


**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Радіотехнічний факультет
Кафедра радіоконструювання та виробництва радіоапаратури**

До захисту допущено:

В.о. зав.кафедрою

 Євгеній НЕЛІН

_____ 20__ р.

**Дипломний проєкт
на здобуття ступеня бакалавра
за освітньо-професійною програмою «Інтелектуальні технології
мікросистемної радіoeлектронної техніки»
за спеціальністю 172 «Телекомунікації та радіотехніка»
на тему: Алгоритм шифрування повідомлень в розподілених мережах
зв'язку**

Виконав:

студент III курсу, групи РІ-зп81

Неділько Максим Олександрович

Керівник: доцент, к.т.н. Шульга А.А.

Рецензент: доцент, к.т.н. Лащевська Н.О.

Засвідчую, що у цьому дипломному
проєкті немає запозичень з праць інших
авторів без відповідних посилань.

Студент

Київ – 2021 року

[illegible]

				РІзп81.464419.001		
	ПІБ	Підп.	Дата	Алгоритм шифрування повідомлень в розподілених мережах зв'язку	Лист	Листів
Розробн.	Неділько М.О	<i>М.О. Неділько</i>			1	1
Керівн.	Шульга А.В.				КПІ ім. Ігоря Сікорського Каф. КіВРА, Гр. РІ-зп81	
Н/контр.						
Зав.каф.	Нелін Є. А.					

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Радіотехнічний факультет
Кафедра радіоконструювання та виробництва радіоапаратури
Рівень вищої освіти – перший (бакалаврський)
Спеціальність – 172 «Телекомунікації та радіотехніка»
Освітньо-професійна програма «Інтелектуальні технології мікросистемної
радіoeлектронної техніки»

ЗАТВЕРДЖУЮ

В.о.зав. кафедрою

Євгеній НЕЛІН
«18» _____ 20__ р.

ЗАВДАННЯ
на дипломний проєкт студенту
Недільку Максиму Олександровичу

1. Тема проєкту «Алгоритм шифрування повідомлень в розподілених мережах зв'язку», керівник проєкту Шульга Аліна Вікторівна, доцент, к.т.н., затверджені наказом по університету від «18»травня 2021 р. №1205-с
2. Термін подання студентом проєкту 09 червня 2021 року
3. Вихідні дані до проєкту: удосконалити алгоритм шифрування інформації на основі симетричного блочного алгоритму AES, використати мову програмування Java.
4. Зміст пояснювальної записки: Вступ; Аналіз технічного завдання; Проблеми та рішення захисту інформації; Опис зовнішньої частини програми. Графічний інтерфейс; Опис внутрішньої частини програми. Програмний код; Програмна реалізація системи; Висновки
5. Перелік графічного матеріалу (із зазначенням обов'язкових креслеників, плакатів, презентацій, тощо): блок схема алгоритма AES; блок схема додатку; функціонування JVM.

6. Дата видачі завдання 12 квітня 2021 року

Календарний план

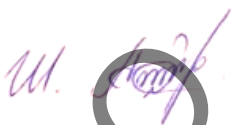
№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
	Вивчення та аналіз задачі	12.04 .2021р.	Виконано
	Розробка архітектури та загальної структури системи	20.04.2021р.-26.04.2021р	Виконано
	Розробка структур окремих підсистем	27.04.2021р.-02.05.2021р	Виконано
	Підготовка матеріалів	03.05.2021р.-08.05.2021р	Виконано
	Програмна реалізація системи	09.05.2021р.-31.05.2021р	Виконано
	Оформлення пояснювальної записки	01.06.2021р.- 09.06.2021р	виконано

Студент



Максим НЕДІЛЬКО

Керівник



♦ Аліна ШУЛЬГА

АНОТАЦІЯ

Метою роботи була розробка системи попередньої обробки та шифрування повідомлень. Кожний користувач вказує свій пароль та передає користувачу для дешифрування. Паролі, якими зашифровані ключі користувачів в системі не зберігаються, натомість вони зберігаються на носії користувача, і при вході в систему остання потребує вказати в певній графі пароль. Система шифрує текст, що передається, обраним методом шифрування, а інші дані, такі як зображення, розбиваються на частини, деякі з яких шифруються.

Користувацький додаток отримує повідомлення у вигляді текстового файлу, записує зашифроване повідомлення в інший файл. Процес дешифрування виконується в зворотньому порядку. Додаток має графічний інтерфейс зручний для користування.

Записка містить 39 сторінок, 10 рисунків та 9 посилань.

ANOTATION

The aim of the work was to develop a system for pre-processing and encryption of messages. Each user enters their password and passes it to the user for decryption. Passwords that do not store encrypted user keys in the system are stored instead on the user's media, and when logging in, the latter needs to specify a password in a certain column. The system encrypts the transmitted text with the selected encryption method, and other data, such as images, are broken down into parts, some of which are encrypted.

The user application receives the message as a text file, writes the encrypted message to another file. The decryption process is performed in reverse order. The application has a user-friendly graphical interface.

The note contains 39 pages, 10 figures and 9 references.

**Пояснювальна записка
до дипломного проекту
на тему: Алгоритм шифрування повідомлень в
розподілених мережах зв'язку**

Київ – 2021 року

ЗМІСТ

Перелік скорочень.....	3
Вступ.....	4
1. Аналіз технічного завдання	5
2. Проблеми та рішення захисту інформації.....	6
2.1 Поняття шифрування.....	6
2.2 Класичні алгоритми шифрування даних.....	9
2.3 Переваги Java над іншими мовами.....	17
2.4 Середовище розробки.....	19
3. Опис зовнішньої частини програми. Графічний інтерфейс.....	21
3.1 Графічний інтерфейс.....	21
3.2 Бібліотека Swing.....	22
3.3 Концепція додатка.....	23
4. Опис внутрішньої частини програми. Програмний код.....	25
4.1 Структура коду.....	25
4.2 Клас з алгоритмом AES.....	26
4.3 Суть методу шифрування.....	29
5. Програмна реалізація системи.....	31
5.1 Java virtual machine	31
5.2 Користування програмною системою.....	33
Висновки.....	38
Перелік джерел посилань.....	39

					Plзп81.464419.001 ПЗ				
ЗМ..	Лист	№ докум	Підпис	Дата	Алгоритм шифрування повідомлень в розподілених мережах зв'язку	Лім.	Лист	Листів	
Розробив	Неділько М.О.					у	2		
Перевірів						Pl-зп81 РТФ			
Н. Контр.	Шульга А. В.								
Затвердив									

ПЕРЕЛІК СКОРОЧЕНЬ

ASCII — американський стандартний код для інформаційного обміну

IC — інформаційна система

ЕЦП — електронний цифровий підпис

TLS — протокол захисту транспортного рівня

SSL — рівень захищених сокетів

HTTPS — схема URI

URI — уніфікований ідентифікатор ресурсів

DES — симетричний алгоритм шифрування

LDAP - протокол полегшеного доступу

IDE — інтегроване середовище розробки

Swing — інструментарій для створення графічного інтерфейсу користувача

GUI — графічний інтерфейс користувача

AES — алгоритм симетричного блочного шифрування

ECB — електронна книга кодів

ВСТУП

Захист інформації перетворюється сьогодні на одну з найактуальніших задач внаслідок надзвичайно широкого розповсюдження як власне різноманітних систем обробки інформації, так і розширення локальних та глобальних комп'ютерних мереж, якими передаються величезні об'єми інформації державного, військового, комерційного, приватного характеру, власники якої часто були б категорично проти ознайомлення з нею сторонніх осіб. Проблема набуває особливої гостроти після прийняття урядом України закону про захист персональних даних, який зобов'язує зберігати та передавати персональні дані працівників лише у захищеному вигляді в інформаційних системах (ІС). Не менш важливим завданням вважається широке впровадження інформаційних технологій у різні сфери людської діяльності в Україні: стрімке зростання обігу пластикових карток, майбутнє введення електронних паспортів та медичних карт, студентських квитків та залікових книжок; зрештою все більше державних установ та приватних підприємств переходять на електронний документообіг, який до того ж, вимагає юридичної чинності підпису фізичної або юридичної особи. Розповсюдження таких технологій також, безперечно, вимагає добре поставленого захисту інформації. Усі ці та багато інших задач покликані вирішувати різноманітні технології захисту інформації.

Враховуючи актуальність даної проблеми, представлена програма вирішує це за допомогою нескладно алгоритма шифрування, який написаний у середовищі JAVA і має можливість адаптовуватись під будь-який пристрій незалежно від програмного забезпечення.

1. АНАЛІЗ ТЕХНІЧНОГО ЗАВДАННЯ

Згідно з завдання необхідно розробити алгоритм шифрування повідомлень в розподілених мережах зв'язку на основі симетричного блочного алгоритму AES, використавши при цьому мову програмування Java та врахувати те, щоб дани алгоритм працював на будь-якому пристрої незалежно від операційної системи.

- Об'єктом дослідження є найпоширеніші алгоритми шифрування і мова програмування Java.
- Предметом дослідження є існуючі криптографічні системи, можливості об'єктно орієнтованої мови програмування Java та новий метод оцінки криптографічних систем.
- Мета роботи полягає у розробці нового методу шифрування для вирішення проблем захисту інформації

2.ПРОБЛЕМИ ТА РІШЕННЯ ЗАХИСТУ ІНФОРМАЦІЇ

Галузь знань про шифри, методи їх побудови та розкриття називається криптографією. Властивість шифру протистояти розкриттю називається криптостійкістю або надійністю і звичайно визначається складністю алгоритму дешифровки.

У практичній криптографії криптостійкість шифру оцінюється з економічних міркувань. Якщо розкриття шифру коштує більше, за саму зашифровану інформацію, то шифр вважається достатньо надійним.

2.1 Поняття шифрування

Шифрування – це оборотне перетворення даних, з метою приховування інформації[1].

Шифрування відбувається із застосуванням криптографічного ключа.

Ключ – це певна кількість символів, сформованих вільним чином з символів, що доступні у системі шифрування.

Шифрування – це кодування даних з метою захисту від несанкціонованого доступу. Процес кодування називається шифруванням, а процес декодування – розшифруванням. Саме кодоване повідомлення називається шифрованим, а застосований метод називається шифром [1].

Основна вимога до шифру полягає в тому, щоб розшифрування було можливе тільки при наявності санкцій, тобто деякої додаткової інформації (або пристрою), які називаються ключем шифру. Процес декодування шифровки без ключа називається дешифруванням.

Симетричні криптосистеми (симетричне шифрування) – спосіб шифрування, в якому для шифрування та дешифрування використовується один й той самий криптографічний ключ[1].

Ключ шифрування має зберігатись у секреті обома сторонами та має бути обраним до початку обміну повідомленнями.

Процес розшифрування заключається в тому, щоби ще раз скласти шифровану послідовність з тією самою гамою шифру.

Описаний метод має суттєвий недолік. Якщо відома хоча б частина вихідного повідомлення, то все повідомлення може бути легко дешифроване.

Більшість симетричних шифрів використовують складну комбінацію великої кількості підстановок та перестановок. Багато таких шифрів виконуються у декілька (до 100) проходів, використовуючи на кожному проході ключ проходу. Множина «ключів проходів» для всіх проходів називається розкладом ключів. Зазвичай, він утворюється з ключа шляхом виконання над ним певних операцій, в тому числі перестановок та підстановок.

Найважливішими параметрами всіх алгоритмів симетричного шифрування є :

- стійкість;
- довжина ключа;
- кількість раундів;
- довжина блоку, який оброблюється;
- складність апаратно/програмної реалізації;
- складність перетворень;

До переваг симетричної системи можна віднести:

- порівняно високу швидкість (приблизно на 3 порядки вищу ніж у асиметричних систем);
- простота реалізації (за рахунок більш простих операцій);
- менша необхідна довжина ключа для відповідної стійкості;

Але є також суттєві недоліки, які практично призводять до того, що дана система майже не використовується на даний час.

- складність керування ключами у великій мережі. Це означає квадратичне збільшення кількості ключів, які необхідно генерувати, зберігати, передавати та знищувати у мережі. Для мережі з 10 абонентів потрібно 45 ключів, для 100 – вже 4950, для 1000 – 499500;

- складність обміну ключами. Для застосування симетричної системи необхідно вирішити проблему надійної передачі ключів до кожного абонента, тому що необхідний секретний канал для передачі кожного ключа обом сторонам.

Криптографічна система з відкритим ключем (або асиметрична криптосистема, асиметричне шифрування), яка вказана на рис. 2.1 – це система шифрування, при якій відкритий ключ передається по відкритому (тобто не захищеному) каналу з'язку та використовується для шифрування повідомлень. Для розшифрування повідомлень використовується секретний (або приватний) ключ [1].

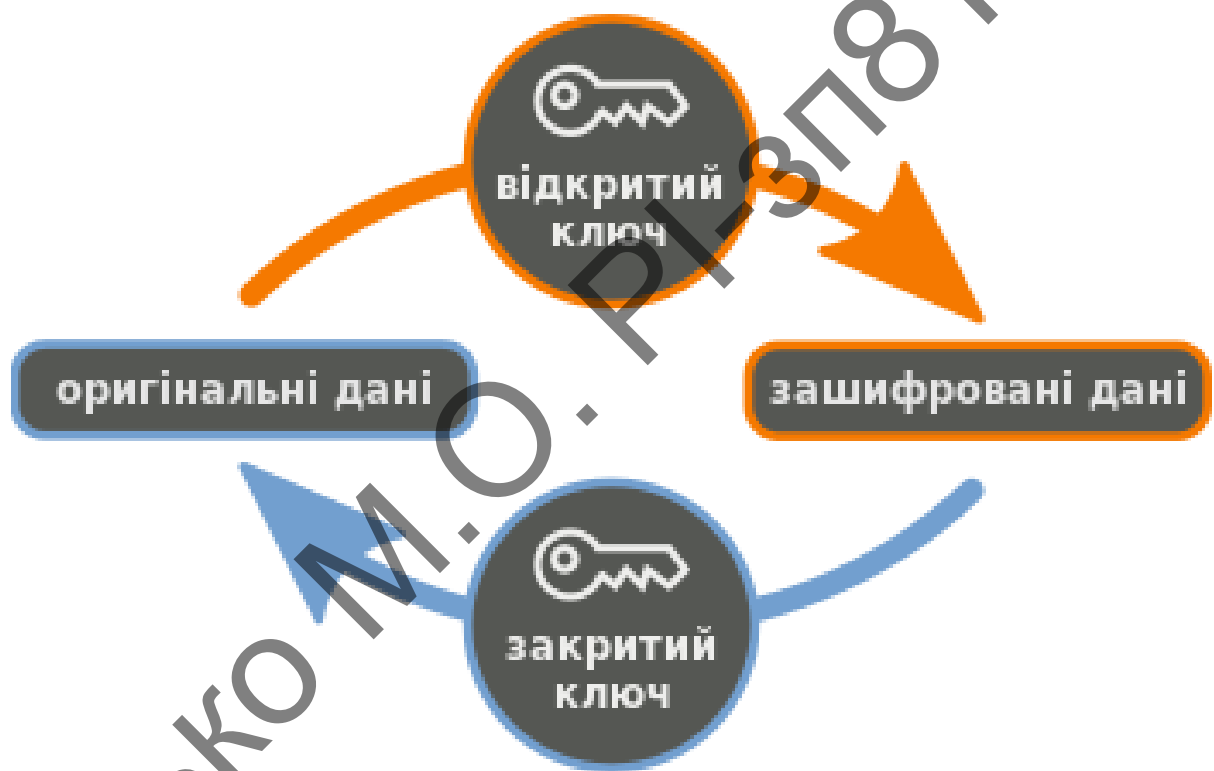


Рисунок 2.1 – Ілюстрація асиметрії

Наявність двох ключів – відкритого та закритого – й робить цю систему асиметричною. Відкритий ключ розсилається всім, хто бажає відправляти повідомлення адресату, а приватний ключ зберігається адресатом і не повинен нікому відправлятися. Навіть якщо знати відкритий ключ та все відправлене розшифроване повідомлення, неможливо знайти приватний ключ [1].

До переваг асиметричної криптосистеми можна віднести :

- не потрібно передавати закритий ключ будь-якими каналами зв'язку;
- - на противагу симетричній криптосистемі, секретний ключ зберігається тільки у одній стороні;
- у симетричній криптосистемі варто змінювати ключ після кожного сеансу передачі даних, у асиметричній ключі можна тримати незмінними достатньо довгий час;
- у більшості мереж кількість ключів при асиметричному шифруванні набагато менша, ніж при симетричному.

Недоліки:

- хоча повідомлення шифруються надійно, але самі сторони «засвічуються» самим фактом передачі (на чому може бути побудована атака);
- асиметричні алгоритми використовують набагато довші ключі ніж симетричні;
- у чистому вигляді асиметричні системи вимагають значних обчислювальних ресурсів, тому на практиці вони частіше використовуються разом з іншими алгоритмами.

Також асиметрична система з відкритим ключем використовується у системах цифрових підписів. Для генерації ЕЦП і для розшифровки повідомлення використовується *секретний ключ*. Криптографічні системи з відкритим ключем в даний час широко застосовуються в різних мережевих протоколах, зокрема, в протоколах TLS і його попереднику SSL (що лежать в основі HTTPS), в SSH. Також використовується в PGP, S / MIME [1].

2.2 Класичні алгоритми шифрування даних

Під захистом інформації іноді розуміється тільки частина з можливих і необхідних заходів у цьому напрямку, зв'язаних із профілем роботи конкретного колективу виконавців [1] .

Вірніше розуміти під цим терміном комплекс заходів, проведених з метою запобігання витоку, розкрадання, утрати, несанкціонованого знищення,

перекручування,	модифікації	(підробки),	несанкціонованого	копіювання,	Лист
Изм.	Лист	№ докум	Подпись	Дата	9

Plzn81.464419.001 ПЗ

блокування інформації і т.д. Оскільки, наприклад, втрата інформації може відбуватися по чисто «технічним», об'єктивним і ненавмисним причинам, під це визначення попадають також і заходи, пов'язані з підвищенням надійності сервера через відмовлення або збої в роботі вінчестерів, недоліків у використовуваному програмному забезпеченні і т.д.[1].

Взагалі, перехід від роботи на персональних комп'ютерах до роботи в мережі ускладнює захист інформації з наступних причин:

- велике число користувачів у мережі і їхній перемінний склад;
- захист на рівні імені і пароля користувача недостатній для запобігання входу в мережу сторонніх осіб;
- велика довжина мережі і наявність багатьох потенційних каналів проникнення в мережу;
- недоліки в апаратному і програмному забезпеченні;
- переваги організаційних засобів – можливість рішення багатьох різномірних проблем, простота реалізації, можливість швидкого реагування на небажані дії в мережі, необмежені можливості модифікації і розвитку;
- недоліки — висока залежність від суб'єктивних факторів, у тому числі від загальної організації роботи в даному конкретному підрозділі;
- шифрування даних являє собою різновид програмних засобів захисту інформації і має особливе значення на практиці як єдиний надійний захист інформації, переданої по протяжних послідовних лініях. [1]

Поняття «шифрування» часто вживається в зв'язку з більш загальним поняттям криптографії. Криптографія включає шифрування і додатково розглядає способи рішення проблем, зв'язаних з можливою підміною цифрових даних: як перевірити вірогідність цифрових даних і як за аналогією з рукописним підписом на папері проставити візу на електронних документах, маючи в розпорядженні лише послідовності нулів і одиниць[1].

Ці проблеми і способи їхнього рішення розглянуті далі.

Число використовуваних програм шифрування обмежено, причому частина з них є стандартами де-факто або де-юре. Однак навіть якщо алгоритм шифрування не являє собою секрету, зробити дешифрування (або розшифровку) без знання закритого ключа надзвичайно складно. Ця властивість у сучасних програмах шифрування забезпечується в процесі багатоступінчастого перетворення вихідної відкритої інформації («*plain text*» в англійській літературі) з використанням ключа (або двох ключів — по одному для шифрування і дешифрування). В кінцевому рахунку кожний з використовуваних складних методів (алгоритмів) шифрування являє собою комбінацію простих методів[1].

Розрізняють наступні класичні алгоритми шифрування:

- підстановка (проста - одноalfавітна, багатоalfавітна однопетльова, багатоalfавітна багатопетльова);
- перестановка (проста, ускладнена);
- гамування (змішування з короткою, довгою або необмеженою маскою). [1]

Стійкість кожного з перерахованих методів до дешифрування без знання ключа характеризується кількісно за допомогою показника S мінімального обсягу, що представляє собою обсяг зашифрованого тексту, який може бути дешифрований за допомогою статистичного аналізу.

Підстановка припускає використання альтернативного алфавіту (або декількох алфавітів), замість вихідного алфавіту[1].

У випадку простої підстановки для символів англійського алфавіту можна запропонувати шифрування, наприклад, що використовує заміну.

Тоді слово «*cache*» у зашифрованому виді представляється як «*usuxk*»

Існує, однак, можливість дешифрування повідомлення за допомогою відомої статистичної частоти повторюваності символів у довільному, досить довгому тексті[1].

Наприклад, символ *E* зустрічається частіше за інші — у середньому 123 рази на кожні 1000 символів або в 12,3% випадків далі за частотою використання символи *T* - 9,6%, *A* - 8,1%, *P* - 7,9%, *N* - 1,2%, *I* - 7,2%, *S* - 6,6%, *R* — 6,0%, *H* - 5,1%, *L* — 4,0% і т.д. Приведені цифри можуть, звичайно, трохи варіюватися в залежності від джерела інформації. Тому показник стійкості до дешифрування *SKB* в даному випадку не перевищує 20...30[2].

При багатоалфавітній підстановці можна домогтися того, що в зашифрованому тексті всі символи будуть зустрічатися приблизно з однаковою частотою, що істотно утруднить дешифрування без знання альтернативних алфавітів і порядку, у якому вони використовувалися при шифруванні[1].

Цифровий ключ складається з неповторюваних цифр, а відповідне йому ключове слово - з неповторюваних символів. Вихідний текст (*plain text*) записується під ключем построчно. Зашифроване повідомлення (*cipher text*) виписується по стовпцях у тім порядку, як це передбачають цифри ключа (або в тім порядку, у якому розташовані окремі символи ключового слова в алфавіті).

Гамування (змішування з маскою) засновано на побітному додаванні по модулю 2 (відповідно до логіки ВИКЛЮЧАЮЧЕ АБО) вихідного повідомлення із заздалегідь обраною двійковою послідовністю (маскою)[1].

Компактним представленням маски можуть служити числа в десятковій системі числення або деякий текст (у цьому випадку розглядаються внутрішні коди символів - для англійського тексту таблиця ASCII).

Операція підсумовування по модулі 2 (ВИКЛЮЧАЮЧЕ АБО) є оборотною, так що при додаванні з тією же маскою (ключем) зашифрованого повідомлення ми знову одержимо вихідний текст (відбудеться дешифрування)[1].

Як маску (ключ) можуть використовуватися константи типу *n* або *e*, і тоді маска буде мати кінцеву довжину. Найбільшу стійкість до дешифрування може забезпечити використання маски з нескінченною довжиною, що утворена генератором випадкових (точніше, псевдовипадкових) послідовностей.

Такий генератор легко реалізується апаратними або програмними засобами, наприклад, за допомогою зсуваючого регістра зі зворотними зв'язками, що використовується при обчисленні перешкодостійкого циклічного коду. У випадку простої підстановки для символів англійського алфавіту можна запропонувати шифрування, наприклад, що використовує заміну[1].

Тоді слово «*cache*» у зашифрованому виді представляється як «*usuxk*»

Існує, однак, можливість дешифрування повідомлення за допомогою відомої статистичної частоти повторюваності символів у довільному, досить довгому тексті. Наприклад, символ *E* зустрічається частіше за інші — у середньому 123 рази на кожні 1000 символів або в 12,3% випадків далі за частотою використання символи *T* - 9,6%, *A* - 8,1%, *P* - 7,9%, *N* - 1,2%, *I* - 7,2%, *S* - 6,6%, *R* — 6,0%, *H* - 5,1%, *L* — 4,0% і т.д. Приведені цифри можуть, звичайно, трохи варіюватися в залежності від джерела інформації. Тому показник стійкості до дешифрування *SKB* в даному випадку не перевищує 20...30.

При багатоалфавітній підстановці можна домогтися того, що в зашифрованому тексті всі символи будуть зустрічатися приблизно з однаковою частотою, що істотно утруднить дешифрування без знання альтернативних алфавітів і порядку, у якому вони використовувалися при шифруванні[1].

Цифровий ключ складається з неповторюваних цифр, а відповідне йому ключове слово - з неповторюваних символів. Вихідний текст (*plain text*) записується під ключем построчно. Зашифроване повідомлення (*cipher text*) виписується по стовпцях у тім порядку, як це передбачають цифри ключа (або в тім порядку, у якому розташовані окремі символи ключового слова в алфавіті).

Гамування (змішування з маскою) засновано на побітному додаванні по модулю 2 (відповідно до логіки ВИКЛЮЧАЮЧЕ АБО) вихідного повідомлення із заздалегідь обраною двійковою послідовністю (маскою)[1].

Компактним представленням маски можуть служити числа в десятковій системі числення або деякий текст (у цьому випадку розглядаються внутрішні коди символів - для англійського тексту таблиця ASCII).

Операція підсумовування по модулю 2 (ВИКЛЮЧАЮЧЕ АБО) є оборотною, так що при додаванні з тією же маскою (ключем) зашифрованого повідомлення ми знову одержимо вихідний текст (відбудеться дешифрування).

Як маску (ключ) можуть використовуватися константи типу n або e , і тоді маска буде мати кінцеву довжину. Найбільшу стійкість до дешифрування може забезпечити використання маски з нескінченною довжиною, що утворена генератором випадкових (точніше, псевдовипадкових) послідовностей. Такий генератор легко реалізується апаратними або програмними засобами, наприклад, за допомогою зсуваючого регістра зі зворотними зв'язками, що використовується при обчисленні перешкодостійкого циклічного коду. Точне відтворення псевдовипадкової послідовності в генераторі на прийомному кінці лінії забезпечується при установці такого ж вихідного стану (вмісту зсуваючого регістра) і тієї ж структури зворотних зв'язків, що й у генераторі на передавальному кінці[1].

Перераховані класичні методи шифрування (підстановка, перестановка і гамування) є лінійними в тім змісті, що довжина зашифрованого повідомлення дорівнює довжині вихідного тексту. Можливо нелінійне перетворення типу підстановки замість вихідних символів (або цілих слів, фраз, речень) заздалегідь обраних комбінацій символів іншої довжини[1].

Стандартні методи шифрування (національний або міжнародні) для підвищення ступеню стійкості до дешифрування реалізують кілька етапів (кроків) шифрування, на кожному з яких використовуються різні класичні методи шифрування відповідно до обраного ключа (або ключей). Існують дві принципово різні групи стандартних методів шифрування:

- шифрування з використанням тих самих ключів (шифрів) при шифруванні і дешифруванні (симетричне шифрування або системи з відкритими ключами - *private-key systems*);
- шифрування з використанням відкритих ключів для шифрування і закритих - для дешифрування (несиметричне шифрування).

Строгий математичний опис алгоритмів, використовуваних у стандартних методах шифрування, занадто складний. Для користувачів важливі в першу чергу споживчі властивості різних методів (ступінь стійкості до дешифрування, швидкість шифрування і дешифрування, порядок і зручність поширення ключів), що і розглядаються нижче[1].

Стандартні методи шифрування.

Стандарт шифрування даних США *DES (Data Encryption Standard)* — стандарт шифрування даних) відноситься до групи методів симетричного шифрування і прийнятий у 1976 р. Використовувані операції - перестановка, гамування і нелінійна підстановка. Число кроків — 16. Довжина ключа складає 56 біт, з яких 8 біт - це перевіірочні розряди парності/непарності[1].

Довгий час ступінь стійкості до дешифрування цього методу вважалася достатньою, однак у 1998 р. з'явилося повідомлення про створення спеціалізованого комп'ютера (*DES cracker*), що дозволяє розкрити зашифрований текст максимум за 9 днів[1].

Аналогом *DES* є ДСТ28147-89, але з довжиною ключа 256 біт, так що його ступінь стійкості до дешифрування істотно вище. Важливо також і те, що в даному випадку передбачається ціла система захисту, що переборює «родовий» недолік симетричних методів шифрування - можливість підміни повідомлень. Такі удосконалення, як імітовставки, хеш-функції й електронні цифрові підписи дозволяють «авторизувати» передані повідомлення.

Достоїнством симетричних методів шифрування є висока швидкість шифрування і дешифрування, недоліком — малий ступінь захисту у випадку, якщо ключ став доступний третій особі[1].

Досить популярні, особливо при використанні електронної пошти в мережі *Internet*, несиметричні методи шифрування або системи з відкритими ключами - *public-key systems*. Типовий представник цієї групи методів - *PGP (Pretty Good Privacy)* - забезпечує досить високу таємність).

Кожен користувач у цьому випадку має пари ключів. Відкриті ключі призначені для шифрування і вільно розсилаються по мережі, але не

дозволяють зробити дешифрування. Для цього потрібні секретні (закриті) ключі[1].

Принцип шифрування в даному випадку ґрунтується на використанні так званих однобічних функцій. Пряма функція $x \rightarrow f(x)$ легко обчислюється на підставі відкритого алгоритму (ключа). Зворотне перетворення $f(x) \rightarrow x$ без знання закритого ключа утруднене і повинне займати досить великий час, що і визначає ступінь труднощів при обчисленні однобічної функції [2].

Ідею системи з відкритими ключами можна пояснити в такий спосіб. Для шифрування повідомлень можна використовувати звичайну телефонну книгу, у якій імена абонентів розташовані за абеткою і передують телефонним номерам. Береться ім'я абонента, що починається на дану букву вихідного слова, і номер телефону використовується як шифроване повідомлення. Зрозуміло, що в користувача є можливість вибору відповідності між символом у вихідному тексті й ім'ям абонента, тобто це багатоалфавітна система, що підвищує її ступінь стійкості до дешифрування. Легальний користувач має «зворотний» телефонний довідник, у якому в першому стовпці розташовуються телефонні номери по зростанню, і легко робить дешифрування. Якщо ж такого «зворотного» довідника немає, то користувач повинен багаторазово переглядати доступний прямий довідник у пошуках потрібних телефонних номерів, що є стомлюючою процедурою. Це і є практична реалізація функції труднощів обчислення.

Сам по собі метод шифрування на основі телефонних довідників навряд чи перспективний хоча б через те, що ніхто не заважає потенційному зломщику скласти «зворотний» телефонний довідник. Однак у використовуваних на практиці методах шифрування даної групи в аспекті надійності захисту все гаразд[1].

На відміну від симетричних методів шифрування, проблема розсилання ключів у несиметричних методах вирішується простіше — пари ключів (відкритий і закритий) генеруються «на місці» за допомогою спеціальних програм[1].

Для розсилання відкритих ключів використовуються спеціальні технології типу *LDAP (Lightweight Directory Access Protocol* - протокол полегшеного доступу до довідника). Ключі, що розсилаються, можуть бути попередньо зашифровані за допомогою одного із симетричних методів шифрування [3].

2.3 Переваги Java над іншими мовами

Мова C++ з появою перших трансляторів знайшла відразу ж дуже широке розповсюдження, на ній було створено величезну кількість програм і застосунків. У міру накопичення досвіду створення великих програмних систем, а особливо в захисті інформації, спливали недоліки, які спонукали до пошуку альтернативних рішень. Таким альтернативним рішенням стала мова Java [4].

Java є типобезпечною мовою. Типобезпека гарантує відсутність у програмах помилок, які важко знайти і які пов'язані з неправильною інтерпретацією пам'яті комп'ютера. Це робить процес розробки надійнішим і передбачуваним, а отже швидшим. Так само це дозволяє залучати до розробки програмістів, що мають меншу кваліфікацію і мати великі групи розробників.

Java-код компілюється спочатку не в машинний код, а в певний проміжний код, який надалі інтерпретується або компілюється, тоді як багато C++ компіляторів орієнтовані на компіляцію в машинний код заданої платформи.

У мові Java є чіткі певні стандарти на введення-виведення, графіку, геометрію, діалог, доступ до баз даних і інших типових застосувань. Завдяки цим особливостям, застосунки на Java мають значно кращу кросплатформність, ніж C++, і часто, будучи написані для певного комп'ютера і операційної системи, працюють під іншими системами без змін. Програмісти, що пишуть на мові Java, не залежать від пакунків, нав'язаних розробниками компіляторів на дане конкретне середовище, що різко спрощує портування програм.

У мові Java реалізовано повноцінне збирання сміття, якого немає в C++. Немає в C++ і засобів перевірки правильності вказівників. З іншого боку, C++ володіє набором засобів (конструктори і деструктори, стандартні шаблони,

посилання), що дозволяють майже повністю виключити виділення і звільнення пам'яті вручну і небезпечні операції з вказівниками. Проте таке виключення вимагає певної культури програмування, тоді як в мові Java воно реалізується автоматично.

За даними компанії Oracle, програми на Java запускаються на 3 млрд девайсів. Це маркетингове повідомлення складно перевірити. Проте Java широко використовується і входить в число найбільш затребуваних мов, це не викликає сумніву.

Наприклад, переважна більшість великих компаній так чи інакше використовують Java. Дуже багато серверних додатків для корпорацій написані на цій мові. Наприклад, мова йде про програми для фінансових організацій, які забезпечують проведення транзакцій, фіксацію торгових операцій.

На Java написано багато веб-додатків. Популярні фреймворки, в тому числі Spring, Struts, JSP, використовуються для створення різних додатків в інтернеті: від ecommerce-проектів до великих порталів, від освітніх платформ до урядових ресурсів.

Популярна комп'ютерна гра Minecraft написана на Java.

Мобільна розробка - ще одна область використання Java. Цією мовою пишуть програми для пристроїв, що працюють під управлінням ОС Android.

На Java створюють клієнтські програми. Простий і близький розробникам приклад: IDE NetBeans написана на «Джаві».

Також Java застосовується для роботи з Big Data, розробки програм для наукових цілей, наприклад, обробки природних мов, програмування приладів - від побутових девайсів до промислових установок.

Тобто на Java можна писати різні типи додатків: веб, мобільний і десктопний софт, ігри і так далі. Традиційно у цієї мови сильні позиції в промисловому програмуванні, в сегменті великих компаній (т.зв. ентєрпрайз).

При написанні прикладної програми, саме використання мови Java, дає можливість втілити в життя практичність цього додатка, оскільки його можна використовувати на будь-якому терміналі. При цьому не вносячи ніяких

поправок в код. Також є можливість використання на смартфонах на базі Android.

2.4 Середовище розробки

Розробка та налагодження програмного продукту відбувалося в IDE IntelliJ.[5]

IntelliJ - одна з найпотужніших і популярних інтегрованих середовищ розробки (IDE) для Java. Ця багатфункціональна IDE забезпечує швидку розробку і допомагає поліпшити якість коду.

IDE розшифровується як інтегроване середовище розробки. Це комбінація кількох інструментів, які роблять процес розробки програмного забезпечення більш простим, надійним і менш схильним до помилок. Він має наступні переваги в порівнянні з текстовим редактором:

- Інтеграція з корисними інструментами, такими як компілятор, відладчик, система контролю версій, інструменти збірки, різні платформи, профіліровщики додатків і так далі;
- Підтримує функції навігації по коду, автозаповнення коду, рефакторінга і генерації коду, що прискорює процес розробки;
- Підтримує модульне тестування, інтеграційне тестування і покриття коду за допомогою плагінів;
- Надає багатий набір плагінів для подальшого розширення функціональності IDE.

IntelliJ IDEA дійсно розуміє і глибоко розуміє ваш код, а також контекст кодера, що робить його таким унікальним серед інших Java IDE.

Інтелектуальне завершення коду - підтримує контекстне завершення коду. Він дає список найбільш значущих символів, які можна застосувати в поточному контексті.

Ланцюгове завершення коду - це розширена функція завершення коду, яка перераховує відповідні символи, доступні через методи або методи отримання в поточному контексті.

Завершення статичного члена - дозволяє використовувати статичні методи або константи і автоматично додає необхідні оператори імпорту, щоб уникнути помилки компіляції

Виявлення дублікатів - Він виявляє фрагменти дубльованого коду на льоту і повідомляє / пропозицію про це користувачеві.

Інспекції та швидкі виправлення. Всякий раз, коли IntelliJ виявляє, що ви збираєтеся зробити помилку, в одному рядку з'являється невелике лампове повідомлення. Натиснувши на неї, ви побачите список пропозицій.

3. ОПИС ЗОВНІШНЬОЇ ЧАСТИНИ ПРОГРАМИ. ГРАФІЧНИЙ ІНТЕРФЕЙС

3.1 Графічний інтерфейс

Графічний користувальницький інтерфейс або просто названий “GUI” - загальний термін, що використовується у світі програмного забезпечення. Графічний інтерфейс представляє програму, яка має візуальний дисплей для користувача з простими у користуванні елементами управління.[6] Графічний інтерфейс зазвичай складається з графічних компонентів, таких як вікна, рамки, кнопки, мітки тощо.

Ми можемо використовувати ці компоненти для взаємодії із системою чи навіть зовнішнім світом. Java пропонує безліч API та багаторазових класів, за допомогою яких ми можемо розробляти графічні програми

Зверніть увагу, що писати графічний інтерфейс не так просто, як малювати деякі фігури або включати зображення. Графічний інтерфейс містить послідовність дій, які також запускають деякі події, які, у свою чергу, виконують деякі дії при виклику компонента або частини компонента, як, натиснувши кнопку, ми запускаємо деякі дії.

Отже, додаток графічного інтерфейсу - це структура, що складається з графічних компонентів та подій, які можуть запускатися на цих компонентах, та дій, які виконуються в результаті активації подій.

При написанні зовнішньої оболонки програми мовою Java, в середовищі розробки IDE IntelliJ було використано можливості бібліотеки Swing. За допомогою цих інструментів побудовано графічний інтерфейс прикладного додатка. Який є простим та практичним у використанні завдяки готовому оптимізованому коду з даної бібліотеки. Саме графічний інтерфейс дає можливість використовувати рядовому користувачу всі можливості алгоритму не вдаючись в подробиці функціонування.

3.2 Бібліотека Swing

Кожна із сучасних мов програмування надає безліч бібліотек для роботи зі стандартним набором елементів управління . Під бібліотекою в програмуванні розуміють набір готових класів та інтерфейсів, призначених для вирішення певного кола завдань.

У Java є декілька бібліотек візуальних компонентів для створення графічного інтерфейсу користувача. Найбільш рання з них називається AWT. Бібліотека Swing розроблена на базі AWT і змінює більшість її компонентів своїми, спроектованими більш ретельно і зручно.

Swing — інструментарій для створення графічного інтерфейсу користувача (GUI) мовою програмування Java. [7]

Елементи інтерфейса :

- JButton - кнопка;
- JCheckBox - кнопка-прапорець;
- JComboBox - список, що випадає;
- JLabel - мітка, напис;
- JList - список;
- JPasswordField - текстове поле для прихованого введення;
- JProgressBar - компонент для відображення числа в деякому діапазоні;
- JRadioButton - переключателі, радіо-кнопки, зазвичай використовується з компонентом ButtonGroup;
- JSlider - компонент дозволяє вибрати значення із заданого діапазону;
- JSpinner - компонент дозволяє вибрати значення з зазначених послідовностей;
- JTable - таблиця;

- JTextField - однорядкове текстове поле;
- JTextArea - багаторядкове текстове поле;
- JTree - дерево.

Особливості Swing :

- Легка вага - компоненти Swing не залежить від API власної операційної системи, так як елементи управління Swing API відображаються в основному з використанням чистого коду JAVA, а не викликів базової операційної системи.
- Багаті елементи управління - Swing надає багатий набір розширених елементів управління, таких як Tree, TabbedPane, слайдер, палітра кольорів і елементи управління таблицями.
- Широкі можливості налаштування - елементи управління Swing можна легко налаштувати, оскільки зовнішній вигляд не залежить від внутрішнього уявлення.
- Змінний зовнішній вигляд - графічний інтерфейс на основі SWING Зовнішній вигляд програми може змінюватися під час виконання в залежності від доступних значень.

3.3 Концепція додатка

На Рисунку 3.1 зображено головне вікно. Додаток складається з двох частин. У верхній додатковій строці можна отримати хеш код в декількох варіантах.

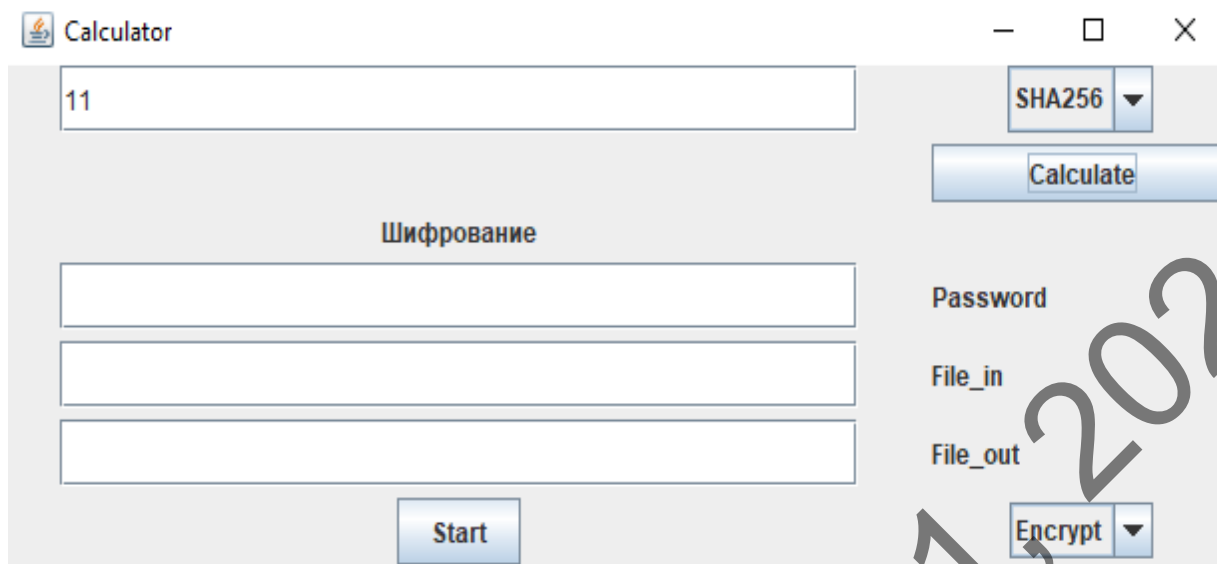


Рисунок 3.1 – Вікно додатка

Інша частина виконує шифрування та дешифрування. Для захисту від не санкціонованого доступу, наявна строка “Password”. При шифруванні ми вигадуємо пароль, а при зворотній операції інший користувач використовує цей пароль для дешифрування.

Для виконання потрібно мати два файли формату .txt. В поле File_in вказуємо шлях до файла в якому знаходиться інформація яку потрібно захистити.

У поле File_out вказуємо шлях до пустого файла в який буде записано зашифроване повідомлення з першого документа.

Вибираємо Encrypt для шифрування, а Decrypt – дешифрування.

На Рисунок 3.2 зображено структуру інтерфейсу розробленого за допомогою JFrame.

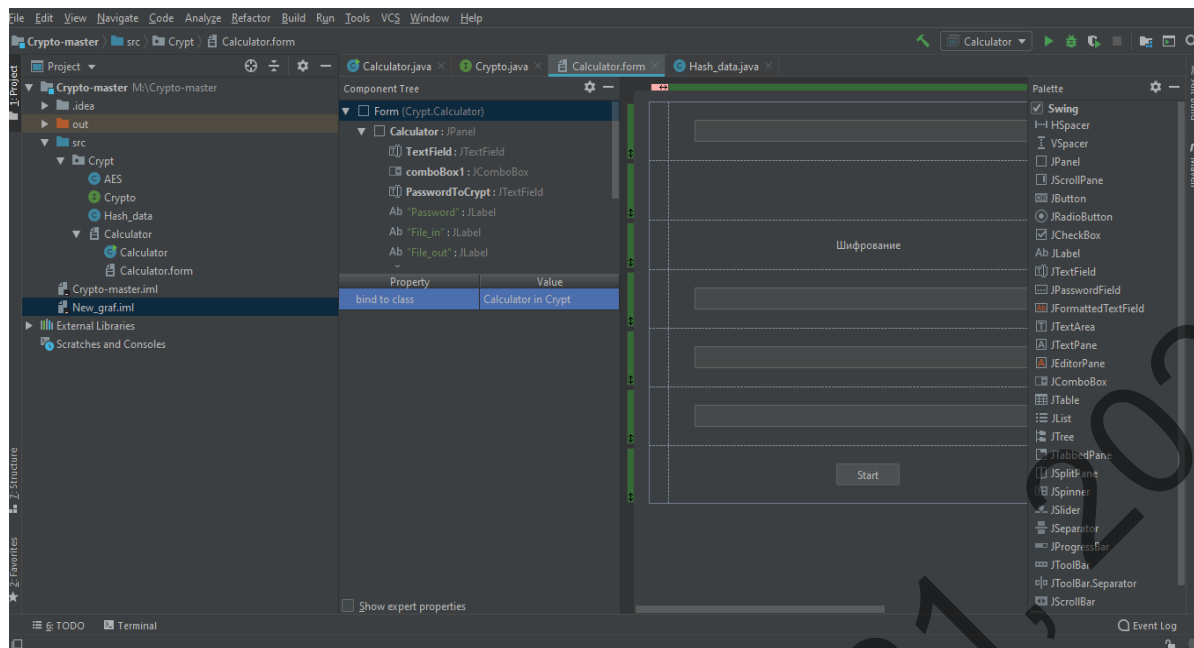


Рисунок 3.2 - структура JFrame

Маючи простий зовнішній вигляд, задіяні майже всі можливості даної бібліотеки.

4. ОПИС ВНУТРІШНЬОЇ ЧАСТИНИ ПРОГРАМИ.

ПРОГРАМНИЙ КОД

Внутрішня частина програми складається з програмного коду. У даному розділі розглянемо структуру методи і точку входу в програму. Описано зв'язки, прикладні методи з коментарями.

4.1 Структура коду

Програмний код включає в себе два класа, один інтерфейс і форму JFrame (рис. 4.1).

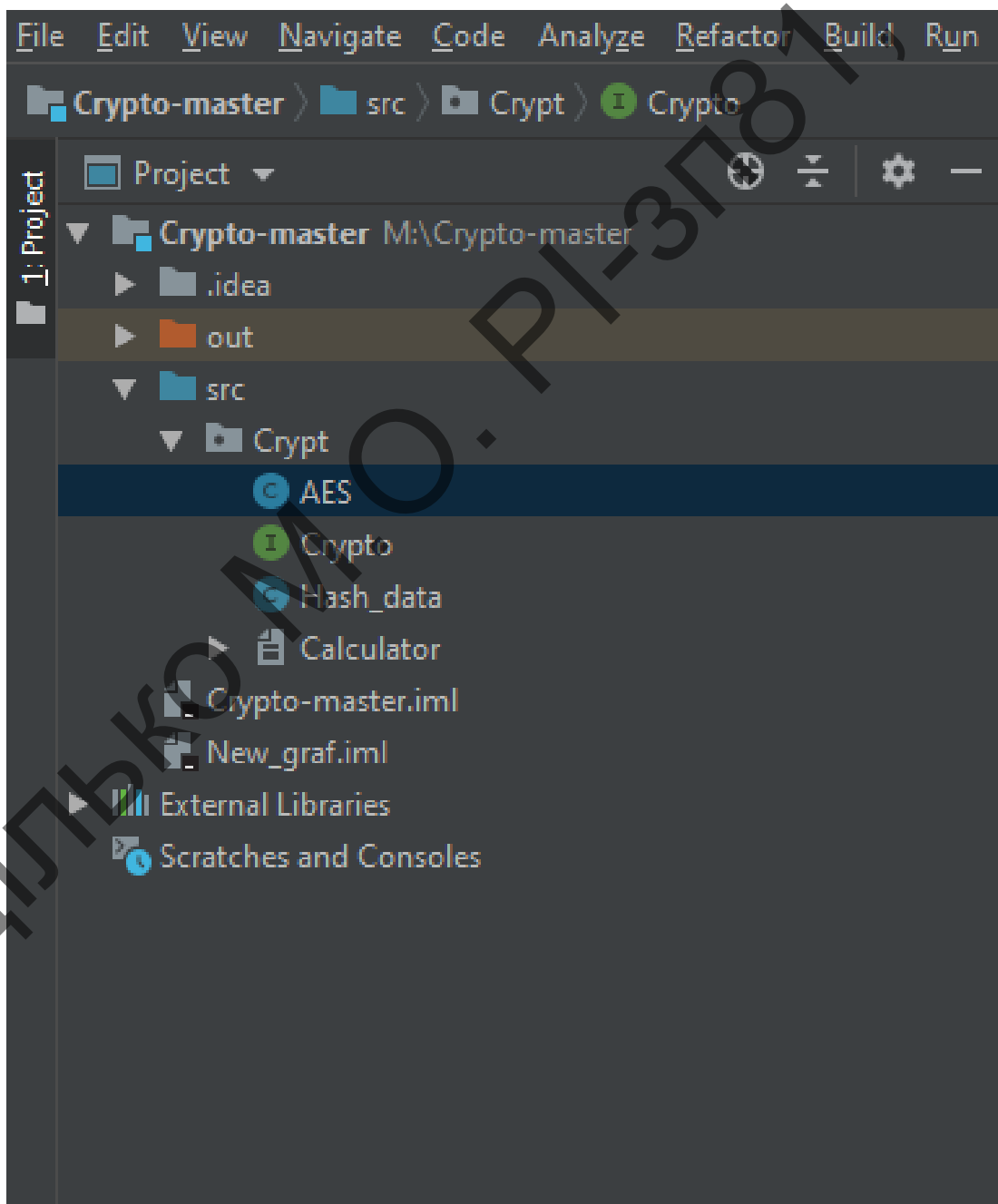


Рисунок 4.1 – Структура коду

Клас — це спеціальна конструкція, яка використовується для групування пов'язаних змінних та функцій. При цьому, згідно з термінологією ООП, глобальні змінні класу (члени-змінні) називаються полями даних (також властивостями або атрибутами), а члени-функції

називаються методами класу. Створений та ініціалізований екземпляр класу називають об'єктом класу. На основі одного класу можна створити безліч об'єктів, що відрізнятимуться один від одного своїм станом (значеннями полів).[8]

Інтерфейс у мові програмування Java — це абстрактний тип, який використовується для визначення поведінки, яку класи повинні реалізовувати. Інтерфейси схожі до протоколів.[8]

Однією з переваг використання інтерфейсів є те, що вони імітують множинне успадкування. Інтерфейси використовуються для описання подібності, яку поділяють різні класи, але які не обов'язково мають класові взаємозв'язки. Використання інтерфейсів — можливість використовувати об'єкт, фактично без наявності інформації про його тип, але про який відомо, що він реалізує певний інтерфейс.

4.2 Клас з алгоритмом AES

```
package Crypt;  
import javax.crypto.Cipher;  
import javax.crypto.spec.SecretKeySpec;  
import java.io.UnsupportedEncodingException;  
import java.security.MessageDigest;  
import java.security.NoSuchAlgorithmException;  
import java.util.Arrays;  
/**
```

*Клас написаний для шифрування даних

* алгоритмом AES в режиме ECB

* @see Calculator

*/

public class AES {

private static SecretKeySpec secretKey;

private static byte[] key;

/**

** Метод для розрахунку ключа шифрування об'єкта*

** @param myKey - включає в себе пароль користувача з*

** якого виходить секретний ключ*

** @see AES*

**/*

private void setKey(String myKey) {

MessageDigest sha = null;

try {

key = myKey.getBytes("UTF-8");

sha = MessageDigest.getInstance("SHA-1");

key = sha.digest(key);

key = Arrays.copyOf(key, 16);

secretKey = new SecretKeySpec(key, "AES");

}

catch (NoSuchAlgorithmException e) {

e.printStackTrace();

}

catch (UnsupportedEncodingException e) {

e.printStackTrace();

/**

** Метод використовується для шифрування даних*

** @param data – змінна включає в себе дані для шифрування*

** @param key - пароль користувача для шифрування*

** @return повертає масив байт уже зашифрованих даних*

**/*

public byte [] Encrypt(byte [] data, String key) {

try

{

```

        setKey(key);
        Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
        cipher.init(Cipher.ENCRYPT_MODE, secretKey);
        //return
        Base64.getEncoder().encodeToString(cipher.doFinal(data.getBytes("UTF-
8"))));
        return cipher.doFinal(data);
    }
    catch (Exception e)
    {
        System.out.println("Error while encrypting: " + e.toString());
    }
    return null;
}
/**

```

* Метод

* @param data – *змінна яка включає в себе дані для розшифровки*

* @param key - *пароль користувача для розшифровки*

* @return *повертає масив байт уже розшифрованих даних*

*/

```

public byte [] Decrypt(byte[] data, String key) {
    try
    {
        setKey(key);
        Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5PADDING");
        cipher.init(Cipher.DECRYPT_MODE, secretKey);
        //return new String(cipher.doFinal(Base64.getDecoder().decode(data)));
        return cipher.doFinal(data);
    }
    catch (Exception e)

```

```

    {
        System.out.println("Error while decrypting: " + e.toString());
    }
    return null;
}
}

```

Даний клас включає в себе повну реалізацію алгоритму шифрування та зворотнього перетворення шифру в дані.

4.3 Суть методу шифрування

Алгоритм складається з двох частин :

- шифрування даних;
- розшифрування інформації за допомогою секретного ключа;

Винайдений новий метод шифрування було виконано на об'єктно орієнтованій мові програмування Java. Суть цієї криптографічної системи полягає в тому, що кожний наступний елемент, який потрібно зашифрувати, шифрується попереднім.

Для прикладу візьмемо повідомлення, яке складається з 5 букв <p> <a> <s> <h> <a> .

Спочатку шифрування починається з кодування символів(в нашому випадку англійського алфавіту), тут використано стандартне кодування в Windows(ASCII). Від А до Z – від 97 до 122.

Шифрування на Java виконується наступним чином . Спочатку кожній букві англійської абетки привласнюють цифри від 97 до 122. Далі пишеться код для виведення повідомлення і створюється поле для введення інформації, а саме для вводу ключа в вигляді цифри.

Схожий код написаний і для вводу повідомлення , після чого виконується цикл, який записує у вказаний файл зашифроване повідомлення по формулі : код букви повідомлення + ключ + попередній код букви (якщо така є). У результаті отримуємо зашифроване повідомлення.

Дешифрування в мові програмування Java виконується в зворотньому порядку при введенні правильного пароля.

Пароль для виконання шифрування та дешифрування не змінний.

Неділько М.О. РІ-3781, 2021

Зм.	Лист	№ докум.	Підпис	Дата

5 ПРОГРАМНА РЕАЛІЗАЦІЯ СИСТЕМИ

Аналізуючи поставлену задачу та деякі методи її вирішення, було вирішено розробити програмну систему як прикладний додаток написаний на Java. Головною перевагою перед іншими варіантами є його універсальність і можливість використання на будь-яких пристроях завдяки JVM. Додатки на Java підтримують всі уснуючі гаджети.

5.1. Java virtual machine

Віртуальна машина Java (JVM) - це програма, призначена для запуску інших програм. В основі лежить проста ідея – написав один раз, запускай всюди.

JVM має дві основні функції:

- Дозволяє запустити Java-додатки на будь-яких пристроях або операційних системах.
- Керує та оптимізує пам'ять, використовує додатки.

У 1995 році, коли Java з'явилася, всі комп'ютерні програми були написані під певними операційними системами та керували пам'яттю, що надходила розробнику програмного забезпечення. Так що поява JVM було революцією.

Існує технічне визначення JVM, а також його основна формулювання:

- Технічне визначення: JVM - це специфікація програмного забезпечення, яке виконує код і надає результати виконання для цього коду
- Попередня формула: JVM - це можливість запуску наших Java-додатків. Ми налаштовуємо параметри JVM, а потім покладаємося на її автоматичне управління ресурсами програм на час виконання.

Глобально JVM складається із трьох частин: специфікація, реалізація та екземпляр. Розглянемо кожен з них.

Перша частина JVM - специфікація, в якій не визначено деталей реалізації JVM, що забезпечує максимальну свободу творчості при її створенні.

JVM - це віртуальна машина, яка запускає Java-програми в портативному режимі.

Термін "Віртуальна машина" означає, що JVM є абстракцією фактичної машини, такою як сервер, на якій працюють програми. Незалежність від операційної системи або технічного забезпечення, JVM створює попередньо визначену середу для запуску в її програмі.

Існує безліч різноманітних реалізацій специфікацій JVM як комерційних, так і з відкритим вихідним кодом. JVM HotSpot від проекту OpenJDK є поетапною реалізацією та містить одну з найбільш перевірених у світі кодових баз. HotSpot також є широко використовуваним JVM.

Майже всі ліцензовані JVM створені, як відповідь на OpenJDK та HotSpot JVM, включаючи ліцензійний JDK від Oracle. Розробники, що створюють ліцензовані продукти на основі OpenJDK, забезпечують мотивацію бажання збільшити продуктивність для певних операційних систем. Зазвичай користувачі завантажують та встановлюють JVM, як частину середовища, виконану Java (JRE).

Після того, як специфікація JVM реалізована та випущена, ви можете завантажити її як додаток. Завантажена програма є екземпляром віртуальних машин.

У більшості випадків, говорячи про JVM, вказують у вигляді екземпляра JVM, який працює в середовищі розробки.

Все в Java є класом, і всі додатки на Java складаються з класів. Пропозиція може статися з одного або тисячі класів. Для запуску додатків JVM слід завантажувати скомпільовані .class-файли в контексті, наприклад, як сервер, де вони будуть доступні. JVM залежить від свого завантажувача класу за час виконання цієї функції.

Завантажувач класів Java є частиною JVM, яка завантажує класи в пам'ять і робить їх доступними для виконання. Завантажувачі класів використовують техніку лінивого завантаження (ледаче завантаження) та кешування, щоб зробити завантаження класів максимально ефективною.

Будь-яка віртуальна машина Java включає в себе завантажувач класів. Специфікація JVM описує стандартні методи для запрошень та управління

завантажувачем у час роботи, але для виконання цих можливостей відповідає конкретна реалізація JVM.

Як тільки завантажувач класів виконав свою роботу, JVM починає виконувати код кожного класу.

Випуск коду включає управління доступом до системних ресурсів. Механізм виконання JVM стоїть між робочими програмами, з її запрошеннями на файлові, мережеві ресурси та ресурси пам'яті та операційну систему, яка забезпечує ці ресурси.

5.2. Користування програмною системою

Запустивши додаток ми бачимо головне вікно (рис. 5.1).

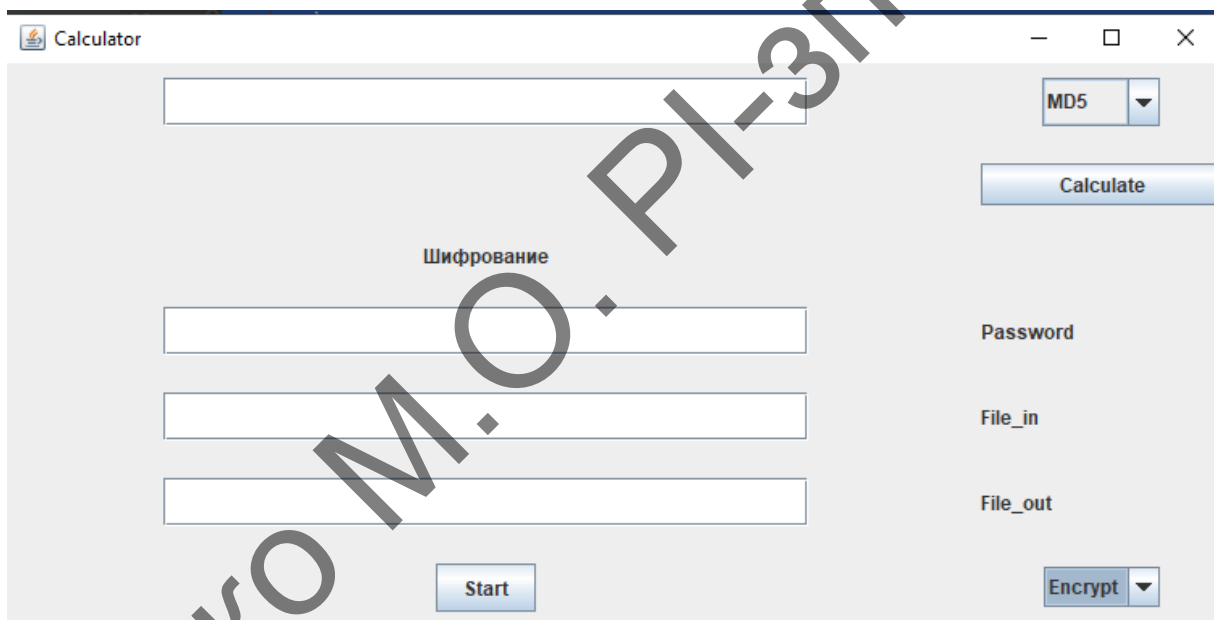


Рисунок 5.1 – Головне вікно додатка

Продемонструємо роботу програми, на видуманій мережі. Наше повідомлення пройде всі етапи тільки виконання буде в текстових файлах.

Маємо три текстові файли, в одному вже є повідомлення (рис.5.2)

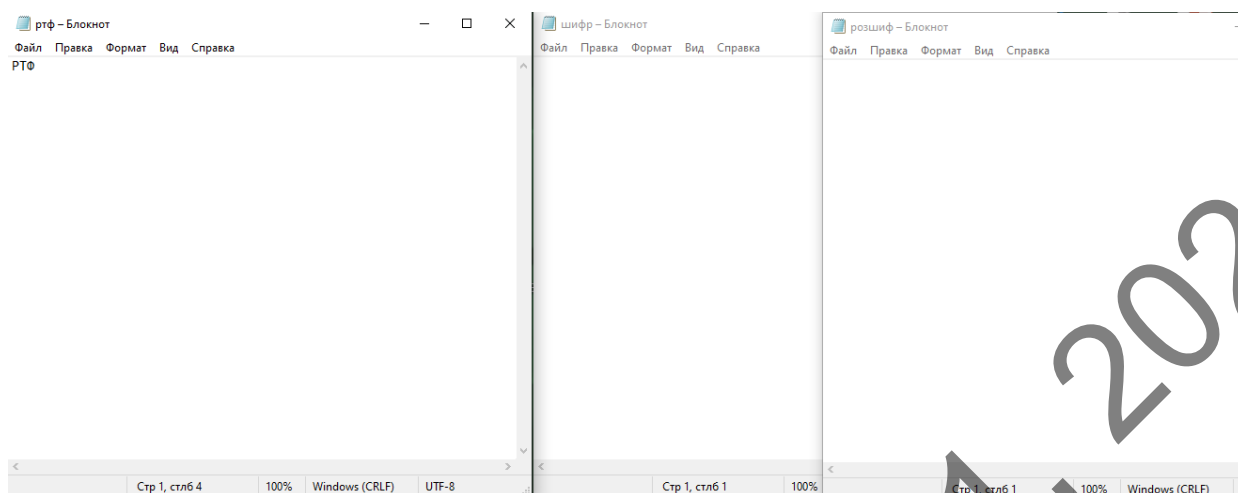


Рисунок 5.2 – Файл з повідомленням

У головному вікні програми вказуємо пароль шифрування. У поле File_in задаємо місце знаходження файлу з повідомленням. У наступне поле вказуємо адресу проміжного файлу, в якому буде зашифроване повідомлення (рис. 5.3)

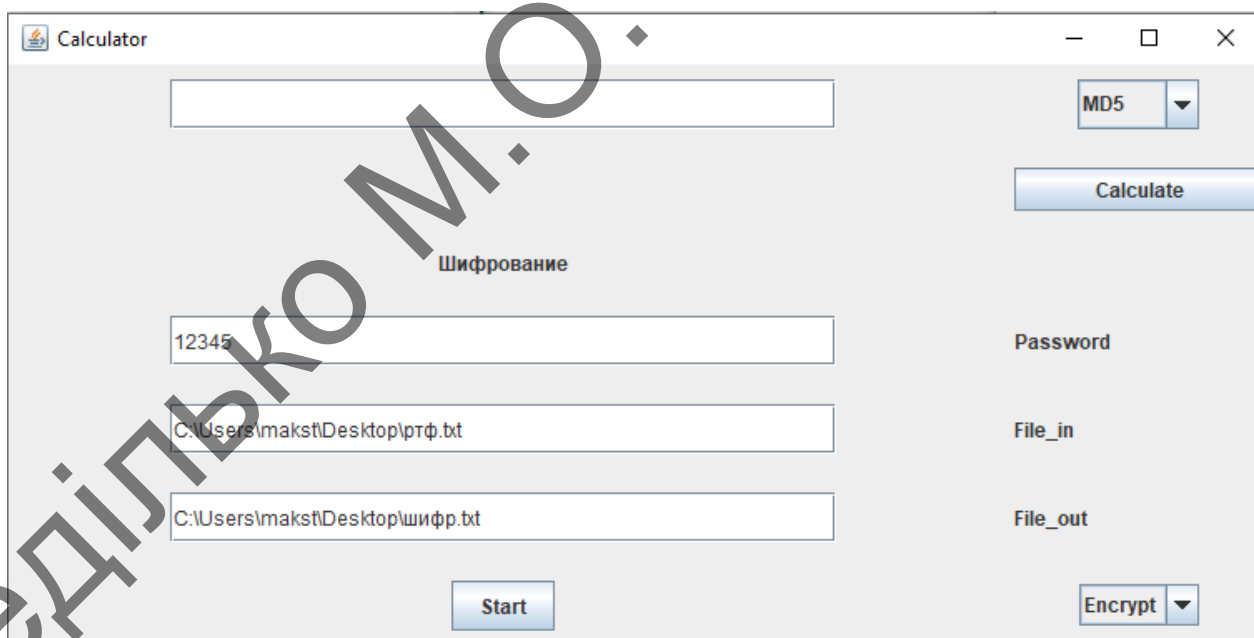


Рисунок 5.3 – Заповнені необхідні поля

Після того як всі поля заповнені (рис. 5.3), виконуємо шифрування повідомлення натиснувши на кнопку *start*.

Отримавши підтвердження (рис. 5.4), перевіряємо результат шифрування в проміжний файл (рис. 5.5)

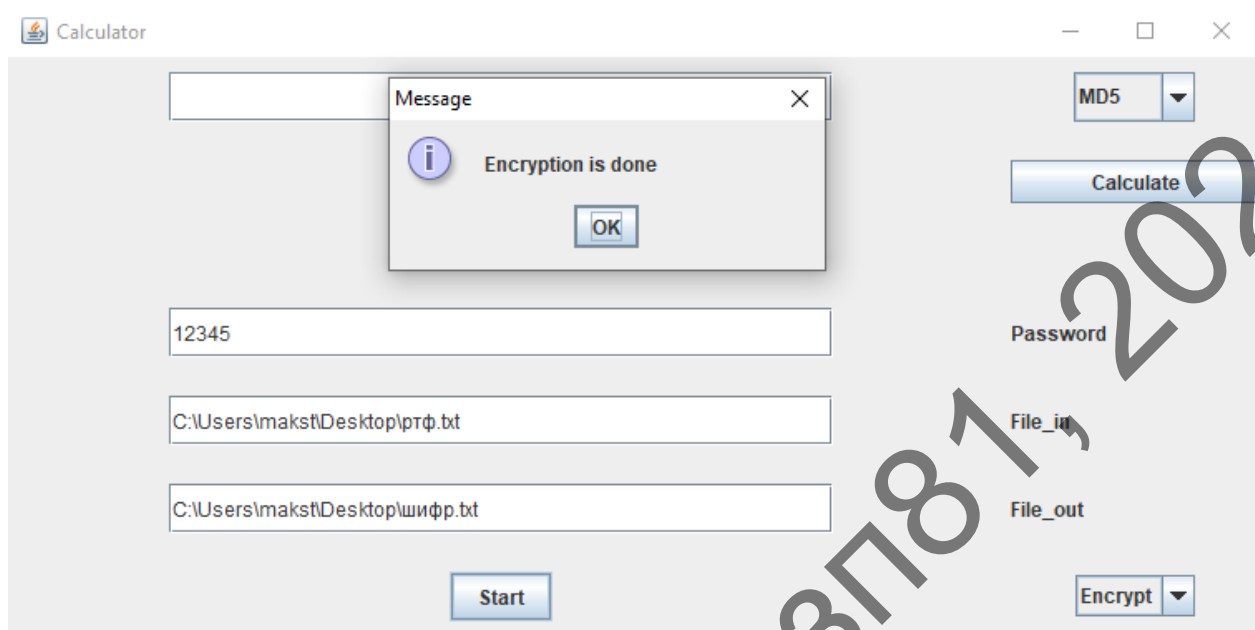


Рисунок 5.4 – Підтвердження шифрування

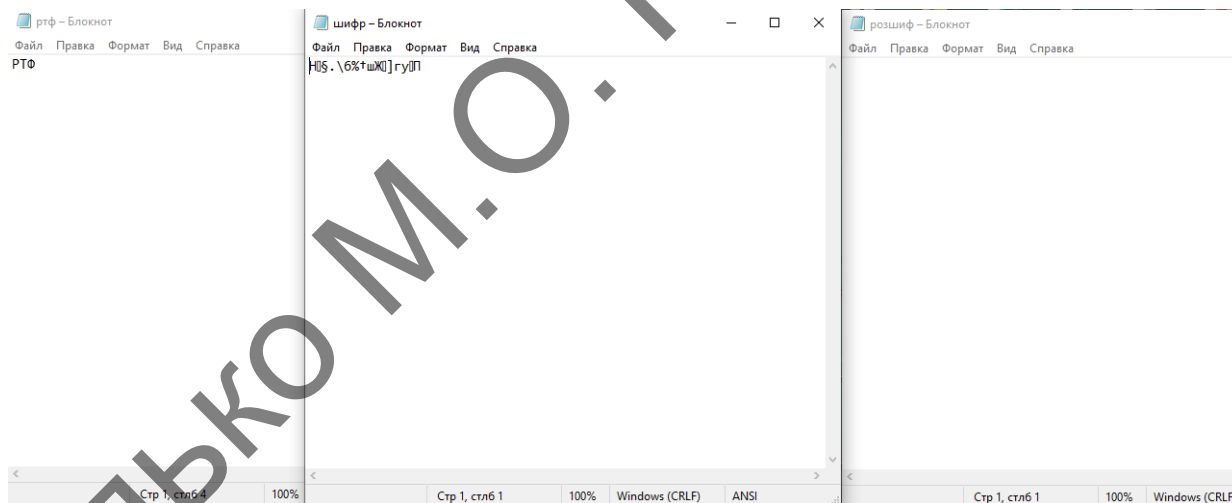


Рисунок 5.5. – Результат шифрування

Алгоритм шифрування виконав свої функції і ми отримали зашифроване повідомлення.

Для того щоб розшифрувати інформацію, потрібно в головному меню програми заповнити всі поля.

На вхід подати файл із зашифрованим повідомленням на вихід новий файл коду буде записано розкодовану інформацію (рис. 5.6).

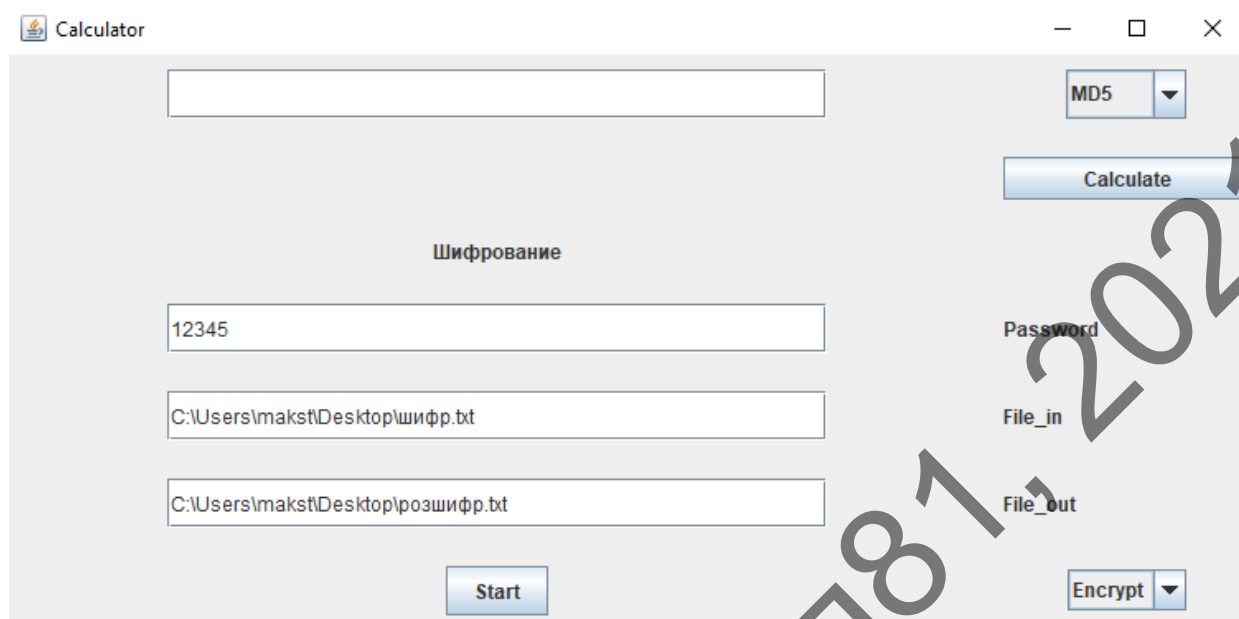


Рисунок 5.6 – Приклад заповнення дешифрування

Для додаткового захисту при дешифруванні, якщо буде не вірний пароль, всерівно програма дасть підтвердження виконання операції (рис. 5.7).

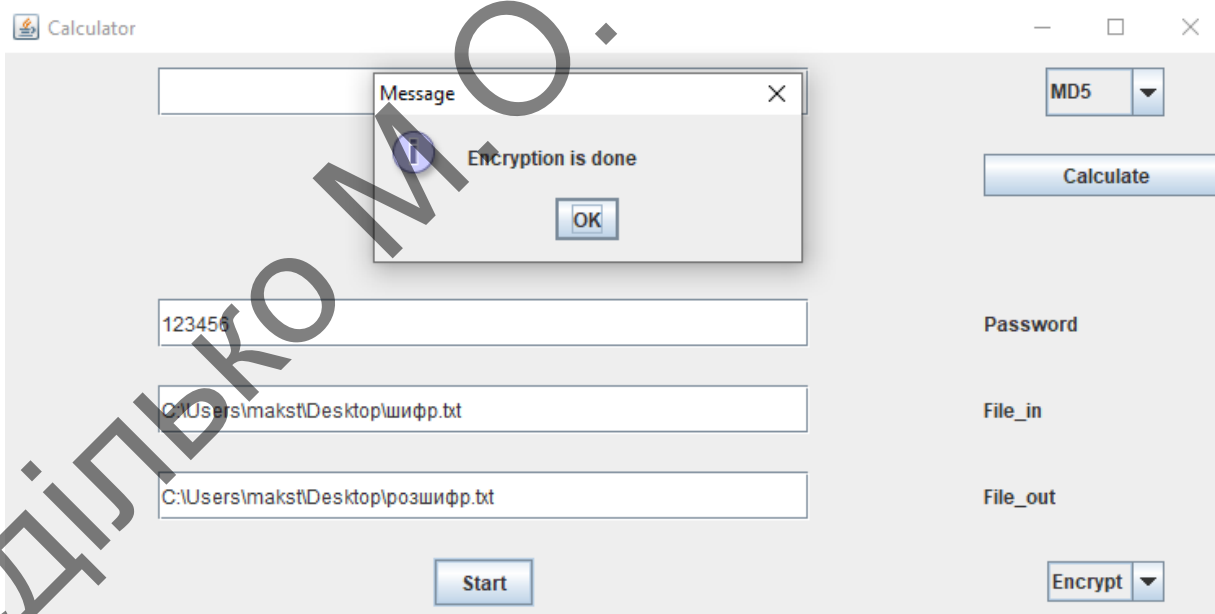


Рисунок 5.7 – Додатковий захист

Але дешифрування не відбудеться. Тільки при введенні того ж самого паролю, що було використано при шифруванні, отримає відпрацювання алгоритму (рис. 5.8).

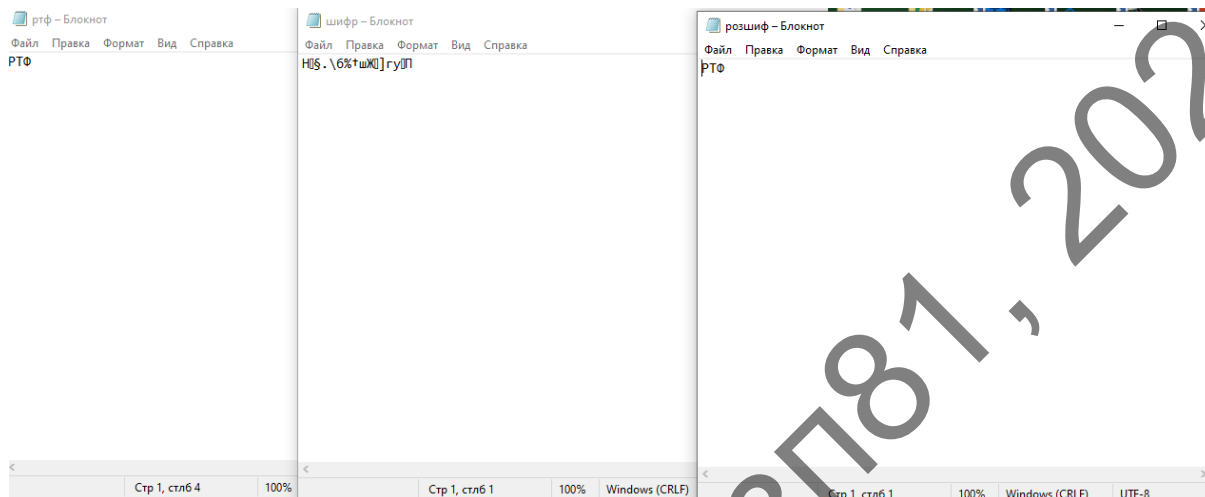


Рисунок 5.8 – Результат дешифрування

При моделюванні мережі наш алгоритм відпрацьовує на відмінно, що гарантує захист персональних даних. Для особливо цінної інформації процес шифрування можна зациклити, і на розшифрування повідомлення, не знаючи скільки разів і яким алгоритмом було виконано дану операцію. Розшифрувати повідомлення майже не можливо.

ВИСНОВКИ

Під час розробки програмної системи досліджено основні принципи та методи шифрування, вирішено проблеми передачі та збереження інформації в незахищених та захищених мережах. В ході аналізу існуючих алгоритмів шифрування досліджено головні переваги та недоліки кожного з них, і обґрунтовано використання асиметричного блочного алгоритму AES.

Досліджено різноманітні техніки та алгоритми захисту даних, було виявлено різноманітні сфери, де можна застосовувати дані методи. Також створено прикладний додаток з зручним графічним інтерфейсом. Який може бути встановлений на будь-якому пристрої. За допомогою якого вирішується проблема захисту інформації.

При створенні продукту було запропоновано простий алгоритм шифрування та зберігання даних в захищеному вигляді. Програма надає можливість неоднократного шифрування повідомлення, що надає можливість більш надійно захистити інформацію, ускладнити життя зломисникам.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. <http://alextexnok.blogspot.com/p/lida-15.html>
2. Грохаем алгоритмы. Иллюстрированное пособие для программистов и любопытствующих. - СПб.: Питер, 2017. - 288 с.: ил. - (Серия «Библиотека программиста»).—124с
3. <https://ru.wikipedia.org/wiki>
4. Шилдт Герберт. Ш57 Java 8: руководство для начинающих, 6-е изд. ООО "И.Д. Вильямс", 2015. - 720 с – 29с
5. <https://www.jetbrains.com/ru-ru/idea/>
6. <https://javarush.ru/groups/posts/2560-vvedenie-v-java-fx>
7. <https://docs.oracle.com/javase/tutorial/uiswing/>
8. Эккель Б. Э38 Философия Java. 4-е полное изд. — СПб.: Питер, 2015. — 1168 с.: ил. — (Серия «Классика computer science»).—45с
9. <https://habr.com/ru/post/449552/>

Додаток А

Неділцько М.О. РІ-3781, 2021

Клас з алгоритмом AES

```
package Crypt;
```

```
import javax.crypto.Cipher;  
import javax.crypto.spec.SecretKeySpec;  
import java.io.UnsupportedEncodingException;  
import java.security.MessageDigest;  
import java.security.NoSuchAlgorithmException;  
import java.util.Arrays;
```

```
/**
```

```
 *Класс написаний для шифрування даних
```

```
 * алгоритмом AES в режиме ECB
```

```
 * @see Hash_data
```

```
 * @see Calculator
```

```
 */
```

```
public class AES {
```

```
    private static SecretKeySpec secretKey;
```

```
    private static byte[] key;
```

```
    /**
```

```
     * Метод для розрахунку ключа шифрування об'єкта
```

```
     * @param myKey – включає в себе пароль користувача з
```

```
     * якого виробляється секретний ключ
```

```
     * @see AES
```

```
     */
```

```
    private void setKey(String myKey) {
```



```

MessageDigest sha = null;
try {
    key = myKey.getBytes("UTF-8");
    sha = MessageDigest.getInstance("SHA-1");
    key = sha.digest(key);
    key = Arrays.copyOf(key, 16);
    secretKey = new SecretKeySpec(key, "AES");
}
catch (NoSuchAlgorithmException e) {
    e.printStackTrace();
}
catch (UnsupportedEncodingException e) {
    e.printStackTrace();
}
}

/**
 * Метод використовується для шифрування даних
 * @param data – змінна яка містить в собі дані для шифрування
 * @param key – пароль користувача для шифрування
 * @return повертає масив байт вже зашифрованих даних
 */
public byte [] Encrypt(byte [] data, String key) {
    try
    {
        setKey(key);

        Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5Padding");
        cipher.init(Cipher.ENCRYPT_MODE, secretKey);
    }
}

```

```

        //return
        Base64.getEncoder().encodeToString(cipher.doFinal(data.getBytes("UTF-8")));
        return cipher.doFinal(data);
    }
    catch (Exception e)
    {
        System.out.println("Error while encrypting: " + e.toString());
    }
    return null;
}

```

```
/**
```

```
* Метод
```

```
* @param data – змінна яка включає в себе дані для розшифрування
```

```
* @param key - пароль користувача для розшифрування
```

```
* @return повертає масив байт уже розшифрованих даних
```

```
*/
```

```

public byte [] Decrypt(byte[] data, String key) {
    try
    {
        setKey(key);
        Cipher cipher = Cipher.getInstance("AES/ECB/PKCS5PADDING");
        cipher.init(Cipher.DECRYPT_MODE, secretKey);
        //return new String(cipher.doFinal(Base64.getDecoder().decode(data)));
        return cipher.doFinal(data);
    }
    catch (Exception e)
    {
        System.out.println("Error while decrypting: " + e.toString());
    }
    return null;
}

```

```
}
```

```
}
```

Клас Hash_data

```
package Crypt;
```

```
/**
```

```
 * Клас для розрахунку Hash з отриманої строки
```

```
 */
```

```
public class Hash_data {
```

```
/**
```

```
 * Метод для отримання хеша з строки даних
```

```
 * @param txt - змінна яка містить в собі хешовану строку
```

```
 * @param hashType - змінна яка визначає тип хеша
```

```
 * @return повертає строку в якій міститься хеш значення
```

```
 */
```

```
public static String Hash_data (String txt, String hashType){
```

```
    try {
```

```
        java.security.MessageDigest
```

```
        md
```

```
        =
```

```
        java.security.MessageDigest.getInstance(hashType);
```

```
        byte[] array = md.digest(txt.getBytes());
```

```
        StringBuffer sb = new StringBuffer();
```

```
        for (int i = 0; i < array.length; ++i) {
```

```
        sb.append(Integer.toHexString((array[i] & 0xFF) |
0x100).substring(1,3));
    }
    return sb.toString();
} catch (java.security.NoSuchAlgorithmException e) {
    //error action
}
return null;

}
```

```
}
```

Інтерфейс Crypt

```
package Crypt;
```

```
public interface Crypto {
```

```
    public void setKey(String myKey);
```

```
    abstract String Encrypt(String data, String key);
```

```
    abstract String Decrypt(String data, String key);
```

```
}
```

Клас Calculator

```
package Crypt;
```

```
import javax.swing.*;
```

```
import java.awt.event.ActionEvent;
```

```
import java.awt.event.ActionListener;
```

```
import java.io.*;
```

```
public class Calculator {
```

```
    static String[] args;
```

```
    private JPanel Calculator;
```

```
    private JComboBox comboBox1;
```

```
    private JTextField TextField;
```

```
    private JButton button1;
```

```
    private JTextField FileOut;
```

```
    private JTextField FileIn;
```

```
    private JTextField PasswordToCrypt;
```

```
    private JComboBox EncrDecrSwitch;
```

```
    private JButton Encrypt_Decrypt;
```

```
    public static void main(String[] args) {
```

```
        //Calculator.args = args;
```

```
        JFrame jf = new JFrame("Hashcalc");
```

```
        jf.setContentPane(new Calculator().Calculator);
```

```
        jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
        jf.setTitle("Calculator");
```

```
        jf.setLocationRelativeTo(null);
```

```
        jf.pack();
```

```
        jf.setVisible(true);
```

```
}
```

```
/**
```

```
* Конструктор класа и роботи самого калькулятора
```

```
*/
```

```
public Calculator() {
```

```
    /**
```

```
    * Дія на натиска кнопки розрахунок хеша
```

```
    * @param actionEvent
```

```
    */
```

```
    button1.addActionListener(new ActionListener() {
```

```
        @Override
```

```
        public void actionPerformed(ActionEvent actionEvent) {
```

```
            String s;
```

```
            int selected;
```

```
            s = TextField.getText();
```

```
            String out = "Another algorithms";
```

```
            selected = comboBox1.getSelectedIndex();
```

```
            //System.out.println("selected: = "+selected);
```

```
            if(selected == 0){
```

```
                out = Hash_data.Hash_data(s,"MD5");
```

```
            }
```

```
            else if(selected == 1){
```

```
                out = Hash_data.Hash_data(s,"SHA-1");
```

```
            }
```

```

else if(selected == 2){
    out = Hash_data.Hash_data(s,"SHA-256");
}
else if(selected == 3){
    out = Hash_data.Hash_data(s,"SHA-512");
}
else if(selected == 4){
    out = Hash_data.Hash_data(s,"SHA-224");
}
else if(selected == 5){
    out = Hash_data.Hash_data(s,"SHA-384");
}else{
    JOptionPane.showMessageDialog(null,"Not supported algorithm");
}

JOptionPane.showMessageDialog(null,out);
//System.out.println(out);

```

```

}
});

```

```

Encrypt_Decrypt.addActionListener(new ActionListener() {

```

```

/**

```

```

 * Дія на натискання кнопки шифрування розшифрування

```

```

 * @param actionEvent

```

```

 */

```

```

@Override

```

```

public void actionPerformed(ActionEvent actionEvent) {

```

```

    String in = FileIn.getText();

```

```
if(in.equals("")){
    JOptionPane.showMessageDialog(null,"File_in field is empty");
}
String out= FileOut.getText();
if(out.equals("")){
    JOptionPane.showMessageDialog(null,"File_out field is empty");
}
AES aes = new AES();
String Pass = PasswordToCrypt.getText();

try(InputStream io = new FileInputStream(in);
    BufferedInputStream bis = new BufferedInputStream(io, 1_000_000);
    OutputStream out_st = new FileOutputStream(out);
    BufferedOutputStream bos = new BufferedOutputStream(out_st,
1_000_000)){

    byte [] array = new byte[bis.available()];
    bis.read(array);
    //System.out.println(new String(array));

    int select = EncrDecrSwitch.getSelectedIndex();
    String out_Message="Some troubles";
    byte[] rez =null;
    if(select == 0){
        rez = aes.Encrypt(array, Pass);
        out_Message = "Encryption is done";
    }else if(select == 1){
        rez = aes.Decrypt(array, Pass);
        out_Message = "Decryption is done";
    }
}
```



```
bos.write(rez);
```

```
bos.flush();
```

```
JOptionPane.showMessageDialog(null, out_Message);
```

```
}catch(FileNotFoundException ext){
```

```
    System.out.println("File in that put:"+in+" not found");
```

```
}
```

```
catch (IOException ex){
```

```
    System.out.println("Error reading/writing File");
```

```
    ex.printStackTrace();
```

```
}
```

```
}
```

```
});
```

```
}
```

```
}
```

Неділько М.О. РІ-3781, 2021