Національний університет «Чернігівський колегіум» імені Т.Г.Шевченка
Природничо-математичний факультет
Кафедра інформатики і обчислювальної техніки

Кваліфікаційна робота

освітнього ступеня «бакалавр»

на тему

РОЗРОБКА СИСТЕМИ ВІЗУАЛІЗАЦІЇ АЛГОРИТМІВ ШИФРУВАННЯ

Виконав:
студент 4 курсу, 44-фмт групи
спеціальності
122 Комп’ютерні науки
Левченко Микола Олександрович

Науковий керівник:
к.п.н., доц. Вінниченко Є.Ф.

Чернігів – 2024

Роботу подано до розгляду «_____» _____ 202__ року.

Студент (ка) _____ Левченко М. О.

(підпис) (прізвище та ініціали)

Науковий керівник _____ Вінниченко Є.Ф.

(підпис) (прізвище та ініціали)

Рецензент _____ _____

(підпис) (прізвище та ініціали)

Кваліфікаційна робота розглянута на засіданні кафедри
Інформатики і обчислювальної техніки
протокол № _____ від «_____» _____ 202__ року.
Студент (ка) допускається до захисту даної роботи в екзаменаційній комісії.

Завідувач кафедри _____ Горошко Ю. В.

(підпис) (прізвище та ініціали)

ЗМІСТ

ВСТУП

У сучасному світі, де інформаційні технології стали невід'ємною частиною повсякденного життя, питання захисту інформації набувають все більшої актуальності. З розвитком глобальної мережі Інтернет, розширенням обсягів оброблюваних даних та зростанням кіберзагроз, забезпечення безпеки даних стає одним із ключових завдань для будь-якої організації. Криптографія, як наука про методи шифрування інформації, відіграє центральну роль у захисті даних від несанкціонованого доступу, зміни та підробки.

Актуальність теми візуалізації алгоритмів шифрування важко переоцінити. З одного боку, криптографія є складною технічною дисципліною, яка вимагає глибоких математичних знань та розуміння принципів роботи алгоритмів. З іншого боку, велика кількість фахівців у різних сферах, які стикаються з необхідністю використання криптографічних методів, не завжди мають можливість чи бажання глибоко вникати в технічні деталі. Візуалізація алгоритмів шифрування дозволяє подолати цей бар'єр, надаючи наочні інструменти для розуміння і використання криптографії.

Важливість візуалізації алгоритмів шифрування полягає в її здатності спрощувати складні концепції і робити їх доступними для людей, які бажають вивчати ці алгоритми. Візуалізація допомагає зрозуміти, як працюють алгоритми, показуючи процеси шифрування та дешифрування у вигляді зрозумілих графічних схем. Це підвищує рівень обізнаності та освіти у сфері криптографії, і сприяє підвищенню безпеки даних.

Завдання цієї дипломної роботи полягають у розробці програмного забезпечення для візуалізації різних алгоритмів шифрування, яке буде корисним для освітніх цілей та для професійного використання. Основною метою є створення інструменту, який дозволить наочно демонструвати принципи роботи таких алгоритмів, як AES, алгоритм Плейфера та інших.

Розробка такого програмного забезпечення передбачає вирішення ряду

ключових завдань. По-перше, необхідно здійснити аналіз існуючих методів і інструментів візуалізації криптографічних алгоритмів, визначити їхні переваги та недоліки, і на основі цього створити власний підхід, який буде враховувати виявлені проблеми. По-друге, необхідно розробити інтуїтивно зрозумілий графічний інтерфейс, який дозволить користувачам взаємодіяти з алгоритмами в реальному часі, спостерігаючи за кожним етапом їх роботи. По-третє, важливо забезпечити можливість гнучкого налаштування параметрів алгоритмів, щоб користувачі могли експериментувати з різними вхідними даними та налаштуваннями.

Також варто розглянути перспективи подальшого розвитку і вдосконалення програмного забезпечення для візуалізації алгоритмів шифрування. Це може включати додавання нових алгоритмів, розширення функціональності, інтеграцію з іншими інструментами та платформами, а також покращення користувацького досвіду за допомогою сучасних технологій, таких як віртуальна та доповнена реальність.

Таким чином, ця дипломна робота спрямована на вирішення проблем у сфері навчання криптографії та кібербезпеки шляхом розробки ефективного інструменту для візуалізації алгоритмів шифрування. Її результати можуть бути корисними в першу чергу для студентів, що вивчають криптографію. Візуалізація алгоритмів шифрування сприятиме підвищенню рівня обізнаності, розуміння та ефективного використання криптографічних методів у різних сферах діяльності.

РОЗДІЛ 1
Загальний огляд шифрування

1.1. Визначення поняття «Шифрування».

Шифрування - це процес перетворення звичайного тексту, який називається "відкритим текстом", у незрозумілий для читання шифрований текст за допомогою спеціального алгоритму, відомого як "шифр". Звідси випливає, що шифр - це алгоритм або метод, який використовується для шифрування або розшифрування даних. Шифр складається з набору правил або інструкцій, за допомогою яких відкритий текст перетворюється у шифротекст, або навпаки. Він може включати в себе різноманітні математичні операції, підстановки, перестановки або змішування символів для забезпечення безпеки та надійності процесу шифрування. Шифрування використовується для забезпечення конфіденційності даних, щоб зловмисники не могли прочитати чи зрозуміти інформацію без необхідного ключа. Навіть якщо зловмисники якимось чином отримають фізичний або електронний доступ до них. Крім того, шифрування використовується для забезпечення цілісності даних та автентифікації користувачів і систем. За допомогою цифрових підписів і хеш-функцій можна перевірити, чи були дані змінені або підроблені, а також перевірити ідентичність користувача чи системи.

Шифрування є важливим елементом в будь-якій сфері, де потрібно зберігати або передавати конфіденційну інформацію. Від банківських транзакцій і медичних записів до корпоративних секретів і особистих повідомлень. У світі, де цифрова інформація стає все більш цінним ресурсом, важливість шифрування лише зростає. З кожним роком з'являються нові загрози та виклики у сфері кібербезпеки, і шифрування є одним з основних засобів захисту від них.

1.2. Види шифрування

Шифрування можна класифікувати за різними критеріями, включаючи тип ключа, кількість ключів та напрямок шифрування. Основні типи алгоритмів шифрування включають симетричне, асиметричне та гібридне шифрування. Давайте розглянемо кожен з цих типів та їх характеристики:

Симетричне шифрування:

- У симетричному шифруванні один і той же ключ використовується як для шифрування, так і для розшифрування даних.
- Цей тип шифрування є швидшим і менш вимогливим до ресурсів, ніж асиметричне шифрування, оскільки він вимагає менше обчислювальних ресурсів.
- Недоліком симетричного шифрування є проблема обміну ключами - обидва боки комунікації повинні мати доступ до спільного ключа перед передачею даних.

Асиметричне шифрування

- У асиметричному шифруванні використовується пара ключів: публічний ключ для шифрування та приватний ключ для розшифрування.
- Цей тип шифрування дозволяє безпечно обмінюватися ключами через ненадійні канали зв'язку, оскільки приватний ключ залишається виключно в руках власника.
- Асиметричне шифрування зазвичай повільніше за симетричне, оскільки використовується більше обчислювальних ресурсів.

Гібридне шифрування

- Гібридне шифрування поєднує переваги симетричного та асиметричного шифрування.

- У цьому підході спочатку використовується асиметричне шифрування для безпечного обміну симетричним ключем, який потім використовується для шифрування та розшифрування фактичних даних.
- Гібридне шифрування дозволяє поєднувати високу безпеку асиметричного шифрування з швидкістю симетричного шифрування.

Основні відмінності між симетричним та асиметричним шифруванням полягають у використанні ключів та процесі шифрування. Симетричне шифрування використовує один ключ для обох процесів, тоді як асиметричне використовує пару ключів для шифрування та розшифрування. Гібридне шифрування поєднує ці два підходи для забезпечення балансу між безпекою та ефективністю.

1.3. Основні принципи шифрування

Шифрування - це складний процес, який базується на декількох ключових принципах для забезпечення ефективного захисту даних. Ось докладний огляд цих принципів:

Конфіденційність. Мабуть, найголовніший принцип шифрування - це забезпечення конфіденційності даних. Це означає, що лише відповідна сторона, яка має правильний ключ чи пароль, може розшифрувати дані і прочитати їх. Конфіденційність даних важлива для запобігання несанкціонованому доступу та збереження приватності користувачів.

Цілісність. Цілісність даних - це забезпечення того, що дані не були змінені чи пошкоджені під час передачі чи зберігання. Шифрування може допомогти впевнитися, що дані залишаються цілими, оскільки будь-яка зміна в шифрованому тексті призведе до некоректного дешифрування.

Аутентифікація. Шифрування також може використовуватися для аутентифікації користувачів чи систем. Це означає, що шифрований обмін даними може включати перевірку того, що обидві сторони є тими, за кого вони

себе видають. Це може бути досягнуто за допомогою цифрових підписів та сертифікатів.

Відмовостійкість. Шифрування має бути відмовостійким, що означає, що навіть при наявності певного рівня атаки чи спроб злому, воно все ще має залишатись безпечним. Це досягається за допомогою використання потужних шифрів та ключів великої довжини.

Загальний принцип шифрування полягає в тому, щоб забезпечити захист даних, зберігаючи при цьому їх доступними для легітимних користувачів.

РОЗДІЛ 2
Огляд існуючих алгоритмів шифрування

2.1. Шифр Цезаря

Шифр Цезаря отримав свою назву на честь римського імператора Гая Юлія Цезаря, який, за легендою, використовував цей метод для шифрування своїх особистих листів. Хоча точний час його виникнення невідомий, але вважається, що шифр був використаний в Римі близько 100 року до нашої ери.

Використання шифру Цезаря дозволяло імператорові Цезарю та його спільникам передавати конфіденційну інформацію без ризику її розголошення під час транспортування чи передачі. Проте, з огляду на його простоту, він не забезпечував надійного захисту від пристрасних криптоаналітиків.

Шифр Цезаря став одним із найдавніших відомих методів шифрування та слугував основою для багатьох подальших розвитків криптографії. Хоча він уже не використовується для захисту конфіденційної інформації через свою простоту та низький рівень безпеки, він залишається важливим етапом в історії розвитку криптографії та захисту даних.

Принцип роботи

Шифр Цезаря базується на простому принципі зсуву букв у алфавіті. Головною ідеєю було замінити кожну букву у повідомленні на букву, розташовану на певну кількість позицій вправо або вліво від неї в алфавіті. Цей параметр зсуву, відомий як ключ шифру, використовується для шифрування та розшифрування повідомлень. Наприклад, якщо взяти за основу англійський алфавіт та використовувати зсув на 3 позиції вправо, то буква "A" стане "D", буква "B" - "E", і так далі (рис.2.1).

Переваги

1. **Простота:** Шифр Цезаря є дуже простим у реалізації і розумінні.

2. **Швидкість:** Шифрування та дешифрування може бути виконане дуже швидко, навіть вручну.

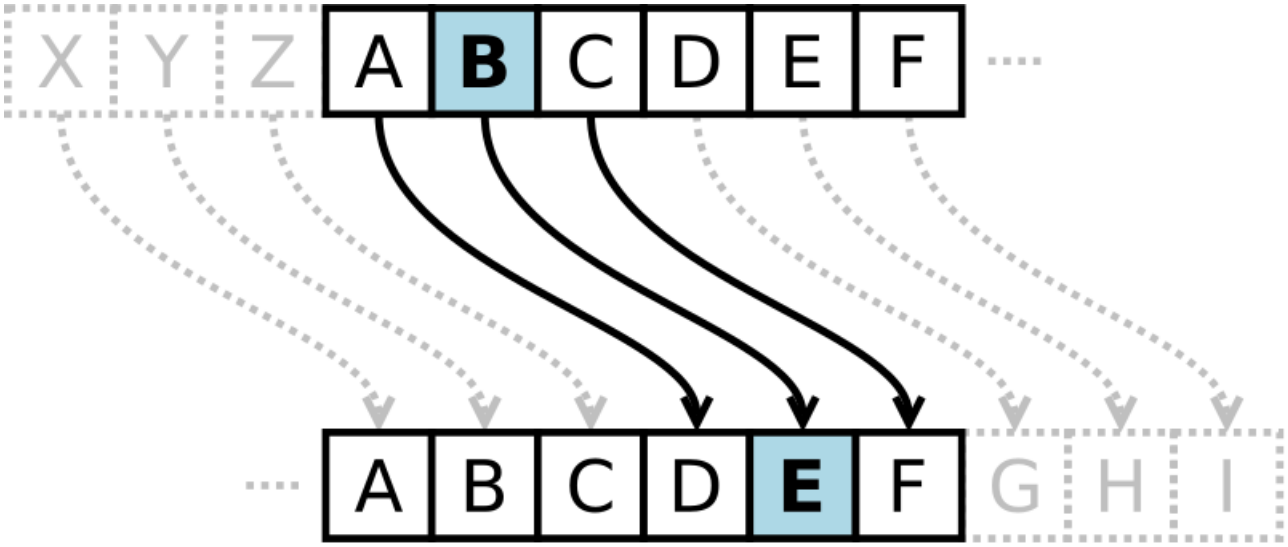


Рис. 2.1

3. **Відносна безпека:** Для зламу шифру Цезаря потрібно випробувати всього лише 25 можливих ключів (якщо вважати алфавіт англійською мовою), що робить його відносно безпечним у застосуванні до приховання простих повідомлень.

Недоліки:

1. **Низький рівень безпеки:** Шифр Цезаря легко піддається злому.
2. **Обмежена кількість ключів:** У шифрі Цезаря кількість можливих ключів обмежена кількістю літер в алфавіті.

Шифр Цезаря використовувався в історичних контекстах, таких як військові комунікації або для простого шифрування повідомлень. Зараз він не актуальний з точки зору захисту даних та найчастіше використовується як освітній приклад для демонстрації принципів шифрування.

2.2. Шифрування перестановкою

Шифрування перестановкою — це як і попередній розглянутий шифр, ще один з найпростіших методів шифрування, який використовує перестановку

символів або байтів у вхідному тексті для створення зашифрованого тексту. Хоча точний час винаходу цього виду шифрування невідомий, але він використовувався ще в давнину.

Перестановка символів була використана в багатьох культурах і цивілізаціях як спосіб захисту конфіденційної інформації. Наприклад, шифр Цезаря, де кожна буква або символ замінювався іншим, що знаходився на певній відстані в алфавіт, може бути розглянутий як специфічний різновид шифрування перестановкою.

Протягом історії шифрування перестановкою використовувалися різні методи та алгоритми, включаючи прості техніки перемішування символів або стовпчиків у матриці тексту. Однак з появою більш складних методів шифрування, таких асиметричне шифрування та криптографічні хеш-функції, методи перестановки стали менш популярними через свою відносну простоту та низький рівень безпеки. Але навіть на сьогоднішній день перестановка символів залишається важливим елементом у складних криптографічних системах, таких як шифри з використанням побітових трансформацій та стеганографії.

Принцип роботи

Принцип роботи цього виду шифрування досить простий та полягає в тому, що порядок символів або байтів змінюється відповідно до певного ключа або правила, яке відоме тільки тим, хто зашифровує повідомлення та повинен отримати та розшифрувати це саме повідомлення.

Переваги шифрування перестановкою включають його простоту в реалізації та високу швидкість шифрування. Цей метод може бути легко реалізований за допомогою простих алгоритмів, що дозволяє використовувати його на різних платформах та в різних програмах. Крім того, шифрування перестановкою може бути ефективним для шифрування коротких повідомлень або символів, особливо, якщо використовується додаткове змішування або обробка.

Однак у шифрування перестановкою є суттєві недоліки. Один з найбільш очевидних - це низький рівень безпеки. Більшість простих методів перестановки можуть бути легко розкриті з використанням статистичного аналізу або методів криптоаналізу. Крім того, використання тільки шифрування перестановкою скоріше за все буде недостатньо для надійного захисту конфіденційної інформації, особливо в контексті сучасних загроз кібербезпеки.

Загалом, шифрування перестановкою залишається корисним методом для деяких простих застосувань, але варто враховувати його обмеження щодо безпеки та надійності. В більш складних системах шифрування перестановкою може використовуватися як один з компонентів більш складної схеми шифрування для забезпечення додаткового рівня безпеки.

2.3. Шифр Плейфера

Шифр Плейфера (Playfair cipher) - це один з класичних методів шифрування, який був винайдений англійським ювеліром і аматорським криптографом Чарльзом Уітстоном у 1854 році, але був вперше опублікований в 1869 році Еджаром Алленом Плейфером. Цей шифр отримав свою назву на честь Плейфера, хоч він і не був його автором.

Він був використаний в армії Великобританії під час Першої світової війни, а також застосовувався в дипломатичних спільнотах. Він був важливим інструментом для захисту конфіденційної інформації до того часу, коли були розроблені більш складні методи шифрування, які були більш стійкими до криптоаналізу.

Принцип роботи

Шифр Плейфера використовує матрицю 5x5, заповнену алфавітом без повторень, для шифрування пар букв. Спочатку утворюється ключове слово, яке використовується для заповнення матриці. Потім пари літер з відкритого

тексту замінюються на літери згідно з певними правилами, які визначаються позиціями літер у матриці (рис.2.2).

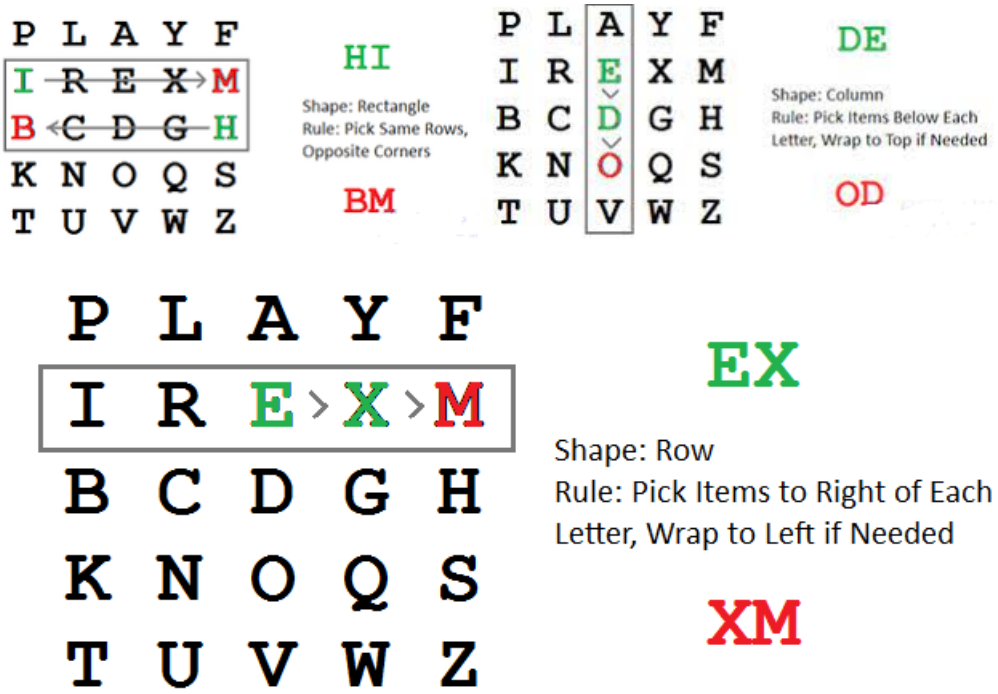


Рис. 2.2

1. Спочатку формується ключове слово, яке використовується для створення квадратної матриці, що складається з 5x5 символів (зазвичай літер алфавіту). Буква "I" (або "J") об'єднується в один символ.
2. Вхідний текст розбивається на пари літер. Якщо кількість літер у тексті непарна, то додається фіктивна літера, наприклад, "X".
3. Кожна пара літер замінюється на зашифровану пару згідно з певними правилами:
    - Якщо літери належать різним рядкам і різним стовпцям, то вони замінюються літерами, які знаходяться на перетині їх рядків у відповідних стовпцях (або навпаки).
    - Якщо літери належать одному рядку, вони замінюються літерами, що знаходяться ліворуч (або праворуч) від них, з огляду на кінець рядка (або на початок).

- Якщо літери належать одному стовпцю, вони замінюються літерами, які знаходяться вище (або нижче) від них, з огляду на кінець стовпця (або на початок).

Переваги шифру Плейфера включають його простоту в реалізації та відносно високу швидкість шифрування. Це дає можливість використовувати його для шифрування коротких повідомлень в умовах, де потрібно швидко захистити конфіденційну інформацію.

Недоліки шифру Плейфера включають обмежену стійкість до криптоаналізу. Оскільки це відомий метод перестановки, його можна легко розкрити методами частотного аналізу та атак "на підставі відкритого тексту". Крім того, він не підтримує шифрування цифр і спеціальних символів, що робить його менш універсальним.

2.4. AES (Advanced Encryption Standard)

Шифр AES (Advanced Encryption Standard) був розроблений у відповідь на необхідність створення більш надійного та ефективного стандарту шифрування для заміни старого стандарту DES (Data Encryption Standard). Офіційний процес створення AES розпочався в 1997 році, коли Національний Інститут Стандартів і Технологій (NIST) Сполучених Штатів Америки оголосив конкурс на новий стандарт шифрування. У конкурсі взяли участь багато кандидатів, проте в 2001 році NIST обрав шифр Rijndael як переможця і новий стандарт AES. Rijndael був розроблений бельгійськими криптографами Вінсентом Ріменом і Йоаном Даеменом. Їхній шифр обрали серед інших учасників переможцем через його високу стійкість, швидкість та ефективність, які були підтверджені та продемонстровані під час конкурсу NIST.

Після обрання Rijndael в якості стандарту AES, цей шифр став важливим елементом в безпеці інформації як у галузі цивільного, так і військового застосування. Його використовують у багатьох сферах, включаючи захищену

передачу даних через Інтернет, шифрування файлів на комп'ютерах, захист мережевої комунікації та багато іншого.

AES став сучасним стандартом для багатьох країн і організацій, враховуючи його високий рівень безпеки та ефективність. Він отримав широке розповсюдження в світі криптографії і захисту інформації і залишається важливим інструментом для забезпечення конфіденційності та цілісності даних у сучасному цифровому світі.

Принцип роботи

1. **Перетворення даних.** Перш за все потрібно перетворити вхідні дані у байтовий формат. Під вхідними даними розуміється інформація, яку потрібно зашифрувати та ключ, за допомогою якого буде здійснюватись цей процес. Потім потрібно розділити всі байти на блоки, розмір блоку залежить від різновиду шифрування (128 біт, 256 біт і т.д.).
2. **Додавання ключа.** До кожного байту з сформованих під час попереднього етапу блоків потрібно додати ключ шифрування.
3. **Перестановка (SubBytes):** Після додавання ключа застосовується процедура підстановки, де кожен байт замінюється на відповідний байт з S-блоку (substitution box).
4. **Зсув рядків (ShiftRows):** Далі застосовується операція зсуву рядків, де кожен рядок блоку даних зсувається вліво на певну кількість позицій. У випадку AES, перший рядок не змінюється, другий зсувається на одну позицію вліво, третій - на дві позиції, а четвертий - на три позиції і т.д.
5. **Змішування стовпців (MixColumns):** Після зсуву рядків кожний стовпець блоку даних перетворюється за допомогою лінійного перетворення. Ця операція змішує байти кожного стовпця таким чином, що вони взаємодіють один з одним.
6. **Додавання ключа раунду (Round Key Addition):** Після змішування стовпців до результату додається ключ раунду (відомий як Round Key).

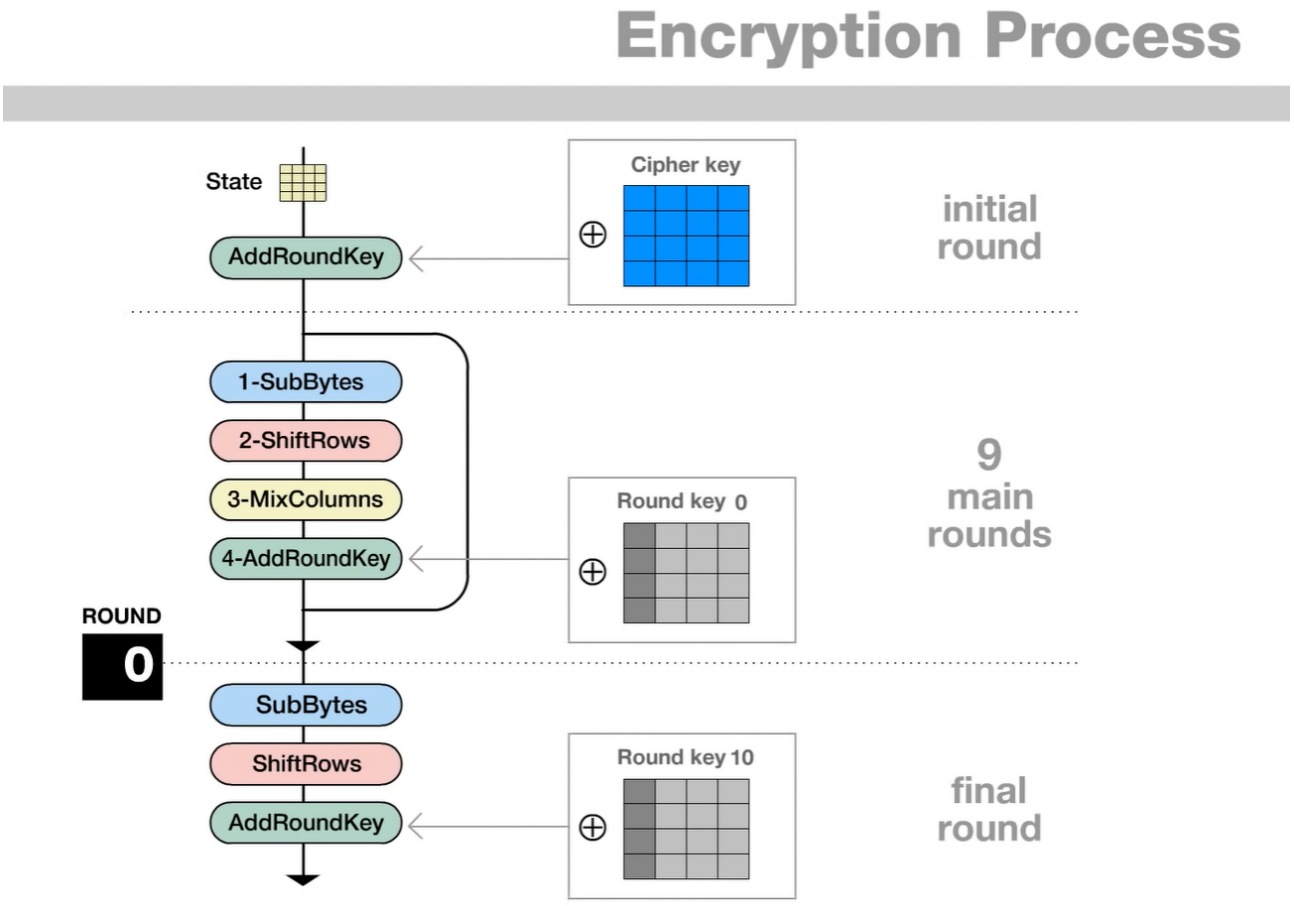


Рис. 2.3

Ці кроки (SubBytes, ShiftRows, MixColumns, Round Key Addition) повторюються для кожного раунду шифрування (у випадку AES, для 10 раундів у режимі AES-128). Після останнього раунду застосовуються всі кроки крім MixColumns, а потім до отриманого результату додається фінальний Round Key.

Переваги

Однією з головних переваг AES є його висока безпека. На даний момент немає відомих ефективних атак, які могли б зламати AES при використанні повного набору раундів і відповідної довжини ключа. Цей алгоритм успішно витримав десятиліття криптоаналітичного аналізу, що свідчить про його надійність.

AES також характеризується високою продуктивністю. Алгоритм оптимізований для ефективної реалізації як в програмному, так і в апаратному

забезпеченні. Це робить його придатним для використання в різних контекстах, від вбудованих систем до високопродуктивних серверів.

Ще однією перевагою AES є його гнучкість. Підтримка різної довжини ключів дозволяє обирати оптимальний баланс між безпекою і продуктивністю залежно від конкретних потреб. Крім того, його структурована та проста реалізація полегшує впровадження та аудити безпеки.

Недоліки

Попри свої численні переваги, AES має і певні недоліки. Один із них пов'язаний з використанням фіксованого розміру блоку даних. Для шифрування даних, що не є кратними 128 бітам, необхідно застосовувати режими роботи блочного шифрування, такі як CBC або GCM, що може ускладнювати реалізацію та призводити до потенційних вразливостей, якщо режими роботи застосовуються неправильно. Режим роботи блочного шифрування CBC (Cipher Block Chaining) поєднує кожен блок відкритого тексту з попереднім блоком шифрованого тексту за допомогою операції XOR перед шифруванням. Це забезпечує залежність кожного блоку шифрованого тексту від усіх попередніх блоків, що підвищує безпеку. Режим роботи блочного шифрування GCM (Galois/Counter Mode) забезпечує одночасно конфіденційність та цілісність даних. GCM використовує режим лічильника для шифрування блоків даних, де кожен блок поєднується з лічильником, що збільшується для кожного блоку. Окрім цього, GCM застосовує аутентифікацію Galois для перевірки цілісності даних.

Іншим недоліком є те, що AES є симетричним шифром, тобто для шифрування та дешифрування використовується один і той самий ключ. Це вимагає безпечного обміну ключами між сторонами, що може бути складним завданням у деяких сценаріях. Для вирішення цієї проблеми часто використовуються асиметричні шифри для обміну ключами, але це додає додаткову складність та накладні витрати.

Також, хоча AES є надзвичайно стійким проти сучасних атак, майбутні розробки в області квантових обчислень можуть становити загрозу для його безпеки. Квантові комп'ютери можуть ефективно використовувати алгоритм Гровера для прискорення атак грубої сили, що знижує ефективну довжину ключа вдвічі. Це означає, що для забезпечення аналогічного рівня безпеки в умовах квантових обчислень може знадобитися використання значно довших ключів.

AES є одним із найважливіших і найширше використовуваних алгоритмів шифрування у світі, що забезпечує високу безпеку та продуктивність. Його переваги включають надійність, ефективність та гнучкість, що робить його придатним для широкого спектру застосувань. Однак, як і будь-який криптографічний алгоритм, AES має свої недоліки, включаючи складності з безпечним обміном ключами та потенційні загрози з боку квантових обчислень. Незважаючи на ці виклики, AES залишається фундаментальним елементом у сучасних системах захисту даних, забезпечуючи конфіденційність та цілісність інформації у цифровому світі.

2.5. RSA (Rivest–Shamir–Adleman)

Шифр RSA (Rivest–Shamir–Adleman) був винайдений в 1977 році трьома вченими: Рональдом Рівестом, Аді Шамиром і Леонардом Адлеманом. Назва шифру складається з перших літер прізвищ цих вчених. RSA вважається першим практично застосованим алгоритмом асиметричного шифрування в історії. Сама ідея асиметричного шифрування була відкрита раніше в ідеях вчених, таких як Уітфілд Діффі та Мартін Хеллман, які розробили концепцію обміну ключами. Однак, алгоритм RSA став першою практичною реалізацією асиметричного шифрування, яка була придатна та ефективна для реального використання.

Шифр RSA став одним із найпопулярніших і широко використовуваних алгоритмів у сфері криптографії. Він застосовується для захисту конфіденційної інформації в Інтернеті, включаючи шифрування електронної пошти, забезпечення безпеки онлайн транзакцій, підпису документів та багато іншого.

Принцип роботи

Принцип роботи RSA полягає у використанні пари ключів: публічного і приватного. Публічний ключ використовується для шифрування повідомлень, тоді як приватний ключ використовується для розшифрування. Публічний ключ може бути розповсюджений вільно, тоді як приватний ключ залишається виключно у власника. RSA базується на математичних властивостях складних чисел.

1. **Генерація ключів:** Спочатку генерується пара ключів: публічний і приватний. Публічний ключ використовується для шифрування даних, тоді як приватний ключ використовується для розшифрування. Для створення ключів спочатку вибирають два великих простих числа p і q. Потім потрібно обчислити їх добуток: $n=p\times q$. Число n називається модулем і є частиною як відкритого, так і приватного ключів. Наступним кроком є обчислення функції Ейлера: $\phi(n)=(p-1)\times(q-1)$. Після цього треба вибрати відкриту експоненту e, яка є взаємно простим з $\phi(n)$. В кінці необхідно обчислити приватну експоненту d, яка задовольняє умову $e\times d\equiv 1$. Відкритий ключ складається з пари чисел (e,n), а приватний ключ — з пари (d,n).

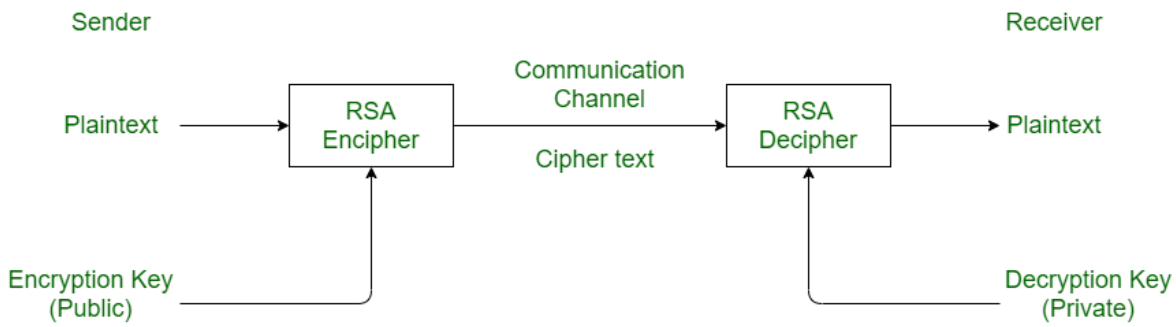


Рис. 2.4

2. **Шифрування:** Відправник, який хоче зашифрувати повідомлення M (де M — числове представлення повідомлення), використовує відкритий ключ одержувача (e,n):

   $C = M^e \mod n$

   де C — це зашифроване повідомлення (цифровий шифротекст).

3. **Розшифрування:** Одержувач використовує свій приватний ключ (d,n) для розшифрування шифротексту C:

   $M = C^d \mod n$

   Таким чином, відновлюється початкове повідомлення M.

Переваги

Однією з найбільших переваг RSA є його безпека. За поточних знань, найефективніші відомі атаки на RSA базуються на факторизації великих простих чисел, що вимагає значних обчислювальних ресурсів та тривалого часу. Правильно вибрані великі прості числа забезпечують велику стійкість проти таких атак.

Ще однією перевагою є його асиметрична природа. Відсутність необхідності обміну секретним ключем дозволяє безпечно використовувати RSA для шифрування та підписування даних через відкриті мережі, такі як Інтернет.

RSA також має високу універсальність. Він широко підтримується у програмному та апаратному забезпеченні, що дозволяє використовувати його в

різних системах та платформах.

Недоліки

Однак, RSA також має свої недоліки. Один із них полягає у його обчислювальній складності. Генерація та перевірка RSA-підписів може бути вимогливою з точки зору обчислювальних ресурсів, особливо для великих повідомлень та ключів.

Ще одним недоліком є розмір ключа. Для забезпечення високого рівня безпеки RSA вимагає великих простих чисел, що призводить до довгих ключів. Це може бути недоцільним для деяких застосувань, особливо у вимогливих до ресурсів середовищах.

Крім того, RSA вразливий до атаки на падіння публічного експонента (public exponent) і атак типу канал-час (timing attack), що можуть бути використані для отримання інформації про приватний ключ та підписи. Атака на падіння публічного експонента полягає у виборі малого значення публічного експонента, наприклад, e = 3. Це може призвести до ситуації, коли за певних умов атакуючий зможе легко розшифрувати зашифрований текст або підробити цифровий підпис. Цей вид атаки можливий, коли зашифрований текст або підпис занадто малий або має певні структурні особливості, які полегшують криптоаналіз. Атака типу канал-час (Timing Attack) експлуатує різницю у часі, який система витрачає на виконання операцій з використанням приватного ключа. Нападник, вимірюючи час обчислення певних криптографічних операцій, може поступово відновити інформацію про приватний ключ. Це можливо через те, що різні бітові значення ключа можуть призводити до різної тривалості обчислень, які можна точно виміряти і проаналізувати.

RSA є важливим асиметричним алгоритмом шифрування та підпису, який забезпечує високий рівень безпеки та універсальність. Відповідно до правильної реалізації та використання, він може забезпечити надійний захист даних та цифрових підписів у різних сценаріях.

2.6. Еліптична криптографія (ECC)

Еліптична криптографія (ECC) є відносно молодою областю в криптографії, що набрала популярність у кінці 20-го століття. Перші теоретичні основи ECC були закладені у 1985 році незалежно Нілом Кобліцем і Віктором Міллером. Вони запропонували використання еліптичних кривих у криптографії як альтернативу вже існуючим методам, таким як RSA та DSA. Ідея полягала в тому, щоб використовувати властивості еліптичних кривих для створення більш ефективних і безпечних криптографічних алгоритмів.

З часом ECC почала набирати популярність завдяки своїй ефективності та високому рівню безпеки при відносно малих розмірах ключів. У 2005 році Національний інститут стандартів і технологій США (NIST) прийняв ECC як частину стандартів для федеральних урядових застосувань. Сьогодні ECC широко використовується в різних додатках, включаючи мобільні пристрої, бездротові мережі та інтернет-протоколи.

Принцип роботи

Еліптична криптографія базується на алгебраїчних структурах еліптичних кривих над скінченними полями. Еліптична крива визначається рівнянням вигляду:

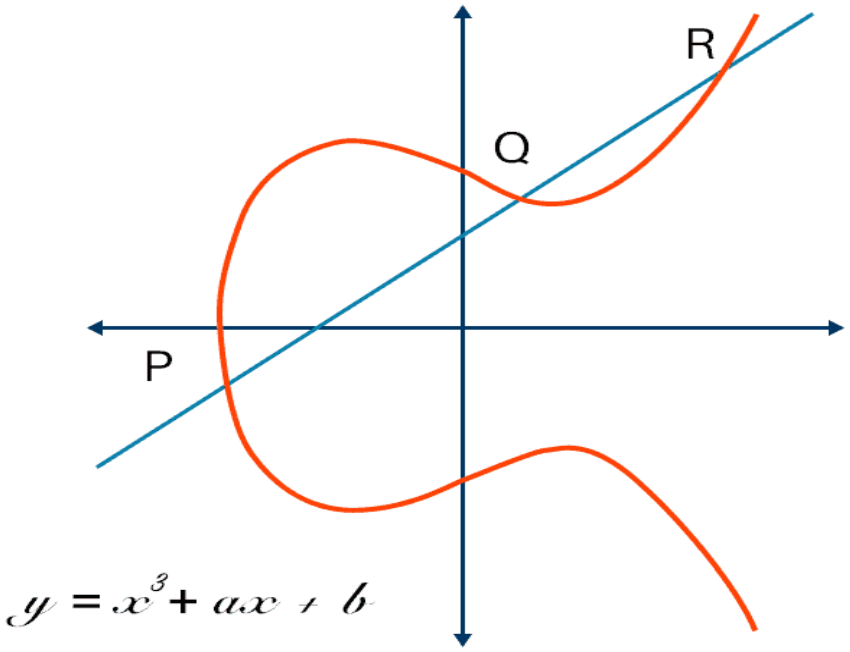
$□^2=□^3+□□+□$

де □ і □ - константи, що визначають форму кривої, і повинні задовольняти певні умови, щоб уникнути виродження.

Основна ідея ECC полягає у використанні точок на еліптичній кривій і операцій над цими точками для створення криптографічних схем. Однією з основних операцій є додавання точок, яке визначає групову структуру. Задача дискретного логарифма для еліптичних кривих (ECDLP) є базовою криптографічною проблемою, на якій базується безпека ECC. ECDLP формулюється так: знайти число □, таке що □=□□, де □ і □ - відомі точки на еліптичній кривій, і □ - невідомий скаляр (рис.2.5).

На основі цих принципів можна створювати різні криптографічні алгоритми, включаючи:

- **ECDH (Elliptic Curve Diffie-Hellman)** - протокол обміну ключами, що дозволяє двом сторонам встановити спільний секретний ключ.
- **ECDSA (Elliptic Curve Digital Signature Algorithm)** - алгоритм



цифрового підпису, що забезпечує автентифікацію і цілісність даних.

Рис. 2.5

Переваги

Однією з головних переваг ECC є висока безпека при менших розмірах ключів порівняно з традиційними методами, такими як RSA. Наприклад, ключ ECC розміром 256 біт забезпечує рівень безпеки, еквівалентний ключу RSA розміром 3072 біт. Це робить ECC особливо привабливим для середовищ з обмеженими ресурсами, таких як мобільні пристрої та інтернет речей (IoT).

Друга важлива перевага - це швидкість. Операції на еліптичних кривих, як правило, виконуються швидше, ніж аналогічні операції у традиційних криптосистемах з великими ключами. Це призводить до зменшення часу шифрування та розшифрування, а також підписання та перевірки цифрових

підписів.

Третя перевага - менше використання пам'яті та пропускної здатності. Менші розміри ключів означають менше використання пам'яті для зберігання ключів та зменшення обсягу даних, що передаються під час криптографічних операцій.

Недоліки

Незважаючи на численні переваги, ECC має також певні недоліки. По-перше, складність реалізації. Реалізація алгоритмів на основі еліптичних кривих вимагає глибоких математичних знань і є більш складною, ніж реалізація традиційних алгоритмів, таких як RSA. Це може призвести до помилок у реалізації, які можуть знизити рівень безпеки.

По-друге, потенційні вразливості до квантових атак. Хоча ECC вважається дуже безпечною проти сучасних обчислювальних методів, поява квантових комп'ютерів може зробити ECDLP вразливим. Алгоритми Шора, що працюють на квантових комп'ютерах, можуть ефективно вирішувати задачі, які вважаються складними для класичних комп'ютерів, включаючи ECDLP.

Третій недолік - патентні обмеження. Деякі аспекти ECC підпадають під патенти, що може ускладнити використання цих технологій у комерційних продуктах без отримання відповідних ліцензій.

Еліптична криптографія є потужним інструментом у сучасній криптографії, що забезпечує високу безпеку при менших розмірах ключів та швидші обчислення. Її використання особливо важливе в умовах обмежених ресурсів, таких як мобільні пристрої та IoT. Однак складність реалізації та потенційні загрози з боку квантових обчислень є серйозними викликами, які потребують подальшого дослідження та розвитку. Незважаючи на ці недоліки, ECC залишається важливою складовою сучасних криптографічних систем, що сприяє забезпеченню конфіденційності та цілісності даних у цифровому світі.

2.7. Secure Hash Algorithm (SHA)

Потрібно одразу зауважити, що SHA (Secure Hash Algorithm) є хеш-функцією, тобто вона призначена для обчислення короткого фіксованого хеш-значення для будь-якого введеного повідомлення. Тобто, SHA не використовується для шифрування даних, а лише для створення унікального "відбитка" (хешу) введеного повідомлення.

SHA-256 (Secure Hash Algorithm 256-bit) є одним з найпоширеніших алгоритмів криптографічного хешування. Він є частиною серії хеш-функцій SHA, які були розроблені американським Національним інститутом стандартів і технологій (NIST). Історія створення SHA-256 пов'язана з попередниками його серії та потребою в більш потужному і надійному алгоритмі хешування.

Перед SHA-256 були розроблені інші версії алгоритму SHA, такі як SHA-0 та SHA-1. Проте виявилося, що вони мали певні недоліки щодо безпеки, зокрема можливість колізій (ситуація, коли два різних вхідні повідомлення дають однаковий хеш-значення). Тому була розпочата робота над створенням більш надійної версії.

SHA-256 був вперше описаний в 2001 році в документі NIST FIPS PUB 180-2, який був розроблений за участі групи криптографів та фахівців з кількох країн. Основними архітекторами SHA-256 були Дейвід Вагнер, Шон Коулі та Ністер Даї. Вони розробили SHA-256 як частину серії алгоритмів хешування, яка включала також SHA-224, SHA-384 і SHA-512.

SHA-256 отримав широке використання в різних сферах, де потрібно забезпечити цілісність даних та їх безпеку. Він використовується у криптовалютах, таких як Bitcoin і Ethereum, для генерації хеш-блоків та транзакцій. Також він застосовується у програмному забезпеченні для забезпечення цілісності даних, включаючи підписи документів та захист паролів. SHA-256 продовжує залишатися важливим інструментом у світі криптографії і безпеки даних.

Принцип роботи

SHA-256 є криптографічною хеш-функцією, яка приймає на вхід

повідомлення будь-якої довжини і виробляє фіксоване 256-бітне (32-байтове) хеш-значення. Основний принцип роботи SHA-256 полягає у застосуванні низки математичних операцій, що включають побітові обчислення, додавання модулем 2^{32} та інші логічні функції.

Процес хешування складається з декількох етапів:

1. **Доповнення повідомлення**: Повідомлення доповнюється таким чином, щоб його довжина стала кратною 512 бітам. Для цього додається одиничний біт, після чого додаються нулі, і в кінці - 64-бітне представлення початкової довжини повідомлення.
2. **Ініціалізація хеш-значення**: Початкове значення хеш-функції складається з восьми 32-бітних констант, які є першими 32-бітними частинами дробових частин квадратних коренів від перших восьми простих чисел.
3. **Обробка повідомлення блоками**: Повідомлення обробляється по 512-бітним блокам. Кожен блок розбивається на 16 слів по 32 біти, після чого генеруються ще 48 додаткових слів, використовуючи попередні слова та спеціальні функції.
4. **Основний цикл обчислень**: Для кожного з 64 кроків основного циклу обчислень використовуються поточні значення хеш-функції, слова блоку та константи. В кінці кожного блоку отримане значення додається до поточного хеш-значення.
5. **Фінальний хеш**: Після обробки всіх блоків, об'єднане значення всіх етапів є кінцевим хеш-значенням, яке складається з восьми 32-бітних слів.

Переваги

Однією з головних переваг SHA-256 є його висока стійкість до колізій та криптоаналітичних атак. Порівняно зі своїми попередниками, такими як SHA-1, SHA-256 забезпечує набагато вищий рівень безпеки, що робить його надійним вибором для різноманітних криптографічних застосувань.

SHA-256 є детермінованим алгоритмом, тобто для одного і того ж вхідного повідомлення завжди генерується однаковий хеш. Це дозволяє використовувати SHA-256 для перевірки цілісності даних та цифрових підписів, де важливо мати можливість точно відтворити хеш-значення.

Крім того, SHA-256 є відносно швидким алгоритмом, що робить його придатним для використання в різних системах з високими вимогами до продуктивності. Його поширене використання та підтримка у багатьох програмних бібліотеках та апаратних реалізаціях робить його легким у впровадженні та інтеграції.

Недоліки

Однак, попри свої численні переваги, SHA-256 не є позбавленим недоліків. Одним із них є розмір вихідного хешу - 256 бітів, що може бути надмірним для деяких застосувань, де важливим є економія простору. Для таких випадків можуть бути більш придатні алгоритми з коротшими хешами, хоча вони забезпечують менший рівень безпеки.

Ще одним недоліком є вразливість до атак з використанням квантових комп'ютерів. Зокрема, алгоритм Гровера може прискорити пошук колізій до квадратного кореня від загальної кількості можливих значень хешу, що знижує ефективну безпеку SHA-256 до 128 бітів проти квантових атак. Хоча 128-бітна безпека все ще є досить високою, це може стати проблемою в майбутньому з розвитком квантових обчислень.

Крім того, SHA-256, як і інші алгоритми хешування, не захищений від атак, заснованих на зловживаннях в реалізації або неправильному використанні алгоритму. Наприклад, якщо хеш використовується без належного солювання у випадках зберігання паролів.

SHA-256 є потужним та широко використовуваним алгоритмом хешування, який забезпечує високий рівень безпеки та надійності. Його переваги включають стійкість до колізій, швидкість обчислень та поширена підтримка в програмному та апаратному забезпеченні. Однак важливо

враховувати його недоліки, зокрема вразливість до квантових атак та необхідність правильного використання для забезпечення максимального рівня безпеки. Загалом, SHA-256 залишається важливим інструментом у сучасній криптографії, що забезпечує захист даних у багатьох критично важливих застосуваннях.

2.8. PGP (Pretty Good Privacy)

Шифр PGP (Pretty Good Privacy) є одним з перших та найбільш відомих програмних засобів для забезпечення конфіденційності електронної пошти та файлів шляхом шифрування та цифрового підпису. Історія його створення пов'язана з ім'ям Філіпа Циммермана, американського криптографа та активіста з питань приватності.

PGP був розроблений Філіпом Циммерманом у середині 1990-х років. У той час у США була проблема з обмеженнями на використання сильних методів шифрування, і Циммерман хотів створити програмне забезпечення, яке надає простий і доступний спосіб шифрування для захисту приватної комунікації в Інтернеті.

Перша версія PGP була опублікована у 1991 році як вільно розповсюджуване програмне забезпечення. Вона включала алгоритми шифрування з відкритим ключем, такі як RSA для шифрування та підпису, і IDEA для симетричного шифрування. PGP також включав механізми для створення та перевірки цифрових підписів, що забезпечувало цілісність повідомлень.

PGP швидко набув популярності серед користувачів, які шукали засіб для захисту своєї приватної комунікації в Інтернеті. Однак Федеральне бюро розслідувань США (FBI) винесло Циммермана на суд за незаконний експорт програмного забезпечення з сильним шифруванням. Справа протягнулася кілька років, але в підсумку Циммерман був звільнений від звинувачень.

На сьогоднішній день PGP залишається одним з найпопулярніших засобів шифрування для захисту електронної пошти та файлів. Його використовують як приватні особи, так і корпоративні клієнти для забезпечення конфіденційності своїх даних та захисту від несанкціонованого доступу.

Принцип роботи

Шифрування: PGP використовує симетричне та асиметричне шифрування для захисту конфіденційності даних. Для обміну даними між користувачами використовуються симетричні ключі шифрування, які обмінюються за допомогою асиметричного шифрування. Також PGP дозволяє створювати цифрові підписи для документів та повідомлень. Це дозволяє перевірити автентичність відправника та цілісність даних. PGP використовує концепцію "Мережі довіри", де користувачі можуть підтверджувати автентичність публічних ключів інших користувачів, що забезпечує додатковий рівень безпеки.

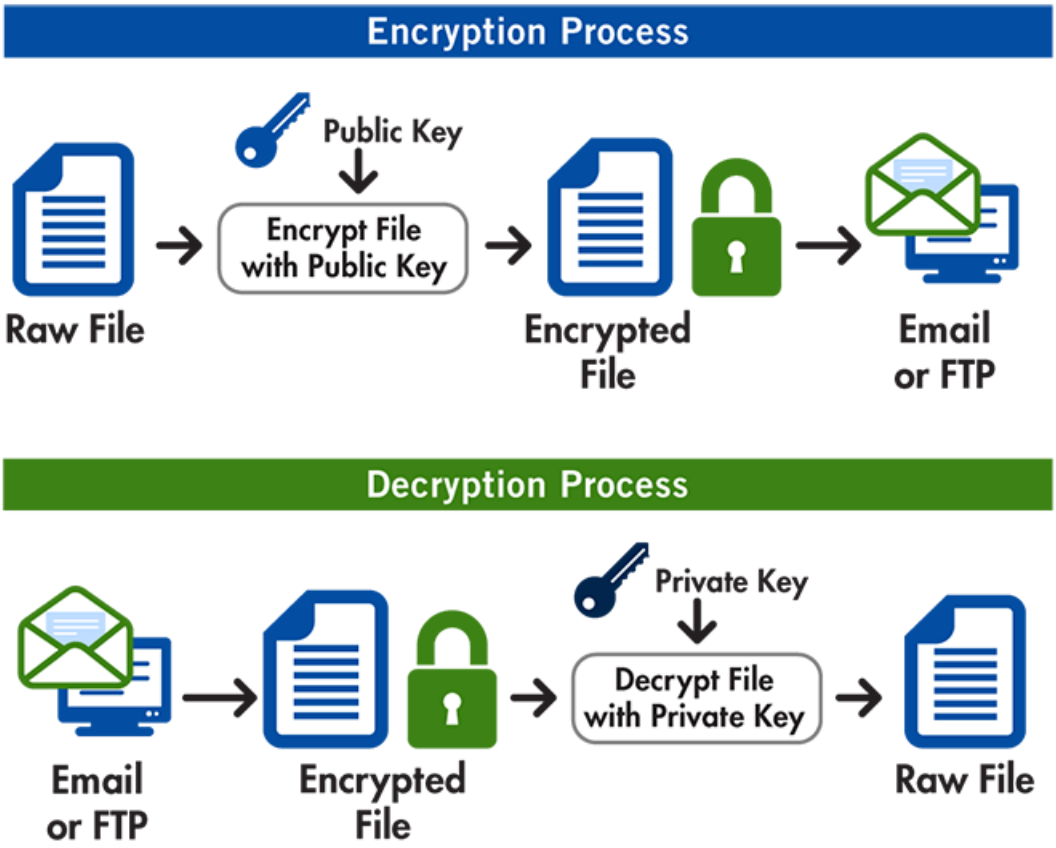


Рис. 2.6

Переваги

1. Конфіденційність: PGP забезпечує високий рівень конфіденційності завдяки потужному шифруванню даних.
2. Автентичність: Завдяки цифровим підписам, PGP дозволяє перевірити автентичність відправника та цілісність даних.
3. Гнучкість ключів: PGP дозволяє користувачам використовувати ключі різного розміру та типу, що забезпечує гнучкість в захисті даних.

Недоліки

1. Складність використання: Для новачків PGP може бути складним у використанні через потребу в розумінні криптографічних концепцій та процесів.
2. Залежність від інфраструктури ключів: Безпека PGP значно залежить від безпеки зберігання та обміну приватних ключів.

РОЗДІЛ 3
Тенденції сучасної криптографії

3.1. Квантова криптографія

Квантова криптографія - це захоплююче поле, що використовує найскладніші принципи квантової механіки для створення надійних методів шифрування та забезпечення безпеки комунікацій. Щоб зрозуміти, як працює квантова криптографія, давайте спробуємо розглянути кілька ключових концепцій.

Принцип квантової переплутаності

Одним з фундаментальних понять квантової криптографії є квантова переплутаність. Це явище, що виникає в квантовій механіці, за якого квантові системи можуть бути взаємозалежними, навіть якщо вони знаходяться на значній відстані одна від одної. Наприклад, якщо дві квантові частинки були переплутані, зміна стану однієї з них миттєво відобразиться на іншій, навіть якщо вони розділені великою відстанню. Це надає квантовим системам унікальну здатність до створення безпечних криптографічних протоколів.

Квантова ключова дистрибуція (QKD)

Одним з найбільш важливих застосувань квантової криптографії є квантова ключова дистрибуція (QKD). Це метод, що дозволяє сторонам обмінюватися криптографічними ключами безпечним шляхом, використовуючи властивості квантової переплутаності. У QKD, вихідні квантові стани надсилаються через канал зв'язку, а подальші спостереження квантових станів дозволяють виявити будь-яке спостереження або зміну стану, що може бути пов'язане з прослуховуванням. Це забезпечує сторонам можливість виявити будь-яке спроби перехоплення або зміни інформації.

Приклад: Протокол BB84

Один із найвідоміших протоколів QKD - це BB84, який був розроблений в 1984 році. У BB84, вихідні біти кодуються в квантові стани фотонів, які надсилаються через канал зв'язку. Одні й ті ж фотони можуть бути відповідно використані для передачі різних бітів, використовуючи різні базові протоколи (наприклад, прямої поляризації або діагональної поляризації). Приймачі випадковим чином вибирають базу для вимірювання і тільки ті фотони, які були виміряні в одній і тій же базі, вважаються корисними бітами. Цей процес забезпечує безпеку передачі ключа, оскільки будь-яка спроба перехоплення фотонів може бути виявлена через зміни у квантових станах.

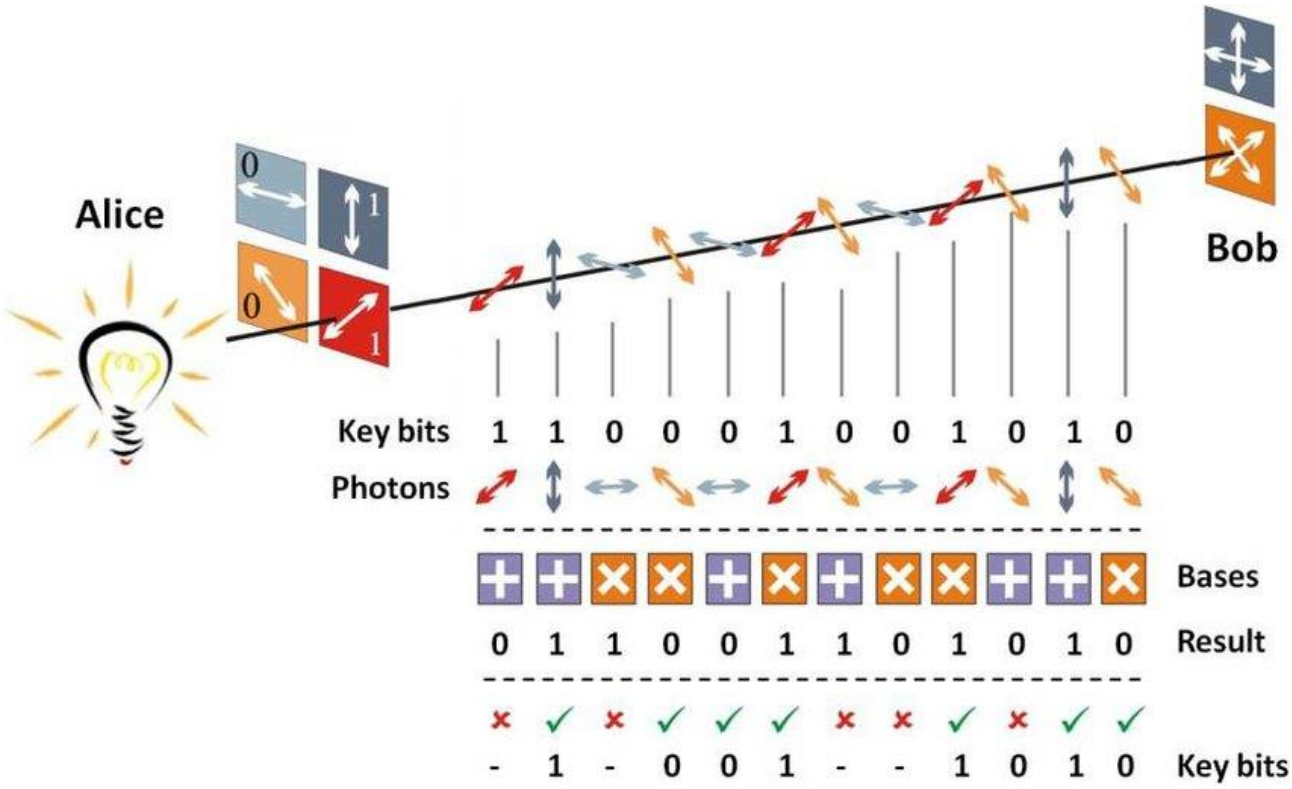


Рис. 3.1

Квантова криптографія відкриває двері до безпеки комунікацій, які раніше були неможливими за допомогою класичних методів шифрування. Її застосування в сферах, таких як фінансові та урядові комунікації, може значно покращити безпеку та приватність даних в цифровому світі. Квантова криптографія представляє собою потужний інструмент у боротьбі з кіберзлочинністю та забезпеченні безпеки цифрових даних у сучасному цифровому світі. Вона відкриває нові можливості для забезпечення

конфіденційності та цілісності інформації і є важливим етапом у розвитку криптографічних технологій.

3.2. Гомоморфне шифрування

В сучасному цифровому світі, де даними обмінюються та обробляють в різних серверах та платформах, захист конфіденційності даних стає все більшою проблемою. Гомоморфне шифрування - це новітній підхід до шифрування, який відкриває нові можливості для обробки даних без розшифрування їх, що має величезне значення для збереження приватності та безпеки.

Гомоморфне шифрування - це метод шифрування, який дозволяє виконувати математичні операції над зашифрованими даними, не розшифровуючи їх. Це означає, що дані можна зашифрувати, виконати певні обчислення над зашифрованими даними, і отримати результат, який буде вірним, навіть коли дані залишаються зашифрованими. Після цього можна розшифрувати результат, і він буде таким самим, як якби були виконані ті ж самі операції над незашифрованими даними.

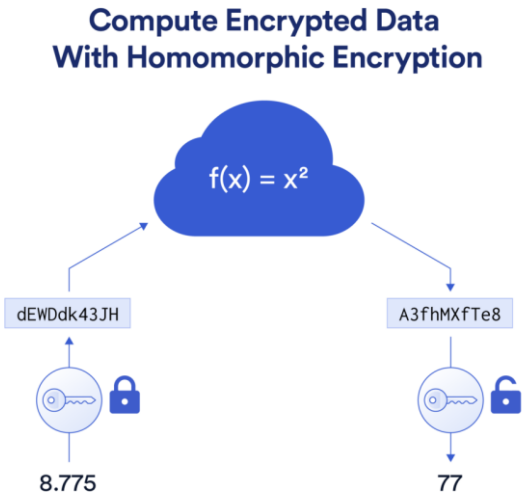


Рис. 3.2

Принципи роботи

Принцип роботи гомоморфного шифрування полягає в тому, що шифрований текст обробляється таким чином, що результат операції над ним в шифрованому вигляді відповідає результату тієї ж операції над відкритим текстом. Наприклад, якщо ми зашифруємо два числа, а потім зашифруємо їхню суму, результат буде такий самий, як якби ми зашифрували суму двох чисел окремо. Це означає, що ми можемо виконувати операції над даними без розшифровування їх, що забезпечує безпеку та конфіденційність.

Переваги

Гомоморфне шифрування має кілька переваг:

- **Конфіденційність**: Дані залишаються зашифрованими протягом всього процесу обробки, що забезпечує їх конфіденційність.
- **Приватність**: Зберігання та обробка даних може відбуватися на захищених серверах без необхідності розкриття відкритого тексту.
- **Безпека**: Навіть якщо зловмисник отримає доступ до зашифрованих даних та результатів операцій, він не зможе отримати доступ до оригінальних даних без ключа розшифрування.

Один із прикладів можливого використання гомоморфного шифрування - це обробка медичних даних. Лікарі можуть використовувати гомоморфне шифрування для обробки конфіденційних медичних даних, таких як результати тестів або зображення, без розкриття особистої інформації пацієнтів.

Інший приклад - це обробка фінансових даних. Банки та фінансові установи можуть використовувати гомоморфне шифрування для безпечної обробки та аналізу фінансових транзакцій без ризику розкриття конфіденційної інформації клієнтів.

Гомоморфне шифрування відкриває нові можливості для безпечної та приватної обробки даних, зберігаючи її конфіденційність та цілісність. Це потужний інструмент у сучасній криптографії, який може бути використаний в різних сферах, включаючи медицину, фінанси та інформаційні технології.

3.3. Криптовалютні технології

Криптовалютні технології - це новий напрямок у фінансовій та технологічній галузях, який забезпечує безпечні та анонімні транзакції в Інтернеті за допомогою набору інноваційних методів шифрування. Основою криптовалют є блокчейн технологія, яка забезпечує децентралізовану систему обліку та переведення коштів. Блокчейн - розподілена база даних, яка забезпечує безпеку та непідробність транзакцій шляхом збереження історії операцій у блоках.

Ключовим принципом цих технологій є децентралізація. Вони використовують розподілену мережу вузлів для підтвердження та запису транзакцій у блокчейні. Кожен вузол мережі має копію блокчейну та відповідно до правил протоколу підтверджує та обробляє нові транзакції. Кожна транзакція підписується цифровим підписом, який гарантує її автентичність та непідробність.

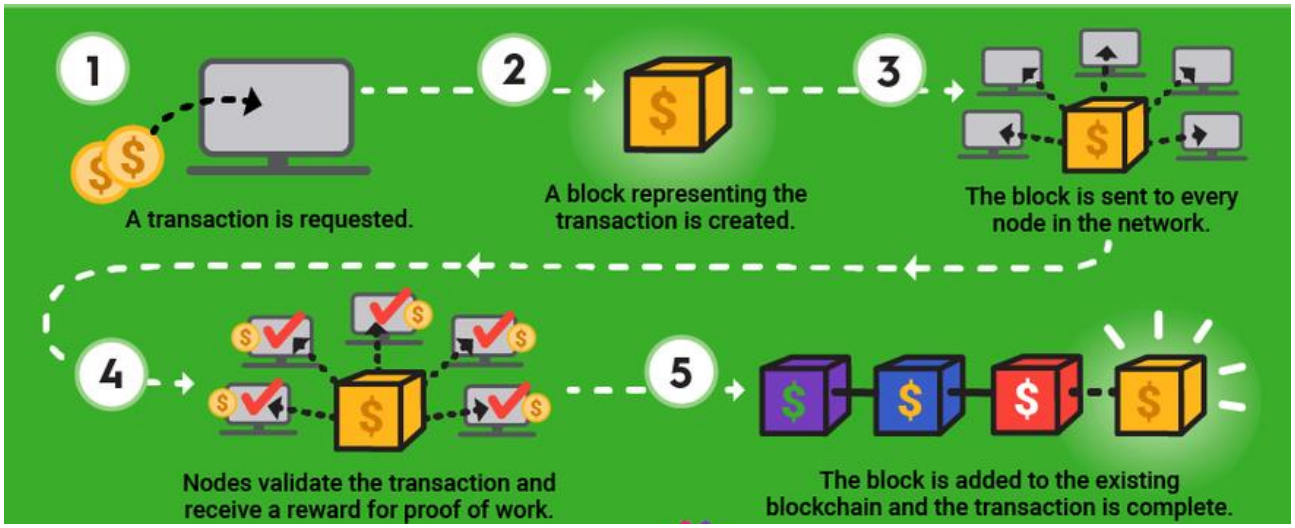


Рис. 3.3

Приклади використання

Одним з найвідоміших прикладів криптовалютної технології є Bitcoin. Bitcoin є першою та найбільш популярною криптовалютою, яка використовує блокчейн технологію для забезпечення безпечних та анонімних транзакцій.

Ще одним прикладом є Ethereum, який використовується для створення розумних контрактів - програм, що автоматизують виконання угод без необхідності посередників.

Переваги

Переваги криптовалютних технологій включають:

- Анонімність: користувачі можуть здійснювати транзакції без необхідності розкривати свою особисту інформацію.
- Низькі комісії: транзакції в криптовалютній мережі можуть мати значно менші комісії порівняно з традиційними банківськими переказами.
- Доступність: криптовалютні технології доступні будь-якому користувачу з Інтернет-підключенням.

Недоліки

Недоліки криптовалютних технологій включають:

- Волатильність: ціни на криптовалюту можуть дуже змінюватись в короткі терміни, що робить їх менш стабільними для збереження вартості.
- Відсутність регуляції: брак регуляції може призвести до зловживань та маніпуляцій на ринку.

Криптовалютні технології відкривають нові можливості для фінансової свободи та конфіденційності в Інтернеті. Вони представляють собою новий етап у розвитку фінансової системи та мають потенціал змінити спосіб, яким ми взаємодіємо з грошима та іншими цифровими активами.

3.4. Мультифакторна аутентифікація

У сучасному цифровому світі, де безпека особистих даних стає все більшою проблемою, мультифакторна аутентифікація є важливим засобом захисту від несанкціонованого доступу до особистої інформації та фінансових ресурсів. Мультифакторна аутентифікація (МФА) - це метод захисту, який

вимагає від користувача подання двох або більше різних видів ідентифікаційних даних для підтвердження його особи перед наданням доступу до системи чи сервісу. Ці види ідентифікації можуть включати щось, що користувач знає (наприклад, пароль), щось, що він має (наприклад, фізичний пристрій або токен), або ще щось особливе, наприклад, біометричні дані.

Принципи роботи

Основна ідея МФА полягає в тому, щоб у разі втрати або компрометації одного фактора ідентифікації, інші фактори все ще залишалися надійними засобами підтвердження особи. Це забезпечує вищий рівень безпеки, оскільки навіть якщо зловмисник отримає доступ до одного фактора (наприклад, пароля), він все одно не зможе отримати доступ до системи без іншого фактора (наприклад, фізичного пристрою або біометричних даних).

Приклади використання

Один з найпоширеніших прикладів використання МФА - це банківські системи онлайн-банкінгу. Коли користувач намагається увійти в свій банківський рахунок, система може вимагати від нього введення пароля, а потім підтвердження доступу за допомогою коду, який відправляється на його мобільний телефон.

Інший приклад - це використання МФА в корпоративних системах. Компанії можуть вимагати від своїх співробітників використовувати спеціальні картки доступу або біометричні дані для входу в інформаційні системи компанії.

Переваги та недоліки мультифакторної аутентифікації

Переваги мультифакторної автентифікації включають:

- **Високий рівень безпеки**: використання кількох факторів ідентифікації забезпечує вищий рівень захисту від несанкціонованого доступу.
- **Зменшення ризику кібератак**: навіть якщо один фактор ідентифікації був компрометований, інші фактори все ще залишаються надійними.

Недоліки мультифакторної автентифікації включають:

- **Складність для користувачів**: деякі люди можуть вважати процес аутентифікації занадто складним або незручним.
- **Вартість імплементації**: впровадження систем МФА може бути витратним завданням для підприємств.

Мультифакторна аутентифікація є ефективним засобом захисту особистих даних та фінансових ресурсів у сучасному цифровому світі. Вона дозволяє забезпечити високий рівень безпеки та захисту від кіберзлочинців, зберігаючи при цьому зручність користувача.

3.5. Майбутнє криптографії

Майбутнє криптографії обіцяє бути багатогранним і насиченим інноваціями, що будуть зумовлені швидким розвитком технологій і новими викликами в сфері кібербезпеки. Одним із найважливіших напрямів розвитку криптографії стане адаптація до квантових обчислень. Квантові комп'ютери, здатні виконувати складні обчислення значно швидше, ніж сучасні комп'ютери, ставлять під загрозу безпеку традиційних криптографічних алгоритмів, таких як RSA та ECC. Це стимулює розвиток квантової та пост-квантової криптографії, яка використовує нові математичні основи для забезпечення стійкості до квантових атак.

Важливим аспектом майбутнього криптографії є розвиток інфраструктури для підтримки нових криптографічних методів. Це включає квантові мережі та комунікаційні канали, які дозволять впроваджувати квантову криптографію на практиці. Крім того, інтеграція криптографічних рішень з іншими технологіями, такими як блокчейн, може забезпечити додаткові рівні безпеки та надійності в різних галузях, включаючи фінансові послуги, управління ланцюгами постачань та охорону здоров'я.

Підвищення автоматизації в криптографії також є важливим напрямом розвитку. Використання штучного інтелекту та машинного навчання для аналізу криптографічних даних і виявлення загроз може значно покращити

захист інформаційних систем. Такі технології здатні автоматично виявляти аномалії та вразливості, пропонуючи оптимальні рішення для їх усунення.

Етичні та правові аспекти криптографії також набуватимуть все більшого значення. Регулювання використання криптографічних методів, зокрема у сфері криптовалют та блокчейн-технологій, буде вимагати уваги з боку урядів та міжнародних організацій. Забезпечення балансу між конфіденційністю, безпекою та законністю використання криптографії стане ключовим завданням.

Таким чином, майбутнє криптографії буде визначатися як технологічними інноваціями, так і здатністю адаптуватися до нових загроз та викликів. Розвиток квантових та пост-квантових алгоритмів, впровадження гомоморфного шифрування, розширення інфраструктури та інтеграція з іншими технологіями будуть критично важливими для забезпечення безпеки інформації в цифрову епоху. Криптографія залишатиметься основним елементом кібербезпеки, а її еволюція матиме значний вплив на захист даних та конфіденційність. Враховуючи всі ці аспекти, майбутнє криптографії обіцяє бути динамічним та насиченим новими викликами і можливостями. Важливість криптографії у забезпеченні безпеки та конфіденційності даних лише зростатиме, і нові дослідження та розробки у цій сфері матимуть значний вплив на наш цифровий світ.

РОЗДІЛ 4
Розробка програмного забезпечення

Розробка програмного забезпечення для візуалізації алгоритмів шифрування є складним і водночас цікавим завданням, яке потребує глибокого розуміння як криптографічних алгоритмів, так і сучасних засобів програмування. У нашій дипломній роботі було поставлено завдання створити інструмент, який би наочно демонстрував роботу різних алгоритмів шифрування, що дозволило б студентам, дослідникам і розробникам легше розуміти принципи їхньої дії. Основною метою було зробити процес вивчення шифрування більш інтуїтивним і зрозумілим за допомогою візуалізацій.

4.1 Обґрунтування вибору мови програмування

Для реалізації нашого програмного забезпечення ми обрали мову програмування Python. Це рішення було прийнято на основі кількох важливих факторів. По-перше, Python є однією з найбільш популярних мов програмування у світі, відомою своєю простотою і читабельністю коду. Це особливо важливо для освітніх цілей, адже основною аудиторією нашого інструменту є студенти та початківці у сфері криптографії, які тільки починають своє знайомство з програмуванням і шифруванням.

По-друге, Python має величезну кількість бібліотек та фреймворків, які значно полегшують розробку складних програмних систем. Зокрема, для нашої роботи ми використали бібліотеки `tkinter` для створення графічного інтерфейсу користувача та `matplotlib` для візуалізації даних. Ці бібліотеки є стандартними для Python і широко використовуються у наукових та освітніх проектах завдяки їхній потужності та гнучкості.

По-третє, Python є інтерпретованою мовою програмування, що забезпечує швидку розробку та тестування коду. Це дозволило нам оперативно вносити

зміни та покращення до нашого програмного забезпечення, забезпечуючи його стабільність та функціональність.

4.2 Реалізація програмного забезпечення

Процес розробки програмного забезпечення розпочався з визначення основних криптографічних алгоритмів, які ми хотіли візуалізувати. Було вирішено зосередитися на класичних алгоритмах, таких як шифр Цезаря, шифр Плейфера, а також на сучасних алгоритмах, таких як AES. Кожен з цих алгоритмів має свої унікальні характеристики та підходи до шифрування даних, що дозволяє отримати всебічне розуміння принципів криптографії.

Для кожного алгоритму було розроблено окремий модуль, який відповідав за його реалізацію та візуалізацію. Наприклад, для шифру Плейфера ми створили програму, яка демонструвала кожен етап шифрування біграм, включаючи створення матриці, розбиття тексту на біграми та заміну символів згідно з правилами шифрування. Ми забезпечили динамічне оновлення матриці та додали візуальні елементи, такі як стрілки, які вказували на переміщення символів під час шифрування.

Для кращого розуміння можливостей розробленого програмного забезпечення, нижче представлено скріншот інтерфейсу користувача. На ньому можна побачити головні елементи управління та візуалізації, які використовуються для демонстрації роботи криптографічних алгоритмів.

4.3 Труднощі та виклики

Розробка програмного забезпечення для візуалізації алгоритмів шифрування зіткнулася з низкою труднощів. Однією з головних проблем було забезпечення динамічної та наочної візуалізації процесу шифрування. Це вимагало створення складних графічних елементів та їх інтеграції з

алгоритмами шифрування таким чином, щоб кожен крок шифрування був зрозумілим та інтуїтивно зрозумілим для користувача. Нам довелося ретельно працювати над синхронізацією змін у матриці та інших візуальних елементах, щоб забезпечити коректне відображення всіх кроків шифрування.

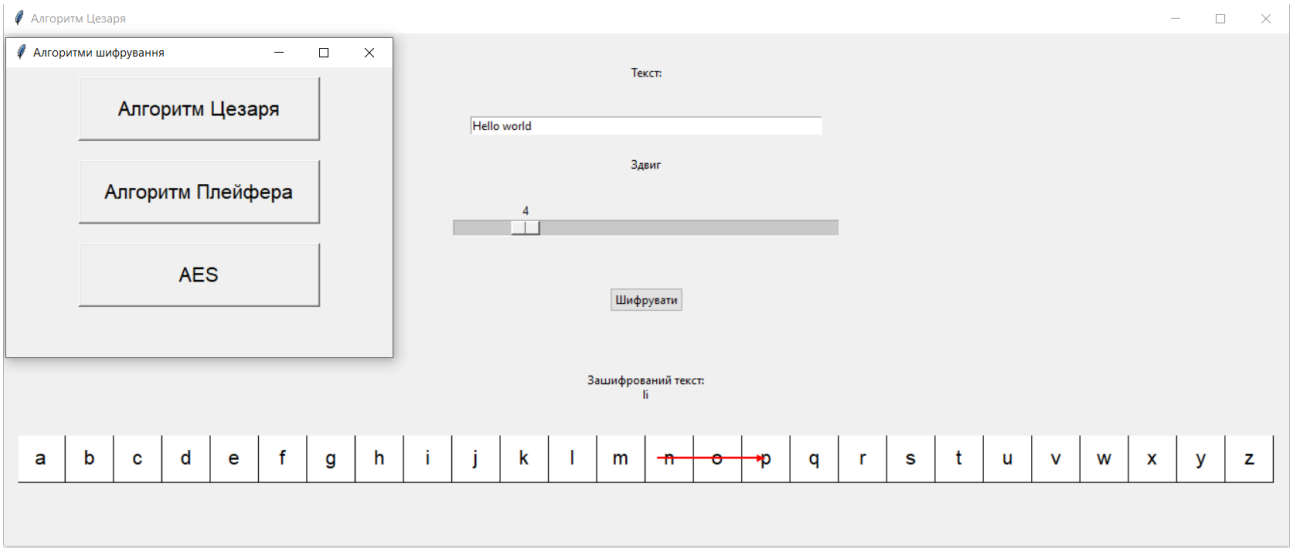


Рис. 4.1

Ще одним викликом було забезпечення взаємодії користувача з програмним забезпеченням. Ми хотіли, щоб користувачі могли вводити власні ключі та тексти для шифрування, а також мали можливість контролювати процес візуалізації. Це вимагало створення зручного та інтуїтивно зрозумілого графічного інтерфейсу користувача з різними елементами керування, такими як поля введення, кнопки та інші інтерактивні елементи.

Крім того, ми зіткнулися з проблемою обробки великих обсягів даних при шифруванні текстів, що вимагало оптимізації алгоритмів для забезпечення їхньої ефективності та швидкодії. Ми працювали над оптимізацією коду, щоб мінімізувати час обробки даних та забезпечити плавність візуалізації.

4.4 Тестування та валідація

Після завершення розробки основних модулів програмного забезпечення ми перейшли до етапу тестування та валідації. Було проведено серію тестів для

перевірки коректності роботи кожного алгоритму та його візуалізації. Ми залучили групу користувачів, які протестували програмне забезпечення та надали зворотний зв'язок щодо його функціональності та зручності використання.

На основі отриманих відгуків ми внесли необхідні покращення та виправили помилки. Було вдосконалено інтерфейс користувача, додано додаткові пояснення та підказки для спрощення роботи з програмою, а також покращено візуальні елементи для забезпечення більшої наочності.

Розробка програмного забезпечення для візуалізації алгоритмів шифрування на мові Python виявилася корисним проектом. Ми змогли створити інструмент, який значно полегшує вивчення криптографічних алгоритмів, роблячи процес їхнього розуміння більш інтуїтивним та наочним. Використання Python дозволило нам швидко та ефективно реалізувати всі необхідні функції, а також забезпечити високу якість та стабільність програмного забезпечення.

Наше програмне забезпечення може бути використане як у навчальних цілях, так і для професійної підготовки фахівців у галузі криптографії. Воно надає користувачам можливість експериментувати з різними алгоритмами шифрування, розуміти їхні основні принципи та аналізувати їхню ефективність. Ми сподіваємось, що цей інструмент стане корисним ресурсом для багатьох людей, які цікавляться криптографією та інформаційною безпекою.

4.5. Можливі напрями вдосконалення розробленого програмного забезпечення

У рамках подальшого розвитку та вдосконалення програмного забезпечення для візуалізації алгоритмів шифрування є кілька перспективних напрямків, які можуть суттєво покращити його функціональність, зручність використання та освітню цінність. По-перше, важливим аспектом є розширення

кількості підтримуваних алгоритмів шифрування. Хоча на даний момент наш інструмент охоплює класичні алгоритми, додавання нових методів, таких як RC4, DES або алгоритмів з еліптичними кривими, дозволить користувачам отримати більш повне уявлення про різноманітність криптографічних підходів. Це розширення особливо важливе для тих, хто прагне глибше вивчати криптографію та працювати з більш складними системами захисту даних.

Другим напрямом удосконалення є поліпшення інтерфейсу користувача. Хоча нинішній графічний інтерфейс забезпечує базову взаємодію з програмою, його можна зробити більш інтуїтивно зрозумілим і привабливим. Важливо додати інтерактивні підказки та інструкції, які допоможуть користувачам краще розуміти функції та можливості програми. Використання сучасних підходів до дизайну інтерфейсу, таких як адаптивний дизайн, дозволить програмі краще відповідати різним пристроям та екранам, забезпечуючи зручність використання як на стаціонарних комп’ютерах, так і на мобільних пристроях.

Третім напрямом удосконалення є інтеграція елементів гейміфікації. Додавання освітніх ігор або змагальних елементів може значно підвищити мотивацію користувачів до вивчення криптографії. Наприклад, можна реалізувати завдання, де користувачі змагатимуться у шифруванні та дешифруванні повідомлень, отримуючи при цьому бали та нагороди. Такий підхід не тільки зробить процес навчання більш цікавим, але й сприятиме закріпленню знань на практиці.

Іншим важливим напрямом є покращення технічної сторони програми, зокрема оптимізація алгоритмів для підвищення продуктивності. Це особливо актуально для обробки великих обсягів даних або при використанні ресурсоємних алгоритмів. Використання більш ефективних структур даних, паралельних обчислень та інших оптимізаційних технік дозволить забезпечити швидку і плавну роботу програми навіть при високих навантаженнях.

Крім того, важливо розглянути можливість інтеграції з іншими освітніми та дослідницькими платформами. Наприклад, розробка модулів для популярних

систем управління навчанням (LMS), таких як Moodle або Blackboard, дозволить використовувати програму у навчальних курсах та семінарах. Також варто розглянути можливість створення API для взаємодії з іншими програмними засобами, що дозволить розширити функціональність та забезпечити більшу гнучкість у використанні програми в різних контекстах.

Ще одним важливим аспектом є розвиток функцій зворотного зв'язку та підтримки користувачів. Додавання можливостей для подання відгуків, звітів про помилки та пропозицій щодо покращення дозволить оперативно реагувати на потреби користувачів та вносити необхідні корективи. Також варто створити детальну документацію та навчальні матеріали, які допоможуть користувачам швидко освоїти програму та ефективно використовувати її можливості.

Нарешті, важливо враховувати безпекові аспекти самої програми. Хоча наша мета - навчання та візуалізація алгоритмів шифрування, програмне забезпечення повинно бути захищеним від потенційних загроз та зловживань. Впровадження сучасних засобів захисту, таких як шифрування даних користувачів, контроль доступу та регулярні оновлення безпеки, дозволить забезпечити надійність та довіру до нашого продукту.

Таким чином, розглянуті напрями вдосконалення програмного забезпечення для візуалізації алгоритмів шифрування охоплюють як технічні, так і користувацькі аспекти. Впровадження цих покращень дозволить не лише підвищити якість та функціональність програми, але й зробить процес вивчення криптографії більш ефективним, цікавим та доступним для широкого кола користувачів.

ВИСНОВКИ

У процесі виконання роботи був проведений аналіз теоретичних та практичних аспектів криптографії. Вивчення різноманітних шифрів та їх реалізацій дозволило глибше зрозуміти основні принципи, на яких базується захист інформації у сучасних умовах. Робота включала огляд історичного розвитку криптографічних методів, аналіз сучасних алгоритмів шифрування, а також розробку програмного забезпечення для їх візуалізації.

На початку роботи було розглянуто класичні алгоритми шифрування, такі як шифр Цезаря, шифр Плейфера, а також більш складні системи, включаючи RSA та AES. Ми детально проаналізували принципи їх роботи, сильні та слабкі сторони, а також області застосування. Особлива увага була приділена алгоритмам симетричного та асиметричного шифрування, що становлять основу сучасних систем захисту інформації. Симетричні алгоритми, такі як AES, забезпечують високу швидкість обробки даних, але потребують безпечного обміну ключами між сторонами. Натомість асиметричні алгоритми, такі як RSA, хоча і є менш ефективними з точки зору швидкості, пропонують зручніші механізми управління ключами та забезпечують високий рівень безпеки за рахунок використання пар ключів.

Особливе місце займали алгоритми хешування, які грають важливу роль у забезпеченні цілісності даних і автентифікації. Ми розглянули такий алгоритм, як SHA-256, і продемонстрували їх застосування у різних контекстах, від захисту паролів до забезпечення цілісності файлів.

Розроблене програмне забезпечення для візуалізації алгоритмів шифрування є важливим інструментом для тих, хто тільки починає знайомитись з криптографією. Цей інструмент дозволяє наочно демонструвати процеси шифрування та дешифрування, що значно полегшує розуміння складних математичних концепцій та алгоритмічних процедур. Програма показує, як змінюються дані на різних етапах шифрування, і дає можливість користувачам експериментувати з різними ключами та алгоритмами, що сприяє

глибшому засвоєнню матеріалу.

Таким чином, проведена робота і розробка програмного забезпечення для візуалізації алгоритмів шифрування зробили значний внесок у розвиток засобів навчання і досліджень у галузі криптографії. Наш інструмент відкриває нові можливості для ефективного вивчення складних криптографічних концепцій, підтримки освітніх програм та професійної підготовки фахівців. Це сприятиме покращенню якості освіти у сфері інформаційної безпеки та стимулюватиме подальший розвиток криптографічних технологій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Криптографія — Вікіпедія Wikipedia
2. Menezes, A. J., van Oorschot, P. C., & Vanstone, S. A. Handbook of Applied Cryptography. 1996.
3. Stallings, W. Cryptography and Network Security: Principles and Practice (7th Edition). Pearson, 2016.
4. Katz, J., & Lindell, Y. Introduction to Modern Cryptography (3rd Edition). CRC Press, 2020.
5. Schneier, B. Applied Cryptography: Protocols, Algorithms, and Source Code in C (2nd Edition). John Wiley & Sons, 1996.
6. Rivest, R. L., Shamir, A., & Adleman, L. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Communications of the ACM, 21(2), 120-126, 1978.
7. Boneh, D., & Shoup, V. A Graduate Course in Applied Cryptography. Draft, 2017.
8. National Institute of Standards and Technology (NIST). Advanced Encryption Standard (AES). FIPS PUB 197, 2001.
9. Buchmann, J. Introduction to Cryptography. Springer, 2004.
10. Павленко, А. В., Козловський, І. Ю. Сучасні методи шифрування: огляд та порівняння. Інформаційна безпека, 2018.
11. Сизов, А. А. Теоретичні основи криптографії: монографія. Харків: ХНУРЕ, 2016.
12. Безручко, В. М. Основи криптографії: підручник. Київ: КНЕУ, 2012.

ДОДАТОК А

Код розробленого програмного забезпечення

```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import string
from tkinter import messagebox, ttk, Canvas, font
import time
import copy
import tkinter as tk
from time import sleep

round_constants = (
    0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80,
    0x1B, 0x36, 0x6C, 0xD8, 0xAB, 0x4D, 0x9A, 0x2F,
    0x5E, 0xBC, 0x63, 0xC6, 0x97, 0x35, 0x6A, 0xD4,
)

sbox = (
    0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5, 0x30, 0x01, 0x67, 0x2B, 0xFE,
0xD7, 0xAB, 0x76,
    0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0, 0xAD, 0xD4, 0xA2, 0xAF,
0x9C, 0xA4, 0x72, 0xC0,
    0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7, 0xCC, 0x34, 0xA5, 0xE5, 0xF1,
0x71, 0xD8, 0x31, 0x15,
    0x04, 0xC7, 0x23, 0xC3, 0x18, 0x96, 0x05, 0x9A, 0x07, 0x12, 0x80, 0xE2, 0xEB,
0x27, 0xB2, 0x75,
    0x09, 0x83, 0x2C, 0x1A, 0x1B, 0x6E, 0x5A, 0xA0, 0x52, 0x3B, 0xD6, 0xB3,
0x29, 0xE3, 0x2F, 0x84,
```

```
    0x53, 0xD1, 0x00, 0xED, 0x20, 0xFC, 0xB1, 0x5B, 0x6A, 0xCB, 0xBE, 0x39,
0x4A, 0x4C, 0x58, 0xCF,
    0xD0, 0xEF, 0xAA, 0xFB, 0x43, 0x4D, 0x33, 0x85, 0x45, 0xF9, 0x02, 0x7F,
0x50, 0x3C, 0x9F, 0xA8,
    0x51, 0xA3, 0x40, 0x8F, 0x92, 0x9D, 0x38, 0xF5, 0xBC, 0xB6, 0xDA, 0x21,
0x10, 0xFF, 0xF3, 0xD2,
    0xCD, 0x0C, 0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17, 0xC4, 0xA7, 0x7E, 0x3D,
0x64, 0x5D, 0x19, 0x73,
    0x60, 0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88, 0x46, 0xEE, 0xB8, 0x14,
0xDE, 0x5E, 0x0B, 0xDB,
    0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C, 0xC2, 0xD3, 0xAC, 0x62,
0x91, 0x95, 0xE4, 0x79,
    0xE7, 0xC8, 0x37, 0x6D, 0x8D, 0xD5, 0x4E, 0xA9, 0x6C, 0x56, 0xF4, 0xEA,
0x65, 0x7A, 0xAE, 0x08,
    0xBA, 0x78, 0x25, 0x2E, 0x1C, 0xA6, 0xB4, 0xC6, 0xE8, 0xDD, 0x74, 0x1F,
0x4B, 0xBD, 0x8B, 0x8A,
    0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E, 0x61, 0x35, 0x57, 0xB9, 0x86,
0xC1, 0x1D, 0x9E,
    0xE1, 0xF8, 0x98, 0x11, 0x69, 0xD9, 0x8E, 0x94, 0x9B, 0x1E, 0x87, 0xE9, 0xCE,
0x55, 0x28, 0xDF,
    0x8C, 0xA1, 0x89, 0x0D, 0xBF, 0xE6, 0x42, 0x68, 0x41, 0x99, 0x2D, 0x0F,
0xB0, 0x54, 0xBB, 0x16,
)

inverted_sbox = (
    0x52, 0x09, 0x6A, 0xD5, 0x30, 0x36, 0xA5, 0x38, 0xBF, 0x40, 0xA3, 0x9E,
0x81, 0xF3, 0xD7, 0xFB,
    0x7C, 0xE3, 0x39, 0x82, 0x9B, 0x2F, 0xFF, 0x87, 0x34, 0x8E, 0x43, 0x44, 0xC4,
0xDE, 0xE9, 0xCB,
```

0x54, 0x7B, 0x94, 0x32, 0xA6, 0xC2, 0x23, 0x3D, 0xEE, 0x4C, 0x95, 0x0B, 0x42, 0xFA, 0xC3, 0x4E,

0x08, 0x2E, 0xA1, 0x66, 0x28, 0xD9, 0x24, 0xB2, 0x76, 0x5B, 0xA2, 0x49, 0x6D, 0x8B, 0xD1, 0x25,

0x72, 0xF8, 0xF6, 0x64, 0x86, 0x68, 0x98, 0x16, 0xD4, 0xA4, 0x5C, 0xCC, 0x5D, 0x65, 0xB6, 0x92,

0x6C, 0x70, 0x48, 0x50, 0xFD, 0xED, 0xB9, 0xDA, 0x5E, 0x15, 0x46, 0x57, 0xA7, 0x8D, 0x9D, 0x84,

0x90, 0xD8, 0xAB, 0x00, 0x8C, 0xBC, 0xD3, 0x0A, 0xF7, 0xE4, 0x58, 0x05, 0xB8, 0xB3, 0x45, 0x06,

0xD0, 0x2C, 0x1E, 0x8F, 0xCA, 0x3F, 0x0F, 0x02, 0xC1, 0xAF, 0xBD, 0x03, 0x01, 0x13, 0x8A, 0x6B,

0x3A, 0x91, 0x11, 0x41, 0x4F, 0x67, 0xDC, 0xEA, 0x97, 0xF2, 0xCF, 0xCE, 0xF0, 0xB4, 0xE6, 0x73,

0x96, 0xAC, 0x74, 0x22, 0xE7, 0xAD, 0x35, 0x85, 0xE2, 0xF9, 0x37, 0xE8, 0x1C, 0x75, 0xDF, 0x6E,

0x47, 0xF1, 0x1A, 0x71, 0x1D, 0x29, 0xC5, 0x89, 0x6F, 0xB7, 0x62, 0x0E, 0xAA, 0x18, 0xBE, 0x1B,

0xFC, 0x56, 0x3E, 0x4B, 0xC6, 0xD2, 0x79, 0x20, 0x9A, 0xDB, 0xC0, 0xFE, 0x78, 0xCD, 0x5A, 0xF4,

0x1F, 0xDD, 0xA8, 0x33, 0x88, 0x07, 0xC7, 0x31, 0xB1, 0x12, 0x10, 0x59, 0x27, 0x80, 0xEC, 0x5F,

0x60, 0x51, 0x7F, 0xA9, 0x19, 0xB5, 0x4A, 0x0D, 0x2D, 0xE5, 0x7A, 0x9F, 0x93, 0xC9, 0x9C, 0xEF,

0xA0, 0xE0, 0x3B, 0x4D, 0xAE, 0x2A, 0xF5, 0xB0, 0xC8, 0xEB, 0xBB, 0x3C, 0x83, 0x53, 0x99, 0x61,

0x17, 0x2B, 0x04, 0x7E, 0xBA, 0x77, 0xD6, 0x26, 0xE1, 0x69, 0x14, 0x63, 0x55, 0x21, 0x0C, 0x7D,

)

```
def pleifer():
    class PlayfairCipherApp:
        def __init__(self, root):
            self.root = root
            self.root.title("Playfair Cipher Visualization")
            self.root.geometry('800x740')

            self.key_label = tk.Label(root, text="Enter Key:")
            self.key_label.pack()
            self.key_entry = tk.Entry(root)
            self.key_entry.pack()
            self.text_label = tk.Label(root, text="Enter Text:")
            self.text_label.pack()
            self.text_entry = tk.Entry(root)
            self.text_entry.pack()
            self.encrypt_button = tk.Button(root, text="Encrypt", command=self.encrypt)
            self.encrypt_button.pack(padx=230, pady=20)
            self.result_label = tk.Label(root, text="Зашифрований текст:\n",
wraplength=800)
            self.result_label.pack(pady=10)

            self.canvas = None

        def create_playfair_matrix(self, key):
            matrix = []
            alphabet = "ABCDEFGHIKLMNOPQRSTUVWXYZ"
            used_chars = set()
```

```
    key = key.upper().replace("J", "I")
    for char in key:
        if char not in used_chars and char in alphabet:
            used_chars.add(char)
            matrix.append(char)

    for char in alphabet:
        if char not in used_chars:
            used_chars.add(char)
            matrix.append(char)

    matrix = np.array(matrix).reshape(5, 5)
    return matrix

def display_matrix(self, matrix, highlight_positions=[], arrows=[]):
    if self.canvas:
        self.canvas.get_tk_widget().pack_forget()

    fig, ax = plt.subplots()
    ax.imshow(np.zeros((5, 5)), cmap="Greys", vmin=0, vmax=1)

    for i in range(5):
        for j in range(5):
            color = 'black'
            if (i, j) in highlight_positions:
                color = 'red'
            ax.text(j, i, matrix[i, j], ha='center', va='center', fontsize=20, color=color)
```

```
            for arrow in arrows:
                start, end = arrow
                ax.annotate('', xy=(start[1], start[0]), xytext=(end[1], end[0]),
                            arrowprops=dict(facecolor='blue', edgecolor='blue', arrowstyle='-
>'))

            self.canvas = FigureCanvasTkAgg(fig, master=self.root)
            ax1 = plt.gca()
            ax1.get_xaxis().set_visible(False)
            ax1.get_yaxis().set_visible(False)
            self.canvas.draw()
            self.canvas.get_tk_widget().pack()
            self.root.update()
            time.sleep(5)

        def prepare_text(self, text):
            text = text.upper().replace("J", "I").replace(" ", "")
            prepared = ""
            i = 0
            while i < len(text):
                if i + 1 < len(text) and text[i] == text[i + 1]:
                    prepared += text[i] + "X"
                    i += 1
                else:
                    prepared += text[i]
                    if i + 1 < len(text):
                        prepared += text[i + 1]
                    i += 2
            if len(prepared) % 2 != 0:
```

```
            prepared += "X"
        return prepared

    def encrypt_pair(self, a, b, matrix):
        pos_a = np.where(matrix == a)
        pos_b = np.where(matrix == b)
        arrows = []

        if pos_a[0] == pos_b[0]:
            new_a = matrix[pos_a[0], (pos_a[1] + 1) % 5][0]
            new_b = matrix[pos_b[0], (pos_b[1] + 1) % 5][0]
            arrows.append(((pos_a[0][0], (pos_a[1][0] + 1) % 5), (pos_a[0][0],
pos_a[1][0])))
            arrows.append(((pos_b[0][0], (pos_b[1][0] + 1) % 5), (pos_b[0][0],
pos_b[1][0])))
        elif pos_a[1] == pos_b[1]:
            new_a = matrix[(pos_a[0] + 1) % 5, pos_a[1]][0]
            new_b = matrix[(pos_b[0] + 1) % 5, pos_b[1]][0]
            arrows.append((((pos_a[0][0] + 1) % 5, pos_a[1][0]), (pos_a[0][0],
pos_a[1][0])))
            arrows.append((((pos_b[0][0] + 1) % 5, pos_b[1][0]), (pos_b[0][0],
pos_b[1][0])))
        else:
            new_a = matrix[pos_a[0], pos_b[1]][0]
            new_b = matrix[pos_b[0], pos_a[1]][0]
            arrows.append(((pos_a[0][0], pos_b[1][0]), (pos_a[0][0], pos_a[1][0])))
            arrows.append(((pos_b[0][0], pos_a[1][0]), (pos_b[0][0], pos_b[1][0])))

        return new_a + new_b, arrows
```

```
    def encrypt(self):
        key = self.key_entry.get().strip().lower()
        text = self.text_entry.get().strip().lower()
        text = text.replace(' ', '')
        key = key.replace(' ', '')

        if not key or not text:
            messagebox.showerror("Input Error", "Please enter both key and text.")
            return
        for char in text:
            if not ('A' <= char <= 'Z' or 'a' <= char <= 'z'):
                messagebox.showerror('Помилка', 'Підтримуються лише символи англійської абетки')
                return
        for char in key:
            if not ('A' <= char <= 'Z' or 'a' <= char <= 'z'):
                messagebox.showerror('Помилка', 'Підтримуються лише символи англійської абетки')
                return

        matrix = self.create_playfair_matrix(key)
        self.display_matrix(matrix)

        prepared_text = self.prepare_text(text)
        encrypted_text = ""

        for i in range(0, len(prepared_text), 2):
            a, b = prepared_text[i], prepared_text[i + 1]
```

```
            encrypted_pair, arrows = self.encrypt_pair(a, b, matrix)
            encrypted_text += encrypted_pair
            self.result_label.config(text=f"Зашифрований текст:\n{encrypted_text}")
            root.update()
            pos_a = tuple(np.where(matrix == a))
            pos_b = tuple(np.where(matrix == b))
            self.display_matrix(matrix, highlight_positions=[pos_a, pos_b],
arrows=arrows)

    root = tk.Tk()
    app = PlayfairCipherApp(root)
    root.mainloop()



def cezar():
    window = tk.Tk()
    window.title("Алгоритм Цезаря")
    window.geometry("1330x530")

    # Фрейми
    text_frame = ttk.Frame(window)
    text_frame.pack(pady=20)

    button_frame = ttk.Frame(window)
    button_frame.pack(pady=20)

    result_frame = ttk.Frame(window)
    result_frame.pack(pady=20)
```

```
    # Елементи інтерфейсу
    text_label = tk.Label(text_frame, text="Текст:\n", wraplength=800)
    text_label.pack(pady=10)
    text_entry = tk.Entry(text_frame, width=60)
    text_entry.pack(pady=10)

    shift_label = tk.Label(text_frame, text="Здвиг\n", wraplength=800)
    shift_label.pack(pady=10)
    key_slider = tk.Scale(text_frame, from_=0, to=25, orient=tk.HORIZONTAL,
length=400)
    key_slider.pack()

    result_label = tk.Label(result_frame, text="Зашифрований текст:\n",
wraplength=800)
    result_label.pack(pady=10)

    alphabet = string.ascii_lowercase
    cell_size = 50

    alphabet_canvas = Canvas(result_frame, width=len(alphabet) * cell_size,
height=cell_size)
    alphabet_canvas.pack(pady=20)

    for i, char in enumerate(alphabet):
        x = i * cell_size
        y = 0
        alphabet_canvas.create_rectangle(x, y, x + cell_size, y + cell_size, fill="white",
outline="black")
        alphabet_canvas.create_text(x + cell_size // 2, y + cell_size // 2, text=char,
```

```
font=("Arial", 16))

    def encrypt():
        text = text_entry.get().strip().lower()
        text = text.replace(' ', '')
        print(text)
        key = int(key_slider.get())

        if not text or not key:
            messagebox.showerror("Помилка", "Введіть текст та ключ")
            return

        for char in text:
            if not ('A' <= char <= 'Z' or 'a' <= char <= 'z'):
                messagebox.showerror('Помилка', 'Підтримуються лише символи
англійської абетки')
                return

        encrypted_text = ""
        animation_delay = 2

        for i, char in enumerate(text):
            if char in alphabet:
                new_index = (alphabet.index(char) + key) % len(alphabet)
                new_char = alphabet[new_index]

                x = alphabet.index(char) * cell_size
                y = 0
                alphabet_canvas.itemconfig(alphabet_canvas.find_withtag(f"{char}"),
```

```
fill="yellow")

            arrow_x = x + cell_size // 2
            arrow_y = y + cell_size // 2
            end_x = alphabet.index(new_char) * cell_size + cell_size // 2
            end_y = y + cell_size // 2
            arrow = alphabet_canvas.create_line(arrow_x, arrow_y, end_x, end_y,
arrow=tk.LAST,
                                                width=2, fill="red")

            for _ in range(int(animation_delay * 100)):
                arrow_x += (end_x - arrow_x) / 100
                alphabet_canvas.coords(arrow, arrow_x, arrow_y, end_x, end_y)
                window.update()
                time.sleep(animation_delay / 100)

            alphabet_canvas.delete(arrow)

            encrypted_text += new_char

            result_label.config(text=f"Зашифрований текст:\n{encrypted_text}")

            window.update()
            time.sleep(animation_delay)

    encrypt_button = ttk.Button(button_frame, text="Шифрувати",
command=encrypt)
    encrypt_button.pack(pady=10)
    window.mainloop()
```

```
def aes():
    matrix_size = 4

    def hex_to_matrix(hex_array):
        hex_matrix = []
        for i in range(16):
            if i % matrix_size == 0:
                hex_matrix.append([hex_array[i]])
            else:
                hex_matrix[int(i / matrix_size)].append(hex_array[i])
        return hex_matrix

    def int_to_hex_string(number):
        if type(number) == int:
            return "%0.2X" % number
        else:
            return str(number)

    def text_to_hex(text):
        hex_array = []
        for char in text:
            hex_array.append(ord(char))
        hex_array = hex_array + [0x01] * (16 - len(hex_array))
        return hex_array

    def key_expansion(key_matrix):
        round_keys = [key_matrix]
```

```
    round_key = key_matrix[:]
    for r in range(0, 10):
        new_key = copy.deepcopy(round_key)
        last = round_key[3]
        new_key[3] = round_last(new_key[3], r)
        new_key[0] = xor_matrix(new_key[0], new_key[3])
        new_key[1] = xor_matrix(new_key[1], new_key[0])
        new_key[2] = xor_matrix(new_key[2], new_key[1])
        new_key[3] = xor_matrix(last, new_key[2])
        round_key = new_key
        round_keys += [new_key]
    return round_keys

def round_last(round_key, r):
    last_column = round_key[1:] + round_key[:1]
    for column in range(matrix_size):
        last_column[column] = sbox[last_column[column]]
    last_column[0] = last_column[0] ^ round_constants[r]
    return last_column

def xor_matrix(first, second):
    first = copy.deepcopy(first)
    for i in range(4):
        first[i] = first[i] ^ second[i]
    return first

def xor_matrices(first, second):
    first = copy.deepcopy(first)
    for i in range(len(first)):
```

```
            first[i] = xor_matrix(first[i], second[i])
        return first

    def confusion(merged_matrix):
        merged_matrix = copy.deepcopy(merged_matrix)
        for i in range(matrix_size):
            for j in range(matrix_size):
                merged_matrix[i][j] = sbox[merged_matrix[i][j]]
        return merged_matrix

    def diffusion(merged_matrix):
        merged_matrix = copy.deepcopy(merged_matrix)
        merged_matrix = [list(element) for element in
list(zip(*reversed(merged_matrix)))]
        for i in range(matrix_size):
            merged_matrix[i] = merged_matrix[i][-i:] + merged_matrix[i][:-i]
        merged_matrix = [list(element)[::-1] for element in
list(zip(*reversed(merged_matrix)))][::-1]
        return merged_matrix

    def mix_columns(merged_matrix):
        merged_matrix = copy.deepcopy(merged_matrix)
        magic = lambda x: (((x << 1) ^ 0x1B) & 0xFF) if (x & 0x80) else (x << 1)
        for i in range(4):
            a = merged_matrix[i]
            t = a[0] ^ a[1] ^ a[2] ^ a[3]
            u = a[0]
            a[0] ^= t ^ magic(a[0] ^ a[1])
            a[1] ^= t ^ magic(a[1] ^ a[2])
```

```
            a[2] ^= t ^ magic(a[2] ^ a[3])
            a[3] ^= t ^ magic(a[3] ^ u)
        return merged_matrix

    def visualize_matrix(canvas, matrix, row, col, title=""):
        cell_width = 50
        cell_height = 50
        padding = 10
        x_start = col * (cell_width * matrix_size + padding)
        y_start = row * (cell_height + padding)
        if title:
            canvas.create_text(x_start, y_start, text=title, anchor='nw',
font=font.Font(size=14, weight='bold'))
        for i in range(matrix_size):
            for j in range(matrix_size):
                x = x_start + j * cell_width
                y = y_start + i * cell_height + 20
                canvas.create_rectangle(x, y, x + cell_width, y + cell_height,
outline="black")
                canvas.create_text(x + cell_width / 2, y + cell_height / 2,
text=int_to_hex_string(matrix[i][j]),
                                   font=font.Font(size=12))

    def encrypt():
        text = input_text.get().strip().lower()
        text = text.replace(' ', '')
        for char in text:
            if not ('A' <= char <= 'Z' or 'a' <= char <= 'z'):
                messagebox.showerror('Помилка', 'Підтримуються лише символи
```

```
англійської абетки')
            return
        passphrase = input_key.get().strip().lower()
        passphrase = passphrase.replace(' ', '')
        for char in passphrase:
            if not ('A' <= char <= 'Z' or 'a' <= char <= 'z'):
                messagebox.showerror('Помилка', 'Підтримуються лише символи
англійської абетки')
                return
        key_matrix = hex_to_matrix(text_to_hex(passphrase))
        text_matrix = hex_to_matrix(text_to_hex(text))
        round_keys = key_expansion(key_matrix)
        merged_matrix = xor_matrices(key_matrix, text_matrix)
        canvas.create_text(600, 20, text='Round 0 Initial round',
font=font.Font(size=24))
        visualize_matrix(canvas, text_matrix, 1, 0, "Text Matrix")
        visualize_matrix(canvas, key_matrix, 1, 2, "Key Matrix")
        visualize_matrix(canvas, merged_matrix, 1, 4, "")
        canvas.create_text(300, 180, text='+', font=font.Font(size=24))
        canvas.create_text(730, 180, text='=', font=font.Font(size=24))
        for r in range(10):
            canvas.update()
            sleep(2)
            canvas.delete('all')
            canvas.create_text(600, 20, text='Round ' + str(r + 1) + ' Sub bytes',
font=font.Font(size=24))
            confused_matrix = confusion(copy.deepcopy(merged_matrix))
            visualize_matrix(canvas, merged_matrix, 1, 1, '')
            canvas.create_text(530, 180, text='=>', font=font.Font(size=24))
```

```
            visualize_matrix(canvas, confused_matrix, 1, 3, '')
            canvas.update()
            sleep(2)
            canvas.delete('all')
            canvas.create_text(600, 20, text='Round ' + str(r + 1) + ' Shift rows',
font=font.Font(size=24))
            diffused_matrix = diffusion(copy.deepcopy(confused_matrix))
            visualize_matrix(canvas, confused_matrix, 1, 1, '')
            canvas.create_text(530, 180, text='=>', font=font.Font(size=24))
            visualize_matrix(canvas, diffused_matrix, 1, 3, '')
            canvas.update()
            sleep(2)
            canvas.delete('all')
            if r == 9:
                canvas.delete('all')
                canvas.create_text(600, 20, text='Round ' + str(r + 1),
font=font.Font(size=24))
                merged_matrix = xor_matrices(diffused_matrix, round_keys[r + 1])
                visualize_matrix(canvas, merged_matrix, 1, 3, "Зашифрований текст")
            else:
                canvas.create_text(600, 20, text='Round ' + str(r + 1) + ' Mix columns',
font=font.Font(size=24))
                visualize_matrix(canvas, diffused_matrix, 1, 1, '')
                mixed_matrix = mix_columns(diffused_matrix)
                canvas.create_text(530, 180, text='=>', font=font.Font(size=24))
                visualize_matrix(canvas, mixed_matrix, 1, 3, '')
                canvas.update()
                sleep(2)
                canvas.delete('all')
```

```
            canvas.create_text(600, 20, text='Round ' + str(r + 1) + ' Add round key',
font=font.Font(size=24))
            merged_matrix = xor_matrices(mixed_matrix, round_keys[r + 1])
            visualize_matrix(canvas, mixed_matrix, 1, 0, "Mixed matrix")
            visualize_matrix(canvas, round_keys[r + 1], 1, 2, "Round key")
            visualize_matrix(canvas, merged_matrix, 1, 4, "")
            canvas.create_text(300, 180, text='+', font=font.Font(size=24))
            canvas.create_text(730, 180, text='=', font=font.Font(size=24))

    root = tk.Tk()
    root.title("AES")

    frame = tk.Frame(root)
    frame.pack(pady=10)

    input_text_label = tk.Label(frame, text="Input Text:")
    input_text_label.grid(row=0, column=0)
    input_text = tk.Entry(frame, width=50)
    input_text.grid(row=0, column=1)

    input_key_label = tk.Label(frame, text="Input Key:")
    input_key_label.grid(row=1, column=0)
    input_key = tk.Entry(frame, width=50)
    input_key.grid(row=1, column=1)

    encrypt_button = tk.Button(frame, text="Encrypt", command=encrypt)
    encrypt_button.grid(row=2, columnspan=2, pady=10)

    canvas = tk.Canvas(root, width=1200, height=800, bg="white")
```

```
    canvas.pack()

    root.mainloop()


def on_close_main():
    main.destroy()


main = tk.Tk()
main.protocol('WM_DELETE_WINDOW', on_close_main)
main.title("Алгоритми шифрування")
main.geometry("400x300")

button_font = ("Times New Roman", 16)

button_cezar = tk.Button(main, text="Алгоритм Цезаря", command=cezar,
font=button_font, width=20, height=2)
button_cezar.pack(pady=10)

button_pleifer = tk.Button(main, text="Алгоритм Плейфера", command=pleifer,
font=button_font, width=20, height=2)
button_pleifer.pack(pady=10)

button_aes = tk.Button(main, text="AES", command=aes, font=button_font,
width=20, height=2)
button_aes.pack(pady=10)

main.mainloop()
```