МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВОЛИНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ЛЕСІ УКРАЇНКИ
Кафедра комп'ютерних наук та кібербезпеки

РЕАЛІЗАЦІЯ МЕТОДУ ШИФРУВАННЯ AES ЗАСОБАМИ ПЛАТФОРМИ .NET

Виконав:
Королюк Денис Володимирович
студент групи КНІТ-43
факультету інформаційних
технологій і математики

Науковий керівник:
Мамчич Тетяна Іванівна
доцент кафедри
комп'ютерних наук та
кібербезпеки, кандидат
фізико-математичних наук

Луцьк 2024

ЗМІСТ

ВСТУП

В сучасному світі, де інформаційні технології проникають у всі сфери життя, захист інформації набуває надзвичайного значення. Зважаючи на швидкий розвиток технологій та збільшення кількості кібератак, забезпечення надійного захисту даних стає критично важливим завданням для всіх установ.

Втрата або несанкціонований доступ до будь-якої інформації може мати серйозні наслідки як для окремих осіб, так і для установ загалом. Незважаючи на впровадження різних заходів безпеки, багато закладів все ще залишаються вразливими до кіберзагроз, що підкреслює необхідність постійного вдосконалення та оновлення існуючих методів захисту.

Один з основних методів захисту інформації – це її шифрування. Шифрування забезпечує конфіденційність даних, роблячи їх незрозумілими для осіб, які не мають відповідних ключів для дешифрування. Серед різних алгоритмів шифрування, одним з найбільш поширених і надійних є Advanced Encryption Standard (AES). Проте, незважаючи на його високу ефективність, завжди існує потреба у вдосконаленні алгоритмів шифрування для забезпечення ще більшої безпеки та ефективності [1].

Аналізуючи поточну ситуацію, можна відзначити, що більшість з використовують застарілі або недостатньо ефективні методи захисту даних. Це пов'язано з обмеженими фінансовими ресурсами, недостатнім рівнем обізнаності персоналу та відсутністю сучасного програмного забезпечення. У таких умовах впровадження новітніх технологій шифрування та захисту інформації стає не просто бажаним, а необхідним.

Критичний аналіз існуючих рішень показує, що навіть найкращі з них мають свої обмеження та недоліки. Наприклад, деякі алгоритми шифрування можуть бути вразливими до певних типів атак, або ж їх впровадження може бути занадто складним та дорогим для невеликих закладів. Тому необхідно

розробляти нові підходи та удосконалювати існуючі методи, адаптуючи їх до конкретних потреб установ.

Порівняння з відомими рішеннями також вказує на те, що сучасні алгоритми шифрування потребують оптимізації для підвищення їх продуктивності та зручності використання. Наприклад, AES, який широко використовується для захисту даних, може бути вдосконалений для забезпечення більш швидкого та ефективного шифрування, що особливо важливо в умовах обмежених ресурсів.

Зважаючи на вищезазначене, актуальність дослідження, спрямованого на вдосконалення алгоритму кодування інформації та його практичне застосування, є беззаперечною. Така робота сприятиме підвищенню рівня інформаційної безпеки, зменшенню ризиків втрати даних та підвищенню довіри до інформаційних систем закладів. Вона також стане важливим кроком у розвитку галузі комп'ютерних наук, пропонуючи нові рішення для забезпечення захисту інформації в умовах зростаючих кіберзагроз.

Мета даного дослідження полягає у вдосконаленні алгоритму кодування інформації AES для забезпечення ефективного захисту даних, а також демонстрації його використання для шифрування, збереження та розшифрування інформації.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

– провести аналіз існуючих методів та засобів захисту інформації, визначити їхні недоліки та вразливі місця;

– розробити вдосконалений алгоритм кодування інформації на основі AES, враховуючи специфічні потреби та умови установ;

– реалізувати та протестувати розроблений алгоритм, перевірити його ефективність у реальних умовах;

– розробити рекомендації та методичні вказівки для впровадження вдосконаленого алгоритму.

Об’єктом дослідження є процес захисту інформації, що включає в себе різноманітні методи та засоби забезпечення інформаційної безпеки, а також дії та заходи, спрямовані на запобігання несанкціонованому доступу до даних.

Предметом дослідження є вдосконалення алгоритму кодування інформації AES, що застосовується для шифрування, збереження та розшифрування даних в установах.

Практичне значення одержаних результатів полягає у підвищенні рівня інформаційної безпеки завдяки впровадженню вдосконаленого алгоритму кодування інформації AES. Це дозволить ефективніше захищати конфіденційні дані від несанкціонованого доступу, забезпечуючи їхню цілісність та конфіденційність.

РОЗДІЛ 1
Теоретичні аспекти захисту інформації та реалізації програмного продукту

1.1 Основні поняття та визначення захисту інформації

Захист інформації у сучасних компаніях набуває все більшої важливості. Це пов'язано зі зростаючими обсягами цінних даних, що зберігаються та обробляються у різних установах, а також з постійними змінами в кіберзагрозах. Одним із головних аспектів ефективного захисту є розуміння основних термінів і понять, пов'язаних з інформаційною безпекою, таких:

– конфіденційність – властивість інформації, що гарантує доступ до неї лише авторизованим користувачам. Захист конфіденційності передбачає застосування методів шифрування, доступу за паролем та інших засобів аутентифікації;

– цілісність – властивість інформації, яка гарантує її точність та повноту. Цілісність забезпечується за допомогою контрольних сум, хешування, цифрових підписів та інших методів, які запобігають несанкціонованим змінам даних;

– доступність – властивість інформації, яка гарантує можливість її отримання авторизованими користувачами у потрібний час. Доступність забезпечується за допомогою резервного копіювання, захисту від відмов, мережевих протоколів та інших методів, що мінімізують ризики недоступності інформації [2].

Найкраща практика визначає управління ризиками інформаційної безпеки в межах активів, загроз, вразливостей і подій. Однак виявити їх в організації може бути складно, оскільки інформаційні активи постійно створюються, обробляються та зберігаються. По-друге, середовище загроз у кіберпросторі постійно змінюється, де нові методи та інструменти ускладнюють ідентифікацію, оцінку та відображення шкідливих атак для

організації. Також зміни в організаційній структурі можуть виявити нові невраховані вразливості.

1.2 Методи шифрування інформації

Шифрування інформації є процесом перетворення даних у форму, яка є недоступною для неавторизованих осіб. Існує декілька загальних методів шифрування інформації, які можна розділити на дві основні категорії: симетричне шифрування та асиметричне шифрування.

У симетричному шифруванні один і той самий ключ використовується як для шифрування, так і для дешифрування даних. Основні методи симетричного шифрування включають в себе блочне шифрування та потокове шифрування. Перевагами симетричного шифрування можна назвати швидкість та ефективність шифрування та можливість шифрування великого обсягу даних. Недоліками в той же період є те що секретний ключ повинен бути безпечно переданий всім сторонам, які потребують доступу до даних і виходячи з цього витік ключа призведе до компрометації всієї інформації, яка була зашифрована цим ключем.

Прикладами блочного симетричного шифрування є:

- DES (Data Encryption Standard): Колишній стандарт шифрування, що використовує 56-бітовий ключ. Вважається застарілим і небезпечним.

- AES (Advanced Encryption Standard): Сучасний стандарт шифрування, що підтримує ключі довжиною 128, 192 та 256 біт. Широко використовується через свою безпеку та ефективність[3].

Асиметричне шифрування використовує два різні ключі: один для шифрування (публічний ключ) і інший для дешифрування (приватний ключ). Основні методи асиметричного шифрування включають:

- RSA (Rivest-Shamir-Adleman): Один з найвідоміших асиметричних алгоритмів, який використовується для безпечного обміну ключами та цифрових підписів. Безпека базується на складності факторизації великих чисел.

- ECC (Elliptic Curve Cryptography): Використовує властивості еліптичних кривих для забезпечення високого рівня безпеки при менших розмірах ключів порівняно з RSA. Широко використовується в мобільних та вбудованих пристроях[4].

Гібридні методи шифрування комбінують симетричне та асиметричне шифрування, щоб скористатися перевагами обох. Наприклад, симетричний ключ може бути зашифрований за допомогою асиметричного алгоритму, а самі дані - за допомогою симетричного алгоритму. Це забезпечує ефективність і безпеку.

Хорошим прикладом гібридного шифрування можна назвати: SSL/TLS (Secure Sockets Layer / Transport Layer Security)

SSL та його наступник TLS є найпоширенішими протоколами гібридного шифрування, що забезпечують безпечну передачу даних через Інтернет. Це відбувається у три кроки. Ключовий обмін відбувається за допомогою асиметричного шифрування (наприклад, RSA або Diffie-Hellman), де клієнт і сервер обмінюються ключами. Після обміну ключами генерується симетричний сеансовий ключ (наприклад, використовуючи AES або ChaCha20), який використовується для шифрування основного потоку даних. Сеансовий ключ передається через зашифрований канал, що забезпечує його захист, і використовується для дешифрування даних.

Також до шифрування даних можна віднести хешування. Хешування - це процес перетворення вхідних даних будь-якої довжини в фіксованого розміру вихідне значення (хеш), за допомогою хеш-функції. Хеш-функції використовуються для перевірки цілісності даних, зберігання паролів, створення цифрових підписів і багато іншого. Основні цілі використання хешування це перевірка цілісності даних, зберігання паролів, криптовалютні транзакції та створення цифрових підписів. Найпоширеніший приклад хешування зараз це SHA-2 (Secure Hash Algorithm 2).

SHA-2 — це сімейство криптографічних хеш-функцій, розроблених Національним інститутом стандартів і технологій США (NIST). Воно

включає кілька варіантів, які відрізняються розміром хешу та структурою алгоритму. SHA-2 вважається безпечним і широко використовується у багатьох додатках, включаючи цифрові підписи, сертифікати SSL/TLS, хешування паролів та криптовалюти[5].

1.3 Аналіз існуючих інструментів розробки

Існує безліч інструментів для розробки додатків шифрування інформації, кожен з яких має свої сильні та слабкі сторони. Вибір найкращого інструменту для вас залежатиме від ваших конкретних потреб і вимог. З популярних інструментів для розробки додатків шифрування інформації можна виділити декілька OpenSSL, CryptoAPI, Common Crypto Library.

- OpenSSL — це потужна та універсальна криптографічна бібліотека, яка забезпечує функції шифрування, дешифрування, генерації ключів і сертифікатів, а також інші криптографічні операції. Вона є основним інструментом для забезпечення безпеки в інтернеті, зокрема, для реалізації протоколів SSL (Secure Sockets Layer) і TLS (Transport Layer Security), які використовуються для захисту переданих даних[6].

Основні функції OpenSSL:

Підтримує симетричні та асиметричні алгоритми шифрування (AES, DES, RSA, ECC тощо).

Може генерувати пари ключів для асиметричних алгоритмів, а також ключі для симетричних алгоритмів.

Генерація запитів на сертифікат (CSR), видача сертифікатів, перевірка сертифікатів і робота з сертифікатами SSL/TLS.

Підтримка різних алгоритмів хешування, таких як SHA-256, MD5 тощо.

Генерація і перевірка цифрових підписів для забезпечення цілісності та автентичності даних.

- CryptoAPI - це набір програмних інтерфейсів, що використовуються в операційних системах Windows для роботи з криптографічними операціями. Він надає розробникам інструменти для захисту інформації шляхом використання криптографічних функцій, таких як шифрування, розшифрування, підписання, перевірка підпису та хешування даних. CryptoAPI використовується різними програмами для забезпечення безпеки даних та забезпечення їх конфіденційності, а також для взаємодії з криптографічними ресурсами операційної системи. Розробники можуть використовувати CryptoAPI через набір API функцій, що надаються, для інтеграції криптографічних можливостей в свої програми, що робить їх більш захищеними від кібератак і несанкціонованого доступу до даних[7].

- Common Crypto Library (CCL) - це бібліотека, яка забезпечує програмістам загальні криптографічні функції для забезпечення безпеки даних у програмному забезпеченні. Основними функціями CCL є шифрування, хешування, генерація випадкових чисел, аутентифікація і підписи даних.

Реалізація CCL може відрізнятися в залежності від платформи і мови програмування. Наприклад, в середовищі мови програмування Java CCL може бути представлено як стандартна бібліотека Java Cryptography Architecture (JCA) або бібліотека Bouncy Castle. У мовах програмування C/C++ або Python CCL може відповідати відповідним криптографічним бібліотекам, таким як OpenSSL або PyCrypto.

Загалом, Common Crypto Library є важливим інструментом для забезпечення безпеки даних в програмному забезпеченні, забезпечуючи основні криптографічні операції, які є необхідними для захисту інформації від несанкціонованого доступу і змін.

1.3.1 Мови програмування

Таблиця 1.1 – Порівняльний аналіз мов програмування

Характеристика	Python	Java	C#
Продуктивність	Помірна, поступається іншим	Висока, завдяки JVM	Висока, завдяки .NET
Зручність використання	Проста, підходить для швидкого прототипування	Відносно складна, але об'єктно-орієнтована	Проста для розробників, звиклих до .NET
Наявність бібліотек	Велика кількість, зокрема PyCryptodome	Велика кількість, зокрема JCE	Вбудовані класи для криптографії
Стійкість та безпека	Середня, залежить від використаних бібліотек	Висока, завдяки розвиненій системі безпеки	Висока, завдяки вбудованим засобам безпеки .NET
Кросплатформеність	Висока, завдяки інтерпретатору	Висока, завдяки JVM	Висока, завдяки .NET Core
Обчислювальна потужність	Середня, інтерпретована мова	Висока, компілюється в байт-код	Висока, компілюється в машинний код
Масштабованість	Відносно висока, підходить для скриптів та малих проектів	Висока, підходить для великих корпоративних проектів	Висока, підходить для великих корпоративних проектів
Розширюваність	Висока, за рахунок модулів та пакетів	Висока, за рахунок потужної платформи	Висока, завдяки модульності .NET
Інструменти розробки	Різноманітні, зокрема PyCharm, VSCode	Розвинені, зокрема IntelliJ IDEA, Eclipse	Розвинені, зокрема Visual Studio

1.3.2 Бази даних

Таблиця 1.2 – Порівняльний аналіз СУБД

Характеристика	MySQL	MS SQL Server	PostgreSQL
Ліцензія	Відкритий вихідний код (GPL)	Пропрієтарна, комерційна ліцензія	Відкритий вихідний код (PostgreSQL License)
Інтеграція	Широка підтримка для веб-додатків	Глибока інтеграція з продуктами Microsoft	Добра підтримка для різних платформ
Продуктивність	Висока, особливо для читання	Висока для читання та запису	Висока, особливо для складних запитів
Безпека	Висока, з підтримкою SSL	Дуже висока, з розширеними функціями	Висока, з багатьма параметрами налаштування
Масштабованість	Підходить для середніх проектів	Підходить для великих проектів	Підходить для великих проектів
Резервне копіювання	Базові функції	Розширені можливості для резервного копіювання	Потужні інструменти для резервного копіювання
Транзакції	Підтримка ACID	Підтримка ACID з розширеними можливостями	Підтримка ACID
Цілісність даних	Обмежена підтримка зовнішніх ключів	Розширена підтримка з інструментами контролю	Висока підтримка зовнішніх ключів

1.3.3 Переваги використання .NET

DotNET є відкритою та масштабованою платформою, яка надає розробникам широкі можливості для створення безпечних і надійних програм. Однією з ключових переваг .NET є наявність вбудованих криптографічних бібліотек, які значно спрощують реалізацію різних алгоритмів шифрування та хешування. Наприклад, класи System.Security.Cryptography включають в себе імплементації таких алгоритмів, як AES (Advanced Encryption Standard), RSA (Rivest–Shamir–Adleman) і SHA (Secure Hash Algorithm), що дозволяє забезпечити високий рівень безпеки для обробки конфіденційної інформації. .NET надає можливості для легкої інтеграції з іншими частинами програмного забезпечення, написаними на цій платформі. Це дозволяє зручно використовувати шифрування в різних модулях програми, забезпечуючи єдність і сумісність між компонентами[8].

Мова програмування C# є однією з основних мов для розробки на платформі .NET і має вбудовану підтримку для роботи з криптографічними бібліотеками. Це значно спрощує процес розробки шифрувальних алгоритмів і дозволяє зосередитися на бізнес-логіці програми, не занадто поглиблюючись у низькорівневі деталі криптографії.

Платформа .NET дозволяє легко інтегрувати розроблені додатки з іншими сервісами і системами, що робить їх використання більш гнучким і ефективним у реальних проектах. Це особливо важливо для програм, які потребують інтеграції з веб-сервісами, базами даних або хмарними сервісами для зберігання і обробки шифрованих даних.

Використання платформи .NET для створення додатків з шифруванням інформації на мові C# надає значні переваги у відношенні до безпеки, ефективності розробки та масштабованості. Розглянуті переваги роблять .NET однією з переважних платформ для реалізації захищених і надійних програм, що вимагають обробки конфіденційної інформації[9].

1.4 Огляд та аналіз програмних засобів для захисту інформації

Для успішної розробки програмного забезпечення для захисту інформації необхідно ретельно вивчити існуючі аналогічні рішення. Це дозволить детально визначити їхній функціонал, переваги та недоліки, що сприятиме створенню ефективного та конкурентоспроможного продукту. Аналіз аналогічних програмних рішень допоможе виявити найкращі практики, які варто впровадити у новий застосунок, а також виявити недоліки, яких слід уникнути. Це забезпечить високу якість та надійність майбутнього застосунку, що відповідатиме потребам сучасних установ. Вивчення ринку дозволить створити програмне забезпечення, яке буде не лише функціональним, але й відповідатиме найвищим стандартам безпеки.

VeraCrypt є потужним програмним засобом для шифрування даних, призначеним для захисту конфіденційної інформації шляхом створення зашифрованих віртуальних дисків (рис. 1.1). Основне призначення VeraCrypt полягає у забезпеченні високого рівня безпеки даних користувачів шляхом шифрування всього тому або окремих файлів та папок. Програма також дозволяє шифрувати системні диски та розділи, забезпечуючи додатковий рівень захисту для операційних систем. VeraCrypt підтримує різні алгоритми шифрування, що підвищує його надійність та стійкість до атак.

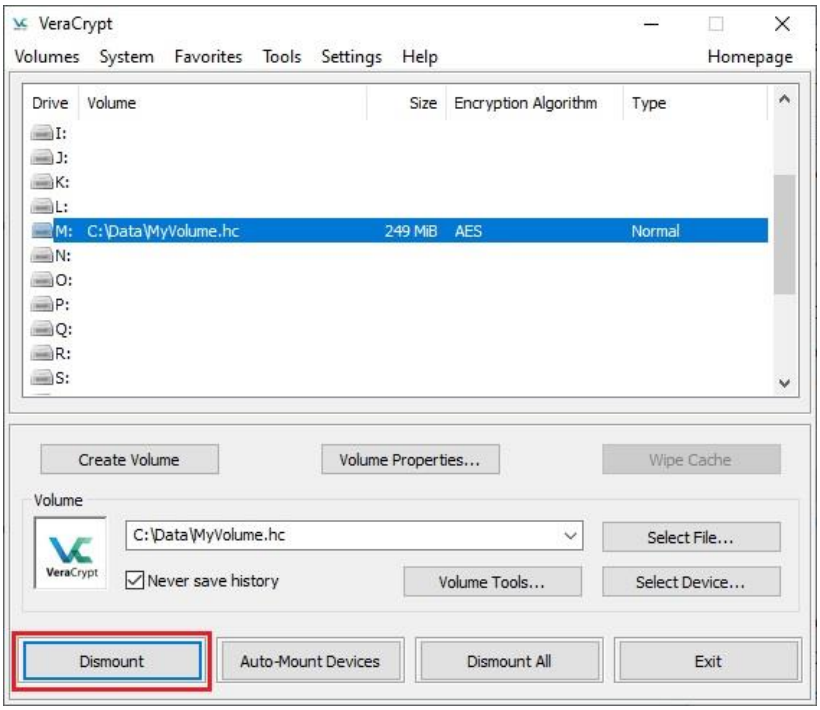


Рисунок 1.1 – Приклад інтерфейсу системи «VeraCrypt»

Архітектура VeraCrypt базується на використанні складних криптографічних алгоритмів, таких як AES, Twofish і Serpent, що забезпечує стійкість до сучасних методів криптоаналізу. Програмний засіб дозволяє створювати зашифровані контейнери, які можуть бути змонтовані як віртуальні диски. Користувач може зберігати дані у цих контейнерах, і вони будуть автоматично шифруватися у режимі реального часу. VeraCrypt також підтримує шифрування системних дисків і розділів, що забезпечує додатковий рівень захисту для операційних систем. Завдяки підтримці різних операційних систем і багатоплатформенності, VeraCrypt є гнучким і універсальним інструментом для захисту даних.

Переваги VeraCrypt складаються із:

– високий рівень безпеки. Використання потужних алгоритмів шифрування, таких як AES, Twofish, і Serpent;

– багатоплатформенність. Підтримка різних операційних систем, включаючи Windows, macOS і Linux;

– безкоштовність і відкритий вихідний код. Користувачі можуть безкоштовно використовувати програму та перевіряти її на наявність вразливостей;

– шифрування в режимі реального часу. Автоматичне шифрування даних під час їх використання без значного впливу на продуктивність системи.

Недоліки VeraCrypt:

– складність використання. Потребує певних технічних знань для налаштування та управління;

– обмежена підтримка. Можливі труднощі з отриманням офіційної підтримки та документації;

– можливі проблеми сумісності. Не всі програми та системи можуть коректно працювати з зашифрованими томами;

– ризик втрати даних. Втрата паролю або ключа шифрування призводить до неможливості відновлення даних [10].

Отже, VeraCrypt є потужним і надійним інструментом для шифрування даних, який забезпечує високий рівень безпеки завдяки використанню потужних криптографічних алгоритмів. Він підходить для багатьох операційних систем і є безкоштовним з відкритим вихідним кодом. Однак VeraCrypt вимагає технічних знань для налаштування і може мати проблеми з сумісністю та підтримкою. Попри ці недоліки, VeraCrypt залишається одним з найефективніших рішень для захисту конфіденційної інформації.

BitLocker – це програмний засіб для шифрування дисків, інтегрований в операційну систему Windows, призначений для захисту конфіденційної інформації. Основне призначення BitLocker полягає у забезпеченні безпеки даних шляхом шифрування всього жорсткого диска або окремих розділів, що дозволяє захистити інформацію від несанкціонованого доступу у разі втрати чи крадіжки пристрою. BitLocker також підтримує шифрування знімних носіїв, таких як USB-флешки, що розширює можливості захисту даних. Завдяки автоматичному шифруванню та інтеграції з TPM, BitLocker забезпечує додатковий рівень безпеки та зручність у використанні.

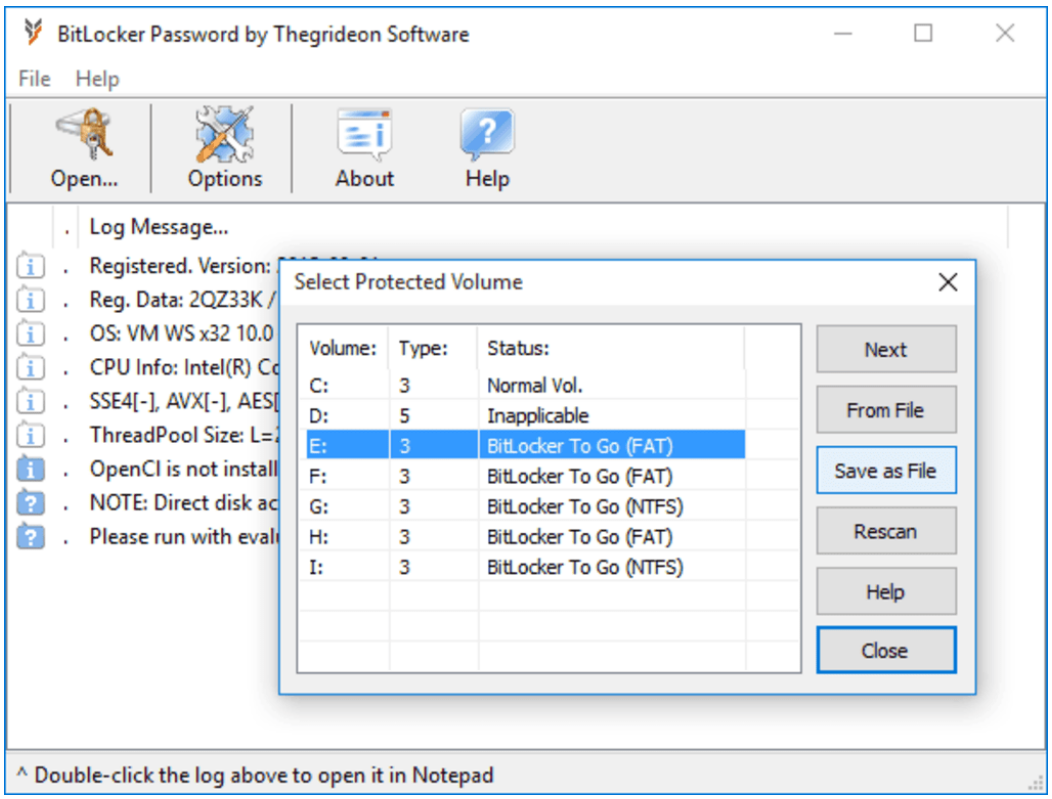


Рисунок 1.2 – Приклад інтерфейсу системи «BitLocker»

Архітектура BitLocker включає використання TPM (Trusted Platform Module) для зберігання ключів шифрування та забезпечення безпечного завантаження системи. BitLocker також підтримує функції автоматичного шифрування та централізованого управління через Active Directory, що полегшує адміністрування в корпоративних середовищах.

Переваги BitLocker:

– інтеграція з Windows. Легко використовувати і налаштовувати в середовищах Windows без необхідності додаткового програмного забезпечення;

– підтримка TPM. Забезпечує додатковий рівень безпеки шляхом зберігання ключів шифрування у Trusted Platform Module;

– централізоване управління. Підтримує централізоване адміністрування через Active Directory, що зручно для корпоративних користувачів;

– автоматичне шифрування. Простота в налаштуванні і використанні, забезпечує автоматичне шифрування нових файлів.

Недоліки BitLocker:

– обмежена платформа. Доступний тільки для операційних систем Windows, що обмежує його використання на інших платформах;

– потреба в TPM. Для максимальної безпеки потрібне апаратне забезпечення TPM, яке може бути відсутнім у деяких пристроях;

– відсутність відкритого коду. Користувачі не можуть перевірити код на наявність потенційних вразливостей;

– втрата ключа. У разі втрати ключа шифрування відновлення доступу до даних може бути неможливим [11].

Отже, BitLocker є потужним інструментом для шифрування дисків, інтегрованим в операційну систему Windows, що забезпечує високий рівень захисту конфіденційної інформації. Його переваги включають простоту використання, підтримку TPM, централізоване управління через Active Directory та автоматичне шифрування даних. Однак, BitLocker обмежений лише платформою Windows, потребує TPM для максимальної безпеки, не має відкритого коду і може стати проблематичним у разі втрати ключа шифрування. Незважаючи на недоліки, BitLocker є ефективним рішенням для захисту даних у корпоративних середовищах Windows.

Cryptomator – це програмний засіб для шифрування файлів, спеціально розроблений для захисту даних, що зберігаються в хмарних сервісах (рис. 1.6). Його основне призначення полягає в тому, щоб забезпечити конфіденційність файлів під час зберігання в хмарних сховищах, таких як Dropbox, Google Drive або OneDrive. Архітектура Cryptomator передбачає створення зашифрованих віртуальних дисків, в яких файли шифруються локально перед завантаженням у хмару. Програмний засіб використовує AES-256 для шифрування файлів, забезпечуючи високий рівень захисту без необхідності спеціальних знань або додаткового апаратного забезпечення. Весь процес шифрування та розшифрування відбувається прозоро для користувача, що робить Cryptomator зручним і ефективним інструментом для захисту конфіденційної інформації.

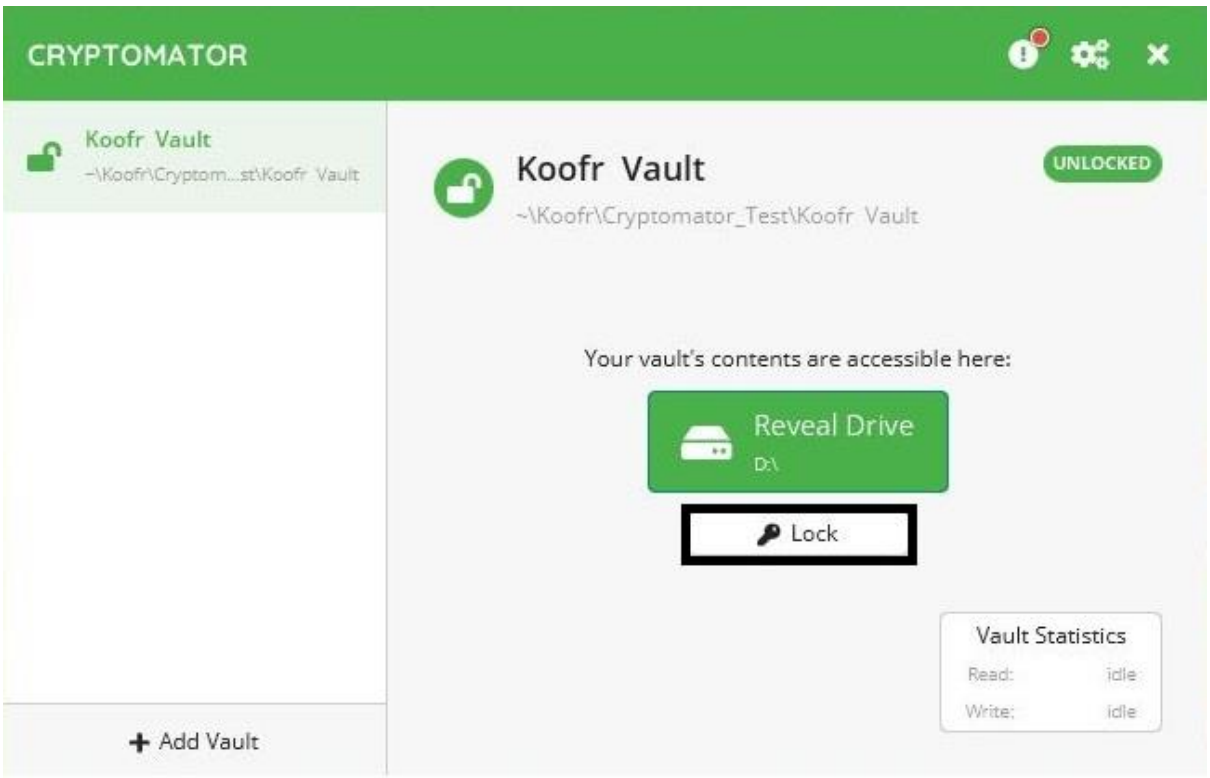


Рисунок 1.3 – Приклад інтерфейсу системи «Cryptomator»

Переваги Cryptomator:

– простота використання. Легкий інтерфейс, що дозволяє швидко налаштувати шифрування файлів без спеціальних знань;

– відкритий вихідний код. Забезпечує прозорість і можливість перевірки на наявність вразливостей;

– підтримка багатьох хмарних сервісів. Інтеграція з популярними хмарними сховищами, такими як Dropbox, Google Drive і OneDrive;

– локальне шифрування. Файли шифруються локально перед завантаженням у хмару, що підвищує рівень безпеки.

Недоліки Cryptomator:

– обмежена функціональність. Відсутність деяких розширених функцій, які пропонують інші платні рішення;

– швидкість шифрування. Процес шифрування може бути повільнішим порівняно з деякими іншими програмними засобами;

– обмежена підтримка платформ. Немає офіційної підтримки для деяких менш популярних операційних систем;

– відсутність централізованого управління. Немає можливості централізованого управління для корпоративних користувачів, що може бути незручним для великих організацій [12].

Отже, Cryptomator є простим і ефективним інструментом для шифрування файлів, призначеним для захисту даних, що зберігаються в хмарних сервісах. Він забезпечує високу конфіденційність завдяки локальному шифруванню файлів перед їх завантаженням у хмару та використовує відкритий вихідний код для забезпечення прозорості. Проте, Cryptomator має обмежену функціональність, може бути повільнішим у шифруванні порівняно з іншими рішеннями та не підтримує централізоване управління для корпоративних користувачів. Незважаючи на ці недоліки, Cryptomator залишається популярним вибором для індивідуальних користувачів, які шукають надійний захист своїх даних у хмарі.

Виходячи з аналізу програмних засобів VeraCrypt, BitLocker та Cryptomator, розробка схожої системи для захисту інформації є доцільною. Існуючі рішення мають свої переваги та недоліки, а новий застосунок може запропонувати вдосконалені алгоритми шифрування та кращу підтримку для різних потреб, що підвищить загальний рівень безпеки та зручності використання.

РОЗДІЛ 2
Проєктування та розробка програмного засобу «CrypterAES»

2.1 Постановка задачі, призначення та вимоги до програмного засобу «CrypterAES»

Розглядаючи сучасні системи захисту інформації, стає зрозуміло, що криптографічний захист відіграє ключову роль. Аналіз існуючих методів захисту вказує на необхідність їх постійного вдосконалення. Особливу увагу привертає метод шифрування AES, який широко використовується у всьому світі, але має свої вразливості, такі як бічні атаки, висока обчислювальна складність та інші.

Метою даного дослідження є підвищення ефективності та безпеки методу шифрування AES. Для досягнення цієї мети передбачено виконання таких завдань:

– розробка нового методу вдосконалення алгоритму AES: Цей метод спрямований на зменшення вразливостей та підвищення загальної ефективності алгоритму шифрування;

– створення програмного забезпечення для реалізації нового методу: Програмне забезпечення забезпечить практичне впровадження та використання вдосконаленого алгоритму AES;

– проведення експериментальної перевірки нового підходу: Мета цієї перевірки – оцінити ефективність та безпеку розробленого методу на практиці.

2.2 Вибір моделі розробки програмного засобу «CrypterAES»

Основна мета системи полягає у забезпеченні надійного шифрування та дешифрування даних з використанням алгоритму AES, що є одним з найбільш популярних і безпечних методів шифрування. У цьому підрозділі розглянуто ключові аспекти реалізації програмних модулів, які складають основу системи «CrypterAES».

Для ефективної взаємодії між програмним забезпеченням проекту та базою даних використовується простір імен System.Data.SqlClient. Ця бібліотека відіграє ключову роль, оскільки забезпечує ряд важливих функцій: вона дозволяє розробникам формулювати SQL-запити, виконувати їх та управляти транзакціями, що є невід'ємною частиною процесу внесення змін у базу даних. Використання цих класів значно спрощує процес взаємодії, роблячи його більш інтуїтивно зрозумілим та ефективним для програмістів.

У файлі конфігурації «App.config» визначено спеціальну змінну підключення «CONNECT», яка містить всі необхідні параметри для ініціалізації підключення до бази даних. Ці параметри детально представлені та описані на рис. 2.1, де кожен компонент підключення вказано з описом їх призначення та способу використання.

```
<!-- Підключення до бази даних -->
<appSettings>
  <add key="CONNECT" value="Data Source=(LocalDB)\MSSQLLocalDB;
       AttachDbFilename=|DataDirectory|\DB.mdf;
       Integrated Security=True" />
</appSettings>
```

Рисунок 2.1 – Реалізація підключення до бази даних

У контексті користувацького інтерфейсу проекту, ключовим елементом є компонент menuStrip, який інтегровано в головне вікно додатку. Цей інструмент інтерфейсу слугує засобом для навігації користувачів та забезпечує легкий доступ до різних функціональних можливостей програми. Він розроблений таким чином, щоб забезпечити інтуїтивно зрозуміле керування, дозволяючи користувачам ефективно переходити між різними розділами програми. Візуальне представлення menuStrip та його компонентів можна переглянути на рис. 2.2, який демонструє як сам елемент, так і його розташування в інтерфейсі користувача.

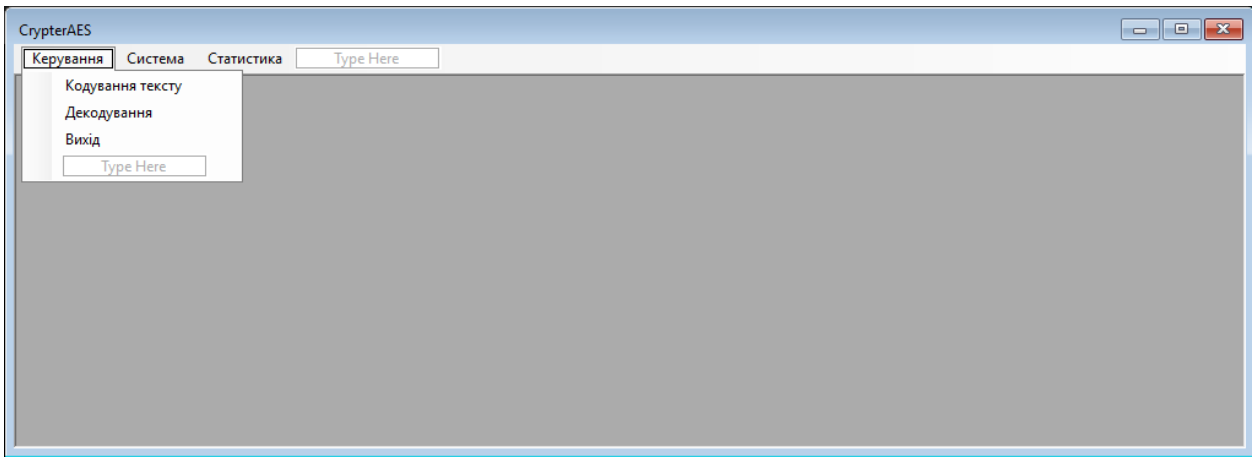


Рисунок 2.2 – Головне вікно програми

В рамках розробки програмного забезпечення, яке включає інтерфейс користувача з меню, особливу увагу необхідно зосередити на розробці логіки взаємодії з формами. Ключовим аспектом є ініціація відповідних форм для кожного елемента меню. У коді програми кожен пункт меню асоціюється з окремою формою, яка активізується, коли користувач обирає цей пункт. Ця активація форми відбувається завдяки тому, що динамічно створений екземпляр форми отримує статус «активний», що дозволяє програмі відобразити відповідне вікно.

Особлива увага в процесі розробки приділяється механізмам управління життєвим циклом активних форм. У програмній реалізації впроваджено логіку, що забезпечує автоматичне закриття поточного активного вікна форми перед тим, як ініціюється нова форма. Такий підхід допомагає запобігти можливим конфліктам, пов'язаним з перекриттям вікон, а також іншим проблемам, що можуть виникати внаслідок одночасного відображення кількох вікон. Ця стратегія є уніфікованою та послідовно застосовується до всіх елементів меню, що забезпечує узгодженість та надійність взаємодії користувача з програмою.

Концепція управління життєвим циклом форм візуалізована на рис. 2.3, де демонструється відповідний фрагмент коду. Цей рисунок ілюструє, як саме програма обробляє переходи між різними вікнами, забезпечуючи гладку та зрозумілу навігацію для кінцевого користувача.

```
1 reference
private void кодуванняТекстуToolStripMenuItem_Click(object sender, EventArgs e) {
    CloseAllWindows();
    CodingForm codingForm = new CodingForm();
    codingForm.MdiParent = this;
    codingForm.WindowState = FormWindowState.Maximized;
    codingForm.Show();
}
```

Рисунок 2.3 – Код для виклику дочірніх форм

Після завершення розробки користувацького інтерфейсу увага команди розробників зосередилася на створенні класів для взаємодії з базою даних. Ці класи є важливими компонентами програмного забезпечення, оскільки вони надають необхідні інструменти для маніпуляції даними — включно з вставкою, оновленням, видаленням та іншими операціями. Ці дії забезпечують управління інформацією всередині бази даних і є ключовими для роботи програмного продукту.

Серед розроблених методів, наприклад, метод «InsertCodings» призначений для додавання записів про зашифровані тексти до бази даних. Цей метод, як і інші аналогічні методи, спроектовані таким чином, щоб оптимізувати взаємодію з базою і забезпечити ефективне та безпечне зберігання даних. Процеси, які використовуються в цих методах, мають складну логіку, що детально відображено на рис. 2.4.

```
public void InsertCodings(string CodingsName, int UsersId,
  byte[] GenerateKey, byte[] CodingText) {
  string SqlString = "INSERT INTO Codings (CodingsName, UsersId," +
    " GenerateKey, CodingText)" +
    " VALUES (@CodingsName, @UsersId, @GenerateKey, @CodingText)";
  using (SqlConnection conn = new SqlConnection(_ConnString)) {
    using (SqlCommand cmd = new SqlCommand(SqlString, conn)) {
      cmd.CommandType = CommandType.Text;
      cmd.Parameters.AddWithValue("@CodingsName", CodingsName);
      cmd.Parameters.AddWithValue("@UsersId", UsersId);
      cmd.Parameters.AddWithValue("@GenerateKey", GenerateKey ?? Encoding.Default.GetBytes(""));
      cmd.Parameters.AddWithValue("@CodingText", CodingText ?? Encoding.Default.GetBytes(""));

      conn.Open();
      cmd.ExecuteNonQuery();
    }
  }
}
```

Рисунок 2.4 – Код методу «InsertCodings»

Метод InsertCodings призначений для вставки запису в таблицю Codings бази даних. Цей метод приймає чотири параметри: CodingsName (назва кодування), UsersId (ідентифікатор користувача), GenerateKey

(згенерований ключ у вигляді масиву байтів) і CodingText (зашифрований текст у вигляді масиву байтів). Спочатку створюється рядок SQL-запиту, який визначає команду вставки даних в таблицю Codings з використанням параметризованого запиту для уникнення SQL-ін'єкцій.

Після цього створюється нове з'єднання з базою даних за допомогою класу SqlConnection, яке використовує рядок підключення _ConnString. З'єднання обгорнуте в блок using, що забезпечує автоматичне закриття з'єднання після завершення операцій. Далі створюється об'єкт SqlCommand, який приймає SQL-запит і об'єкт з'єднання conn. Команда налаштовується на виконання текстового запиту шляхом встановлення властивості CommandType в значення CommandType.Text. Після цього до команди додаються параметри, використовуючи метод AddWithValue, який зв'язує кожен параметр SQL-запиту з відповідним значенням змінної методу. Параметри GenerateKey і CodingText перевіряються на null, і в разі потреби замінюються на порожній масив байтів.

Після підготовки команди з'єднання відкривається методом Open, і виконується команда ExecuteNonQuery, яка виконує SQL-запит без повернення будь-яких результатів. Це забезпечує вставку нового запису в таблицю Codings з відповідними даними. Після виконання операції з'єднання і команда автоматично закриваються завдяки використанню блоку using, забезпечуючи належне вивільнення ресурсів.

Також реалізовано метод GetAllCodings призначений для отримання списку всіх кодувань, пов'язаних з певним користувачем, і приймає як параметр currentUserId, що є ідентифікатором поточного користувача (рис. 2.5). Спочатку створюється рядок SQL-запиту, який вибирає поля CodingsId та CodingsName з таблиці Codings, де UsersId дорівнює заданому ідентифікатору користувача. Результати впорядковуються за CodingsName у зростаючому порядку.

```
public List<Codings> GetAllCodings(int currentUserId) {
    int i = 0;
    string SqlString = "SELECT CodingsId, CodingsName FROM Codings WHERE " +
      "UsersId = @UsersId ORDER BY CodingsName ASC";
    List<Codings> listAllCodings = new List<Codings>();
    using (SqlConnection conn = new SqlConnection(_ConnString)) {
      using (SqlCommand cmd = new SqlCommand(SqlString, conn)) {
        cmd.Parameters.AddWithValue("@UsersId", currentUserId);
        conn.Open();
        using (SqlDataReader reader = cmd.ExecuteReader()) {
          while (reader.Read()) {
            Codings oneCodings = new Codings();
            oneCodings.Number = ++i;
            oneCodings.CodingsId = Convert.ToInt32(reader["CodingsId"]);
            oneCodings.CodingsName = reader["CodingsName"].ToString();
            listAllCodings.Add(oneCodings);
          }
        }
      }
    }
    if (listAllCodings.Count == 0) {
      Codings noCodings = new Codings();
      noCodings.CodingsId = 0;
      noCodings.Message = NamesMy.NoDataNames.NoDataInCodings;
      listAllCodings.Add(noCodings);
    }
    return listAllCodings;
}
```

Рисунок 2.5 – Код методу «GetAllCodings»

Ініціалізується змінна i, яка використовується для нумерації записів, а також створюється список listAllCodings, який буде зберігати об'єкти Codings. Далі встановлюється з'єднання з базою даних за допомогою SqlConnection, використовуючи рядок підключення _ConnString, і це з'єднання обгорнуте в блок using, що забезпечує його автоматичне закриття після виконання операцій.

Створюється об'єкт SqlCommand, який приймає SQL-запит та об'єкт з'єднання. Параметр @UsersId прив'язується до значення currentUserId за допомогою методу AddWithValue. Після цього з'єднання відкривається методом Open, і виконується команда ExecuteReader, яка повертає об'єкт SqlDataReader для зчитування результатів запиту.

В циклі while об'єкт SqlDataReader читає кожен рядок результату. Для кожного рядка створюється новий об'єкт Codings, і встановлюються його властивості: Number (нумерація записів), CodingsId (ідентифікатор кодування, перетворений в ціле число), і CodingsName (назва кодування, перетворена в рядок). Цей об'єкт додається до списку listAllCodings. Після

завершення зчитування, якщо список listAllCodings порожній (тобто, не було знайдено жодного запису), створюється об'єкт Codings з CodingsId рівним нулю і повідомленням Message, що інформує про відсутність даних. Цей об'єкт також додається до списку. У кінці, метод повертає список listAllCodings, який містить всі знайдені записи або повідомлення про відсутність даних.

Після завершення фази проектування та розробки класів, призначених для обробки і керування даними, розробники перейшли до створення користувацького інтерфейсу. Цей етап є критично важливим, оскільки користувацький інтерфейс служить основним мостом між користувачем та функціоналом системи. Важливою частиною інтерфейсу є спеціалізована форма для шифрування текстової інформації, яка візуалізується на рис.2.6.

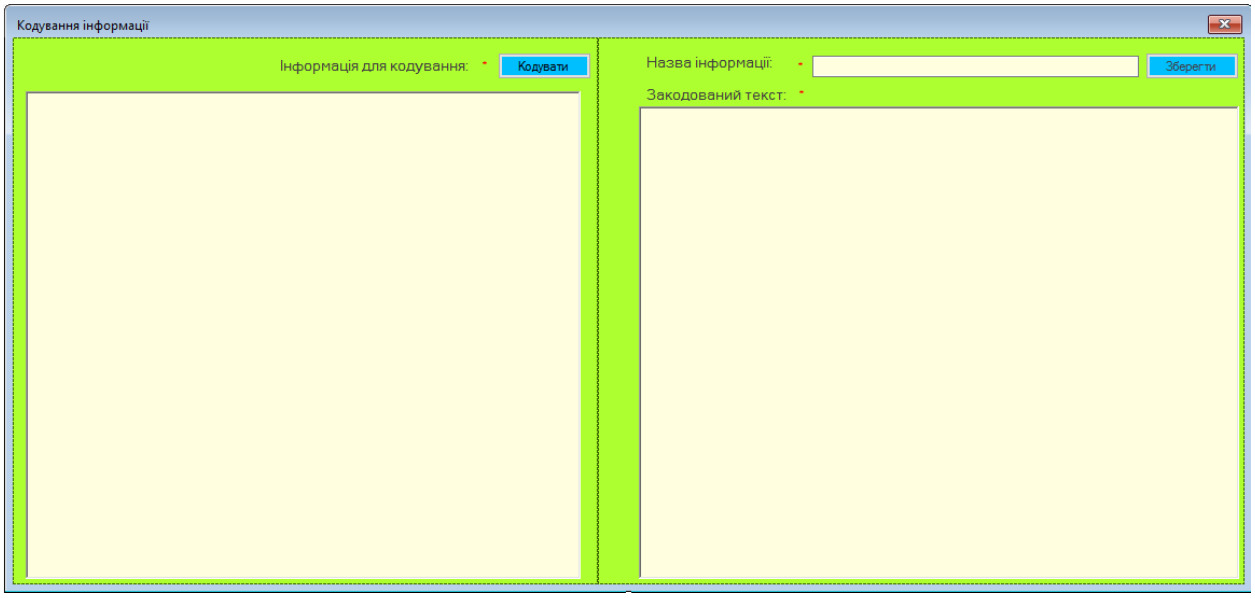


Рисунок 2.6 – Інтерфейс форми для шифрування інформації

Ця форма не тільки забезпечує візуальні засоби для введення і шифрування тексту, але й є інтегрованим елементом системи, тісно пов'язаним з розробленими раніше класами на рівні даних. У її складі присутні поля для введення тексту, кнопки та інші елементи управління, які дозволяють користувачу вводити текст, вибирати ключі для шифрування, запускати процес шифрування і при потребі зберігати зашифровані дані у базі даних.

Для шифровано тексту та виведення його на екран розроблено метод CodingBtn_Click, який спрацьовує про події натискання кнопки, яка ініціює процес шифрування введеного тексту (рис. 2.7).

```
private void CodingBtn_Click(object sender, EventArgs e) {
  if (IsTextCodingCorrect()) {
    RandomNumberGenerator rng = RandomNumberGenerator.Create();
    rng.GetBytes(_Key);
    _EncryptedData = _InfoCrypter.EncryptData(TextForCodingRTBox.Text, _Key);
    CodingTextRTBox.Text = Encoding.UTF8.GetString(_EncryptedData);
  }
}
```

Рисунок 2.7 –Кодування інформації

На початку метод перевіряє, чи введений текст для шифрування є коректним, викликаючи функцію IsTextCodingCorrect. Якщо функція повертає значення true, тобто введений текст відповідає всім необхідним критеріям, відбувається наступна послідовність дій. Створюється об'єкт генератора випадкових чисел RandomNumberGenerator через виклик статичного методу Create(). Цей генератор використовується для створення криптографічно стійких випадкових чисел. Викликається метод GetBytes генератора випадкових чисел, який заповнює масив _Key випадковими байтами. Цей масив буде використовуватися як ключ для шифрування. Далі викликається метод EncryptData об'єкта _InfoCrypter, який здійснює шифрування тексту, введеного у текстове поле TextForCodingRTBox. Введений текст і згенерований ключ передаються як параметри. Результатом шифрування є масив байтів _EncryptedData.

Після цього зашифровані дані перетворюються у рядок за допомогою кодування UTF-8 і результат присвоюється текстовому полю CodingTextRTBox, яке відображає зашифрований текст. Таким чином, користувач бачить результат шифрування на екрані у текстовому полі після натискання кнопки.

У контексті забезпечення інженерної надійності системи, для кожної з розроблених форм користувацького інтерфейсу було визначено процедуру валідації даних. Ця валідація реалізована через спеціалізовані методи під

назвою IsDataEnteringCorrect, кожен з яких розроблений окремо для кожної форми (рис. 2.8).

```
private bool IsDataEnteringCorrect() {
  bool isCorrect = true;
  if (_validation.IsDataEntering(CodingsNameTBox.Text)) {
    CodingsNameValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
  } else {
    CodingsNameValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
    isCorrect = false;
  }
  if (_validation.IsDataEntering(CodingTextRTBox.Text)) {
    CodingTextValiadtionLbl.Text = NamesMy.ProgramButtons.RequiredValidation;
  } else {
    CodingTextValiadtionLbl.Text = NamesMy.ProgramButtons.ErrorValidation;
    isCorrect = false;
  }
  return isCorrect;
}
```

Рисунок 2.8 – Методу перевірки введення даних

Метод IsDataEnteringCorrect використовується для перевірки коректності даних, введених користувачем у форму. Цей метод визначає, чи відповідають введені дані у формі встановленим критеріям валідації. Метод починає свою роботу з ініціалізації змінної isCorrect як true, що означає, що дані вважаються коректними до проведення перевірки.

Перевірка валідації починається з поля для введення назви кодування (CodingsNameTBox). Метод використовує заздалегідь визначений об'єкт валідації _validation, для перевірки тексту, введеного у це поле через метод IsDataEntering. Якщо текст задовольняє вимоги валідації, текст підказки (CodingsNameValiadtionLbl) встановлюється як повідомлення про необхідність заповнення поля. У разі, якщо текст не проходить перевірку, текст підказки змінюється на повідомлення про помилку, а змінна isCorrect встановлюється в false, що вказує на наявність помилки у введених даних.

Той самий процес валідації застосовується до поля для введення зашифрованого тексту (CodingTextRTBox). Спочатку перевіряється введений текст: якщо він проходить валідацію, підпис поля валідації (CodingTextValiadtionLbl) вказує на необхідність заповнення. В іншому випадку, встановлюється повідомлення про помилку, і змінна isCorrect також змінюється на false.

На завершення метод повертає булеве значення isCorrect, яке показує, чи були всі поля форми заповнені правильно. Якщо будь-яке з полів не задовольняє вимогам валідації, метод повертає false, в іншому випадку — true. Це дозволяє системі визначити, чи можна безпечно продовжувати обробку даних або ж необхідно спершу виправити помилки вводу.

2.3 Загальний опис проєкту «CrypterAES»

Математичне моделювання методу шифрування AES є ключовим компонентом в процесі дослідження та оптимізації криптографічних алгоритмів, які використовуються у системах. Ретельне моделювання не тільки дозволяє глибше зрозуміти структуру та принципи роботи алгоритму, але й відкриває можливість оцінити його ефективність та визначити потенційні слабкі місця, які можуть бути вразливими до атак. Завдяки математичному моделюванню можна формалізувати процеси шифрування і дешифрування, що сприяє більш точному виявленню векторів атак та підвищенню загальної безпеки системи.

Процес шифрування у моделі AES можна розглядати як послідовність перетворень, кожне з яких виконується над даними, представленими у формі матриці розміром 4×4. Кожен блок даних, P_i, при шифруванні трансформується згідно з визначеними правилами AES, що включають перестановки та заміни. Структура кожного блоку P_i представляє собою матрицю з елементами p_{ij}, де і відповідає номеру рядка, а j – номеру стовпця у матриці. Наприклад, матриця для одного блоку P_i може бути представлена таким чином:

$$P_i = \begin{pmatrix} p_{11}p_{12}p_{13}p_{14} \\ p_{21}p_{22}p_{23}p_{24} \\ p_{31}p_{32}p_{33}p_{34} \\ p_{41}p_{42}p_{43}p_{44} \end{pmatrix} \quad (2.1)$$

Ця матриця є основою для проведення різних криптографічних операцій, таких як додавання ключа, заміна байтів, зсув рядків та перемішування стовпців, кожна з яких є частиною загального циклу шифрування AES.

Ключ шифрування в алгоритмі AES, позначений як K, є основним елементом, який визначає унікальність процесу шифрування. Для цього ключ шифрування також представляється у вигляді матриці розміром 4×4, що дозволяє інтегрувати його безпосередньо в алгоритм шифрування. Структура матриці ключа має наступний вигляд:

$$K = \begin{pmatrix} k_{11}k_{12}k_{13}k_{14} \\ k_{21}k_{22}k_{23}k_{24} \\ k_{31}k_{32}k_{33}k_{34} \\ k_{41}k_{42}k_{43}k_{44} \end{pmatrix} \quad (2.2)$$

Процес шифрування в AES включає кілька послідовних етапів, кожен з яких спрямований на забезпечення конфіденційності інформації:

1. Перетворення *SubBytes*. Перший етап, знаний як *SubBytes*, полягає у перетворенні кожного байту блоку P_i за допомогою спеціалізованої таблиці заміни, відомої як *S-box*. Ця таблиця виконує заміну кожного байту на інший за певним правилом, що значно збільшує стійкість алгоритму до атак.

$$SubBytes(P_i) = [S(p_{ij})] \quad (2.3)$$

2. Перетворення *ShiftRows*. Другий крок, *ShiftRows*, здійснює циклічний зсув рядків у матриці блоку P_i. Зсув в кожному рядку залежить від номера рядка, що сприяє додатковому перемішуванню байтів і підвищує складність розшифрування без відомого ключа.

$$SShiftRows(P_i) = [p_{i(j+i)}] \quad (2.4)$$

2. Перетворення *MixColumns*. Третій етап, *MixColumns*, включає перемішування стовпців матриці P_i за допомогою лінійного перетворення. Цей процес здійснюється множенням кожного стовпця на фіксовану матрицю M, що забезпечує додаткову дифузію в межах блоку.

$$MixColumns(P_i) = [M \times p_{ij}] \quad (2.5)$$

4. Додавання ключа для шифрування. На цьому етапі до кожного блоку даних P_i додається ключ шифрування K. Ця операція виконується шляхом побітового виключаючого або XOR між кожним байтом блоку даних та відповідним байтом ключа шифрування. Процес можна математично виразити як:

$$AddRoundKey(P_iK) = [p_{ij} \oplus k_{ij}], \quad (2.6)$$

де $\oplus$ позначає операцію XOR. Цей метод є фундаментальним для забезпечення безпеки, оскільки він інтегрує ключ шифрування безпосередньо у процес шифрування, роблячи алгоритм стійким до атак.

Сумарний процес шифрування кожного блоку P_i можна описати послідовністю перетворень, що включають перераховані вище етапи. Весь процес шифрування виглядає наступним чином:

$$C_i = AddRoundKey \quad (2.7)$$

На кожному етапі дані трансформуються, забезпечуючи високий рівень захисту та складність шифру. Ці послідовні трансформації відбуваються через декілька раундів — зазвичай 10, 12 або 14, залежно від довжини ключа шифрування. Кількість раундів збільшується зі зростанням довжини ключа, що веде до вищої стійкості шифру проти спроб розшифрування. Завершення цих раундів призводить до формування кінцевого зашифрованого блоку C_i, який містить дані, захищені від несанкціонованого доступу.

Процес дешифрування в алгоритмі AES можна вважати інверсією процесу шифрування. Він заснований на використанні набору операторів, які функціонально зворотні до тих, що використовуються під час шифрування. Цей процес є критично важливим для відновлення оригінального тексту з шифрованої інформації без втрати даних.

На початку процесу дешифрування, нехай C_i визначає зашифрований блок даних, а K – це ключ дешифрування. Обидва ці параметри представлені у форматі матриць розміром 4×4, що дозволяє їм взаємодіяти безпосередньо під час процесу дешифрування.

Процес дешифрування складається з кількох ключових етапів, кожен з яких відновлює операції, здійснені під час шифрування, але у зворотному порядку:

1. Зворотнє додавання ключа (*InvAddRoundKey*): Перший крок дешифрування повторює зворотню дію останнього кроку шифрування, де ключ K був доданий до блоку даних. Ця операція знову використовує *XOR* для видалення впливу ключа з зашифрованих даних, тобто:

$$InvAddRoundKey(C_i, K) = [c| \quad |ik \oplus k_{ij}] \tag{2.8}$$

Це відновлює стан даних перед останнім застосуванням ключа шифрування.

2. Зворотнє перемішування стовпців (*InvMixColumns*): Наступний крок відновлює дані, оброблені під час *MixColumns*. Замість застосування матриці перетворення M, використовується обернена матрицяM^{-1}. Цей процес здійснюється через множення оберненої матриці на стовпці зашифрованого блоку:

$$InvMixColumns(C_i) = [M^{-1} \times c_{ij}] \tag{2.9}$$

2. Третій крок у процесі дешифрування AES, званий *InvShiftRows*, є повною інверсією етапу *ShiftRows*, який виконується під час шифрування. Цей процес полягає в зворотному циклічному зсуві рядків у зашифрованому блоку. Конкретно, зсув здійснюється вліво, на відміну від правого зсуву, виконаного під час шифрування. Це гарантує, що кожен рядок блоку повертається до його оригінального порядку, який існував перед застосуванням *ShiftRows*:

$$InvShiftRows(\mathrm{C}_i) = [c_{i(j-i)}] \quad (2.10)$$

4. Четвертий крок, *InvSubBytes*, є протилежністю кроку *SubBytes*. На цьому етапі кожен байт зашифрованого блоку замінюється з використанням інверсної таблиці замін, відомої як S^{-1}, або інверсна *S-Box*. Це перетворення дозволяє відновити оригінальні значення байтів, які були змінені на етапі *SubBytes* під час шифрування:

$$InvSubBytes(C_i) = [S^{-1}(c_{ij})] \quad (2.11)$$

Завершальний процес дешифрування для блоку C_iвключає виконання згаданих вище інверсних операцій у зворотньому порядку до того, як вони були застосовані під час шифрування. Це дозволяє послідовно відновити оригінальний незашифрований блок даних. Математично, цей процес можна представити наступним чином:

$$P_i = InvSubBytes$$
$$\big(InvMixColumns\big(InvAddRoundKey(C_i, K)\big)\big) \quad (2.12)$$

Ця послідовність кроків повторюється для кожного раунду в кількості, яка залежить від довжини ключа шифрування, і може становити 10, 12 або 14 раундів. Така послідовність раундів забезпечує високий рівень безпеки та ефективність дешифрування. Використання цієї математичної моделі є фундаментальним для забезпечення глибокого розуміння внутрішньої структури алгоритму AES та методів його інверсії, гарантуючи здатність надійно відновлювати оригінальні дані з зашифрованої інформації.

2.4 Обгрунтування вибору інструментальних засобів розробки «CrypterAES»

При розробці програмного засобу «CrypterAES» важливо обрати відповідні технології, які забезпечать високу продуктивність, надійність та безпеку системи. Це завдання є ключовим, оскільки від правильного вибору технологій залежить ефективність роботи програмного забезпечення, його стійкість до зовнішніх загроз та можливість масштабування в майбутньому.

Тому особливу увагу було приділено аналізу мов програмування, які можуть найкраще відповідати вимогам проекту.

Враховуючи специфіку задачі, було детально проаналізовано декілька популярних мов програмування, зокрема Python, Java та C#. Кожна з цих мов має свої особливості, що можуть значно вплинути на кінцевий результат розробки. Аналіз включав оцінку продуктивності, зручності використання, наявності бібліотек для роботи з криптографією, а також підтримку спільноти розробників.

Python привертає увагу своєю простотою та читабельністю коду, що значно полегшує процес розробки та тестування [13]. Ця мова відома своєю багатою екосистемою бібліотек, зокрема для криптографії, таких як PyCryptodome. Однак, через інтерпретовану природу, Python може поступатися в швидкодії іншим мовам, що може бути критично важливим для застосувань, де необхідна висока обчислювальна потужність.

Java є об'єктно-орієнтованою мовою програмування, яка забезпечує високу продуктивність завдяки віртуальній машині Java (JVM) [14]. Її використовують у багатьох корпоративних середовищах для створення надійних і масштабованих систем. Java має потужні криптографічні бібліотеки, такі як Java Cryptography Extension (JCE), що робить її придатною для реалізації безпечних систем. Висока продуктивність та кросплатформеність є значними перевагами Java.

C# є мовою програмування, розробленою компанією Microsoft, яка поєднує в собі простоту, продуктивність та надійність [15]. Вона добре інтегрується з платформою .NET, що дозволяє розробляти різноманітні додатки, включаючи криптографічні. C# має багатий набір вбудованих класів для шифрування, таких як System.Security.Cryptography, що робить її ефективною для створення безпечних програмних засобів. Додатковою перевагою є сильна підтримка з боку Microsoft та активна спільнота розробників.

Вибір мови програмування C# для розробки системи «CrypterAES» обумовлений її високою продуктивністю та потужними можливостями для криптографії завдяки вбудованим класам System.Security.Cryptography. C# забезпечує надійність і безпеку завдяки інтеграції з платформою .NET, що дозволяє легко реалізувати захищені інформаційні системи. Крім того, Visual Studio, як основний інструмент розробки для C#, пропонує розширені можливості для налагодження та оптимізації коду. Інтеграція з Windows і кросплатформенність за допомогою .NET Core роблять C# ідеальним вибором для розробки масштабованих та ефективних програмних рішень.

Вибір системи управління базами даних (СУБД) є критично важливим етапом у процесі розробки будь-якої інформаційної системи. СУБД має забезпечувати надійне зберігання, обробку та доступ до даних, відповідати вимогам безпеки та мати високу продуктивність. Розглянемо три популярні СУБД, які можуть бути використані для розробки системи: MySQL, MS SQL Server та PostgreSQL.

MySQL є однією з найбільш відомих і широко використовуваних СУБД з відкритим вихідним кодом [16]. Вона забезпечує високу продуктивність та масштабованість, що робить її популярною для веб-додатків і середніх за розміром проектів. MySQL підтримує численні платформи і має велике співтовариство користувачів, що полегшує пошук допомоги та ресурсів.

MS SQL Server – це потужна СУБД, розроблена компанією Microsoft, яка пропонує високий рівень інтеграції з іншими продуктами Microsoft [17]. Вона забезпечує високу надійність, безпеку та продуктивність, а також підтримує складні аналітичні запити і бізнес-інтелект функції. MS SQL Server має зручний графічний інтерфейс для адміністрування та розширені можливості для резервного копіювання та відновлення даних.

PostgreSQL – це об'єктно-реляційна СУБД з відкритим вихідним кодом, відома своєю високою надійністю та функціональністю [18]. Вона підтримує широкий спектр типів даних і складних запитів, що робить її ідеальною для наукових та аналітичних застосувань. PostgreSQL має потужні інструменти

для управління транзакціями та забезпечення цілісності даних, а також активну спільноту розробників.

Для обґрунтованого вибору СУБД для розробки системи важливо порівняти основні характеристики популярних СУБД. Це допоможе визначити найбільш підходящу технологію, яка забезпечить необхідний рівень продуктивності, безпеки та інтеграції. У табл. 1.2 розглянуто порівняння MySQL, MS SQL Server та PostgreSQL за ключовими параметрами.

Вибір MS SQL Server для розробки даної системи обґрунтований кількома ключовими перевагами. По-перше, MS SQL Server пропонує високу інтеграцію з іншими продуктами Microsoft, що сприяє зручній взаємодії з існуючими корпоративними інструментами та платформами. По-друге, система забезпечує дуже високий рівень безпеки завдяки розширеним функціям захисту, таким як Transparent Data Encryption та Advanced Threat Protection. По-третє, розширені можливості для аналітики та бізнес-інтелекту дозволяють ефективно обробляти великі обсяги даних і проводити складні аналітичні запити. Нарешті, зручний графічний інтерфейс для адміністрування та розширені інструменти для резервного копіювання та відновлення даних роблять MS SQL Server надійним рішенням для розробки безпечної та ефективної системи.

2.5 Особливості програмної реалізації та основні режими роботи «CrypterAES»

Розробка ефективної методики шифрування базується на детальному розгляді ключових параметрів: рівня безпеки, енерговитрат на процеси кодування та декодування, а також обсягу даних, які можна обробити з високою ефективністю. У цьому контексті пріоритетним завданням стає досягнення оптимальної швидкості обробки, ресурсної ефективності та надійності зберігання і передачі інформації.

Ретельний аналіз вимог до системи шифрування дозволив визначити необхідність розробки алгоритму, який гармонійно поєднує в собі захист даних із оптимальним використанням системних ресурсів. У фазі програмної реалізації цього алгоритму було створено спеціалізований клас під назвою «InfoCrypter», який є ключовою частиною реалізації процесу шифрування.

Клас «InfoCrypter» включає два основні методи: «EncryptData» та «DecryptData». Метод «EncryptData» використовується для шифрування тексту, що передається як вхідний рядок разом із секретним ключем у вигляді масиву байтів. Він застосовує алгоритм AES для перетворення вхідного тексту в зашифровані дані. Особливість цього методу полягає у використанні генерованого вектору ініціалізації (IV), що забезпечує додатковий захист. Кінцевим продуктом є масив байтів, який включає зашифрований текст та IV, забезпечуючи високий рівень безпеки при зберіганні та передачі інформації.

У ліст. 2.1 можна ознайомитися з кодом методу «EncryptData», що демонструє реалізацію процесу кодування текстових даних для користувачів.

Лістинг 2.1 – Реалізація коду методу «EncryptData»

```
public byte[] EncryptData(string plainText, byte[] key) { // Метод для шифрування тексту за допомогою ключа.
    byte[] encryptedData; // Змінна для збереження зашифрованих даних.
    using (Aes aesAlg = Aes.Create()) { // Створення нового екземпляра AES.
      aesAlg.Key = key; // Задаємо ключ для шифрування.
      aesAlg.GenerateIV(); // Генеруємо вектор ініціалізації.
      aesAlg.Padding = PaddingMode.PKCS7; // Встановлюємо режим доповнення блоку.
      ICryptoTransform encryptor = aesAlg.CreateEncryptor(aesAlg.Key, aesAlg.IV); // Створення трансформера для шифрування.
      using (MemoryStream msEncrypt = new MemoryStream()) { // Використання потоку пам'яті для зберігання шифрованих даних.
        using (CryptoStream csEncrypt = new CryptoStream(msEncrypt, encryptor, CryptoStreamMode.Write)) { // Потік для шифрування даних.
          using (StreamWriter swEncrypt = new StreamWriter(csEncrypt)) {
```

```
                swEncrypt.Write(plainText); // Запис нешифрованого тексту для
шифрування.
                }
                encryptedData = msEncrypt.ToArray(); // Зберігання шифрованих даних у
масив.
            }
        }
        byte[] result = new byte[encryptedData.Length + 16]; // Створення кінцевого
масиву для зберігання результату.
        Buffer.BlockCopy(encryptedData, 0, result, 16, encryptedData.Length); //
Копіювання шифрованих даних.
        Buffer.BlockCopy(aesAlg.IV, 0, result, 0, 16); // Копіювання вектора ініціалізації
в масив.
        return result; // Повернення кінцевого масиву шифрованих даних.
    }
}
```

На початку методу оголошується змінна encryptedData, яка буде використовуватися для зберігання зашифрованих даних. Потім створюється новий екземпляр алгоритму AES за допомогою виклику Aes.Create(). Цей екземпляр використовується для налаштування параметрів шифрування, таких як ключ шифрування key, який передається в метод як аргумент. Далі генерується вектор ініціалізації (IV) за допомогою методу GenerateIV(). Вектор ініціалізації є важливим компонентом, що додає додатковий рівень безпеки до процесу шифрування. Режим доповнення блоку встановлюється на PaddingMode.PKCS7, що забезпечує правильне заповнення блоку, якщо розмір даних не кратний довжині блоку шифрування.

Для шифрування даних створюється криптографічний трансформер за допомогою методу CreateEncryptor, який використовує ключ і вектор ініціалізації. Цей трансформер буде використовуватися для шифрування вхідного тексту. Далі відкривається потік пам'яті (MemoryStream), який буде зберігати зашифровані дані. Цей потік обгорнуто у CryptoStream, який забезпечує шифрування даних, що записуються в потік пам'яті. Потім

створюється StreamWriter, який записує нешифрований текст (plainText) у криптографічний потік для його шифрування.

Після запису тексту у криптографічний потік, шифровані дані зберігаються у масиві encryptedData, отриманому з потоку пам'яті. Далі створюється новий масив байтів result, який має довжину, що дорівнює довжині зашифрованих даних плюс 16 байтів для вектора ініціалізації. Копіювання зашифрованих даних і вектора ініціалізації здійснюється за допомогою методу Buffer.BlockCopy. Спочатку зашифровані дані копіюються у result, починаючи з 16 байта, щоб залишити місце для вектора ініціалізації на початку масиву. Потім вектор ініціалізації копіюється у перші 16 байтів масиву result. На завершення метод повертає масив result, який містить зашифровані дані разом з вектором ініціалізації, що забезпечує додаткову безпеку при зберіганні та передачі зашифрованої інформації.

Ліст. 2.2 відображає код реалізації методу декодування.

Лістинг 2.2 – Реалізація коду методу «DecryptData»

```
public string DecryptData(byte[] cipherText, byte[] key) { // Метод для дешифрування зашифрованих даних.
    string plainText; // Змінна для збереження дешифрованого тексту.
    using (Aes aesAlg = Aes.Create()) { // Створення нового екземпляра AES.
      aesAlg.Key = key; // Встановлення ключа дешифрування.
      aesAlg.IV = cipherText.Take(16).ToArray(); // Встановлення вектора ініціалізації з зашифрованих даних.
      aesAlg.Padding = PaddingMode.PKCS7; // Встановлення режиму доповнення.
      ICryptoTransform decryptor = aesAlg.CreateDecryptor(aesAlg.Key, aesAlg.IV); // Створення трансформера для дешифрування.
      using (MemoryStream msDecrypt = new MemoryStream(cipherText.Skip(16).ToArray())) { // Використання потоку пам'яті для дешифрованих даних.
        using (CryptoStream csDecrypt = new CryptoStream(msDecrypt, decryptor, CryptoStreamMode.Read)) { // Потік для читання дешифрованих даних.
          using (StreamReader srDecrypt = new StreamReader(csDecrypt)) {
            plainText = srDecrypt.ReadToEnd(); // Зчитування дешифрованого тексту.
          }
```

```
            }
        }
    }
    return plainText; // Повернення дешифрованого тексту.
}
```

Метод DecryptData повертає дешифрований текст у вигляді рядка. Спочатку в методі оголошується змінна plainText, яка буде використовуватися для збереження результату дешифрування. Його процес починається зі створення нового екземпляра алгоритму AES за допомогою методу Aes.Create(). Потім встановлюється ключ дешифрування, переданий як аргумент key. Вектор ініціалізації (IV) отримується з перших 16 байтів зашифрованого тексту (cipherText). Для цього використовується метод Take(16).ToArray, який бере перші 16 байтів і перетворює їх у масив. Цей вектор ініціалізації необхідний для правильного дешифрування даних.

Режим доповнення встановлюється на PaddingMode.PKCS7, що дозволяє алгоритму правильно обробляти дані, якщо їх розмір не кратний довжині блоку. Потім створюється криптографічний трансформер для дешифрування за допомогою методу CreateDecryptor, який використовує ключ і вектор ініціалізації. Далі відкривається потік пам'яті (MemoryStream), який містить решту зашифрованих даних, починаючи з 16 байта. Це здійснюється за допомогою методу Skip(16).ToArray, який пропускає перші 16 байтів (вектор ініціалізації) і бере решту байтів зашифрованого тексту, перетворюючи їх у масив. Потік пам'яті обгортається у криптографічний потік (CryptoStream), який використовує трансформер для дешифрування даних у режимі читання. Потім створюється StreamReader, який читає дешифровані дані з криптографічного потоку.

За допомогою методу ReadToEnd вміст криптографічного потоку зчитується і зберігається у змінній plainText. Після завершення всіх операцій зчитування і дешифрування, метод повертає змінну plainText, яка містить результат дешифрування у вигляді рядка. Це дозволяє відновити оригінальний текст з зашифрованих даних.

Зберігання даних у системі відбувається за допомогою реалізованого методу «SaveBtn_Click», код якого приведено на рис. 2.9.

```
private void SaveBtn_Click(object sender, EventArgs e) {
  if (IsDataEnteringCorrect()) {
    _CodingsProvider.InsertCodings(CodingsNameTBox.Text,
      LoginForm.CurrentUser.UsersId, _Key, _EncryptedData);
    _LogsProvider.InsertLogs(LoginForm.CurrentUser.UsersId,
      "Був закодований текст із назвою "
      + CodingsNameTBox.Text, DateTime.Now);
    _HistorysProvider.InsertHistorys(DateTime.Now,
      CodingsNameTBox.Text);
    ClearAllControls();
    MessageBox.Show("Закодований текст збережений у базу даних!",
      "Повідомлення");
  }
}
```

Рисунок 2.9 – Метод для зберігання інформації користувача

Метод SaveBtn_Click виконується при натисканні кнопки «Зберегти» і призначений для збереження зашифрованих даних у базу даних. Спочатку перевіряється правильність введених даних за допомогою методу IsDataEnteringCorrect. Якщо введені дані відповідають вимогам валідації, виконуються наступні дії.

Метод _CodingsProvider.InsertCodings використовується для вставки нового запису про зашифровані дані у базу даних. Він отримує назву кодування з текстового поля CodingsNameTBox.Text, ідентифікатор користувача з властивості LoginForm.CurrentUser.UsersId, ключ шифрування _Key та зашифровані дані _EncryptedData. Цей виклик забезпечує збереження всієї необхідної інформації про кодування у базі даних.

Після цього викликається метод _LogsProvider.InsertLogs, який додає запис у логах про те, що було здійснено шифрування тексту. Метод отримує ідентифікатор користувача, повідомлення про здійснене шифрування, яке включає назву кодування, та поточну дату і час (DateTime.Now). Це дозволяє зберігати історію дій користувача для подальшого аналізу. Далі, метод _HistorysProvider.InsertHistorys додає запис у історію, включаючи поточну дату і час та назву кодування. Це дозволяє відслідковувати хронологію операцій шифрування.

Після збереження всіх необхідних даних у базу викликається метод ClearAllControls, який очищає всі елементи керування форми, готуючи її до наступного використання.

У кінці, відображається повідомлення користувачу за допомогою MessageBox.Show, яке інформує про успішне збереження зашифрованого тексту у базу даних. Це повідомлення містить текст «Закодований текст збережений у базу даних!» та заголовок «Повідомлення», що підтверджує успішне виконання операції.

Для дешифрування збережених даних із бази даних реалізовано метод «CodingsGridView_CellClick», код якого показано на рис. 2.10.

```
private void CodingsGridView_CellClick(object sender,
  DataGridViewCellEventArgs e) {
  if (e.RowIndex >= 0 &&
    CodingsGridView[0, e.RowIndex].Value.ToString() != _CodingsList[0].Message) {
    Codings selectedCodings = new Codings();
    selectedCodings = _CodingsProvider.SelectedCodingsByCodingsId(
      Convert.ToInt32(CodingsGridView[0, e.RowIndex].Value.ToString()));
    CodingTextRTBox.Text = _InfoCrypter.DecryptData(
      selectedCodings.CodingText, selectedCodings.GenerateKey);
  }
}
```

Рисунок 2.10 – Реалізація методу «CodingsGridView_CellClick»

Метод CodingsGridView_CellClick виконується при натисканні на комірку в елементі CodingsGridView. Він обробляє подію кліку на конкретну комірку і відповідає за відображення зашифрованого тексту у текстовому полі CodingTextRTBox.

Спочатку перевіряється, чи індекс рядка (e.RowIndex) є більшим або рівним нулю, щоб переконатися, що вибрана комірка не є заголовком. Далі, перевіряється значення комірки у першому стовпці (CodingsGridView[0, e.RowIndex].Value.ToString) поточного рядка. Це значення не повинно відповідати повідомленню _CodingsList[0].Message, що вказує на те, що дані у комірці є дійсними і не містять повідомлення про відсутність даних.

Якщо обидві умови виконуються, створюється новий об'єкт Codings під назвою selectedCodings. Цей об'єкт наповнюється даними за допомогою виклику методу SelectedCodingsByCodingsId, який приймає ідентифікатор кодування, перетворений з рядкового значення комірки у перший стовпець

вибраного рядка у ціле число. Цей метод повертає об'єкт Codings, що містить дані про вибране кодування.

Потім текстове поле CodingTextRTBox наповнюється дешифрованими даними. Це здійснюється за допомогою виклику методу _InfoCrypter.DecryptData, який приймає зашифрований текст (selectedCodings.CodingText) і згенерований ключ (selectedCodings.GenerateKey) як параметри. Результат дешифрування – оригінальний текст – встановлюється як текст у текстовому полі CodingTextRTBox.

Цей процес забезпечує відображення дешифрованого тексту у користувацькому інтерфейсі при натисканні на відповідну комірку у таблиці CodingsGridView.

2.6 Організація тестування та налагодження програмного засобу «CrypterAES»

Для експериментальної перевірки ефективності та безпеки розробленого методу шифрування даних використовується спеціально створений додаток, який піддався всебічному тестуванню. Робота з додатком починається з процедури автентифікації, зображеної на рис. 2.11. Ця процедура є важливим елементом системи безпеки, оскільки запобігає несанкціонованому доступу до конфіденційної інформації.

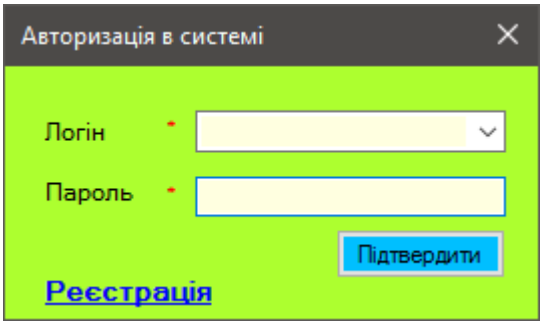


Рисунок 2.11 – Форма автентифікації користувача

Після введення логіна та пароля активуються алгоритми перевірки автентичності, які здійснюють валідацію введених даних. Успішне проходження автентифікації надає доступ до головного інтерфейсу додатка,

що містить різноманітні опції для роботи з методом шифрування. Користувач може ініціювати процес шифрування, зберігати результати в базі даних, переглядати журнали активності та створювати звіти.

Для нових користувачів, які ще не мають облікового запису, передбачена можливість реєстрації. Процес реєстрації включає заповнення форми, яка містить кілька обов'язкових полів, таких як назва облікового запису та пароль, як показано на рис. 2.12. Ці дані будуть використовуватися для подальшої ідентифікації користувача в системі. Успішна реєстрація автоматично активує новий обліковий запис, що дозволяє користувачеві проводити експериментальну перевірку методу шифрування, відстежувати результати та оцінювати рівень захищеності даних.

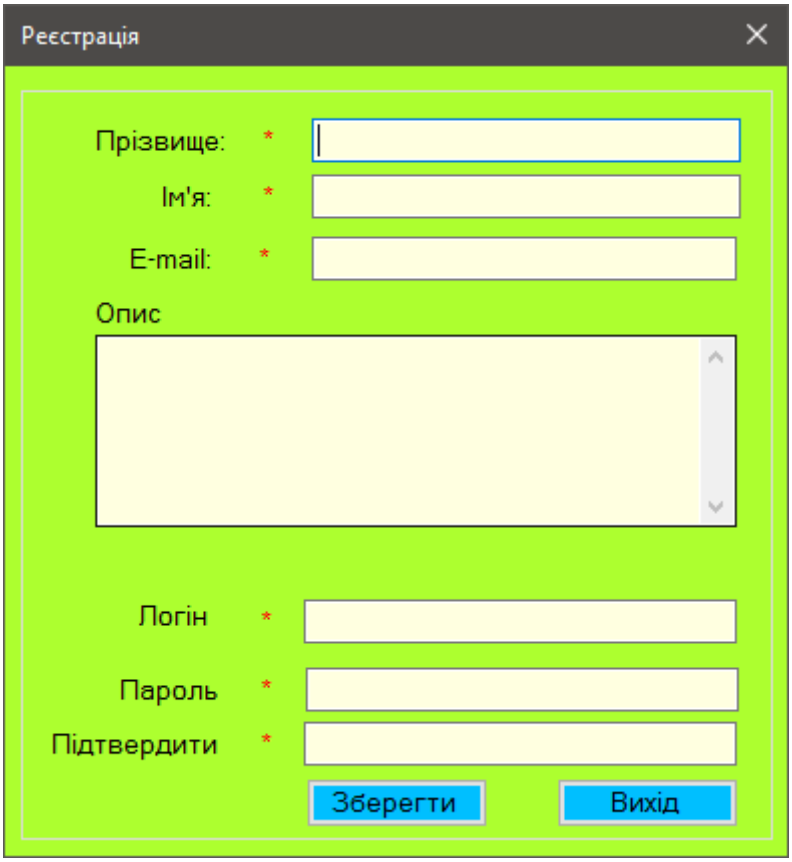


Рисунок 2.12 – Форма для реєстрації користувача

У рамках експериментального оцінювання розробленої методики кодування даних, ключовим аспектом є аналіз взаємодії користувачів з програмним забезпеченням. Важливим елементом є призначення ролі системного адміністратора для першого зареєстрованого користувача, що

забезпечує високий рівень управління системою від моменту її ініціації. Наступні користувачі мають ролі звичайних користувачів, що є звичайною практикою для створення ієрархічної структури доступу.

Для підвищення ефективності користувацького інтерфейсу в систему інтегровано випадаючий список для оперативного пошуку імен користувачів, що поліпшує взаємодію завдяки автоматичному сортуванню за введеними даними.

Проходячи аутентифікацію, користувач дістає доступ до основного інтерфейсу програми, який зображений на рис. 2.13. Цей інтерфейс є центром всіх доступних функцій та налаштувань, включно з кодуванням, декодуванням, переглядом зашифрованих даних та інше, що сприяє легкій орієнтації користувача у можливостях програми.

Рисунок 2.13 – Основний екран програми

В контексті експериментальної перевірки системи варто акцентувати увагу на ролях користувачів, що включають системного адміністратора з широким спектром управлінських прав та звичайного користувача.

Переходячи до експериментальної перевірки методу кодування, користувачеві слід вибрати опцію «Кодування» у вкладці меню «Управління», яка веде до вікна, схематично зображеного на рис. 2.14. У цьому вікні користувач може провести експериментальні дії, такі як введення і збереження інформації, аналізуючи принципи роботи алгоритму, його

швидкість та надійність. Це дозволяє не тільки взаємодіяти з системою, але й ефективно оцінити впроваджені методи безпеки даних.

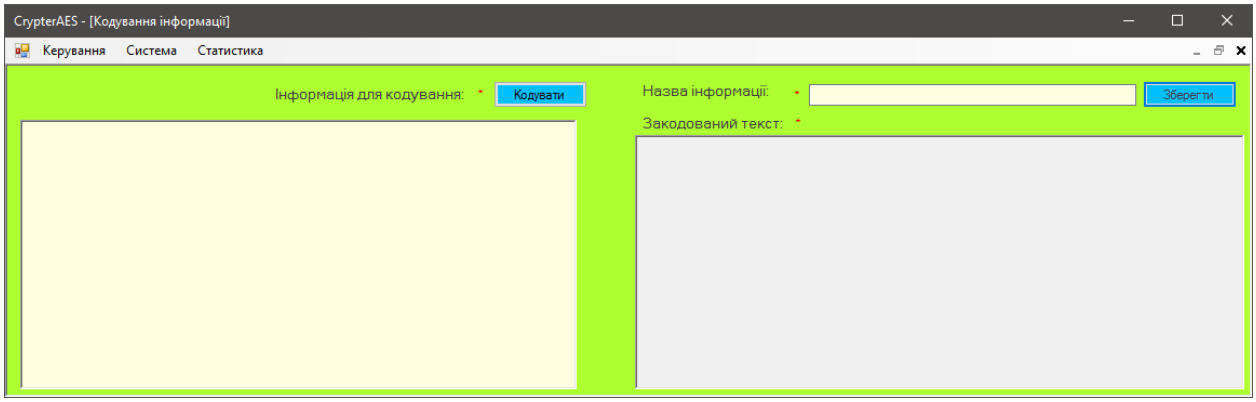


Рисунок 2.14 – Форма для кодування інформації

У контексті експериментального аналізу ефективності та безпеки імплементованого методу кодування даних, зосередимо увагу на інтеракції користувача зі специфічним інтерфейсом. Першим кроком у цьому дослідженні є введення або вставка інформації у поле під назвою «Інформація для кодування». Після цього, користувач ініціює процес кодування, натискаючи на кнопку «Кодувати». Подальший етап полягає в застосуванні розробленої методики кодування для трансформації даних, які відображаються у правій частині екрану, згідно з рис. 2.15.

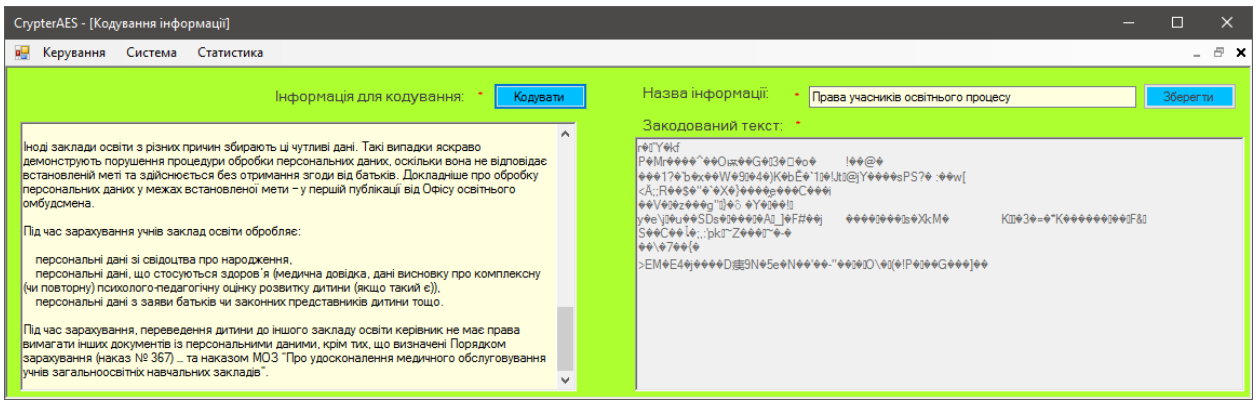


Рисунок 2.15 – Візуалізація процесу кодування введеної інформації.

Ця процедура не лише ілюструє можливості системи, а й служить ключовим засобом для вимірювання її ефективності та безпеки. Важливим аспектом експерименту є аналіз швидкості кодування, ступеня захисту і надійності методу за різноманітних умов обробки даних. Такий підхід

дозволяє встановити придатність розробленої методики для застосування в реальних сценаріях.

Після ініціації процесу шифрування, існує можливість зберегти результати у базі даних. Це робиться шляхом введення назви у поле «Назва інформації» та натискання на кнопку «Зберегти». Операція збереження підтверджується системним повідомленням, що демонструється на рис. 2.16.

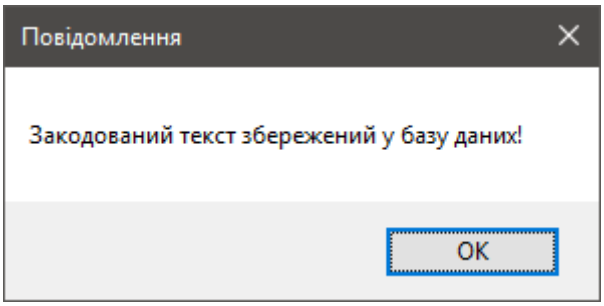


Рисунок 2.16 – Повідомлення про успішне кодування і збереження

Експеримент дозволяє аналізувати не тільки безпеку зашифрованої інформації, але й ефективність стиснення даних за допомогою використаного методу. Перегляд рис. 2.16 підтверджує, що метод ефективно виконує стиснення та шифрування інформації.

Щодо відновлення збереженої інформації, необхідно перейти через меню «Управління» до розділу «Декодування», де відкривається нове вікно інтерфейсу, представлене на рис. 2.17. Цей етап надає змогу експериментально перевірити не лише точність декодування, але й надійність зберігання зашифрованих даних. Комплексний підхід у цій перевірці забезпечує багаторівневу оцінку ефективності та безпеки використовуваного методу в реальних умовах експлуатації.

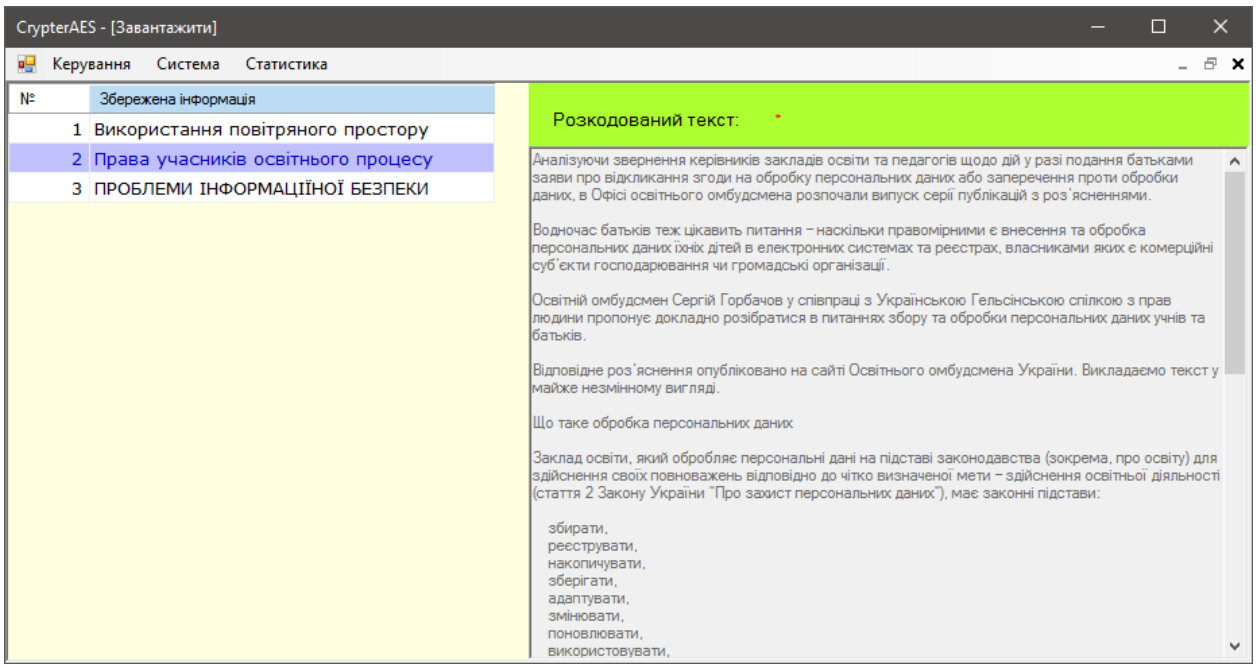


Рисунок 2.17 – Інтерфейс завантаження та розшифрування інформації

У контексті експериментального дослідження, система відіграє ключову роль у забезпеченні безпеки та конфіденційності за допомогою використання розробленої технології шифрування. Всі дані, які вносять користувачі, зберігаються у базі в зашифрованому форматі. Доступ до дешифрованих даних можливий тільки через вибір відповідного запису з інтерфейсу, розташованого в лівій частині екрану. Це не тільки автоматизує процес збереження даних, але й значно підвищує рівень їхньої безпеки.

Додатково, система містить функції управління обліковими записами. Користувачі можуть ввести свої особисті дані, такі як ім'я, прізвище, логін і пароль, які після цього шифруються і зберігаються в базі. Ці дані не лише ідентифікують особу, але й гарантують конфіденційність та інтегритет інформації, як це показано на рис. 2.18.

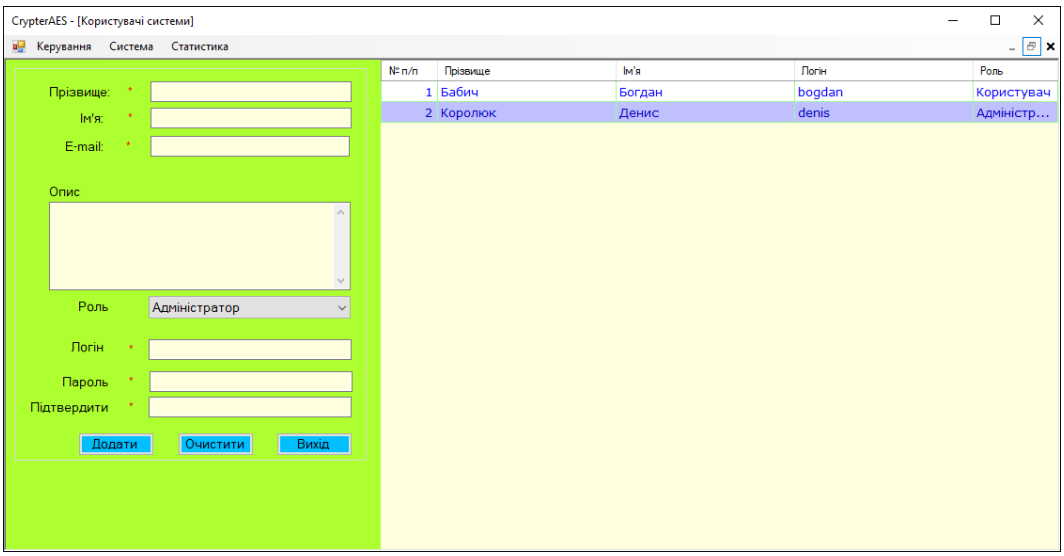


Рисунок 2.18 – Обробка облікових даних

У процесі експериментальної перевірки функціональності та безпеки модуля «Персоналізація» системи, було виконано ряд тестів, під час яких користувачі могли змінювати свої персональні дані через інтерфейс, зображений на рис. 2.19. Це дослідження демонструє, наскільки система здатна адаптуватися до індивідуальних потреб користувачів, забезпечуючи при цьому надійний рівень захисту.

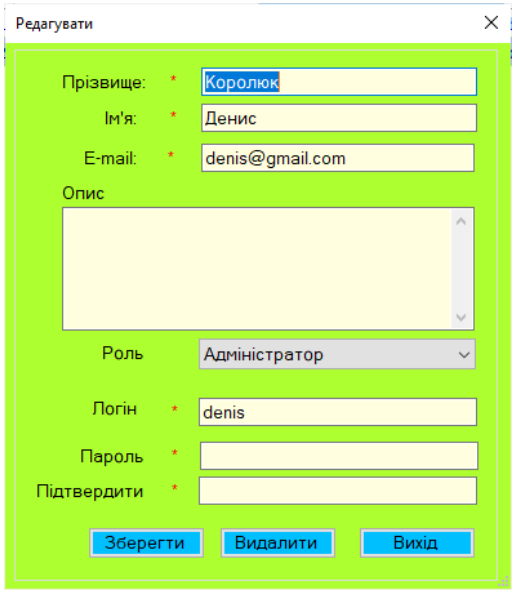
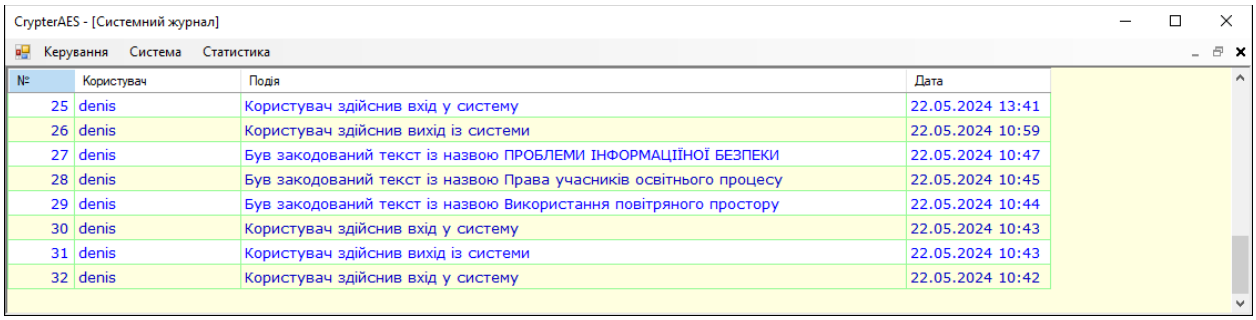


Рисунок 2.19 – Інтерфейс для зміни даних облікового запису

Модуль «Події», розташований в меню «Система» (рис. 2.20) надає адміністратору інструменти для моніторингу активності користувачів у межах платформи. Результати експериментальної перевірки підтвердили, що ця функція ефективно реєструє усі основні дії користувачів, включаючи

зміни в облікових записах, доступ до зашифрованої інформації та інші важливі системні дії. Це не тільки підвищує безпеку, але також уможливлює проведення аудиту безпеки з метою виявлення потенційних вразливостей чи незаконних дій.



№	Користувач	Подія	Дата
25	denis	Користувач здійснив вхід у систему	22.05.2024 13:41
26	denis	Користувач здійснив вихід із системи	22.05.2024 10:59
27	denis	Був закодований текст із назвою ПРОБЛЕМИ ІНФОРМАЦІЙНОЇ БЕЗПЕКИ	22.05.2024 10:47
28	denis	Був закодований текст із назвою Права учасників освітнього процесу	22.05.2024 10:45
29	denis	Був закодований текст із назвою Використання повітряного простору	22.05.2024 10:44
30	denis	Користувач здійснив вхід у систему	22.05.2024 10:43
31	denis	Користувач здійснив вихід із системи	22.05.2024 10:43
32	denis	Користувач здійснив вхід у систему	22.05.2024 10:42

Рисунок 2.20 – Вікно події

Модуль статистики використання програми «CrypterAES» призначений для отримання статистичної інформації щодо використання програми (рис. 2.21). Він надає адміністратору можливість аналізувати активність у системі за обраний період, що сприяє підвищенню ефективності контролю та моніторингу роботи користувачів.

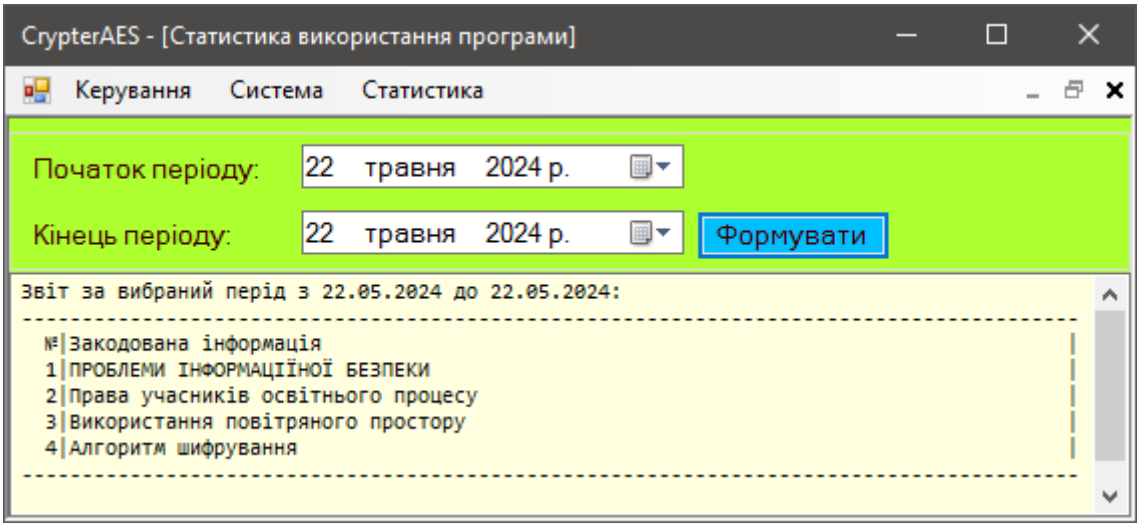


Рисунок 2.21 – Статистика використання програми

Інтерфейс модуля складається з полів вибору періоду, кнопки формування звіту та поля відображення звіту. Поля вибору періоду дозволяють визначити початок та кінець періоду для формування звіту. Вони розташовані у верхній частині вікна і включають випадаючі списки для вибору дати початку та кінця періоду. Кнопка «Формувати», розташована

праворуч від полів вибору дати, запускає процес генерації звіту за обраний період. Поле відображення звіту знаходиться у нижній частині вікна і містить результати аналізу у вигляді списку, що включає інформацію про використання програми, зокрема номери та опис подій чи категорій активності.

Для отримання звіту адміністратор повинен спочатку вибрати період аналізу, використовуючи випадаючі списки «Початок періоду» та «Кінець періоду». Після вибору дат необхідно натиснути кнопку «Формувати», після чого система автоматично згенерує звіт і відобразить його у нижній частині вікна. Адміністратор може переглянути згенерований звіт, який містить перелік категорій активності за обраний період, і використати отриману інформацію для аналізу роботи користувачів, виявлення потенційних проблем та прийняття рішень щодо покращення ефективності системи.

Вибір коротких періодів може допомогти у виявленні короткострокових тенденцій або аномалій у використанні програми. Рекомендується регулярно генерувати звіти для забезпечення постійного моніторингу та своєчасного реагування на будь-які відхилення у роботі системи. Модуль статистики використання програми «CrypterAES» є важливим інструментом для адміністратора, що дозволяє отримувати докладну інформацію про активність користувачів, сприяючи підвищенню рівня безпеки та ефективності роботи системи.

2.7 Рекомендації по використанню та впровадженню програмного засобу «CrypterAES»

Програмний засіб «CrypterAES» розроблено для забезпечення високого рівня безпеки даних шляхом їх шифрування. Цей програмний продукт може бути ефективно інтегрований у різноманітні системи, де важлива конфіденційність та захист інформації. Наведені нижче рекомендації допоможуть максимально ефективно використовувати та впроваджувати «CrypterAES».

Перед впровадженням «CrypterAES» необхідно визначити оптимальне середовище для його функціонування. Рекомендується використовувати сучасні сервери з високим рівнем обчислювальної потужності, щоб забезпечити швидкість обробки великих обсягів даних. Також важливо мати надійне мережеве з'єднання для безперервного функціонування програми.

Для успішного впровадження та ефективного використання програмного засобу «CrypterAES» необхідно дотримуватись кількох важливих кроків. Ці кроки допоможуть забезпечити належний рівень безпеки даних, ефективність системи та підготовку персоналу. Основні етапи включають:

– аналіз вимог. Провевести детальний аналіз вимог до безпеки даних. Визначте, які саме дані потребують шифрування і який рівень захисту є необхідним;

– тестування. Перед впровадженням у робоче середовище, необхідно провести тестування «CrypterAES» у тестовій інфраструктурі. Це дозволить виявити можливі проблеми та налаштувати систему відповідно до специфічних потреб;

– навчання персоналу. Потрібно забезпечити навчання для користувачів і адміністраторів системи. Важливо, щоб всі залучені працівники розуміли принципи роботи з програмним засобом та дотримувалися правил безпеки.

Для успішного впровадження програмного засобу «CrypterAES» важливо дотримуватись чітко визначених етапів, що забезпечать ефективне використання системи та відповідність її роботи до вимог безпеки. У разі відсутності інсталятора, необхідно використовувати наявну папку Debug, яка містить програму та конфігураційний файл «InfProtectionApp.exe.config». Це вимагає особливої уваги до налаштувань з'єднання з базою даних. Впровадження системи складається з наступних кроків:

– інсталяція. Потрібно виконати інсталяцію програмного засобу на сервер або робочі станції, згідно з технічними вимогами. Необхідно

переконатись, що всі необхідні компоненти з папки Debug скопійовані до відповідних директорій на цільових машинах;

– конфігурація. Налаштувати «CrypterAES» відповідно до вимог відповідного закладу. Це включає налаштування параметрів шифрування, управління користувачами та правами доступу. Особливу увагу слід приділити конфігураційному файлу «InfProtectionApp.exe.config», де необхідно вказати правильні параметри з'єднання з базою даних;

– інтеграція. Також потрібно інтегрувати програмний засіб з існуючими системами та базами даних. Потрібно переконатись, що всі дані, які потребують захисту, коректно шифруються і зберігаються. Важливо також перевірити сумісність з іншими програмами, що використовуються в закладі, для забезпечення безперебійної роботи.

Для ефективного використання системи «CrypterAES» необхідно дотримуватись визначених процедур, що забезпечать належне шифрування, дешифрування та моніторинг даних. Кожен етап має важливе значення для підтримки високого рівня безпеки та функціональності системи. Основні етапи використання системи включають:

– шифрування даних. Використання інтерфейсу програми для введення, шифрування та збереження конфіденційних даних. Система забезпечує високий рівень захисту, шифруючи дані одразу після їх введення. Це гарантує, що всі конфіденційні дані будуть захищені від несанкціонованого доступу з моменту їх створення;

– дешифрування даних. Для доступу до зашифрованої інформації необхідно скористатись функцією дешифрування, вибираючи необхідні записи через інтерфейс програми. Цей процес забезпечує відновлення даних до їх первісного стану, дозволяючи користувачам працювати з інформацією в зручному та безпечному форматі;

– моніторинг активності. Використання модуля «Події» для моніторингу активності користувачів. Це дозволяє відслідковувати всі ключові дії в системі та забезпечувати додатковий рівень безпеки.

Моніторинг активності допомагає виявляти потенційні загрози та несанкціоновану діяльність, тим самим підвищуючи загальний рівень захищеності даних.

Ці кроки є критично важливими для забезпечення ефективного використання системи «CrypterAES» та підтримки високого рівня безпеки даних в організації.

Для забезпечення безперебійної роботи та високого рівня захисту даних при використанні програмного засобу «CrypterAES», важливо дотримуватись належних процедур підтримки та обслуговування. До цих процедур належать:

– регулярні оновлення. Забезпечення регулярного оновлення програмного забезпечення для отримання нових функцій і виправлення виявлених вразливостей. Це дозволяє системі залишатися актуальною та ефективно протистояти новим загрозам;

– аудит безпеки. Проведення регулярного аудиту безпеки для перевірки ефективності заходів захисту та виявлення можливих загроз. Аудит дозволяє своєчасно виявляти і усувати слабкі місця в системі безпеки, підвищуючи загальний рівень захищеності даних;

– резервне копіювання. Регулярне створення резервних копій даних для запобігання втраті інформації у випадку збою або несанкціонованого доступу. Резервні копії гарантують можливість відновлення критично важливої інформації, забезпечуючи безперервність бізнес-процесів та збереження даних.

Дотримання цих процедур підтримки та обслуговування є необхідним для забезпечення надійної та безпечної роботи програмного засобу «CrypterAES», що, в свою чергу, сприяє підвищенню довіри користувачів і захисту конфіденційної інформації.

Інтеграція та використання програмного засобу «CrypterAES» вимагають ретельного планування та підготовки. Дотримання вищенаведених рекомендацій допоможе забезпечити високу ефективність та

безпеку вашої системи, що є критично важливим у сучасному інформаційному середовищі.

ВИСНОВКИ

Дана робота присвячена розробці методу захисту інформації через створення програмного засобу «CrypterAES». «CrypterAES» призначений для забезпечення конфіденційності та цілісності даних шляхом їхнього шифрування з використанням алгоритму AES. Основний функціонал системи включає можливості шифрування та дешифрування даних, що дозволяє зберігати інформацію у зашифрованому вигляді та відновлювати її лише уповноваженим користувачам. Система також містить модуль управління обліковими записами користувачів, який дозволяє вводити, шифрувати та зберігати персональні дані, такі як ім'я, прізвище, логін та пароль, забезпечуючи їх захист від несанкціонованого доступу. Додатково, програмний засіб оснащений функцією моніторингу активності користувачів через модуль «Події», що дозволяє відслідковувати всі ключові дії в системі та забезпечувати додатковий рівень безпеки. Таким чином, «CrypterAES» забезпечує комплексний захист інформації, поєднуючи сучасні методи шифрування, управління доступом та моніторинг безпеки.

У першому розділі була здійснена теоретична основа захисту інформації, де розглянуто основні поняття та визначення, що стосуються інформаційної безпеки, зокрема конфіденційність, цілісність та доступність. Були розглянуті існуючі методи захисту інформації, як технічні, так і організаційні, включаючи шифрування даних, системи контролю доступу, антивірусне програмне забезпечення та політики інформаційної безпеки. Крім того, проведено огляд програмних засобів VeraCrypt, BitLocker та Cryptomator з аналізом їх переваг і недоліків.

Другий розділ був присвячений проектуванню, розробці та тестуванню методу шифрування інформації за допомогою програмного засобу «CrypterAES». На початку цього розділу було визначено завдання, призначення та вимоги до програмного засобу. Здійснено вибір технологій для його розробки, де серед проаналізованих мов програмування (Python,

Java, C#) обрано C#, а серед СУБД (MySQL, MS SQL Server, PostgreSQL) – MS SQL Server. Розроблено математичну модель вдосконаленого методу кодування на базі AES, описано реалізацію основних програмних модулів та інтеграцію алгоритму шифрування та дешифрування в систему «CrypterAES». Проведено функціональне тестування всіх функцій системи для забезпечення її надійності та ефективності. В результаті розробки були сформовані рекомендації щодо впровадження та ефективного використання програмного засобу. Варто зазначити, що дана система ще не впроваджена у робоче середовище.

Проведена робота демонструє застосування сучасних методів захисту інформаці, що дозволяє значно підвищити рівень безпеки конфіденційних даних. Завдяки розробленому програмному засобу «CrypterAES», який базується на алгоритмі AES, забезпечується високий рівень захисту даних. Використання MS SQL Server як СУБД гарантує надійне зберігання та управління інформацією. Отримані результати підтверджують ефективність запропонованих рішень та їхню придатність для реального використання в умовах сучасних корпораціях після впровадження системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Dworkin, M. , Barker, E. , Nechvatal, J. , Foti, J. , Bassham, L. , Roback, E. and Dray, J. (2001), Advanced Encryption Standard (AES), Federal Inf. Process. Stds. (NIST FIPS), National Institute of Standards and Technology, Gaithersburg, MD, URL: https://doi.org/10.6028/NIST.FIPS.197 (Дата звернення 05.05.2024).

2. Остапов С. Е. Технології захисту інформації : навчальний посібник / С. Е. Остапов, С. П. Євсеєв, О. Г. Король. – Х. : Вид. ХНЕУ, 2012. – 476 с.

3. Withdrawn NIST Technical Series Publication URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf (Дата звернення 05.05.2024).

4. Federal Information Processing Standards Publication 186-4. URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf (Дата звернення 05.05.2024).

5. Federal Information Processing Standards Publication 180-4. URL: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf (Дата звернення 05.05.2024).

6. What is OpenSSL? By Robert Sheldon. URL: https://www.techtarget.com/whatis/definition/OpenSSL (Дата звернення 05.05.2024).

7. CryptoAPI System Architecture. URL: https://learn.microsoft.com/en-us/windows/win32/seccrypto/cryptoapi-system-architecture (Дата звернення 05.05.2024).

8. What is .NET? URL: https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet (Дата звернення 05.05.2024).

9. What is .NET, and why should you choose it? URL: https://devblogs.microsoft.com/dotnet/why-dotnet/ (Дата звернення 05.05.2024).

10. Free Open source disk encryption with strong security URL: https://www.veracrypt.fr/en/Home.html (Дата звернення 05.05.2024).

11. BitLocker URL: https://learn.microsoft.com/uk-ua/windows/security/operating-system-security/data-protection/bitlocker/ (Дата звернення 05.05.2024).

12. Cryptomator - Free Cloud Encryption for Dropbox & Co URL: https://cryptomator.org/ (Дата звернення 05.05.2024).

13. Бондаренко В.В., Коваленко В.В. Розробка програмного забезпечення з використанням Python, Java та C#: Збірник наукових праць. - Одеса: Видавництво "Сонячна Україна", 2018. - 250 с.

14. Еккель Б. Мислити на Java: Навчальний посібник. - Харків: Видавництво Харківського національного університету імені В.Н. Каразіна, 2016. - 1150 с.

15. Безменов М.І., Безменова О.М., Калінін Д.В. Основи візуального програмування мовою C#: навч. посіб. для студентів навчально-наукового інституту комп'ютерних наук та інформаційних технологій. – Харків: ФОП Панов А. М., 2023. 648 с.

16. Nixon R. Learning PHP, MySQL, JavaScript, and CSS: A step-by-step guide to creating dynamic websites. – «O'Reilly Media, Inc.», 2012. – 315 p.

17. Microsoft SQL Server : веб-сайт. URL: https://www.metabase.com/data_sources/microsoft-sql-server (дата звернення 05.05.2024).

18. Integration of Data Mining Techniques to PostgreSQL Database Manager System. URL: https://www.sciencedirect.com/science/article/pii/S1877050919309949 (дата звернення 05.05.2024).

АНОТАЦІЯ

Королюк Д.В. Реалізація методу шифрування AES засобами .Net

Кваліфікаційна робота на здобуття освітнього ступеня «бакалавр» за спеціальністю 122 Комп'ютерні науки. Волинський національний університет імені Лесі Українки, Луцьк, 2024 р.

Дана робота присвячена дослідженню процесу захисту інформації, включаючи методи, засоби забезпечення інформаційної безпеки, дії та заходи, спрямовані на запобігання несанкціонованому доступу до даних

Дослідження проводилось з метою вивчення та реалізації алгоритму AES кодування текстової інформації для використання для шифрування, збереження та розшифрування інформації.

Для досягнення поставленої мети вирішено наступні завдання:

– проведего аналіз існуючих методів та засобів захисту інформації;

– розроблено вдосконалений алгоритм кодування на основі AES;

– реалізувано та протестовано розроблений алгоритм;

– розроблено рекомендації та методичні вказівки для використання програми.

Практичним результатом є створення завершеного діючого програмного продукту для шифрування, збереження та розшифрування текстової інформації.

Практичне значення одержаних результатів полягає у підвищенні рівня інформаційної безпеки користувачів.

Ключові слова: алгоритм, кодування, AES, шифрування даних, розшифрування даних, інформаційна безпека, .NET.