МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
КАФЕДРА КОМП'ЮТЕРИЗОВАНИХ СИСТЕМ ЗАХИСТУ ІНФОРМАЦІЇ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри
Комп'ютеризованих систем захисту інформації

_____ Михайло СТЕПАНОВ
«____»_____ 2023 р.

На правах рукопису
УДК
004.056.55(043.3)

КВАЛІФІКАЦІЙНА РОБОТА

ЗДОБУВАЧА ВИЩОЇ ОСВІТИ
ОСВІТНЬОГО СТУПЕНЯ «МАГІСТР»

Тема: Розробка програмного засобу для захисту даних з різними методами шифрування

Виконавець: Денис ГРИНЕНКО

Керівник: к.т.н., доцент Андрій ПЕТРЕНКО

Консультант розділу «Охорона навколишнього середовища»: к.т.н., доцент Тетяна ДМИТРУХА

Нормоконтролер: к.т.н., доцент Андрій ПЕТРЕНКО

Київ 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет: Кібербезпеки та програмної інженерії

Кафедра: Комп'ютеризованих систем захисту інформації

Освітній ступінь: Магістр

Спеціальність: 125 «Кібербезпека»

Освітньо-професійна програма: «Безпека інформаційних і комунікаційних систем»

ЗАТВЕРДЖУЮ
Завідувач кафедри Комп'ютеризованих систем захисту інформації

_____ Михайло СТЕПАНОВ
«___» _____ 2023 р.

ЗАВДАННЯ

на виконання кваліфікаційної роботи

здобувача вищої освіти Гриненка Денис Сергійович

1. Тема: Розробка програмного засобу для захисту даних з різними методами шифрування
затверджена наказом ректора від «15» вересня 2023 № 1814/ст.
2. Термін виконання з 16 жовтня 2023р. по 31 грудня 2023р.
3. Вихідні дані: проаналізувати існуючі методи шифрування даних, на основі аналізу виділити вхідні та вихідні параметри, завдяки яким можливо провести шифрування даних, виявлення їх переваг і недоліків; розробити програмний засіб шифрування даних та інтегрувати його в телеграм бот.
4. Зміст пояснювальної записки: теоретичні аспекти шифрування даних, аналіз алгоритмів шифрування даних, реалізація програмного коду, інтеграція програмного засобу в телеграм, розробка рекомендацій щодо зберігання ключа та захисту даних.

КАЛЕНДАРНИЙ ПЛАН

виконання кваліфікаційної роботи

№ з/п	Етапи виконання кваліфікаційної роботи	Термін виконання етапів	Примітка
1.	Уточнення постановки задачі	16.10.2023	*Виконано*
2.	Аналіз літературних джерел	18.10.2023	*Виконано*
3.	Обґрунтування вибору рішення	25.10.2023	*Виконано*
4.	Збір інформації	26.10.2023	*Виконано*
5.	Дослідження методів шифрування даних	1.11.2023	*Виконано*
6.	Розробка методики та структури системи захисту даних симетричними та асиметричними методами шифрування	5.11.2023	*Виконано*
7.	Розробка алгоритму та програмного забезпечення захисту даних симетричними та асиметричними методами шифрування	15.11.2023	*Виконано*
8.	Перевірка на антиплагіат	12.11.2023	*Виконано*
9.	Оформлення і друк пояснювальної записки	12.11.2023	*Виконано*
10.	Оформлення презентації	12.11.2023	*Виконано*
11.	Отримання рецензій	22.12.2023	*Виконано*

6. Консультанти з окремих розділів

Розділ	Консультант (посада, П.І.Б.)	Дата, підпис	
		Завдання видав	Завдання прийняв
Охорона навколишнього середовища	Дмитруха Т.І.		

7. Дата видачі завдання: «16» жовтня 2023 р.

Здобувач вищої освіти Денис ГРИНЕНКО

(підпис, дата)

Керівник кваліфікаційної роботи Андрій ПЕТРЕНКО

(підпис, дата)

РЕФЕРАТ

Кваліфікаційна робота складається зі вступу, п’яти розділів, загальних висновків, списку використаних джерел, додатків і має 74 сторінки основного тексту, 24 рисунка, 1 таблицю, 24 сторінки додатків. Список використаних джерел містить 36 найменувань і займає 3 сторінки. Загальний обсяг роботи становить 107 сторінок.

Сучасний світ відзначається зростаючим обсягом і важливістю цифрової інформації, яка потребує надійного захисту від несанкціонованого доступу. Розробка програмного засобу для захисту даних з різними методами шифрування стає надзвичайно актуальною задачею. Цей реферат розглядає процес розробки програмного засобу з використанням різних методів шифрування для забезпечення безпеки цифрової інформації.

Збільшення кількості інформації, яка зберігається і обробляється на комп'ютерах та інших пристроях, робить справу про безпеку даних надзвичайно важливою. Несанкціонований доступ до даних може призвести до витоку конфіденційної інформації, фінансових втрат і порушення приватності користувачів.

Для захисту даних використовуються різні методи шифрування, такі як симетричне шифрування, асиметричне шифрування та геш-функції. Симетричне шифрування використовує один ключ для шифрування і розшифрування даних, тоді як асиметричне шифрування використовує пару ключів: публічний і приватний.

Програмний засіб для захисту даних має наступні основні функціональність:

- Шифрування та розшифрування даних за допомогою симетричних ключів;
- Використання пари ключів для асиметричного шифрування;
- Можливість зміни ключів та паролів користувачів;
- Простота використання;

- Доступність для користувачів.

Розробка програмного засобу для захисту даних з різними методами шифрування є критично важливою для забезпечення безпеки цифрової інформації. Використання різних методів шифрування дозволяє створити надійний інструмент для захисту даних від несанкціонованого доступу. Подальші дослідження можуть спрямовуватися на вдосконалення і розширення функціональності програмного засобу для ще більшої надійності в захисті даних.

Для практичної реалізації програмного засобу було використано сучасні програмні мови та бібліотеки, такі як Python. Програмний засіб буде реалізований через телеграм бот для доступності для всіх користувачів.

Отриманий програмний засіб може бути використаний в різних галузях, де важлива безпека даних. Це може бути застосовано в банківській сфері для захисту фінансових транзакцій, в медицині для зберігання конфіденційної медичної інформації, а також в корпоративних системах для забезпечення конфіденційності корпоративних даних.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

AES – Advanced Encryption Standard.

DES – Data Encryption Standard.

IDEA – International Data Encryption Algorithm.

RSA – Rivest, Shamir і Adleman.

ВСТУП

У сучасному інформаційному суспільстві, де кількість цифрової інформації зростає експоненційно, захист конфіденційності та цілісності даних стає найважливішою проблемою. Несанкціонований доступ до важливої інформації може призвести до серйозних наслідків, включаючи фінансові втрати, порушення приватності і порушення законодавства.

Однією з ключових стратегій для забезпечення безпеки цифрової інформації є використання сучасних методів шифрування. Шифрування використовується для захисту даних від несанкціонованого доступу, перетворюючи їх у незрозумілу форму, яку може розшифрувати тільки авторизований користувач.

У цій кваліфікаційній роботі розглянемо методи шифрування даних, процес розробки програмного засобу, спрямованого на захист даних з використанням різних методів шифрування. Ми розглянемо важливі аспекти розробки, практичну реалізацію, тестування та застосування такого програмного засобу. Також ми дослідимо актуальність та важливість цієї теми в сучасному світі інформаційних технологій та комп'ютерної безпеки.

Актуальність. У сучасному інформаційному суспільстві, де обсяги обробки та передачі конфіденційної інформації зростають експоненційно, захист особистих даних стає критично важливою проблемою. Забезпечення безпеки інформації вимагає впровадження сучасних методів шифрування, спроможних відповідати викликам сучасних технологій та злочинців, які постійно вдосконалюють свої техніки.

Метою роботи є розробка програмного засобу, який забезпечить криптографічний захист конфіденційної інформації за допомогою симетричних та асиметричних методів шифрування.

Об’єктом дослідження є захист даних шляхом шифрування симетричними та асиметричними методами.

Предметом дослідження є методи шифрування такі як AES, DES, IDEA, RSA та ElGamal, які можна використовувати для шифрування.

Методи дослідження включають аналіз і порівняння симетричних та асиметричних методів шифрування, розробку та реалізацію програмного засобу.

Наукова новизна полягає в удосконаленні криптографічного захисту конфіденційних даних за рахунок використання алгоритмів, таких як: AES, DES, IDEA, RSA та ElGamal. Удосконалення реалізовано у єдиному програмному засобі, який успішно інтегровано для використання у популярний месенджер Telegram.

У роботі висвітлено інтеграцію розробленого програмного засобу в популярний месенджер Telegram, що вносить новий аспект у використання шифрування даних у реальних умовах. Також, надані рекомендації щодо зберігання ключа та захисту даних, що є додатковою вагомою складовою наукової новизни дослідження.

Практична цінність отриманих результатів полягає у шифруванні даних в реальному часі через програмний засіб інтегрований в месенджер Telegram, що дає змогу швидко і доступно зашифрувати необхідні дані та відразу отримати результати шифрування разом з ключами шифрування безпосередньо в самому месенджері.

Апробація. Основні положення роботи доповідалися та обговорювалися на таких конференціях:

- Гриненко Д.С. Розробка програмного засобу для захисту даних з різними методами шифрування/ Д.С. ГРИНЕНКО, А.Б. ПЕТРЕНКО// Розвиток сучасної науки: актуальні питання теорії та практики. – 2023 – с 301-303.

Розділ 1. ТЕОРЕТИЧНІ АСПЕКТИ ШИФРУВАННЯ ДАНИХ

1.1. Основні поняття і принципи шифрування

У сучасному інформаційному суспільстві, де обмін конфіденційною інформацією здійснюється за допомогою різноманітних електронних систем, важливим аспектом є забезпечення безпеки даних. Шифрування виступає ключовим елементом в цьому контексті, забезпечуючи конфіденційність та цілісність інформації.

Основні поняття шифрування включають в себе алгоритми, ключі та криптосистеми. Алгоритми шифрування є математичними процедурами, застосовуваними до вхідних даних для перетворення їх у незрозумілу форму, яку можна безпечно передавати. Ключі використовуються для налаштування алгоритмів та визначення унікальності кожного шифрованого повідомлення. Криптосистеми об'єднують алгоритми та ключі, створюючи комплексні системи шифрування.

Принципи шифрування базуються на математичних методах, таких як симетричне та асиметричне шифрування. У симетричному шифруванні використовується один ключ для як шифрування, так і розшифрування даних. З іншого боку, асиметричне шифрування використовує два ключі - публічний та приватний. Публічний ключ використовується для шифрування інформації, тоді як приватний ключ використовується для розшифрування. Ці принципи визначають ефективність та безпеку систем шифрування.

У даному дослідженні ми розглянемо основні поняття та принципи сучасних методів шифрування, вивчаючи їхню структуру, властивості та застосування в різних сферах інформаційної безпеки. Робота спрямована на аналіз і порівняння різних методів шифрування.

Іноді процес шифрування є досить простим та автоматичним. Але іноді у ньому трапляються збої. Чим більше ви знаєте про шифрування, тим у більшій безпеці ви будете у разі подібних ситуацій.

Шифрування - це математичний процес, який використовується для перетворення інформації в нечитаний (зашифрований) вигляд. У цьому відновлення початкового виду інформації (розшифровування) можливе лише з допомогою спеціальних знань. У процесі шифрування використовуються шифр та ключ.

Шифр - це набір правил (алгоритм), що використовується при шифруванні та розшифровуванні. Це чітко визначені, виражені у вигляді формули правила, яким слідувати.

Ключом є інструкція для шифру: як саме потрібно шифрувати та розшифровувати дані. Ключі є однією з найважливіших концепцій у шифруванні [1].

При використанні симетричного шифрування існує єдиний ключ як шифрування, і розшифровування даних.

Симетричне шифрування досі застосовується сьогодні. Воно часто існує у формі «струмових» та «блочних шифрів», які використовують складні математичні процеси для ускладнення злому. В даний час процес шифрування даних проходить у кілька етапів, максимально ускладнюючи вилучення вихідних даних за відсутності відповідного ключа. Сучасні алгоритми симетричного шифрування, такі як алгоритм Advanced Encryption Standard (AES), є надійними та швидкими. Симетричне шифрування широко використовується комп'ютерами таких завдань, як шифрування файлів, розділів жорстких дисків на комп'ютері, повного шифрування дисків і пристроїв, і навіть шифрування баз даних, наприклад у менеджерах паролів. При розшифровуванні інформації, зашифрованої за допомогою симетричного шифрування, ви запитуватимете пароль.

Наявність лише одного ключа може бути корисною, якщо ви єдина людина, яка потребує доступу до зашифрованої інформації. Але володіння єдиним ключем може стати проблемою при бажанні поділитися доступом до зашифрованої інформації з другом, що знаходиться далеко від вас.

Асиметричне шифрування, також зване шифруванням з відкритим ключем, вирішує ці проблеми. Асиметричне шифрування використовує два ключі: закритий ключ (для розшифровування) та відкритий ключ (для шифрування) [2].

Криптографічні методи захисту інформації - це спеціальні методи шифрування, кодування або іншого перетворення інформації, в результаті якого її зміст стає недоступним без пред'явлення ключа криптограми і зворотного перетворення. Криптографічний метод захисту, безумовно, самий надійний метод захисту, так як охороняється безпосередньо сама інформація, а не доступ до неї. Ці методи вже являються високим рівнем захисту інформації.

Світ все більше будується навколо віртуалізації та хмари. Хмара пропонує значні переваги з погляду вартості та гнучкості. Тим не менш, деякі IT-менеджери все ще не наважуються зберігати конфіденційні дані в хмарі, воліючи підтримувати власний центр обробки даних, який вони контролюють. Шифрування даних дозволяє використовувати модель Cloud and Infrastructure as a Service, зберігаючи при цьому конфіденційність даних. Нижче наведено основні переваги шифрування даних у хмарі:

- Допомагає організаціям перейти у хмару;
- Організація володіє ключами і може легко їх відкликати та виводити з експлуатації;
- Допомагає забезпечити безпечну роботу безлічі користувачів у хмарі;
- Відділення даних від ключових сервісів може завадити постачальникам послуг отримати доступ до даних або випадково розкрити їх;
- Допомагає дотримуватись нормативно-правових вимог;
- Забезпечує організації зону безпеки завдяки повідомленням про порушення;
- Може забезпечити постачальникам послуг конкурентну перевагу;
- Вселяє впевненість у збереженні даних у багатохмарному світі;

- Дозволяє організаціям убезпечити свій віддалений офіс.

Великі дані — це термін для збирання та аналізу величезних обсягів інформації з різних джерел виявлення тенденцій і прив'язок, які можуть застосовуватися для бізнес-прогнозування. Оскільки великі дані надходять з різних джерел, вони створюють безліч загроз безпеки даних в порівнянні з меншими наборами даних. Надійне шифрування великих даних вимагає межі, сертифікованої за стандартом FIPS 140-2 рівня 3, для захисту ключів шифрування великих даних та розвантаження криптографічної обробки: саме це дозволяє забезпечити шифрування з низькою затримкою та апаратним прискоренням [3].

Процес шифрування включає 2 складові - безпосередньо саме шифрування даних та їх розшифрування. За допомогою шифрування можна забезпечити 3 стани інформаційної безпеки:

Повну конфіденційність. Шифрування дозволяє приховати дані від сторонніх осіб у процесі надсилання/отримання або зберігання інформації .

Цілісність. Шифрування може гарантувати, що в процесі розшифровки інформація не зміниться та не буде пошкоджена.

Можливість ідентифікації. Шифрування — це метод аутентифікації джерела даних, що надає доступ до даних лише тим особам, яким ця інформація призначається.

Щоб отримати доступ до зашифрованої інформації, у користувача обов'язково має бути спеціальний ключ і дешифратор - обладнання, яке дозволяє виконати процес розшифрування даних.

Основна суть шифрування даних полягає в тому, що, якщо зловмисник все ж таки зможе перехопити зашифровану інформацію, через відсутність ключа для її розшифрування у нього не вийде ні прочитати, ні внести зміни до цієї інформації [8].

У світі часто використовуються шифрувальні криптографічні системи з відкритими ключами для шифрування і розшифровування даних. Тобто для отримання доступу до інформації можна використовувати різні ключі.

Але зважаючи на розвиток криптоаналізу, сьогодні можна використовувати такі способи захисту даних, при яких можна дешифрувати файли без ключа. Такі методи базуються на математичному аналізі передачі з різних каналів зв’язку.

1.2. Типи шифрування: симетричне і асиметричне

Шифрування забезпечує 3 компоненти захисту інформації (рис.1.1.):

Рис. 1.1. Компоненти захисту інформації

Конфіденційність. Шифрування приховує інформацію від неавторизованих користувачів під час передачі чи зберігання.

Цілісність. Шифрування використовується для запобігання зміні інформації під час передачі або зберігання.

Доступність. Шифрування допомагає аутентифікувати джерело інформації та не дозволяє відправнику інформації заперечувати, що він насправді був відправником даних[3].

Шифрування використовує математичні алгоритми та ключі. Алгоритм - це набір математичних операцій, необхідних для виконання певного процесу шифрування, а ключі - це рядки тексту та цифр, які використовуються для шифрування та розшифрування даних.

Існує два основних типи шифрування – симетричне та асиметричне, – які різняться за типом ключів, що використовуються для шифрування та дешифрування.

Симетричне шифрування - найстаріший метод шифрування, відомий людству. Протягом майже всієї історії криптографії, яка налічує близько 4000 років, це був єдиний метод шифрування інформації.

Симетричне шифрування, також зване шифруванням із закритим ключем, - це коли дані шифруються і розшифровуються відправником і одержувачем з використанням одного і того ж секретного ключа. Це означає, що ключ повинен передаватися безпечно, щоб тільки одержувач міг отримати доступ до нього [4].

Ось як працює процес захисту інформації за допомогою симетричного шифрування, приклад зображений на рис. 1.2.

1. Відправник (або одержувач) вибирає алгоритм шифрування, генерує ключ, інформує одержувача (або відправника, залежно від випадку) про обраний алгоритм і відправляє ключ захищеним каналом зв'язку.

2. Відправник шифрує повідомлення за допомогою ключа та надсилає зашифроване повідомлення одержувачу.

3. Отримувач отримує зашифроване повідомлення та розшифровує його за допомогою того ж ключа.

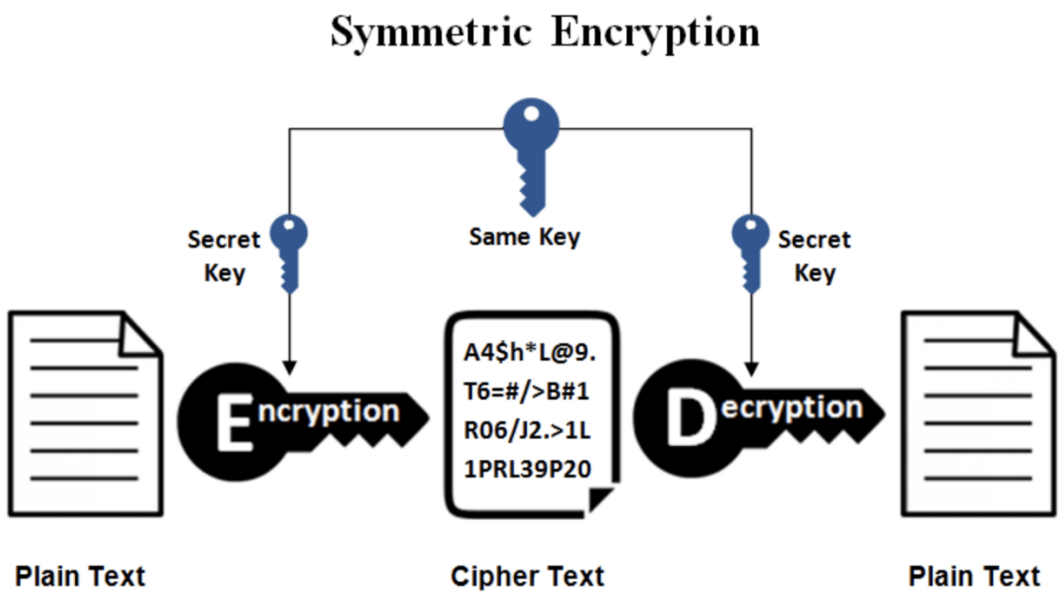

Рис. 1.2. Симетричне шифрування

Існує два основних типи симетричних шифрів: блокові та потокові.

При блоковому шифруванні інформація розбивається на фіксовані блоки довжини (наприклад, 64 або 128 біт). Потім ці блоки шифруються один за одним. Ключ застосовується до кожного блоку у заданому порядку. Зазвичай це передбачає кілька циклів змішування та заміщення. Блоковий шифр є важливим компонентом багатьох криптографічних протоколів і широко використовується для захисту даних, що передаються мережею.

Кожен вихідний символ перетворюється на зашифрований потоковий шифр залежно від використовуваного ключа та його розташування у вихідному тексті. Поточні шифри мають більш високу швидкість шифрування, ніж блокові, але вони також мають більшу вразливість.

Найбільш помітною перевагою симетричного шифрування є його простота, оскільки для шифрування та дешифрування використовується один ключ. Таким чином, симетричні алгоритми шифрування значно швидше за асиметричні і вимагають менше обчислювальної потужності[4].

Водночас той факт, що для шифрування та дешифрування використовується один і той самий ключ, є основною вразливістю систем симетричного шифрування. Необхідність передачі ключа іншій стороні є вразливістю системи безпеки, оскільки, якщо він потрапить до чужих рук, інформація буде розшифрована. Відповідно особливу увагу слід приділити можливим способам перехоплення ключа та підвищення безпеки передачі.

Асиметричне шифрування - відносно нова криптографічна система, що з'явилася у 1970-х роках. Його основна мета - усунути вразливість симетричного шифрування, тобто використання одного ключа.

Асиметричне шифрування, також зване шифруванням з відкритим ключем, є криптографічною системою, що використовує два ключі. Відкритий ключ може передаватися незахищеним каналом і використовується для шифрування повідомлення. Для розшифровки повідомлення використовується закритий ключ, відомий лише отримувачу [7].

Пара ключів математично пов'язана один з одним, тому можна обчислити відкритий ключ, знаючи закритий, але не навпаки.

Ось як працює асиметричне шифрування, яке зображено на рис.1.3.:

Одержувач вибирає алгоритм шифрування та генерує пару відкритого та закритого ключів.

Отримувач передає відкритий ключ відправнику.

Відправник шифрує повідомлення за допомогою відкритого ключа та надсилає зашифроване повідомлення одержувачу.

Отримувач отримує зашифроване повідомлення та розшифровує його, використовуючи свій закритий ключ [9].

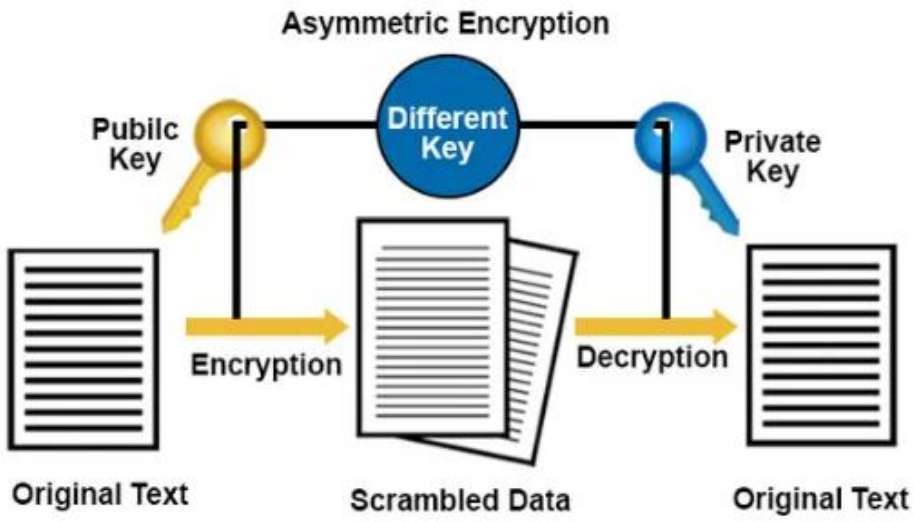

Рис. 1.3. Асиметричне шифрування

Найбільш очевидною перевагою цього шифрування є його безпека, оскільки закритий ключ не потрібно нікому передавати. Звичайно, це значно спрощує керування ключами у великих мережах.

Однак цей метод шифрування має недоліки. Одним із прикладів є більш висока складність, нижча швидкість і вищий попит на обчислювальні ресурси. Крім того, незважаючи на високу безпеку асиметричного шифрування, воно, як і раніше, вразливе для атаки "людина посередині" (MITM), при якій

зловмисник перехоплює відкритий ключ, відправлений одержувачем відправнику. Потім зловмисник створює власну пару ключів і маскується під одержувача, відправляючи відправнику помилковий відкритий ключ, який відправник вважає відкритим ключем, відправленим одержувачем [5]. Зловмисник перехоплює зашифровані повідомлення від відправника до одержувача, розшифровує їх за допомогою свого закритого ключа, повторно шифрує за допомогою відкритого ключа одержувача і відправляє повідомлення одержувачу. У цьому випадку ніхто з учасників не усвідомлює, що третя сторона перехоплює повідомлення або підміняє його хибним. Це наголошує на необхідності аутентифікації з відкритим ключем [10].

Таблиця 1.1.

Порівняння симетричного та асиметричного шифрування

Симетричне шифрування	Асиметричне шифрування
Швидке	Повільне
Не потребує великих обчислювальних потужностей	Вимагає великих обчислювальних потужностей
Використовується для шифрування як великих, так і коротких повідомлень	Використовується для шифрування невеликих повідомлень
Необхідний обмін ключами як для шифрування, так і для розшифровування	Ключем для розшифровування ділитися не можна, необхідно передати лише відкритий ключ, який використовується для шифрування
Не може використовуватись для перевірки особи (автентифікації)	Може використовуватись для перевірки особи (автентифікації)

Симетричне та асиметричне шифрування часто використовуються спільно для шифрування під час передачі даних [6].

1.3. Висновки до розділу

У даному розділі були розглянуті основні теоретичні аспекти шифрування даних. Актуальність проблеми забезпечення конфіденційності інформації в сучасному цифровому світі підкреслено швидким розвитком технологій та зростанням загроз кібербезпеки.

Визначено, що існує різноманітність методів шифрування, серед яких симетричні та асиметричні алгоритми, шифрування на рівні файлів чи баз даних, а також використання хеш-функцій. Розглянуті основні принципи кожного з цих методів, їх переваги та недоліки.

Важливим елементом висвітлення стали підходи до вибору методів шифрування залежно від конкретних вимог та контексту використання. Аналіз різних сценаріїв застосування різних методів дозволяє краще зрозуміти їх придатність у різних умовах.

Висновки цього розділу надають теоретичну основу для подальших розділів роботи, спрямованих на розробку програмного засобу для захисту даних з використанням визначених методів шифрування. Також вони вказують на важливість вибору оптимального підходу до забезпечення безпеки інформації в залежності від конкретних умов та завдань користувача.

Розділ 2. ФУНКЦІОНАЛЬНІСТЬ ПРОГРАМНОГО ЗАСОБУ

2.1. Алгоритми шифрування даних

2.1.1 Алгоритм шифрування AES

Алгоритм шифрування AES (Advanced Encryption Standard) є одним із найпопулярніших і найбільш надійних симетричних алгоритмів шифрування в сучасному світі. Він використовується для захисту конфіденційності і цілісності даних у багатьох застосунках, включаючи безпеку інформації в мережах, зберігання даних на пристроях і багато інших. У цьому тексті ми розглянемо основні принципи роботи AES і його основні етапи.

AES був прийнятий Національним інститутом стандартів і технологій (NIST) США в 2001 році як заміна DES (Data Encryption Standard). Він використовується для шифрування блоків даних розміром 128 біт і підтримує ключі різної довжини: 128, 192 і 256 біт.

AES включає в себе декілька основних етапів обробки блоку даних:

1. Додавання ключа (Key Addition): Початковий ключ додається до блоку даних для початкової перестановки.

2. SubBytes: Кожен байт блоку даних замінюється на відповідний байт з S-Box, який є заздалегідь визначеною таблицею замін.

3. ShiftRows: Байти в кожному рядку блоку даних зсуваються вліво на різну кількість позицій. Це допомагає відсунути дані для покращення дифузії.

4. MixColumns: Байти в кожному стовпці блоку даних перемішуються з використанням математичних операцій, що підвищують дифузію даних [11].

5. Додавання ключа раунду (Round Key Addition): Ключ раунду додається до блоку даних. Ключ раунду генерується з основного ключа.

Ці етапи повторюються кілька разів (10 раундів для 128-бітного ключа, 12 раундів для 192-бітного ключа і 14 раундів для 256-бітного ключа) для забезпечення надійного шифрування.

Функції і особливості AES:

- Симетричний алгоритм: AES використовує один і той же ключ для шифрування і розшифрування даних.

- Блочний шифр: AES шифрує дані по блоках фіксованого розміру (128 біт), що робить його відмінним від потокових шифрів.

- Висока безпека: AES вважається надійним і важким для атаки, якщо використовується достатньо довгий ключ.

- Широке застосування: AES використовується в різних сферах, включаючи захист інформації в мережах, шифрування файлів, захист даних на пристроях і багато інших [12].

AES є сучасним і дуже ефективним алгоритмом шифрування, який забезпечує безпеку для захисту конфіденційної інформації. Його робота базується на складних математичних операціях і послідовному виконанні раундів, що дозволяє забезпечити надійний захист даних. Успіх AES полягає в комбінації високої безпеки і ефективності обчислень, що робить його ідеальним вибором для багатьох застосувань у сфері інформаційної безпеки.

Основним елементом, яким оперує алгоритм AES, є байт – послідовність 8 біт, оброблюваних як єдине ціле. Для формування байтів 128 біт блоку відкритого тексту, вихідного блоку шифротексту та ключа шифру діляться на групи з 8- ми поруч біт, що стоять так, щоб в цілому вийшов масив байт. Нижче на рис.2.1 представлена прийнята нумерація біт в межах кожного байта:

№ біта на вході	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	…
№ байта	0								1								2								…
№ біта на байті	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	…

Рис. 2.1. Прийнята нумерація біт в межах кожного байта

Задавати значення байта зручно у шістнадцятковій системі обчислення. Для цього байт ділиться на дві групи із 4-х біт: група старших біт у байті є

першим шістнадцятковим символом, а група молодших біт – другим. Наприклад, для байта 10101100 отримаємо

10101100 = 1010 1100=AC

Позначимо:

- in_0, in_1, ..., in_{15} - 16 байт блоку відкритого тексту;

- k_0, k_1, ..., k_{15} - 16 байт ключа шифру;

- 16 байт блоку шифротексту.

Вхідними даними для операцій шифрування є масив із 16 байт in_0, in_1, ..., in_{15}. Перед початком шифрування байти цього масиву озміщуються послідовно в стовпці матриці InputBlock (згори вниз). Усередині алгоритму операції виконуються над матрицею байт, званою матрицею станів State або просто станом. Кінцеве значення матриці стану OutputBlock є виходом алгоритму і перетворюється на послідовність байтів шифротексту out_0, out_1, ..., out_{15}. Аналогічно в стовпці матриці InputKey потрапляють і 16 байтів k_0, k_1, ..., k_{15}ключа шифру. Розмірність усіх матриць – 4 х 4.

Розглянемо докладніше перетворення шифрування раунду.

1. Операція SubBytes.

Операція виконує нелінійну заміну байтів, що виконується незалежно з кожним байтом матриці State. Заміна оборотна та обудована шляхом комбінації двох перетворень над вхідним байтом:

- знаходження зворотного (інвертованого) елемента щодо множення в полі 8 GF(2) (вважається, що нульовий байт {00} переходить сам у себе);

- виконання деякого афінного перетворення: множення інвертованого байта на багаточлен a(x) = $x^4 + x^3 + x^2 + x + 1$ та додавання з многочленом b(x) = $x^6 + x^5 + x + 1$ в поле $F_2[x]/x^8 + 1$.

Зауважимо, що $a^{-1}(x) = x^6 + x^3 + x$ та $a^{-1}(x)b(x) = x^2 + 1$

У матричній формі процедура SubBytes записується як

$$\begin{pmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix}^{-1} + \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix},$$

Рис. 2.2. матрична форма

де через *x* позначені вхідні біти, а через *y* - вихідні. Якщо на вхід функції потрапляє нульовий байт, то результатом заміни буде число $y = b$. Процес заміни байтів за допомогою таблиці підстановки ілюструє малюнок. Нелінійність перетворення обумовлена нелінійністю інверсії x^{-1}, а оборотність – оборотністю матриці. Створену на основі цієї операції спеціальну таблицю замін
байтів у шістнадцятковій системі називають S-боксом, зображена на рис.2.3.

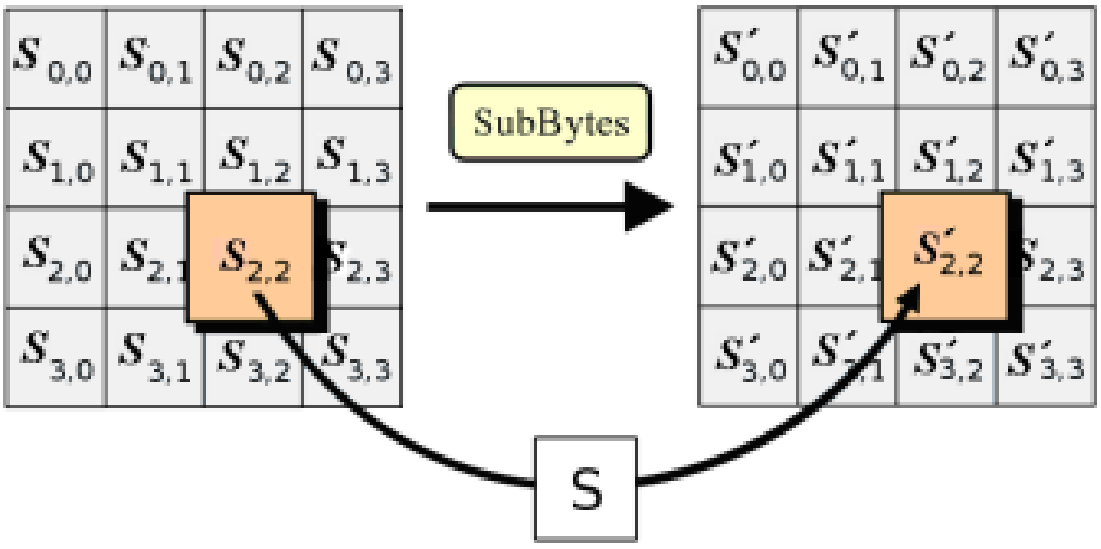

Рис. 2.3 S-бокс

2. Операція AddRoundKey.

Функція AddRoundKey (State, RoundKey)(зображена на рис.2.4.) побітово складає елементи змінної RoundKey та елементи змінної State по принципу: *i*-й стовпець даних (*i* = 0,1,2,3) складається з певним 4-байтовим фрагментом розширеного ключа *W*[4r + 1], де r – номер потокового раунду алгоритму. При шифруванні перша додавання ключа раунду відбувається до першого виконання операції SubBytes.

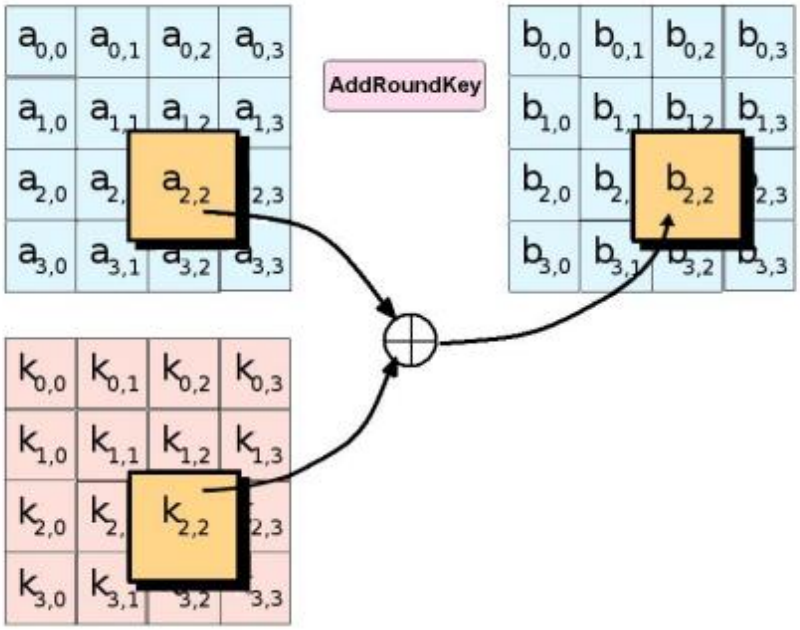

Рис.2.4. Функція AddRoundKey

Наступний рисунок демонструє властивості розсіювання та перемішування інформації в ході шифрування алгоритмом AES. Видно, що два раунди забезпечують повне розсіювання та перемішування інформації. Досягається це за рахунок використання функцій ShiftRows та MixColumns. Операція SubBytes додає шифрування стійкість проти диференціального криптоаналізу, а операція AddRoundKey забезпечує необхідну секретну випадковість[12].

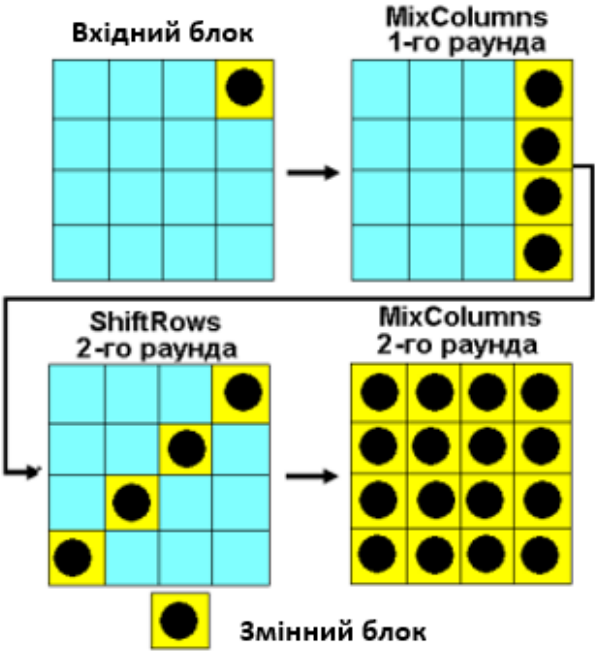

Рис. 2.5. Властивості розсіювання та перемішування інформації

2.1.2 Алгоритм шифрування DES

Алгоритм шифрування DES (Data Encryption Standard) був розроблений у 1970-х роках і був першим стандартом для симетричного шифрування. DES був широко використовуваним і здатним алгоритмом для захисту конфіденційності даних протягом багатьох років, але в результаті зростання обчислювальної потужності комп'ютерів виявився вразливим до атак brute-force.

Основні принципи роботи DES

1. Розширення ключа (Key Expansion): Початковий 56-бітний ключ DES розширюється до 64 біт за допомогою бітової перестановки.

2. Початкова перестановка (Initial Permutation): Повідомлення розбивається на блоки даних розміром 64 біти і піддається початковій перестановці.

3. Раунди шифрування (Encryption Rounds): DES включає в себе 16 раундів шифрування, в кожному з яких виконуються операції заміни

(Substitution) і перестановки (Permutation) над блоками даних і підключами ключа.

4. Поширена перестановка (Final Permutation): Після останнього раунду шифрування виконується остання перестановка, і отримується зашифроване повідомлення.

5. Розширення ключа навпаки (Key Compression): Останній підключ ключа розширюється навпаки до початкового ключа перед розшифруванням [15].

Проблеми і обмеження DES

1. Довжина ключа: DES використовує ключ довжиною лише 56 біт, що робить його вразливим до атак brute-force. В наш час така довжина ключа не вважається надійною.

2. Атака з вибором середини (Meet-in-the-Middle Attack): Атака, при якій атакуючий може знайти ключ, витрачаючи лише 2^56 операцій, що важко визначити в реальних умовах.

3. Нестійкість до атак зі зміною тексту (Differential Cryptanalysis): DES виявився вразливим до атак, які використовують статистичну інформацію про різницю між вхідним і вихідним текстами.

У зв'язку з виявленими вразливостями і обмеженнями DES, NIST вирішив відкликати його як стандарт і запропонував більш безпечний стандарт AES (Advanced Encryption Standard), який був описаний в попередньому текст [14].

У сучасних системах шифрування DES вже не рекомендується використовувати через його низьку безпеку. Відомий криптографічний стандарт AES став більш сучасним і надійним алгоритмом для захисту конфіденційності даних [16].

2.1.3 Алгоритм шифрування IDEA

Алгоритм шифрування IDEA (International Data Encryption Algorithm) - це симетричний блочний шифр, розроблений в 1990 році Xuejia Lai і James Massey. IDEA був призначений для забезпечення високого рівня безпеки шифрування та підтримує ключі різної довжини. У цьому тексті ми докладно розглянемо основні принципи роботи IDEA та його структуру.

Основні характеристики IDEA:

1. Довжина блоку і ключа: IDEA працює з блоками даних розміром 64 біти і підтримує ключі трьох різних довжин: 128, 192 і 256 біт.

2. Симетричний шифр: IDEA є симетричним шифром, тобто використовує один і той же ключ для шифрування та розшифрування даних.

3. Блочний шифр: Інформація шифрується по блоках фіксованого розміру (64 біти), що робить IDEA блочним шифром.

4. Складність і безпека: IDEA відомий своєю складністю і надійністю. Він відповідає сучасним стандартам безпеки та розглядається як безпечний алгоритм шифрування.

Алгоритм IDEA складається з наступних основних етапів:

1. Підготовка ключа: Ключ розширюється та підготовлюється для використання в раундах шифрування.

2. Раунди шифрування: IDEA використовує 8 раундів шифрування для обробки блоку даних. Кожен раунд включає в себе операції побітового додавання, множення і обміну байтами.

3. Фінальний раунд: Після виконання восьми раундів, виконується останній фінальний раунд, що включає в себе додавання ключа і обмін байтами.

4. Завершення інверсією ключа: Зашифрований блок даних піддається інверсії ключа для отримання оригінального тексту [17].

Операції IDEA:

- Побітове додавання (XOR): IDEA використовує операцію побітового додавання для об'єднання даних.

- Множення: Множення виконується в галузі арифметики за модулем 2^16 + 1, і це основна операція IDEA.

- Обмін байтами: В IDEA виконується обмін байтами для покращення дифузії даних.

Хоча IDEA був розроблений досить давно, він все ще використовується в деяких сферах, де важлива висока безпека і низька обчислювальна складність. Проте в більшості випадків IDEA був замінений більш сучасними алгоритмами шифрування, такими як AES, які вважаються надійнішими і забезпечують вищий рівень безпеки.

IDEA - це симетричний блочний шифр, який відомий своєю складністю і надійністю. Хоча він був розроблений давно і втратив свою популярність в деяких сучасних системах шифрування, він все ще може бути корисним в певних застосунках, де потрібна висока безпека даних.

Хоча IDEA був розроблений понад 30 років тому, він все ще зберігає своє значення в багатьох галузях:

Електронна комунікація: Використовується для захисту конфіденційності та цілісності даних при передаванні через мережу.

Фінансові системи: Застосовується для забезпечення безпеки фінансових транзакцій та захисту важливої інформації.

Зберігання даних: Використовується для шифрування даних, які зберігаються на пристроях або серверах. При цьому важливо зазначити, що вибір алгоритму шифрування повинен бути обґрунтованим і враховувати потреби і вимоги конкретного застосунку [18].

Алгоритм шифрування IDEA є значущим досягненням в області криптографії і залишається надійним інструментом для захисту конфіденційності та цілісності даних. Його висока безпека та ефективність роблять його популярним в різних галузях, де потрібна надійна система шифрування. Ураховуючи потенційні зміни в криптографічних стандартах,

IDEA може продовжувати займати важливе місце в сучасному світі інформаційної безпеки.

2.1.4 Алгоритм шифрування RSA

Алгоритм RSA (Rivest–Shamir–Adleman) є одним із найважливіших та найпоширеніших алгоритмів шифрування у світі криптографії. Цей алгоритм, розроблений у 1977 році Роном Рівестом, Аді Шаміром та Леонардом Адлеманом, визнаний одним з найбезпечніших методів шифрування та використовується для захисту конфіденційності даних та безпеки інформаційного обміну. У цьому тексті ми докладно розглянемо структуру, принципи роботи та застосування алгоритму RSA.

Алгоритм RSA базується на використанні математичних принципів і створений для захисту даних шляхом шифрування і підпису інформації. RSA широко використовується в різних областях, таких як електронна комунікація, фінанси, безпека даних та багато інших [19].

Алгоритм RSA базується на математичних операціях над простими числами. Основні компоненти RSA включають в себе:

1. Генерація ключів (Key Generation): Спершу створюється пара ключів - публічний ключ і приватний ключ. Публічний ключ використовується для шифрування повідомлень, тоді як приватний ключ залишається секретним і використовується для розшифрування.

2. Шифрування (Encryption): Публічний ключ використовується для зашифрування повідомлень перед відправленням. Під час цієї операції використовуються математичні формули, які можуть бути легко виконані з публічним ключем.

3. Розшифрування (Decryption): Приватний ключ використовується для розшифрування зашифрованих повідомлень. Цей процес використовує математичні операції, які можна виконати тільки з приватним ключем.

4. Підпис (Signing): Приватний ключ також використовується для створення цифрового підпису повідомлень, що дозволяє перевірити автентичність джерела даних.

Основними математичними принципами, на яких базується RSA, є:

1. Факторизація (Factorization): На цьому принципі ґрунтується безпека RSA. Він ґрунтується на складності розкладання великих простих чисел на їхні прості множники. Ця операція є обчислювально складною для великих чисел і надійно захищає систему від атак brute-force.

2. Модульна арифметика (Modular Arithmetic): RSA використовує модульну арифметику для шифрування і розшифрування повідомлень. Операції виконуються в межах певного модуля, що дозволяє зберегти конфіденційність даних.

4. Застосування RSA

Алгоритм RSA використовується у багатьох галузях та сферах діяльності:

1. Електронна комунікація: RSA використовується для захисту електронної пошти, забезпечення безпеки онлайн-комунікацій та шифрування даних, які передаються через Інтернет.

2. Безпека даних: RSA застосовується для шифрування та розшифрування даних, що зберігаються на серверах, в хмарних обчисленнях і в інших засобах зберігання інформації.

3. Фінансові системи: RSA використовується для захисту фінансових транзакцій та безпеки банківських операцій.

4. Цифровий підпис: RSA дозволяє створювати цифрові підписи, які підтверджують автентичність джерела даних.

Майбутнє RSA залишається світлім і блискучим завдяки його безпеці та надійності. Однак існує постійний пошук для більш ефективних та швидких алгоритмів, особливо в контексті квантових обчислень, які можуть стати загрозою для RSA.

Алгоритм RSA є ключовим елементом сучасної криптографії та відіграє важливу роль у захисті даних та інформаційної безпеки. Забезпечуючи надійність, конфіденційність та цілісність даних, RSA є важливим інструментом для захисту інформації у цифровому світі [20].

2.1.5 Алгоритм шифрування ElGamal

Алгоритм шифрування ElGamal є одним із важливих асиметричних криптографічних алгоритмів, що використовується для шифрування та підпису повідомлень. Цей алгоритм був розроблений в 1985 році Тахіром Ель Гамалем і відомий своєю безпечністю та високою стійкістю до різних атак. У цьому тексті ми розглянемо структуру, принципи роботи та застосування алгоритму шифрування ElGamal для вашої дипломної роботи [21].

Алгоритм ElGamal є асиметричним алгоритмом шифрування, що базується на математичних операціях над простими числами та алгеброю груп. Він був розроблений для розв'язання проблеми обміну секретною інформацією між двома сторонами, які не мають спільного секретного ключа [22].

2. Структура алгоритму ElGamal

Алгоритм ElGamal включає в себе декілька ключових компонентів та операцій:

1. Генерація ключів (Key Generation): Спочатку генеруються ключі: публічний ключ та приватний ключ. Публічний ключ використовується для шифрування повідомлень, тоді як приватний ключ лишається секретним і використовується для розшифрування.

2. Шифрування (Encryption): Публічний ключ використовується для зашифрування повідомлень перед відправленням. Цей процес включає в себе випадково обрану точку на еліптичній кривій, що дозволяє забезпечити криптографічну безпеку.

3. Розшифрування (Decryption): Приватний ключ використовується для розшифрування зашифрованих повідомлень. Цей процес включає в себе операції над елементами еліптичної кривої та математичні обчислення.

4. Підпис (Signing): ElGamal може бути використаний для створення цифрового підпису повідомлень, що дозволяє перевіряти автентичність джерела даних.

Основні математичні принципи, на яких базується алгоритм ElGamal, включають:

1. Еліптичні криві (Elliptic Curves): ElGamal використовує математичні операції над еліптичними кривими для створення криптографічних ключів та шифрування повідомлень.

2. Дискретний логарифм (Discrete Logarithm Problem): Безпека ElGamal базується на складності обчислення дискретного логарифма в еліптичних кривих.

Алгоритм ElGamal знаходить застосування у різних галузях та сферах діяльності:

1. Електронна комунікація: ElGamal використовується для шифрування та захисту електронної пошти, онлайн-передачі повідомлень і інших видів електронної комунікації.

2. Захист даних: Великі обсяги даних, зокрема у хмарних обчисленнях, можуть бути захищені за допомогою ElGamal.

3. Електронний голосування: ElGamal використовується для забезпечення безпеки та аутентифікації електронних голосів та голосів на віддалених виборах.

4. Фінансові системи: Алгоритм ElGamal застосовується для захисту фінансових транзакцій і зберігання фінансової інформації.

Алгоритм ElGamal залишається актуальним і важливим в сфері криптографії, особливо в контексті росту інтересу до безпеки та конфіденційності даних. Проте слід враховувати, що швидкісні та

обчислювальні обмеження можуть вимагати оптимізації та розвитку альтернативних методів шифрування в майбутньому.

Алгоритм шифрування ElGamal є важливим інструментом у сучасній криптографії, який забезпечує безпеку та конфіденційність даних. Його математичні принципи, засновані на еліптичних кривих, роблять його відмінним вибором для захисту інформації. Докладне розуміння структури і принципів роботи алгоритму ElGamal є важливим для розуміння його застосувань та майбутнього розвитку в сфері криптографії [23].

2.2. Вибір мови програмування, бібліотек та середовища реалізації програми.

2.2.1 Мова програмування

Для розробки телеграм-бота було обрано мову програмування Python. Python є популярною та потужною мовою програмування, яка має широкий спектр бібліотек для роботи з телеграм-ботами та криптографією. Основні переваги обраної мови програмування включають:

Простота вивчення та читабельний синтаксис: Python відомий своєю легкістю вивчення, що робить його ідеальним вибором для стартових розробників. Читабельний синтаксис сприяє швидкій розробці та підтримці коду[24.

Велика кількість бібліотек: Python має велику кількість бібліотек та модулів, що спрощують роботу з різними аспектами розробки. У випадку телеграм-ботів, бібліотека telebot дозволяє легко взаємодіяти з API Telegram.

Активна спільнота розробників: Python має велику та активну спільноту розробників, яка постійно розробляє нові інструменти та бібліотеки. Це дозволяє отримувати підтримку та рішення для проблем швидше.

Платформонезалежність: Python підтримується на багатьох операційних системах, що дозволяє запускати телеграм-бота на різних платформах без значних змін.

Розширюваність: Python дозволяє легко інтегрувати сторонні бібліотеки та розширювати функціональність програми.

З урахуванням цих переваг, вибір мови програмування Python став оптимальним для створення телеграм-бота з використанням криптографічних функцій [25].

2.2.2 Бібліотеки

В пункті 2.2.2 "Бібліотеки" будемо розглядати детальніше вибір та обґрунтування обраного набору бібліотек для реалізації функціональності телеграм-бота та криптографічних операцій.

1. `telebot`:

- Опис: `telebot` - це Python-бібліотека, яка дозволяє легко взаємодіяти з API Telegram Bot. Вона надає інтерфейс для створення та налаштування ботів для спілкування з користувачами Telegram.

- Обгрунтування вибору: Використання цієї бібліотеки спрощує розробку та обслуговування телеграм-бота, дозволяючи взаємодіяти з Telegram API і обробляти повідомлення користувачів [26,27].

2. `pyDes`:

- Опис: `pyDes` - це бібліотека для реалізації алгоритму DES (Data Encryption Standard) для симетричного шифрування.

- Обґрунтування вибору: Обрано для реалізації симетричного шифрування, так як DES є одним із стандартів шифрування та може використовуватися для захисту конфіденційності даних [28,29].

3. `Cryptodome`:

- Опис: `Cryptodome` - це бібліотека для роботи з різними криптографічними алгоритмами, такими як AES, RSA, ElGamal та інші. Вона є покращеною версією `PyCryptodome`.

- Обґрунтування вибору: `Cryptodome` надає широкий спектр криптографічних алгоритмів, що дозволяє використовувати сучасні та безпечні методи шифрування в проекті [30,31].

4. `cryptography`:

- Опис: `cryptography` - це бібліотека для роботи з криптографічними алгоритмами вищого рівня, такими як PKCS#1 OAEP, PKCS#1 v1.5, SHA-256 і інші.

- Обґрунтування вибору: `cryptography` надає високорівневий інтерфейс для криптографічних операцій, що спрощує використання складних алгоритмі [32].

Обрані бібліотеки були вибрані через їхню потужність, активну спільноту розробників, наявність документації та прикладів використання. Цей набір бібліотек дозволяє легко інтегрувати криптографічну функціональність та взаємодіяти з Telegram API для створення функціонального та безпечного телеграм-бота [33,34].

2.2.3 Середовище реалізації

У пункті 2.2.3 "Середовище реалізації" розглянуто детальніше вибір та обґрунтування обраного інтегрованого середовища розробки (IDE) для створення програми, а також вказано короткий опис використаного коду, написаного в цьому середовищі.

1. Visual Studio Code (VS Code):

- Опис: Visual Studio Code (VS Code) - це популярне, безкоштовне інтегроване середовище розробки, розроблене компанією Microsoft. Воно підтримує багато мов програмування, включаючи Python, і надає

розширювану функціональність для розробки, налагодження та розгортання програм.

Обґрунтування вибору: Використання VS Code для розробки програми було обрано через наступні переваги:

- Широкий вибір розширень: VS Code має велику кількість розширень, що полегшують роботу з різними мовами програмування і фреймворками.

- Вбудована підтримка Python: VS Code має вбудовану підтримку Python, включаючи автодоповнення, перевірку синтаксису і можливість встановлення віртуальних середовищ.

- Потужність та продуктивність: VS Code дозволяє зосередитися на розробці завдяки швидкому доступу до інструментів та налагодженому робочому середовищу.

- Можливість спільної роботи: VS Code підтримує інтеграцію з різними системами контролю версій, що дозволяє команді спільно працювати над проектом.

2. Короткий опис використаного коду:

- У наданий код включено імпорт необхідних бібліотек для реалізації функціональності телеграм-бота та криптографічних операцій.

- Далі в коді визначено основні обробники повідомлень, включаючи команду `/start` та вибір методу шифрування (симетричного або асиметричного).

- Код також містить налаштування та запуск бота з використанням токену, що дозволяє боту взаємодіяти з користувачами через Telegram API.

З використанням інтегрованого середовища розробки VS Code та створеного коду було досягнуто легкої розробки, налагодження та виконання програми. VS Code надав зручне робоче середовище для автора, що дозволило йому легко розширювати та модифікувати функціональність бота під вимоги завдання [35].

2.3. Висновки до розділу

У цьому розділі детально розглянуті аспекти функціональності розробленого програмного засобу для захисту даних за допомогою різних алгоритмів шифрування.

Вивчені та обґрунтовані різні алгоритми шифрування, що лягли в основу програми. Зокрема, розглянуті алгоритми, такі як AES, DES, IDEA, RSA та ElGamal. Кожен з них має свої особливості, переваги та обмеження, і вибір конкретного алгоритму може залежати від потреб користувача та конкретного сценарію використання.

Проаналізовані та обґрунтовані вибір мови програмування Python, використання конкретних бібліотек та середовища реалізації програми Visual Studio Code. Це визначено для забезпечення оптимальної продуктивності та підтримки програмного засобу в майбутньому.

В цілому, цей розділ надає повний огляд функціональних можливостей програми, визначаючи вибрані алгоритми шифрування та технічні характеристики їхньої реалізації. Важливою частиною є ретельний аналіз вибору інструментів програмування та реалізації, що визначить успішну розробку та впровадження програмного засобу для захисту даних.

Розділ 3. РЕАЛІЗАЦІЯ ПРОГРАМНОГО ЗАСОБУ

3.1 Реалізація алгоритмів шифрування

3.1.1 Реалізація алгоритму шифрування AES

Для реалізації алгоритму шифрування AES використана бібліотека `Cryptodome` у мові програмування Python. Вибір Python обумовлений його зручністю та широкою підтримкою криптографічних бібліотек [13].

Бібліотеку `Cryptodome` було успішно інтегровано у твій проект. Імпорт необхідних модулів виглядає наступним чином:

```
from Cryptodome.Cipher import AES
from Cryptodome.Random import get_random_bytes.
```

Для генерації ключа AES використовується функція `get_random_bytes`, яка генерує випадковий набір байтів заданої довжини:

```
key = get_random_bytes(16)  # 16 байтів для AES-128
```

При шифруванні файлу використовано режим шифрування `AES.MODE_EAX` для отримання аутентифікованого шифротексту. Для шифрування був розроблений наступний код:

```
def encrypt(message):
    if message.document:
        # Отримання файлу для шифрування
        file_info = bot.get_file(message.document.file_id)
        file_to_encrypt = bot.download_file(file_info.file_path)
            # Зчитування ключа з файлу
        with open("encryption_key_received.key", "rb") as received_key_file:
            key = received_key_file.read()
            # Ініціалізація AES
        cipher = AES.new(key, AES.MODE_EAX)
            # Шифрування файлу
```

```
        ciphertext, tag = cipher.encrypt_and_digest(file_to_encrypt)
            # Збереження зашифрованого файлу
        with open("encrypted_file.txt", "wb") as encrypted_file:
            [encrypted_file.write(x) for x in (cipher.nonce, tag, ciphertext)]
            # Відправлення зашифрованого файлу користувачу
        bot.send_document(message.chat.id, open("encrypted_file.txt", "rb"))
    }
```

Цей код є частиною Telegram-бота і відповідає за шифрування файлів за допомогою алгоритму шифрування AES (Advanced Encryption Standard). Розглянемо його дії крок за кроком:

1. Отримання файлу для шифрування:
- Зчитується інформація про файл, який користувач відправив боту.
- Ідентифікатор файлу (file_id) витягується з об'єкта message, який містить інформацію про повідомлення.
2. Зчитування ключа з файлу:
- Ключ для AES зчитується з файлу з назвою "encryption_key_received.key".
3. Ініціалізація AES:
- Створюється новий об'єкт AES, використовуючи отриманий ключ і режим AES.MODE_EAX.
4. Шифрування файлу:
- Використовуючи створений об'єкт AES, викликається метод encrypt_and_digest, щоб зашифрувати вміст файлу.
- Отримуються зашифрований текст (ciphertext) та тег (tag).
5. Збереження зашифрованого файлу:
- Зберігається зашифрований файл, включаючи параметри, такі як nonce (ініціалізаційний вектор), тег і сам зашифрований текст.
6. Відправлення зашифрованого файлу користувачу:
- Відправляється отриманий зашифрований файл користувачеві через Telegram.

Для розшифрування використовується той самий ключ та режим роботи. Для розшифрування використовується наступний код:

```
def decrypt(message):
    if message.document:
        file_info = bot.get_file(message.document.file_id)
        file_to_decrypt = bot.download_file(file_info.file_path)
        with open("decryption_key_received.key", "rb") as received_key_file:
            key = received_key_file.read()
        # Разделяем байты на nonce, tag и ciphertext
        nonce = file_to_decrypt[:16]
        tag = file_to_decrypt[16:32]
        ciphertext = file_to_decrypt[32:]
        cipher = AES.new(key, AES.MODE_EAX, nonce=nonce)
        decrypted_file = cipher.decrypt(ciphertext)
        bot.send_document(message.chat.id, decrypted_file)
```

Розглянемо його дії крок за кроком:

1. Отримання файлу для розшифрування:
- Зчитується інформація про файл, який користувач відправив боту.
- Ідентифікатор файлу (file_id) витягується з об'єкта message, який містить інформацію про повідомлення.
2. Зчитування ключа з файлу:
- Ключ для розшифрування AES зчитується з файлу з назвою "decryption_key_received.key".
3. Розбивка байтів на nonce, tag і ciphertext:
- Зашифрований файл розбивається на три частини: nonce (ініціалізаційний вектор), tag та ciphertext (зашифрований текст).
4. Розшифрування файлу:
- Створюється об'єкт AES, використовуючи отриманий ключ, nonce і режим AES.MODE_EAX.

- Викликається метод decrypt для розшифрування зашифрованого тексту.

5. Відправлення розшифрованого файлу користувачу:

- Відправляється отриманий розшифрований файл користувачеві через Telegram.

Таким чином, реалізація AES використовуючи бібліотеку `Cryptodome` у проекті дозволяє шифрувати та розшифровувати файли з використанням сучасних стандартів безпеки.

Режим шифрування AES.MODE_EAX є одним з аутентифікованих режимів роботи AES. Його особливість полягає в тому, що він надає не тільки конфіденційність (шифрування) даних, але й аутентифікацію, тобто забезпечує впевненість у тому, що дані не були змінені.

3.1.2 Реалізація алгоритму шифрування DES

Алгоритм шифрування DES (Data Encryption Standard) використовується для захисту конфіденційності інформації шляхом шифрування даних.

Код генерації ключа виглядає наступним чином:

```
@bot.message_handler(func=lambda message: message.text == "Сгенерувати ключ DES")
def generate_des_key(message):
    key = get_random_bytes(8)  # Генерація 8-байтного ключа DES
    with open("des_encryption_key.key", "wb") as key_file:
        key_file.write(key)
    bot.send_document(message.chat.id, open("des_encryption_key.key", "rb"))
```

Цей код відповідає за генерацію ключа DES та надсилання його у вигляді документа користувачеві. Розглянемо його крок за кроком:

1. Тригер функції: Ця функція викликається, коли користувач відправляє повідомлення з текстом "Сгенерувати ключ DES".

2. Генерація ключа: Використовуючи функцію `get_random_bytes(8)`, генерується випадковий ключ DES довжиною 8 байтів.

3. Збереження ключа: Згенерований ключ записується у файл "des_encryption_key.key" у бінарному режимі (`wb`).

4. Надсилання ключа користувачеві: Файл, в якому збережений згенерований ключ, відсилається як документ користувачеві за допомогою `bot.send_document`.

Отже, результатом виконання цієї функції є те, що користувач отримує випадково згенерований ключ DES у вигляді документа, який може бути використаний для шифрування та розшифрування файлів за допомогою DES.

Для виконання шифрування даних був розроблений наступний код:

```
def encryptdes(message):
    if message.document:
        file_info = bot.get_file(message.document.file_id)
        file_to_encrypt = bot.download_file(file_info.file_path)
        with open("des_encryption_key.key", "rb") as received_key_file:
            key = received_key_file.read()
        cipher = des(key, CBC, IV=b"\0\0\0\0\0\0\0\0", pad=None, padmode=PAD_PKCS5)
        ciphertext = cipher.encrypt(file_to_encrypt)
        with open("encrypted_file_des.txt", "wb") as encrypted_file:
            encrypted_file.write(ciphertext)
        bot.send_document(message.chat.id, open("encrypted_file_des.txt", "rb"))
```

Цей код відповідає за шифрування файлу методом DES, використовуючи попередньо отриманий ключ. Розглянемо, як він працює:

1. Перевірка наявності документа: Функція перевіряє, чи користувач відправив повідомлення у вигляді документа (за допомогою умови `if message.document`).

2. Отримання інформації про файл: Якщо документ відправлено, використовуючи `bot.get_file`, отримується інформація про файл, який користувач надіслав.

3. Завантаження файлу для шифрування та ключа: Функція завантажує вміст отриманого документа, який представляє файл для шифрування методом DES. Також завантажує ключ DES з файлу "des_encryption_key.key".

4. Шифрування файлу: Використовуючи ключ і параметри шифрування DES (режим CBC, IV), створюється об'єкт шифрування `cipher`. Потім використовується цей об'єкт для шифрування вмісту файлу за допомогою методу `encrypt`. Результат (шифртекст) записується в змінну `ciphertext`.

5. Запис шифртексту у файл: Шифртекст записується у файл "encrypted_file_des.txt" у бінарному режимі.

6. Відправлення шифрованого файлу: За допомогою `bot.send_document`, бот відправляє користувачеві шифрований файл.

Отже, ця функція виконує процес шифрування файлу методом DES і відправляє шифрований файл користувачеві.

Для реалізації розшифровки файлу була реалізована функція def decryptdes. Код виглядає наступним чином:

```
def decryptdes(message):
    if message.document:
        file_info = bot.get_file(message.document.file_id)
        file_to_decrypt = bot.download_file(file_info.file_path)
        with open("decryption_key_received.key", "rb") as received_key_file:
            key = received_key_file.read()
        cipher = des(key, CBC, IV=b"\0\0\0\0\0\0\0\0", pad=None, padmode=PAD_PKCS5)
        decrypted_data = cipher.decrypt(file_to_decrypt)
```

```
            with open("decrypted_file_des.txt", "wb") as decrypted_file:
                decrypted_file.write(decrypted_data)
            bot.send_document(message.chat.id, open("decrypted_file_des.txt", "rb"))
```

Цей код відповідає за розшифрування файлу, який був зашифрований методом DES. Розглянемо його крок за кроком:

1. Перевірка наявності документа: Функція перевіряє, чи користувач відправив повідомлення у вигляді документа (за допомогою умови `if message.document`).

2. Отримання інформації про файл для розшифрування: Якщо документ відправлено, використовуючи `bot.get_file`, отримується інформація про файл, який користувач надіслав для розшифрування.

3. Завантаження файлу для розшифрування та ключа: Функція завантажує вміст отриманого документа, який представляє зашифрований файл DES. Також завантажує ключ DES для розшифрування з файлу "decryption_key_received.key".

4. Розшифрування файлу: Використовуючи ключ і параметри розшифрування DES (режим CBC, IV), створюється об'єкт розшифрування `cipher`. Потім використовується цей об'єкт для розшифрування вмісту файлу за допомогою методу `decrypt`. Результат (розшифрований текст) записується в змінну `decrypted_data`.

5. Запис розшифрованого тексту у файл: Розшифрований текст записується у файл "decrypted_file_des.txt" у бінарному режимі.

6. Відправлення розшифрованого файлу: За допомогою `bot.send_document`, бот відправляє користувачеві розшифрований файл.

Отже, ця функція виконує процес розшифрування файлу, який був зашифрований методом DES, і відправляє розшифрований файл користувачеві.

3.1.3 Реалізація алгоритму шифрування IDEA

У даному дослідженні використовується алгоритм шифрування IDEA (International Data Encryption Algorithm). IDEA є блочним шифром з 64-бітним блоком даних і 128-бітним ключем. IDEA був обраний для реалізації в даному проекті з метою забезпечення безпеки обміну файлами в месенджері Telegram.

Алгоритм IDEA був обраний з ряду конкуруючих шифрів через свою стійкість до криптоаналітичних атак і надійність у захисті конфіденційності даних. Його висока швидкість і низький рівень обчислювальних витрат роблять його ідеальним для застосування в додатках реального часу, таких як Telegram-бот.

Реалізація IDEA в проекті використовує бібліотеку pyCryptodome для Python. Код Telegram-бота містить обробники подій для шифрування та розшифрування файлів за допомогою алгоритму IDEA.

Розроблений код виглядає для генерації ключів виглядає наступним чином:

```
def generate_idea_key_message(message):
    key = generate_idea_key()
    with open("idea_encryption_key.key", "wb") as key_file:
        key_file.write(key)
    bot.send_document(message.chat.id, open("idea_encryption_key.key", "rb"))
```

Цей фрагмент коду відповідає за генерацію ключа IDEA та його подальше відправлення користувачеві через Telegram. Розглянемо кожен рядок коду:

1. `key = generate_idea_key()`: Викликає функцію `generate_idea_key()`, яка, ймовірно, містить логіку для генерації ключа IDEA. Припустимо, що ця функція створює випадковий ключ IDEA або використовує який-небудь інший метод для його генерації.

2. `with open("idea_encryption_key.key", "wb") as key_file:`: Відкриває файл з ім'ям "idea_encryption_key.key" в режимі запису бінарних даних (`"wb"`). Це важливо, оскільки ключі зазвичай представляються бінарними даними.

3. `key_file.write(key)`: Записує згенерований ключ `key` у відкритий файл `key_file`.

4. `bot.send_document(message.chat.id, open("idea_encryption_key.key", "rb"))`: Відправляє згенерований ключ у чат, з якого було отримано повідомлення `message`. `open("idea_encryption_key.key", "rb")` відкриває файл у режимі читання бінарних даних, і `bot.send_document` відправляє його як документ користувачу.

Отже, цей фрагмент коду реалізує процес генерації ключа IDEA, збереження його у файлі "idea_encryption_key.key" та надсилання користувачеві через Telegram у вигляді документу.

Для шифрування був розроблений наступний код:

```
def encryptidea(message):
    if message.document:
        file_info = bot.get_file(message.document.file_id)
        file_to_encrypt = bot.download_file(file_info.file_path)
        with open("idea_encryption_key.key", "rb") as received_key_file:
            key = received_key_file.read()
                cipher = Cipher(algorithms.IDEA(key), modes.ECB())
        encryptor = cipher.encryptor()
        padder = padding.PKCS7(64).padder()
        padded_data = padder.update(file_to_encrypt) + padder.finalize()
        ciphertext = encryptor.update(padded_data) + encryptor.finalize()
        with open("encrypted_file_idea.txt", "wb") as encrypted_file:
            encrypted_file.write(ciphertext)
        bot.send_document(message.chat.id, open("encrypted_file_idea.txt",
"rb"))
```

Цей фрагмент коду відповідає за шифрування файлу за допомогою алгоритму IDEA та відправлення зашифрованого файлу користувачеві через Telegram. Розглянемо кожен рядок коду:

1. `if message.document:`: Перевіряє, чи містить повідомлення `message` документ (файл).

2. `file_info = bot.get_file(message.document.file_id)`: Отримує інформацію про файл, який долучений до повідомлення.

3. `file_to_encrypt = bot.download_file(file_info.file_path)`: Завантажує вміст файлу для шифрування.

4. `with open("idea_encryption_key.key", "rb") as received_key_file:`: Відкриває файл із збереженим ключем IDEA у режимі читання бінарних даних.

5. `key = received_key_file.read()`: Зчитує ключ IDEA з файлу.

6. `cipher = Cipher(algorithms.IDEA(key), modes.ECB())`: Ініціалізує об'єкт шифрування IDEA, використовуючи зчитаний ключ та режим ECB.

7. `encryptor = cipher.encryptor()`: Створює об'єкт, який буде використовуватися для шифрування.

8. `padder = padding.PKCS7(64).padder()`: Використовує доповнювач PKCS7 для доповнення даних до блока, якщо це необхідно.

9. `padded_data = padder.update(file_to_encrypt) + padder.finalize()`: Доповнює дані та отримує доповнені дані.

10. `ciphertext = encryptor.update(padded_data) + encryptor.finalize()`: Шифрує доповнені дані та отримує зашифрований текст.

11. `with open("encrypted_file_idea.txt", "wb") as encrypted_file:`: Відкриває файл для запису зашифрованого тексту у режимі бінарних даних.

12. `encrypted_file.write(ciphertext)`: Записує зашифрований текст у файл.

13. `bot.send_document(message.chat.id, open("encrypted_file_idea.txt", "rb"))`: Відправляє зашифрований файл у чат, з якого було отримано

повідомлення `message`, через Telegram. Файл відкривається у режимі читання бінарних даних перед відправленням.

Для розшифрування був розроблений наступний код:

```
def decryptidea(message):
    if message.document:
        file_info = bot.get_file(message.document.file_id)
        file_to_decrypt = bot.download_file(file_info.file_path)
        with open("idea_encryption_key.key", "rb") as received_key_file:
            key = received_key_file.read()
        ciphertext = file_to_decrypt
        cipher = Cipher(algorithms.IDEA(key), modes.ECB())  # Режим ECB для прикладу, оберіть відповідний режим
        decryptor = cipher.decryptor()
        unpadder = padding.PKCS7(64).unpadder()
        decrypted_padded_data = decryptor.update(ciphertext) + decryptor.finalize()
        plaintext = unpadder.update(decrypted_padded_data) + unpadder.finalize()
        with open("decrypted_file.txt", "wb") as decrypted_file:
            decrypted_file.write(plaintext)
        bot.send_document(message.chat.id, open("decrypted_file.txt", "rb"))
```

Цей фрагмент коду відповідає за розшифрування файлу, який був зашифрований алгоритмом IDEA, та відправлення розшифрованого файлу користувачеві через Telegram. Розглянемо кожен рядок коду:

1. `if message.document:`: Перевіряє, чи містить повідомлення `message` документ (файл).

2. `file_info = bot.get_file(message.document.file_id)`: Отримує інформацію про файл, який долучений до повідомлення.

3. `file_to_decrypt = bot.download_file(file_info.file_path)`: Завантажує вміст зашифрованого файлу для подальшого розшифрування.

4. `with open("idea_encryption_key.key", "rb") as received_key_file:`: Відкриває файл із збереженим ключем IDEA у режимі читання бінарних даних.

5. `key = received_key_file.read()`: Зчитує ключ IDEA з файлу.

6. `ciphertext = file_to_decrypt`: Отримує шифртекст, який потрібно розшифрувати.

7. `cipher = Cipher(algorithms.IDEA(key), modes.ECB())`: Ініціалізує об'єкт розшифрування IDEA, використовуючи зчитаний ключ та режим ECB (у цьому прикладі).

8. `decryptor = cipher.decryptor()`: Створює об'єкт, який буде використовуватися для розшифрування.

9. `unpadder = padding.PKCS7(64).unpadder()`: Використовує доповнювач PKCS7 для видалення доповнення.

10. `decrypted_padded_data = decryptor.update(ciphertext) + decryptor.finalize()`: Розшифровує шифртекст та отримує розшифровані дані, які можуть бути доповнені.

11. `plaintext = unpadder.update(decrypted_padded_data) + unpadder.finalize()`: Видаляє доповнення та отримує оригінальні дані.

12. `with open("decrypted_file.txt", "wb") as decrypted_file:`: Відкриває файл для запису розшифрованих даних у режимі бінарних даних.

13. `decrypted_file.write(plaintext)`: Записує розшифровані дані у файл.

14. `bot.send_document(message.chat.id, open("decrypted_file.txt", "rb"))`: Відправляє розшифрований файл у чат, з якого було отримано повідомлення `message`, через Telegram. Файл відкривається у режимі читання бінарних даних перед відправленням.

3.1.4 Реалізація алгоритму шифрування RSA

У коді використовується бібліотека Cryptodome для реалізації алгоритму RSA. Основні етапи реалізації включають:

1. Генерація ключів.

2. Збереження ключів у файлах.

3. Завантаження ключів з файлів.

4. Шифрування файлу за допомогою публічного ключа.

5. Розшифрування файлу за допомогою приватного ключа.

6. Обробник команди для генерації ключів.

7. Обробник команди для шифрування файлу.

8. Обробник команди для розшифрування файлу.

Реалізація алгоритму шифрування RSA у боті Telegram дозволяє користувачеві зашифрувати та розшифрувати файли, забезпечуючи конфіденційність передаваних даних. Код забезпечує використання RSA разом із зручним інтерфейсом чат-бота.

Для генерації ключів був розроблений наступний код:

```
def generate_rsa_keys():
    key = RSA.generate(2048)  # Змініть розмір ключа за потребою
    private_key = key.export_key()
    public_key = key.publickey().export_key()
    return private_key, public_key
```

Функція `generate_rsa_keys` генерує пару ключів RSA із заданою довжиною 2048 біт. Далі наведено пояснення для кожного етапу функції:

1. `key = RSA.generate(2048)`: Генерується пара ключів RSA, приватний та публічний, із довжиною 2048 біт.

2. `private_key = key.export_key()`: Експортується приватний ключ у вигляді байтового рядка. Приватний ключ є конфіденційним і призначеним тільки для власника ключа.

3. `public_key = key.publickey().export_key()`: Експортується публічний ключ у вигляді байтового рядка. Публічний ключ може бути розповсюджений і використовується для шифрування даних для власника приватного ключа.

4. `return private_key, public_key`: Повертається пара ключів у вигляді кортежу, де перший елемент - приватний ключ, а другий - публічний ключ.

Ця функція може використовуватися для генерації ключів для подальшого використання в алгоритмах шифрування RSA.

Для того шифрування даних був використаний наступний код:

```
def perform_encryption(message, public_key):
    if message.document:
        file_info = bot.get_file(message.document.file_id)
        file_to_encrypt = bot.download_file(file_info.file_path)
            cipher_rsa = PKCS1_OAEP.new(public_key)
        encrypted_file = cipher_rsa.encrypt(file_to_encrypt)
            with open("encrypted_file.txt", "wb") as encrypted_file_file:
          encrypted_file_file.write(encrypted_file)
            bot.send_document(message.chat.id, open("encrypted_file.txt", "rb"))
```

Ця функція використовує публічний ключ RSA для шифрування файлу. Нижче наведено пояснення для кожного етапу функції:

1. `if message.document:`: Перевіряє, чи містить отримане повідомлення документ (файл).

2. `file_info = bot.get_file(message.document.file_id)`: Отримує інформацію про файл, використовуючи ідентифікатор файлу.

3. `file_to_encrypt = bot.download_file(file_info.file_path)`: Завантажує вміст файлу для подальшого шифрування.

4. `cipher_rsa = PKCS1_OAEP.new(public_key)`: Створюється об'єкт шифрування з використанням алгоритму PKCS#1 OAEP та публічного ключа.

5. `encrypted_file = cipher_rsa.encrypt(file_to_encrypt)`: Застосовується шифрування до вмісту файлу за допомогою публічного ключа.

6. `with open("encrypted_file.txt", "wb") as encrypted_file_file:`: Відкривається новий файл для запису зашифрованого вмісту.

7. `encrypted_file_file.write(encrypted_file)`: Записується зашифрований вміст у файл.

8. `bot.send_document(message.chat.id, open("encrypted_file.txt", "rb"))`: Відправляється зашифрований файл назад у чат.

Ця функція може бути використана для шифрування файлів за допомогою алгоритму RSA та публічного ключа, отриманого в результаті генерації ключів.

Для розшифрування файлу був розроблений наступний код:

```
def perform_decryption(message, private_key):
    if message.document:
        file_info = bot.get_file(message.document.file_id)
        file_to_decrypt = bot.download_file(file_info.file_path)
            cipher_rsa = PKCS1_OAEP.new(private_key)
        decrypted_file = cipher_rsa.decrypt(file_to_decrypt)
            bot.send_document(message.chat.id, decrypted_file)
```

Ця функція використовує приватний ключ RSA для розшифрування зашифрованого файлу. Нижче наведено пояснення для кожного етапу функції:

1. `if message.document:`: Перевіряє, чи містить отримане повідомлення документ (файл).

2. `file_info = bot.get_file(message.document.file_id)`: Отримує інформацію про файл, використовуючи ідентифікатор файлу.

3. `file_to_decrypt = bot.download_file(file_info.file_path)`: Завантажує вміст зашифрованого файлу для подальшого розшифрування.

4. `cipher_rsa = PKCS1_OAEP.new(private_key)`: Створюється об'єкт розшифрування з використанням алгоритму PKCS#1 OAEP та приватного ключа.

5. `decrypted_file = cipher_rsa.decrypt(file_to_decrypt)`: Застосовується розшифрування до вмісту зашифрованого файлу за допомогою приватного ключа.

6. `bot.send_document(message.chat.id, decrypted_file)`: Відправляється розшифрований файл назад у чат.

3.1.5 Реалізація алгоритму шифрування ElGamal

Для реалізації алгоритму шифрування ElGamal у чат-боті на платформі Telegram було використано бібліотеку Telebot та кілька інших бібліотек для криптографічних операцій. Даний алгоритм базується на використанні криптосистеми з відкритим ключем, яка включає в себе генерацію ключів, шифрування та розшифрування файлів.

Процес генерації ключів для ElGamal передбачає створення випадкового генератора, який використовується для генерації приватного і публічного ключів. Відповідні функції зберігають ключі у відповідних файлах для подальшого використання.

Код має наступний вигляд:

```
def generate_elgamal_keysel():
    random_generator = Random.new().read
    key = ElGamal.generate(2048, random_generator)
    private_key = key
    public_key = key.publickey()
    return private_key, public_key
```

Ця функція `generate_elgamal_keysel` використовує алгоритм ElGamal для генерації пари ключів: приватного ключа та відкритого ключа (публічного ключа). Розглянемо кожен етап роботи функції:

1. `random_generator = Random.new().read`: Створюється об'єкт генератора випадкових чисел (PRNG - Pseudorandom Number Generator) за допомогою класу `Random` з модуля `Crypto.Random`. Цей генератор буде використовуватися для створення випадкових значень у подальших етапах.

2. `key = ElGamal.generate(2048, random_generator)`: Генерується ключ ElGamal з параметром довжини ключа 2048 біт. Для цього використовується функція `generate` класу `ElGamal`, яка приймає розмір ключа та генератор випадкових чисел.

3. `private_key = key`: Приватний ключ (`private_key`) стає просто об'єктом ключа ElGamal, який містить в собі приватну інформацію.

4. `public_key = key.publickey()`: З отриманого приватного ключа отримується відкритий ключ (публічний ключ) за допомогою методу `publickey()`. Публічний ключ є об'єктом, який містить лише відкриту інформацію.

5. `return private_key, public_key`: Пара ключів повертається як кортеж з приватним і публічним ключами.

Отже, функція `generate_elgamal_keysel` генерує пару ключів для алгоритму ElGamal і повертає їх у вигляді кортежу `(private_key, public_key)`.

При виборі опції "Зашифрувати файл ElGamal", користувач має можливість вибрати файл для шифрування. Після вибору файлу генерується сесійний ключ, який потім шифрується за допомогою алгоритму RSA разом із використанням публічного ключа отримувача. Сам файл шифрується за допомогою ElGamal, використовуючи згенерований сесійний ключ.

Розроблений код має наступний вигляд:

```
def encrypt_fileel(message, public_key):
    bot.send_message(message.chat.id, " Будь-ласка, відправте файл для шифрування:")
    bot.register_next_step_handler(message, lambda m: perform_encryptionel(m, public_key))
def perform_encryptionel(message, public_key):
    if message.document:
        file_info = bot.get_file(message.document.file_id)
        file_to_encrypt = bot.download_file(file_info.file_path)
            session_key = Random.new().read(32)
        cipher_rsa = PKCS1_OAEP.new(public_key)
        encrypted_session_key = cipher_rsa.encrypt(session_key)
            cipher_elgamal = ElGamal.new(public_key)
        ciphertext = cipher_elgamal.encrypt(file_to_encrypt, session_key)
```

```
            with open("encrypted_file.txt", "wb") as encrypted_file_file:
        encrypted_file_file.write(encrypted_session_key + ciphertext)
            bot.send_document(message.chat.id, open("encrypted_file.txt",
"rb"))
```

Цей код відповідає за шифрування файлів за допомогою алгоритму ElGamal у вашому чат-боті на платформі Telegram. Розглянемо кроки, які відбуваються в процесі роботи цього коду:

1. Користувач викликає функцію `encrypt_fileel`, написавши команду чи обравши відповідний пункт меню. Запускається відправлення повідомлення: " Будь-ласка, відправте файл для шифрованн:".

2. Далі викликається функція `bot.register_next_step_handler`, яка реєструє наступну функцію для обробки наступного повідомлення в чаті. У цьому випадку це лямбда-функція `lambda m: perform_encryptionel(m, public_key)`.

3. При отриманні наступного повідомлення в чаті, викликається лямбда-функція `perform_encryptionel`, яка приймає повідомлення `message` і публічний ключ `public_key`.

4. У функції `perform_encryptionel` перевіряється, чи було отримано документ у повідомленні (файл для шифрування).

5. Якщо файл був отриманий, витягується інформація про файл, його ідентифікатор, та викликається `bot.download_file` для отримання вмісту файлу.

6. Генерується сесійний ключ (`session_key`) довільною кількістю байтів (у цьому випадку 32 байти).

7. За допомогою алгоритму RSA (PKCS1_OAEP) шифрується сесійний ключ за допомогою публічного ключа.

8. Створюється об'єкт ElGamal (`cipher_elgamal`) на основі публічного ключа, та використовуючи згенерований сесійний ключ, шифрується вміст файлу.

9. Записується шифрований сесійний ключ та шифрований текст у файл "encrypted_file.txt".

10. Отриманий шифрований файл надсилається в чат за допомогою `bot.send_document`.

Отже, цей код дозволяє користувачам шифрувати файли за допомогою алгоритму ElGamal, використовуючи введений публічний ключ.

При обранні опції "Розшифрувати файл ElGamal", користувач повинен вибрати файл для розшифрування. Приватний ключ використовується для розшифрування сесійного ключа, який далі використовується для розшифрування основного вмісту файлу за допомогою алгоритму ElGamal.

Код має наступний вигляд:

```
def decrypt_fileel(message, private_key):
    bot.send_message(message.chat.id, " Будь-ласка, відправте файл для Розшифрування:")
    bot.register_next_step_handler(message, lambda m: perform_decryptionel(m, private_key))
def perform_decryptionel(message, private_key):
    if message.document:
        file_info = bot.get_file(message.document.file_id)
        file_to_decrypt = bot.download_file(file_info.file_path)
        session_key_length = 32
        encrypted_session_key = file_to_decrypt[:session_key_length]
        ciphertext = file_to_decrypt[session_key_length:]
        cipher_rsa = PKCS1_OAEP.new(private_key)
        session_key = cipher_rsa.decrypt(encrypted_session_key)
        cipher_elgamal = ElGamal.new(private_key)
        decrypted_text = cipher_elgamal.decrypt(ciphertext, session_key)
        bot.send_document(message.chat.id, decrypted_text)
```

Цей код відповідає за розшифрування файлів, які були зашифровані алгоритмом ElGamal у вашому чат-боті на платформі Telegram. Розглянемо, як працює цей код:

1. Користувач викликає функцію `decrypt_fileel`, написавши команду чи обравши відповідний пункт меню. Запускається відправлення повідомлення: " Будь-ласка, відправте файл для шифрованн:".

2. Далі викликається функція `bot.register_next_step_handler`, яка реєструє наступну функцію для обробки наступного повідомлення в чаті. У цьому випадку це лямбда-функція `lambda m: perform_decryptionel(m, private_key)`.

3. При отриманні наступного повідомлення в чаті, викликається лямбда-функція `perform_decryptionel`, яка приймає повідомлення `message` і приватний ключ `private_key`.

4. У функції `perform_decryptionel` перевіряється, чи було отримано документ у повідомленні (зашифрований файл).

5. Якщо файл був отриманий, витягується інформація про файл, його ідентифікатор, та викликається `bot.download_file` для отримання вмісту файлу.

6. Зчитуються перші 32 байти зашифрованого файлу, які є зашифрованим сесійним ключем.

7. Використовуючи алгоритм RSA (PKCS1_OAEP) та приватний ключ, розшифровується зашифрований сесійний ключ.

8. Створюється об'єкт ElGamal (`cipher_elgamal`) на основі приватного ключа, та використовуючи розшифрований сесійний ключ, розшифровується вміст файлу.

9. Розшифрований текст (вміст файлу) надсилається в чат за допомогою `bot.send_document`.

Отже, цей код дозволяє користувачам розшифрувати файли, які були зашифровані алгоритмом ElGamal, використовуючи введений приватний ключ.

3.2 Розробка програмного коду для взаємодії з користувачем

В даному пункті розглянемо створення програмного коду, який взаємодіє з користувачем та реалізує вибір та застосування конкретних шифрувальних алгоритмів.

```
@bot.message_handler(commands=['start'])
def start(message):
    markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
    item_simetric = types.KeyboardButton("Симетричне")
    item_asimetric = types.KeyboardButton("Асиметричне")
    markup.add(item_simetric, item_asimetric)
    bot.send_message(message.chat.id, " Виберіть метод шифрування:", reply_markup=markup)
    @bot.message_handler(func=lambda message: message.text == "Симетричне")
def sim(message):
    markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
    item_aes = types.KeyboardButton("AES")
    item_des = types.KeyboardButton("DES")
    item_idea = types.KeyboardButton("IDEA")
    markup.add(item_aes, item_des,item_idea)
    bot.send_message(message.chat.id, " Виберіть метод шифрування:", reply_markup=markup)

@bot.message_handler(func=lambda message: message.text == "Асиметричне")
def asim(message):
    markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
    item_RSA = types.KeyboardButton("RSA")
    item_ECC = types.KeyboardButton("ElGamal")
```

markup.add(item_RSA, item_ECC)

bot.send_message(message.chat.id, " Виберіть метод шифрування:", reply_markup=markup)

Цей код реалізує обробники повідомлень для команди `/start` та вибору методу шифрування (симетричного або асиметричного). Розглянемо кожну частину коду:

1. `@bot.message_handler(commands=['start'])`: Ця функція буде викликана, коли користувач введе команду `/start`. Відбувається налаштування клавіатури з кнопками для вибору методу шифрування (симетричного чи асиметричного).

2. `@bot.message_handler(func=lambda message: message.text == "Симетричне")`: Ця функція викликається, якщо користувач обрав симетричний метод шифрування. Знову створюється клавіатура, але тепер з кнопками для вибору конкретного симетричного алгоритму (AES, DES, IDEA).

3. `@bot.message_handler(func=lambda message: message.text == "Асиметричне")`: Ця функція викликається, якщо користувач обрав асиметричний метод шифрування. Також створюється клавіатура з кнопками для вибору конкретного асиметричного алгоритму (RSA, ElGamal).

Отже, в реалізації взаємодії з користувачем спробовано реалізувати простий інтерфейс вибору методу шифрування та конкретного алгоритму. Користувач може вибрати, чи хоче він використовувати симетричний чи асиметричний метод, і подальше обрати конкретний алгоритм залежно від своїх потреб.

Загальна блок схема для симетричного шифрування виглядає наступним чином:

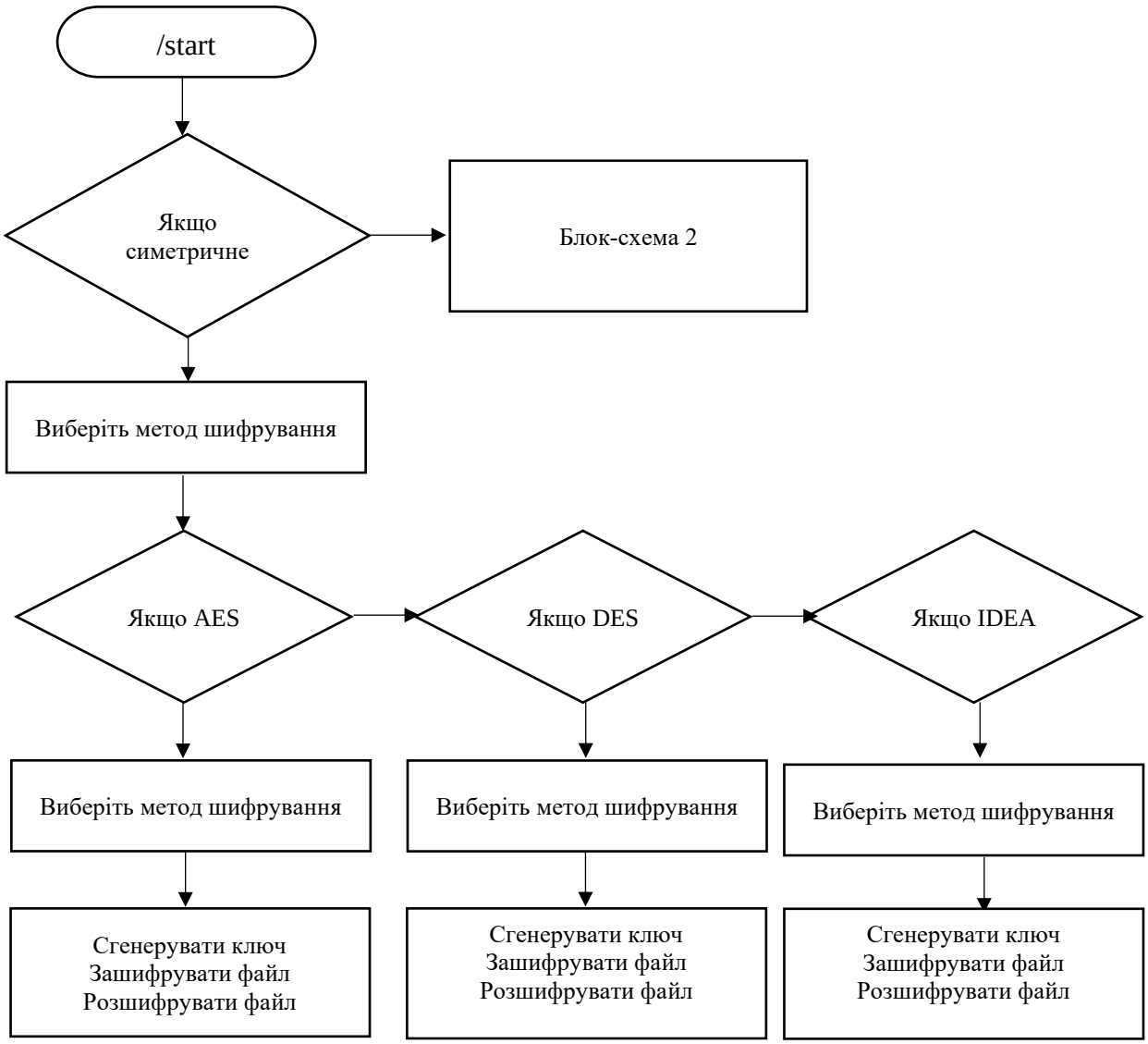

Рис.3.1. Блок-схема для симетричного шифрування

Продовження блок-схеми для асиметричного шифрування, зображена на рис. 3.2.

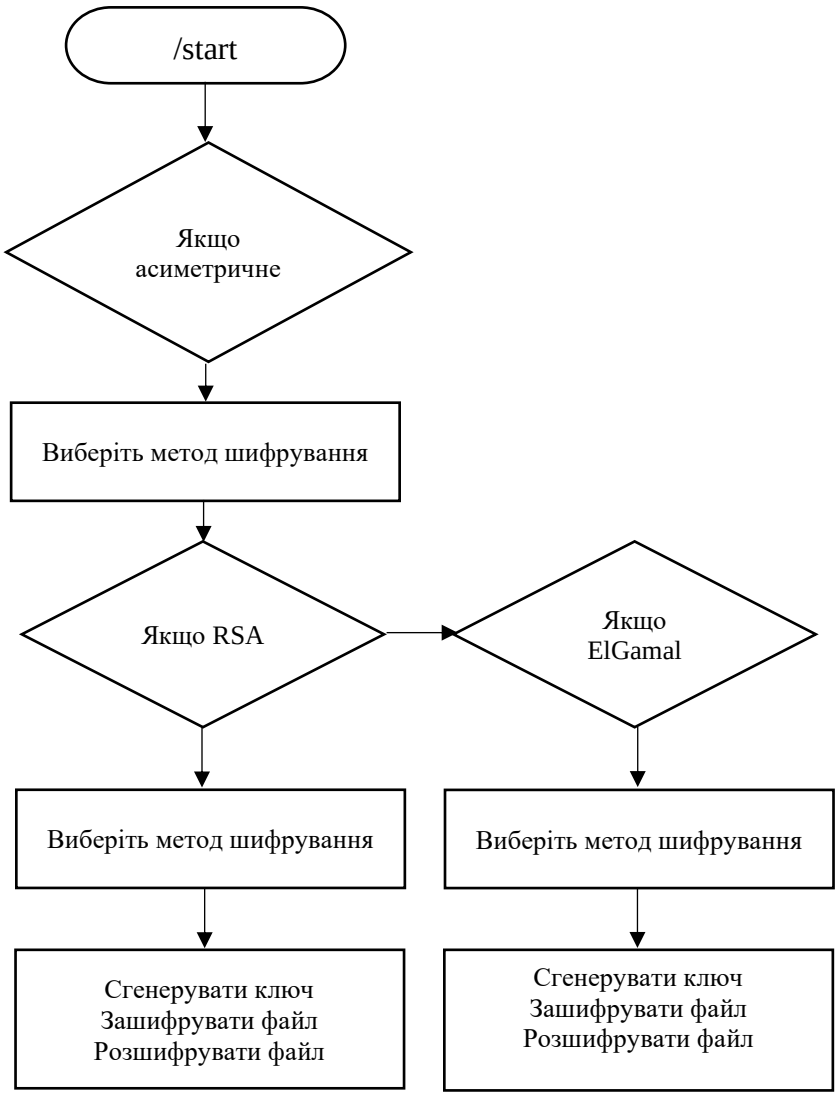

Рис.3.2. Блок-схема для симетричного шифрування

3.3. Висновки до розділу

У цьому розділі детально описана реалізація програмного засобу для захисту даних, включаючи розробку алгоритмів шифрування та програмного коду для взаємодії з користувачем.

Проаналізовані та висвітлені деталі реалізації різних алгоритмів шифрування, зокрема AES, DES, IDEA, RSA та ElGamal. Кожен алгоритм втілений в програмному засобі, враховуючи їхні унікальні характеристики та особливості. Важливою є забезпечення ефективності та безпеки в реалізації кожного алгоритму, а також їх взаємодії у загальній структурі програми.

Описано деталі реалізації алгоритму AES з урахуванням ключових аспектів та важливих параметрів.

Надана інформація про реалізацію алгоритму DES, включаючи деталі ключової довжини та режимів роботи.

Описана реалізація алгоритму RSA, включаючи генерацію ключів та процес шифрування/розшифрування.

Подано деталі реалізації алгоритму ElGamal, включаючи генерацію ключів та операції шифрування/розшифрування.

Пояснена структуру та функціональність програмного коду, який взаємодіє з користувачем. Визначено основні можливості та інтерфейс для забезпечення зручного та інтуїтивно зрозумілого користування програмою.

В цілому, цей розділ надає повний огляд реалізації програмного засобу, включаючи алгоритми шифрування та взаємодію з користувачем. Розглянуті аспекти дозволяють зрозуміти, як програма функціонує та як користувач може взаємодіяти з нею для забезпечення безпеки своїх даних.

Розділ 4. ІНТЕГРАЦІЯ ПРОГРАМНОГО ЗАСОБУ В ТЕЛЕГРАМ ТА ТЕСТУВАННЯ

4.1 Інтеграція бота в телеграм та тестування

Інтегрування бота в Telegram - це процес створення, налаштування та запуску бота для взаємодії з користувачами через платформу Telegram.

Для початку треба зайти в офіційний телеграм бот BotFather. Інформація про бота зображена на рис.4.1.

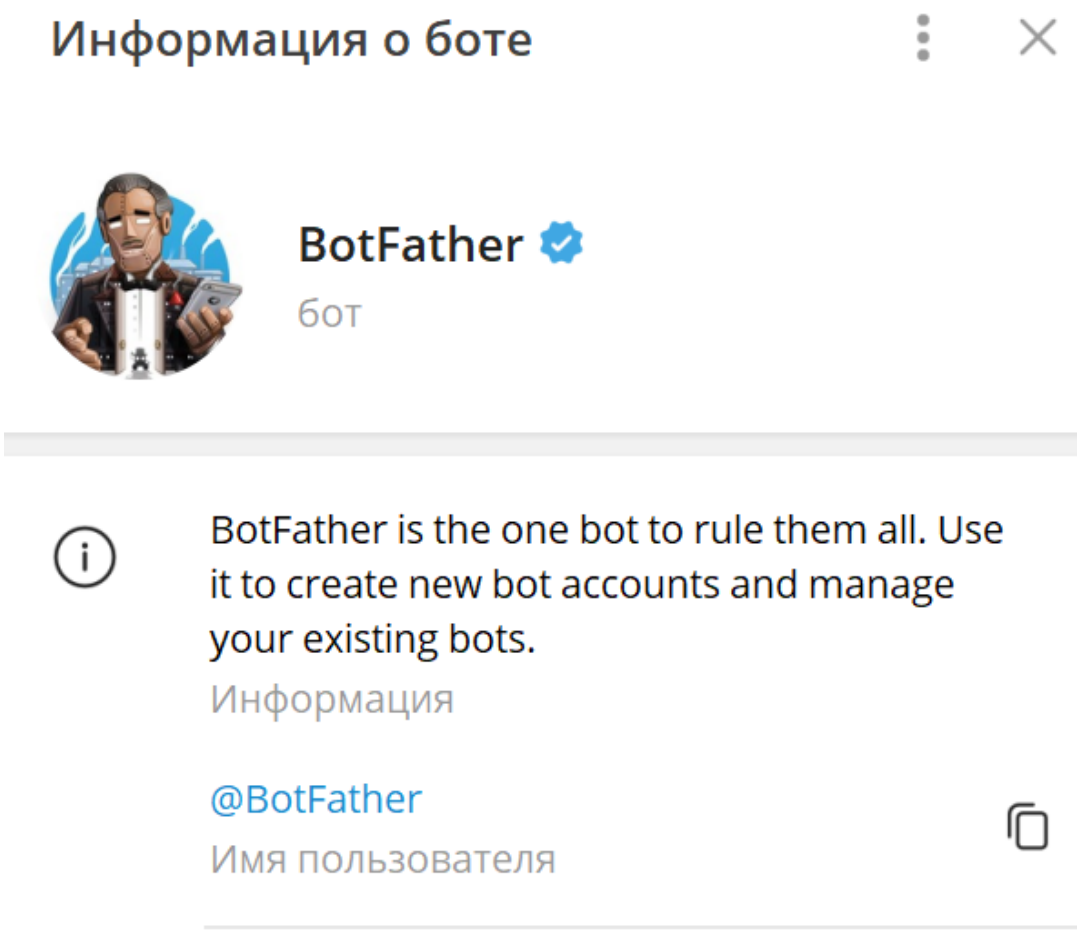

Рис.4.1. Офіційний телеграм бот BotFather

Для того, щоб отримати токен для нашого боту треба вести команду /start. Після цього обирається назви боту (рис.4.2.)

Рис.4.2. Реєстрація назви боту

Після цього обирається username для боту, та отримуємо токен для боту. Повідомлення яке отримуємо зображено на рис.4.3.

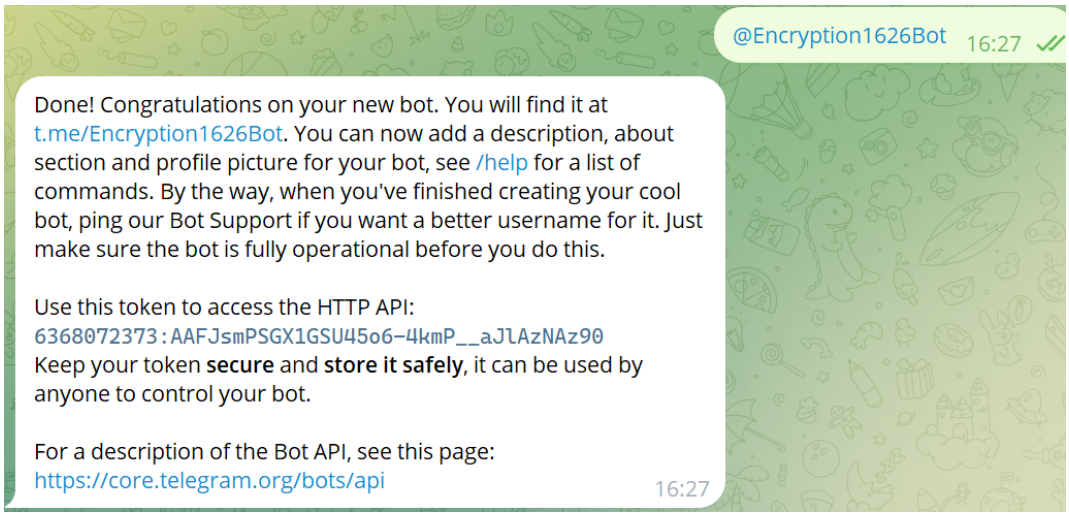

Рис.4.3. Токен телеграм боту

Цей токен потрібен для того щоб, вставити його в код та для того щоб телегарам міг зв’язатися з ним.

Для того щоб код взаємодіяв з телеграмом імпортується бібліотека Telebot.

telebot - це бібліотека для мови програмування Python, яка надає інтерфейс для спрощення взаємодії з Telegram API для створення та управління ботами в Telegram. Ця бібліотека робить розробку ботів більш зручною та ефективною, дозволяючи програмістам зосередитися на функціональності свого бота, не вдаючись у складності роботи з Telegram API напряму.

Основні можливості telebot включають:

Зручний інтерфейс для обробки подій:

- telebot надає зручний механізм обробки подій, таких як отримання повідомлень, фотографій, команд, тощо. Ви можете використовувати декоратори для визначення функцій-обработчиків для різних типів подій.

Легка взаємодія з Telegram API:

- Завдяки telebot вам не потрібно безпосередньо працювати з HTTP-запитами до Telegram API. Бібліотека надає високорівневі методи для надсилання повідомлень, редагування повідомлень, роботи з клавіатурою, тощо.

Підтримка клавіатур та відповідей:

- telebot дозволяє легко створювати клавіатури для взаємодії з користувачами. Це може бути корисно для створення інтерактивних ботів.

Робота з мультимедіа:

- Ви можете легко обробляти фотографії, документи, аудіо та інші мультимедійні дані за допомогою telebot.

Підтримка веб-хуків:

- telebot дозволяє використовувати веб-хуки для отримання повідомлень, що полегшує розгортання бота на власному сервері.

Інші корисні функції:

- Бібліотека також надає інші корисні можливості, такі як відправлення голосових повідомлень, робота з онлайн-режимом, отримання інформації про користувачів, тощо.

Як результат програмний засіб виглядає наступним чином в інформації телеграму (рис.4.4.)

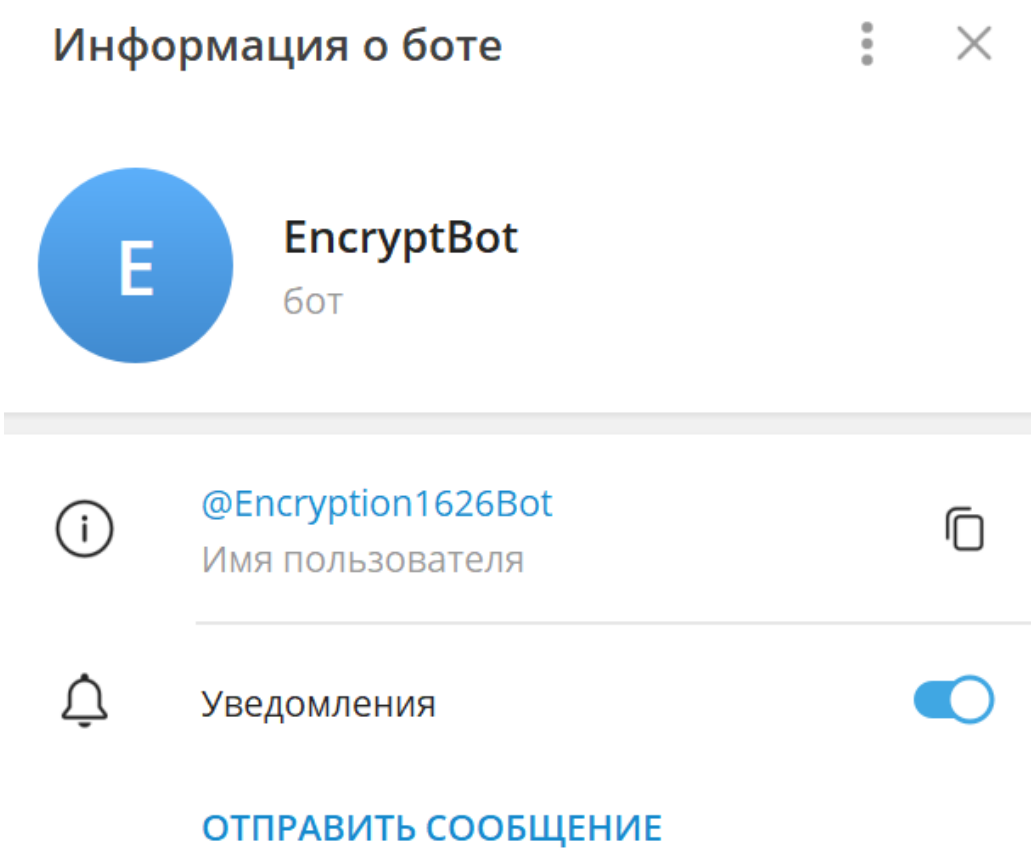

Рис.4.4. Інформація про телеграм бота

При введенні команди /start з’являється вікно, яке зображено на рис. 4.5.

Рис. 4.5. Вікно вибору методу шифрування.

Після вибору методу шифрування відкривається наступне вікно для вибору методу шифрування AES,DES,IDEA, представлено на рис.4.6.

Рис.4.6. Вибір методу шифрування

Натиснувши відповідний метод користувачу відкривається вікно вибору наступних кроків. А саме: Згенерувати ключ, Зашифрувати файл, Розшифрувати файл. Вигляд вікна представлений на рис.4.7.

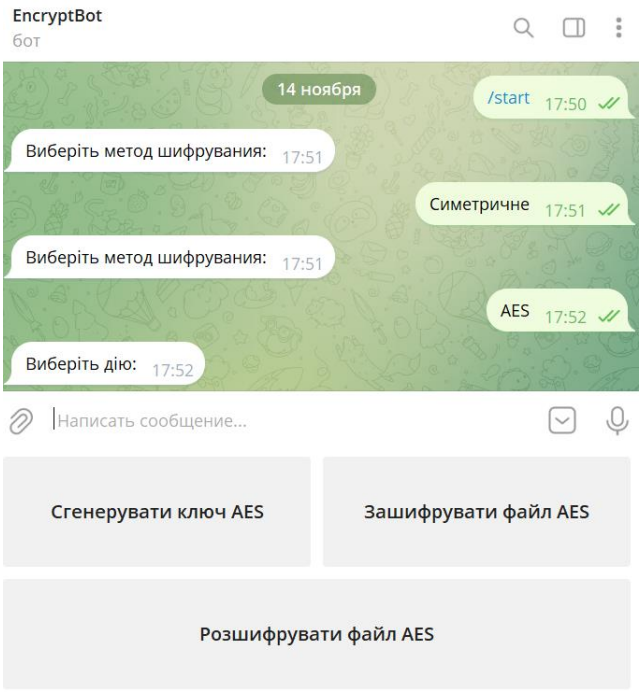

Рис.4.7. Вибір наступних дій

Натиснувши кнопку генерувати ключ бот нам відправляє ключ у вигляді файлу с назвою encryption_key.key.

Для того, зашифрувати чи розшифрувати файл натискаємо відповідну кнопку. Після цього бот відправить повідомлення “Будь-ласка відправте файл ключа.” Відправивши файл ключа бот запросить файл для шифрування.

Після того як програма отримує ці файли відбудеться процес шифрування та бот відправить зашифрований файл. Процес зображений на рис.4.8.

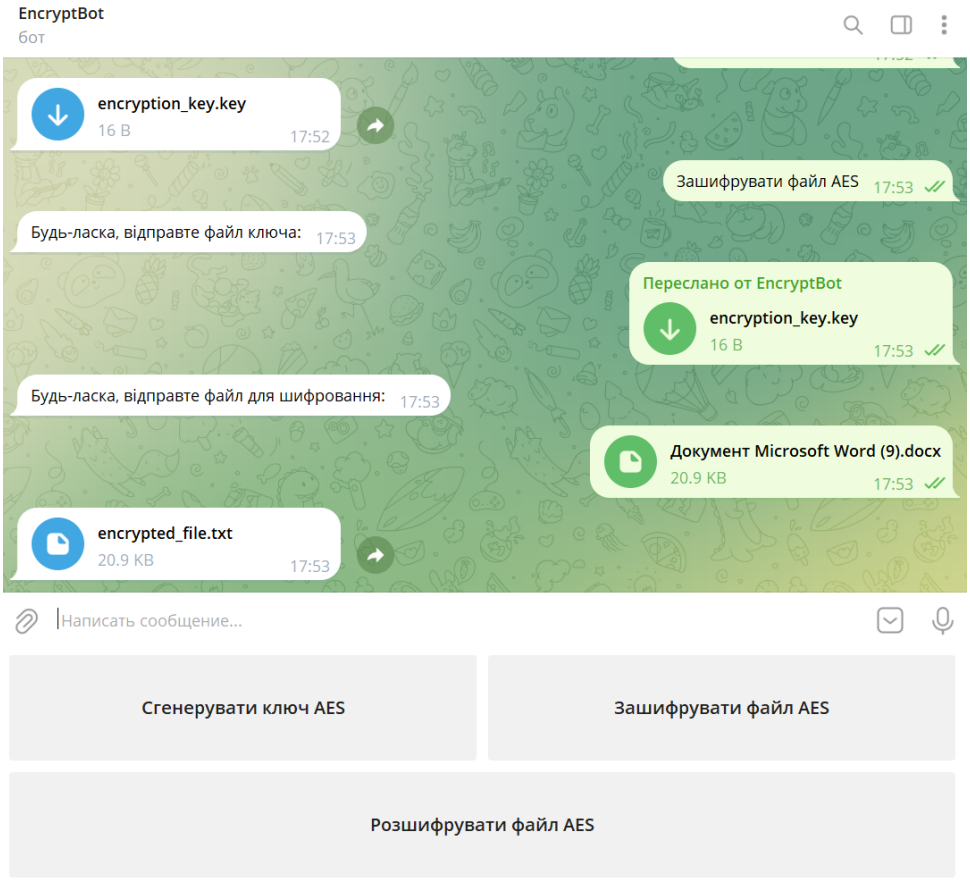

Рис.4.8. Процес зашифрування файлу

Для розшифрування файлу потрібно натиснути клавішу Розшифрувати файл. Далі на вимогу бота відправити файл ключа та зашифрований файл. Процес зображений на рис.4.9.

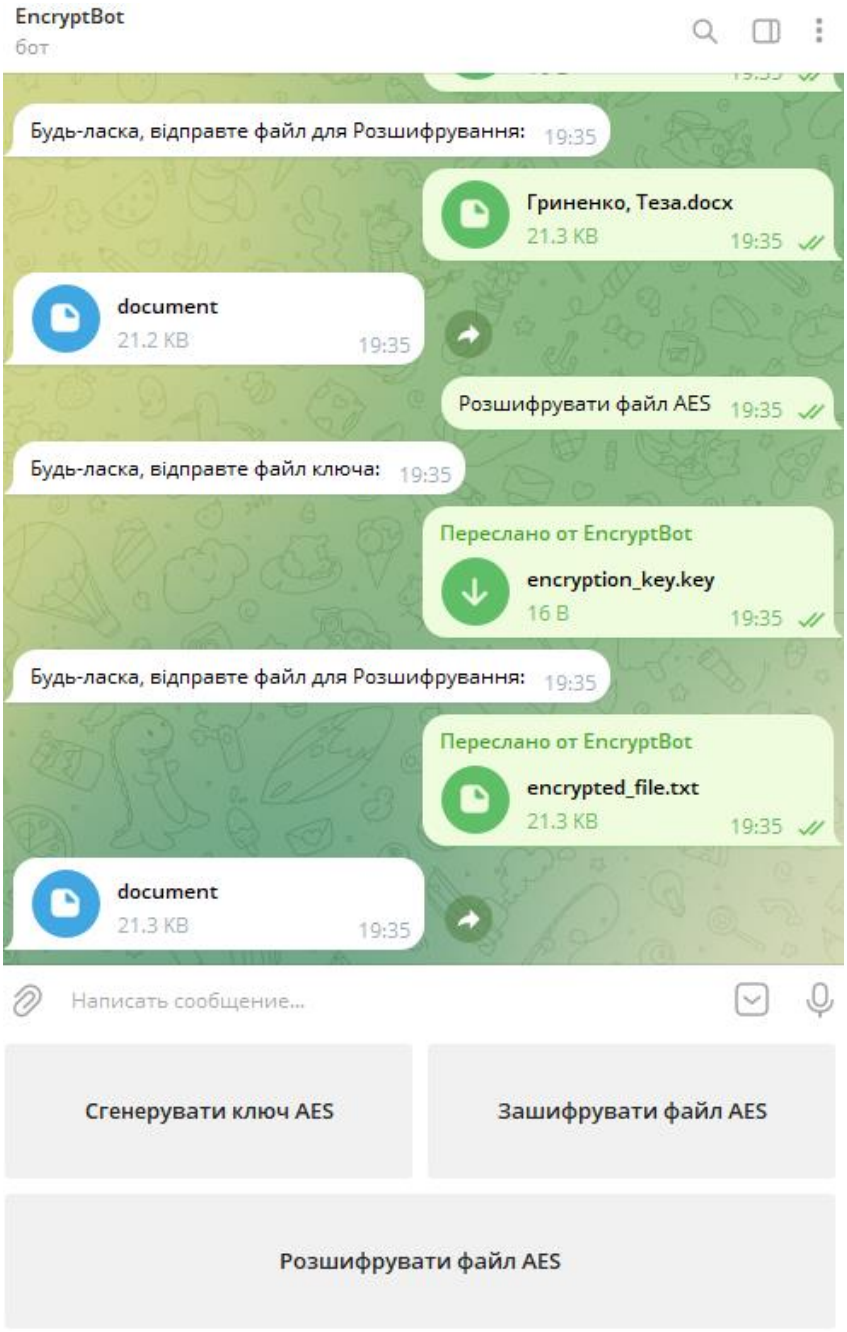

Рис.4.9. Процес розшифрування файлу

Швидкодія телеграм бота залежить від трьох основних факторів, а саме:

1. Пристрою на якому запущений цей телеграм бот.

2. Швидкість інтернету на пристрою на якому запущений телеграм бот.

3. Швидкість інтернету на пристрою яким користувач користується при роботі з цим програмним засобом.

Якість шифрування полягає у тому, що файл залишається цілим без змін самих даних. Надійність полягає у тому, що зашифрований файл без ключа не дозволяє отримати данні.

При відкритті зашифрованого файлу, зображеного на рис.4.10., можна побачити, що файл не можливо прочитати.

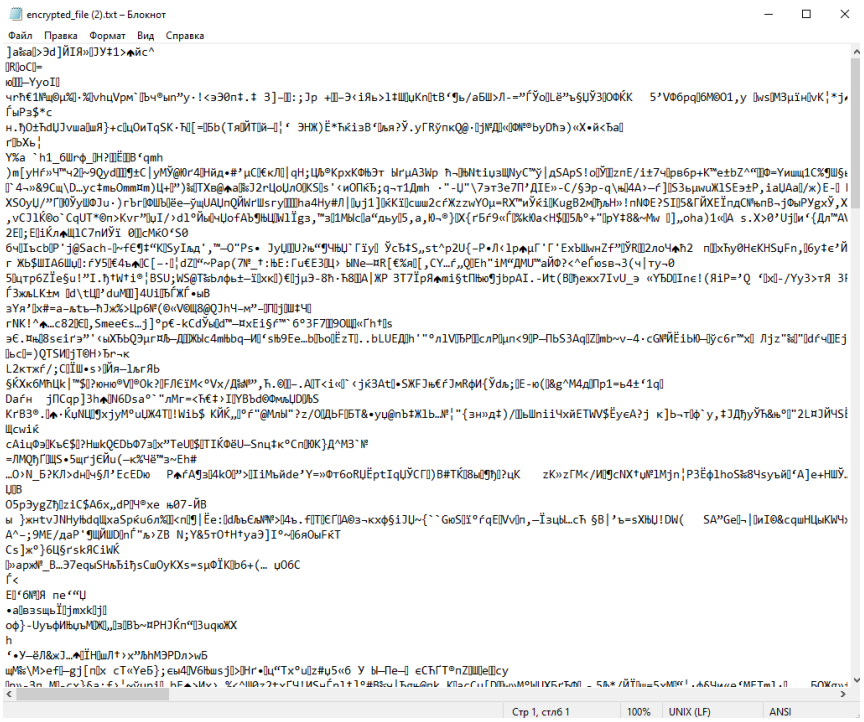

Рис. 4.10. Зашифрований файл

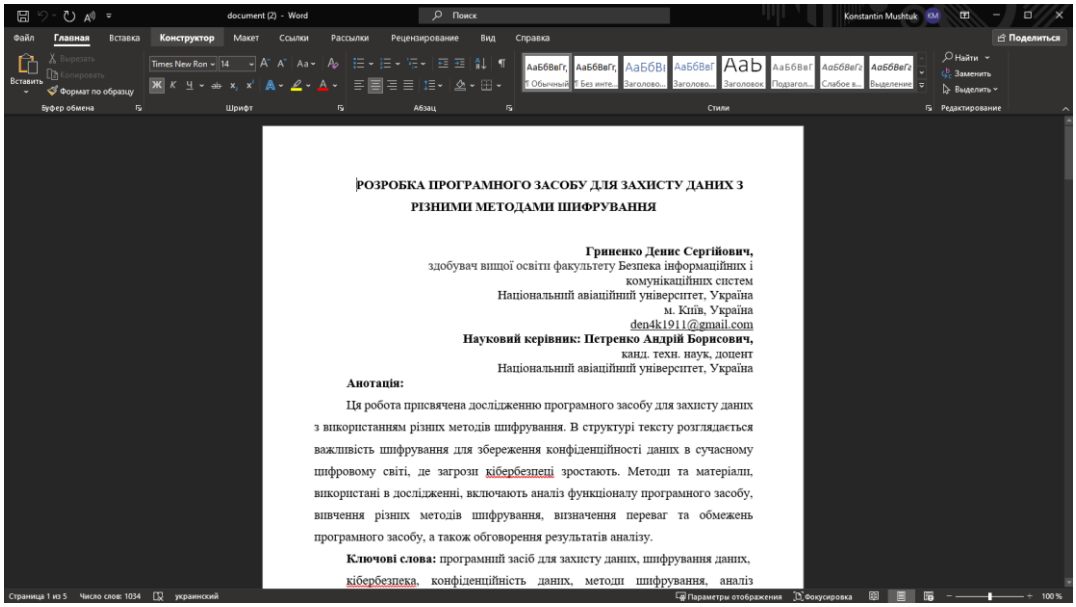

Рис. 4.11. Розшифрований файл

Після розшифрування файлу (рис.4.11.) файл знову можливо прочитати.

4.2 Криптоаналіз зашифрованих файлів методом Brute Force

Для того, щоб перевірити надійність розробленого програмного засобу було використано програмний засіб CrypTool.

"Cryptool" - це відкрите програмне забезпечення, яке призначене для аналізу та шифрування інформації. Ця програма використовується у сфері криптографії для вивчення, тестування та застосування різноманітних криптографічних методів. Основною метою Cryptool є надання засобів для навчання, експериментування та вивчення криптографії, а також для розвитку та тестування криптографічних алгоритмів.

Основні функції Cryptool включають:

1. Аналіз криптографічних методів: Cryptool дозволяє користувачам досліджувати та аналізувати різноманітні криптографічні методи, включаючи симетричне та асиметричне шифрування, хеш-функції, цифрові підписи та інші.

2. Навчання та освіта: Програма має навчальний характер і надає можливості для вивчення основ криптографії, пропонуючи користувачам інтерактивні інструменти та матеріали.

3. Експерименти з алгоритмами: Cryptool дозволяє користувачам випробувати різні криптографічні алгоритми та спостерігати за їхнім впливом на дані.

4. Реалізація алгоритмів: Крім аналізу, програма також дозволяє реалізовувати криптографічні алгоритми та перевіряти їхню працездатність.

5. Тестування стійкості криптосистем: Cryptool може використовуватися для тестування стійкості криптографічних систем та алгоритмів до атак.

Cryptool використовується як навчальний інструмент в університетах, а також у дослідницьких проектах у галузі криптографії. Важливо враховувати, що стан програми та її можливості можуть змінюватися з часом, тому корисно

перевіряти останні версії та документацію для отримання актуальної інформації.

Для того, щоб провести криптоаналіз, потрібно вибрати вкладку Analysis /Symmetric Encryption та вибрати метод.

Для початку був зроблений криптоаналіз методу AES, зображений на рис.4.12. Як можна побачити для того щоб зламати шифр методом грубої сили потрібно 1,3e+025years.

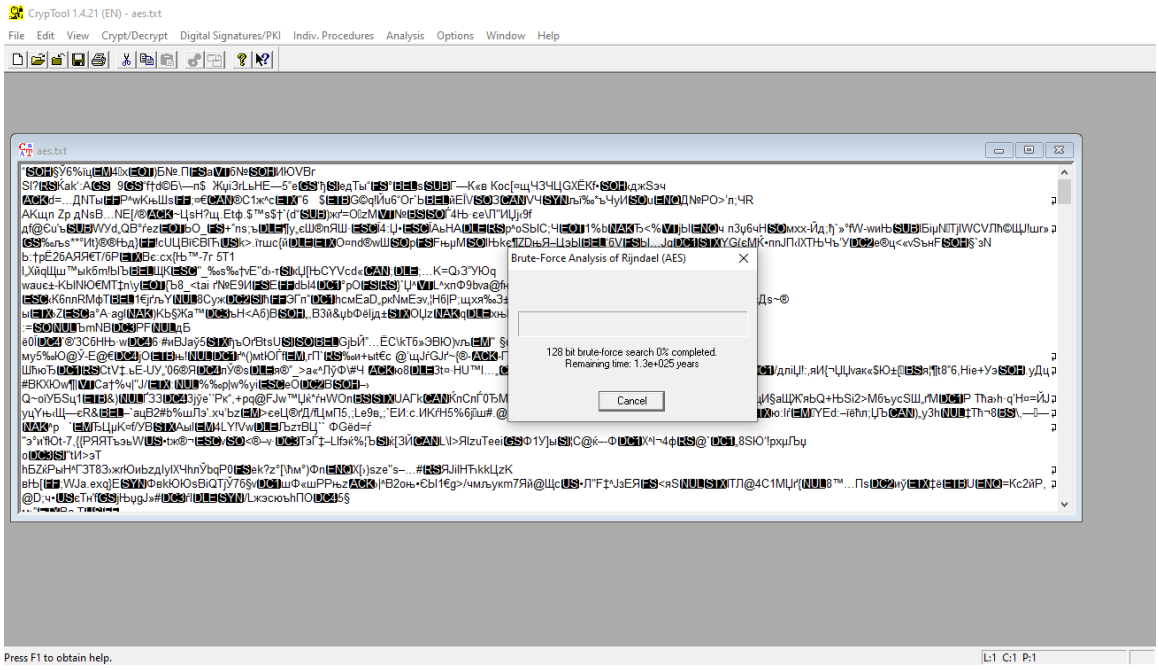

Рис.4.12. Криптоаналіз методу AES

Далі був зроблений криптоаналіз методу AES, зображений на рис.4.13. Як можна побачити для того щоб зламати шифр методом грубої сили потрібно 2,4e+004years.

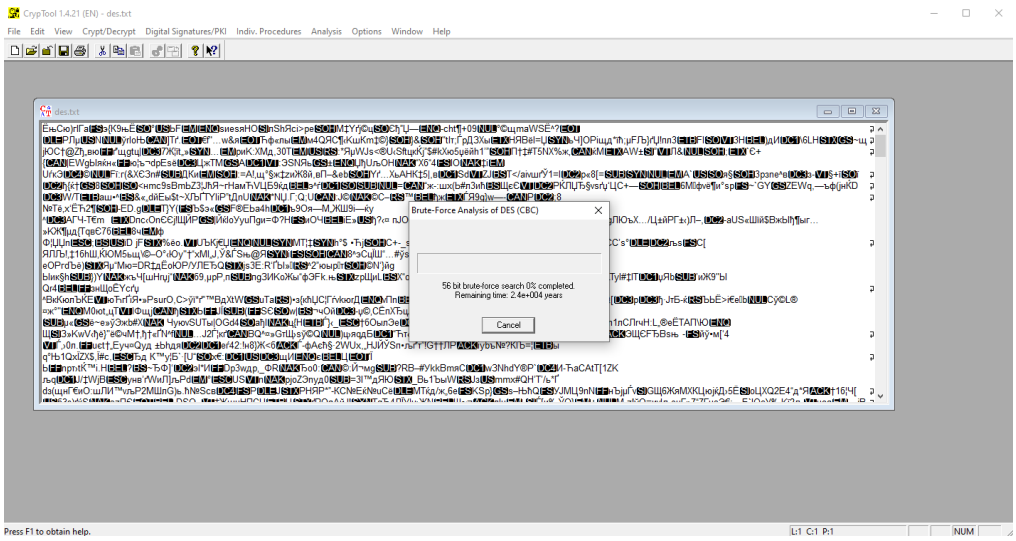

Рис.4.13. Криптоаналіз методу DES

Був зроблений криптоаналіз методу AES, зображений на рис.4.14. Як можна побачити для того щоб зламати шифр методом грубої сили потрібно 1,3e+026years.

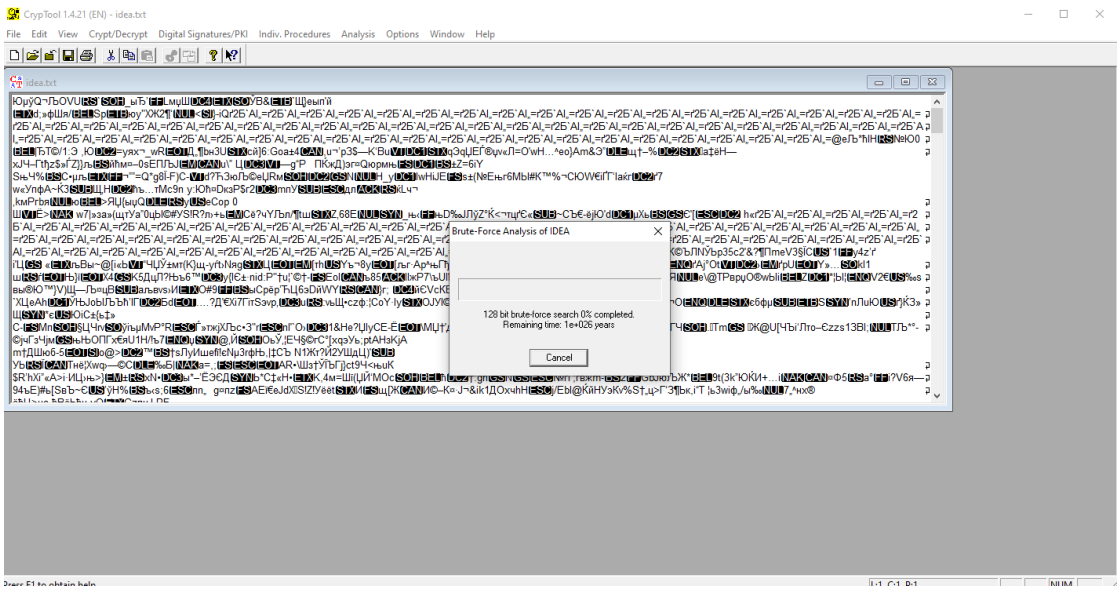

Рис.4.14. Криптоаналіз методу IDEA

Криптоаналіз методом "brute force" для алгоритму RSA також є практично неможливим через складність завдання факторизації великих простих чисел, що лежить в основі безпеки RSA. Факторизація великих чисел потребує експоненційного часу та ресурсів, що робить цей метод неможливим для практичного використання.

4.3 Розробка рекомендацій щодо зберігання ключа та захисту даних

Для забезпечення надійного зберігання ключа та ефективного захисту даних пропонується розробити і впровадити систему комплексних рекомендацій. Ці рекомендації спрямовані на забезпечення конфіденційності, цілісності та доступності інформації, а також на мінімізацію ризиків втрати ключа та несанкціонованого доступу до нього.

Фізичне забезпечення:

Забезпечення безпеки фізичного місця, де зберігається ключ, визначається рядом обов'язкових заходів:

- Обладнані та захищені приміщення: Рекомендується використовувати спеціально призначені приміщення, обладнані системами безпеки, такими як системи відеоспостереження, контролю доступу та виявлення несанкціонованих вторгнень. Це забезпечить контроль за фізичним доступом до ключа.

- Обмежений доступ: Забезпечення фізичного місця, що зберігає ключ, повинно передбачати обмежений доступ лише для авторизованих осіб. Встановлення фізичних бар'єрів, таких як біометричні системи або кард-читачі, допоможе попереджати несанкціонований доступ.

Шифрування ключа:

Шифрування ключа перед зберіганням є важливим етапом для захисту від потенційних атак. Деталізація цього процесу включає:

- Використання сучасних алгоритмів: Застосування надійних алгоритмів шифрування, таких як AES (Advanced Encryption Standard) чи RSA (Rivest–Shamir–Adleman), гарантує високий рівень захисту ключа від криптоаналітичних атак.

- Управління ключами шифрування: Система повинна включати ефективний механізм управління ключами, що дозволяє створювати, обновляти та вилучати ключі. Це підвищує безпеку системи в цілому.

Резервне копіювання:

Регулярне створення та зберігання резервних копій ключа є важливим елементом стратегії зберігання ключа:

- Регулярність копіювання: Ключі повинні регулярно створюватися та зберігатися у відокремлених резервних копіях. Це дозволяє зберігати актуальні дані та готовити систему до можливих втрат або пошкоджень ключа.

- Фізичне розташування резервних копій: Зберігайте резервні копії у фізично безпечних місцях, де доступ мають лише відповідальні особи. Це зменшить ризик втрати даних у випадку аварії чи інциденту.

Всі ці заходи спрямовані на створення комплексної системи зберігання ключа, що поєднує фізичні та криптографічні методи для найвищого рівня безпеки та надійності.

Моніторинг доступу:

Здійснюйте систематичний моніторинг доступу до даних, особливо до ключа. Зафіксовані непередбачені або підозрілі дії повинні негайно залучати увагу адміністраторів безпеки.

Системи журналювання: Встановіть системи журналювання, які реєструють всі дії, пов'язані з доступом до ключа та даних. Це дозволяє вчасно виявляти аномалії та небезпечні дії.

Системи виявлення інцидентів: Використовуйте системи виявлення інцидентів для автоматичного виявлення аномалій в поведінці системи та негайного реагування на можливі загрози.

Оновлення безпеки:

Регулярно оновлюйте заходи безпеки, включаючи програмне та апаратне забезпечення. Це дозволяє уникнути використання вразливостей системи з метою отримання несанкціонованого доступу.

Патчі та оновлення: Регулярно встановлюйте оновлення та патчі для програмного забезпечення та операційних систем. Це допомагає виправляти виявлені вразливості та забезпечує стійкість до сучасних загроз.

Аудит безпеки: Проводьте аудит безпеки системи для виявлення можливих слабких місць та дотримання встановлених безпекових стандартів.

Фізичний та логічний розділ доступу:

Обмежте фізичний та логічний доступ до системи, де зберігаються дані та ключ. Використовуйте різні рівні доступу для різних користувачів згідно їхніх повноважень.

Розділ доступу: Встановлюйте різні рівні фізичного та логічного доступу в залежності від ролі користувача. Це зменшує ризик несанкціонованого доступу та внутрішніх загроз.

Багаторівневий контроль: Використовуйте багаторівневий контроль доступу, де особи мають доступ лише до тих ресурсів та функцій, які необхідні для їхніх обов'язків.

Загальна мета цих рекомендацій полягає в створенні комплексної системи захисту даних, що включає постійний моніторинг та оновлення заходів безпеки, а також ретельний контроль над фізичним і логічним доступом. Це сприяє максимальному зниженню ризиків втрати ключа та несанкціонованого доступу до даних, забезпечуючи високий рівень безпеки та конфіденційності інформації.

4.4 Висновки до розділу

У цьому розділі детально описана інтеграція розробленого програмного засобу в месенджер Telegram, а також проведене тестування функціоналу. Крім того, враховані аспекти зберігання ключа та захисту даних.

Проаналізовані етапи та результати інтеграції бота в Telegram. Зазначено важливість забезпечення зручного та безпечного інтерфейсу в месенджері для користувача. Проведене тестування функціоналу, включаючи взаємодію з ботом, відправлення та отримання зашифрованих повідомлень.

Сформульовані рекомендації щодо безпечного зберігання ключів і забезпечення надійного захисту даних користувача. Звернута увага на важливість дотримання кращих практик безпеки, включаючи використання паролів високого рівня та регулярну зміну ключів.

У цілому, цей розділ вказує на успішну інтеграцію програмного засобу в популярний месенджер та успішно проведене тестування його функціоналу. Розроблені рекомендації забезпечують користувачів надійними інструкціями щодо збереження конфіденційності своїх ключів та даних під час використання програмного засобу.

Розділ 5. ОХОРОНА НАВКОЛИШНЬГО СЕРЕДОВИЩА

5.1. Завдання сучасної екології

Сучасна екологія стоїть перед великими викликами, пов'язаними зі змінами клімату, виснаженням ресурсів та загрозами для біорізноманіття. Задача наших часів - забезпечити гармонію між суспільством та навколишнім середовищем. Цей текст розглядає ключові завдання, які постають перед сучасною екологією.

1. Зменшення викидів та перехід до відновлювальних джерел енергії.

Одним із найактуальніших завдань екології є зниження викидів шкідливих речовин у повітря. Використання відновлювальних джерел енергії, таких як сонячна та вітрова, може виявитися ключовим етапом у досягненні цієї мети. Розвиток нових технологій для зберігання та передачі енергії також важливий. Громадська свідомість про енергоефективність та необхідність переходу до зелених технологій може стати драйвером для змін в енергетичному секторі.

Одним із перших кроків у зменшенні викидів є перехід до відновлювальних джерел енергії, таких як сонячна та вітрова. Вони є чистими та невичерпними джерелами енергії, що не видають шкідливих газів або інших забруднюючих речовин під час виробництва. Підтримка та розвиток цих технологій є важливим етапом у створенні сталої та екологічно чистої енергетичної системи.

Подальшим кроком є розвиток технологій для ефективного зберігання та передачі енергії. Зберігання великих обсягів енергії з відновлювальних джерел, таких як сонячна та вітрова, вимагає інноваційних рішень. Розвиток технологій зберігання, таких як акумулятори великої ємності та системи зеленої водородної енергії, дозволить ефективно використовувати відновлювальні ресурси та забезпечити стабільний енергетичний підйом.

Для досягнення значних змін у енергетичному секторі необхідно враховувати важливий аспект - громадську свідомість. Інформаційна кампанія та освіта громадян про переваги відновлювальних джерел енергії та енергоефективності можуть визначити успіх переходу до зеленого енергетичного майбутнього. Сприяння свідомості громади про позитивний вплив зелених технологій допоможе створити попит на сталі зміни в енергетичному секторі.

В цілому, перехід до сталої та екологічно чистої енергетики вимагає комплексного підходу, що охоплює технологічний розвиток, законодавчі зміни та активну участь громадськості. Тільки взаємодія всіх цих факторів може забезпечити успішність зусиль зниження викидів шкідливих речовин у повітря та перехід до сталої та екологічно чистої енергетики.

2. Збереження водних ресурсів та боротьба із забрудненням водойм.

Сучасний стан водних ресурсів свідчить про необхідність дій. Забруднення водойм хімічними речовинами, пластиковими відходами та іншими шкідливими речовинами може призвести до катастрофічних наслідків для водних екосистем та здоров'я людей. Сучасні технології очищення води та стратегії управління водними ресурсами повинні бути впроваджені на всіх рівнях суспільства.

Сучасний стан водних ресурсів свідчить про нагальну потребу в системних заходах для їхнього збереження та ефективного використання. Зростаюча кількість забруднень, експлуатація водойм та зміни клімату ставлять під загрозу екосистеми та життя тварин та рослин, які залежать від цих водних об'єктів.

Однією з головних проблем є забруднення водойм різноманітними шкідливими речовинами. Хімічні речовини від промисловості, сільського господарства та побутового використання можуть впливати на якість води, знижуючи її безпеку для людини та природи.

3. Захист біорізноманіття та відновлення екосистем.

Зменшення різноманіття видів є серйозною загрозою для екосистем та людського благополуччя. Знищення природних середовищ через вирубку лісів, забруднення та зміни клімату ставлять під загрозу існування багатьох видів. Завданням екологів є збереження та відновлення природних екосистем, створення охоронних зон та резерватів. Сприяння сталому використанню лісових ресурсів та відновлення родючості ґрунтів є ключовими аспектами цього завдання.

4. Розробка ефективних стратегій управління відходами.

Велика кількість відходів, які сьогодні викидаються в природу, створює проблеми забруднення та вичерпання природних ресурсів. Розробка ефективних стратегій управління відходами, включаючи їхню переробку та вторинне використання, є критичною для збереження навколишнього середовища. Сприяння усвідомленості громадськості про важливість роздільного збору та відповідального використання ресурсів може допомогти у подоланні цієї проблеми.

5. Розвиток екологічної освіти та підвищення екологічної свідомості.

Одним із ключових аспектів збереження природи є розвиток екологічної освіти. Люди повинні розуміти вплив своїх дій на навколишнє середовище та бути обізнаними з екологічними проблемами. Забезпечення доступу до інформації про екологічні питання, впровадження програм екологічної освіти в школах та університетах допоможе формувати екологічно свідомих громадян, готових приймати відповідальні рішення.

Сучасна екологія стоїть перед великими завданнями, але водночас вона пропонує можливості для розвитку нових технологій, стратегій управління та громадської свідомості. Збереження природи та забезпечення сталого розвитку вимагає спільних зусиль всіх сфер суспільства. Лише шляхом об'єднання зусиль ми зможемо забезпечити здорове навколишнє середовище для сучасних та майбутніх поколінь.

ВИСНОВКИ

Кваліфікаційна робота була спрямована на розробку програмного засобу для захисту даних з використанням різноманітних методів шифрування. За допомогою вивчення теоретичних аспектів шифрування в Розділі 1, визначення функціональності програмного засобу та вибору алгоритмів шифрування в Розділі 2, а також реалізації та тестування програмного засобу в Розділі 3, було досягнуто кілька важливих висновків.

1. Теоретичні аспекти шифрування:

В результаті аналізу основних понять і принципів шифрування було визначено, що сучасний захист вимагає використання різних методів шифрування, враховуючи типи симетричного та асиметричного шифрування.

2. Функціональність програмного засобу:

Обрані алгоритми шифрування, такі як AES, DES, IDEA, RSA та ElGamal, були впізнані як надійні для реалізації. Вибір мови програмування, бібліотек та середовища реалізації був обґрунтований для забезпечення оптимальності та продуктивності програмного засобу.

3. Реалізація програмного засобу:

Розроблено програмний засіб, який включає реалізацію обраних алгоритмів шифрування та взаємодію з користувачем через інтерфейс Telegram. Результати тестування підтвердили коректну роботу програмного засобу та його безпеку при обробці конфіденційної інформації.

Криптографічний метод захисту, безумовно, самий надійний метод захисту, так як охороняється безпосередньо сама інформація, а не доступ до неї.

4. Інтеграція в Telegram та рекомендації щодо безпеки:

Бот був успішно інтегрований в Telegram, надаючи користувачам зручний і безпечний спосіб використання програмного засобу. Розроблені рекомендації щодо зберігання ключів та захисту даних надають важливі вказівки для користувачів.

Якість шифрування полягає у тому, що файл залишається цілим без змін самих даних. Надійність полягає у тому, що зашифрований файл без ключа не дозволяє отримати данні.

Отже, розроблений програмний засіб виявився і надійним засобом захисту даних з використанням різноманітних методів шифрування. Реалізація програмного коду і його інтеграція в Telegram надають користувачам простий та безпечний інструмент для захисту їхніх конфіденційних даних. Робота вносить важливий внесок у сферу кібербезпеки та використовує передові технології для забезпечення конфіденційності та цілісності інформації в цифровому середовищі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Key Concepts in Encryption [Електронний ресурс] : SURVEILLANCE SELF-DEFENSE // - Режим доступу: World Wide Web. – URL: https://ssd.eff.org/module/key-concepts-encryption.
2. Що таке шифрування та як воно працює? [Електронний ресурс] : // - Режим доступу: https://www.kingston.com/ua/blog/data-security/what-is-encryption.
3. Засоби та методи захисту інформації [Електронний ресурс] : // - Режим доступу: World Wide Web. – URL: https://buklib.net/books/28625/.
4. Симметричное и асимметричное шифрование [Електронний ресурс] : Crypto News // - Режим доступу: World Wide Web. – URL: https://cryptonews.net/en/news/other/3374079/.
5. Симетричне та асиметричне шифрування [Електронний ресурс] : // - Режим доступу: https://academy.binance.com/uk/articles/symmetric-vs-asymmetric-encryption
6. Технології криптографії - Принцип безпеки та техніка криптографії [Електронний ресурс] : Education-WIKI.com // - Режим доступу: World Wide Web. – URL: https://uk.education-wiki.com/8487633-cryptography-techniques.
7. Технології криптографічного захисту інформації [Електронний ресурс] : // - Режим доступу: https://stud.com.ua/50144/informatika/tehnologiyi_kriptografichnogo_zahistu_informatsiyi
8. Інформаційні технології у криптографії [Електронний ресурс] : // - Режим доступу:
9. ЩО ТАКЕ АСИМЕТРИЧНЕ ТА СИМЕТРИЧНЕ ШИФРУВАННЯ? [Електронний ресурс]: // - Режим доступу: World Wide Web. – URL: https://www.websiterating.com/uk/vpn/glossary/what-is-asymmetric-symmetric-encryption/.

10. ПОРІВНЯННЯ СИМЕТРИЧНОГО І АСИМЕТРИЧНОГО ШИФРУВАННЯ [Електронний ресурс]: // - Режим доступу: https://openarchive.nure.ua/server/api/core/bitstreams/a47f80d7-cf00-4b39-aba6-31843899f002/contenthttps://openarchive.nure.ua/server/api/core/bitstreams/a47f80d7-cf00-4b39-aba6-31843899f002/content.
11. ЩО ТАКЕ ШИФРУВАННЯ AES-256 І ЯК ВОНО ПРАЦЮЄ? [Електронний ресурс]: // - Режим доступу: World Wide Web. – URL: https://www.websiterating.com/uk/cloud-storage/what-is-aes-256-encryption/.
12. Advanced Encryption Standard (AES) [Електронний ресурс] : // - Режим доступу: World Wide Web. – URL: https://www.geeksforgeeks.org/advanced-encryption-standard-aes/.
13. AES-128. Детали и реализация на python [Електронний ресурс] : // - Режим доступу: World Wide Web. – URL: https://habr.com/ru/articles/212235/.
14. Стандарт шифрования данных DES (Data Encryption Standard) Електронний ресурс] : // - Режим доступу: World Wide Web. – URL: https://nmetau.edu.ua/file/info_defend_lab_3.pdf.
15. Шифрування за допомогою DES Python [Електронний ресурс] : // - Режим доступу: World Wide Web. – URL: https://spy-soft.net/des-python/.
16. Data encryption standard (DES) [Електронний ресурс] : // - Режим доступу: World Wide Web. - URL: https://www.geeksforgeeks.org/data-encryption-standard-des-set-1/.
17. Алгоритм шифрування даних IDEA [Електронний ресурс] : // - Режим доступу: World Wide Web. – URL: http://um.co.ua/9/9-8/9-88125.html.
18. Simplified International Data Encryption Algorithm (IDEA) [Електронний ресурс] : // - Режим доступу: World Wide Web. - URL: https://www.geeksforgeeks.org/simplified-international-data-encryption-algorithm-idea/.

19. Що таке шифрування RSA і як це працює? [Електронний ресурс] : // - Режим доступу: World Wide Web. – URL: https://instagalleryapp.com/informacijna-bezpeka/shho-take-shifruvannja-rsa-i-jak-ce-pracjue/.
20. Алгоритм шифрування RSA, види атак на нього. Реалізація мовою Python [Електронний ресурс] : // - Режим доступу: World Wide Web. – URL: https://dou.ua/forums/topic/43026/.
21. RSA Algorithm in Cryptography [Електронний ресурс] : // - Режим доступу: World Wide Web. – URL: https://www.geeksforgeeks.org/rsa-algorithm-cryptography/.
22. Алгоритм Эль Гамаля [Електронний ресурс] : // - Режим доступу: World Wide Web. – URL: https://kivyfreakt.github.io/elgamal-site/.
23. ElGamal Encryption Algorithm [Електронний ресурс] : // - Режим доступу: World Wide Web. – URL: https://www.geeksforgeeks.org/elgamal-encryption-algorithm/.
24. Підручник з Python [Електронний ресурс] : // - Режим доступу: https://docs.python.org/uk/3/tutorial/index.html.
25. Посібник з мови програмування Python [Електронний ресурс] : // - Режим доступу: https://metanit.com/python/tutorial/.
26. pyTelegramBotAPI [Електронний ресурс] : // - Режим доступу: https://pypi.org/project/pyTelegramBotAPI/.
27. pyTelegramBotAPI Documentation 4.12.0 [Електронний ресурс] : // - Режим доступу: https://pytba.readthedocs.io/ru/latest/.
28. pyDes [Електронний ресурс] : // - Режим доступу: https://github.com/twhiteman/pyDes.
29. twhiteman / pyDes [Електронний ресурс] : // - Режим доступу: https://github.com/twhiteman/pyDes
30. pycryptodome 3.19.0 [Електронний ресурс] : // - Режим доступу: https://pypi.org/project/pycryptodome/.

31. Welcome to PyCryptodome’s documentation [Електронний ресурс] : // - Режим доступу: https://pycryptodome.readthedocs.io/en/latest/.
32. cryptography 41.0.5 [Електронний ресурс] : // - Режим доступу: World Wide Web. – URL: https://pypi.org/project/cryptography/.
33. Cryptography [Електронний ресурс] : // - Режим доступу: https://cryptography.io/en/latest/.
34. Шифрування за допомогою Cryptography Python [Електронний ресурс] : // - Режим доступу: https://spy-soft.net/cryptography-python/.
35. Documentation for Visual Studio Code [Електронний ресурс] : // - Режим доступу: https://code.visualstudio.com/docs.
36. Гриненко Д.С. РОЗРОБКА ПРОГРАМНОГОЗАСОБУ ДЛЯ ЗАХИСТУ ДАНИХ З РІЗНИМИ МЕТОДАМИ ШИФРУВАННЯ/ Д.С. Гриненко, А.Б. Петренко// РОЗВИТОК СУЧАСНОЇ НАУКИ: АКТУАЛЬНІ ПИТАННЯ ТЕОРІЇ ТА ПРАКТИКИ. – 2023 – с 301-303.

ДОДАТКИ

Додаток А. Код програмного засобу

```
import telebot
import time
import pyDes
from pyDes import des, PAD_PKCS5, CBC
from telebot import types
from Cryptodome.Cipher import AES
from Cryptodome.Random import get_random_bytes
import os
from cryptography.hazmat.primitives import padding
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from Cryptodome.PublicKey import RSA
from Cryptodome.Cipher import PKCS1_OAEP
from Cryptodome.Signature import pkcs1_15
from Cryptodome.Hash import SHA256
from Cryptodome.PublicKey import ElGamal
from Cryptodome.Cipher import PKCS1_OAEP
from Cryptodome import Random
import base64


bot = telebot.TeleBot("6368072373:AAFJsmPSGX1GSU45o6-
4kmP__aJlAzNAz90")

@bot.message_handler(commands=['start'])
def start(message):
    markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
    item_simetric = types.KeyboardButton("Симетричне")
```

```
    item_asimetric = types.KeyboardButton("Асиметричне")
    markup.add(item_simetric, item_asimetric)
    bot.send_message(message.chat.id, "Виберіть метод шифрування:",
reply_markup=markup)

@bot.message_handler(func=lambda message: message.text ==
"Симетричне")
def sim(message):
    markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
    item_aes = types.KeyboardButton("AES")
    item_des = types.KeyboardButton("DES")
    item_idea = types.KeyboardButton("IDEA")
    markup.add(item_aes, item_des,item_idea)
    bot.send_message(message.chat.id, "Виберіть метод шифрування:",
reply_markup=markup)

@bot.message_handler(func=lambda message: message.text ==
"Асиметричне")
def asim(message):
    markup = types.ReplyKeyboardMarkup(resize_keyboard=True)
    item_RSA = types.KeyboardButton("RSA")
    item_ECC = types.KeyboardButton("ElGamal")
    markup.add(item_RSA, item_ECC)
    bot.send_message(message.chat.id, "Виберіть метод шифрування:",
reply_markup=markup)

#AES початок
@bot.message_handler(func=lambda message: message.text == "AES")
def variationsAES(message):
    markup = telebot.types.ReplyKeyboardMarkup(row_width=2)
```

```
    item1 = telebot.types.KeyboardButton("Сгенерувати ключ AES")
    item2 = telebot.types.KeyboardButton("Зашифрувати файл AES")
    item3 = telebot.types.KeyboardButton("Розшифрувати файл AES")
    markup.add(item1, item2, item3)
    bot.send_message(message.chat.id, "Виберіть дію:",
reply_markup=markup)

# Генерация ключа и отправка его пользователю
@bot.message_handler(func=lambda message: message.text == "Сгенерувати
ключ AES")
def generate_key(message):
    key = get_random_bytes(16)
    with open("encryption_key.key", "wb") as key_file:
        key_file.write(key)
    bot.send_document(message.chat.id, open("encryption_key.key", "rb"))

# Шифрование файла
@bot.message_handler(func=lambda message: message.text == "Зашифрувати
файл AES")
def encrypt_file(message):
    bot.send_message(message.chat.id, "Будь-ласка, відправте файл ключа:")
    bot.register_next_step_handler(message, get_key_and_file)

def get_key_and_file(message):
    if message.document:
        file_info = bot.get_file(message.document.file_id)
        key_file = bot.download_file(file_info.file_path)
        with open("encryption_key_received.key", "wb") as received_key_file:
            received_key_file.write(key_file)
```

```
    bot.send_message(message.chat.id, "Будь-ласка, відправте файл для шифрування:")
    bot.register_next_step_handler(message, encrypt)

def encrypt(message):
    if message.document:
        file_info = bot.get_file(message.document.file_id)
        file_to_encrypt = bot.download_file(file_info.file_path)
        with open("encryption_key_received.key", "rb") as received_key_file:
            key = received_key_file.read()
        start_time = time.time()  # Фіксуємо час початку шифрування
        cipher = AES.new(key, AES.MODE_EAX)
        ciphertext, tag = cipher.encrypt_and_digest(file_to_encrypt)
        end_time = time.time()
        elapsed_time = end_time - start_time
        print(f"Час шифрування: {elapsed_time} секунд")
        with open("encrypted_file.txt", "wb") as encrypted_file:
            [encrypted_file.write(x) for x in (cipher.nonce, tag, ciphertext)]
        bot.send_document(message.chat.id, open("encrypted_file.txt", "rb"))

# Расшифровка файла
@bot.message_handler(func=lambda message: message.text == "Розшифрувати файл AES")
def decrypt_file(message):
    bot.send_message(message.chat.id, "Будь-ласка, відправте файл ключа:")
    bot.register_next_step_handler(message, get_key_and_encrypted_file)

def get_key_and_encrypted_file(message):
    if message.document:
        file_info = bot.get_file(message.document.file_id)
```

```
        key_file = bot.download_file(file_info.file_path)
        with open("decryption_key_received.key", "wb") as received_key_file:
            received_key_file.write(key_file)
        bot.send_message(message.chat.id, "Будь-ласка, відправте файл для
Розшифрування:")
        bot.register_next_step_handler(message, decrypt)

def decrypt(message):
    if message.document:
        file_info = bot.get_file(message.document.file_id)
        file_to_decrypt = bot.download_file(file_info.file_path)
        with open("decryption_key_received.key", "rb") as received_key_file:
            key = received_key_file.read()
        start_time = time.time()  # Фіксуємо час початку шифрування
        # Разделяем байты на nonce, tag и ciphertext
        nonce = file_to_decrypt[:16]
        tag = file_to_decrypt[16:32]
        ciphertext = file_to_decrypt[32:]

        cipher = AES.new(key, AES.MODE_EAX, nonce=nonce)
        decrypted_file = cipher.decrypt(ciphertext)
        end_time = time.time()
        elapsed_time = end_time - start_time
        print(f"Час шифрування: {elapsed_time} секунд")
        bot.send_document(message.chat.id, decrypted_file)

#AES кінець

#DES start
```

```
@bot.message_handler(func=lambda message: message.text == "DES")
def variationsDES(message):
    markup = telebot.types.ReplyKeyboardMarkup(row_width=2)
    item1 = telebot.types.KeyboardButton("Сгенерувати ключ DES")
    item2 = telebot.types.KeyboardButton("Зашифрувати файл DES")
    item3 = telebot.types.KeyboardButton("Розшифрувати файл DES")
    markup.add(item1, item2, item3)
    bot.send_message(message.chat.id, "Виберіть дію:",
reply_markup=markup)

# Генерация ключа и отправка его пользователю
@bot.message_handler(func=lambda message: message.text == "Сгенерувати
ключ DES")
def generate_des_key(message):
    key = get_random_bytes(8)  # Генерация 8-байтного ключа DES
    with open("des_encryption_key.key", "wb") as key_file:
        key_file.write(key)
    bot.send_document(message.chat.id, open("des_encryption_key.key", "rb"))



# Шифрование файла
@bot.message_handler(func=lambda message: message.text == "Зашифрувати
файл DES")
def encrypt_filedes(message):
    bot.send_message(message.chat.id, "Будь-ласка, відправте файл ключа:")
    bot.register_next_step_handler(message, get_key_and_filedes)

def get_key_and_filedes(message):
    if message.document:
        file_info = bot.get_file(message.document.file_id)
```

```
        key_file = bot.download_file(file_info.file_path)
        with open("encryption_key_received.key", "wb") as received_key_file:
            received_key_file.write(key_file)
        bot.send_message(message.chat.id, "Будь-ласка, відправте файл для
шифрування:")
        bot.register_next_step_handler(message, encryptdes)

def encryptdes(message):
    if message.document:
        file_info = bot.get_file(message.document.file_id)
        file_to_encrypt = bot.download_file(file_info.file_path)
        with open("encryption_key_received.key", "rb") as received_key_file:
            key = received_key_file.read()
        start_time = time.time()  # Фіксуємо час початку шифрування
        cipher = des(key, CBC, IV=b"\0\0\0\0\0\0\0\0", pad=None,
padmode=PAD_PKCS5)
        ciphertext = cipher.encrypt(file_to_encrypt)
        with open("encrypted_file_des.txt", "wb") as encrypted_file:
            encrypted_file.write(ciphertext)
        end_time = time.time()
        elapsed_time = end_time - start_time
        print(f"Час шифрування: {elapsed_time} секунд")
        bot.send_document(message.chat.id, open("encrypted_file_des.txt", "rb"))

# Расшифровка файла
@bot.message_handler(func=lambda message: message.text ==
"Розшифрувати файл DES")
def decrypt_filedes(message):
    bot.send_message(message.chat.id, "Будь-ласка, відправте файл ключа:")
    bot.register_next_step_handler(message, get_encrypted_filedes)
```

```
def get_encrypted_filedes(message):
    if message.document:
        file_info = bot.get_file(message.document.file_id)
        key_file = bot.download_file(file_info.file_path)
        with open("decryption_key_received.key", "wb") as received_key_file:
            received_key_file.write(key_file)
        bot.send_message(message.chat.id, "Будь-ласка, відправте файл для
розшифрування:")
        bot.register_next_step_handler(message, decryptdes)

def decryptdes(message):
    if message.document:
        file_info = bot.get_file(message.document.file_id)
        file_to_decrypt = bot.download_file(file_info.file_path)
        start_time = time.time()  # Фіксуємо час початку шифрування
        with open("des_encryption_key.key", "rb") as received_key_file:
            key = received_key_file.read()
        cipher = des(key, CBC, IV=b"\0\0\0\0\0\0\0\0", pad=None,
padmode=PAD_PKCS5)
        decrypted_data = cipher.decrypt(file_to_decrypt)

        with open("decrypted_file_des.txt", "wb") as decrypted_file:
            decrypted_file.write(decrypted_data)
        end_time = time.time()
        elapsed_time = end_time - start_time
        print(f"Час шифрування: {elapsed_time} секунд")
        bot.send_document(message.chat.id, open("decrypted_file_des.txt", "rb"))

#DES end
```

```
#IDEA start

@bot.message_handler(func=lambda message: message.text == "IDEA")
def variationsIDEA(message):
    markup = telebot.types.ReplyKeyboardMarkup(row_width=2)
    item1 = telebot.types.KeyboardButton("Сгенерувати ключ IDEA")
    item2 = telebot.types.KeyboardButton("Зашифрувати файл IDEA")
    item3 = telebot.types.KeyboardButton("Розшифрувати файл IDEA")
    markup.add(item1, item2, item3)
    bot.send_message(message.chat.id, "Виберіть дію:",
reply_markup=markup)

def generate_idea_key():
    return os.urandom(16)  # В IDEA ключ имеет длину 16 байтов

@bot.message_handler(func=lambda message: message.text == "Сгенерувати
ключ IDEA")
def generate_idea_key_message(message):
    key = generate_idea_key()
    with open("idea_encryption_key.key", "wb") as key_file:
        key_file.write(key)
    bot.send_document(message.chat.id, open("idea_encryption_key.key", "rb"))

# Шифрование файла
@bot.message_handler(func=lambda message: message.text == "Зашифрувати
файл IDEA")
def encrypt_fileidea(message):
    bot.send_message(message.chat.id, "Будь-ласка, відправте файл ключа:")
    bot.register_next_step_handler(message, get_key_and_fileidea)
```

```
def get_key_and_fileidea(message):
    if message.document:
        file_info = bot.get_file(message.document.file_id)
        key_file = bot.download_file(file_info.file_path)
        with open("idea_encryption_key.key", "wb") as received_key_file:
            received_key_file.write(key_file)
        bot.send_message(message.chat.id, "Будь-ласка, відправте файл для
шифрования:")
        bot.register_next_step_handler(message, encryptidea)

def encryptidea(message):
    if message.document:
        file_info = bot.get_file(message.document.file_id)
        file_to_encrypt = bot.download_file(file_info.file_path)
        with open("idea_encryption_key.key", "rb") as received_key_file:
            key = received_key_file.read()
        start_time = time.time()  # Фіксуємо час початку шифрування
        cipher = Cipher(algorithms.IDEA(key), modes.ECB())
        encryptor = cipher.encryptor()
        padder = padding.PKCS7(64).padder()
        padded_data = padder.update(file_to_encrypt) + padder.finalize()
        ciphertext = encryptor.update(padded_data) + encryptor.finalize()

        with open("encrypted_file_idea.txt", "wb") as encrypted_file:
            encrypted_file.write(ciphertext)
        end_time = time.time()
        elapsed_time = end_time - start_time
        print(f"Час шифрування: {elapsed_time} секунд")
        bot.send_document(message.chat.id, open("encrypted_file_idea.txt", "rb"))
```

```
# Расшифровка файла

@bot.message_handler(func=lambda message: message.text ==
"Розшифрувати файл IDEA")
def decrypt_fileidae(message):
    bot.send_message(message.chat.id, "Будь-ласка, відправте файл:")
    bot.register_next_step_handler(message, get_encrypted_fileidea)

def get_encrypted_fileidea(message):
    if message.document:
        file_info = bot.get_file(message.document.file_id)
        key_file = bot.download_file(file_info.file_path)
        with open("decryption_key_received.key", "wb") as received_key_file:
            received_key_file.write(key_file)
        bot.send_message(message.chat.id, "Будь-ласка, відправте файл для
розшифровки:")
        bot.register_next_step_handler(message, decryptidea)

def decryptidea(message):
    if message.document:
        file_info = bot.get_file(message.document.file_id)
        file_to_decrypt = bot.download_file(file_info.file_path)
        start_time = time.time()  # Фіксуємо час початку шифрування
        with open("idea_encryption_key.key", "rb") as received_key_file:
            key = received_key_file.read()
        ciphertext = file_to_decrypt
        cipher = Cipher(algorithms.IDEA(key), modes.ECB())  # Режим ECB для
прикладу, оберіть відповідний режим
        decryptor = cipher.decryptor()
```

```
        unpadder = padding.PKCS7(64).unpadder()
        decrypted_padded_data = decryptor.update(ciphertext) +
decryptor.finalize()
        plaintext = unpadder.update(decrypted_padded_data) + unpadder.finalize()
        with open("decrypted_file.txt", "wb") as decrypted_file:
            decrypted_file.write(plaintext)
        end_time = time.time()
        elapsed_time = end_time - start_time
        print(f"Час шифрування: {elapsed_time} секунд")
        bot.send_document(message.chat.id, open("decrypted_file.txt", "rb"))
#IDEA end

#rsa start
@bot.message_handler(func=lambda message: message.text == "RSA")
def variationsIDEA(message):
    markup = telebot.types.ReplyKeyboardMarkup(row_width=2)
    item1 = telebot.types.KeyboardButton("Сгенерувати ключ RSA")
    item2 = telebot.types.KeyboardButton("Зашифрувати файл RSA")
    item3 = telebot.types.KeyboardButton("Розшифрувати файл RSA")
    markup.add(item1, item2, item3)
    bot.send_message(message.chat.id, "Виберіть дію:",
reply_markup=markup)

def generate_rsa_keys(public_key_path, private_key_path, key_size=2048):
    # Генерація ключів
    private_key = rsa.generate_private_key(
        public_exponent=65537,
        key_size=key_size,
        backend=default_backend()
    )
```

```
    # Отримання відкритого ключа
    public_key = private_key.public_key()

    # Збереження відкритого ключа
    with open(public_key_path, 'wb') as public_key:
        public_key.write(
            public_key.public_bytes(
                encoding=serialization.Encoding.PEM,
                format=serialization.PublicFormat.SubjectPublicKeyInfo
            )
        )

    # Збереження закритого ключа
    with open(private_key, 'wb') as private_key:
        private_key.write(
            private_key.private_bytes(
                encoding=serialization.Encoding.PEM,
                format=serialization.PrivateFormat.PKCS8,
                encryption_algorithm=serialization.NoEncryption()
            )
        )
# Збереження ключів у файлах
def save_keys_to_files(private_key, public_key):
    with open("private_key.pem", "wb") as private_key_file:
        private_key_file.write(private_key)
    with open("public_key.pem", "wb") as public_key_file:
        public_key_file.write(public_key)

# Завантаження ключів з файлів
```

```
def load_keys_from_files():
    with open("private_key.pem", "rb") as private_key_file:
        private_key = RSA.import_key(private_key_file.read())
    with open("public_key.pem", "rb") as public_key_file:
        public_key = RSA.import_key(public_key_file.read())
    return private_key, public_key

# Шифрування файлу за допомогою публічного ключа
def encrypt_file(message, public_key):
    bot.send_message(message.chat.id, "Будь-ласка, відправте файл для шифрування:")
    bot.register_next_step_handler(message, lambda m: perform_encryption(m, public_key))

def perform_encryption(message, public_key):

    if message.document:
        file_info = bot.get_file(message.document.file_id)
        file_to_encrypt = bot.download_file(file_info.file_path)
        start_time = time.time()  # Фіксуємо час початку шифрування
        cipher_rsa = PKCS1_OAEP.new(public_key)
        encrypted_file = cipher_rsa.encrypt(file_to_encrypt)

        with open("encrypted_file.txt", "wb") as encrypted_file_file:
            encrypted_file_file.write(encrypted_file)
        end_time = time.time()
        elapsed_time = end_time - start_time
        print(f"Час шифрування: {elapsed_time} секунд")
        bot.send_document(message.chat.id, open("encrypted_file.txt", "rb"))
```

```
# Розшифрування файлу за допомогою приватного ключа
def decrypt_file(message, private_key):
    bot.send_message(message.chat.id, "Будь-ласка, відправте файл для расшифровки:")
    bot.register_next_step_handler(message, lambda m: perform_decryption(m, private_key))

def perform_decryption(message, private_key):
    if message.document:
        file_info = bot.get_file(message.document.file_id)
        file_to_decrypt = bot.download_file(file_info.file_path)
        start_time = time.time()  # Фіксуємо час початку шифрування
        cipher_rsa = PKCS1_OAEP.new(private_key)
        decrypted_file = cipher_rsa.decrypt(file_to_decrypt)
        end_time = time.time()
        elapsed_time = end_time - start_time
        print(f"Час шифрування: {elapsed_time} секунд")
        bot.send_document(message.chat.id, decrypted_file)

# Обробник команди для генерації ключів
@bot.message_handler(func=lambda message: message.text == "Сгенерувати ключ RSA")
def generate_rsa_keys_command(message):
    private_key, public_key = generate_rsa_keys()
    save_keys_to_files(private_key, public_key)
    bot.send_document(message.chat.id, open("private_key.pem", "rb"))
    bot.send_document(message.chat.id, open("public_key.pem", "rb"))

# Обробник команди для шифрування файлу
```

```
@bot.message_handler(func=lambda message: message.text == "Зашифрувати
файл RSA")
def encrypt_file_command(message):
    private_key, public_key = load_keys_from_files()
    encrypt_file(message, public_key)

# Обробник команди для розшифрування файлу
@bot.message_handler(func=lambda message: message.text ==
"Розшифрувати файл RSA")
def decrypt_file_command(message):
    private_key, public_key = load_keys_from_files()
    decrypt_file(message, private_key)

#rsa end

#ElGamal start

@bot.message_handler(func=lambda message: message.text == "ElGamal")
def variationsElGamal(message):
    markup = telebot.types.ReplyKeyboardMarkup(row_width=2)
    item1 = telebot.types.KeyboardButton("Сгенерувати ключ ElGamal")
    item2 = telebot.types.KeyboardButton("Зашифрувати файл ElGamal")
    item3 = telebot.types.KeyboardButton("Розшифрувати файл ElGamal")
    markup.add(item1, item2, item3)
    bot.send_message(message.chat.id, "Виберіть дію:",
reply_markup=markup)

def generate_elgamal_keysel():
    random_generator = Random.new().read
    key = ElGamal.generate(2048, random_generator)
```

```
    private_key = key
    public_key = key.publickey()
    return private_key, public_key

# Збереження ключів у файлах
def save_keys_to_filesel(private_key, public_key):
    with open("private_key.pem", "wb") as private_key_file:
        private_key_file.write(private_key.export_key().encode())
    with open("public_key.pem", "wb") as public_key_file:
        public_key_file.write(public_key.export_key().encode())

# Завантаження ключів з файлів
def load_keys_from_filesel():
    with open("private_key.pem", "rb") as private_key_file:
        private_key = ElGamal.import_key(private_key_file.read())
    with open("public_key.pem", "rb") as public_key_file:
        public_key = ElGamal.import_key(public_key_file.read())
    return private_key, public_key

# Шифрування файлу за допомогою публічного ключа
def encrypt_fileel(message, public_key):
    bot.send_message(message.chat.id, "Будь-ласка, відправте файл для
шифрования:")
    bot.register_next_step_handler(message, lambda m:
perform_encryptionel(m, public_key))

def perform_encryptionel(message, public_key):
    if message.document:
        file_info = bot.get_file(message.document.file_id)
        file_to_encrypt = bot.download_file(file_info.file_path)
```

```
        session_key = Random.new().read(32)
        cipher_rsa = PKCS1_OAEP.new(public_key)
        encrypted_session_key = cipher_rsa.encrypt(session_key)
        start_time = time.time()  # Фіксуємо час початку шифрування
        cipher_elgamal = ElGamal.new(public_key)
        ciphertext = cipher_elgamal.encrypt(file_to_encrypt, session_key)

        with open("encrypted_file.txt", "wb") as encrypted_file_file:
            encrypted_file_file.write(encrypted_session_key + ciphertext)
        end_time = time.time()
        elapsed_time = end_time - start_time
        print(f"Час шифрування: {elapsed_time} секунд")
        bot.send_document(message.chat.id, open("encrypted_file.txt", "rb"))

# Розшифрування файлу за допомогою приватного ключа
def decrypt_fileel(message, private_key):
    bot.send_message(message.chat.id, "Будь-ласка, відправте файл для
расшифровки:")
    bot.register_next_step_handler(message, lambda m:
perform_decryptionel(m, private_key))

def perform_decryptionel(message, private_key):
    if message.document:
        file_info = bot.get_file(message.document.file_id)
        file_to_decrypt = bot.download_file(file_info.file_path)
        start_time = time.time()  # Фіксуємо час початку шифрування
        session_key_length = 32
        encrypted_session_key = file_to_decrypt[:session_key_length]
        ciphertext = file_to_decrypt[session_key_length:]
```

```
        cipher_rsa = PKCS1_OAEP.new(private_key)
        session_key = cipher_rsa.decrypt(encrypted_session_key)

        cipher_elgamal = ElGamal.new(private_key)
        decrypted_text = cipher_elgamal.decrypt(ciphertext, session_key)
        end_time = time.time()
        elapsed_time = end_time - start_time
        print(f"Час шифрування: {elapsed_time} секунд")
        bot.send_document(message.chat.id, decrypted_text)

# Обробник команди для генерації ключів
@bot.message_handler(func=lambda message: message.text == "Сгенерувати
ключ ElGamal")
def generate_elgamal_keys_commandel(message):
    print(f"Ч1")
    private_key, public_key = generate_elgamal_keysel()
    print(f"Ч2")
    save_keys_to_files(private_key, public_key)
    print(f"Ч3")
    bot.send_document(message.chat.id, open("private_key.pem", "rb"))
    bot.send_document(message.chat.id, open("public_key.pem", "rb"))

# Обробник команди для шифрування файлу
@bot.message_handler(func=lambda message: message.text == "Зашифрувати
файл ElGamal")
def encrypt_file_commandel(message):
    private_key, public_key = load_keys_from_files()
    encrypt_fileel(message, public_key)
```

```
# Обробник команди для розшифрування файлу
@bot.message_handler(func=lambda message: message.text ==
"Розшифрувати файл ElGamal")
def decrypt_file_commandel(message):
    private_key, public_key = load_keys_from_files()
    decrypt_fileel(message, private_key)

#ElGamal end

if __name__ == "__main__":
    bot.polling()
```