ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Навчально–науковий інститут Інформаційних технологій

Кафедра інженерії програмного забезпечення

Пояснювальна записка

до бакалаврської роботи
на ступінь вищої освіти бакалавр

на тему: «**РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ШИФРУВАННЯ ТА ДЕШИФРУВАННЯ ФАЙЛІВ МОВОЮ C#**»

Виконав: студент 4 курсу, групи ПД–43
спеціальності
121 Інженерія програмного забезпечення
(шифр і назва спеціальності)
Музика В.П.
(прізвище та ініціали)
Керівник: Трінтіна Н.А.
(прізвище та ініціали)
Рецензент _____
(прізвище та ініціали)

Київ – 2022

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення
Ступінь вищої освіти - «Бакалавр»
Спеціальність підготовки – 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
Негоденко О.В.
"____" _____2022 року

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ РОБОТУ СТУДЕНТА

МУЗИЦІ ВЛАДИСЛАВУ ПАВЛОВИЧУ
(прізвище, ім'я, по батькові)

1. Тема роботи: «Розробка програмного забезпечення для шифрування та дешифрування файлів мовою C#»

Керівник роботи: Трінтіна Наталія Альбертівна, к.т.н., доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом вищого навчального закладу від «18» лютого 2022 року №__

2. Строк подання студентом роботи «3» червня 2022 року

3. Вхідні дані до роботи:

Алгоритми шифрування та дешифрування даних;

Науково-технічна література з питань, пов'язаних з програмним забезпеченням, щодо методів шифрування та дешифрування даних;

4. Зміст розрахунково-пояснювальної записки(перелік питань, які потрібно розробити).

4.1 Криптографія. Різновиди методів шифрування

4.2 Вимоги та оцінка якості системи.

4.3 Опис проектування програми.

4.4 Опис використаних технологій.

5. Перелік демонстраційного матеріалу (назва основних слайдів)

5.1 Титульний слайд

5.2 Аналіз аналогів

5.3 Мета, об’єкт та предмет дослідження

5.4 Технічне завдання

5.5 Інструменти використані в роботі

5.6 Діаграми проектування

5.7 Екранні форми продукту

5.8 Апробація результатів дослідження

5.9 Висновки

6. Дата видачі завдання «11» квітня 2022

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	11.04.22-16.04.22	Виконано
2	Розділ 1.Криптографія. Різновиди методів шифрування	18.04.22-1.05.22	Виконано
3	Розділ 2. Вимоги та оцінка якості системи	2.05.22-9.05.22	Виконано
4	Розділ 3. Опис проектування програми	10.05.22-17.05.22	Виконано
5	Розділ 4. Опис використаних технологій	18.05.22-24.05.22	Виконано
6	Вступ, висновки, реферат	25.05.22-27.05.22	Виконано
7	Розробка обов’язкових демонстраційних матеріалів	28.05.22	Виконано
8	Попередній захист роботи	31.05.22	Виконано
9	Здача роботи	3.06.22	

Студент _____ Музика В.П.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Трінтіна Н.А.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Текстова частина дипломної роботи 60 с., 46 рис., 15 джерел.

ШИФРУВАННЯ ТА ДЕШИФРУВАННЯ ФАЙЛІВ, МОВА ПРОГРАМУВАННЯ C#, КРИПТОГРАФІЧНІ АЛГОРИТМИ ПЕРЕТВОРЕННЯ ІНФОРМАЦІЇ.

Об’єкт дослідження – захист інформації шляхом криптографічних перетворень інформації задля збереження приватності даних.

Предмет дослідження – алгоритми, методики та принципи роботи криптографічних перетворень інформації.

Метою роботи є поліпшення методик шифрування за рахунок створення програмному застосунку з можливістю послідовного об’єднання алгоритмів у нові гібридні системи.

Методи дослідження – емпіричні та теоритичні методи: тести, аналіз документів, вивчення продуктів діяльності, складення діаграм, експерементальні висновки.

Проаналізовано методи та види шифрування та принципи їх роботи.

Обрано найбільш відповідні технології для реалізації продукту, з використанням сучасних розробок в даній області з розділенням внутрішнього функціоналу на модулі відділені один від одного для найбільш ефективної розробки з виключенням помилок не локальних помилок та підтримки продукту.

Зважаючи на всі вимоги, для розробки програмного продукту використано об’єктно орієнтовану мову програмування C# з використанням крос-платформової технології .NET Framework у середовищі розробки Visual Studio.

Використано програмний інтерфейс застосунку– Windows Forms API, що є частиною технології .NET Framework. Яка надає можливість налаштувати елементи управління за власним розсудом та швидким доступом до коду.

Практично продукт може бути застосований у передачі захищених шифруванням даних, збереженням даних на локальних носіях інформації чи ПК для захисту від перегляду іншими користувачами або зловмисниками.

Галузь використання – шифрування, дешифрування файлів.

Перспективи дослідження – оновлення та модифікація реалізованих алгоритмів, впровадження новітніх .

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ПК – персональний комп’ютер.

ПЗ – програмне забезпечення.

Завдання ТП – завдання таємної передачі.

API (Application Programming Interface) – програмний інтерфейс програми.

DES (Data Encryption Standard) – стандарт шифрування даних.

DEA (Data Encryption Algorithm) – алгоритм шифрування даних.

ANSI (American National Standards Institute) – Американський інститут національних стандартів.

ISO (International Organization for Standardization) – Міжнародна організація зі стандартизації.

XOR (eXclusive OR) – Виняткова диз'юнкція.

NIST (National Institute of Standards and Technology) – Національний інститут стандартів та технологій.

AES (Advanced Encryption Standard) – розширений стандарт шифрування.

HTTP (Hyper Text Transfer Protocol) – токол передачі гіпертекстових документів.

UI (User Interface) – інтерфейс користувача.

GUI (Graphical user interface) – графічний інтерфейс користувача.

XML (Extensible Markup Language) – розширювана мова розмітки.

IDE (Integrated Development Environment) – Інтегроване середовище розробки.

SCM (Source Code Management) – Система керування версіями.

ВСТУП

У всі часи зловмисники без зупинки намагаються дізнатися персональні або секретні дані та бажали використати їх в своїх цілях, тому ця проблема була актуальна раніше та її актуальність збільшується пропорційно зі збільшенням кількості даних, що передаються у сучасному світі.

Кожен користувач ПК має право на зберігання власної, конфіденціальної інформації або бажає надсилати таємну інформацію чи дані іншому користувачу без можливості третій стороні отримати доступ до цих даних. В цьому їм може допомогти шифрування власних даних.

На сьогодні існує не так багато методів шифрування через те, що сучасні комп'ютери не можуть надати максимальний захист даних через обмежені обчислювальні можливості нинішніх технологій.

Багато методик шифрування беруть свої коріння, ще з стародавніх часів та в деякій мірі повторюють одна одну. Нові методики поєднують в собі весь досвід попередніх та додають ті технології, що доступні на сьогодні та модифікуючи їх.

Найбільшими користувачами таких технологій є великі компанії, що надають зв'язок, державні та військові структури, онлайн сервіси та банки. Їх робота прихована від звичайних користувачів, але вони гарантують безпеку наданих ними даних. Звичайна людина не може в повній мірі зрозуміти як працює шифрування її даних та може бути не впевнена в результаті та надійності його роботи.

Об'єктом дослідження є захист інформації шляхом криптографічних перетворень інформації задля збереження приватності даних.

Предметом дослідження є алгоритми, методики та принципи роботи криптографічних перетворень інформації.

Метою роботи є поліпшення методик шифрування за рахунок створення програмному застосунку з можливістю послідовного об'єднання алгоритмів у нові гібридні системи.

В процесі дослідження вирішувалися наступні завдання:

- Можливість об'єднання різних криптографічних алгоритмів.

– Створення програмного продукту для шифрування та дешифрування локальних файілв.

Методика дослідження:

Детально ознайомлено з теоретичними матеріалами, щодо внутрішнього влаштування методів, видами алгоритмів шифрування та принципом їх роботи.

Обрано найбільш відповідні технології для реалізації продукту, з використанням сучасних розробок в даній технологічній області, для створення застосунку з розділенням внутрішнього функціоналу на модулі відділені один від одного для найбільш ефективної розробки з виключенням помилок не локальних помилок та підтримки продукту.

Наукова новизна роботи полягає в застосуванні різних видів шифрування з можливістю простого об'єднання видів шифрування та реалізації швидкого доступу до програми з використанням реєстру.

Практична значущість результатів дослідження полягає у вирішенні практичної задачі та можливого проведення подальших наукових досліджень в цій області.

Практично продукт може бути застосований у передачі захищених шифруванням даних, збереженням цих даних на локальних носіях інформації чи ПК для захисту від перегляду іншими користувачами або зловмисниками, незважаючи на це даний продукт може бути застосованих необмеженою кількістю користувачів.

1 КРИПТОГРАФІЯ. РІЗНОВИДИ МЕТОДІВ ШИФРУВАННЯ

1.1 Предмет криптографії

У всі часи людство було пов’язано з передачею інформації один від одного, але деяка інформація не повинна досягати певних осіб, або навпаки адресуватися певній особі, та не бути перехопленою небажаними індивідами. Така проблема практичної передачі секретної інформації отримала назву «завдання ТП» (таємного пересилання). Спроби вирішити це завдання тривають не одну тисячу років, за цей час було винайдено сотні або й тисячі варіантів секретної передачі даних, деякі з них повторювали один одного. Винайти панацею щоб вирішила цю проблему раз і назавжди не є можливим на сьогодні, тому дана проблема є безсуперечно актуальною в сучасному світі та напевно і в далекому майбутньому.

Є різні підходи до вирішення даного завдання, а саме:

- Створення абсолютно надійного, закритого від сторонніх обличь, каналу зв’язку між абонентами.
- Використання публічного каналу зв’язку, але приховання факту передачі інформації.
- Використання загальнодоступного каналу зв’язку з передачею по ньому інформації, що була модифікована чи трансформована, з обліком на те, що тільки отримувач інформації зможе її відновити.

Основною з проблем в першому випадку є те, що у сучасному світі рівень наукового та технологічного розвитку є недостатнім для створення багаторазового віддаленого каналу зв’язку з можливістю передачі великих об’ємів інформації.

В іншому випадку, це завдання намагається вирішити стенографія, приховуючи факт передачі інформації, наприклад обхід систем моніторингу промислового шпигунства, захист авторського права шляхом нанесення невидимих для людини знаків на зображення, тощо, але розпізнаються спеціальним ПЗ або камуфлювання ПЗ під видом стандартного. Стенографію часто відносять до

криптографії, але ці поняття принципово відрізняються за своїми теоретичними та практичними напрямками. Проблемою такого підходу є широке поширення методів приховування.

Останнім підходом до вирішення цього завдання займається наука криптографія, використовуючи фундаментальні науки такі як математика для трансформування інформації з ціллю її захисту від небажаних користувачів, тобто створюють шифри. Де шифрування є процесом застосування цих модифікацій інформації, що захищається, перетворюючи відкритий текст на шифротекст, використовуючи математичні формули, що містяться у шифрі. Та дешифрування, що виконує обернену функцію повертаючи відкритий текст з шифротексту з використанням схожих функцій шифру. Проблемою даного підходу є те, що криптографія є також залежною від рівню розвитку технологій, засобів зв'язку та методів передачі даних. Тобто захист на пряму залежить від обчислювальних можливостей системи, а не від шифру, де найбезпечніший шифр не можливо реалізувати на нинішніх системах через недостатню продуктивність та швидкість його обробки.

Якщо підсумувати, що предметом криптографії є захист інформації, що її потребує. Така інформація повинна бути прихована від пересічних осіб, бо вона містить таємницю, яка конфіденціальною чи секретною. Прикладом такої інформації може бути державна таємниця, воєнна, комерційна, юридична або лікарська таємниці. Такою інформацією можуть володіти тільки невеликий круг користувачів, де інша сторона зловмисників, користувачів від яких йде захист, бажає заволодіти цією інформацією, для використання її в своїх цілях, або для маніпулювання та її зміненням.

Криптографія займається перетворенням інформації в неможливий для звичайного прочитання текст чи шифр, тим самим перешкоджаючи зловмиснику який зміг перехопити дані повідомлення, що були переслані через відкритий канал зв'язку. Тобто інформація, яку отримав зловмисник представлена для нього у вигляді шифру, який він повинен зламати аби отримати відкритий текст, не знаючи ні типу шифрування, ні ключа за допомогою якого він був зашифрований.

Також на ряду з криптографією існують інші дисципліни, що відіграють безпосередню роль в розробці, тестуванні та підтримці напрямку криптографііїї, такі як:

Криптосинтез – галузь, що займається безпосередньо розробкою нових криптографічних засобів захисту інформації.

Криптоаналіз – займається пошуком методів та засобів для злому криптографічних схем.

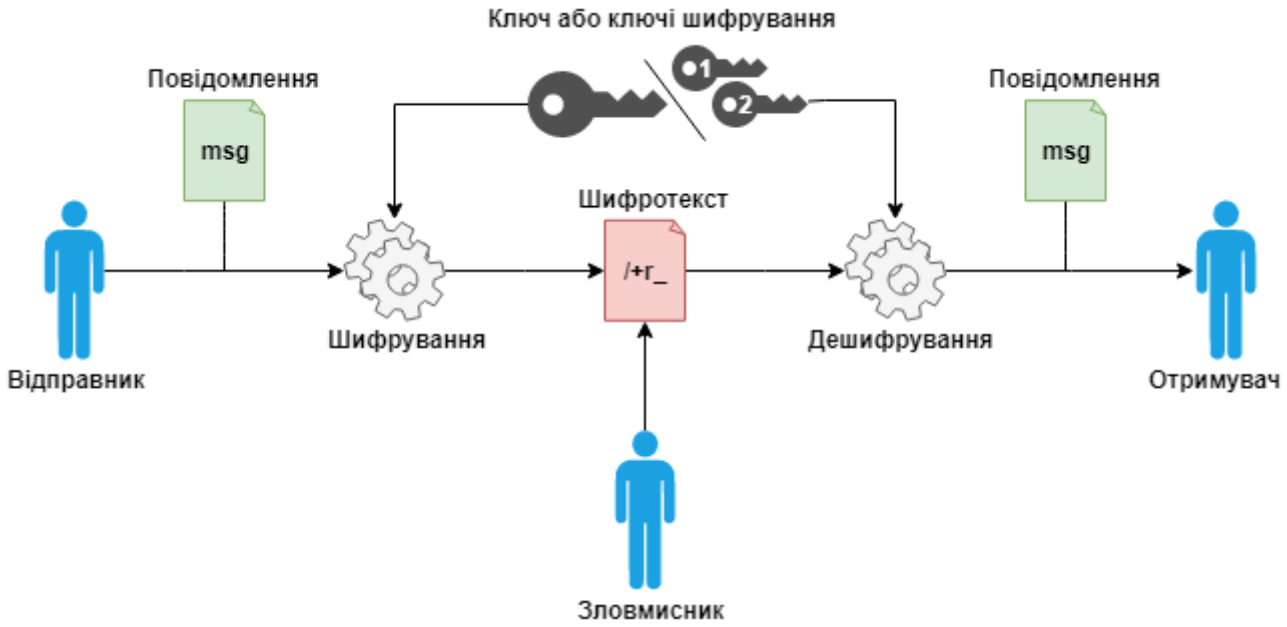


Рисунок 1.1.1. – Типова криптографічна ситуація

В цій схемі відправник та отримувач є легальними користувачами між якими проходить передача інформації через загальнодоступний, публічний канал зв’язку. Зловмисник – незаконний користувач, ціль якого перехоплення повідомлення з подальшим вилученням цінної для нього інформації.

Повідомлення подається від відправника та обробляється шифруванням, що перетворює інформацію в шифротекст, але так як цей канал передачі є відкритим, то зловмисник може отримати доступ до цього шифротексту. В ситуації коли зловмисник не грає ролі, ця інформацію надходить до отримувача, де

дешифрується та повертає шифротекст до початкового його виду. Одну з основних ролей грає ключ.

В криптографії ключ позначає змінний елемент шифру, що застосовується під час операцій шифрування та дешифрування. В залежності від виду шифрування кількість ключів може відрізнятися. Зловмисник не має доступу до ключа, тому він залишається з власною задачею без ключа відтворити шифр в зворотному порядку для отримання бажаної інформації або в деяких випадках методом підбору дізнатися сам ключ.

Ще одною з проблем криптографії є оцінка відношення цінності інформації із відношенням витрат на її захист та витрат на видобутку інформації шляхом злому. Ця проблема породжує такі питання:

- Чи є цінність інформації вищою для злочинця ніж ціна затрачена на процес її злому;
- Наскільки інформація цінна у порівнянні з вартістю затраченою на її захист.

Саме ці питання є критичними під час вибору підходящого підходу для захисту даних.

Різна інформація потребує різного ступеня захисту, тому один єдиний шифр не може підходити для всіх випадків. Вибір методу шифрування залежить від особливостей інформації, її цінності і можливостей власників. Так для великих обсягів інформації вимагається швидкість обробки, для більш таємної інформації, наприклад державні або військові таємниці, що повинні зберігатися десятки років, а для таких, як біржові навпаки короткий термін секретності в декілька годин.

Також слід розуміти проти якого виду злочинців направлений захист, це може бути як зловмисник одинак, група злочинців так навіть і спеціально направлена державна структура.

Дану характеристику називають стійкістю шифру – здатність шифру протидіяти всіляким нападам на нього. Це поняття є основним в криптографії, стійкість не є визначеною для кожного окремо взятого шифру, тому його стійкість

оцінюють на практиці, шляхом різноманітних спроб злому та кваліфікації криптоаналітиків, що проводять атаки на шифр. Тема стійкості шифрів є дуже популярна серед спеціалістів, на протязі всього часу їх існування та породжує суперечки щодо надійності кожного з них.

1.2 Симетричне шифрування

Один з двох основних типів алгоритму шифрування, в якому ключ шифрування одночасно виступає ключем для дешифрування і навпаки, також його можна назвати, як алгоритм з єдиним секретним ключем. Такий ключ повинен зберігатися в секреті до моменту отримання даних так, як ключ є певною потаємною інформацією, за якою приховується розгадка шифру.

Такі алгоритми є дуже швидкими, якщо порівнювати з асиметричими, вони поділяються на дві категорії. Перша працює з блоками відкритого тексту, в яких біти групуються в блоки, через це й називається «блочний алгоритм». Застосовуючи до кожного блоку ключ на кожному з етапів та перемішуючи, досягаючи при цьому не відповідності біт між початковими блоками та блоками зашифрованих даних.

Друга категорія працює за схожою схемою, але оброблює відкритий текст побітно, працюючи з потоком, тому й має назву «потоковий алгоритм», також на текст попередньо накладається послідовність випадкових чисел.

Рисунок 1.2.1. Симетричне шифрування

1.2.1 DES-алгоритм симетричного шифрування

Стандарт шифрування даних DES (Data Encryption Standard) або DEA (Data Encryption Algorithm), як його найменував ANSI (American National Standards Institute) та ISO (International Organization for Standardization) – DEA-1 – є світовим стандартом, що й досі зберігає статус безпечного незважаючи на довгі роки перевірок криптоаналізом, але беручи до уваги його вік на сьогодні, можна вважати, що він в деякій мірі застарів.

DES являє собою шифр, що розбиває дані на 64-бітові блоки, де на кожен такий блок введеного відкритого тексту, виходить такого самого розміру блок шифротексту. Даний алгоритм є симетричним, тому для шифрування та дешифрування використовуються однакові алгоритми та ключ.

Ключ представляє деякий пароль, від якого напряму залежить безпека даних, його максимальна довжина складає 56 біт.

Алгоритмом його роботи є використання двох методів: зміщення та дифузії, до даних застосовується одинична комбінація методів підстановки та перестановки, що залежить від ключа. Одна така ітерація є «етапом», кількість застосувань таких етапів до відкритих даних – 16.

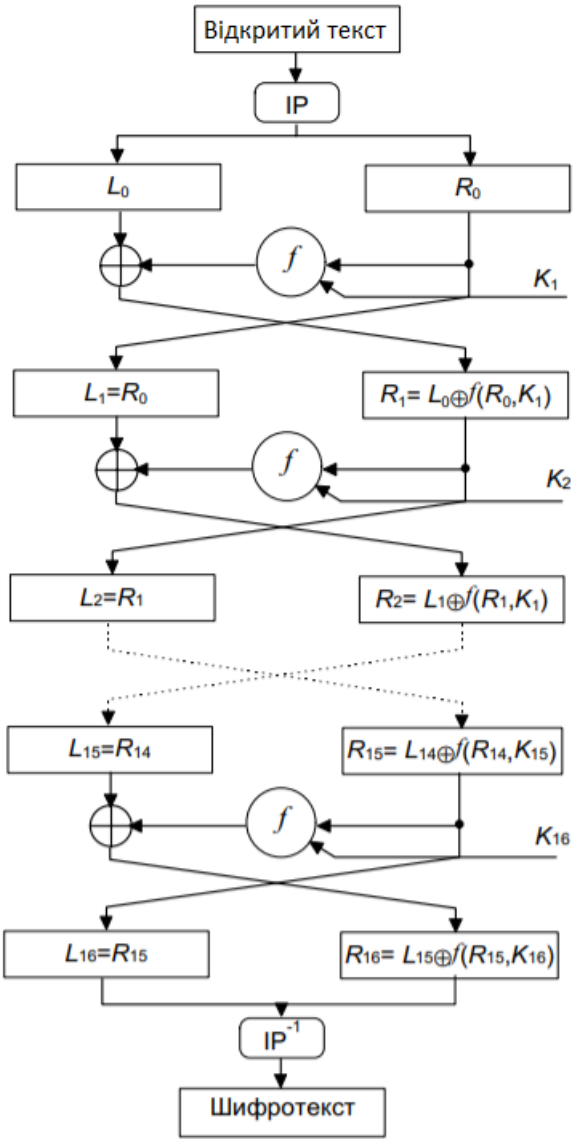


Рисунок 1.2.1.1. – Алгоритм роботи DES

Після першої перестановки в кожному етапі 64-бітовий блок розбивається на два 32-бітних блоки, далі виконується 16 однакових ітерацій етапів функції f, в яких дані об’єднуються з ключем, після останньої ітерації останній блок об’єднується завершальною зворотною перестановкою.

Під час кожного етапу біти ключа зміщуються, з 56 біт ключа обираються 48 та додаються до правої частини за допомогою перестановки з розширенням, використовуючи XOR, проходять через 8 S-блоків, утворюючи 32 нових біти та переставляються знову. Всі ці операції виконуються функцією f, далі результат цієї функції об’єднується з лівою частиною знову використовуючи XOR. В результаті

цих дій з’являється нова права половина, а ліва замінюється старою правою і так проходить кожен з 16 етапів.

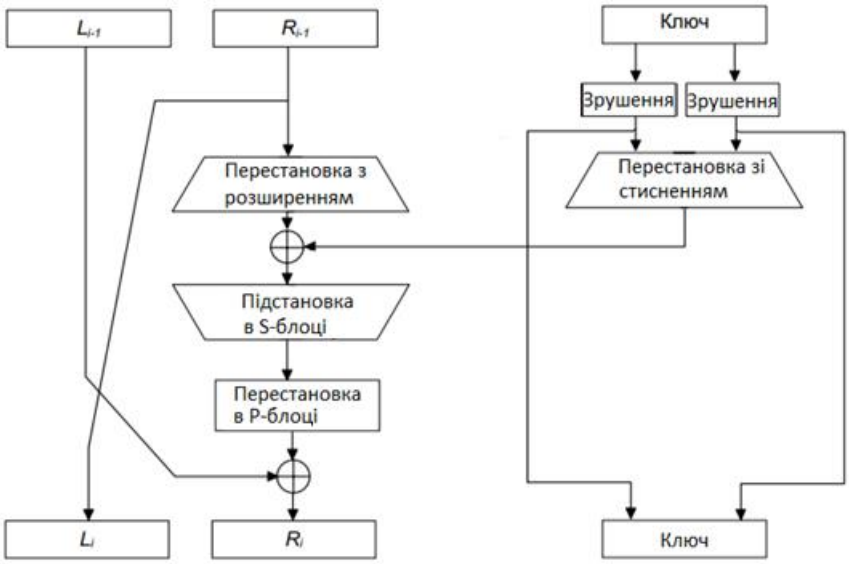


Рисунок 1.2.1.2. – Один етап DES

Якщо B_i є результатом i-тої ітерації, B_i,K_i – ключ для етапу i то даний етап можливо представити як (1.2.1.1):

$$L_i = R_{i-1}$$
$$R_i = L_{i-1} \oplus \mathrm{f}(R_{i-1}, K_i) \qquad (1.2.1.1)$$

де L_i і R_{i-} права та ліва половини, f – функція, що виконує всі підстановки та перестановки та XOR з ключем.

Для дешифрування DES використовується ті самі перетворення, тобто виконує ту саму функцію, але з однією відмінністю. Ключ використовується у зворотному порядку, також алгоритм генерації ключа в кожному етапі – циклічний.

Є 4 режими роботи визначені в FIPS PUB 81: ECB, CBC, OFB та CFB.

– ECB (electronic codebook) – найпростіший режим електронної шифрувальної книжки, в якому блок відкритого тексту замінюється на блок шифротексту і навпаки.

– CBC(cipher block chaining) – режим щеплення блоків шифру, в якому перед шифруванням відкритого тексту та попереднім блоком шифротексту здійснюється операція XOR.

– OFB(output-feedback) – режим вихідного зворотного зв’язку , метод блокового шифрування, що виступає в якості синхронного потокового шифру, схожий за принципом з CFB, з відмінністю в якій *n* біт попереднього блоку зміщуються в крайні праві позиції черги.

– CFB(cipher-feedback) – режим зворотного зв’язку по шифру, що реалізований, як самосинхронізуючий потоковий шифр, в цьому режимі шифрування починається тільки якщо був отриманий повний блок даних.

1.2.2 Алгоритм симетричного шифрування Triple DES

Найбільшим фактором ризику розкрита для звичайного DES шифрування є так зване застосування грубої сили, тобто підбір ключа на пряму. DES вважається безпечним алгоритмом, але якщо брати до уваги те, що він має невелику довжину ключа, а з розвитком технологій та обчислювальних потужностей комп’ютерів, час для операції з грубим підбором ключа зменшується пропорційно. Тому для модернізації даного методу шифрування було прийнято рішення модернізувати DES шляхом об’єднання блочного алгоритму. Одним з таких способів є багатократне шифрування, де для одного блоку відкритого тексту може використовуватися декілька ключів.

При спробі розробки подвійного DES передбачалось, що під час підбору ключа ворогу замість 2^n спроб, знадобиться 2^{2n}, але такий спосіб був приречений, так як було придумано замінити пам’ять на деякий час, що дозволило розкрити таку систему за 2^{n+1} спроб. Такий спосіб мав назву «зустріч посередині», в якому з одної сторони виконувалося шифрування, а з іншої дешифрування і при зустрічі в середині результат зрівнювався.

На заміну подвійному з’явився триразовий стандарт шифрування даних (Triple DES), в ньому було декілька варіантів.

Першим запропонованих методів потрійного шифрування був метод з двома ключами, в якому блок оброблявся тричі з використанням двох ключів, першим, другим і знову першим. Це було зроблено для запобігання розкриття методом «зустрічі посередині», але творці цього методу розробили інший спосіб розміну пам’яті на якийсь час, який дозволяє зламати цей метод шифрування за 2^{n-1} дій, використовуючи 2^n блоки пам’яті.

Виконання цього розтину з вибраним відкритим текстом вимагає великого обсягу пам’яті. Знадобиться 2^n часу та пам’яті, а також 2^m вибраних відкритих текстів. Розтин не дуже практично, але все ж таки чутливість до нього є слабкістю алгоритму.

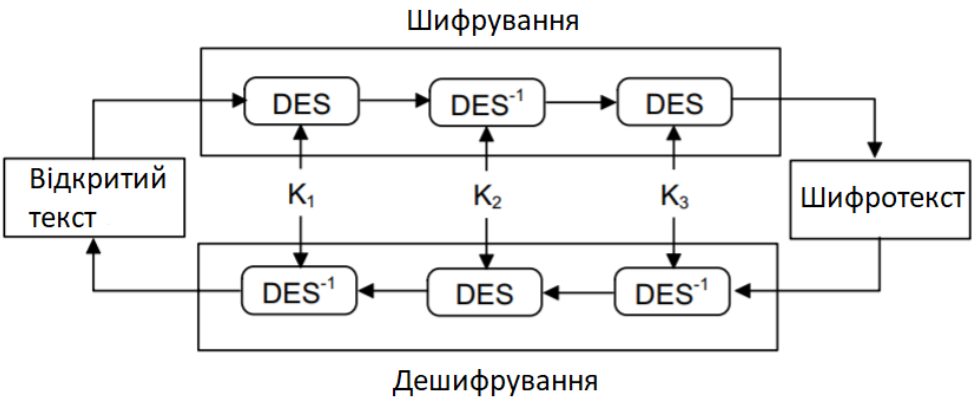


Рисунок 1.2.2.1. – TDES шифрування з двума ключами (encrypt-decrypt-encrypt, EDE)

Потрійне шифрування з трьома ключами, в якому загальна довжина ключа є більшою і за своєю безпекою представляє такий результат як від при подвійному шифруванні, тобто 2^{2n} спроб.

Також були варіанти використання гібридних та змішаних режимів шифрування, або навіть п’ятикратне шифрування, каскадні системи блочних шифрів та їх об’єднання.

1.2.3 Алгоритм симетричного шифрування AES

На заміну загальнонаціональному стандарту DES, національний інститут стандартів та технологій США (National Institute of Standards and Technology, NIST) розпочав конкурс на розробку нового стандарту, швидшого та більш гнучкого під робочою назвою AES (Advanced Encryption Standard). Переможцем конкурсу було обрано алгоритм Рейндала (Rijndael), була розпочата процедура його стандартизації.

Цей стандарт визначає алгоритм, як симетричний блоковий шифр. AES був розроблений для роботи зі збільшеними розмірами блоків і довжиною ключів.

Він може обробляти блоки даних по 128 біт, використовуючи один з трьох розмірів ключа шифрування, довжиною 128, 192 і 256 біт.

Дані завантажуються в матрицю стану, для розміру блока в 128 біт це матриця - 4x4. Алгоритм виконує 10, 12 або 14 раундів операцій в залежності від розміру ключа. В первинному раунді кожен наявний байт на вході піддається операції швидкого переключення байтів – XOR з відповідним байтом ключа першого раунду.

Для кожного з раундів потрібен ключ, його формують з початкового ключа, використовуючи змішування. Верхній байт останнього стовбця попереднього ключа переміщується під нижній, потім кожен з байтів пропускають через блок заміни. Далі проводиться операція XOR між стовбцем та «раундовою константою», що є окремою для кожного раунду. Отриманий перший стовбець знову піддається XOR з першим стовбцем попереднього раундового ключа. Інші стовбці об’єднуються операцією XOR попереднього стовбця з таким самим стовбцем попереднього раундового ключа.

Далі слідує виконання проміжних раундів, в якому відбувається перемішування для ускладнення зв’язків між байтами, кожен байт поміщається в блок заміни (S - блок), котрий відображає його в інший байт.

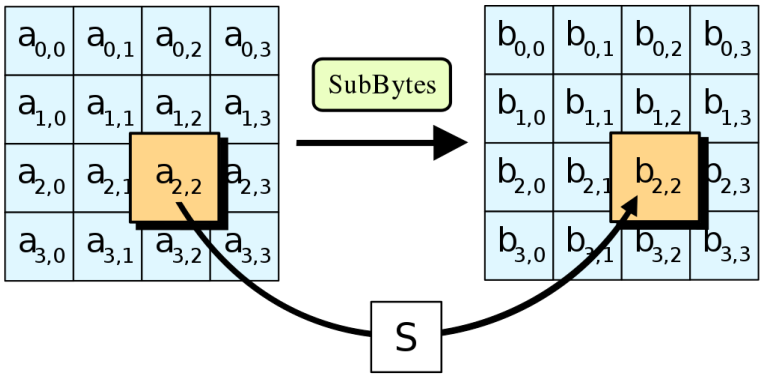


Рисунок 1.2.3.1. – Заміна байтів через S - блок

Потім застосовується розсіювання, строки зрушуються вліво та переміщуються на місця пропусків з іншої сторони.

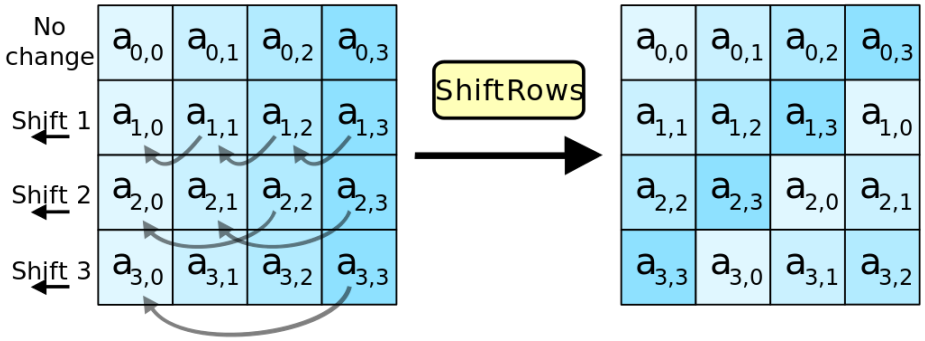


Рисунок 1.2.3.2. – Трансформація за допомогою розсіювання

Далі в кожному зі стовбців біти перемішуються.

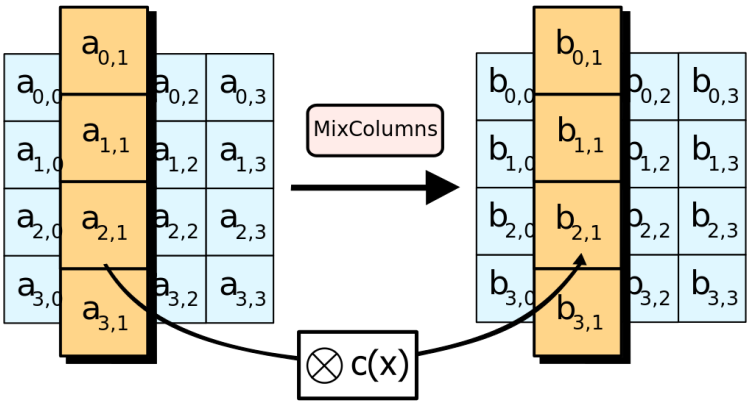


Рисунок 1.2.3.3. –Змішування стовбців

В кінці кожного раунду накладається черговий раундовий ключ за допомогою XOR

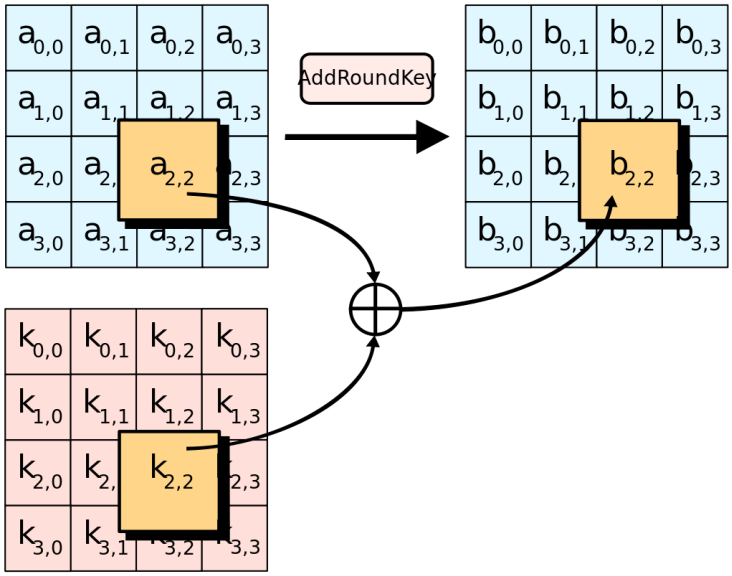


Рисунок 1.2.3.4. –Накладення раундового ключа

В останньому раунді крок зі змішуванням всередині стовбців пропускається, так як підвищити безпеку він вже не може, а лише уповільнює процес тому, що розсіювання що проходить в цьому етапі не може перейти на наступний раунд.

Від кількості раундів залежить число раундів змішування та розсіювання між бітами, що підвищує безпеку. З іншої сторони кількість раундів також впливає на на завантаження системи, тому безпека в певній мірі прямо пропорційно залежить від продуктивності.

Розшифрування в даному алгоритмі означає проходження всіх раундів в зворотному порядку.

1.3 Асиметричне шифрування

Інший підхід до методики шифрування, який представляє собою алгоритм з використанням двох ключів, першого відкритого та другого закритого. Алгоритм отримав свою назву «алгоритм з відкритим ключем» саме через те, що один з його ключів можна вільно передавати по відкритих каналах, так як він використовується лише для шифрування, зловмисник не має змоги за допомогою одного ключа здійснити обхід, вирахувати другий ключ та дешифрувати дані.

Для дешифрування використовується другий ключ – закритий. З його допомогою здійснюється дешифрування, з чого виходить, що цей ключ є секретним і доступ до нього повинна мати лише конкретна сторона.

Основним негативним аспектом такого шифрування є швидкість його роботи, алгоритми з відкритим ключем працюють в десятки, сотні або й в тисячу разів повільніше, якщо зрівнювати з симетричними, також даний алгоритм не підходить для виконання криптографічних перетворень на великих потоках даних, так як вони математично обмежені в розмірі шифрованих даних.

Цей алгоритм також використовується для формування цифрових підписів, цифрові підписи підтверджують особу відправника і допомагають захистити цілісність даних. Використовуючи відкритий ключ згенерований однією зі сторін, інша сторона може перевірити чи справді перша сторона надсилала дані, порівнявши підпис з даними та ключем відправника.

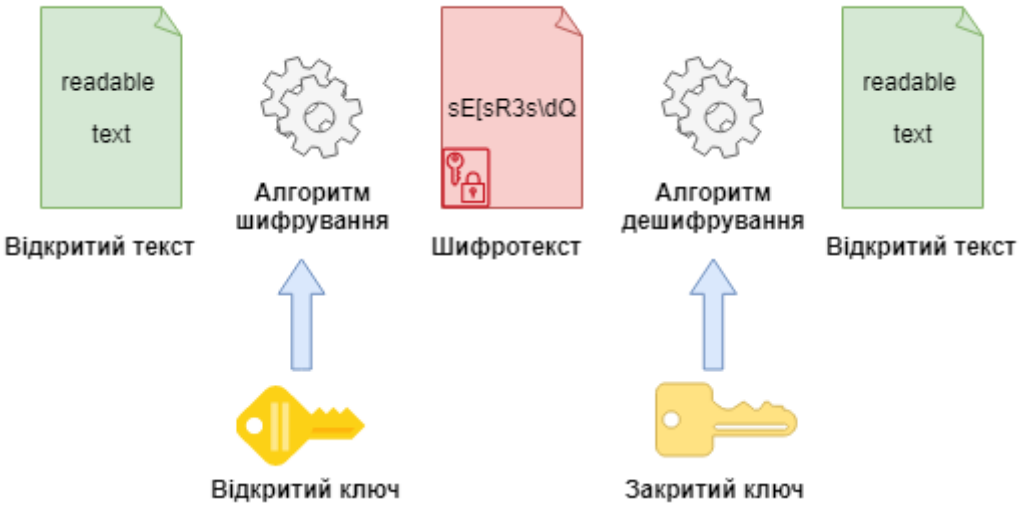


Рисунок 1.3.1 Асиметричне шифрування

1.3.1 Алгоритм асиметричного шифрування RSA

Один з найпопулярніших повноцінних алгоритмів з відкритим ключем, отримав свою назву в честь його творців Рона Рівеста (Ron Rivest), Аді Шаміра (Adi Shamir) і Леонарда Едлмана (Leonard Adleman). Довгі роки досліджень криптоаналізом не спростували його надійності, але й не обґрунтували її, але по суті рівень довіри до алгоритму не падає.

Безпека RSA заснована на труднощах розкладу на множники великих чисел так, як його відкритий та закритий ключі виступають в ролі функцій двух великих простих чисел, понад 100 розрядів. Через це при спробі відновлення відкритого тексту з шифротексту і відкритому ключу дорівнює розкладу на множники двух таких чисел.

Ключ генерується з двох випадкових великих чисел p и q однакової довжини та розраховується вираз (1.3.1.1):

$$n = p\ q \qquad (1.3.1.1)$$

Потім обирається випадковий ключ e , такий при якому e і $(p\text{-}1)(q\text{-}1)$ будуть взаємно простими числами. Далі за допомогою розширеного алгоритму Евкліда розраховується закритий ключ d, такий при якому $ed = 1\ (\mathrm{mod}(p\text{-}1)(q\text{-}1))$.

Для шифрування повідомлення *m*, воно спочатку розбивається на цифрові блоки, менші *n* (для двійкових даних вибирається найбільша ступінь числа 2, менша *n*). Тобто, якщо *p* и *q* - 100-розрядні прості числа, то *n* міститиме близько 200 розрядів, і кожен блок повідомлення m_i має бути близько 200 розрядів у довжину. Зашифроване повідомлення *c* буде складатися з блоків c_i тієї ж самої довжини. Формула шифру виглядає як (1.3.1.2):

$$c_i = m_i^e \bmod n \qquad (1.3.1.2)$$

Для розшифровки повідомлення візьміть кожен зашифрований блок c_i і обчисліть (1.3.1.3)

$$m_i = c_i^d \bmod n \qquad (1.3.1.3)$$

Оскільки (1.3.1.4)

$$c_i^d = (m_i^e)^d = m_i^{ed} = m_i^{k(p-1)(q-1)+1} = m_i m_i^{k(p-1)(q-1)} = m_i * 1 = m_i \qquad (1.3.1.4)$$

В якому все (mod *n*), формула відновлює повідомлення.

Прийнято вважати, що безпека RSA залежить від проблеми розкладу на множники великих чисел, але технічно це не є безпечним, особливо якщо зміняться методи криптоаналізу.

RSA є світовим стандартом, слугуючи доповненням до ISO 9796 та використовується в якості банківського стандарту в ряді країн.

1.4 Гібридне шифрування

У кожного з різновидів шифрування будь то симетричний або асиметричний є свої недоліки, в реальному світі підвищена безпека асиметричних алгоритмів не означає, що даний алгоритм може замінити симетричні алгоритми. Велику роль грає швидкість обробки великих обсягів інформації, так симетричні алгоритми справляються з ними набагато швидше, приблизно в тисячу разів, але не дають такого надійного захисту як асиметричні.

Тому зародилася ідея об’єднання цих алгоритмів, в якій самі дані шифруються за допомогою симетричних алгоритмів з використанням закритого ключа, а потім цей ключ шифрується асиметричним алгоритмом. Такий спосіб вирішує багато проблем з безпекою та додає завдань криптоаналізу на роки вперед.

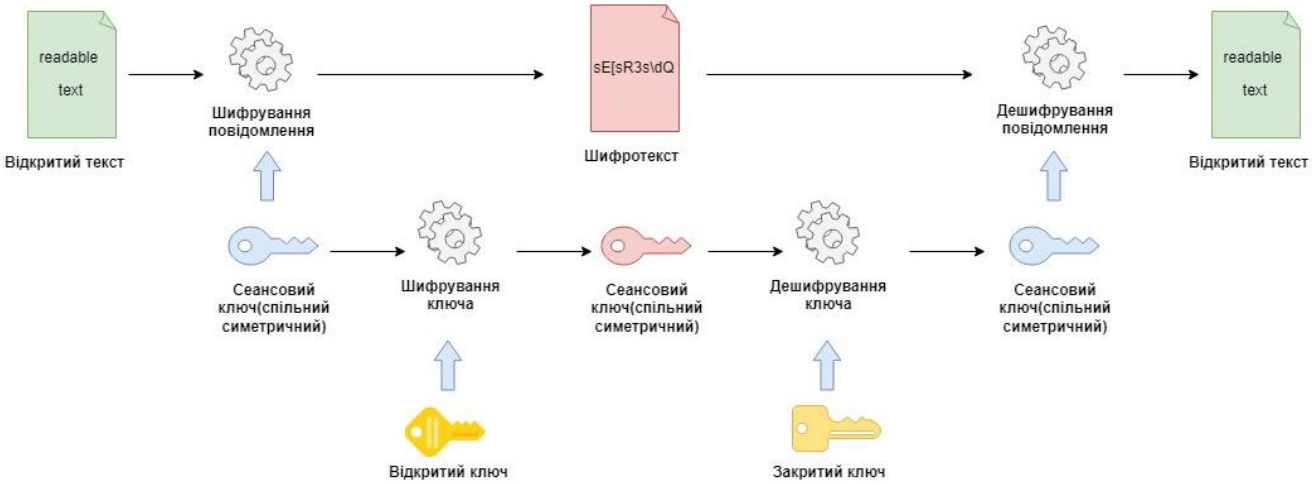


Рисунок 1.4.1 – Гібридне шифрування

2 ВИМОГИ ТА ОЦІНКА ЯКОСТІ СИСТЕМИ

2.1 Загальний опис

Система представляє собою програмне забезпечення(ПЗ), що дає можливість будь-якому користувачу керувати доступом до локальних файлів персонального комп’ютера шляхом їх шифрування з можливістю зворотного отримання доступу за допомогою дешифрування.

2.2 Функціональні вимоги

Система повинна:

- Здійснювати шифрування обраного файлу одним з обраних способів.
- Здійснювати дешифрування зашифрованих файлів

2.2.1 Вимоги до безпеки

Безпека забезпечується шляхом тестування на вразливості системи, підстановкою та пошуком вразливостей та їх обробкою на всіх етапах розробки ПЗ.

До методів виявлення вразливостей безпеки програмного продукту можна віднести:

1. Огляд «білої скриньки», тобто коду, де інженер аналізує початковий код на ймовірні вразливості безпеки, перевіряє структуру окремих елементів програми на коректність та запобігає виявленню проблем.
2. Огляд «чорної скриньки», перевіряється функціональна частина програми без можливості перевірки внутрішньої структури, тобто введенням вхідних значень в роботу програми та вирахуванням вихідних значень.
3. Використання спеціалізованих інструментів (сканерів, антивірусів, фаєрволів і т.п.) для перевірки на наявність дефектів в системі.

До основних видів загроз для програм відносять:

Вхід (валідація) — перевантаження запитів на сервер.

Вторгнення програми — спеціалізований софт для модифікування поведінки роботи програми з метою отримання доступу до закритого від користувача функціоналу.

Аутентифікація — атака шляхом підбору, або викрадення облікових даних.

Авторизація — використання та розкриття конфіденціальної інформації.

Управління конфігурацією — отримання доступу до налаштувань процесів та сервісів.

Чутлива інформація — доступ до незахищеного коду або даних.

Управління сеансом —відтворення сеансу роботи користувачів.

Криптографія — помилка генерації ключів або погане управління ключами.

Маніпуляція параметрами — маніпуляції з файлами cookie та гіпертекстом (HTTP — Hyper Text Transfer Protocol).

Управління винятками — розсекречення інформації, відмова в обслуговуванні (Deny of Service — DoS).

Аудит і вхід у систему — блокування користувачем виконання операції.

2.2.2 Опис архітектури додатку

Програмний продукт має внутрішню структуру, утворену взаємопов'язаними програмними модулями. Структуризація програми виконується насамперед для зручності розробки, програмування, налагодження та внесення змін до програмного продукту.

Розрізняють такі види модулів:

- головний модуль - управляє запуском програмного продукту (існує в однині);
- керуючий модуль – забезпечує виклик інших модулів на обробку;
- робочі модулі – виконують функції обробки;

– сервісні модулі та бібліотеки, утиліти - здійснюють обслуговуючі функції.

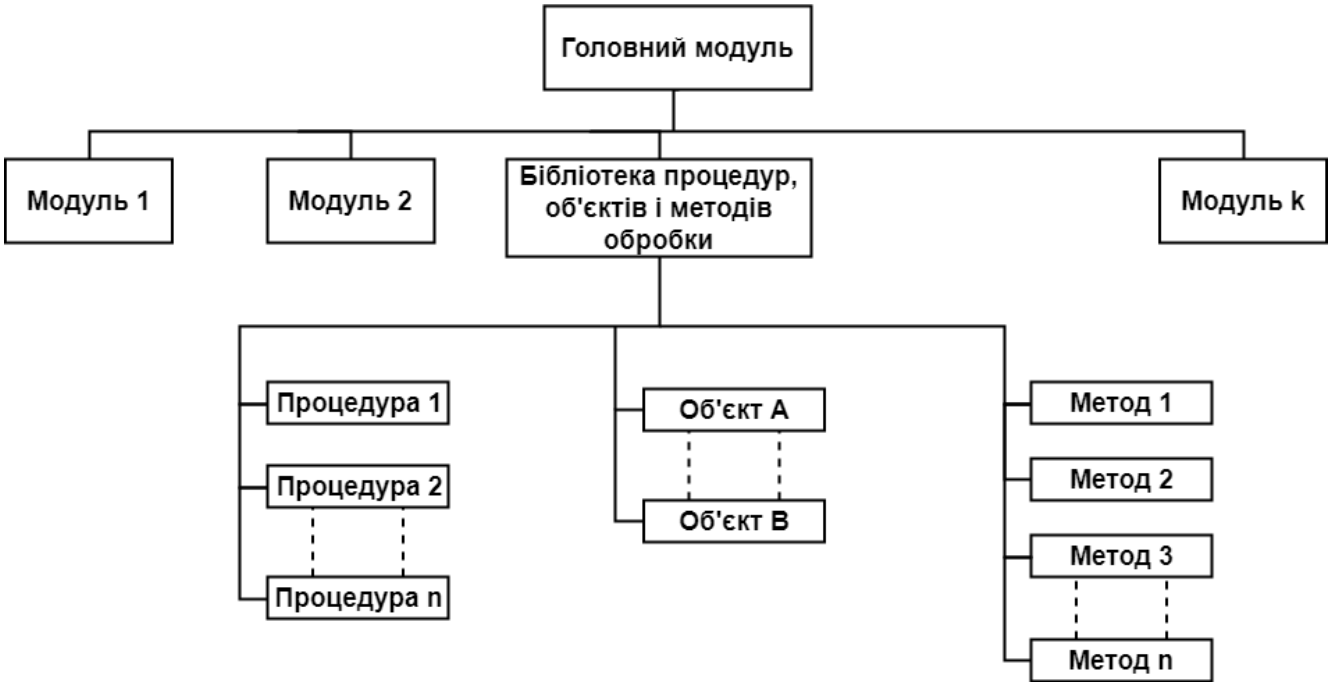


Рисунок 2.2.2.1. – Структура програмного продукту

У роботі програмного продукту активізуються потрібні програмні модулі. Керуючі модулі задають послідовність виклику виконання чергової модуля.

Кожен модуль може оформлятися як файл, що самостійно зберігається; Для функціонування програмного продукту потрібна наявність програмних модулів у складі.

Проектована система представляється як безлічі сутностей чи акторів, взаємодіючих із системою за допомогою прецедентів. При цьому актором (actor) або дійовою особою називається будь-яка сутність, що взаємодіє із системою ззовні. Іншими словами, кожен варіант використання визначає певний набір дій, скоєний системою при діалозі з актором. Діаграма прецедентів не показує те, яким чином буде реалізована взаємодія акторів із системою.

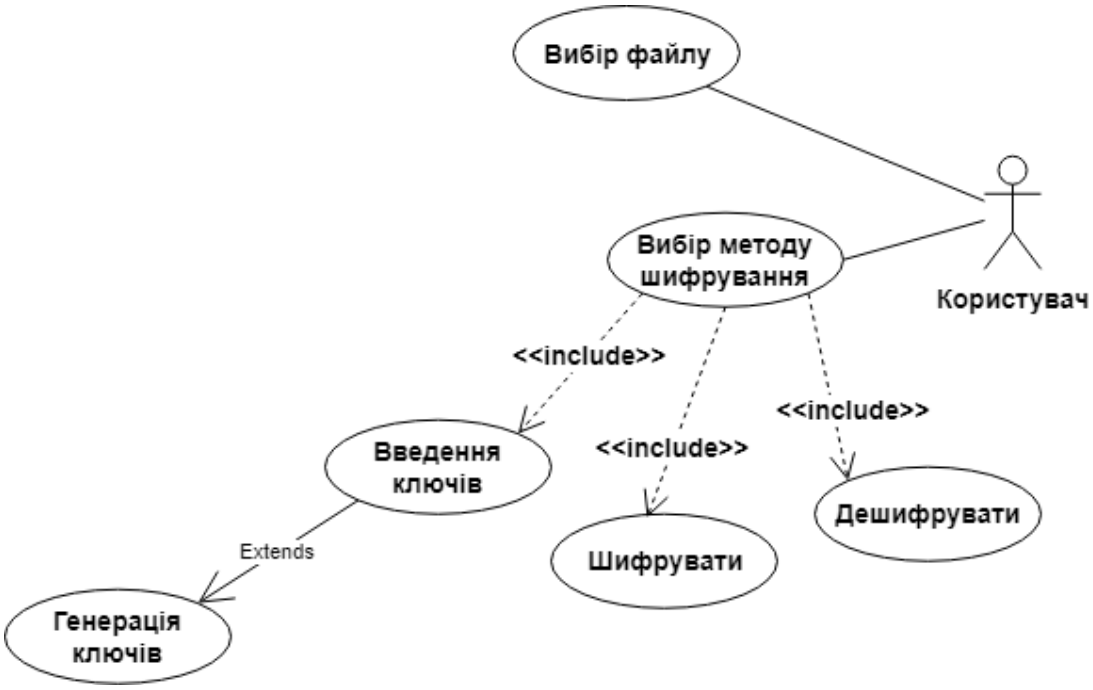


Рисунок 2.2.2.2. – Діаграма прецедентів (Use Case Diagram)

Вербальні специфікації прецедентів

1. **Короткий опис**

   Користувач здійснює шифрування або дешифрування файлу

2. **Суб'єкт** – Користувач

3. **Передумова**

   Обрати файл, метод шифрування та ввести ключі

4. **Основний потік**

   4.1 Після введення та заповнення всіх необхідних полів було натиснуто кнопку шифрування, відбувається шифрування обраного файлу;

   4.1.1 Якщо було допущено помилку у введені даних то виконується A1;

   4.2 Після натиснутої кнопку дешифрування, відбувається дешифрування раніше зашифрованого файлу;

   4.2.1 Якщо було допущено помилку у введені даних то виконується A1;

5. **Альтернативні потоки**

A1 Введені данні були не вірні. Якщо Користувач не вірно ввів дані, система виводить повідомлення про помилку на екран та надає можливість ввести їх повторно. Прецедент продовжується.

6. **Постумови**

   Файл оновлюється в залежності від обраного потоку.

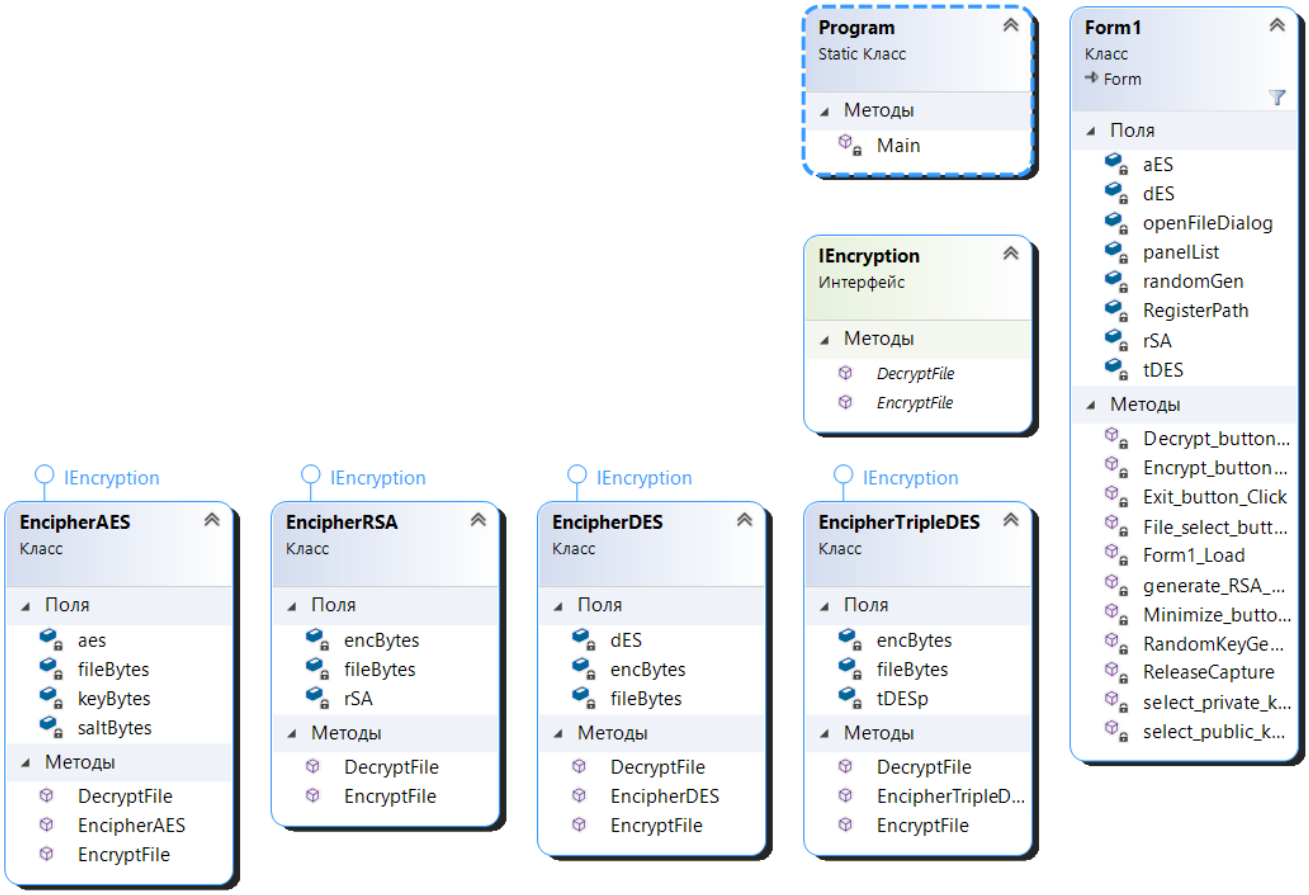


Рисунок 2.2.2.3. – Діаграма класів (Class Diagram)

Діаграма класів служить для представлення статичної структури моделі системи у термінології класів об’єктно-орієнтированого програмування. Діаграма класів може відображати різні взаємозв'язки між окремими сутностями предметної області, такими як об'єкти та підсистеми, а також описує їхню внутрішню структуру (поля, методи) та типи відносин (успадкування, реалізація інтерфейсів).

На даній діаграмі не вказується інформація про тимчасові аспекти функціонування системи. З цього погляду діаграма класів є подальшим розвитком концептуальної моделі проектованої системи.

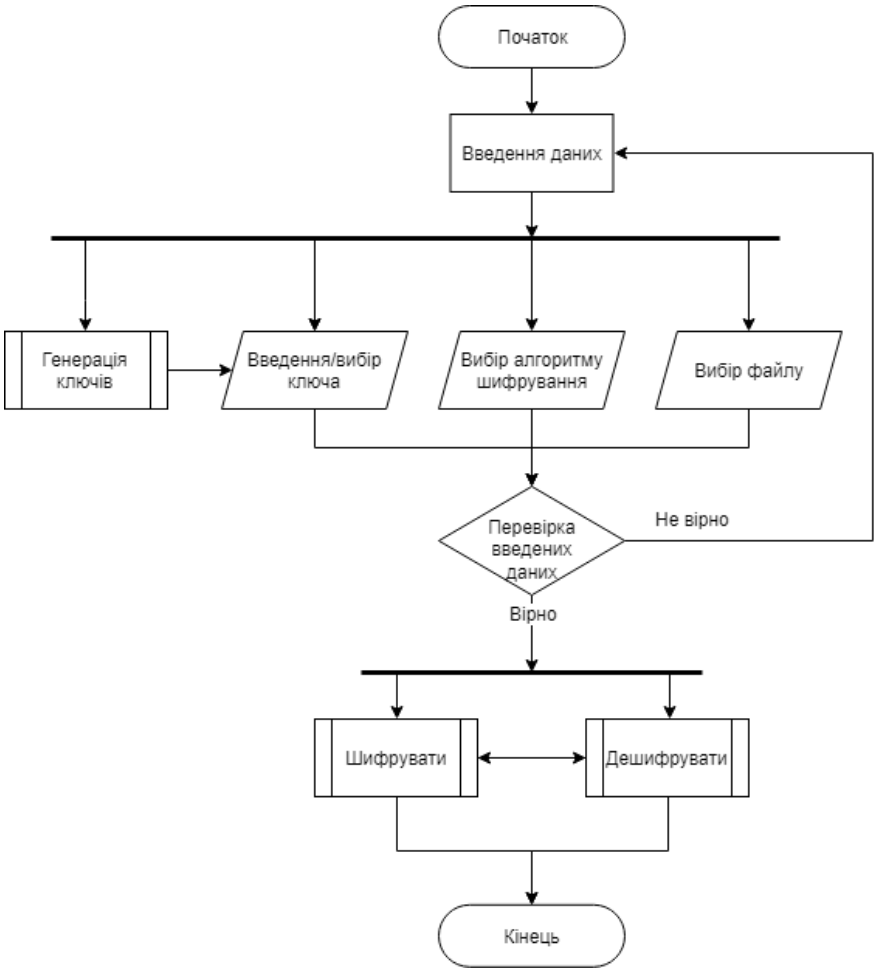


Рисунок 2.2.2.4. – Алгоритм роботи програми

Алгоритм роботи програми показує схему дій, сукупність послідовних кроків, що визначає процес переходу від первинних даних до бажаного результату.

2.2.3 Операційне середовище

Програма реалізована мовою програмування C# в середовищі розробки Microsoft Visual Studio 2019. Функціонування програми здійснюватиметься під керівництвом операційних системи сімейства Windows: MS Windows 7, MS Windows 8, MS Windows 10 та інших з підтримкою .NET Framework версії 4.7.2.

2.3 Нефункціональні вимоги

2.3.1 Інтерфейси користувача

UI (User Interface) користувача представлений у вигляді WIMP(Window - вікно, Image - образ, Menu - меню, Pointer - показчик) інтерфейсу, де діалог з користувачем виконується за допомогою графічних образів та інших елементів, цей інтерфейс реалізований з включенням двох технологій чистого WIMP та GUI (Graphical user interface, графічний інтерфейс користувача), в якому команди подаються замасковано.

2.3.2 Програмні інтерфейси

Можна розрізняти два види програмного інтерфейсу, той що використовується при створенні прикладних програм та той, що використовується при створенні системних компонентів.

Перший зазвичай називається інтерфейсом програмування програм API (Application Programming Interface), друга називається інтерфейсом програмування компонентів операційної системи чи інтерфейсом системного програмування SPI (System Programming Interface). Крім того, програмні інтерфейси можуть бути різнорівневими, ставлячись або до рівня програмного коду (API та SPI), або до рівня машинного коду.

Програмний інтерфейс для програмного продукту створений за допомогою графічного дизайнера у Visual Studio, що приймає вхідні дані від користувача, такі як ключ або вибір шляху розташування файлу та кнопки взаємодії.

Також використано інтерфейс для встановлення програми за допомогою інсталятора.

2.3.3 Атрибути програмного продукту

1. Надійність – властивість, що дозволяє зберігати значення всіх параметрів протягом встановленого часу, які характеризують здатність виконувати

потрібні функції в заданих режимах та умовах застосування, технічного обслуговування, зберігання та транспортування.

2. Доступність – властивість, що надає користувачу який володіє відповідними повноваженнями, можливість використовувати ресурс відповідно до правил, встановлених політикою безпеки.

3. Безпека – забезпечує конфіденційність, доступність і цілісність інформації від несанкціонованого доступу, використання, оприлюднення, ознайомлення.

4. Супроводжуваність – процес покращення, оптимізації та виправлення дефектів у програмному забезпеченні після його вводу до експлуатації. Цей процес стандартизовано організацією ISO — ISO/IEC 14764.

5. Переносимість – показники, що показують здатність ПЗ адаптуватися до роботи в нових умовах, тобто при перенесенні чи встановленні на іншу систему в залежності від апаратних чи програмних можливостей.

6. Продуктивність – здатність виконувати свої функції при заданих обсягах навантаження чи об’єму даних.

3 ОПИС ПРОЕКТУВАННЯ ПРОГРАМИ

Програмний продукт представляє собою невелику програму-утиліту, що здатна шифрувати та дешифрувати файли розташовані на локальних зберігачах пам’яті комп’ютера

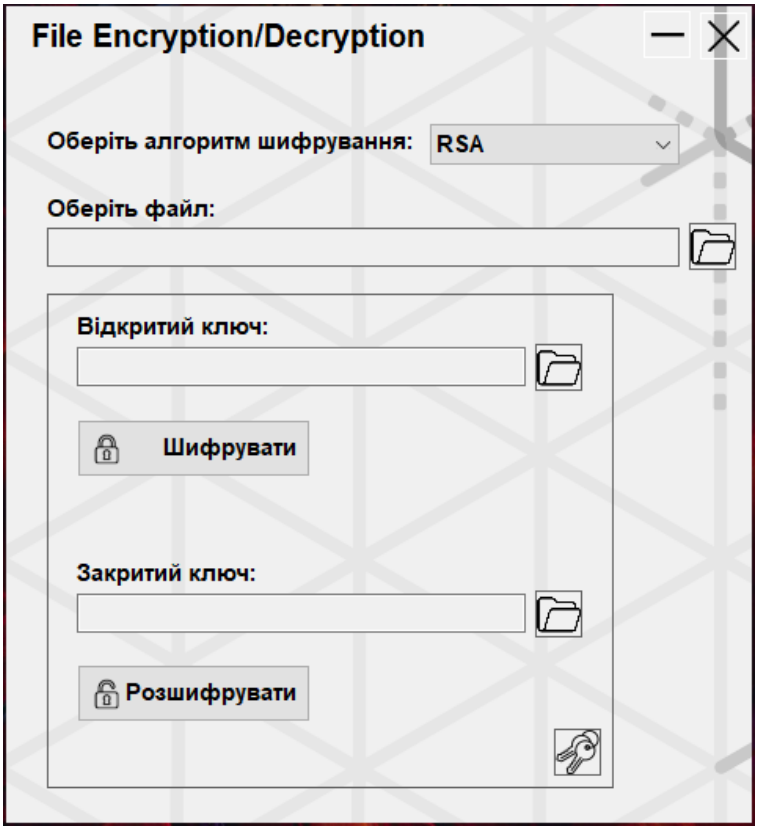


Рисунок 3.1.– Загальний вигляд програмного продукту

Програма може здійснювати шифрування одним з 4х способів на вибір

Рисунок 3.2.– Список доступних алгоритмів

Кожен з алгоритмів, окрім вибору самого алгоритму містить обов’язкові для заповнення поля:

- Шлях до обраного файлу
- Поле для ключа
- Кнопку генерації ключа
- Кнопки «Шифрувати» та «Розшифрувати»

В залежності від обраного алгоритму змінюється вікно та поля на ньому.

```
List<Panel> panelList = new List<Panel>();
private void Form1_Load(object sender, EventArgs e)
{
    path_textBox.Text = RegisterPath;
    comboBox1.SelectedIndex = 0;
    panelList.Add(DES_panel);
    panelList.Add(TripleDES_panel);
    panelList.Add(AES_panel);
    panelList.Add(RSA_panel);
}
private void comboBox1_SelectionChangeCommitted(object sender, EventArgs e)
{
    for (int i = 0; i < panelList.Count; i++)
        panelList[i].Visible = panelList[i].Name.Equals(comboBox1.Text + "_panel") ? true : false;
}
```

Рисунок 3.3.– Реалізація вибору алгоритму та зміни активного вікна

Створюється список вікон в який додаються всі існуючі вікна. При виборі в списку одного з них активність вікна змінюється на обрану, інші стають не активними.

Розглянемо окремо кожен з методів шифрування. Першим є стандарт шифрування даних DES.

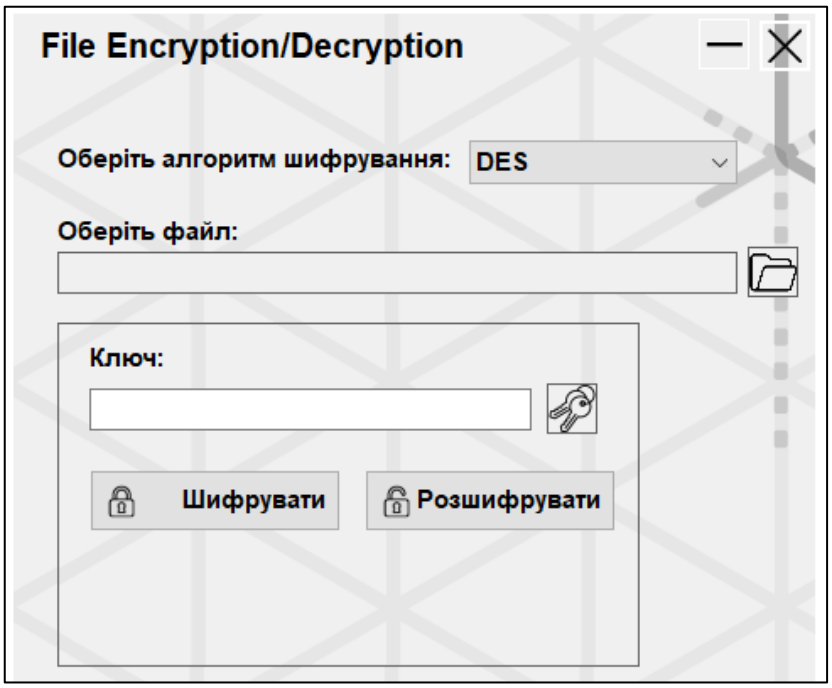


Рисунок 3.4.– Вікно алгоритму DES

Як і на всіх інших на ньому розміщена кнопка, що відкриває провідник для вибору файлу

Рисунок 3.5.– Кнопка вибору файлу за допомогою провідника

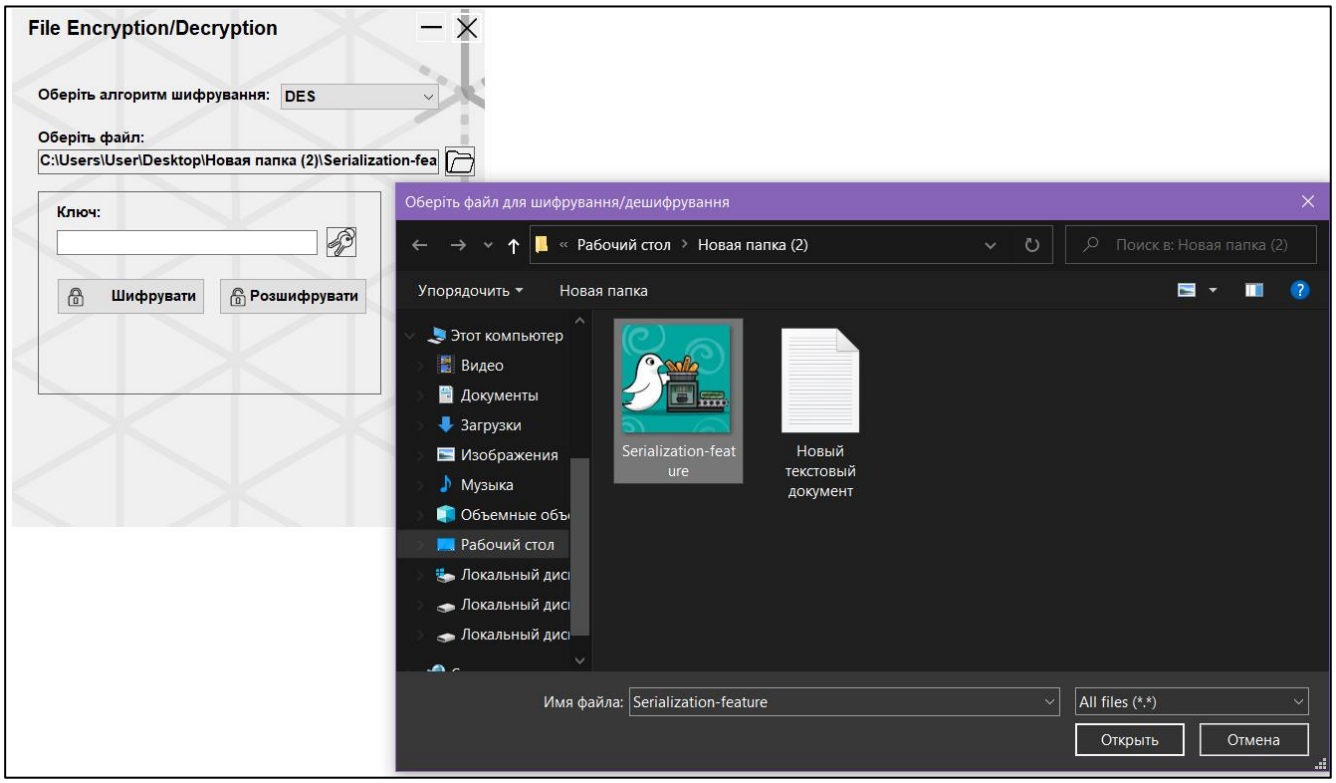


Рисунок 3.6.– Файловий провідник та вибір файлу

```
OpenFileDialog openFileDialog = new OpenFileDialog();

private void File_select_button_Click(object sender, EventArgs e)
{
    openFileDialog.Filter = "All files (*.*)|*.*";
    openFileDialog.Title = "Оберіть файл для шифрування/дешифрування";
    if (openFileDialog.ShowDialog() == DialogResult.OK)
    {
        path_textBox.Text = openFileDialog.FileName;
    }
}
```

Рисунок 3.7.– Реалізація вибору файлу в провіднику

За допомогою створеного об’єкту класу OpenFileDialog та методу відображення діалогу, шлях передається на вікно програми та використовується в подальшому.

Для швидкого доступу до програми, при її встановленні вона додається в контекстне меню Windows. Це підвищує зручність її використання та пришвидшує процес вибору файлу для виконання дій над ним.

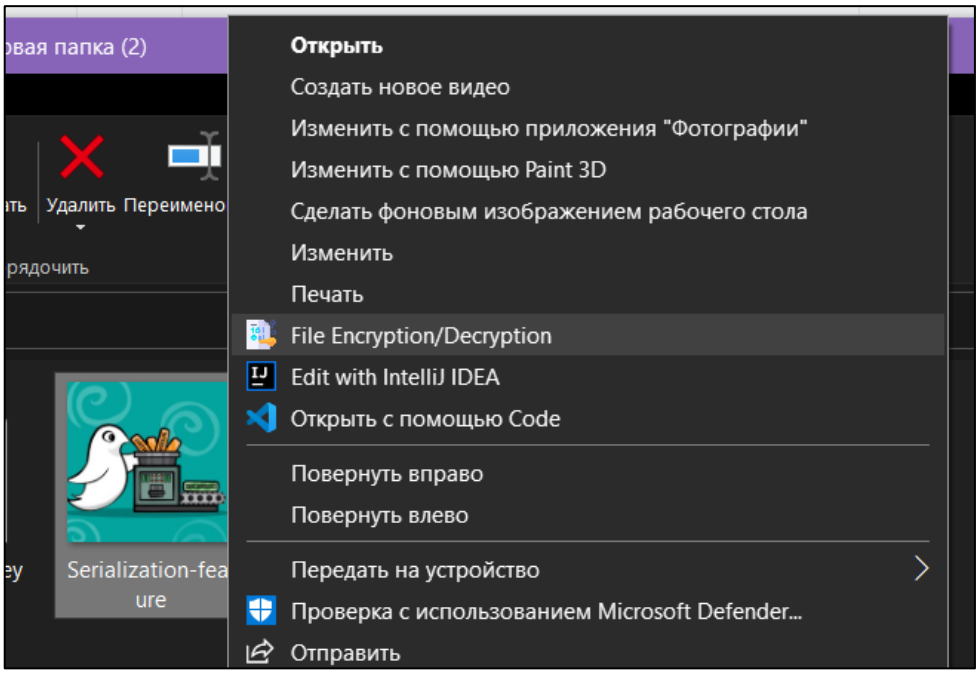


Рисунок 3.8.–Програма в контекстному меню Windows

Для цього створюється клас інсталятору, що дає можливість додати функціонал при встановленні, оновленні чи видаленні програми в інсталяторі.

Через клас Registry в потрібному розділі створюються підрозділи для виведення в меню нашої програми також при завантаженні форми через аргументи на поле шляху передається місцезнаходження обраного файлу.

```
public partial class Installer1 : Installer
{
    const string rootKey = "*\\shell";
    const string keyCommand = "\\command";
    const string openParam = " \"%1\"";
    const string fileName = "\\File Encrypt/Decrypt";

    public Installer1()
    {
        InitializeComponent();
    }

    public override void Commit(IDictionary stateSaver)
    {
        base.Commit(stateSaver);
        string pathStr = Context.Parameters["assemblypath"];
        string path = pathStr + openParam;
        Registry.ClassesRoot.CreateSubKey(rootKey + fileName).SetValue("", "File Encryption/Decryption");
        Registry.ClassesRoot.CreateSubKey(rootKey + fileName).SetValue("Icon", pathStr);
        Registry.ClassesRoot.CreateSubKey(rootKey + fileName + keyCommand).SetValue("", path);
    }
    public override void Uninstall(IDictionary stateSaver)
    {
        base.Uninstall(stateSaver);
        Registry.ClassesRoot.DeleteSubKeyTree(rootKey + fileName);
    }
}
```

Рисунок 3.9.– Додавання в контекстне меню

Наступним обов’язковим полем для заповнення є ключ. Його можна ввести вручну або згенерувати за допомогою відповідної кнопки. Для різних алгоритмів довжина ключа відрізняється.

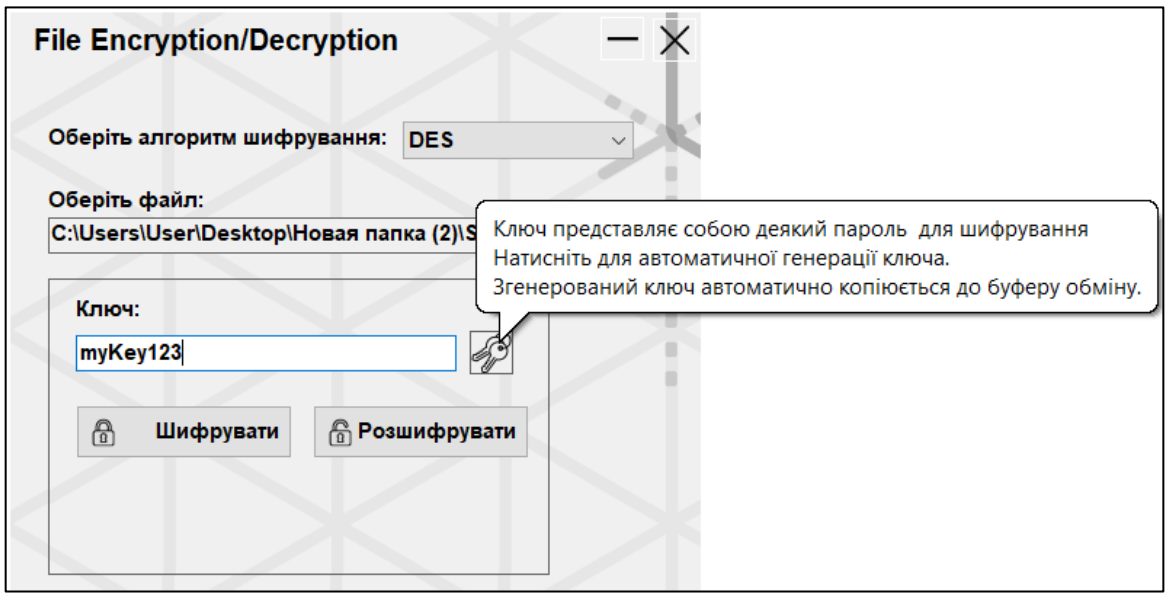


Рисунок 3.10.– Введення ключа та його генерація

```
RandomGen randomGen = new RandomGen(Option.IncludeCapital);
private void RandomKeyGenarator(object sender, EventArgs e)
{
    foreach (Panel panel in panelList)
    {
        if (panel.Visible == true)
        {
            foreach (Control control in panel.Controls)
            {
                if (control is TextBox)
                {
                    control.Text = randomGen.Gen(((TextBox)control).MaxLength);
                    Clipboard.SetText(((TextBox)control).Text);
                }
            }
        }
    }
}
```

Рисунок 3.11.– Метод генерації ключа

Для генерації ключа в активній панелі перевіряється активність її стану та доступна довжина поля ключа, після чого за допомогою класу RandomGen створюється ключ, що містить випадкові символи та літери.

Якщо всі поля були заповнені вірно та була натиснута кнопка «Шифрувати», то файл буде зашифровано, якщо було допущено помилку, то програма виведе вікно з описом помилки.

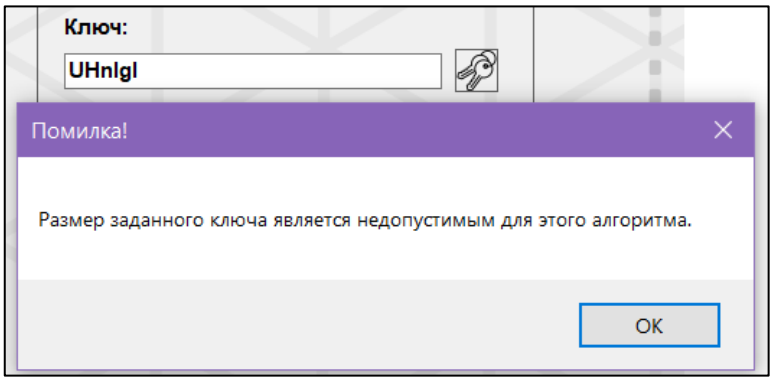


Рисунок 3.12.– Приклад помилки

При успішному шифруванні обраний файл зашифрується та його бітова складова зміниться, що не дасть можливість розібрати текст або відкрити деякі файли.

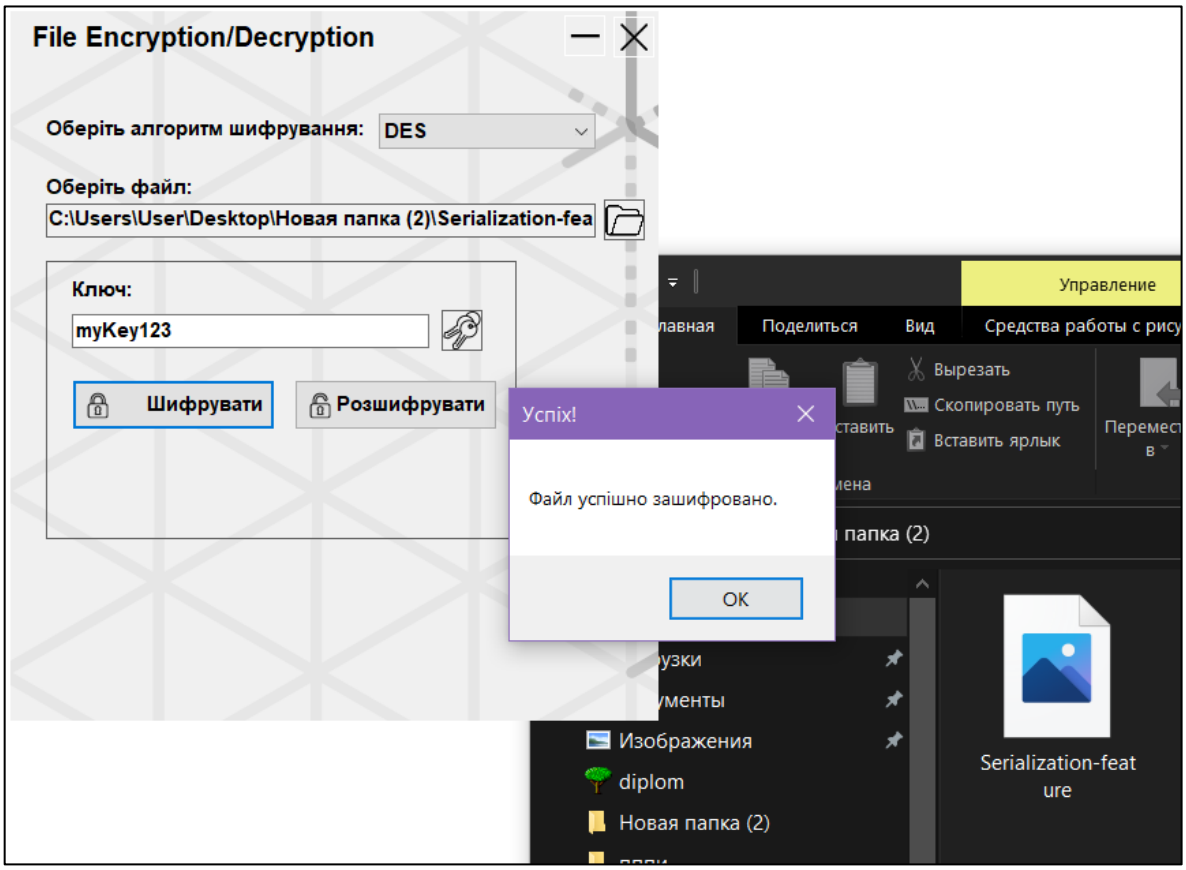


Рисунок 3.13.– Успішне шифрування зображення

При спробі відкрити зашифрований файл система повідомить, що не має можливості його відкрити.

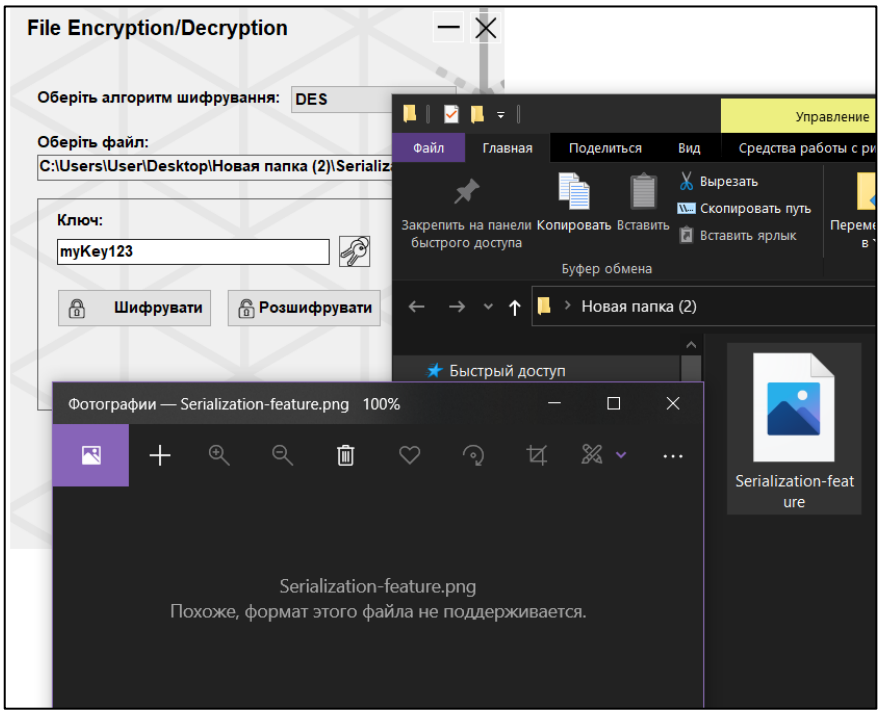


Рисунок 3.14.– Файл після шифрування

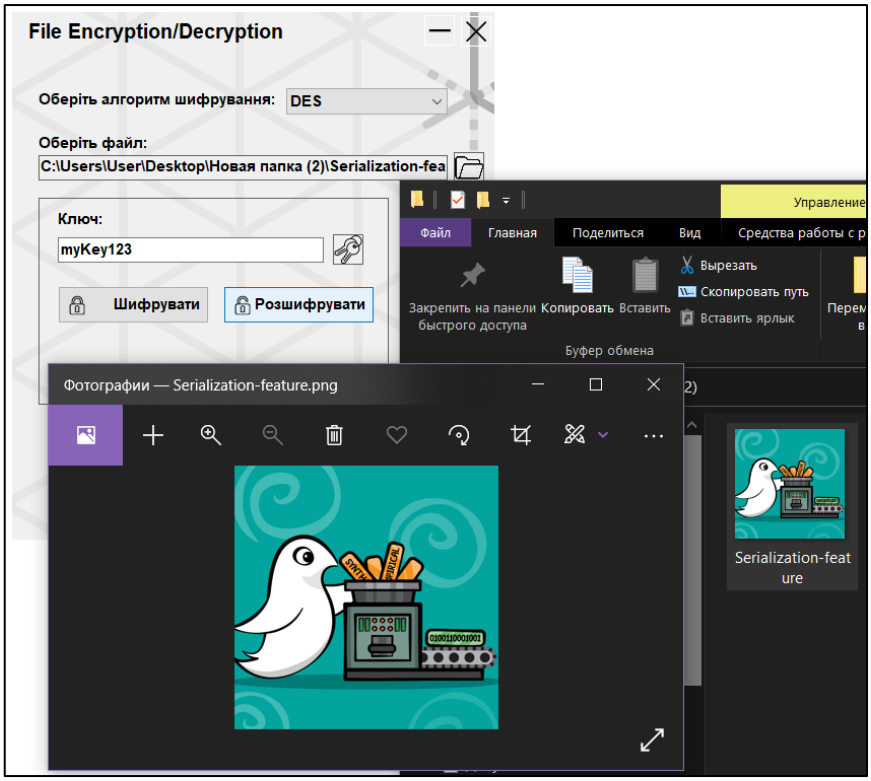


Рисунок 3.15.– Файл після розшифрування

Розшифрування виконується обернено до шифрування, обирається раніше зашифрований файл та натискається кнопка «Розшифрувати», після чого файл повертається до початкового вигляду та вмісту.

```
interface IEncryption
{
    void EncryptFile(string filePath, string publicKeyPath);
    void DecryptFile(string filePath, string publicKeyPath);

}

class EncipherDES : IEncryption
{
     DESCryptoServiceProvider dES = new DESCryptoServiceProvider();
     byte[] fileBytes;
     byte[] encBytes;

     public void EncryptFile(string filePath, string key)
     {
         try
         {
             dES.Key = Encoding.UTF8.GetBytes(key);
             fileBytes = File.ReadAllBytes(filePath);
             encBytes = dES.CreateEncryptor().TransformFinalBlock(fileBytes, 0, fileBytes.Length);
             File.WriteAllBytes(filePath, encBytes);
             MessageBox.Show("Файл успішно зашифровано.", "Успіх!");
         }
         catch (Exception ex)
         {
             MessageBox.Show(ex.Message, "Помилка!");
             return;
         }
     }
     public void DecryptFile(string filePath, string key)
     {
         try
         {
             dES.Key = Encoding.UTF8.GetBytes(key);
             fileBytes = File.ReadAllBytes(filePath);
             encBytes = dES.CreateDecryptor().TransformFinalBlock(fileBytes, 0, fileBytes.Length);
             File.WriteAllBytes(filePath, encBytes);
             MessageBox.Show("Файл успішно розшифровано.", "Успіх!");
         }
         catch (Exception ex)
         {
             MessageBox.Show(ex.Message, "Помилка!");
             return;
         }
     }
```

Рисунок 3.16.– Методи шифрування та дешифрування DES

Методи шифрування та дешифрування визначені в інтерфейсі та реалізовані в класі. Змінюється кодування ключа, перетворюючи його в послідовність байтів. Файл зчитується побайтно, створюється симетричний об’єкт-шифратор з використанням ключа та вектор ініціалізації, що трансформує задану область масиву байтів файлу, далі файл перезаписується на місце старого вже з

трансформованими байтами. Дешифрування відрізняється лише тим, що створюється об'єкт-дешифратор та виконуються ті самі дії.

Що стосується алгоритму шифрування Triple DES, то воно повторює реалізацію звичайного DES, модифікуючи його та додає одну відмінність, в ньому довжина ключа містить 16 символів на відміну від звичайного DES, що підвищує рівень його захисту.

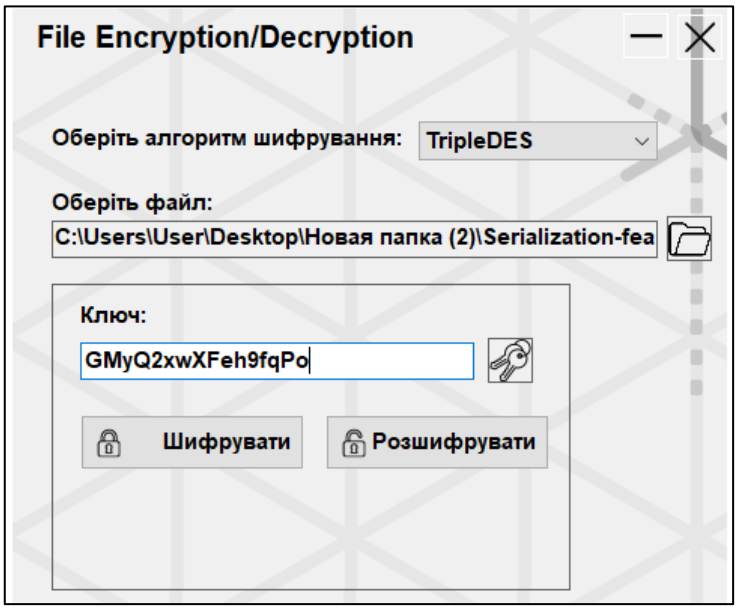


Рисунок 3.17.– Розмір ключа в алгоритмі Triple DES

Наступним є алгоритм AES. Містить схожі елементи управління, за відміною збільшеного поля для ключа, так як ключ в AES може досягати розміру в 256 біт.

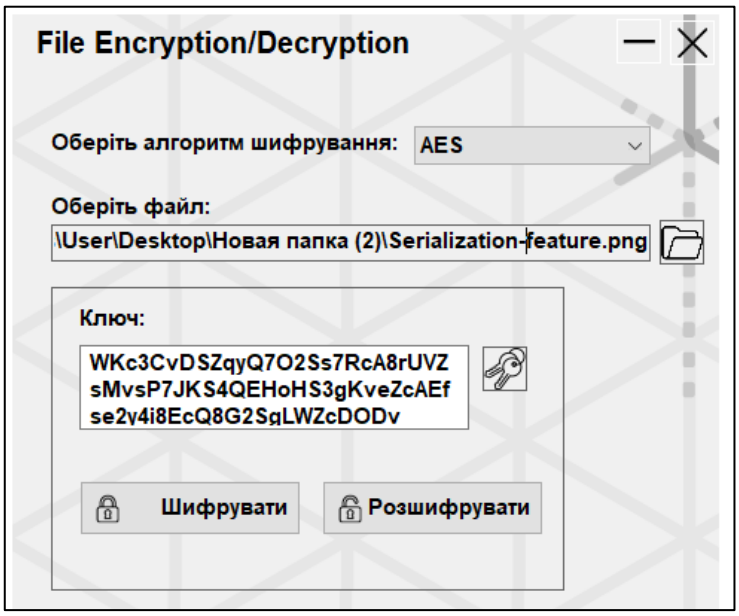


Рисунок 3.18.– Розмір ключа в алгоритмі AES

```
class EncipherAES : IEncryption
{
    RijndaelManaged aes = new RijndaelManaged();
    byte[] keyBytes;
    byte[] saltBytes = { 1, 2, 3, 4, 5, 6, 7, 8 };
    byte[] fileBytes;

    public void EncryptFile(string filePath, string strKey)
    {
        try
        {
            keyBytes = Encoding.UTF8.GetBytes(strKey);
            using (MemoryStream memory = new MemoryStream())
            {
                Rfc2898DeriveBytes key = new Rfc2898DeriveBytes(keyBytes, saltBytes, 2000);
                aes.Key = key.GetBytes(aes.KeySize / 8);
                aes.IV = key.GetBytes(aes.BlockSize / 8);
                using (CryptoStream cryptoStream = new CryptoStream(memory, aes.CreateEncryptor(), CryptoStreamMode.Write))
                {
                    fileBytes = File.ReadAllBytes(filePath);
                    cryptoStream.Write(fileBytes, 0, fileBytes.Length);
                    cryptoStream.FlushFinalBlock();
                }
                File.WriteAllBytes(filePath, memory.ToArray());
                MessageBox.Show("Файл успішно зашифровано.", "Успіх!");
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message, "Помилка!");
        }
    }

    public void DecryptFile(string filePath, string strKey)
    {
        try
        {
            keyBytes = Encoding.UTF8.GetBytes(strKey);
            using (MemoryStream memory = new MemoryStream())
            {
                Rfc2898DeriveBytes key = new Rfc2898DeriveBytes(keyBytes, saltBytes, 2000);
                aes.Key = key.GetBytes(aes.KeySize / 8);
                aes.IV = key.GetBytes(aes.BlockSize / 8);
                using (CryptoStream cryptoStream = new CryptoStream(memory, aes.CreateDecryptor(), CryptoStreamMode.Write))
                {
                    fileBytes = File.ReadAllBytes(filePath);
                    cryptoStream.Write(fileBytes, 0, fileBytes.Length);
                    cryptoStream.FlushFinalBlock();
                }
                File.WriteAllBytes(filePath, memory.ToArray());
                MessageBox.Show("Файл успішно розшифровано.", "Успіх!");
            }
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message, "Помилка!");
        }
    }
```

Рисунок 3.19.– Методи шифрування та дешифрування AES

Методи шифрування та дешифрування засновані на алгоритмі Rijndael також визначені в інтерфейсі та реалізовані в класі. Як і раніше змінюється кодування ключа, перетворюючи його в послідовність байтів, далі створюється потік в пам’яті з можливістю розширення, за допомогою класу Rfc2898DeriveBytes формується ключ на основі паролю, значення розширення(солі) та кількість ітерацій для формування ключа.

Налаштовуються розміри ключа та блоку криптографічних операцій. Створюється потік виконання криптографічного перетворення, використовуючи цільовий потік даних, об’єкт шифратор або дешифратор для перетворення та обирається режим потоку. Далі файл зчитується побайтово та записує послідовність цих байтів у поточний поток переміщуючи поточну позицію всередині потоку число записаних байтів. Оновлює буфер та записує файл з модифікованими даними на заміну старим.

Останнім з методів шифрування є RSA. Ключі в якому реалізовані, як файли, що містять відкритий та закритий ключ, програма читає дані ключі з XML файлів.

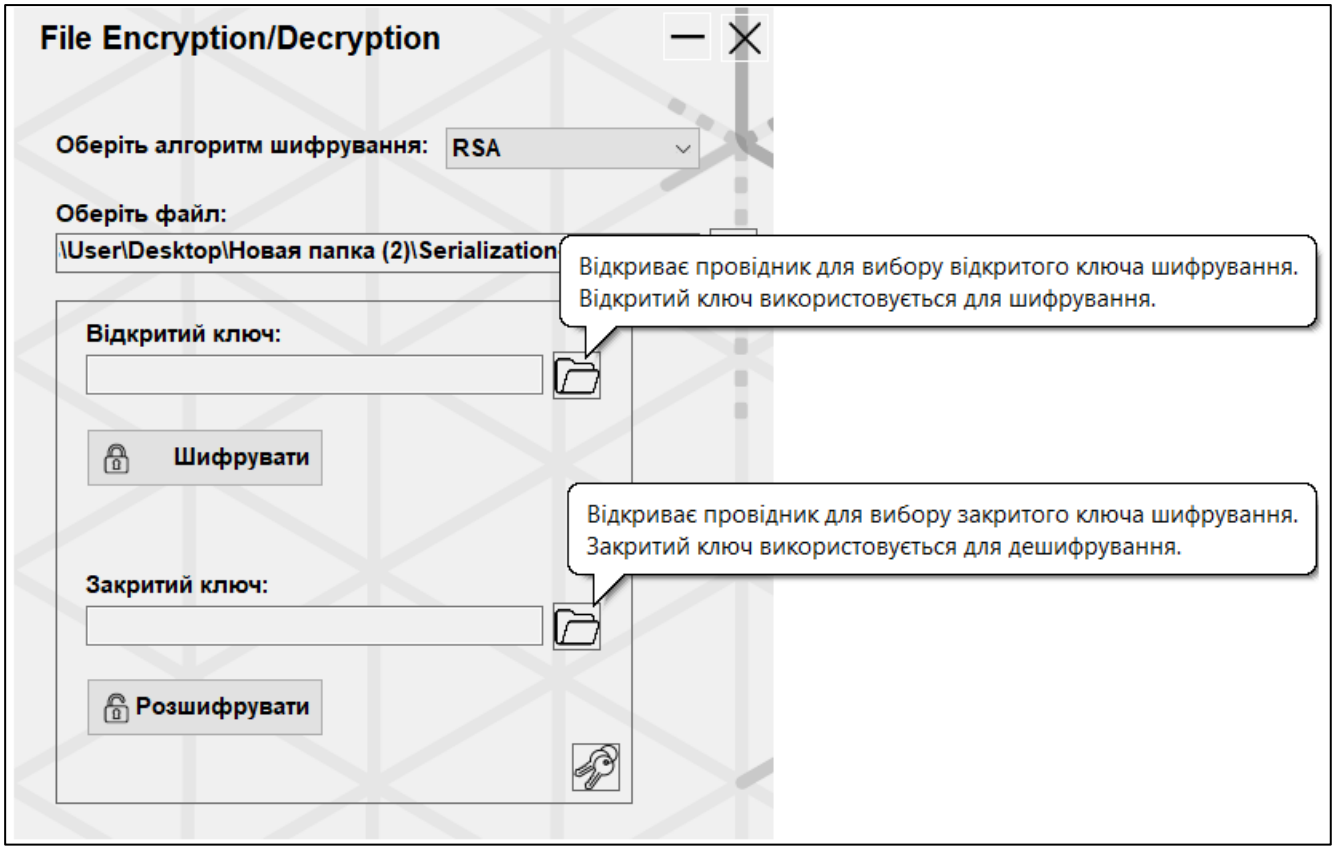


Рисунок 3.20.– Вікно алгоритму RSA

```
private void select_public_key_button_Click(object sender, EventArgs e)
{
    openFileDialog.Filter = "XML - File | *.xml";
    openFileDialog.Title = "Оберіть відкритий RSA ключ";
    if (openFileDialog.ShowDialog() == DialogResult.OK)
    {
        rSA_public_key_textBox.Text = openFileDialog.FileName;
    }
}

private void select_private_key_button_Click(object sender, EventArgs e)
{
    openFileDialog.Filter = "XML - File | *.xml";
    openFileDialog.Title = "Оберіть закритий RSA ключ";
    if (openFileDialog.ShowDialog() == DialogResult.OK)
    {
        rSA_private_key_textBox.Text = openFileDialog.FileName;
    }
}
```

Рисунок 3.21.– Реалізація вибору RSA ключів у провіднику

При генерації ключів через провідник також будуть створені нові файли, що містять ключі для алгоритму.

Де за допомогою методу класу SaveFileDialog відкривається діалог збереження файлу, вказується шлях та ім’я. Після збереження в файл записується згенерований рядок XML, що містить ключ поточного об'єкта.

```
private void generate_RSA_Keys_Click(object sender, EventArgs e)
{
    RSACryptoServiceProvider rSACryptoService = new RSACryptoServiceProvider();
    SaveFileDialog saveFileDialog1 = new SaveFileDialog();
    saveFileDialog1.Title = "Збережіть RSA ключ";
    saveFileDialog1.FileName = "RSA Public Key";
    saveFileDialog1.Filter = "XML - File | *.xml";
    if (saveFileDialog1.ShowDialog() == DialogResult.OK)
    {
        File.WriteAllText(saveFileDialog1.FileName, rSACryptoService.ToXmlString(false));
    }
    saveFileDialog1.FileName = "RSA Private Key";
    if (saveFileDialog1.ShowDialog() == DialogResult.OK)
    {
        File.WriteAllText(saveFileDialog1.FileName, rSACryptoService.ToXmlString(true));
    }
}
```

Рисунок 3.22.– Реалізація генерації RSA ключів

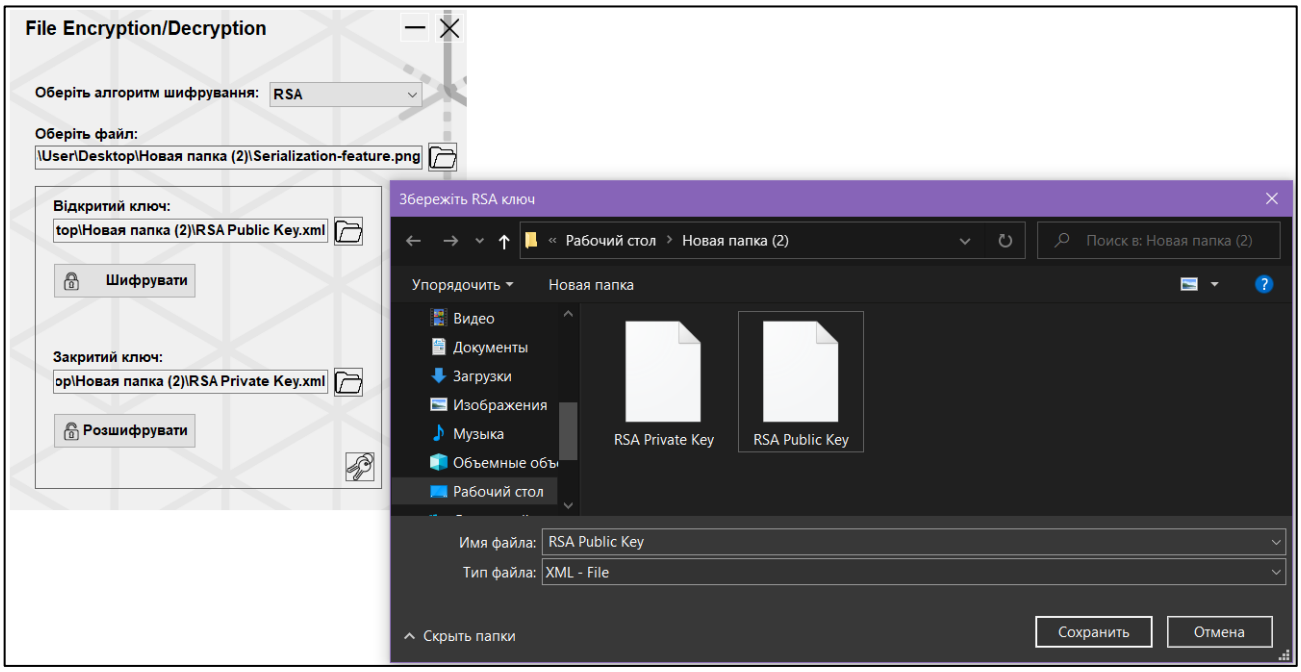


Рисунок 3.23.– Збереження RSA ключів у провіднику

Створені ключі можна використати в будь-який момент, перезаписати або видалити.

```
class EncipherRSA : IEncryption
{
    RSACryptoServiceProvider rSA = new RSACryptoServiceProvider();
    byte[] fileBytes;
    byte[] encBytes;

    public void EncryptFile(string filePath, string publicKeyPath)
    {
        try
        {
            rSA.FromXmlString(File.ReadAllText(publicKeyPath));
            fileBytes = File.ReadAllBytes(filePath);
            encBytes = rSA.Encrypt(fileBytes, false);
            File.WriteAllBytes(filePath, encBytes);
            MessageBox.Show("Файл успішно зашифровано.", "Успіх!");
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message, "Помилка!");
        }
    }
    public void DecryptFile(string filePath, string privateKeyPath)
    {
        try
        {
            rSA.FromXmlString(File.ReadAllText(privateKeyPath));
            fileBytes = File.ReadAllBytes(filePath);
            encBytes = rSA.Decrypt(fileBytes, false);
            File.WriteAllBytes(filePath, encBytes);
            MessageBox.Show("Файл успішно розшифровано.", "Успіх!");
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message, "Помилка!");
        }
    }
```

Рисунок 3.24.– Методи шифрування та дешифрування RSA

Методи шифрування та дешифрування засновані на алгоритмі також визначені в інтерфейсі та реалізовані в класі. Ключ зчитується з раніше згенерованого файлу. Файл так само зчитується побітово, створюються об’єкти шифратор та дешифратор та модифікований файл перезаписується.

Слід відмітити , що при шифруванні RSA існує обмеження на розмір даних, що підлягають шифруванню, і максимальний розмір не перевищує довжину ключа. Наприклад, при використанні 1024-бітного ключа може зашифрувати дані розміром до 1024/8 = 128 байт.

4 ОПИС ВИКОРИСТАНИХ ТЕХНОЛОГІЙ

4.1 Платформи, які використовуються для розробки

4.1.1 Visual Studio IDE

Рисунок 4.1.1.1.– Логотип Visual Studio

Інтегроване середовище розробки (integrated development environment, IDE) Visual Studio представляє собою програму, що містить у собі велику кількість функціональних інструментів для розробки ПЗ. Як і більшість IDE, вона може бути використана для редагування, налагоджування та створення програмного коду з подальшою його публікацією у вигляді готового продукту. Окрім базового набору функцій Visual Studio містить вбудований компілятор, інструменти завершення коду, графічні дизайнери та інші функції для зручності при розробці ПЗ.

Visual Studio поширюється для систем на Windows і Mac та оптимізована для кросплатформних і мобільних додатків.

Існує три версії Visual Studio: Community, Professional і Enterprise, що деяким чином відрізняються одна від одної за функціоналом, якщо коротко описати кожну то:

Community – є безкоштовною, але й одночасно повністю функціональною IDE для використання окремо взятими розробниками або для навчання та розробки відкритого ПЗ.

Professional – надає професійні інструменти та служби для розробки та підтримує одночасну розробку невеликими групами розробників.

Enterprise – необмежена ніякими рамками щодо розміру команд розробників чи функціоналу, задовольняючи максимальні вимоги якості.

Серед цікавих функцій даної IDE можна відмітити наступні:

– Хвилясті підкреслення, що здатні візуально повідомити про помилки або потенційні проблеми під час написання коду, якщо навести на них то відобразиться інформація про помилку та меню швидких дій де показані варіанти її вирішення.

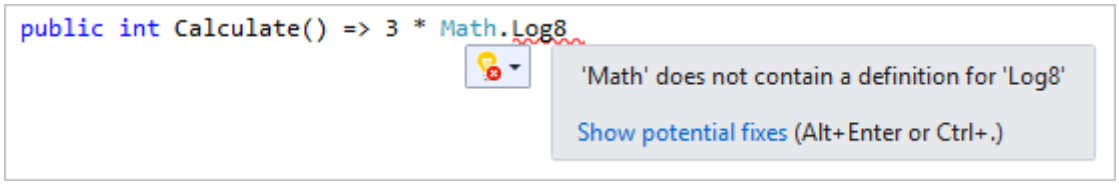


Рисунок 4.1.1.2.– Візуальне представлення помилки в Visual Studio

– Автоматичне очищення та форматування коду, для вибору та застосування зі списку запропонованих стилів та аналізаторів. Очищення доступне лише для мови C#, допомагаючи вирішенню проблеми ще до перевірки коду.

– Швидкий рефакторинг всередині проекту, дає можливість перейменовувати змінні у будь якому місці в проекті, вилучати рядки в методи і заміняти порядок параметрів в методах.

– IntelliSense – цілий набір функцій для відображення інформації стосовно коду в самому редакторі, що відкриває частину документації з описом виділеного фрагменту структури коду. Також здатний запропоновувати та підказувати деякі фрагменти коду опираючись на дії користувача. Функції IntelliSense змінюються в залежності від мови програмування.

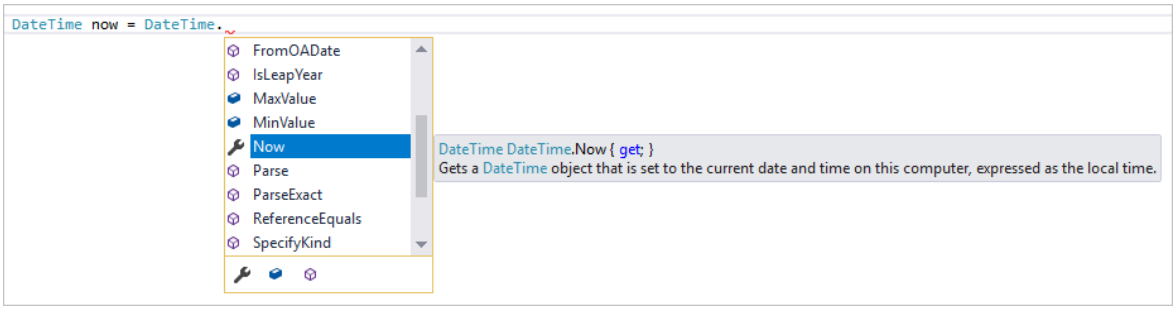


Рисунок 4.1.1.3.– Представлення функції IntelliSense

– Весь набір меню, параметрів чи властивостей може здаватися чималим, тому пошук допомагає в знаходженні будь-якої частини коду, функції, тощо.

– Live Share – функці, що надає можливість поширювати свій проект, спільно налаштовувати та редагувати його з іншими людьми в режимі реального часу.

– Ієрархія викликів, показує методи, які викликають обраний метод. Така інформація бути корисною, якщо ви намагаєтесь відслідкувати роботу методу, або коли ви намагаєтесь відстежити помилку.

– CodeLens допомагає знайти посилання на код, зміни коду, пов’язані помилки, робочі елементи, огляди коду та модульні тести, не виходячи з редактора.

– Також функції для перегляду методів без переходу до них Peek Definition та перехід до місця визначення функції або типу та іншого.

4.1.2 SCM Git

Рисунок 4.1.2.1.– Логотипи GIT та GitHub

SCM (Source Code Management) Git – це розподілена система керування версіями ПЗ випущена під загальною публічною ліцензією GNU версії 2.0, яка є ліцензією з відкритим вихідним кодом. Призначена для керування локальною розробкою ПЗ, де кожен з програмістів може працювати над одним проектом, будуючи власті версії та розділяючи їх на копії з повною історією змін з подальшою можливістю злиття цих версій.

Основною відмінністю Git від інших SCM моделей є модель розгалуження. Ця функція створює можливості для експериментів з кодом без впливу на основний проект, для тестування та імплементації нових функцій, де кожна нова функція може мати свою власну гілку. Також ви можете відсилати та об’єднувати обрані гілки між собою або з основною гілкою. В Git існує певна проміжна область, робочий каталог, що містить розміщені вами файли, де коміти можна відформатувати та переглянути перед їх завершенням, а також відсилати частинами.

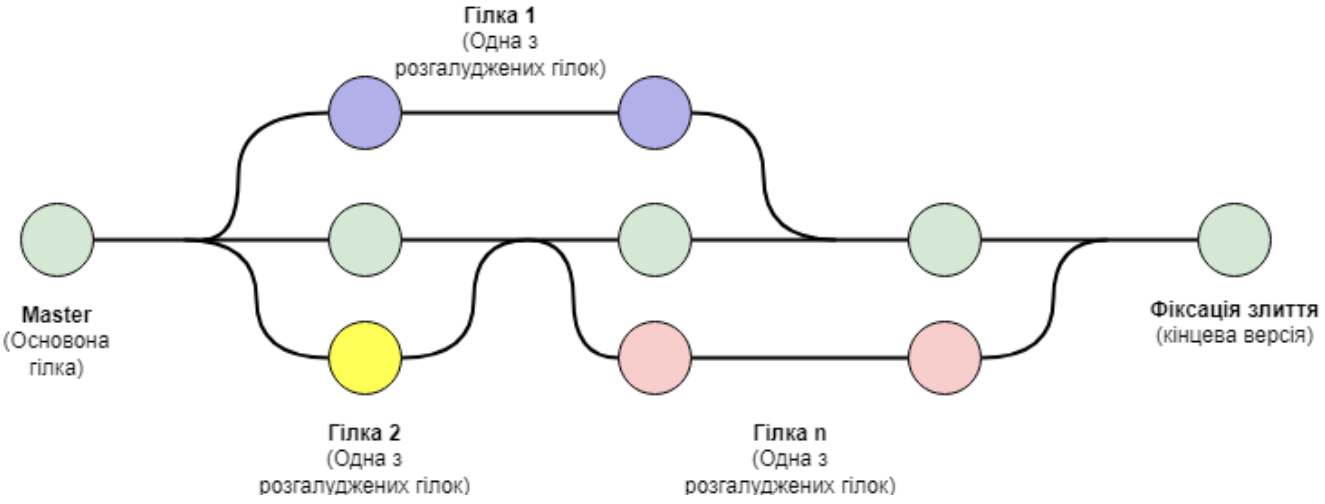


Рисунок 4.1.2.2.– Принцип роботи гілок в Git

Одною з проблем централізованої системи іноді виступає те, що люди можуть одночасно відсилати зміни цим самим створювати ризик виникнення помилок. Git задля усунення таких помилок обмежує одночасні відправки на сервер, автоматично забороняючи відправку, до моменту закінчення попередньої.

Всі операції з Git виконуються локально, що є перевагою в порівнянні з серверними централізованими системами. Git був написаний на мові C та розроблений на ядрі Linux, де при його створенні головною метою були продуктивність та швидкість роботи.

Для своєї роботи Git клонує дані всього репозиторію, тобто кожен з користувач має резервну копію цілого серверу. Це означає, що при збої або пошкодженні файлів, сервер можна відновити з копії.

В середині таких проектів формуються нові ролі, а саме менеджер інтеграції – людина, що бере на себе відповідальність збереження «благословенного» сховища, яке виступає в ролі чистового варіанту репозиторію. Тільки інтегратор може вносити зміни в це сховище.

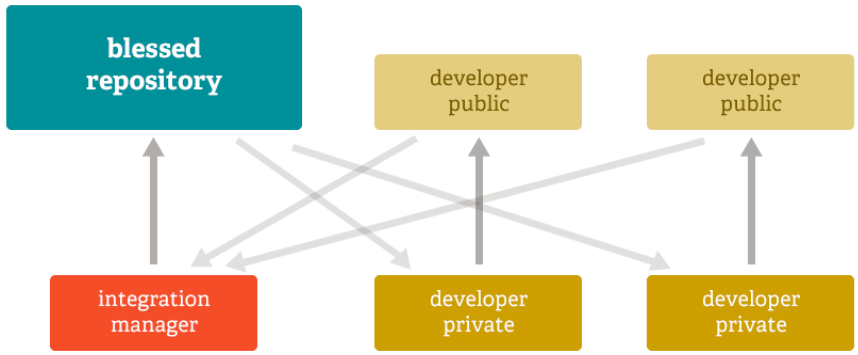


Рисунок 4.1.2.3.– Схема роботи інтегратора

Для більш масштабних проектів часто застосовують іншу модель розробки. Ця модель містить роль «лейтенанта» – людини, що відповідає лише за одну конкретну підсистему та працюють тільки в її області. Роль «диктатора», виступаюча інтегратором, отримує зміни лише від своїх підлеглих лейтенантів та може надсилати зміни до «благословенного» репозиторію, робота продовжується вже з клоном оновленого репозиторію.

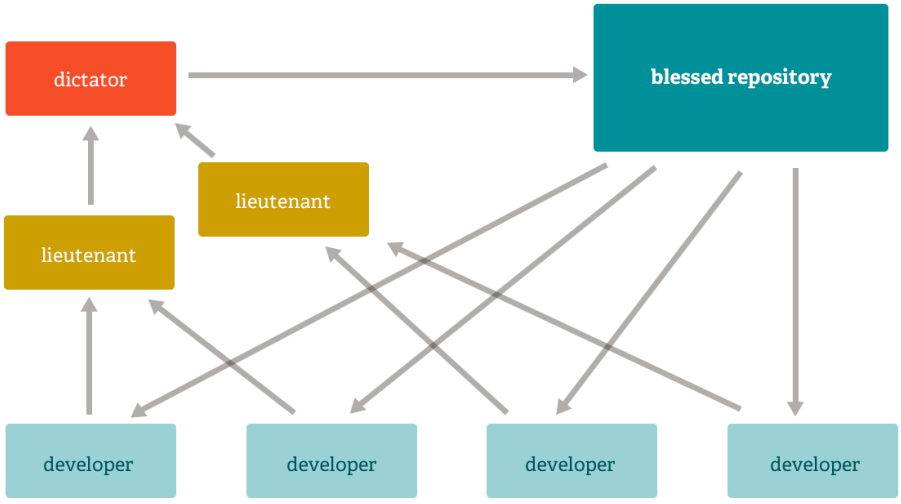


Рисунок 4.1.2.4.– Схема роботи лейтенанта та диктатора

Модель даних, яку використовує Git, забезпечує криптографічну цілісність кожного біта вашого проекту. Кожен файл і коміти підлягають підрахунку контрольної суми і витягуються за контрольною сумою, коли повертаються. З Git неможливо отримати що-небудь, окрім точних бітів, які ви вкладали.

Також неможливо змінити будь-який файл, дату, повідомлення про фіксацію або будь-які інші дані в репозиторії Git, не змінюючи ідентифікатори всього після нього. Це означає, що якщо у вас є ідентифікатор фіксації, ви можете бути впевнені не тільки в тому, що ваш проект точно такий же, як і на момент його фіксації, але й у тому, що в його історії нічого не змінено.

Більшість централізованих систем контролю версій за замовчуванням не забезпечують такої цілісності.

Найпопулярнішим та найбільшим серед SCM систем, що використовує Git виступає – GitHub, розроблений на Ruby on Rails і Erlang компанією GitHub, Inc. Він характеризується, як вебсервіс для спільної розробки програмного забезпечення.

Сервіс безкоштовний для проектів з відкритим вихідним кодом, з наданням користувачам усіх своїх можливостей (включаючи SSL), а для окремих індивідуальних проектів пропонуються різні платні тарифні плани.

4.2 Бібліотеки, що використовуються в роботі

4.2.1 Простір імен System.Security.Cryptography

Цей простір імен надає криптографічні послуги, містить у собі зв’язані з криптографією класи, структури, інтерфейси та перерахування, також забезпечує безпечне кодування та декодування даних, а також багато інших операцій, таких як хешування, генерація випадкових чисел та аутентифікація повідомлень.

У .NET класи в просторі імен System.Security.Cryptography управляють багатьма деталями криптографії, деякі з яких є обгортками для реалізацій операційної системи, тоді як інші є суто керованими реалізаціями. При генерації нового екземпляру одного з класів алгоритму шифрування, ключі автоматично генеруються для простоти використання, а властивості за замовчуванням є максимально безпечними та надійними.

.NET надає класи, які реалізують алгоритми шифрування з секретним та відкритим ключем (симетричне та асиметричне шифрування), цифрового підпису, алгоритми хешування та реалізацію алгоритму генератора випадкових чисел, а також маніфест ClickOnce та допоміжні класи криптографії нового покоління(CNG).

До класів, що реалізують алгоритми шифрування з секретним ключем можна віднести:

Aes – представляє абстрактний базовий клас, від якого повинні успадковуватися всі реалізації розширеного стандарту шифрування AES, таких як:

- AesCryptoServiceProvider – виконує симетричне шифрування та дешифрування за допомогою криптографічних програмних інтерфейсів (CAPI) алгоритму AES.
- Aes Managed – забезпечує керовану реалізацію симетричного алгоритму AES.

DES – Представляє базовий клас для алгоритму стандарту шифрування даних DES, від якого мають походити всі реалізації DES, такі як:

- DESCryptoServiceProvider – визначає об’єкт обгортки для доступу до версії алгоритму шифрування даних DES постачальника криптографічних послуг (CSP). Цей клас не може бути успадкований.

DSA – представляє абстрактний базовий клас, від якого повинні успадковуватися всі реалізації алгоритму цифрового підпису DSA.

- DSACryptoServiceProvider – визначає об’єкт обгортки для доступу до реалізації алгоритму DSA постачальника криптографічних послуг (CSP) . Цей клас не може бути успадкований.
- DSACng – Забезпечує реалізацію алгоритму цифрового підпису (DSA) у криптографії наступного покоління (CNG).

TripleDES – представляє базовий клас для стандартних алгоритмів потрійного шифрування даних, від якого мають походити всі реалізації TripleDES, а саме:

- TripleDESCryptoServiceProvider – визначає об’єкт обгортки для доступу до версії алгоритму TripleDES постачальника криптографічних послуг (CSP). Цей клас не може бути успадкований.
- TripleDESCng – Забезпечує реалізацію алгоритму потрійного стандарту шифрування даних (3DES) для криптографії наступного покоління (CNG).

CspParameters – містить параметри, які передаються постачальнику криптографічних послуг (CSP), який виконує криптографічні обчислення. Цей клас не може бути успадкований.

Та інші, а до класів, що реалізують алгоритми шифрування з відкритим ключем:

RSA – представляє базовий клас, від якого успадковуються всі реалізації алгоритму RSA , такі як:

- RSACryptoServiceProvider – виконує асиметричне шифрування та дешифрування за допомогою реалізації алгоритму RSA , наданого

постачальником криптографічних послуг (CSP). Цей клас не може бути успадкований.

- RSACng – забезпечує реалізацію алгоритму RSA криптографії наступного покоління (CNG).
- RSAOpenSsl – забезпечує реалізацію алгоритму RSA, що підтримується OpenSSL.
- RSAEncryptionPadding – визначає режим заповнення та параметри для використання з операціями шифрування або дешифрування RSA.

Rijndael – представляє базовий клас, від якого повинні успадковуватися всі реалізації алгоритму симетричного шифрування Рейндала, а саме:

- RijndaelManaged – отримує доступ до керованої версії алгоритму Рейндаля . Цей клас не може бути успадкований.

DSA – представляє абстрактний базовий клас, від якого повинні успадковуватися всі реалізації алгоритму цифрового підпису DSA, такі як:

- DSACryptoServiceProvider – визначає об’єкт обгортки для доступу до реалізації алгоритму DSA постачальника криптографічних послуг (CSP) . Цей клас не може бути успадкований.
- DSACng – забезпечує реалізацію алгоритму цифрового підпису (DSA) у криптографії наступного покоління (CNG).

ECDsa – надає абстрактний базовий клас, який інкапсулює алгоритм цифрового підпису еліптичної кривої (ECDSA).

- ECDsaCng – забезпечує реалізацію алгоритму цифрового підпису еліптичної кривої (ECDSA) у криптографії наступного покоління (CNG).
- ECDsaOpenSsl– забезпечує реалізацію алгоритму цифрового підпису еліптичної кривої (ECDSA), що підтримується OpenSSL.

Для реалізації цифрового підпису повідомлення, створюється дайджест повідомлення за допомогою хеш-алгоритму, воно представляє собою коротке та

унікальне представлення даних. Далі за допомогою закритого ключа одна зі сторін шифрує отриманий дайджест повідомлення для створення особистого підпису. Це повідомлення розшифровується іншою стороною за допомогою відкритого ключа, відновлюючи дайджест та хешує повідомлення тим самим алгоритмом, що був використаний першою стороною. Якщо зашифрований дайджест відповідає розшифрованому, то повідомлення гарантовано було надіслано від власника закритого ключа, що був відправником.

Хеш-алгоритми відображають двійкові значення довільної довжини в менші двійкові значення фіксованої довжини, відомі як хеш-значення. Хеш-значення — це числове представлення частини даних. Якщо ви хешуєте абзац відкритого тексту і змінюєте хоча б одну літеру абзацу, наступне хешування дасть інше значення. Якщо хеш криптографічно надійний, його значення значно зміниться. Наприклад, якщо змінити один біт повідомлення, сильна хеш-функція може вивести вихід, який відрізняється на 50 відсотків. Багато вхідних значень можуть хешувати до одного вихідного значення. Однак обчислювально неможливо знайти два різні вхідні дані, які хешують до однакового значення.

Генерація випадкових чисел є невід'ємною частиною багатьох криптографічних операцій. Наприклад, криптографічні ключі повинні бути максимально випадковими, щоб їх було неможливо відтворити. Криптографічні генератори випадкових чисел повинні генерувати вихідні дані, які неможливо передбачити з обчислювальної точки зору з імовірністю, що перевищує половину. Тому будь-який метод передбачення наступного вихідного біта не повинен працювати краще, ніж випадкове вгадування. В .NET клас RandomNumberGenerator використовують для генерації випадкових чисел при створенні криптографічних ключів.

Класи Cryptography Next Generation (CNG) забезпечують керовану обгортку навколо рідних функцій CNG. Центральним у класах обгортки CNG є клас контейнера ключів CngKey , який абстрагує зберігання та використання ключів CNG. Цей клас дозволяє безпечно зберігати пару ключів або відкритий ключ і звертатися до них за допомогою простого імені рядка. Клас сигнатури ECDsaCng

на основі еліптичної кривої та клас шифрування ECDiffieHellmanCng можуть використовувати об’єкти CngKey .

Клас CngKey використовується для різноманітних додаткових операцій, включаючи відкриття, створення, видалення та експорт ключів. Він також надає доступ до базового дескриптора клавіші для використання під час безпосереднього виклику рідних функцій.

.NET також включає в себе різноманітні допоміжні класи CNG, наприклад такі:

- CngProvider підтримує ключового постачальника сховища.
- CngAlgorithm підтримує алгоритм CNG.
- CngProperty підтримує часто використовувані ключові властивості.

4.2.2 Простір імен Microsoft.Win32

Надає два типи класів: ті, які обробляють події, викликані операційною системою, і ті, які маніпулюють системним реєстром.

Registry – надає об’єкти RegistryKey , які представляють кореневі ключі в реєстрі Windows і статичні методи доступу до пар ключ/значення.

Цей клас надає набір стандартних кореневих ключів, знайдених у реєстрі на машинах під керуванням Windows. Реєстр зберігає інформацію про програми, користувачів і системні параметри за замовчуванням. Ці дані можуть контролювати різні користувачі, так як інформація зберігається в різних розділах реєстру.

RegistryKey – являє собою вузол на рівні ключа в реєстрі Windows. Цей клас є інкапсуляцією реєстру.

Базові або кореневі екземпляри RegistryKey , які відкриває Registry клас, окреслюють основний механізм зберігання підрозділів і значень у реєстрі. Усі ключі доступні лише для читання, оскільки реєстр залежить від їх існування.

Клас Registry відкриває такі ключові поля:

ClassesRoot – визначає типи (або класи) документів та властивості, пов’язані з цими типами. У цьому полі зчитується базовий ключ реєстру Windows HKEY_CLASSES_ROOT.

CurrentConfig – містить інформацію про конфігурацію апаратного забезпечення, яка не є специфічною для користувача. У цьому полі зчитується базовий ключ реєстру Windows HKEY_CURRENT_CONFIG.

CurrentUser – містить інформацію про поточні налаштування користувача. У цьому полі зчитується базовий ключ реєстру Windows HKEY_CURRENT_USER.

LocalMachine – містить дані конфігурації для локальної машини. У цьому полі зчитується базовий ключ реєстру Windows HKEY_LOCAL_MACHINE.

PerformanceData – містить інформацію про продуктивність програмних компонентів. У цьому полі зчитується базовий ключ реєстру Windows HKEY_PERFORMANCE_DATA.

Users – містить інформацію про конфігурацію користувача за замовчуванням. У цьому полі зчитується базовий ключ реєстру Windows HKEY_USERS.

А також методи:

GetValue(String, String, Object) – отримує значення, пов’язане з вказаним ім’ям, у вказаному розділі реєстру. Якщо ім’я не знайдено у зазначеному ключі, повертає значення за замовчуванням, яке ви надаєте, або null якщо вказаний ключ не існує.

SetValue(String, String, Object, RegistryValueKind) – встановлює пару ім’я/значення для зазначеного розділу реєстру, використовуючи вказаний тип даних реєстру, якщо потрібно. Якщо вказаний ключ не існує, він створюється.

Клас RegistryKey надає такі методи:

CreateSubKey(String) – створює новий підрозділ або відкриває наявний підрозділ для доступу до запису, має перевантаження з різними параметрами.

DeleteSubKey(String) – видаляє вказаний підрозділ, має перевантаження з різними параметрами.

DeleteSubKeyTree(String) – рекурсивно видаляє підрозділ і будь-які дочірні підрозділи.

DeleteValue(String) – видаляє вказане значення з цього ключа.

Dispose() – звільняє всі ресурси, які використовуються поточним екземпляром класу RegistryKey .

Flush() – записує всі атрибути вказаного відкритого ключа реєстру в реєстр.

Так інші.

4.2.3 Простір імен System.IO

Містить класи, що дозволяють читати та записувати файли та потоки даних, а також типи для базової підтримки файлів і папок, деякі з них:

File – надає статичні методи створення, копіювання, видалення, переміщення та відкриття одного файлу, а також допомагає при створенні об'єктів FileStream .

File Stream – надає Stream у файлі, підтримуючи синхронні та асинхронні операції читання та запису.

Memory Stream – створює потік, резервним сховищем якого є пам'ять.

Binary Reader – зчитує примітивні типи даних як двійкові значення заданого кодування.

Binary Writer – записує примітивні типи у двійковий потік та підтримує запис рядків у заданому кодуванні.

Stream Reader – реалізує об'єкт TextReader , який зчитує символи з потоку байтів у певному кодуванні.

Stream Writer – реалізує TextWriter для запису символів у потік у певному кодуванні.

Та інші.

ВИСНОВКИ

Сумуючи всю виконану роботу можна дійти до декількох висновків, а саме зрозуміти те, що актуальність теми криптографії з плином часу буде лише зростати і будуть з’являтися все нові методи, алгоритми та обчислювальні потужності, які дадуть новий ривок в цій галузі. Адже кількість даних, що передаються та створюються по всьому світі лише зростає в прогресії, разом з потужностями новітніх обчислювальних систем, що полегшують можливість злому старіших алгоритмів, тому на заміну старим алгоритмам будуть приходити нові.

Перший розділ знайомить нас з поняттями, предметом та галузями криптографії, те як інформація захищається шляхом її перетворення. Проводиться аналіз існуючих на сьогодні методів шифрування та алгоритмів, їх об’єднання, модифікації та застосовані технології для їх реалізації, а також принципів їх роботи.

Завданням на цю роботу було розробити програмний продукт, що надає можливість локально шифрувати та дешифрувати файли на локальних носіях ПК, згідно визначеним вимогам до якості системи у другому розділі, такими як функціональні та нефункціональні вимоги, наведено опис структури проекту у вигляді діаграм.

Зважаючи на всі вимоги, було розроблено програмний продукт на об’єктно орієнтованій мові програмування C# з використанням крос-платформової технології .NET Framework у середовищі розробки Visual Studio. Було використано програмний інтерфейс застосунку – Windows Forms API, що детально описані в четвертому розділі.

Кінцевим продуктом об’єднання теоретичних матеріалів у програмному проекті є створена програма для шифрування та дешифрування файлів, що містить в собі реалізацію розглянутих криптографічних алгоритмів, приклад роботи якої та її структура детально описана в третьому розділі.

ПЕРЕЛІК ПОСИЛАНЬ

1. Введение в криптографию / Под общ. ред. В. В. Ященко. — 4-е изд., доп. М.: МЦНМО, 2012. — 10-17, 91-93 с.;
2. Rivest R. L., Shamir A., Adleman L. A method for obtaining digital signatures and public key cryptosystems // Commun. ACM. V.21, No 2, 1978.;
3. Компьютерная стеганография. Теория и практика.— К.: "МКПресс", 2006. — 18-21 с.;
4. Moserware [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: http://www.moserware.com/2009/09/stick-figure-guide-to-advanced.html (дата звернення 30.04.2022) – A Stick Figure Guide to the Advanced Encryption Standard (AES);
5. Related-key Cryptanalysis of the Full AES-192 and AES-256 — Alex Biryukov and Dmitry — Khovratovich University of Luxembourg, 2009 — 19 p.
6. FIPS-197: Advanced Encryption Standard – November 2001 [Електронний ресурс] –Режим доступу: https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.197.pdf — 51 p.;
7. Основы современной криптографии — Баричев С.Г. и др. — М.: «Горячая линия – Телеком», 2001 – с. 10-11, 21-24.;
8. Прикладная криптография — Брюс Шнайер — 2-е издание, Протоколы, алгоритмы и исходные тексты на языке, 2016 — 610 с.;
9. National Bureau of Standards, NBS FIPS PUB 81, "DES Modes of Operation," U.S. Department of Commerce, Dec 1980;
10. IEEE Std 830-1998 — IEEE Recommended Practice for Software Requirements Specifications -Description, 1998 — 37p
11. Microsoft Docs [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://docs.microsoft.com/en-us/visualstudio/get-started/visual-studio-ide?view=vs-2019 (дата звернення 2.05.2022) – Welcome to the Visual Studio IDE;

12. Microsoft Docs [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://docs.microsoft.com/en-us/dotnet/api/system.security.cryptography?view=net-6.0

13. (дата звернення 4.05.2022) – System.Security.Cryptography Namespace;

14. Microsoft Docs [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://docs.microsoft.com/en-us/dotnet/api/system.io?view=net-6.0 (дата звернення 2.05.2022) – System.IO Namespace;

15. Microsoft Docs [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://docs.microsoft.com/en-s/dotnet/api/microsoft.win32?view=net-6.0 (дата звернення 4.05.2022) – Microsoft.Win32 Namespace;

16. Git-SCM [Електронний ресурс] : [Веб-сайт]. – Електронні дані. – Режим доступу: https://git-scm.com/about (дата звернення 5.05.2022) – About Git;

ДОДАТКИ

Додаток А. Лістинг

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Runtime.InteropServices;
using System.Security.Cryptography;
using System.Windows.Forms;
using AAJGen;

namespace File_encryption_decryption
{
    public partial class Form1 : Form
    {
        #region Variables
        private string RegisterPath;
        List<Panel> panelList = new List<Panel>();
        RandomGen randomGen = new RandomGen(Option.IncludeCapital);
        OpenFileDialog openFileDialog = new OpenFileDialog();
        EncipherAES aES = new EncipherAES();
        EncipherRSA rSA = new EncipherRSA();
        EncipherDES dES = new EncipherDES();
        EncipherTripleDES tDES = new EncipherTripleDES();
        #endregion

        public Form1(string[] args)
        {
            InitializeComponent();
            if (args.Length > 0)
            {
                RegisterPath = args[0];
            }
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            path_textBox.Text = RegisterPath;
            comboBox1.SelectedIndex = 0;
            panelList.Add(DES_panel);
            panelList.Add(TripleDES_panel);
            panelList.Add(AES_panel);
            panelList.Add(RSA_panel);
        }
        private void Form1_Layout(object sender, LayoutEventArgs e)
        {
            path_textBox.BringToFront();
        }


        private void Exit_button_Click(object sender, EventArgs e)
        {
            Application.Exit();
        }



        #region Form
        [DllImport("user32.DLL", EntryPoint = "ReleaseCapture")]
        private extern static void ReleaseCapture();
        [DllImport("user32.DLL", EntryPoint = "SendMessage")]
        private extern static void SendMessage(IntPtr hWnd, int wMsg, int wParam, int lParam);
        private void panel1_MouseDown(object sender, MouseEventArgs e)
```

```
        {
            ReleaseCapture();
            SendMessage(this.Handle, 0x112, 0xf012, 0);
        }

        private void Minimize_button_Click(object sender, EventArgs e)
        {
            WindowState = FormWindowState.Minimized;
        }

        private void label1_MouseDown(object sender, MouseEventArgs e)
        {
            ReleaseCapture();
            SendMessage(this.Handle, 0x112, 0xf012, 0);
        }
        #endregion

        private void comboBox1_SelectionChangeCommitted(object sender, EventArgs e)
        {
            for (int i = 0; i < panelList.Count; i++)
                panelList[i].Visible = panelList[i].Name.Equals(comboBox1.Text + "_panel") ?
true : false;
        }


        private void RandomKeyGenarator(object sender, EventArgs e)
        {
            foreach (Panel panel in panelList)
            {
                if (panel.Visible == true)
                {
                    foreach (Control control in panel.Controls)
                    {
                        if (control is TextBox)
                        {
                            control.Text = randomGen.Gen(((TextBox)control).MaxLength);
                            Clipboard.SetText(((TextBox)control).Text);
                        }
                    }
                }
            }
        }


        private void File_select_button_Click(object sender, EventArgs e)
        {
            openFileDialog.Filter = "All files (*.*)|*.*";
            openFileDialog.Title = "Оберіть файл для шифрування/дешифрування";
            if (openFileDialog.ShowDialog() == DialogResult.OK)
            {
                path_textBox.Text = openFileDialog.FileName;
            }
        }

        private void Encrypt_button_Click(object sender, EventArgs e)
        {
            for (int i = 0; i < panelList.Count; i++)
                if (panelList[i].Visible)
                {
                    switch (i)
                    {
                        case 0:
                            dES.EncryptFile(path_textBox.Text, dES_key_textBox.Text);
                            break;
                        case 1:
```

```
                    tDES.EncryptFile(path_textBox.Text, tDES_key_textBox.Text);
                    break;
                case 2:
                    aES.EncryptFile(path_textBox.Text, aES_key_textBox.Text);
                    break;
                case 3:
                    rSA.EncryptFile(path_textBox.Text, rSA_public_key_textBox.Text);
                    break;
            }
            break;
        }
}
private void Decrypt_button_Click(object sender, EventArgs e)
{
    for (int i = 0; i < panelList.Count; i++)
        if (panelList[i].Visible)
        {
            switch (i)
            {
                case 0:
                    dES.DecryptFile(path_textBox.Text, dES_key_textBox.Text);
                    break;
                case 1:
                    tDES.DecryptFile(path_textBox.Text, tDES_key_textBox.Text);
                    break;
                case 2:
                    aES.DecryptFile(path_textBox.Text, aES_key_textBox.Text);
                    break;
                case 3:
                    rSA.DecryptFile(path_textBox.Text, rSA_public_key_textBox.Text);
                    break;
            }
            break;
        }
}


private void select_public_key_button_Click(object sender, EventArgs e)
{
    openFileDialog.Filter = "XML - File | *.xml";
    openFileDialog.Title = "Оберіть відкритий RSA ключ";
    if (openFileDialog.ShowDialog() == DialogResult.OK)
    {
        rSA_public_key_textBox.Text = openFileDialog.FileName;
    }
}

private void select_private_key_button_Click(object sender, EventArgs e)
{
    openFileDialog.Filter = "XML - File | *.xml";
    openFileDialog.Title = "Оберіть закритий RSA ключ";
    if (openFileDialog.ShowDialog() == DialogResult.OK)
    {
        rSA_private_key_textBox.Text = openFileDialog.FileName;
    }
}

private void generate_RSA_Keys_Click(object sender, EventArgs e)
{
    RSACryptoServiceProvider rSACryptoService = new RSACryptoServiceProvider();
    SaveFileDialog saveFileDialog1 = new SaveFileDialog();
    saveFileDialog1.Title = "Збережіть RSA ключ";
    saveFileDialog1.FileName = "RSA Public Key";
    saveFileDialog1.Filter = "XML - File | *.xml";
    if (saveFileDialog1.ShowDialog() == DialogResult.OK)
```

```
            {
                File.WriteAllText(saveFileDialog1.FileName,
rSACryptoService.ToXmlString(false));
            }
            saveFileDialog1.FileName = "RSA Private Key";
            if (saveFileDialog1.ShowDialog() == DialogResult.OK)
            {
                File.WriteAllText(saveFileDialog1.FileName,
rSACryptoService.ToXmlString(true));
            }
        }
    }
}

using System;
using System.Windows.Forms;

namespace File_encryption_decryption
{
     static class Program
    {
        [STAThread]


        static void Main(string[] args)
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1(args));

        }
    }
}

namespace File_encryption_decryption
{
    interface IEncryption
    {
        void EncryptFile(string filePath, string publicKeyPath);
        void DecryptFile(string filePath, string publicKeyPath);

    }
}

using Microsoft.Win32;
using System.Collections;
using System.ComponentModel;
using System.Configuration.Install;

namespace File_encryption_decryption
{
    [RunInstaller(true)]
    public partial class Installer1 : Installer
    {
        const string rootKey = "*\\shell";
        const string keyCommand = "\\command";
        const string openParam = " \"%1\"";
        const string fileName = "\\File Encrypt/Decrypt";

        public Installer1()
        {
            InitializeComponent();
        }

        public override void Commit(IDictionary stateSaver)
```

```
        {
            base.Commit(stateSaver);
            string pathStr = Context.Parameters["assemblypath"];
            string path = pathStr + openParam;
            Registry.ClassesRoot.CreateSubKey(rootKey + fileName).SetValue("", "File
Encryption/Decryption");
            Registry.ClassesRoot.CreateSubKey(rootKey + fileName).SetValue("Icon", pathStr);
            Registry.ClassesRoot.CreateSubKey(rootKey + fileName + keyCommand).SetValue("",
path);
        }
        public override void Uninstall(IDictionary stateSaver)
        {
            base.Uninstall(stateSaver);
            Registry.ClassesRoot.DeleteSubKeyTree(rootKey + fileName);
        }
    }
}


using System;
using System.Text;
using System.Security.Cryptography;
using System.IO;
using System.Windows.Forms;

namespace File_encryption_decryption
{
    class EncipherDES : IEncryption
    {
        DESCryptoServiceProvider dES = new DESCryptoServiceProvider();
        byte[] fileBytes;
        byte[] encBytes;

        public EncipherDES()
        {
            try
            {
                dES.Mode = CipherMode.ECB;
                dES.Padding = PaddingMode.PKCS7;
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Помилка!");
            }
        }


        public void EncryptFile(string filePath, string key)
        {
            try
            {
                dES.Key = Encoding.UTF8.GetBytes(key);
                fileBytes = File.ReadAllBytes(filePath);
                encBytes = dES.CreateEncryptor().TransformFinalBlock(fileBytes, 0,
fileBytes.Length);
                File.WriteAllBytes(filePath, encBytes);
                MessageBox.Show("Файл успішно зашифровано.", "Успіх!");
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Помилка!");
                return;
            }
        }
```

```
        public void DecryptFile(string filePath, string key)
        {
            try
            {
                dES.Key = Encoding.UTF8.GetBytes(key);
                fileBytes = File.ReadAllBytes(filePath);
                encBytes = dES.CreateDecryptor().TransformFinalBlock(fileBytes, 0,
fileBytes.Length);
                File.WriteAllBytes(filePath, encBytes);
                MessageBox.Show("Файл успішно розшифровано.", "Успіх!");
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Помилка!");
                return;
            }
        }


    }
}


using System;
using System.Text;
using System.Security.Cryptography;
using System.IO;
using System.Windows.Forms;

namespace File_encryption_decryption
{
    class EncipherTripleDES : IEncryption
    {
        private TripleDESCryptoServiceProvider tDESp = new TripleDESCryptoServiceProvider();
        byte[] fileBytes;
        byte[] encBytes;

        public EncipherTripleDES()
        {
            try
            {
                tDESp.Mode = CipherMode.CBC;
                tDESp.Padding = PaddingMode.PKCS7;
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Помилка!");
            }
        }

        public void EncryptFile(string filePath, string key)
        {
            try
            {
                tDESp.Key = Encoding.UTF8.GetBytes(key);
                fileBytes = File.ReadAllBytes(filePath);
                encBytes = tDESp.CreateEncryptor().TransformFinalBlock(fileBytes, 0,
fileBytes.Length);
                File.WriteAllBytes(filePath, encBytes);
                MessageBox.Show("Файл успішно зашифровано.", "Успіх!");
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Помилка!");
            }
```

```
        }
        public void DecryptFile(string filePath, string key)
        {
            try
            {
                tDESp.Key = Encoding.UTF8.GetBytes(key);
                fileBytes = File.ReadAllBytes(filePath);
                encBytes = tDESp.CreateDecryptor().TransformFinalBlock(fileBytes, 0,
fileBytes.Length);
                File.WriteAllBytes(filePath, encBytes);
                MessageBox.Show("Файл успішно розшифровано.", "Успіх!");
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Помилка!");
            }
        }


    }
}


using System;
using System.Text;
using System.Security.Cryptography;
using System.IO;
using System.Windows.Forms;

namespace File_encryption_decryption
{
    class EncipherAES : IEncryption
    {
        RijndaelManaged aes = new RijndaelManaged();
        byte[] keyBytes;
        byte[] saltBytes = { 1, 2, 3, 4, 5, 6, 7, 8 };
        byte[] fileBytes;

        public EncipherAES()
        {
            aes.Mode = CipherMode.CBC;
            aes.Padding = PaddingMode.PKCS7;
            aes.KeySize = 256;
            aes.BlockSize = 128;
        }


        public void EncryptFile(string filePath, string strKey)
        {
            try
            {
                keyBytes = Encoding.UTF8.GetBytes(strKey);
                using (MemoryStream memory = new MemoryStream())
                {
                    Rfc2898DeriveBytes key = new Rfc2898DeriveBytes(keyBytes, saltBytes,
2000);
                    aes.Key = key.GetBytes(aes.KeySize / 8);
                    aes.IV = key.GetBytes(aes.BlockSize / 8);
                    using (CryptoStream cryptoStream = new CryptoStream(memory,
aes.CreateEncryptor(), CryptoStreamMode.Write))
                    {
                        fileBytes = File.ReadAllBytes(filePath);
                        cryptoStream.Write(fileBytes, 0, fileBytes.Length);
                        cryptoStream.FlushFinalBlock();
```

```
                    }
                    File.WriteAllBytes(filePath, memory.ToArray());
                    MessageBox.Show("Файл успішно зашифровано.", "Успіх!");
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Помилка!");
            }
        }

        public void DecryptFile(string filePath, string strKey)
        {
            try
            {
                keyBytes = Encoding.UTF8.GetBytes(strKey);
                using (MemoryStream memory = new MemoryStream())
                {
                    Rfc2898DeriveBytes key = new Rfc2898DeriveBytes(keyBytes, saltBytes,
2000);
                    aes.Key = key.GetBytes(aes.KeySize / 8);
                    aes.IV = key.GetBytes(aes.BlockSize / 8);
                    using (CryptoStream cryptoStream = new CryptoStream(memory,
aes.CreateDecryptor(), CryptoStreamMode.Write))
                    {
                        fileBytes = File.ReadAllBytes(filePath);
                        cryptoStream.Write(fileBytes, 0, fileBytes.Length);
                        cryptoStream.FlushFinalBlock();
                    }
                    File.WriteAllBytes(filePath, memory.ToArray());
                    MessageBox.Show("Файл успішно розшифровано.", "Успіх!");
                }
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Помилка!");
            }
        }


    }
}


using System;
using System.IO;
using System.Security.Cryptography;
using System.Windows.Forms;

namespace File_encryption_decryption
{
    class EncipherRSA : IEncryption
    {
        RSACryptoServiceProvider rSA = new RSACryptoServiceProvider();
        byte[] fileBytes;
        byte[] encBytes;

        public void EncryptFile(string filePath, string publicKeyPath)
        {
            try
            {
                rSA.FromXmlString(File.ReadAllText(publicKeyPath));
                fileBytes = File.ReadAllBytes(filePath);
                encBytes = rSA.Encrypt(fileBytes, false);
                File.WriteAllBytes(filePath, encBytes);
```

```
                MessageBox.Show("Файл успішно зашифровано.", "Успіх!");
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Помилка!");
            }
        }

        public void DecryptFile(string filePath, string privateKeyPath)
        {
            try
            {
                rSA.FromXmlString(File.ReadAllText(privateKeyPath));
                fileBytes = File.ReadAllBytes(filePath);
                encBytes = rSA.Decrypt(fileBytes, false);
                File.WriteAllBytes(filePath, encBytes);
                MessageBox.Show("Файл успішно розшифровано.", "Успіх!");
            }
            catch (Exception ex)
            {
                MessageBox.Show(ex.Message, "Помилка!");
            }
        }


    }
}
```

ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ТЕЛЕКОМУНІКАЦІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
Кафедра Інженерії програмного забезпечення

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ШИФРУВАННЯ ТА ДЕШИФРУВАННЯ ФАЙЛІВ МОВОЮ C#

Виконавець: студент 4 курсу,
групи ПД-43
Музика Владислав Павлович
Керівник роботи: к.т.н., доцент
Трінтіна Наталія Альбертівна

Київ 2022

Аналіз та порівняння аналогів

AxCrypt

gKrypt Engine

Критерії оцінювання	Назва продукту			
	AxCrypt	Silver Key	gKrypt Engine	File Encrypttion/Decryption
Інтеграція в контекстне меню Windows	+	+	-	+
Спосіб шифрування	AES 128 AES 256	AES 256	AES 256	AES 256 DES TDES RSA
Особливості	Можливість перегляду зашифрованих файлів з телефону	Можливість завантажувати в хмарнесховище	Може використовувати GPU для прискорення шифрування	Можливість послідовного шифрування
Можливість генерації ключа	-	-	-	+

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

Мета роботи – поліпшення методик шифрування за рахунок створення програмному застосунку з можливістю послідовного об’єднання алгоритмів у нові гібридні системи.

Об’єкт дослідження – захист інформації шляхом криптографічних перетворень інформації задля збереження приватності даних.

Предмет дослідження – алгоритми, методики та принципи роботи криптографічних перетворень інформації.

Технічні завдання

1. Виконувати шифрування обраного файлу одним з обраних способів.
2. Виконувати дешифрування зашифрованих файлів.
3. Інтеграція програми в контексте меню Windows.
4. Можливість генерації ключів.
5. Можливість послідовного комбінованування алгоритмів.

Інструменти, використані в роботі

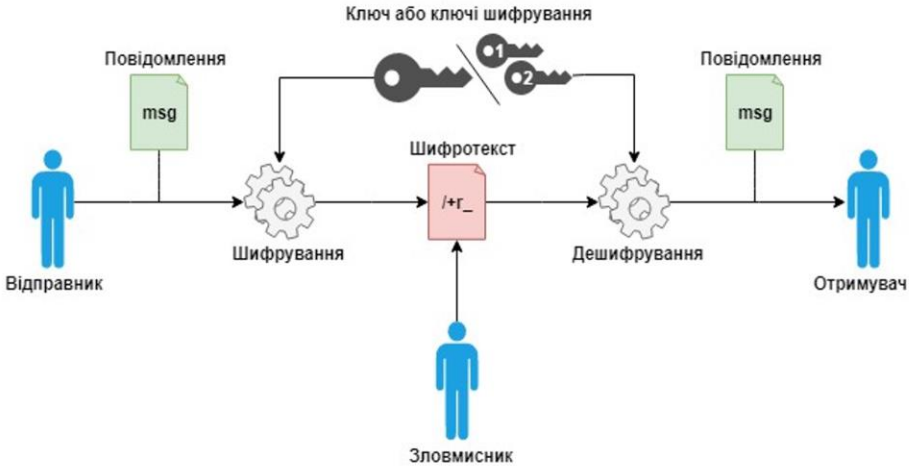
IDE (Integrated development environment) Visual Studio — інтегроване середовище розробки програмного забезпечення з сукупністю інших інструментальних засобів.

C# — об'єктно-орієнтована мова програмування. Розроблена як мова для розробки програм для платформи Microsoft **.NET Framework** і **.NET Core**

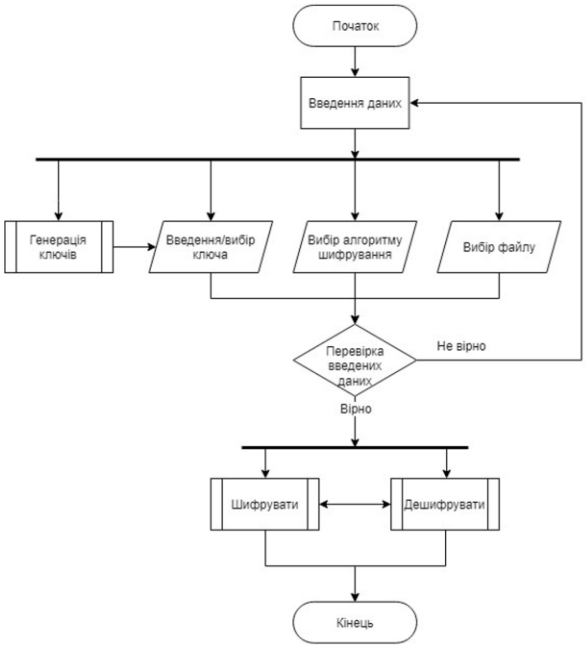
SCM (Source Code Management) Git — розподілена система керування версіями файлів та спільної роботити.

GitHub — найпопулярніша серед **SCM** систем, що використовує **Git** виступає, як вебсервіс для спільної розробки програмного забезпечення.

Типова криптогрфічна ситуація

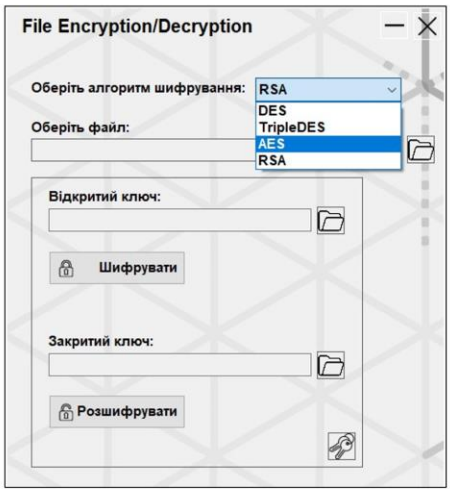


Алгоритм роботи програми

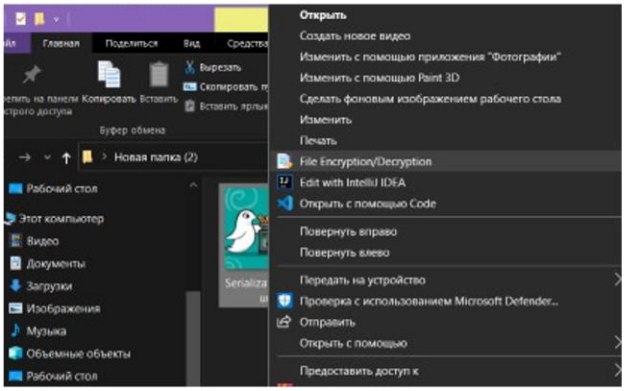


Діаграма варіантів використання користувача

Екранні форми



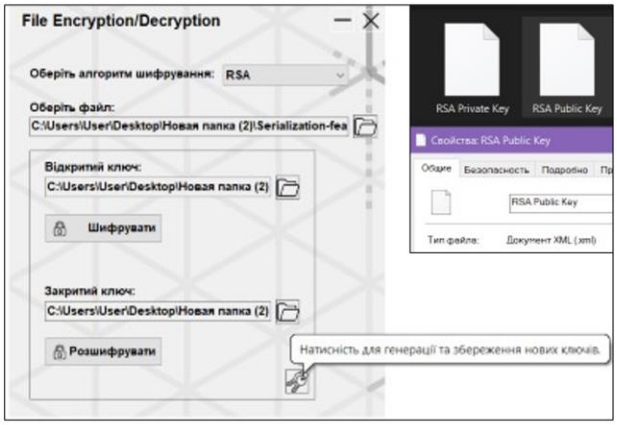
Вибір алгоритму

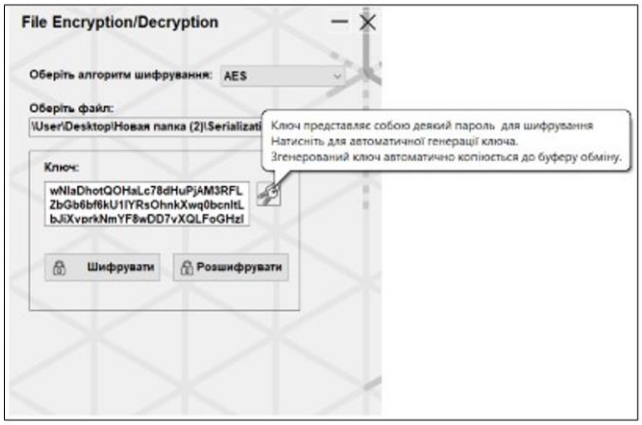


Інтеграція в контекстне меню Windows

Вибір файлу

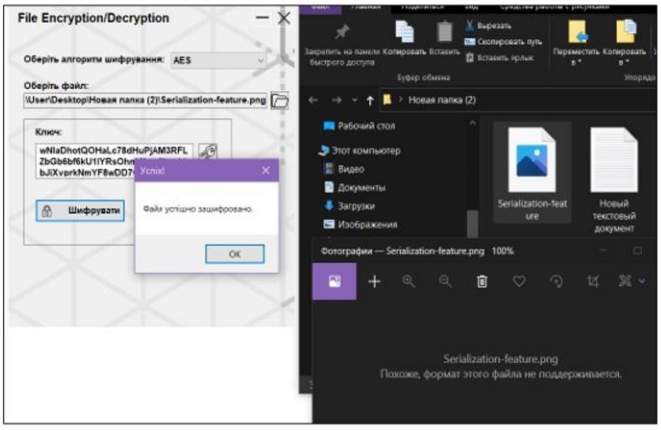
Екранні форми



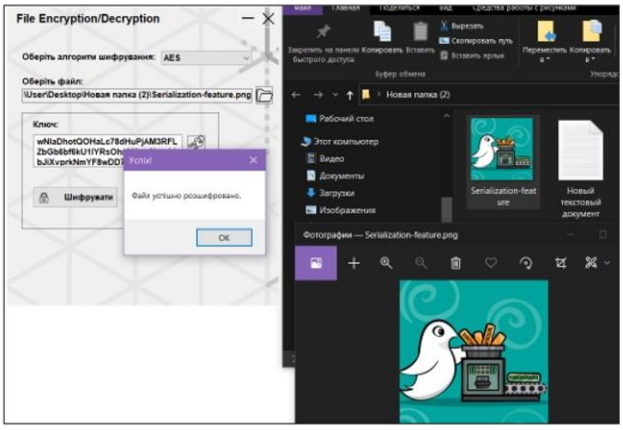


Генерація ключів

Екранні форми



Результат шифрування



Результат дешифрування

АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

Музика В.П., Основні поняття криптографії. Стародавня та сучасна криптографія / Науково-технічна конференція «Застосування програмного забезпечення в інфокомунікаційнихтехнологіях». Збірник тез 20.04.2022, ДУТ, м.Київ – К.: ДУТ, 2021. С. 56-57.

Висновки

1. Розглянуто та обгрунтованоактуальність дослідженої теми.
2. Розроблено алгоритм роботи системи.
3. Розроблено програмний продукт для шифрування та дешифрування файлів.
4. Реалізована можливість послідовного шифрування.
5. Реалізована інтеграція програми в контексте менюWindows.
6. Впроваджена можливість генерації ключів.
7. Проведено мануальне функціональне та нефункціональне тестування додатку.

Дякую за увагу!