зображень. Мінімалістичний дизайн робить його зручним для користувача. Вихідні коди застосунку подано в Додатку А.

3.3 Розробка функціональності програмного забезпечення

JavaScript-скрипти вписані безпосередньо всередині тегу <script> у HTML кодах документів, що дозволяє інтегрувати функціональність JavaScript з HTML-структурою сторінки. Детальніший опис:

1. Включення JS бібліотек

На початку кожної HTML сторінки є секція <head>, де підключаються

бібліотеки JavaScript, такі як:

- crypto-js.js: бібліотека для криптографії, що використовується для шифрування та дешифрування даних;
- fileSaver.js: бібліотека, яка дозволяє зберігати файли на локальному комп'ютері (див. Рис. 3.15).

```
<!DOCTYPE html>
<html lang="uk"> <!-- Встановлення української мови -->
<head>
  <!-- Підключення бібліотек для криптографії та збереження файлів -->
  <script src="./js/crypto-js.js"></script>
  <script src="./js/FileSaver.js"></script>
  <meta charset="UTF-8">
  <title>Шифрування</title> <!-- Заголовок сторінки -->
```

Рисунок 3.15 – Підключення необхідних бібліотек

Ці бібліотеки підключаються через теги <script> перед закриваючим тегом </head>.

2. JS код у секції <body>

У секції <body> містяться функції JavaScript, які виконують основну логіку:

- loadFile(event): функція, яка обробляє завантаження файлів. Вона отримує об'єкт файлу, створює URL для його відображення та оновлює атрибут src» елемента <img> для показу завантаженого зображення;
- обробка зображення: всередині обробника події onload для зображення, створюється новий <canvas>, на якому зображення малюється, і отримані дані конвертуються у формат Base64 [22]. Це дозволяє подальше шифрування зображення (див. Рис. 3.16);

```
<!-- Зона для завантаження файлу -->
<div class="zone">
  <div id="dropZ">
    <i class="fa fa-cloud-upload"></i>
    <div>Перетягніть файл сюди</div> <!-- Опис дії -->
    <span>АБО</span>
    <div class="selectFile">
      <label for="file">Виберіть файл</label> <!-- Вибір файлу -->
      <input type="file" accept="image/*" name="files[]" id="file" onchange="loadFile(event)">
    </div>
    <p>Обмеження на розмір файлу: 10 МБ</p> <!-- Опис обмеження -->
  </div>
</div>
```

Рисунок 3.16 – Головні функції застосунку за допомогою мови JavaScript

- encryptToDES()» (див. Рис. 3.17): ця функція відповідає за шифрування зображення. Вона отримує секретний ключ з текстового поля і перевіряє, чи ключ введено. Якщо так, шифрується зображення за допомогою Triple DES, а результат зберігається у файл для завантаження.

```
<!-- Зона для відображення зображення -->
<div class="zone-image">
  <img id="output" width="400" style="margin-top: 30px; position: relative;"/>
  <label>
    <input type="text" id="secret" placeholder="Введіть свій секретний ключ">
    <button onclick="encryptToDES()">Зашифрувати</button> <!-- Кнопка шифруван
  </label>
</div>
```

Рисунок 3.17 – Функція застосунку encryptToDES

3. Взаємодія з HTML елементами. JS код безпосередньо взаємодіє з HTML елементами через методи DOM (Document Object Model). Наприклад:

- отримання значення з текстового поля за допомогою document.getElementById('secret').value»;
- оновлення тексту у <textarea> або зображення у <img> (див. Рис. 3.18).

```
// Функція для завантаження зображення
var loadFile = function(event) {
    var image = document.getElementById('output');
    image.src = URL.createObjectURL(event.target.files[0]);
};

var img = document.getElementById('output');
img.crossOrigin = 'Anonymous';

// Після завантаження зображення перетворюємо його в base64
img.onload = function(){
    var canvas = document.createElement('canvas');
    var ctx = canvas.getContext('2d');
    canvas.height = this.naturalHeight;
    canvas.width = this.naturalWidth;
    ctx.drawImage(this, 0, 0);
    data = canvas.toDataURL('image/jpeg'); // Отримуємо дані зображення у форматі base64
};

// Функція для шифрування зображення за допомогою Triple DES
function encryptToDES() {
  var secretKey = document.getElementById('secret').value; // Отримуємо секретний ключ
  if(secretKey != ''){
    var fileName = "encrypted_image"; // Ім'я зашифрованого файлу
    var plainText = data; // Дані зображення
    var encryptedText = CryptoJS.TripleDES.encrypt(plainText, secretKey); // Шифруємо зображення
    saveAs(new File([encryptedText], fileName, {type: "text/plain;charset=utf-8"})); // Зберігаємо зашифрований файл
  }
  else {
    alert("Будь ласка, введіть свій секретний ключ!"); // Попередження про відсутність ключа
  }
}
```

Рисунок 3.18 – Взаємодія з HTML елементами

4. Виклики функцій. Кнопки на сторінці (<button>) використовують атрибут onclick», щоб викликати відповідні JavaScript функції. Це забезпечує інтерактивність на веб-сторінці, дозволяючи користувачеві виконувати дії (завантажувати файли, шифрувати зображення) (див. Рис. 3.19).

```
<!-- Зона для відображення результатів дешифрування -->
<div class="zone-image">
  <textarea id="content" class="result" rows="20" cols="50"></textarea>
  <label>
    <input type="text" id="secret" placeholder="Введіть свій секретний ключ">
    <button onclick="decryptFromDES()">Дешифрувати</button>
  </label>
</div>
```

Рисунок 3.19 – Виклики функцій за допомогою кліку

Таким чином, JavaScript код вбудований в HTML, що дозволяє розробникам створювати динамічні та інтерактивні веб-сторінки. Це підвищує функціональність, дозволяючи користувачеві взаємодіяти зі сторінкою в реальному часі. Вихідні коди застосунку подано в Додатку А.

3.4 Робота програмного забезпечення

Робота програмного забезпечення для шифрування та дешифрування зображень реалізується на основі алгоритму Triple DES, який забезпечує високий рівень безпеки шляхом трьохкратного шифрування даних. Процес функціонування програмного забезпечення можна розділити на кілька основних етапів.

Відкривши файл «dashboard.html», користувач бачить на екрані початковий вигляд програмного забезпечення, як показано на Рис. 3.3.1. Після наведення курсору на вікно браузера із програмним забезпеченням, користувач бачить перед собою вибір функцій: шифрування та дешифрування, як показано на Рис. 3.20-3.22.

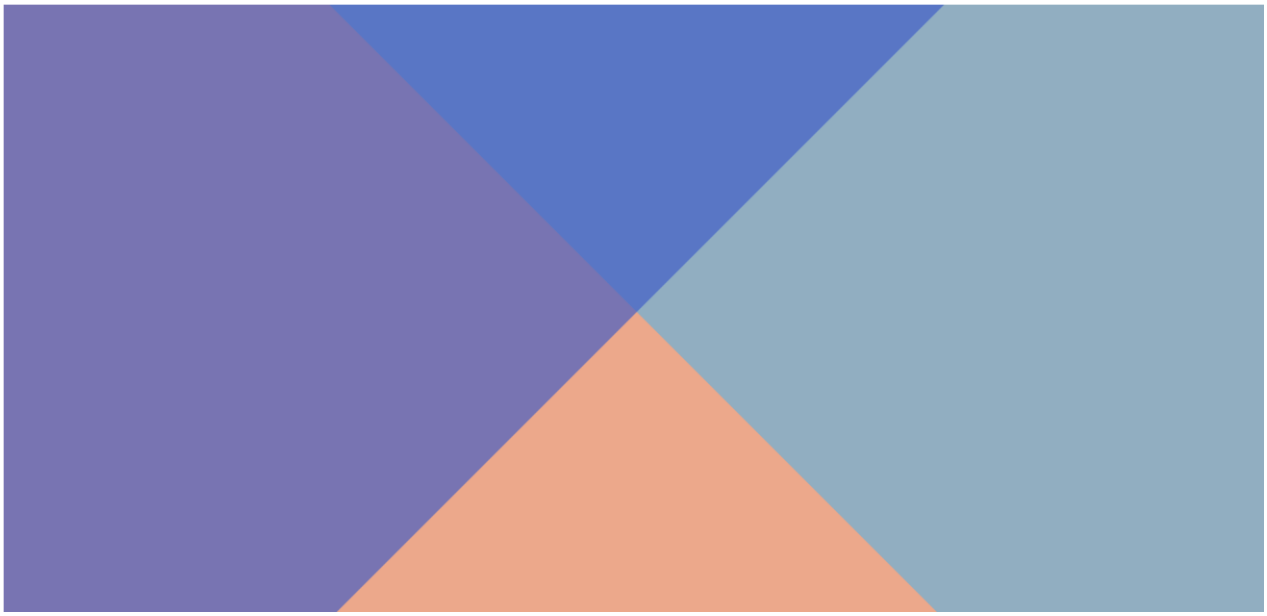

Рисунок 3.20 – Початковий вигляд програмного забезпечення

Рисунок 3.21 – Початковий вигляд програмного забезпечення після наведення курсора на вікно браузера

Рисунок 3.22 – Вибір функцій: шифрування та дешифрування на початковому вікні

По-перше, користувачеві надається можливість завантажити зображення з локального пристрою через інтуїтивно зрозумілий інтерфейс. Це можна зробити як за допомогою перетягування файлу в спеціальну зону на веб-сторінці, так і через стандартне діалогове вікно вибору файлу. Після вибору або завантаження зображення, воно автоматично відображається на екрані, що дає користувачу можливість попередньо переглянути завантажене зображення. Ці можливості програмного забезпечення продемонстровано на Рис. 3.23-3.25.

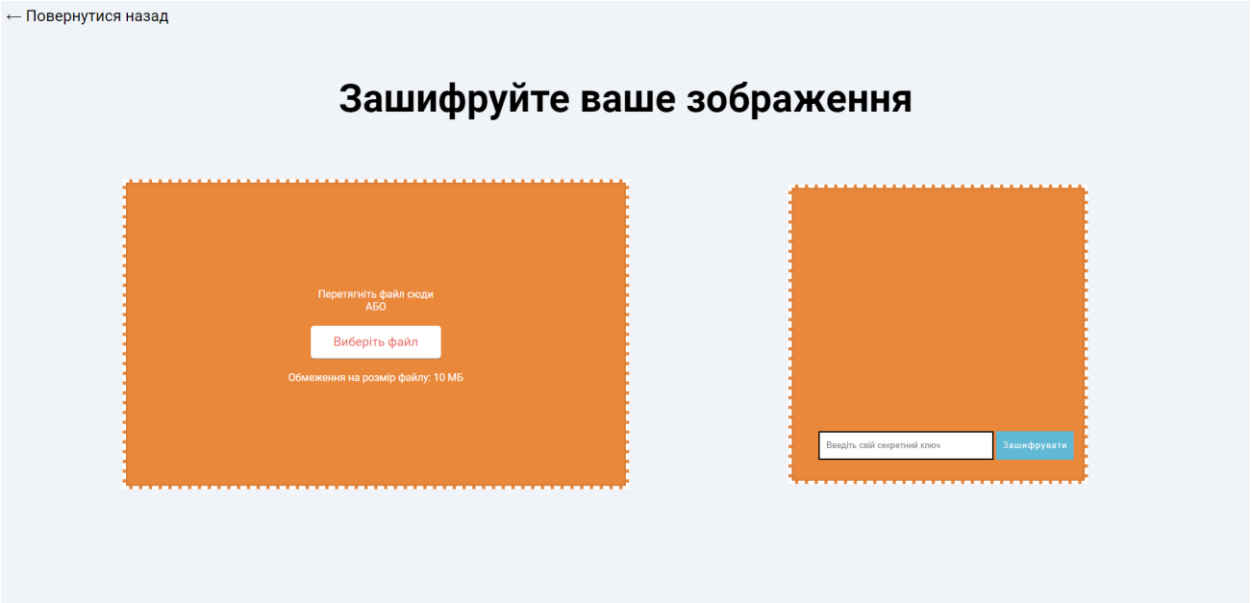

Рисунок 3.23 – Початковий вигляд сторінки для шифрування зображень

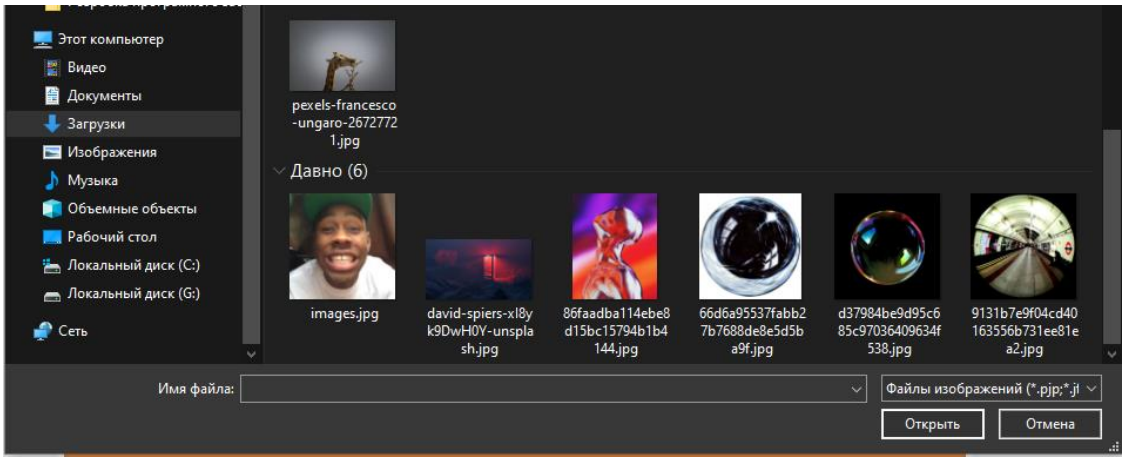

Рисунок 3.24 – Вікно вибору зображень

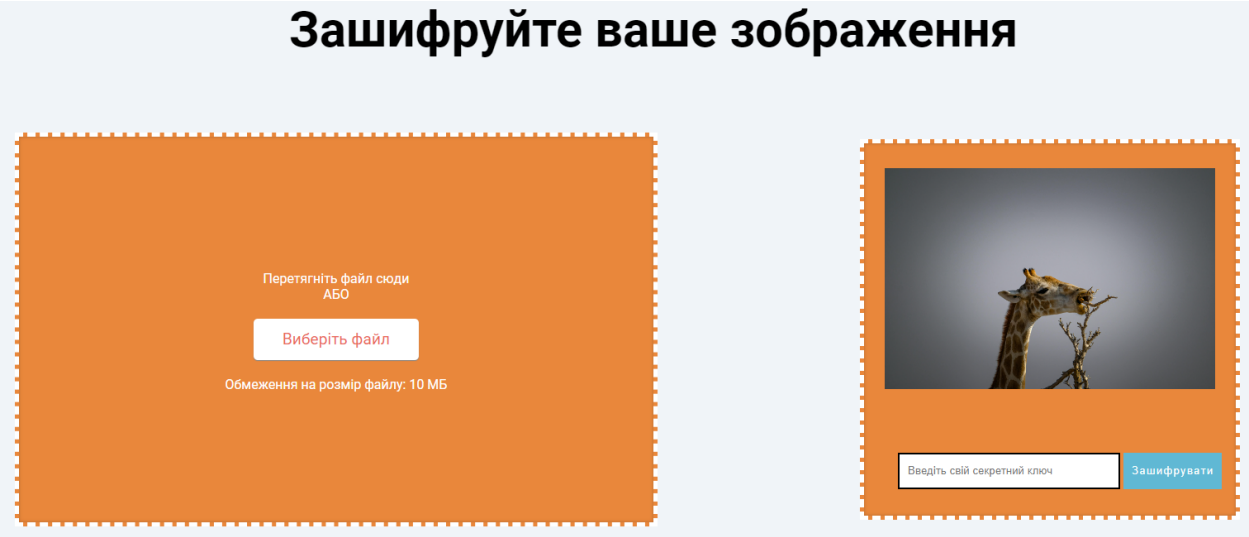

Рисунок 3.25 – Вигляд сторінки після вибору зображення

Далі користувачеві пропонується ввести секретний ключ (див. Рис. 3.26), який буде використано для шифрування або дешифрування зображення. Введений ключ є критичним компонентом процесу, оскільки саме він забезпечує надійний захист інформації.

Рисунок 3.26 – Введення ключа

Під час шифрування завантажене зображення спочатку перетворюється у формат base64, який дозволяє представити дані зображення у вигляді тексту. Це необхідно для подальшого оброблення його криптографічними алгоритмами. Після цього програмне забезпечення застосовує алгоритм Triple DES для шифрування зображення з використанням наданого секретного ключа. У результаті користувач отримує зашифрований текст, який можна зберегти у вигляді окремого файлу (див. Рис. 3.27-3.28).

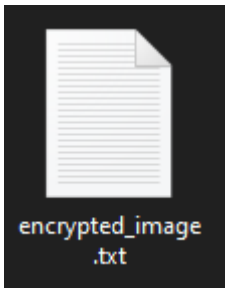

Рисунок 3.27 – Створений файл після шифрування зображення

Рисунок 3.28 – Вміст створеного файлу

Процес дешифрування працює аналогічно, але у зворотному порядку. Користувач вводить секретний ключ, який використовується для дешифрування раніше зашифрованого тексту. Програмне забезпечення розшифровує текст за допомогою Triple DES і повертає його у форматі зображення, яке можна завантажити (див. Рис. 3.29-3.33).

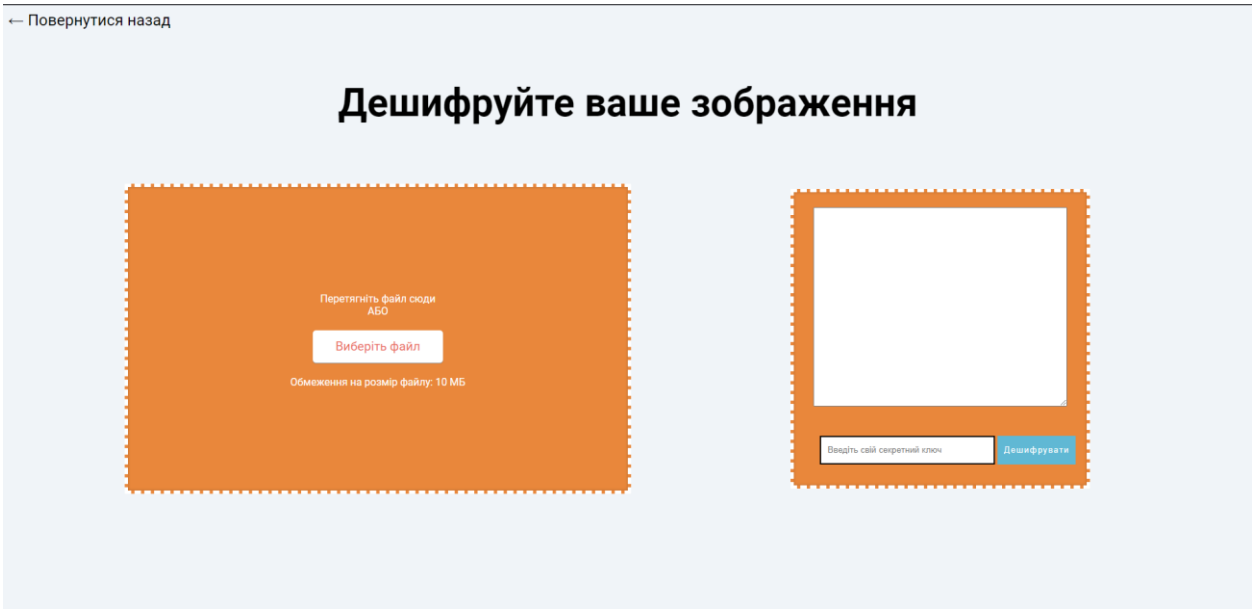

Рисунок 3.29 – Початковий вигляд сторінки для шифрування зображень

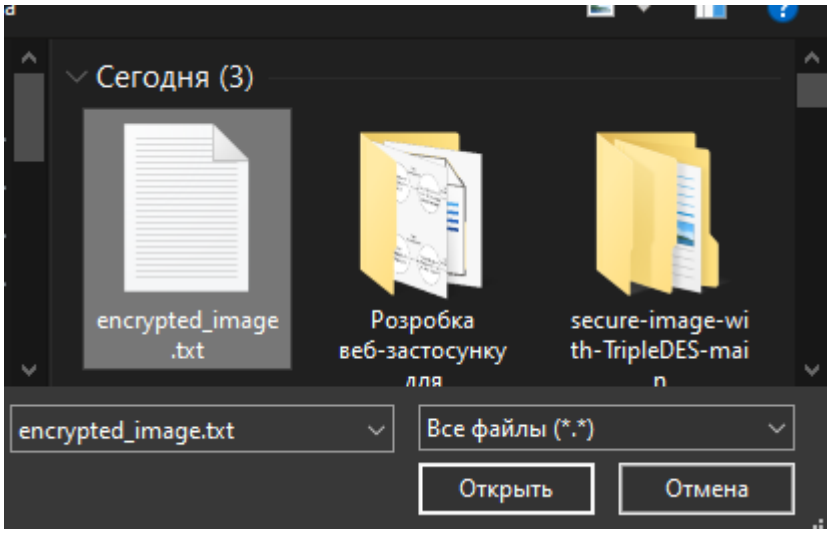

Рисунок 3.30 – Вікно вибору файла

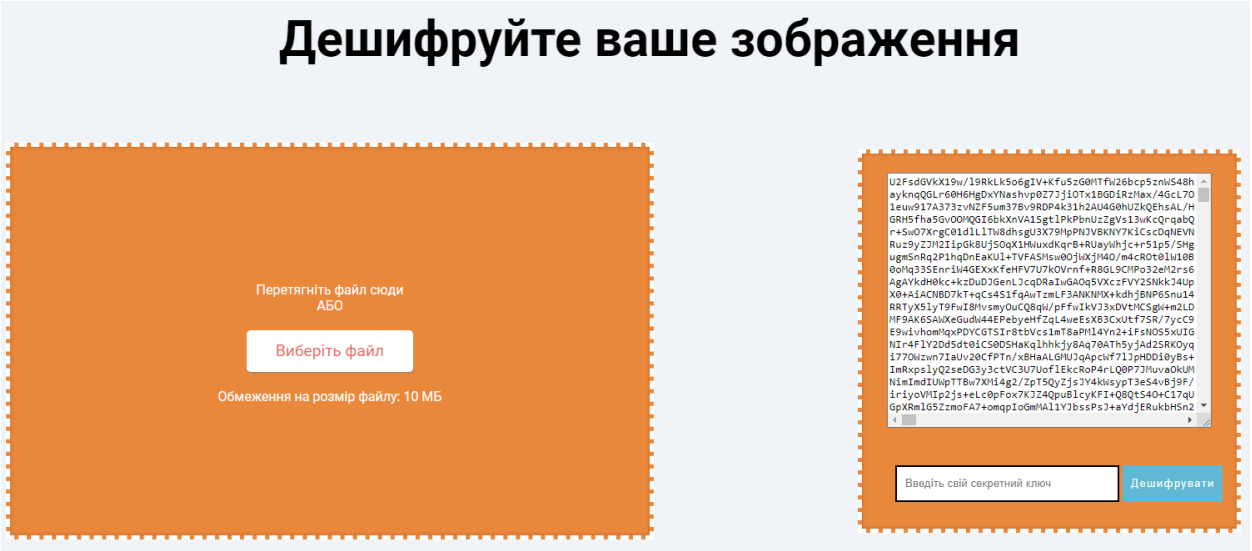

Рисунок 3.31 – Вигляд сторінки після зчитування файлу

Рисунок 3.32 – Введення ключа

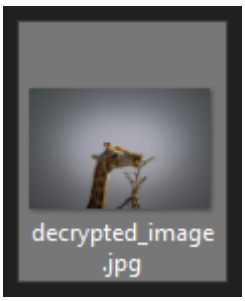

Рисунок 3.33 – Файл розшифрованого зображення

На Рис. 3.34-3.35 показано оригінальне та зашифроване-дешифроване зображення, на Рис. 3.36 порівняно характеристики двох файлів-зображень.

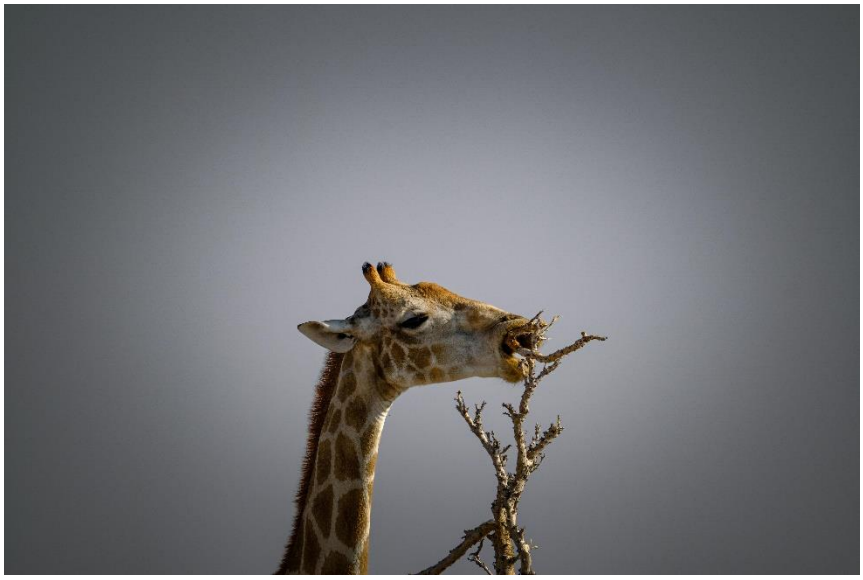

Рисунок 3.34 – Оригінальне зображення

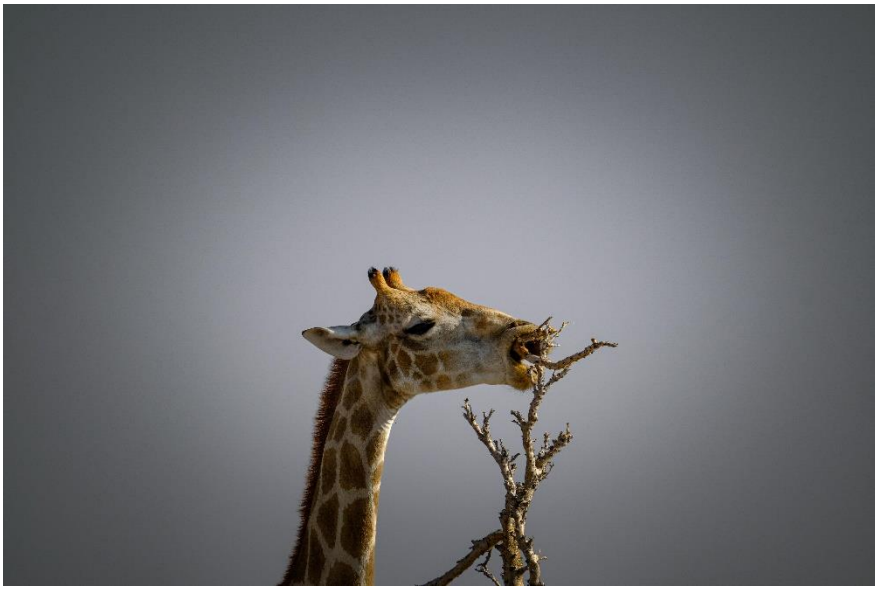

Рисунок 3.35 – Зображення після розшифрування

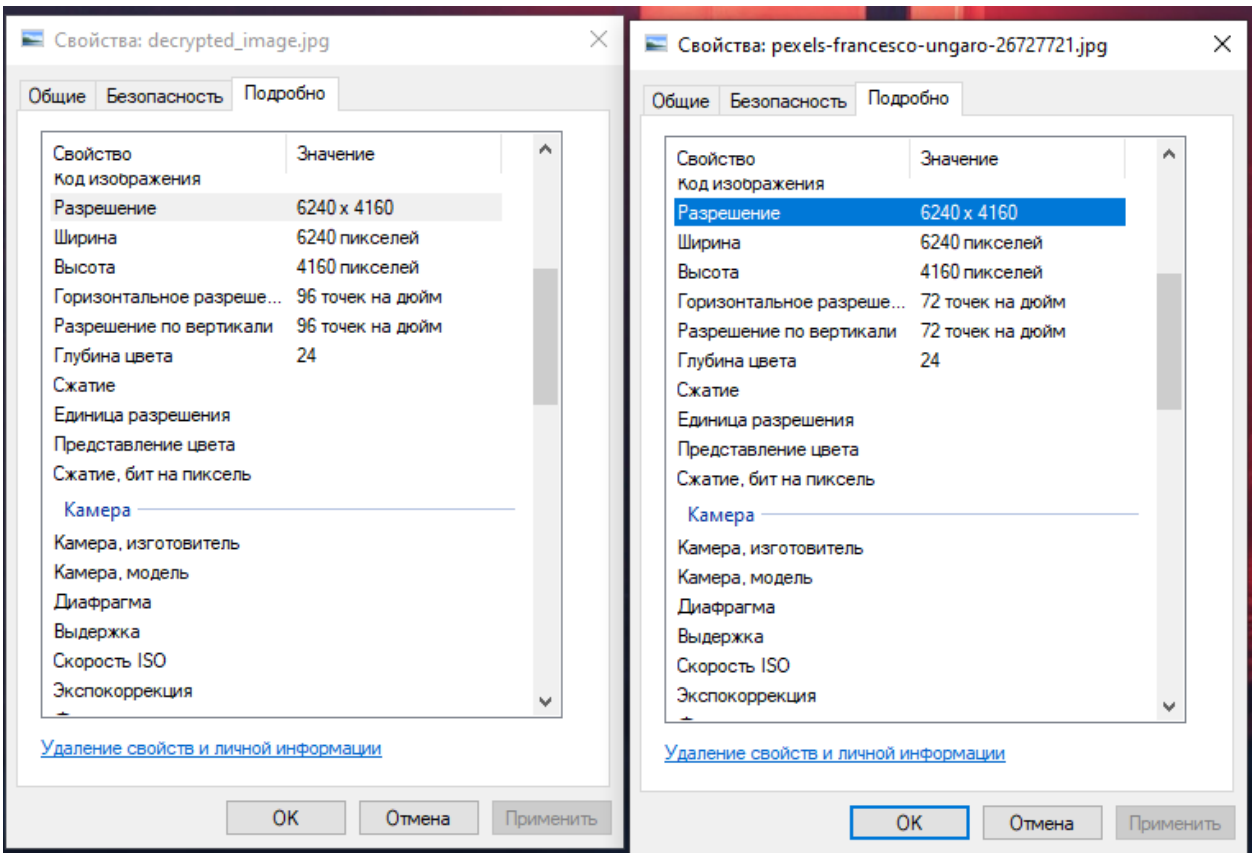

Рисунок 3.36 – Порівняння характеристик зображень (справа – оригінал, зліва – розшифроване зображення)

Крім того, при введенні неправильного ключа програмне забезпечення повідомляє користувача про помилку, як показано на Рис. 3.37.

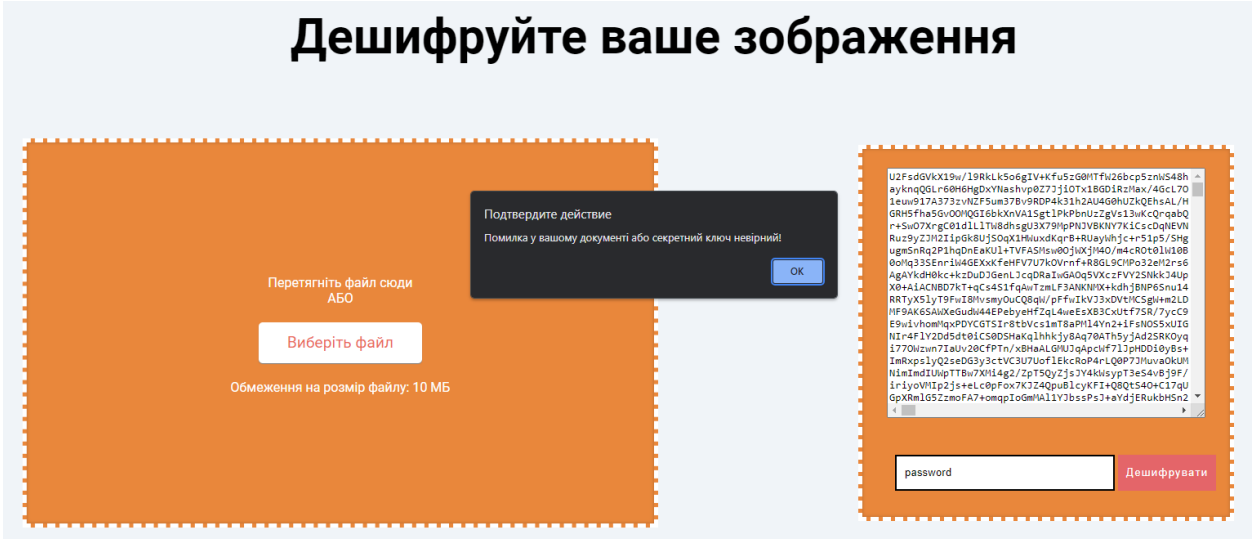

Рисунок 3.37 – Повідомлення про помилку

При неправильному вхідному файлі для дешифрування також програмне

забезпечення повідомляє користувача про помилку, як показано на Рис. 3.38.

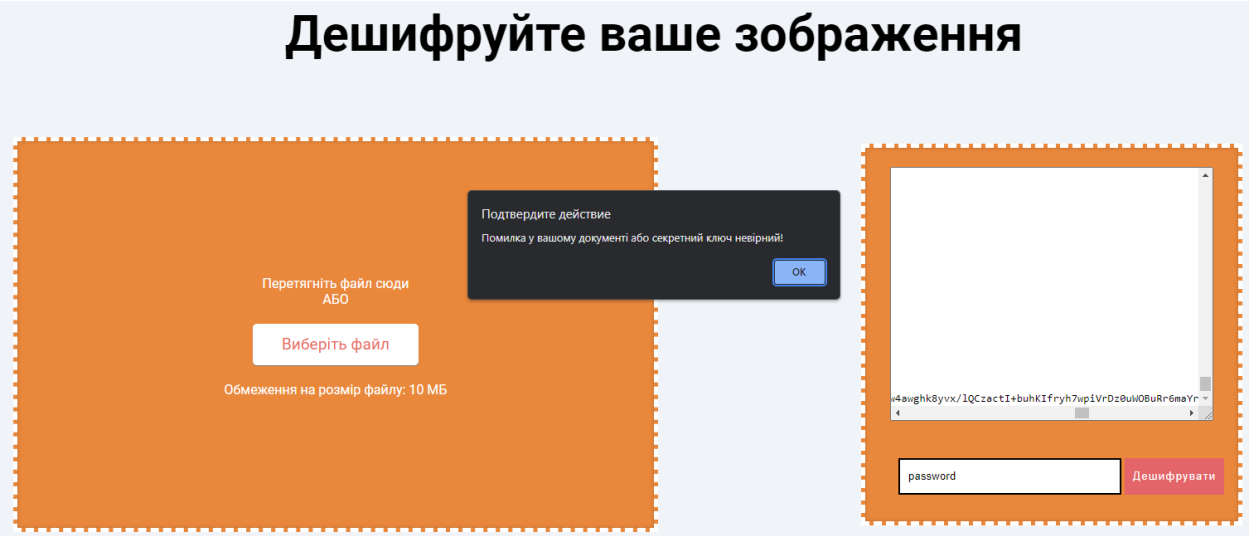

Рисунок 3.38 – Повідомлення про помилку

Таким чином, робота програмного забезпечення забезпечує простий та зрозумілий процес шифрування та дешифрування зображень, де на перший план виходить захист даних за допомогою перевірених криптографічних методів.

РОЗДІЛ 4
ТЕСТУВАННЯ ТА ОЦІНКА ЕФЕКТИВНОСТІ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Чек-лист тестування програмного забезпечення

Контрольний список тестування (чек-лист) – це список завдань або пунктів, які тестувальник повинен виконати, щоб забезпечити ретельне тестування програмного додатку.

Контрольні списки можна використовувати для тестування специфічних особливостей програмного забезпечення або для виконання загальних завдань тестування, таких як регресійне тестування. На відміну від тестових кейсів, контрольні списки не такі детальні і не містять покрокових інструкцій для виконання тесту.

Контрольні списки корисні, коли потрібно виконати один і той же набір тестів кілька разів, або коли вам потрібно виконати набір тестів швидко.

Контрольні списки також корисні, коли потрібно переконатися, що набір базових вимог виконано, перш ніж переходити до більш складного тестування. Наприклад, якщо йде процес тестування веб-сайт електронної комерції, можна використовувати контрольний список, щоб переконатися, що кошик для покупок працює правильно, процес оформлення замовлення проходить безперебійно, а платіжний шлюз безпечний [22].

Чек-лист представлений у вигляді Таблиці 4.1

Таблиця 4.1 – Чек-лист для ПЗ

№	Сценарій тестування	Результат очікуваний
1	Завантаження файлу	
2	Перетягування файлу в зону завантаження	Зображення завантажене
3	Вибір файлу через діалогове вікно	Зображення завантажене
4	Обмеження на розмір файлу (до 10 МБ)	Попередження про розмір, якщо більше 10 МБ
5	Підтримка зображень у форматі JPEG, PNG	Файл завантажений
6	Попередній перегляд зображення	Зображення відображено на екрані
	Введення секретного ключа	
7	Обов'язкове введення ключа для шифрування та дешифрування	Попередження при відсутності ключа
8	Повідомлення про помилку при відсутності ключа	Попередження відображено
9	Повідомлення про неправильний ключ під час дешифрування	Попередження відображено
	Процес шифрування	
10	Шифрування зображення при правильному ключі	Файл зашифрований
11	Завантаження зашифрованого файлу	Файл завантажений у форматі тексту
12	Використання Triple DES для шифрування	Зображення зашифроване
	Процес дешифрування	
13	Дешифрування при введенні правильного ключа	Зображення дешифроване
14	Повідомлення про помилку при неправильному ключі	Попередження відображено
15	Завантаження дешифрованого зображення	Файл завантажений
	Повідомлення про помилки	
16	Повідомлення при перевищенні розміру файлу	Попередження відображено
17	Повідомлення при неправильному форматі файлу	Попередження відображено
18	Попередження про відсутність ключа	Попередження відображено
	Візуальні елементи	
19	Відображення зображення до та після шифрування/дешифрування	Зображення відображене
20	Перевірка роботи кнопки повернення на попередню сторінку	Кнопка працює
	Кросбраузерність	
21	Робота у різних браузерах (Chrome, Firefox, Safari, Edge)	ПЗ працює коректно
	Кросплатформність	
23	Робота ПЗ на різних операційних системах (Windows, macOS, Linux)	ПЗ працює коректно
	Продуктивність	
24	Швидкість шифрування та дешифрування великих файлів	Процес проходить швидко без затримок

4.2 Тест-кейси для тестування програмного забезпечення

Тест-кейс – це дії, необхідні для перевірки певної функції або функціональності при тестуванні програмного забезпечення. У тестовому кейсі детально описані кроки, дані, передумови, необхідні для перевірки функції.

Він містить конкретні змінні, які потрібні QA для порівняння очікуваних і фактичних результатів, щоб зробити висновок про те, чи працює функція. Компоненти тестового кейсу включають вхідні дані, виконання та очікуваний результат/відповідь. Він вказує інженерам, що робити, як робити і які результати є прийнятними [23].

Тест-кейси представлені у вигляді Таблиць 4.2-4.11.

Таблиця 4.2 – Тест-кейс для тестування ПЗ

Поле	**Опис**
Назва	Перетягування файлу
Передумови	Відкритий веб-застосунок
Кроки	1. Перетягнути файл у зону завантаження
Очікуваний результат	Файл успішно завантажений
Фактичний результат	Файл успішно перетягнуто та завантажено

Таблиця 4.3 – Тест-кейс для тестування ПЗ

Поле	**Опис**
Назва	Обмеження розміру файлу
Передумови	Відкритий веб-застосунок
Кроки	1. Завантажити файл більше 10 МБ
Очікуваний результат	Виведення повідомлення про обмеження розміру файлу
Фактичний результат	Повідомлення про перевищення розміру файлу успішно виведено

Таблиця 4.4 – Тест-кейс для тестування ПЗ

Поле	**Опис**
Назва	Шифрування з правильним ключем
Передумови	Завантажене зображення
Кроки	1. Ввести коректний ключ
	2. Натиснути "Зашифрувати"
Очікуваний результат	Зображення зашифроване
Фактичний результат	Зображення успішно зашифроване та збережено

Таблиця 4.5 – Тест-кейс для тестування ПЗ

Поле	**Опис**
Назва	Шифрування без введення ключа
Передумови	Завантажене зображення
Кроки	1. Не вводити ключ
	2. Натиснути "Зашифрувати"
Очікуваний результат	Виведення повідомлення про відсутність ключа
Фактичний результат	Повідомлення про відсутність ключа успішно виведено

Таблиця 4.6 – Тест-кейс для тестування ПЗ

Поле	**Опис**
Назва	Дешифрування з правильним ключем
Передумови	Зашифроване зображення
Кроки	1. Ввести коректний ключ
	2. Натиснути "Дешифрувати"
Очікуваний результат	Файл дешифрований і доступний для завантаження
Фактичний результат	Файл успішно дешифрований і збережений

Таблиця 4.7 – Тест-кейс для тестування ПЗ

Поле	**Опис**
Назва	Дешифрування з неправильним ключем
Передумови	Зашифроване зображення
Кроки	1. Ввести некоректний ключ
	2. Натиснути "Дешифрувати"
Очікуваний результат	Виведення повідомлення про помилку дешифрування
Фактичний результат	Повідомлення про помилку дешифрування успішно виведено

Таблиця 4.8 – Тест-кейс для тестування ПЗ

Поле	**Опис**
Назва	Завантаження непідтримуваного формату файлу
Передумови	Відкритий веб-застосунок
Кроки	1. Завантажити файл у непідтримуваному форматі
Очікуваний результат	Виведення повідомлення про непідтримуваний формат
Фактичний результат	Повідомлення про непідтримуваний формат успішно виведено

Таблиця 4.9 – Тест-кейс для тестування ПЗ

Поле	**Опис**
Назва	Завантаження порожнього файлу
Передумови	Відкритий веб-застосунок
Кроки	1. Завантажити порожній файл
Очікуваний результат	Виведення повідомлення про помилку
Фактичний результат	Повідомлення про помилку при завантаженні порожнього файлу успішно виведено

Таблиця 4.10 – Тест-кейс для тестування ПЗ

Поле	**Опис**

Назва	Завантаження не зображення
Передумови	Відкритий веб-застосунок
Кроки	1. Завантажити текстовий файл
Очікуваний результат	Виведення повідомлення про неправильний тип файлу
Фактичний результат	Повідомлення про неправильний тип файлу успішно виведено

Таблиця 4.11 – Тест-кейс для тестування ПЗ

Поле	**Опис**
Назва	Видалення зашифрованого файлу після завантаження
Передумови	Зашифроване зображення
Кроки	1. Завантажити зашифрований файл
	2. Перевірити видалення
Очікуваний результат	Файл видаляється після успішного завантаження
Фактичний результат	Файл успішно видалено після завантаження

4.3 Оцінка ефективності програмного забезпечення

Оцінка ефективності програмного забезпечення для шифрування і дешифрування може бути проведена за двома основними критеріями: час обробки файлів та кількість використаної пам'яті під час виконання процесів.

Одним з ключових аспектів ефективності є швидкість шифрування та дешифрування зображень. Для цього можна виміряти час, необхідний для шифрування різних файлів, а також час, потрібний для дешифрування зашифрованих файлів. Тестування слід проводити з файлами різних форматів і розмірів (наприклад, 1 МБ, 5 МБ, 10 МБ), щоб оцінити, як програмне забезпечення працює в різних умовах.

Критерії оцінки:

- чим менший час обробки файлів, тим ефективніше працює програмне забезпечення;

- швидке виконання операцій з великими файлами свідчить про високу продуктивність ПЗ.

Додатковим важливим показником ефективності є кількість оперативної пам'яті, що використовується під час шифрування і дешифрування. Це дозволяє оцінити, наскільки оптимально ПЗ використовує ресурси системи. Для цього варто відстежувати пік використання пам'яті під час обробки файлів різного розміру.

Критерії оцінки використання пам'яті:

- програмне забезпечення повинно споживати мінімальну кількість пам'яті для малих файлів і пропорційно збільшувати використання для більших файлів, але не перевищувати очікувані значення;
- надмірне споживання ресурсів або витоки пам'яті свідчать про неефективність роботи ПЗ;
- оптимальне використання пам'яті при обробці великих файлів (до 100-200 МБ) є ознакою добре оптимізованого програмного забезпечення [24].

Поєднання часу шифрування/дешифрування та кількості використаної пам'яті дає повну картину продуктивності програмного забезпечення. Якщо програма працює швидко і не використовує надмірні ресурси навіть для великих файлів, це свідчить про її високу ефективність та оптимальне управління ресурсами.

Щоб отримати ці показники ефективності роботи програмного забезпечення, було додано кілька команд до початкового коду ПЗ, як показано на Рис. 4.1-4.2.

```
var startTime = performance.now(); // Початок вимірювання часу
var encryptedText = CryptoJS.TripleDES.encrypt(plainText, secretKey); // Шифруємо зображення
var endTime = performance.now(); // Кінець вимірювання часу

// Вимірювання використаної пам'яті (може не підтримуватись в усіх браузерах)
var usedMemory = performance.memory.usedJSHeapSize / (1024 * 1024); // Перетворюємо в МБ

saveAs(new File([encryptedText], fileName, {type: "text/plain;charset=utf-8"})); // Зберігаємо зашифрований файл

// Виводимо результати
document.getElementById('timeResult').innerText = "Час шифрування: " + (endTime - startTime).toFixed(2) + " мс";
document.getElementById('memoryResult').innerText = "Використано пам'яті: " + usedMemory.toFixed(2) + " МБ";
```

Рисунок 4.1 – Замір часу для шифрування зображення

```
var startTime = performance.now(); // Початок вимірювання часу
try {
    // Дешифрування за допомогою бібліотеки CryptoJS
    var decryptedText = CryptoJS.TripleDES.decrypt(encryptedText, secretKey).toString(CryptoJS.enc.Utf8);
    var endTime = performance.now(); // Кінець вимірювання часу

    // Створення посилання для завантаження дешифрованого файлу
    var element = document.createElement('a');
    element.setAttribute('href', decryptedText);
    element.setAttribute('download', fileName);
    element.style.display = 'none';
    document.body.appendChild(element);
    element.click(); // Завантаження файлу
    document.body.removeChild(element);

    // Виводимо результати
    document.getElementById('timeResult').innerText = "Час дешифрування: " + (endTime - startTime).toFixed(2) + " мс";
    // Вимірювання використаної пам'яті (може не підтримуватись в усіх браузерах)
    var usedMemory = performance.memory.usedJSHeapSize / (1024 * 1024); // Перетворюємо в МБ
    document.getElementById('memoryResult').innerText = "Використано пам'яті: " + usedMemory.toFixed(2) + " МБ";
}
```

Рисунок 4.2 – Замір часу для дешифрування зображення

Отримані результати занесено до Таблиці 4.1, яка стала основою для діаграм, що більш зрозуміло та наглядно відображають залежність розміру зображення та необхідних ресурсів машини (часу виконання та кількості оперативної пам’яті), які наведено у вигляді Рис. 4.3-4.6.

Таблиця 4.1 – Результати тестування ефективності ПЗ

Розмір зображення, кб	**Час шифрування, мс**	**Пам'ять для шифрування, Мб**	**Час дешифрування, мс**	**Пам'ять для дешифрування, Мб**
271	782	11,35	474	19,97
57	298	11,35	289	19,97
501	2511	11,35	2545	19,97
36	169	11,35	173	19,97
130	815	11,35	825	19,97
16	68	11,35	70	19,97
1045	9148	11,35	9683	19,97

Рисунок 4.3 – Діаграма залежності використаних ресурсів від розміру зображення для шифрування

Рисунок 4.4 – Діаграма залежності використаних ресурсів від розміру зображення для шифрування

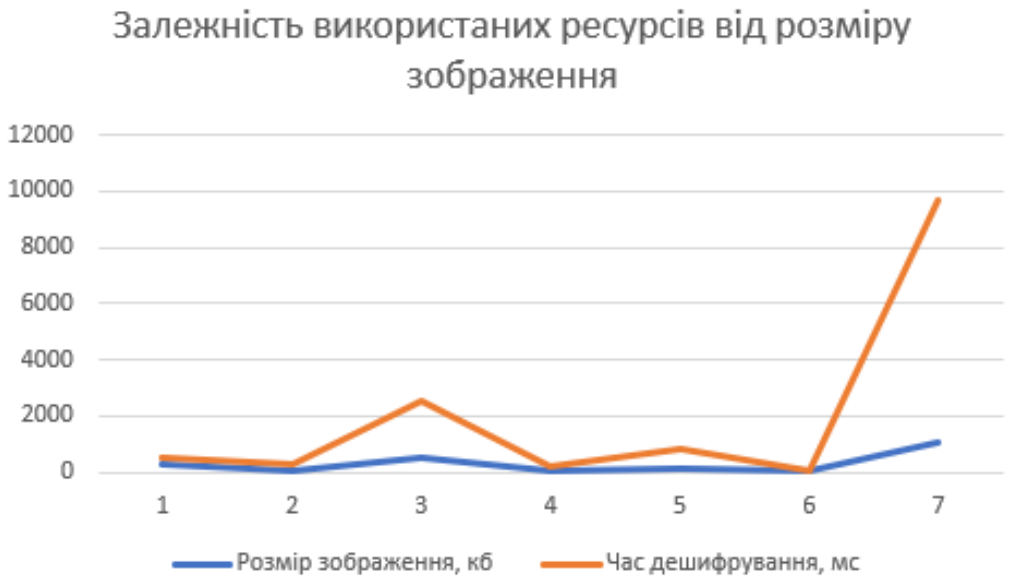

Рисунок 4.5 – Діаграма залежності використаних ресурсів від розміру зображення для дешифрування

Рисунок 4.6 – Діаграма залежності використаних ресурсів від розміру зображення для дешифрування

Таким чином, обидва процеси шифрування і дешифрування демонструють прямий зв'язок між часом обробки і розміром зображення, з значним збільшенням часу для великих файлів.

Використання пам'яті під час шифрування і дешифрування є стабільним і не залежить від розміру зображення, що вказує на ефективність алгоритму з

точки зору споживання ресурсів.

Оптимізація алгоритмів або покращення апаратних ресурсів можуть бути необхідними для зменшення часу обробки великих зображень, що може бути важливим для додатків з високими вимогами до швидкості, однак для даної роботи такі результати ефективності більш, ніж влаштовують автора, тому немає потреби.

ВИСНОВКИ

У ході виконання дипломної було створено веб-застосунок, який забезпечує ефективне та надійне шифрування й розшифрування зображень за допомогою алгоритму Triple DES. Цей результат дозволяє вирішити актуальну задачу захисту зображень у цифрових системах від несанкціонованого доступу.

Веб-застосунок був розроблений з використанням сучасних технологій веб-розробки, що дозволяє легко інтегрувати його в існуючі системи захисту інформації. Окрім цього, було забезпечено інтуїтивно зрозумілий інтерфейс для користувачів, що підвищує зручність роботи з застосунком.

Практична значущість розробки полягає в тому, що веб-застосунок може бути використаний для захисту конфіденційних даних як приватними користувачами, так і організаціями, що працюють з чутливими візуальними матеріалами. Він також може бути базою для подальших удосконалень або розширення функціональності, зокрема шляхом інтеграції інших криптографічних алгоритмів.

Таким чином, виконана робота має як теоретичну, так і практичну цінність. Вивчення та впровадження криптографічних методів у веб-застосунки є важливим напрямком сучасних досліджень у галузі кібербезпеки, і розроблений веб-застосунок може слугувати важливим інструментом для захисту даних в інформаційному середовищі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Агравал М., Каял Н., Саксена Н. PRIMES is in P // Annals of Mathematics. 2004.

2. Гілберт Е. Н., МакВільямс Ф. Дж., Слоан Н. Дж. А. Codes which detect deception // Bell Systems Technical Journal. 1974.

3. Барбулеску Р., Годрі П., Жу А., Томе Е. A heuristic quasi-polynomial algorithm for discrete logarithm in finite fields of small characteristic. 2014.

4. Chapman & Hall. Security. Introduction to Modern Cryptography [Електронний ресурс]. URL: https://eclass.uniwa.gr/modules/document/file.php/CSCYB105/Reading%20Material/%5BJonathan_Katz%2C_Yehuda_Lindell%5D_Introduction_to_Mo%282nd%29.pdf

5. Голдрайх О. Foundations of Cryptography. Vol. 2: Basic Applications. Cambridge: Cambridge University Press, 2004.

6. Абдалла М., Белларе М., Рогауей П. The oracle Diffie-Hellman assumptions and an analysis of DHIES. 2001.

7. Фіат А., Шамір А. How to prove yourself: Practical solutions to identification and signature problems.

8. Івата Т., Куросава К. OMAC: One-key CBC MAC. 2003.

9. Christof Paar, Jan Pelzl. Understanding Cryptography [Електронний ресурс]. URL: https://gnanavelrec.wordpress.com/wp-content/uploads/2019/06/2.understanding-cryptography-by-christof-paar-.pdf

10. Адамс К., Ллойд С. Understanding PKI: Concepts, Standards, and Deployment Considerations. 2nd ed. 2002.

11. Кац Дж., Юнг М. Unforgeable encryption and chosen ciphertext secure modes of operation. 2000.

12. Белларе М., Голдрайх О., Мітіагін А. The power of verification queries in message authentication and authenticated encryption.

13. Nigel Smart. Cryptography: An Introduction (3rd Edition) [Електронний ресурс]. URL: https://www.cs.umd.edu/~waa/414-F11/IntroToCrypto.pdf

14. Digital Image Encryption Techniques: Article Review [Електронний ресурс]. URL: https://www.researchgate.net/publication/358894343_Digital_Image_Encryption_Techniques_Article_Review

15. crypto-js [Електронний ресурс]. URL: https://www.npmjs.com/package/crypto-js

16. A Brief Overview of Crypto-js [Електронний ресурс]. URL: https://medium.com/@steffisbootcampdrive/a-brief-overview-of-crypto-js-in-2020-28dea0ca6650

17. Introduction to JavaScript [Електронний ресурс]. URL: https://www.geeksforgeeks.org/introduction-to-javascript/

18. What is JavaScript? [Електронний ресурс]. URL: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript

19. What are HTML and CSS used for? The basics of coding for the web [Електронний ресурс]. URL: https://www.futurelearn.com/info/blog/what-are-html-css-basics-of-coding

20. The Sublime Text Code Editor – An In-Depth Review [Електронний ресурс]. URL: https://www.elegantthemes.com/blog/resources/the-sublime-text-code-editor-an-in-depth-review

21. АлФардан Н. Дж., Бернстейн Д. Дж., Патерсон К. Г., Петтерінг Б., Шульдт Дж. К. Н. On the security of RC4 in TLS and WPA. 2013.

22. Checklist VS Test Case Set [Електронний ресурс]. URL: https://medium.com/@case_lab/checklist-vs-test-case-set-3b304aed7a2d#:~:text=A%20Testing%20Checklist%20is%20a,testing%20tasks%20like%20regression%20testing

23. How to write Test Cases in Software Testing? (with Format & Example) [Електронний ресурс]. URL: https://www.browserstack.com/guide/how-to-write-test-cases#:~:text=A%20test%20case%20refers%20to,necessary%20to%20verify%20a%20feature

24. What is Software Evaluation, & How to do it Effectively? [Електронний ресурс]. URL: https://testsigma.com/blog/software-evaluation/

ДОДАТОК А

ВИХІДНИЙ КОД ОСНОВНИХ КОМПОНЕНТІВ

Файл dashboard.html

```
<!DOCTYPE html>
<html>
<head>
 <title>Ласкаво просимо</title> <!-- Заголовок сторінки -->
 <link rel="stylesheet" href="./css/style_dashboard.css"> <!-- Підключення
стилів -->
</head>
<body>
<!-- Контейнер для розміщення елементів інтерфейсу -->
<div class="container" onclick="onclick">
  <!-- Верхня частина контейнера -->
  <div class="top"></div>

  <!-- Нижня частина контейнера -->
  <div class="bottom"></div>

  <!-- Центральна частина з основним вмістом -->
  <div class="center">
   <!-- Заголовок з описом функціональності -->
   <h2 style="font-size: 30px; text-align:
center;">Шифрування/Дешифрування Triple DES</h2>

   <!-- Кнопка для переходу на сторінку шифрування -->
   <button onclick="location.href='./encrypt.html'">Шифрувати</button>

   <!-- Кнопка для переходу на сторінку дешифрування -->
   <button onclick="location.href='./decrypt.html'">Дешифрувати</button>

   <h2> </h2>
  </div>
</div>
</body>
</html>
```

Файл decrypt.html

```
<!DOCTYPE html>
<html lang="uk"> <!-- Мова сторінки -->
<head>
 <!-- Підключення бібліотеки для криптографії -->
 <script src="./js/crypto-js.js"></script>
 <meta charset="UTF-8">
 <title>Дешифрування</title> <!--Заголовок сторінки -->
```

```
 <link rel="stylesheet" href="./css/style.css">
</head>
<body>
 <!-- Кнопка для повернення на попередню сторінку -->
 <div>
  <a href="javascript:history.back()" style="text-decoration:none; font-
size: 24px;">
   ← Повернутися назад
  </a>
 </div>

 <!-- Заголовок сторінки -->
 <h1 id="title">Дешифруйте ваше зображення</h1>

 <!-- Зона завантаження файлу -->
 <div class="zone">
  <div id="dropZ">
   <i class="fa fa-cloud-upload"></i>
   <div>Перетягніть файл сюди</div>
   <span>АБО</span>
   <div class="selectFile">
    <label for="file">Виберіть файл</label>
    <input type="file" name="files[]" id="file" onchange="previewFile()">
<!-- Поле для вибору файлу -->
   </div>
   <p>Обмеження на розмір файлу: 10 МБ</p>
  </div>
 </div>

 <!-- Зона для відображення результатів дешифрування -->
 <div class="zone-image">
  <textarea id="content" class="result" rows="20" cols="50"></textarea>
  <label>
   <input type="text" id="secret" placeholder="Введіть свій секретний
ключ"> <!-- Поле для введення ключа -->
   <button onclick="decryptFromDES()">Дешифрувати</button>
  </label>
 </div>

 <script>
  // Функція для перегляду вмісту файлу
  function previewFile() {
    const content = document.querySelector('.result'); // Відображення
результату в текстовому полі
    const [file] = document.querySelector('input[type=file]').files; //
Отримання вибраного файлу
    const reader = new FileReader(); // Читання файлу

    // Коли файл завантажено, показати його в текстовому полі
    reader.addEventListener("load", () => {
      content.innerText = reader.result;
```

```
    }, false);

    if (file) {
      reader.readAsText(file); // Читання файлу як текст
    }
  }

  // Функція для дешифрування зображення за допомогою Triple DES
  function decryptFromDES() {
    var secretKey = document.getElementById('secret').value; // Отримання
секретного ключа
    if(secretKey != ''){
     var fileName = "decrypted_image"; // Назва дешифрованого файлу
     var encryptedText = document.getElementById('content').value; //
Отримання зашифрованого тексту
     try {
       // Дешифрування за допомогою бібліотеки CryptoJS
       var decryptedText = CryptoJS.TripleDES.decrypt(encryptedText,
secretKey).toString(CryptoJS.enc.Utf8);

       // Створення посилання для завантаження дешифрованого файлу
       var element = document.createElement('a');
       element.setAttribute('href', decryptedText);
       element.setAttribute('download', fileName);
       element.style.display = 'none';
       document.body.appendChild(element);
       element.click(); // Завантаження файлу
       document.body.removeChild(element);
     }
     catch(err) {
       // Повідомлення про помилку, якщо ключ неправильний
       alert("Помилка у вашому документі або секретний ключ невірний!");
     }
    }
    else {
     // Повідомлення, якщо не введено секретний ключ
     alert("Будь ласка, введіть свій секретний ключ!");
    }
  }
 </script>
</body>
</html>
```

Файл encrypt.html

```
<!DOCTYPE html>
<html lang="uk"> <!-- Встановлення української мови -->
<head>
 <!-- Підключення бібліотек для криптографії та збереження файлів -->
 <script src="./js/crypto-js.js"></script>
```

```
 <script src="./js/FileSaver.js"></script>
 <meta charset="UTF-8">
 <title>Шифрування</title> <!-- Заголовок сторінки -->

 <link rel="stylesheet" href="./css/style.css">

</head>
<body>
 <!-- Кнопка для повернення на попередню сторінку -->
 <div>
  <a href="javascript:history.back()" style="text-decoration:none; font-
size: 24px;">
   ← Повернутися назад
  </a>
 </div>

 <!-- Заголовок сторінки -->
 <h1 id="title">Зашифруйте ваше зображення</h1> <!-- Заголовок сторінки --
>

 <!-- Зона для завантаження файлу -->
 <div class="zone">
  <div id="dropZ">
   <i class="fa fa-cloud-upload"></i>
   <div>Перетягніть файл сюди</div> <!-- Опис дії -->
   <span>АБО</span>
   <div class="selectFile">
    <label for="file">Виберіть файл</label> <!-- Вибір файлу -->
    <input type="file" accept="image/*" name="files[]" id="file"
onchange="loadFile(event)">
   </div>
   <p>Обмеження на розмір файлу: 10 МБ</p> <!-- Опис обмеження -->
  </div>
 </div>

 <!-- Зона для відображення зображення -->
 <div class="zone-image">
  <img id="output" width="400" style="margin-top: 30px; position:
relative;"/>
  <label>
   <input type="text" id="secret" placeholder="Введіть свій секретний
ключ"> <!-- Поле введення секретного ключа -->
   <button onclick="encryptToDES()">Зашифрувати</button> <!-- Кнопка
шифрування -->
  </label>
 </div>

 <script>
  var data; // Змінна для збереження даних зображення

  // Функція для завантаження зображення
```

```
  var loadFile = function(event) {
    var image = document.getElementById('output');
    image.src = URL.createObjectURL(event.target.files[0]);
  };

  var img = document.getElementById('output');
  img.crossOrigin = 'Anonymous';

  // Після завантаження зображення перетворюємо його в base64
  img.onload = function(){
    var canvas = document.createElement('canvas');
    var ctx = canvas.getContext('2d');
    canvas.height = this.naturalHeight;
    canvas.width = this.naturalWidth;
    ctx.drawImage(this, 0, 0);
    data = canvas.toDataURL('image/jpeg'); // Отримуємо дані зображення у
форматі base64
  };

  // Функція для шифрування зображення за допомогою Triple DES
  function encryptToDES() {
   var secretKey = document.getElementById('secret').value; // Отримуємо
секретний ключ
   if(secretKey != ''){
    var fileName = "encrypted_image"; // Ім'я зашифрованого файлу
    var plainText = data; // Дані зображення
    var encryptedText = CryptoJS.TripleDES.encrypt(plainText, secretKey);
// Шифруємо зображення
    saveAs(new File([encryptedText], fileName, {type:
"text/plain;charset=utf-8"})); // Зберігаємо зашифрований файл
   }
   else {
    alert("Будь ласка, введіть свій секретний ключ!"); // Попередження про
відсутність ключа
   }
  }
 </script>
</body>
</html>
```

Файл style.css

```
@font-face {
 font-family: 'Roboto'; /* Завантаження шрифту Roboto */
 font-style: normal;
 font-weight: 300;
 src:
url('https://fonts.googleapis.com/css2?family=Roboto:wght@300&display=swap
') format('truetype');
}
```

```
@font-face {
 font-family: 'Roboto';
 font-style: normal;
 font-weight: 400;
 src:
url('https://fonts.googleapis.com/css2?family=Roboto:wght@400&display=swap
') format('truetype');
}
@font-face {
 font-family: 'Roboto';
 font-style: normal;
 font-weight: 500;
 src:
url('https://fonts.googleapis.com/css2?family=Roboto:wght@500&display=swap
') format('truetype');
}
@font-face {
 font-family: 'Roboto';
 font-style: normal;
 font-weight: 700;
 src:
url('https://fonts.googleapis.com/css2?family=Roboto:wght@700&display=swap
') format('truetype');
}
body {
 background: #f0f4f8; /* Колір фону */
 font-family: "Roboto"; /* Використання шрифту Roboto */
}
.zone {
 margin: auto;
 position: absolute;
 top: 10%;
 left: 0;
 bottom: 0;
 right: 40%;
 background-color:#e9873b;
 width: 40%;
 height: 50%;
 border: 5px dashed white;
 text-align: center;
 color: white;
 z-index: 20;
 transition: all 0.3s ease-out;
 box-shadow: 0 0 0 1px rgba(255, 255, 255, 0.05), inset 0 0 0.25em 0
rgba(0, 0, 0, 0.25);
}

.zone-image {
 margin: auto;
 position: absolute;
 top: 10%;
```

```
 left: 50%;
 bottom: 0;
 right: 0;
 background-color:#e9873b;
 width: 450px;
 height: 450px;
 border: 5px dashed white;
 text-align: center;
 color: white;
 z-index: 20;
 transition: all 0.3s ease-out;
 box-shadow: 0 0 0 1px rgba(255, 255, 255, 0.05), inset 0 0 0.25em 0
rgba(0, 0, 0, 0.25);
}

.zone .btnCompression .active {
 background: #eb6a5a;
 color: white;
}
.zone i {
 text-align: center;
 font-size: 10em;
 color: #fff;
 margin-top: 50px;
}
.zone .selectFile {
 height: 50px;
 margin: 20px auto;
 position: relative;
 width: 200px;
}
.zone label,
.zone input {
 cursor: pointer;
 display: block;
 height: 50px;
 left: 0;
 position: absolute;
 top: 0;
 width: 100%;
 border-radius: 5px;
}
.zone label {
 background: #fff;
 color: #eb6a5a;
 font-size: 1.2em;
 line-height: 50px;
 box-shadow: 0 1px 1px gray;
}
.zone input[type="file"] {
 opacity: 0;
```

```
}
.zone.in {
 color: white;
 border-color: white;
 background-color:#e9873b;
}
.zone.in i {
 color: #fff;
}
.zone.hover {
 color: gray;
 border-color: white;
 background: #fff;
 border: 5px dashed gray;
}
.zone.hover i {
 color: #eb6a5a;
}
.zone.fade {
 transition: all 0.3s ease-out;
 opacity: 1;
}

#dropZ {
 margin-top: 21%;
}

#title {
 text-align: center;
 color: rgb(0, 0, 0);
 font-size: 60px;
 margin-top: 4%;
}

label {
 position: absolute;
 top: 83%;
 right: 3.7%;
}
input[type="text"] {
 border: none;
 width: 245px;
 height: 40px;
 padding: 0 10px;
 border: 2px solid black;
}
button {
 position: relative;
 background: #d39e2b;
 color: white;
 padding: 0 10px;
```

```
 letter-spacing: 1.2px;
 border: 2px solid black;
 cursor: pointer;
 height: 44px;
}

.result {
 position: relative;
 top: 5%;
}

button{
 background-color:#60b8d4; /* Початковий колір кнопки */
  color: white; /* Колір тексту */
  border: none; /* Без обрамлення */
  cursor: pointer; /* Курсор при наведенні */
  transition: background-color 0.2s ease; /* Плавний перехід кольору */
}

button:hover{
 cursor: pointer;
 background-color: #e46569; /* Напівпрозорий колір при наведенні */

}

a{
 color: black;
}
```

Файл style_dashboard.css

```
@import url("https://fonts.googleapis.com/css?family=Roboto:400,700"); /*
Завантаження шрифту Roboto */
*, *:before, *:after {
 box-sizing: border-box; /* Включення моделі коробки для всіх елементів */
}

body {
 min-height: 100vh; /* Мінімальна висота тіла */
 font-family: "Roboto", sans-serif; /* Використання шрифту Roboto */
 overflow: hidden; /* Сховати переповнені елементи */
}

.container {
 position: absolute;
 width: 100%;
 height: 100%;
 overflow: hidden;
}
```

```
.container:hover .top:before, .container:hover .top:after,
.container:hover .bottom:before, .container:hover .bottom:after,
.container:active .top:before, .container:active .top:after,
.container:active .bottom:before, .container:active .bottom:after {
 margin-left: 200px; /* Зсув елементів при наведенні */
 transform-origin: -200px 50%;
 transition-delay: 0s;
}

.container:hover .center, .container:active .center {
 opacity: 1; /* Зробити центр видимим */
 transition-delay: 0.2s;
}

.top:before, .top:after, .bottom:before, .bottom:after {
 content: "";
 display: block;
 position: absolute;
 width: 200vmax;
 height: 200vmax;
 top: 50%;
 left: 50%;
 margin-top: -100vmax;
 transform-origin: 0 50%;
 transition: all 0.5s cubic-bezier(0.445, 0.05, 0, 1);
 z-index: 10;
 opacity: 0.65;
 transition-delay: 0.2s;
}

.top:before {
 transform: rotate(45deg);
 background: #e46569; /* Колір верхнього елемента */
}
.top:after {
 transform: rotate(135deg);
 background: #ecaf81; /* Інший колір верхнього елемента */
}

.bottom:before {
 transform: rotate(-45deg);
 background: #60b8d4; /* Колір нижнього елемента */
}
.bottom:after {
 transform: rotate(-135deg);
 background: #3745b5; /* Інший колір нижнього елемента */
}

.center {
 position: absolute;
 width: 400px;
```

```
 height: 400px;
 top: 50%;
 left: 50%;
 margin-left: -200px;
 margin-top: -200px;
 display: flex;
 flex-direction: column;
 justify-content: center;
 align-items: center;
 padding: 30px;
 opacity: 0;
 transition: all 0.5s cubic-bezier(0.445, 0.05, 0, 1);
 transition-delay: 0s;
 color: #333;
}

.center button {
 width: 100%;
 padding: 15px;
 margin: 5px;
 border-radius: 1px;
 border: 1px solid #ccc;
 font-family: inherit; /* Використання шрифту за замовчуванням */
}

button{
 background-color:#60b8d4; /* Початковий колір кнопки */
  color: white; /* Колір тексту */
  padding: 15px 30px; /* Внутрішні відступи */
  border: none; /* Без обрамлення */
  border-radius: 5px; /* Закруглені краї */
  cursor: pointer; /* Курсор при наведенні */
  transition: background-color 0.2s ease; /* Плавний перехід кольору */
}

button:hover{
 cursor: pointer;
 background-color:#e46569; /* Напівпрозорий колір при наведенні */

}
```