Міністерство освіти і науки України
«Київський національний університет імені Тараса Шевченка»

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ПОЯСНЮВАЛЬНА ЗАПИСКА

кваліфікаційної роботи магістра

галузь знань *12 Інформаційні технології*
(шифр і назва галузі знань)

спеціальність *125 Кібербезпека*
(код і назва спеціальності)

освітній ступень *магістр*
(назва освітньої програми)

освітньо-наукова програма *Кібербезпека*

на тему: «Методи криптографічного захисту інформації на об’єктах інформаційної діяльності»

Виконавець: студент ІІ курсу, групи КБм-21

_____ **Ярослав АРХИПОВ**
(підпис) (прізвище ім’я по-батькові)

	Ім’я, ПРІЗВИЩЕ	**Підпис**
Науковий керівник	Сергій ТОЛЮПА	
Нормоконтроль	Юрій ЩЕБЛАНІН	

Київ 2023

Міністерство освіти і науки України
«Київський національний університет імені Тараса Шевченка»

Факультет інформаційних технологій
Кафедра кібербезпеки та захисту інформації

ЗАТВЕРДЖЕНО:
В.О. завідувач кафедри
кібербезпеки
та захисту інформації
_____ Сергій ТОЛЮПА
«24» жовтня 2022 р.

ЗАВДАННЯ
на виконання кваліфікаційної роботи

спеціальності *125 Кібербезпека*
(код і назва спеціальності)

освітній ступень *магістр*

Здобувача(ки) КБм-21 (група) Архипову Ярославу Анатолійовичу (прізвище ім'я по-батькові)

Тема кваліфікаційної роботи «Методи криптографічного захисту інформації на об'єктах інформаційної діяльності»

1. ПІДСТАВИ ДЛЯ ПРОВЕДЕННЯ РОБОТИ

Рішення засідання кафедри кібербезпеки та захисту інформації факультету інформаційних технологій протокол № 3 від 20.10.2022

2. МЕТА ТА ВИХІДНІ ДАНІ ДЛЯ ПРОВЕДЕННЯ РОБІТ

Об'єкт досліджень	Методи та підходи до застосування криптографічних алгоритмів для захисту інформації.
Предмет досліджень	Процес захисту інформації на ОІД шляхом криптографічних методів захисту.
Мета	Порівняння методів криптографічного захисту інформації та реалізація криптографічного алгоритму для захисту інформації на об'єкті інформаційної діяльності.

Вихідні дані для проведення роботи Криптографічні методи захисту інформації.

3. ОЧІКУВАНІ НАУКОВІ РЕЗУЛЬТАТИ

Наукова новизна Визначено ефективність модифікації алгоритмів RSA. Запропоновано використання модифікації CRT RSA.

Практична цінність Досягнення ефективності криптографічного алгоритму RSA, шляхом використання його модифікація для зменшення часу дешифрування у веб застосунку на ОІД.

4. ВИМОГИ ДО РЕЗУЛЬТАТІВ ВИКОНАННЯ РОБОТИ

Робота виконана у повному обсязі відповідно до теми.

5. ЕТАПИ ВИКОНАННЯ РОБОТИ

Найменування етапів робіт	Строки виконання робіт (початок-кінець)
Розробка плану для досягнення мети роботи	24.10.2022 – 09.01.2023
Аналіз літературних джерел	09.01.2023 – 25.02.2023
Програмна реалізація веб застосунку	25.02.2023 – 21.04.2023
Оформлення і друк пояснювальної записки	01.05.2023 – 19.05.2023

6. РЕАЛІЗАЦІЯ РЕЗУЛЬТАТІВ ТА ЕФЕКТИВНІСТЬ

Економічний ефект Зниження збитків з причини криптографічного захисту

Соціальний ефект Підвищення ефективності криптографічного захисту на об’єкті інформаційної діяльності.

7. ДОДАТКОВІ ВИМОГИ

Завдання видав _____ Сергій ТОЛЮПА
(підпис) (Ім’я, ПРІЗВИЩЕ)

Завдання прийняв до виконання _____ Ярослав АРХИПОВ
(підпис) (Ім’я, ПРІЗИЩЕ)

Дата видачі завдання: 24.10.2022 р.
Термін подання кваліфікаційної роботи до ЕК 19.05.2023 р.

УДК. 004.432.16

РЕФЕРАТ

Пояснювальна записка до кваліфікаційної роботи «Методи криптографічного захисту інформації на об’єктах інформаційної діяльності» складається зі списку скорочень, вступу, основної частини, що містить 3 розділи, висновків, списку літератури та джерел. Загальний обсяг роботи – 124 сторінки. Робота містить 36 рисунків та 4 таблиці. Список використаних джерел включає 34 джерела.

Метою даної роботи є порівняння методів криптографічного захисту інформації та реалізація криптографічного алгоритму для захисту інформації на об’єкті інформаційної діяльності.

У роботі проаналізовано існуюча література щодо загроз безпеці інформації на ОІД, виконаний аналіз методів криптографічного захисту інформації, проведено дослідження модифікацій алгоритму RSA, розроблено веб застосунок.

Досягнення ефективності криптографічного алгоритму RSA, шляхом використання його модифікація для зменшення часу дешифрування у веб застосунку на ОІД.

Ключові слова: Криптографія, шифрування, дешифрування, RSA, ключ, захист інформації, AES, DES.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

ОІД	–	Об’єкт інформаційної діяльності
ТЗПІ	–	Технічні засоби передачі та обробки інформації
АС	–	Автоматизовані системи
ДТЗПІ	–	Допоміжні технічні засоби передачі і обробки інформації
ІзОД	–	Інформації з обмеженим доступом
КСЗІ	–	Комплексна система захисту інформації
СЗІ	–	Системи захисту інформації
ТКВІ	–	Технічний канал витоку інформації
ПЕМВ	–	Побічні електромагнітні випромінювання
ВЧ	–	Високочастотні
НЧ	–	Низькочастотні
ТЗІ	–	Технічний захист інформації
CIA	–	Confidentiality, integrity, availability
ENISA	–	European Network and Information Security Agency
IoT	–	Internet of things
AES	–	Advanced Encryption Standard
TLS	–	Transport Layer Security
SSL	–	Secure Socket Layers
RSA	–	Rivest–Shamir–Adleman
NIST	–	National Institute of Standards and Technology
IPsec	–	Internet Protocol security
SHA	–	Secure Hash Algorithm
HMAC	–	Hash-based Message Authentication Code
FIPS	–	Federal Information Processing Standards
DES	–	Data Encryption Standard
IBM	–	International Business Machines
ANSI	–	American National Standards Institute
DEA	–	Data Encryption Algorithm
XOR	–	eXclusive OR
EFF	–	Electronic Frontier Foundation
ECB	–	Electronic Code Book
CBC	–	Cipher Block Chaining
CFB	–	Cipher Feedback
OFB	–	Output Feedback

TDEA	–	Triple Data Encryption Algorithm
HTTPS	–	Hyper Text Transfer Protocol Secure
SSH	–	Secure Shell
CRT	–	Chinese Reminder Theorem
API	–	Application Programming Interface
HTTP	–	HyperText Transfer Protocol
REST	–	Representational State Transfer
URI	–	Uniform Resource Identifier
JSON	–	JavaScript Object Notation
XML	–	eXtensible Markup Language
UML	–	Unified Modeling Language

ЗМІСТ

ВСТУП

Криптографічний захист є важливим елементом захисту інформації на об'єктах інформаційної діяльності, таких як банки, державні установи, промислові підприємства, інформаційні системи тощо. Основні причини, чому криптографічний захист є актуальним на об'єктах інформаційної діяльності, такі:

– Захист конфіденційної інформації.

– Захист від кібератак.

– Захист від шпигунства.

– Захист від вірусів та зловмисного програмного забезпечення.

– Відповідність законодавству.

Усі ці причини демонструють важливість застосування криптографічного захисту на об'єктах інформаційної діяльності. Використання криптографії дозволяє зберегти конфіденційність, цілісність та доступність інформації, що забезпечує надійний захист від кіберзагроз та забезпечує дотримання вимог законодавства.

Жорсткі вимоги сучасності та збільшення кількості атак на інформаційні системи потребують проведення дослідження та пошуку найкращих рішень на основі застосування різних методів криптографічного шифрування для захисту інформації на об’єктах інформаційної діяльності.

Метою даної роботи є порівняння методів криптографічного захисту інформації та реалізація криптографічного алгоритму для захисту інформації на об’єкті інформаційної діяльності.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

– Розглянути та виконати порівняльний аналіз існуючих криптографічних методів та підходів.

– Протестувати та виконати аналіз різних варіацій криптографічних алгоритмів.

– Створити веб-додаток для шифрування та дешифрування повідомлень.

– Виконати валідацію розробленого програмного забезпечення веб-додатку.

Об'єктом дослідження в даній роботі є методи та підходи до застосування криптографічних алгоритмів для захисту інформації.

Предметом дослідження в даній роботі є криптографічний алгоритм для захисту інформації від загроз. Перевірка роботи алгоритму та його оцінка.

Практична цінність - досягнення ефективності криптографічного алгоритму RSA, шляхом використання його модифікація для зменшення часу дешифрування у веб застосунку на ОІД.

Методи дослідження у кваліфікаційній роботі:

- аналіз літератури;
- аналіз наявних рішень;
- проведення власних досліджень;
- аналіз документації;
- порівняння криптографічних алгоритмів.

РОЗДІЛ 1
ОБ’ЄКТ ІНФОРМАЦІЙНОЇ ДІЯЛЬНОСТІ ТА АНАЛІЗ ЗАГРОЗ БЕЗПЕЦІ ІНФОРМАЦІЇ НА ОБ’ЄКТІ ІНФОРМАЦІЙНОЇ ДІЯЛЬНОСТІ

1.1 Об’єкт інформаційної діяльності

В останні роки в Україні соціальні та економічні зміни створили сприятливі умови для широкого впровадження новітніх інформаційних технологій. Створення та використання перспективних інформаційних, телекомунікаційних систем зв'язку, автоматизованих систем обробки та передачі інформації як у державному, так і у недержавному секторах.

Однак створення індустрії обробки інформації, яка надає об'єктивні передумови для значного підвищення ефективності життєдіяльності людства, породжує складні та великомасштабні проблеми. Одна з таких проблем - організація захисту інформації, яка циркулює та обробляється в системах на ОІД.

Забезпечення безпеки інформації полягає у встановленні та підтримці системи заходів - як технічних, так і нетехнічних. Які дозволяють запобігти або ускладнити можливість реалізації загроз, а також знизити можливі збитки. Отже, захист інформації має на меті забезпечення безпеки інформації в цілому, збереження заданих властивостей інформації та захист персональних даних, що обробляються. Для досягнення цієї мети створюється комплексна система захисту інформації, основними об'єктами якої є:

- інформаційні ресурси, що містять відомості, які належать або до державної таємниці, або до конфіденційної інформації;
- технічні засоби передачі та обробки інформації, а саме:

а) системи та засоби інформатизації (обчислювальна техніка, інформаційно – обчислювальні комплекси, мережі та системи);

б) програмні засоби (операційні системи, системи управління базами даних та інше загальносистемне та прикладне програмне забезпечення);

в) системи зв’язку; АС управління; ТЗПІ (звукозапис, звукопідсилення, звукосупроводження, переговорні та телевізійні пристрої; пристрої тиражування і виготовлення документів та інші технічні засоби обробки графічної, алфавітно – цифрової та текстової інформації), їх інформативні фізичні поля;

- допоміжні технічні засоби передачі і обробки інформації, тобто технічні засоби, які не належать до ТЗПІ, але розташовані в приміщеннях, де обробляється інформація [1].

При організації захисту інформації, ТЗПІ слід розглядати як комплексну систему, що складається з різноманітних компонентів, таких як стаціонарне обладнання, периферійні пристрої, з'єднувальні лінії, розподільні та комутаційні пристрої, а також системи електроживлення та заземлення.

Технічні засоби, що використовуються для обробки інформації з обмеженим доступом, разом з приміщенням, де вони розташовані, утворюють об’єкт інформаційної діяльності. У нашій країні, ОІД являє собою інженерно – технічну споруду, приміщення, де здійснюється адміністративна, виробнича та інша діяльність, пов'язана з інформацією, що підлягає захисту від витоку технічними каналами.

Об’єкт інформаційної діяльності - це складна система, яка складається з безлічі локальних підсистем (інформаційних вузлів), оснащених програмно-апаратними засобами, необхідними для забезпечення інформаційних технологій. Крім того, об’єкт інформаційної діяльності включає безліч засобів зв'язку та взаємодії між цими підсистемами, щоб надати користувачам широкий спектр інформаційних послуг.

Об'єкт інформаційної діяльності складається з двох основних компонентів - інформаційної системи та телекомунікаційної системи, які працюють взаємодіючи між собою і утворюють єдину інформаційно-телекомунікаційну систему. Технічні та програмні засоби використовуються для обробки інформації та забезпечення її передачі шляхом випромінювання, приймання або передавання у формі звукових, візуальних або інших сигналів.

Об'єктом захисту є інформаційно-телекомунікаційна система, яка складається з наступних елементів:

- Діяльність або завдання, що призводять до створення інформації з обмеженим доступом.
- Носії інформації різних типів, таких як персонал, технічні пристрої, магнітні носії, фізичні поля та середовища.
- Канали зв'язку між носіями інформації, що забезпечують функціонування системи.
- Протоколи збереження, обробки і обміну інформацією між носіями.
- Функціональні параметри системи та її елементів, які визначають характеристики режиму збереження, обробки і передачі інформації з обмеженим доступом і її зміну в часі.
- Фізичний простір, у якому знаходяться носії та канали зв'язку, такі як споруди, транспорт, територія тощо.

Всі елементи інформаційно-телекомунікаційної системи містять важливу інформацію та потребують захисту від зовнішніх та внутрішніх загроз. Захист інформації в автоматизованих системах передбачає дії, спрямовані на забезпечення безпеки інформації та системи в цілому, з метою запобігання або ускладнення можливості реалізації загроз та зменшення можливих збитків, які можуть виникнути внаслідок реалізації цих загроз.

Автоматизовану систему можна розділити на три класи [2]. Ця система є досить складною, тому її доцільно поділити на окремі елементи, щоб спростити проектування. Як результат, кожен елемент буде автономним та незалежним від інших, а у комплексі вони утворять цілісну систему, яка потребує відповідну КСЗІ.

Розділення автоматизованої системи на окремі елементи дозволяє службі захисту інформації:

- Ефективно реагувати на певні види загроз, які властиві для кожного окремого елемента;
- Швидко впроваджувати систему захисту інформації в нові елементи, що додалися;
- Спрощувати процедуру контролю системи захисту інформації в цілому.

З поступовим збільшенням кількості елементів та складності структури захисту, АС стає більш складною, оскільки потребує використання не одного типу захисних функцій у всіх елементах, а їх комбінацію. Це робить побудову системи захисту інформації невід’ємною складовою розробки ефективної комплексної системи захисту інформації. КСЗІ представляє собою сукупність організаційних, інженерних, програмно-апаратних заходів, що забезпечують захист інформації в автоматизованих системах.

З цього видно, що, наприклад, просте екранування приміщення при проектуванні КСЗІ недостатнє, оскільки цей метод забезпечує лише захист інформації від витоку по радіоканалу. В загальному випадку термін "комплексність" відноситься до рішень, які об'єднують два або більше завдань з різних плоскостей в рамках єдиної концепції (цільова комплексність), використовуючи для вирішення однієї задачі різнопланові інструментальні засоби (інструментальна комплексність) або комбінуючи обидва підходи (загальна комплексність). Цільова комплексність передбачає побудову системи інформаційної безпеки таким чином, щоб:

- Захистити інформацію, інформаційні ресурси та системи від зовнішніх і внутрішніх загроз.
- Захиститися від негативного інформаційного впливу.

Інструментальна комплексність включає інтеграцію всіх видів і напрямків інформаційної безпеки з метою досягнення поставлених цілей. Сьогоднішні комплексні системи захисту інформації повинні мати структурну, функціональну та тимчасову комплексність. Структурна комплексність забезпечує належний рівень захисту всіх елементів системи обробки інформації. Функціональна комплексність передбачає застосування методів захисту для всіх функцій системи обробки інформації. Тимчасова комплексність означає постійне здійснення заходів щодо захисту інформації на всіх етапах життєвого циклу об'єкта обробки інформації, включаючи процес безпосередньої обробки.

1.2 Аналіз загроз безпеці інформації, яка обробляється на об’єкті інформаційної діяльності

Інформація, що обробляється на ОІД, може бути перехоплена за допомогою технічних каналів. Технічний канал витоку інформації включає об’єкт розвідки, технічний засіб розвідки та фізичне середовище, через яке поширюється інформаційний сигнал. ТКВІ є методом отримання інформації про об’єкт розвідки з використанням технічних засобів розвідки, при цьому формат подачі інформації може бути будь-яким [3].

Сигнали являються матеріальними носіями інформації, їх природа може бути електричною, електромагнітною, акустичною, оптичною та іншими. Залежно від цієї природи, сигнали можуть поширюватись в різних фізичних середовищах, таких як газові, рідинні, тверді, такі як повітря, будівельні конструкції, струмопровідні кабелі та дроти, заземлені ділянки ґрунту та інші.

У технічних засобах принцип утворення потенційно небезпечних сигналів залежить від наявності фізичних перетворювачів з різними функціями, які працюють на основі різних фізичних принципів. Щоб виявити можливі неконтрольовані прояви фізичних полів, які можуть стати каналом витоку, необхідно мати розуміння принципу дії кожного перетворювача (рис. 1.1) [4].

У системах зв’язку, АС управління та обробки інформації є багато первинних перетворювачів, які відрізняються фізичною природою:

– фотоелектричні;
– термоелектричні;
– п’єзоелектричні;
– акустоелектричні.

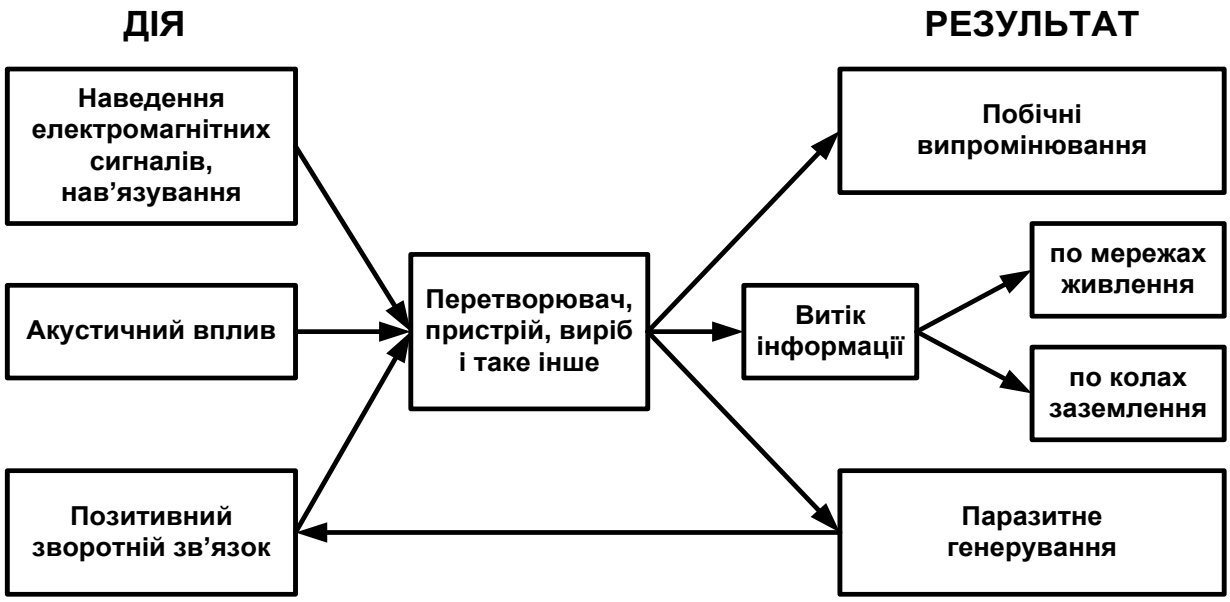

Рисунок 1.1 – Варіанти утворення небезпечних сигналів

З урахуванням фізичної природи появи інформаційних сигналів, середовища їх поширення та методів їх перехоплення, ТКВІ можна класифікувати на наступні види:

- електромагнітні, електричні та параметричні, що використовуються для передачі телекомунікаційної інформації;
- повітряні (прямі акустичні), вібраційні (віброакустичні), акустоелектричні, оптико-електронні та параметричні.

Електромагнітні ТКВІ можуть бути перехоплені через:

- побічні електромагнітні випромінювання елементів ТЗПІ;
- ПЕМВ на частотах роботи високочастотних генераторів у ТЗПІ та ДТЗПІ;
- ПЕМВ на частотах само збурення низькочастотних підсилювачів ТЗПІ.
- Засобами радіо та радіотехнічної розвідки, розміщених за межами контрольованої зони, можуть бути перехоплені ПЕМВ ТЗПІ.

Електричні ТКВІ використовуються для збору наступних типів сигналів:

- наведених сигналів ПЕМВ ТЗПІ зі з'єднувальних ліній ДТЗПІ та сторонніх провідників;
- інформаційних сигналів з ліній електроживлення ТЗПІ;
- інформаційних сигналів з мереж заземлення ТЗПІ та ДТЗПІ;
- інформації, яка може бути перехоплена за допомогою електронних пристроїв, розміщених в ТЗПІ.

Закладні пристрої або апаратні закладки - це мініпередавачі, які можуть бути використані для перехоплення інформації, оскільки їх сигнали модулюються інформаційними сигналами.

Параметричні ТКВІ, з іншого боку, створюються високочастотним опроміненням ТЗПІ. Для перехоплення інформації, що передається такими каналами, потрібні високочастотні генератори з антенами, які мають вузьку діаграму спрямованості, а також спеціальні радіоприймальні пристрої [1].

Акустичні ТКВІ використовують повітря як середовище передачі сигналу. Для перехоплення акустичних сигналів використовують мікрофони. Отримані сигнали можуть бути записані на пристрої звукозапису або передані за допомогою передавачів на приймальні пункти.

Для перехоплення акустичної (мовної) інформації доступні різноманітні методи, серед яких можна виділити:

- портативні диктофони та дротові мікрофони, що використовуються для таємного звукозапису;
- спрямовані мікрофони, які здатні збирати акустичні сигнали з конкретної точки;
- акустичні радіо закладки, що передають інформацію по радіоканалу;
- акустичні мережеві закладки, які передають сигнали по лініях силових мереж електроживлення;
- акустичні інфрачервоні закладки, які передають інформацію по оптичному каналу в інфрачервоному діапазоні;
- акустичні телефоні закладки, що передають інформацію по телефонних лініях зв’язку на високих частотах;

В вібраційних (віброакустичних) ТКВІ, акустичні сигнали поширюються через різні конструкційні елементи споруд та будівель, такі як стіни, перекриття, підлога, а також тверді тіла, такі як труби водопостачання та каналізації.

Для перехоплення звукових коливань, що поширюються через вібраційні ТКВІ, використовуються розвідувальні технічні засоби з контактними мікрофонами,

зокрема електронні та радіочастотні стетоскопи. Радіо стетоскопи дозволяють передавати зібрану інформацію по радіоканалу [1].

Оптико-електричний (лазерний) ТКВІ виникає, коли тонкі відбиваючі поверхні (такі як скляні вікна, картини, дзеркала тощо) вібрують у акустичному полі, опромінені лазерним променем. Складні лазерні акустичні локаційні системи використовуються для перехоплення мовної інформації через такий канал, іноді їх називають лазерними мікрофонами.

Параметричні ТКВІ виникають в результаті високочастотного опромінення приміщення, в якому знаходяться напівактивні закладні пристрої або технічні засоби з елементами, параметри яких змінюються відповідно до змін акустичного (мовного) сигналу. Для перехоплення інформації за допомогою такого каналу потрібні спеціальний передавач з направленим променем і приймач.

Аналіз потенційних загроз інформації, яка циркулює на ОІД, вказує на те, що загроза в традиційному розумінні є можливою небезпекою (яка може бути потенційною або існуючою в реальному світі) будь-яких дій чи бездіяльності, які спрямовані на об'єкт захисту (інформаційні ресурси) і призводять до збитку для власника або користувача, проявляючись як ризик спотворення або втрати інформації.

Серед потенційних небезпек для безпеки інформації можна виділити три основні категорії загроз (рис. 1.2):

- Загрози конфіденційності, які включають крадіжку або копіювання інформації та засобів її оброблення, а також випадкову або несанкціоновану втрату інформації та засобів її оброблення.
- Загрози доступності, такі як блокування інформації, знищення інформації та засобів її оброблення.
- Загрози цілісності, які включають спотворення (модифікацію) інформації, заперечення автентичності інформації та нав'язування фальшивої інформації.

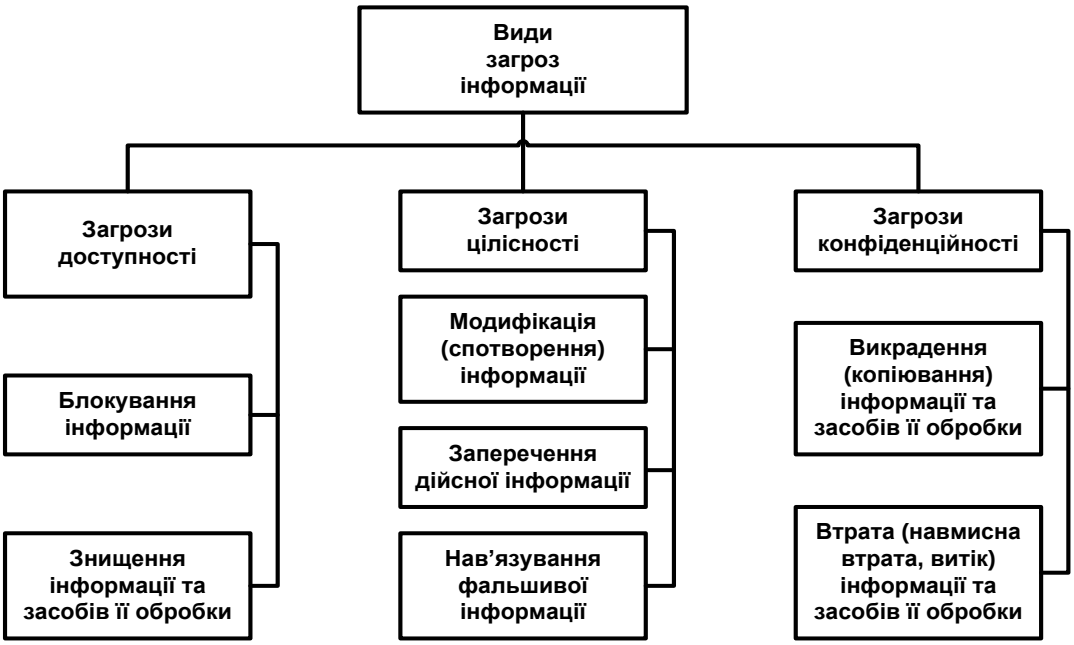

Рисунок 1.2 – Класифікація загроз безпеці інформації, що циркулює на ОІД

ІзОД може бути піддана загрозам безпеки через джерела, які можуть бути як суб'єктивними (особи), так і об'єктивними проявами. Джерела загроз можуть бути як внутрішні, тобто від суб'єктів, які мають прямий доступ до інформації, так і зовнішні, що виникають з-за меж ІзОД.

Можна розподілити всі джерела загроз безпеці інформації на три групи (рис. 1.3):

- Джерела загроз, що обумовлені діями суб'єкта (антропогенні джерела загроз).
- Джерела загроз, що обумовлені стихійними явищами.
- Джерела загроз, що обумовлені технічними засобами (техногенні джерела загроз).

Рисунок 1.3 – Класифікація джерел загроз безпеці інформації

Антропогенні джерела загроз інформаційній безпеці виникають в результаті дій суб'єктів, які можна класифікувати як випадкові або умисні злочини. Ця група джерел загроз є найбільш розповсюдженою та цікавою для організації захисту, оскільки дії суб'єкта можна зрозуміти, передбачити та вжити необхідні заходи для запобігання таким загрозам. Методи захисту від антропогенних джерел загроз керовані.

Друга категорія джерел загроз об'єднує непереборні обставини, які мають об'єктивний та абсолютний характер і впливають на всіх. У законодавстві та практиці угод ці обставини відомі як "непереборна сила" і можуть включати стихійні лиха та інші подібні обставини, які неможливо передбачити або запобігти, або можуть бути передбачені, але не можуть бути запобіжені на сучасному рівні знань та можливостей людини [5, 6].

У третій групі джерел загроз знаходяться фактори, що виникають через технократичну діяльність людини та розвиток цивілізації. Незважаючи на те, що людина відповідає за їх виникнення, наслідки цих дій можуть виходити з-під її контролю та мати самостійний вплив.

Для передачі інформації за допомогою будь-якого технічного каналу, включаючи канали витоку, необхідно використовувати три головні складові елементи (рис. 1.4):

- джерело сигналу;
- середовище поширення;
- приймач.

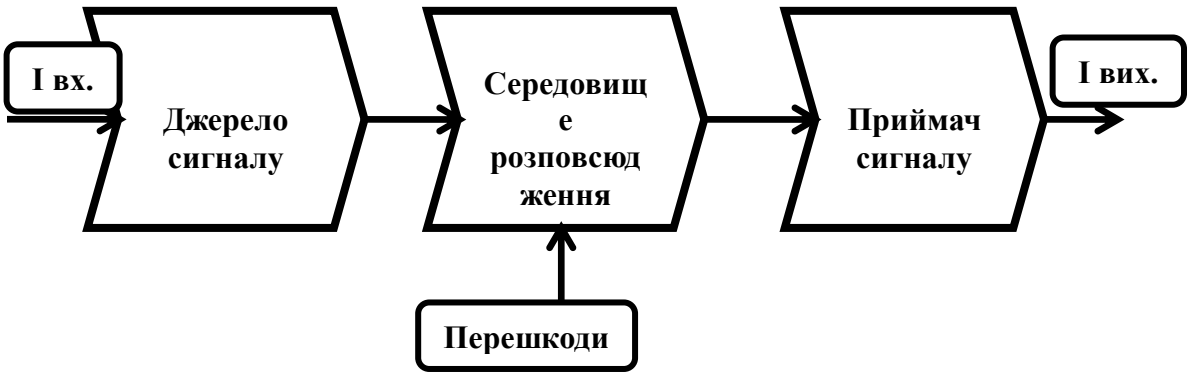

Рисунок 1.4 – Структура каналу передавання інформації

Інформація, що надходить на вхід каналу, передається у вигляді первинного сигналу. Цей сигнал може бути носієм інформації від джерела або з попереднього каналу. Можливими джерелами сигналу є:

– об’єкт спостереження, що відбиває електромагнітні та акустичні хвилі;

– об’єкт спостереження, що відбиває власні (теплові) електромагнітні хвилі в оптичному та радіодіапазонах;

– передавач функціонального каналу зв’язку;

– закладний пристрій;

– джерело небезпечного сигналу;

– джерело акустичних хвиль, модульованих інформацією.

Сигнали можуть походити як від джерел функціональних каналів зв’язку, так і від небезпечних джерел. Небезпечні сигнали можуть містити конфіденційну інформацію, яка з'являється випадково для джерела інформації та не контролюється ним.

Середовище, в якому передається інформаційний носій, є частинкою простору, що має свій власний набір параметрів, що визначають умови руху цього носія. Щоб

повноцінно описати середовище поширення, необхідно враховувати ключові параметри, такі як:

– фізичні перепони для суб’єктів і матеріальних тіл;

– міра ослаблення (або пропускання) сигналу на одиницю довжини;

– частотна характеристика (нерівномірність ослаблення частотних складових спектра сигналу);

– вид і потужність завад для сигналу.

Приймач – виконує функції, обернені функціям передавача. Описані процеси включають вибір носія, який містить необхідну інформацію для одержувача, підсилення прийнятого сигналу до потрібного рівня для зняття інформації, зняття інформації з носія (демодуляція та декодування) і перетворення інформації в форму сигналу, яку можна зрозуміти користувачем (будь то людина або технічний пристрій) [2].

Якщо інформацію отримує людина, то необхідно, щоб вихідний сигнал був поданий у формі, зрозумілій для людини, якщо ж інформація призначена для технічних засобів, то формат подання має бути зрозумілим для цього пристрою. На рисунку 1.5 зображена класифікація ТКВІ.

На даному рисунку ТКВІ класифікуються за їх фізичною природою носія інформації. Ця класифікація включає такі типи як оптичні, радіоелектронні, акустичні та матеріально-речовинні.

Проведений аналіз показує, що існує велика кількість можливих ТКВІ, що становлять загрозу безпеці інформації. Досвід досліджень та заходів, спрямованих на розв'язання проблеми технічного захисту інформації, свідчить про те, що засоби зловживання для здобуття неправомірного доступу до інформації, що циркулює на ОІД, розвиваються швидше, ніж засоби захисту.

Розвиток ТЗІ визначається потребою вчасного вжиття заходів, які відповідають масштабам загрози для інформації та ґрунтуються на принципах правової демократичної держави, що враховують права суб'єктів інформаційних відносин на доступ до інформації та її захист.

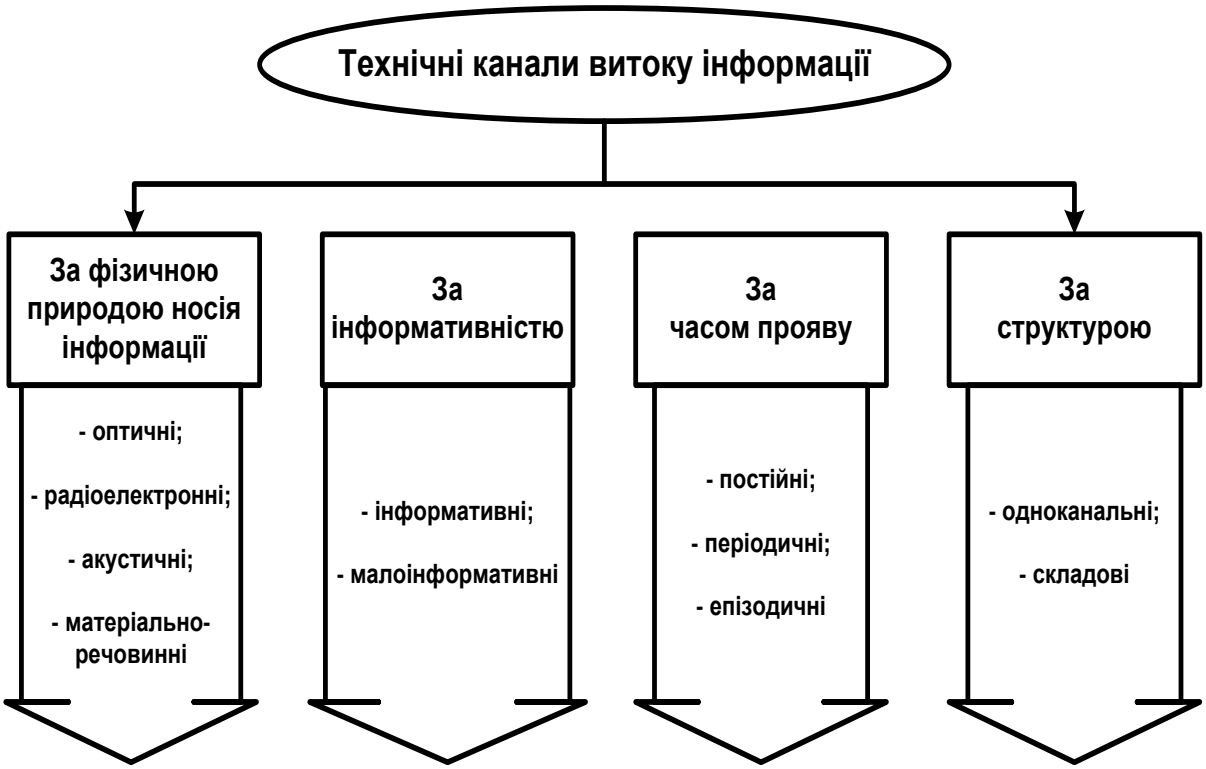

Рисунок 1.5 – Класифікація технічних каналів витоку інформації.

1.3 Криптографічні методи захисту інформації на об’єкті інформаційної діяльності

Криптографічні методи захисту використовуються для захисту інформації на об'єктах інформаційної діяльності від несанкціонованого доступу, крадіжки, зміни, втрати та інших загроз безпеці інформації.

Основні криптографічні методи захисту включають:

- Шифрування даних. Шифрування дозволяє захистити інформацію від несанкціонованого доступу. Для цього використовуються алгоритми шифрування, які перетворюють звичайний текст в незрозумілий для сторонніх осіб, які намагаються отримати доступ до інформації.
- Електронний підпис. Електронний підпис забезпечує автентифікацію документів та повідомлень, що передаються по мережі. Він гарантує, що документи не були змінені після створення та що вони були підписані власником документа.

- Цифровий автентифікатор. Цифровий автентифікатор (також відомий як «токен») - це електронний пристрій, який генерує одноразові паролі, що використовуються для автентифікації користувачів у мережі.
- Ключові сертифікати. Ключові сертифікати використовуються для підтвердження автентичності користувача, який надає доступ до інформації. Вони містять інформацію про ключі шифрування, що використовуються для захисту інформації.
- Захист мережевого трафіку. Криптографічні протоколи, такі як SSL / TLS, використовуються для захисту мережевого трафіку від несанкціонованого доступу. Ці протоколи шифрують дані, що передаються між комп'ютерами, щоб забезпечити конфіденційність та цілісність даних.
- Хешування даних. Хешування даних - це процес створення унікального числа, яке може служити цифровим підписом для даних. Цей процес забезпечує цілісність даних, тому що будь-яка зміна даних призведе до зміни хеш-значення.
- Ключове управління. Ключове управління - це процес створення, зберігання та керування ключами шифрування. Цей процес дозволяє забезпечити безпеку ключів шифрування та їх використання, щоб зменшити ризик несанкціонованого доступу до інформації.
- Автентифікація користувачів. Автентифікація користувачів - це процес перевірки ідентифікаційних даних користувача перед наданням доступу до захищеної інформації. Для цього можуть використовуватись різноманітні методи, такі як паролі, біометричні дані, смарт-карти тощо.
- Ідентифікація та управління доступом. Ідентифікація та управління доступом - це процес контролю доступу до інформації та ресурсів. Цей процес забезпечує безпеку та конфіденційність даних, обмежує доступ до інформації тільки авторизованим користувачам та визначає рівень доступу кожного користувача.
- Шифрування файлів та комунікацій. Шифрування файлів та комунікацій - це процес перетворення даних в такий формат, що доступ до них можливий тільки з використанням спеціального ключа. Цей процес забезпечує конфіденційність та захист даних від несанкціонованого доступу.

- Фізична безпека. Фізична безпека - це заходи безпеки, які стосуються захисту фізичного обладнання та інфраструктури від вторгнення, вандалізму та інших фізичних загроз. Цей процес забезпечує захист від потенційних загроз, таких як крадіжки, вандалізм та природні катастрофи.

Ці методи можуть застосовуватись окремо або в поєднанні між собою для максимального захисту інформації на об'єктах інформаційної діяльності. При використанні криптографічних методів захисту важливо враховувати вимоги до безпеки та конфіденційності інформації, що обробляється на об'єктах інформаційної діяльності, а також використовувати сучасні алгоритми шифрування та методи автентифікації.

Висновки за розділом 1

В першому розділі було проаналізовано об’єкт інформаційної діяльності в цілому та загрози безпеці інформації, яка обробляється на об’єкті інформаційної діяльності, був наведений детальний аналіз наступних питань:

- класифікація загроз безпеці інформації, що циркулює на ОІД;
- класифікація джерел загроз інформації.

На основі проаналізованого матеріалу можна зробити висновок, що застосування криптографічних методів захисту допомагає забезпечити безпеку та конфіденційність інформації на об'єктах інформаційної діяльності, зменшити ризики несанкціонованого доступу та витоку даних. Для ефективного застосування криптографічних методів необхідно враховувати специфіку кожного конкретного об'єкта інформаційної діяльності, оцінювати ризики та вибирати відповідні методи захисту. Крім того, необхідно забезпечувати регулярне оновлення та моніторинг криптографічних методів захисту, оцінювати їх ефективність та вчасно внести зміни при необхідності.

РОЗДІЛ 2
МЕТОДИ КРИПТОГРАФІЧНОГО ЗАХИСТУ ІНФОРМАЦІЇ

2.1 Принципи криптографічного захисту інформації

Криптографія є важливою галуззю захисту інформації в сучасному світі. Це наука про забезпечення конфіденційності, цілісності та доступності інформації, використовуючи математичні алгоритми та протоколи. Методи криптографічного захисту інформації на об'єкті інформаційної діяльності можуть бути класифіковані в залежності від двох аспектів: застосування та форми захисту. Щодо застосування, методи криптографічного захисту інформації на об'єкті інформаційної діяльності можуть бути розділені на три групи: захист даних в спокійному стані (у статичному вигляді), захист даних під час передачі (у динамічному вигляді) та захист даних під час обробки. Щодо форм захисту, методи криптографічного захисту інформації на об'єкті інформаційної діяльності можуть бути розділені на дві групи: симетричні та асиметричні.

Симетричні методи використовують один ключ для шифрування та розшифрування повідомлення (рис. 2.1).

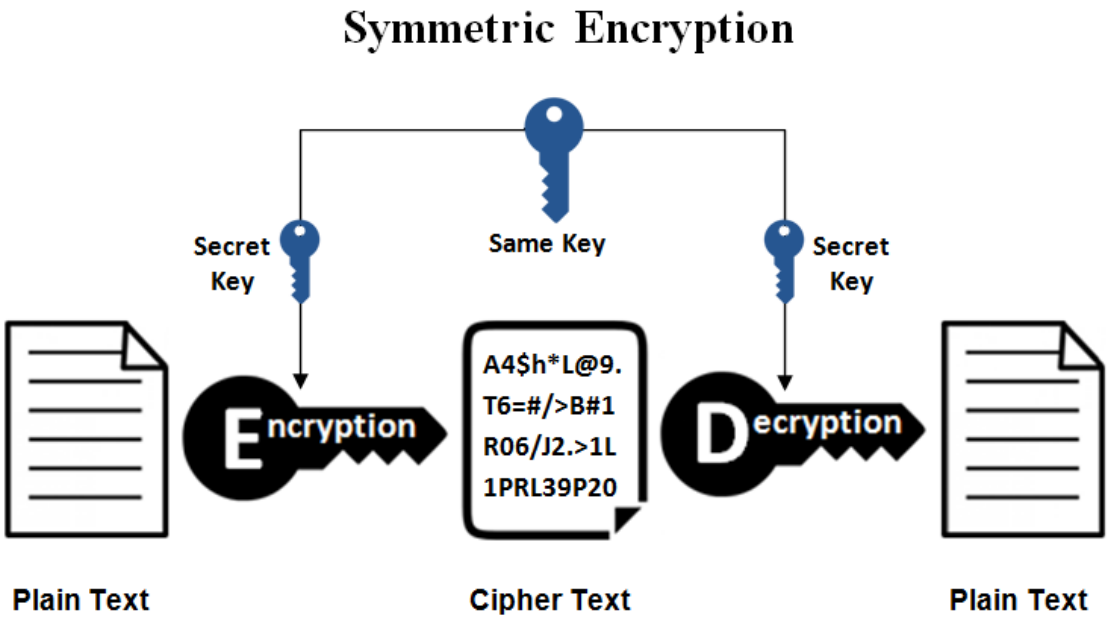

Рисунок 2.1 - Загальне представлення симетричних методів шифрування

Ці методи мають наступні переваги: швидкість шифрування та розшифрування, надійність захисту від злому та відсутність необхідності великого обсягу обчислювальних ресурсів. Однак симетричні методи мають певні недоліки, зокрема, потребують безпечного обміну ключами між відправником та одержувачем.

Асиметричні методи використовують два ключі - публічний та приватний - для шифрування та розшифрування повідомлення (рис. 2.2).

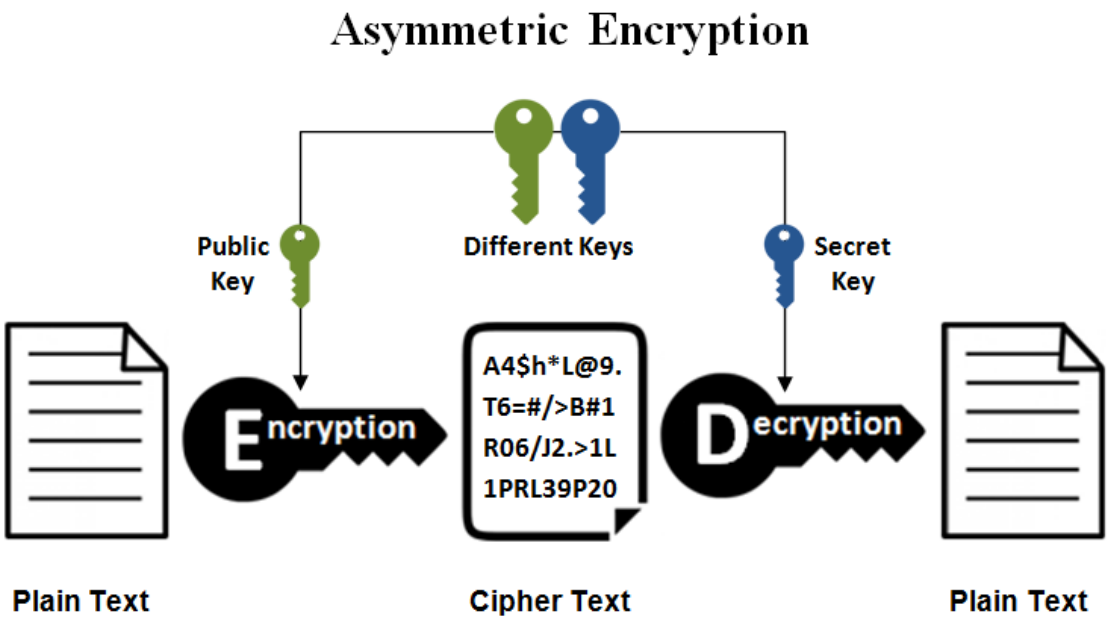

Рисунок 2.2 - Загальне представлення асиметричних методів шифрування

Публічний ключ доступний всім, тоді як приватний ключ безпечно зберігається тільки у власника ключа. Ці методи мають наступні переваги: відсутність необхідності в безпечному обміні ключами, можливість підписування повідомлень для перевірки автентичності та неможливість злому без знання приватного ключа. Однак асиметричні методи мають певні недоліки, зокрема, повільність шифрування та розшифрування, складність розподілу публічного ключа та можливість атак з використанням атаки на середовище.

Серед найпопулярніших методів криптографічного захисту інформації на об'єкті інформаційної діяльності можна відзначити такі:

- Алгоритми шифрування DES, AES, Blowfish та інших симетричних алгоритмів.

- Алгоритми шифрування RSA, ElGamal та інших асиметричних алгоритмів.
- Протоколи SSL, TLS та інші протоколи захисту передачі даних через мережу.
- Електронні підписи та цифрові сертифікати для перевірки автентичності повідомлень та користувачів.
- Криптографічні пристрої, такі як криптографічні токени, які забезпечують безпечне зберігання ключів.

Загалом, застосування методів криптографічного захисту інформації є важливим кроком для забезпечення безпеки та конфіденційності даних на об'єктах інформаційної діяльності. Вибір конкретного методу залежить від потреб захисту, а також від технічних характеристик та можливостей об'єкта застосування.

Захист даних часто описується як процес на основі трьох компонентів: конфіденційність, цілісність та доступність (рис. 2.3).

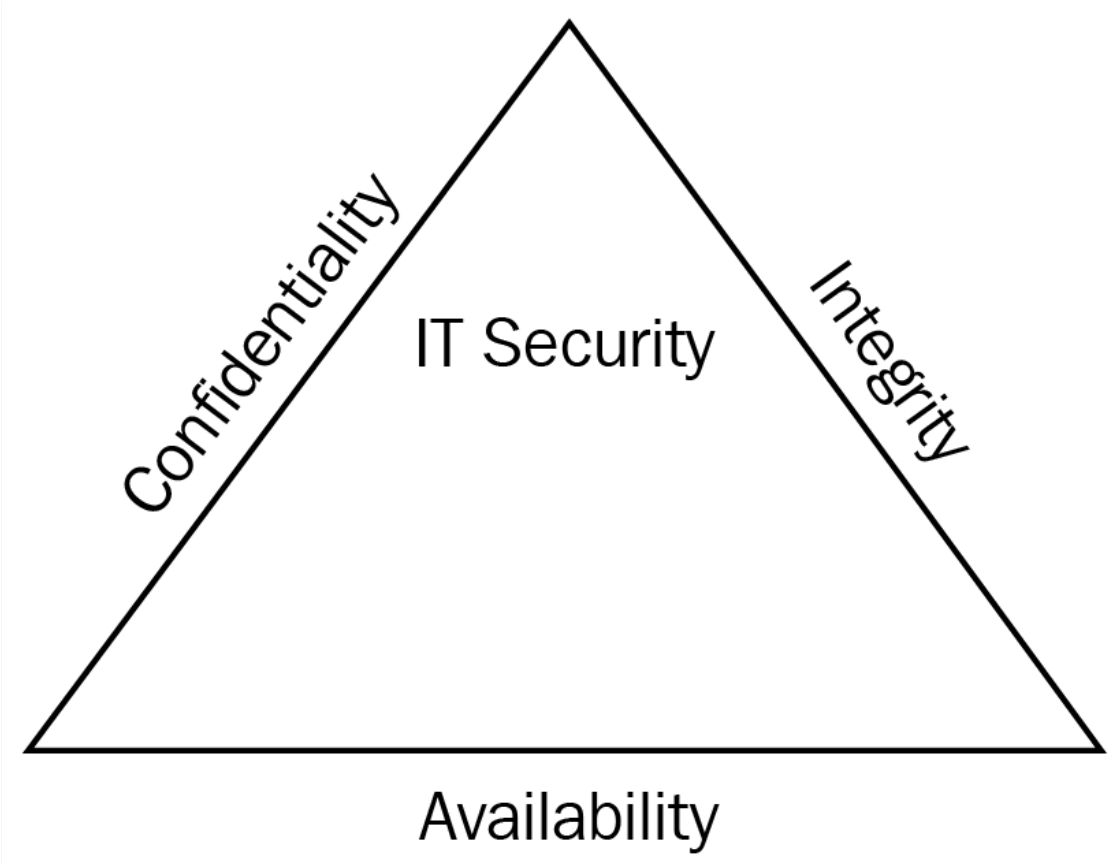

Рисунок 2.3 - CIA трикутник

Незважаючи на важливість цих трьох вимог, проблема безпеки інформації має більш широкі розміри [7].

ENISA розробила та представила рекомендовані криптографічні заходи для забезпечення високого рівня кібербезпеки в Європейському Союзі. Документ містить детальні настанови щодо застосування криптографічних методів для захисту інформації, зокрема щодо застосування симетричних та асиметричних алгоритмів шифрування, хеш-функцій, цифрових підписів та інших заходів.

Recommended Cryptographic Measures також включає рекомендації щодо використання криптографічних протоколів для забезпечення безпеки мережевих комунікацій, таких як SSL/TLS та IPsec. Документ містить поради та рекомендації щодо належного використання криптографічних заходів, а також вказує на можливі ризики та вразливості при неправильному застосуванні криптографічних методів.

Recommended Cryptographic Measures є важливим джерелом інформації для організацій та підприємств, що займаються кібербезпекою та допоможе забезпечити високий рівень захисту інформації від кібератак [8].

2.1.1. Конфіденційність

У криптографії конфіденційність означає, що інформація повинна бути захищена від незаконного доступу і розголошення. Для досягнення цього принципу застосовуються криптографічні алгоритми, які дозволяють зашифрувати інформацію, щоб лише авторизованим користувачам було доступне її розшифрування.

Криптографічні алгоритми можуть використовувати симетричні ключі або асиметричні ключі.

Крім того, існує також поняття цифрових підписів, які дозволяють підтвердити автентичність документа чи повідомлення та відстежувати його автора. Для створення цифрових підписів використовуються криптографічні алгоритми, що дозволяють забезпечити непідробність та цілісність даних.

Отже, в криптографії конфіденційність є важливим принципом, і його досягнення забезпечує захист від несанкціонованого доступу та збереження конфіденційної інформації в таємниці.

2.1.2. Цілісність

Цілісність в криптографії - це принцип захисту інформації, що вимагає забезпечення не зміни даних під час їх передачі або зберігання. Це означає, що інформація повинна зберігатися та передаватися у такому вигляді, в якому була створена, без будь-яких несанкціонованих змін.

Для забезпечення цілісності даних використовуються криптографічні методи, зокрема хеш-функції. Хеш-функції приймають на вхід повідомлення будь-якої довжини і створюють вихідний хеш-код фіксованої довжини. Хеш-код вважається унікальним для кожного повідомлення, тому якщо він змінюється, то це означає, що дані були змінені.

Хеш-функції використовуються для забезпечення цілісності даних, наприклад, під час передачі повідомлень через мережу. Користувач може відправити хеш-код разом з повідомленням, а отримувач може перевірити цілісність повідомлення, порівнявши отриманий хеш-код з хеш-кодом, який був відправлений разом з повідомленням.

Також використовуються цифрові підписи, які не лише забезпечують автентичність даних, а й гарантують їх цілісність. Цифровий підпис складається з хеш-коду даних та криптографічного підпису автора, і він може бути перевірений, щоб переконатися, що дані не були змінені та що підписав їх саме автор.

Отже, забезпечення цілісності даних є важливим принципом криптографії, і його досягнення.

2.1.3. Доступність

Дані повинні бути легко доступні в будь-який момент, коли вони потрібні. Якщо дані присутні у базі даних у зашифрованому і автентифікованому вигляді, але з якихось причин їх не можна експортувати з бази даних та використовувати, то порушується властивість наявності. Немає криптографічної техніки, яку можна застосувати до самої інформації, щоб зробити її більш доступною. Однак криптографія може використовуватися для захисту методи доступу до даних (контроль доступу).

Доступність інформації не повинна бути забута при застосуванні криптографічних методів. Наприклад, якщо доступ до даних забезпечується за допомогою сильних методів шифрування, то це може ускладнити процес доступу до даних тим, хто має право на нього. Таким чином, при розробці криптографічних методів необхідно враховувати і доступність інформації, щоб забезпечити, що тільки відповідні особи мають доступ до неї.

2.1.4 Автентифікація

Автентифікація в криптографії - це процес перевірки того, що об'єкт (особа, пристрій, програма тощо) дійсно той, за кого він себе видає. Цей принцип забезпечує, що тільки відповідні особи мають доступ до конфіденційної інформації або можуть виконувати певні дії.

Для забезпечення автентифікації використовуються різні методи, такі як паролі, біометричні методи (наприклад, сканер відбитків пальців), криптографічні ключі тощо.

Один з найбільш поширених методів автентифікації в криптографії - це використання цифрових підписів. Цифровий підпис - це криптографічний метод, який забезпечує ідентифікацію автора повідомлення та його цілісність. Для створення цифрового підпису використовується публічний ключ, який забезпечує можливість перевірки підпису за допомогою відповідного приватного ключа. Цей метод

забезпечує, що повідомлення було створене саме тим, хто видається автором, і що воно не було змінене під час передачі.

Іншим методом автентифікації є використання криптографічних хеш-функцій. Хеш-функція - це алгоритм, який перетворює будь-який вхідний набір даних фіксованої довжини. Хеш-функція може використовуватися для створення "відбитка" повідомлення, який можна використовувати для перевірки того, що повідомлення не було змінене під час передачі.

Автентифікація є важливим принципом в критографії, оскільки забезпечує надійне визначення осіб та пристроїв, які мають доступ до конфіденційної інформації. Без автентифікації будь-який користувач може ввійти в систему з допомогою простого пароля або використати інші методи атаки для здобуття доступу до конфіденційної інформації.

У криптографії також існують інші методи автентифікації, такі як криптографічні токени та смарт-карти, які забезпечують більш високий рівень безпеки, але вимагають спеціального обладнання та програмного забезпечення для їх використання.

Всі ці методи автентифікації використовуються в криптографії з метою забезпечення високого рівня безпеки інформації, що зберігається або передається через мережу. Комбінація різних методів автентифікації дозволяє створювати надійні системи захисту даних, які забезпечують надійність та безпеку передачі та зберігання конфіденційної інформації.

2.1.5 Невідмовність

Невідмовність - це принцип криптографічного захисту, який забезпечує можливість пізнішої перевірки того, що певна дія або транзакція дійсно відбулася, та відповідність цієї дії або транзакції до визначеного часу та користувача.

Цей принцип забезпечується за допомогою цифрових підписів та інших методів автентифікації, які дозволяють перевіряти автентичність повідомлень та транзакцій, а також їхню відповідність визначеному часу та користувачу.

Невідмовність є важливим принципом в електронній комунікації та електронній комерції, оскільки вона дозволяє довести автентичність та цілісність електронних документів та транзакцій, що здійснюються в електронному вигляді.

Цей принцип також застосовується в криптографії для захисту від відмови у здійсненні транзакцій, коли одна зі сторін заперечує, що здійснювала певну дію або транзакцію.

Застосування принципу невідмовності дозволяє створювати надійні системи електронного документообігу та електронної комерції, які забезпечують високий рівень захисту від зловмисних атак та зловживань.

2.1.6 Надійність

Надійність в криптографії - це здатність криптографічних систем та протоколів захисту інформації досягати своєї мети - забезпечення конфіденційності, цілісності, доступності та невідмовності - в умовах можливих зловживань, атак та інших загроз безпеці.

Щоб забезпечити надійність криптографічних систем, застосовуються різні методи та підходи. Загальний рівень надійності криптографічних систем може бути визначений за допомогою різних метрик, таких як: складність алгоритму, довжина ключа, кількість можливих ключів, витривалість, реалізація, підтримка, відкритість.

Загалом, надійність криптографічних систем є дуже важливою, оскільки вона забезпечує захист конфіденційної інформації від несанкціонованого доступу, викрадення та розголошення. При проектуванні та використанні криптографічних систем необхідно враховувати всі метрики надійності та використовувати тільки перевірені протоколи та алгоритми. Також важливо вести моніторинг криптографічної системи та вчасно виявляти та виправляти вразливості. Для забезпечення високого рівня надійності криптографічної системи, рекомендується використовувати комбінацію різних криптографічних методів, таких як шифрування та підписи.

Отже, надійність криптографічної системи є важливим чинником захисту конфіденційної інформації від несанкціонованого доступу та забезпечення її цілісності та достовірності. Для досягнення максимальної надійності, необхідно враховувати різні метрики, використовувати комбінацію криптографічних методів та застосовувати правильні процедури використання та управління ключами.

2.1.7 Масштабованість

Масштабованість в криптографії - це здатність криптографічних систем функціонувати ефективно та надійно при збільшенні розміру та складності системи, кількості користувачів, обсягу передаваної інформації та інших факторів.

З ростом кількості користувачів і обсягу передаваної інформації, необхідно збільшувати швидкість обробки даних та масштабність системи. Криптографічні системи повинні бути готові до такого масштабування, щоб можливість їх використання була реалістичною для великих мереж.

2.1.8 Ефективність

Ефективність в криптографії відноситься до швидкості і складності алгоритмів шифрування та розшифрування, а також до міцності криптографічних протоколів.

Швидкість шифрування та розшифрування є важливим фактором в ефективності криптографії, оскільки чим більші обчислювальні ресурси потрібні для виконання цих операцій, тим більше часу та ресурсів потрібно для захисту даних. Крім того, деякі застосування, такі як мобільні пристрої або IoT-пристрої, мають обмежені обчислювальні ресурси, тому шифрування та розшифрування повинні бути досить швидкими, щоб бути ефективними.

Складність алгоритмів шифрування та розшифрування відноситься до складності обчислень, які потрібні для зламування шифру. Якщо алгоритм шифрування має дуже високу складність зламування, то він вважається більш ефективним.

Міцність криптографічних протоколів відноситься до їхньої здатності захищати дані від зламування. Якщо протокол є міцним і відповідно захищає дані, то він вважається більш ефективним.

Загалом, ефективність в криптографії залежить від балансу між складністю алгоритмів шифрування та розшифрування, швидкістю їх виконання та міцністю криптографічних протоколів.

2.1.9 Гнучкість

Гнучкість в криптографії відноситься до здатності криптографічних систем працювати в різних умовах та виконувати різні функції. Це може включати зміну параметрів алгоритмів, використання різних криптографічних протоколів для різних застосувань та забезпечення зв'язку між різними пристроями та системами.

Наприклад, гнучкість може бути важливою в IoT-системах, де пристрої можуть мати різні обмеження щодо обчислювальних ресурсів, енергоспоживання та пропускної здатності. Криптографічна система повинна бути здатна працювати з різними пристроями та виконувати різні функції залежно від їхніх можливостей та обмежень.

Гнучкість також може включати здатність до апгрейдів та змін до криптографічних систем без необхідності перекладу всієї системи заново. Це може бути важливо для забезпечення безпеки та забезпечення сумісності з новими стандартами та протоколами.

Загалом, гнучкість в криптографії дозволяє системам бути більш адаптивними до змін у вимогах та умовах застосування, що робить їх більш ефективними та придатними для різних застосувань.

2.1.10 Керованість

Керованість в криптографії відноситься до здатності адміністратора керувати криптографічними системами та параметрами їхньої роботи. Це означає, що

адміністратор може встановлювати та змінювати параметри криптографічних алгоритмів, які використовуються в системі, а також встановлювати правила для доступу до зашифрованих даних та ключів шифрування.

Керованість важлива для забезпечення безпеки в криптографічних системах, оскільки дозволяє адміністратору контролювати доступ до конфіденційної інформації та ключів шифрування. Адміністратор може встановлювати правила доступу до цих даних та контролювати, як вони використовуються, щоб запобігти несанкціонованому доступу до конфіденційної інформації.

Крім того, керованість дозволяє адміністраторам забезпечувати сумісність криптографічних систем з іншими системами та протоколами, що допомагає забезпечувати інтероперабельність та ефективність роботи в різних середовищах.

Узагальнюючи, керованість є важливим аспектом криптографії, оскільки дозволяє забезпечувати безпеку та ефективність роботи криптографічних систем в різних умовах та середовищах [8].

2.2 Вибір показників для оцінки криптографічного захисту інформації

Оцінка криптографічного захисту інформації передбачає аналіз різних факторів, які впливають на безпеку передачі та збереження інформації. Основними критеріями оцінки криптографічного захисту інформації є наступні.

2.2.1 Довжина ключа

Довжина ключа — вона впливає на складність атаки на криптографічний алгоритм. Чим довший ключ, тим важче розшифрувати зашифровану інформацію. Довжина ключа в криптографії вимірюється у бітах і вказує на кількість бітів, які використовуються для шифрування і розшифрування повідомлення. Довжина ключа впливає на складність криптоаналізу алгоритму шифрування.

Наприклад, у криптосистемі RSA довжина ключа визначається числом бітів, які використовуються для генерації ключа. Для забезпечення безпеки повідомлення, в

даний час рекомендовано використовувати ключі довжиною 2048 бітів або більше. Довжина ключа може варіюватися в залежності від потреб користувача та рівня безпеки, який він бажає забезпечити.

Інші криптографічні алгоритми, такі як AES, також вимірюють довжину ключа у бітах. Наприклад, для AES-128 ключ складається з 128 біт, а для AES-256 - з 256 біт. Отже, довжина ключа визначається кількістю бітів, які використовуються для захисту інформації від несанкціонованого доступу або розкриття. Чим довший ключ, тим більшою буде його криптографічна стійкість і складність криптоаналізу.

2.2.2 Складність алгоритму

Складність алгоритму - це кількість ресурсів (часу, пам'яті, обчислювальної потужності тощо), необхідних для виконання алгоритму. У криптографії, складність алгоритму має прямий вплив на стійкість криптосистеми.

В криптографії, складність алгоритму оцінюється відповідно до кількості операцій, необхідних для взлому криптографічного алгоритму. Чим більше операцій потрібно виконати, тим складнішим є алгоритм. У багатьох випадках, складність алгоритму пов'язана з довжиною ключа - чим довший ключ, тим складніше зламати криптосистему.

Загальноприйнятим показником складності алгоритму є його обчислювальна складність. Цей показник вимірюється в асимптотичних обчислювальних операціях. Наприклад, якщо алгоритм має складність $O(n^2)$, то час виконання алгоритму збільшується квадратично при збільшенні розміру вхідних даних n.

У криптографії, складність алгоритму повинна бути достатньою, щоб забезпечити стійкість криптосистеми до атак. Наприклад, складність алгоритму RSA залежить від довжини ключа, тому для забезпечення безпеки вимагається використання ключів довжиною 2048 біт або більше. Інші криптографічні алгоритми, такі як AES, також мають високу обчислювальну складність, що забезпечує їхню стійкість до атак.

При виборі криптографічного алгоритму, необхідно звертати увагу на складність його атак, а також на складність самого алгоритму. Атаки можуть бути різними - від атак на перебір ключів до атак на знайдення слабкої точки в самому алгоритмі. Для забезпечення стійкості криптосистеми, необхідно, щоб складність атак була великою, а складність алгоритму була достатньою для захисту від цих атак.

Один з показників складності алгоритму - його часова складність. Це кількість операцій, які потрібно виконати для обробки даних. Часова складність може бути виміряна у таких одиницях, як кількість операцій, кількість бітів, кількість байтів тощо. Іншим показником складності алгоритму є його пам'ятева складність - кількість пам'яті, необхідної для збереження даних та результатів обробки.

Ще один показник складності алгоритму - його обчислювальна складність, яка вимірюється в асимптотичних обчислювальних операціях. Це дозволяє порівнювати складність різних алгоритмів з точки зору часових ресурсів та пам’яті, які вони вимагають.

Також важливо враховувати можливості сучасних обчислювальних систем, оскільки розвиток технологій дозволяє зламувати криптографічні алгоритми, які раніше були вважалися безпечними. Тому необхідно регулярно оновлювати алгоритми та довжину ключів, щоб забезпечити стійкість криптосистеми до нових атак.

Отже, складність алгоритму є важливим показником в криптографії, оскільки складність алгоритму безпосередньо пов'язана з його стійкістю, то важливо обирати такі алгоритми, які забезпечують необхідну стійкість за рахунок достатньої складності. Наприклад, алгоритм RSA забезпечує стійкість за рахунок складності факторизації великих чисел, але також має високу часову складність для обробки великих блоків даних.

2.2.3 Стійкість до криптоаналітичних атак

Стійкість до криптоаналітичних атак - це інший важливий критерій, який використовується для оцінки криптографічних алгоритмів. Криптоаналітична атака -

це спроба розкрити або зламати криптографічний алгоритм шляхом використання певних аналітичних технік. Отже, стійкість до криптоаналітичних атак вимірюється в термінах того, наскільки важко зламати криптографічний алгоритм за допомогою таких технік.

Існує багато різних типів криптоаналітичних атак, включаючи методи, які базуються на відомостях про відкритий текст, статистичні методи, лінійні та диференціальні криптоаналітичні атаки, а також багато інших.

Щоб забезпечити стійкість до криптоаналітичних атак, криптографічні алгоритми повинні мати високу ентропію та дифузію. Ентропія вимірює кількість невизначеності в системі, тоді як дифузія вимірює те, наскільки швидко зміни в одному біті повинні розповсюджуватися на всю систему.

Для того, щоб забезпечити стійкість до криптоаналітичних атак, також важливо, щоб криптографічні алгоритми були розроблені з використанням надійних математичних основ, таких як теорія чисел та теорія складності обчислень. Крім того, алгоритми повинні бути перевірені на безпеку та відповідність стандартам, щоб забезпечити їхню надійність.

Одним з найбільш відомих прикладів криптографітичних алгоритмів, що володіють стійкістю до криптоаналітичних атак, є алгоритм AES. Цей алгоритм використовується у багатьох системах шифрування даних.

Один з інших відомих криптографічних алгоритмів, що володіють стійкістю до криптоаналітичних атак, - це алгоритм RSA. Він використовується для шифрування та підписування даних і ґрунтується на складності факторизації великих простих чисел.

Інші криптографічні алгоритми, що володіють стійкістю до криптоаналітичних атак, включають алгоритми Діффі-Хеллмана, Еліптичні криві, шифр Хілла та багато інших

Основним методом для оцінки стійкості криптографічних алгоритмів до криптоаналітичних атак є проведення тестів на безпеку та вразливість до атак. Для цього використовуються різні техніки, такі як аналіз стійкості до лінійних та

диференціальних криптоаналітичних атак, аналіз стійкості до атак з відкритим текстом, аналіз стійкості до атак з вибірковою фазою та інші.

Крім того, існують такі методи, як криптоаналітичні атаки з використанням перебору, аналіз побічних каналів, аналіз часових характеристик та інші. Усі ці методи дозволяють виявити можливі вразливості алгоритмів та розробити ефективні методи їх захисту.

Найбільш розповсюдженим тестом на безпеку та стійкість криптографічних алгоритмів є тест NIST, який використовується для оцінки якості та стійкості різних криптографічних алгоритмів. Цей тест включає в себе набір вимог, що визначаються за допомогою математичних алгоритмів та криптографічних протоколів.

Наприклад, для того, щоб алгоритм вважався стійким до криптоаналітичних атак, він повинен відповідати таким вимогам:

1. Ключ повинен бути досить довгим, щоб забезпечити стійкість до перебірних атак
2. Алгоритм повинен бути стійким до лінійних та диференціальних криптоаналітичних атак.
3. Алгоритм повинен бути стійким до атак з відкритим текстом.
4. Алгоритм повинен бути стійким до атак з вибірковою фазою.
5. Алгоритм повинен бути стійким до атак з використанням перебору та аналізу побічних каналів.

Зважаючи на те, що криптографічні алгоритми є складними математичними конструкціями, їх стійкість до криптоаналітичних атак може бути підірвана розвитком нових математичних методів та алгоритмів. Тому важливо постійно вдосконалювати криптографічні алгоритми та оновлювати їх для забезпечення максимальної стійкості. Також необхідно враховувати, що під час реалізації криптографічного алгоритму можуть виникнути помилки та вразливості, тому слід використовувати найкращі практики безпеки програмного забезпечення та регулярно перевіряти алгоритми на стійкість до нових типів атак.

У сучасному світі деякі криптографічні алгоритми, такі як RSA та AES, досить стійкі до більшості криптоаналітичних атак і застосовуються в широкому спектрі

криптографічних застосувань. Однак, існують нові вектори атак, такі як квантові атаки, які можуть досить швидко зламувати деякі з цих алгоритмів, що робить необхідним постійний розвиток та вдосконалення криптографічних алгоритмів.

У заключенні можна сказати, що стійкість до криптоаналітичних атак є одним з найважливіших критеріїв для оцінки криптографічних алгоритмів. Для забезпечення максимальної стійкості до атак необхідно використовувати алгоритми з досить довгими ключами, вдосконалювати алгоритми залежно від розвитку математичних методів та нових типів атак, а також використовувати найкращі практики безпеки програмного забезпечення.

2.2.4 Швидкість алгоритму

Швидкість криптографічного алгоритму зазвичай оцінюється в термінах швидкості шифрування та розшифрування даних. Швидкість шифрування вимірюється в бітах на секунду, тоді як швидкість розшифрування вимірюється в кількості розшифрованих бітів на секунду. Швидкість криптографічних алгоритмів може варіюватися в залежності від різних факторів, таких як розмір ключа, складність алгоритму та кількість операцій, які необхідно виконати для шифрування або розшифрування даних.

Наприклад, швидкість шифрування та розшифрування даних для криптографічного алгоритму AES залежить від розміру ключа. Для AES-128 швидкість шифрування може складати близько 1 Гбіт/с, тоді як для AES-256 ця швидкість може бути приблизно вдвічі меншою.

Одним з головних факторів, які впливають на швидкість криптографічного алгоритму, є його складність. Наприклад, RSA є дуже складним алгоритмом, тому його швидкість виконання може бути значно нижчою, ніж для шифрування алгоритмів, таких як AES, які є менш складними і можуть бути виконані значно швидше.

Інший фактор, який може впливати на швидкість криптографічного алгоритму-це розмір ключа. Більші ключі зазвичай потребують більшої кількості операцій для шифрування та розшифрування даних, що може зменшити швидкість алгоритму.

Крім того, кількість операцій, які необхідно виконати для шифрування та розшифрування даних, також може впливати на швидкість криптографічного алгоритму. Наприклад, алгоритми, які використовують більш складні операції, можуть виконуватися повільніше, ніж ті, які використовують менш складні операції.

Оскільки швидкість криптографічного алгоритму є важливим фактором при виборі алгоритму для захисту даних, розробники криптографічних алгоритмів зазвичай працюють над зменшенням кількості операцій та складності алгоритму для забезпечення більшої швидкості.

Отже, при виборі криптографічного алгоритму важливо враховувати не лише його швидкість, а й безпеку та стійкість до атак. Швидкість не повинна ставати на перше місце, якщо це знижує безпеку захисту даних.

2.2.5 Підтримка стандартів

Підтримка стандартів для криптографічних алгоритмів - це важливий аспект, який дозволяє забезпечити сумісність та інтероперабельність між різними системами, які використовують криптографічні алгоритми. Стандарти забезпечують уніфікований підхід до захисту даних та забезпечують, що криптографічні алгоритми можуть використовуватися у різних контекстах.

Один з найвідоміших стандартів для криптографічних алгоритмів - це стандарт AES. AES є симетричним алгоритмом шифрування, який може бути використаний для захисту даних в багатьох системах, включаючи бездротові мережі, інтернет-протоколи та програмне забезпечення. AES використовується у багатьох стандартах, включаючи стандарти TLS та IPsec.

Іншим важливим стандартом є стандарт RSA. RSA є асиметричним алгоритмом шифрування, який використовується для захисту даних від несанкціонованого

доступу та автентифікації користувачів. RSA використовується у багатьох системах, включаючи безпеку електронної пошти та підпису документів.

Інші важливі стандарти для криптографічних алгоритмів включають стандарт SHA, який використовується для хешування даних та створення цифрових підписів, та стандарт HMAC, який використовується для забезпечення цілісності та автентифікації даних.

Одним з найбільш важливих стандартів для криптографічних алгоритмів є стандарт FIPS. Стандарт FIPS встановлює вимоги до криптографічних алгоритмів, які використовуються для захисту федеральної інформації та інформації, яка обробляється в урядових системах США. Цей стандарт встановлює такі вимоги, як довжина ключа шифрування, швидкість алгоритму, стійкість до атак та інші технічні характеристики.

Стандарти FIPS дуже важливі для захисту даних в урядових системах США та інших країнах, які використовують подібні стандарти. Ці стандарти дозволяють забезпечити високий рівень безпеки для захисту конфіденційної інформації та забезпечити інтероперабельність між різними системами.

Підтримка стандартів для криптографічних алгоритмів також включає в себе оновлення стандартів з часом. Швидкий розвиток технологій та зростання обсягу даних, які потрібно захистити, означають, що стандарти повинні бути оновлені, щоб забезпечити найвищий рівень безпеки.

Наприклад, останнім часом широко використовуваний стандарт SHA-1 був знятий з експлуатації через його вразливість до атак. Замість цього було запропоновано більш безпечні стандарти, такі як SHA-256 та SHA-3.

2.2.6 Відкритість алгоритму

Відкритість криптографічного алгоритму є важливим аспектом забезпечення безпеки та довіри до криптографічних систем. Відкритість означає, що криптографічний алгоритм може бути вивчений та перевірений будь-якою зацікавленою стороною, включаючи незалежних дослідників, експертів з

криптографії та безпеки, а також потенційних користувачів системи. Криптографічні системи повинні бути побудовані на основі принципу "безпека шляхом конструкції", що означає, що система має бути стійкою до атак, навіть якщо зловмисник знає, як вона працює. Це досягається завдяки використанню криптографічних алгоритмів, які є відкритими та можуть бути перевірені дослідниками з усього світу.

Відкритість криптографічних алгоритмів дозволяє незалежним дослідникам перевірити, чи є алгоритм стійким до атак, ідентифікувати можливі вразливості та розробляти нові методи атаки. Такі перевірки та дослідження допомагають покращувати безпеку систем, що використовують криптографічні алгоритми.

Крім того, відкритість криптографічних алгоритмів дозволяє створювати більш довірливі та стабільні системи, забезпечуючи конкуренцію між різними реалізаціями криптографічних алгоритмів. За наявності відкритих стандартів, різні компанії та розробники можуть створювати свої реалізації криптографічних алгоритмів, що дозволяє вибрати найкращий варіант для конкретної системи.

Крім того, відкритість криптографічних алгоритмів дозволяє забезпечити взаємну довіру між різними країнами та організаціями, що використовують криптографічні системи. Наприклад, стандарти для криптографії, прийняті у США, є відкритими та доступними для перевірки всіма країнами, що дозволяє забезпечити взаємне довіру між країнами у використанні алгоритмів шифрування. Проте існують і деякі аргументи проти відкритості криптографічних алгоритмів. Одним з них є те, що відкритість може спричинити збільшення кількості атак на системи, що використовують такі алгоритми. Також існує ризик того, що відомість про вразливості криптографічних алгоритмів може бути використана злочинцями для злому систем.

Однак, загалом відкритість криптографічних алгоритмів є корисною та необхідною для забезпечення безпеки систем та довіри між різними країнами та організаціями. Більше того, відкритість може сприяти швидкому виявленню та вирішенню вразливостей, що забезпечує постійний розвиток та покращення безпеки систем.

2.2.7 Кросплатформеність

Криптографічні алгоритми мають бути кросплатформними, щоб бути застосовуваними на будь-яких пристроях та операційних системах. Кросплатформеність дозволяє зберігати та передавати дані між різними платформами без втрати цілісності даних або їх порушення.

Криптографія використовується для захисту даних на різних платформах та пристроях. Якщо криптографічні алгоритми не є кросплатформними, то можуть виникнути проблеми з їх застосуванням на різних платформах та операційних системах. Наприклад, якщо деякий криптографічний алгоритм працює тільки на Windows, то він не буде доступний на macOS або Linux. Це може зробити неможливим захист даних на цих платформах. Крім того, відсутність кросплатформеності може створити проблеми зі збереженням та передачею даних між різними платформами. Це може призвести до втрати даних або їх порушення.

Інший спосіб забезпечити кросплатформеність - це використання криптографічних бібліотек, які мають кросплатформну підтримку. Криптографічні бібліотеки забезпечують безпеку даних та підтримують кросплатформність, дозволяючи застосовувати криптографічні алгоритми на різних платформах.

2.2.8 Масштабованість

Масштабованість криптографічного алгоритму - це його здатність працювати ефективно при збільшенні обсягу даних та кількості користувачів. Для криптографічних алгоритмів, які використовуються у великих мережах, таких як Інтернет, масштабованість є дуже важливою.

Одним із основних чинників, які впливають на масштабованість криптографічного алгоритму, є його складність. Найбільш складні криптографічні алгоритми зазвичай мають високу обчислювальну складність, що може стати перешкодою для їх застосування в масштабованих системах. Для забезпечення масштабованості криптографічних алгоритмів можна використовувати різні

стратегії. Наприклад, можна розробити алгоритм, який може працювати паралельно на багатьох процесорах або на кластері комп'ютерів. Це зменшує час, необхідний для обчислення криптографічних функцій, і забезпечує ефективнішу роботу в масштабованих системах.

Ще однією стратегією для забезпечення масштабованості є використання різних видів криптографії, таких як симетрична криптографія та асиметрична криптографія.

Для забезпечення масштабованості криптографічних алгоритмів також можна використовувати технології розподіленого збереження даних, такі як блокчейн. У блокчейні дані розподіляються між багатьма вузлами, що забезпечує безпеку та масштабованість. Крім того, можна використовувати різні методи шифрування, такі як шифрування на рівні пристрою, що дозволяє зберігати дані в зашифрованому вигляді на пристрої, тим самим забезпечуючи безпеку та конфіденційність.

Узагальнюючи, масштабованість є важливим фактором для ефективної роботи криптографічних алгоритмів у великих мережах та системах. Для забезпечення масштабованості можна використовувати різні стратегії, такі як паралельні обчислення, різні види криптографії та технології розподіленого збереження даних. Важливо враховувати масштабованість при виборі та розробці криптографічного алгоритму для конкретної системи.

2.2.9 Вартість

Вартість криптографічних алгоритмів може варіюватися в залежності від різних факторів, таких як рівень безпеки, складність реалізації та обчислювальні вимоги.

Найбільш відомі криптографічні алгоритми, такі як AES та RSA, є безкоштовними для використання та розповсюдження. Однак, реалізація цих алгоритмів може вимагати значних витрат на обчислювальні ресурси, а також час та зусилля для розробки та тестування.

Деякі криптографічні алгоритми, які зазвичай використовуються в комерційних або промислових застосуваннях, можуть мати певні вартісні обмеження, залежно від

виробника або постачальника послуг. Наприклад, популярний криптографічний алгоритм SSL / TLS може потребувати придбання ліцензії для комерційних застосувань.

При виборі криптографічного алгоритму для конкретного застосування важливо враховувати його вартість, а також його ефективність та безпеку. Крім того, важливо враховувати можливість масштабування алгоритму та його відповідність різним стандартам та рекомендаціям.

2.2.10 Зручність використання

Зручність використання криптографічних алгоритмів залежить від багатьох факторів, таких як складність реалізації, швидкість обробки даних, зручність інтерфейсу користувача, здатність до масштабування та інші.

Один з найважливіших факторів для зручного використання криптографічних алгоритмів - це складність реалізації. Якщо реалізація алгоритму є складною, то користувачам може бути важко його використовувати. Для того, щоб спростити реалізацію алгоритму, можуть використовуватися бібліотеки та інструменти, які забезпечують готові модулі для реалізації криптографічних алгоритмів.

Інший важливий фактор для зручного використання криптографічних алгоритмів - це швидкість обробки даних. Швидкість обробки даних може впливати на швидкість роботи програми та час відповіді на запит користувача. Тому важливо використовувати криптографічні алгоритми, які мають високу швидкість обробки даних.

Зручність інтерфейсу користувача також є важливим фактором для зручного використання криптографічних алгоритмів. Інтерфейс користувача повинен бути зрозумілим та зручним для користувача. Для зручності користувачів можуть використовуватися графічні інтерфейси, які дозволяють легко налаштувати та використовувати криптографічні алгоритми [9].

Ці критерії не є вичерпним списком, але можуть служити основою для оцінки криптографічного захисту інформації та вибору оптимального криптографічного алгоритму для конкретного застосування.

2.3 Підходи та методи криптографічного захисту інформації

У цьому підрозділі буде розглянуто різноманітні криптографічні алгоритми для їх подальшого аналізу та порівняння.

2.3.1 Data Encryption Standard

DES був розроблений на початку 1975 року в лабораторіях IBM Горстом Фістелем. DES був схвалений Національним бюро стандартів в 1978 році. DES був стандартизований Американським національним інститутом стандартів під назвою ANSI X3.92, краще відомим як DEA. Раніше DES був домінуючим алгоритмом симетричного шифрування електронних даних. Але тепер це застарілий метод шифрування даних за допомогою симетричного ключа. DES використовує 56-бітний ключ для шифрування та дешифрування. Він робить 16 раундів шифрування на кожному блоку даних розміром 64 біти. Стандарт шифрування даних працює за певним принципом (рис. 2.4).. Стандарт шифрування даних - це симетрична система шифрування, яка використовує блоки даних розміром 64 біти, з яких 8 бітів (один октет) використовуються для перевірки парності (для підтвердження цілісності ключа) [7]. Кожен біт парності ключа використовується для перевірки одного з октетів ключа за допомогою непарної парності, тобто кожен біт парності відкоригований так, щоб у октеті, до якого він належить, було непарне число «1». Таким чином, ключ має дійсну корисну довжину 56 бітів, що означає, що в алгоритмі фактично використовуються тільки 56 бітів . Тому для знаходження правильного ключа потрібно максимум 256 або 72 057 594 037 927 936 спроб [8].

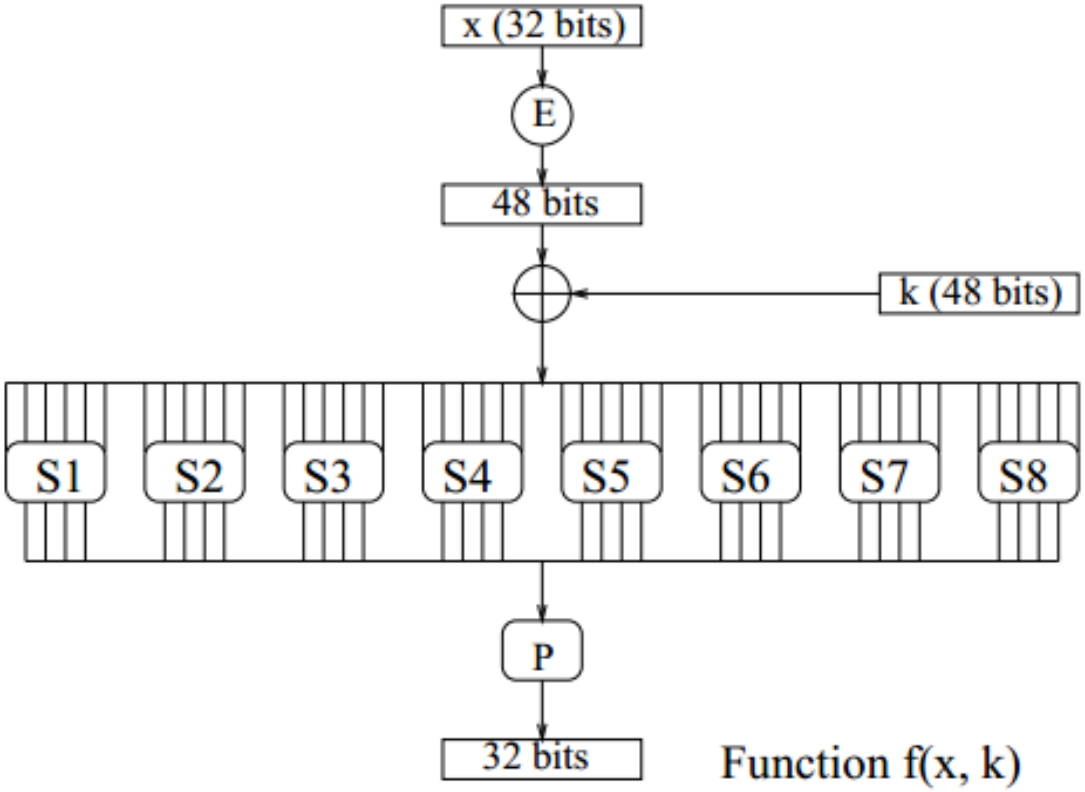

Рисунок 2.4 - Функція F для DES

Структура функції F алгоритму DES. Блок повідомлення поділяється на дві половини. Права половина розширюється з 32 до 48 біт за допомогою іншої фіксованої таблиці. Результат поєднується з підключем для того раунду за допомогою операції XOR. Використовуючи S-блоки, 48 отриманих бітів знову перетворюються на 32 біти, які подальшим чином переставляються за допомогою ще однієї фіксованої таблиці. Ця права половина, яка тепер вже повністю перемішана, поєднується з лівою половиною за допомогою операції XOR. На наступному раунді ця комбінація використовується як нова ліва половина. Багато експертів з безпеки вважали, що довжина ключа 56 біт була недостатньою ще до того, як DES був прийнятий як стандарт. Однак, DES залишався довіреним та широко використовуваним алгоритмом шифрування до середини 1990-х років [7]. Але в 1998 році комп'ютер, побудований EFF, розшифрував DES-закодоване повідомлення за 56 годин. Наступного року, використовуючи потужність тисяч мережевих комп'ютерів, EFF зменшила час розшифрування до 22 годин. Стандарт шифрування даних (DES) також може використовуватися для шифрування для одного користувача, наприклад, для зберігання деяких даних на жорсткому диску (рис. 2.5).

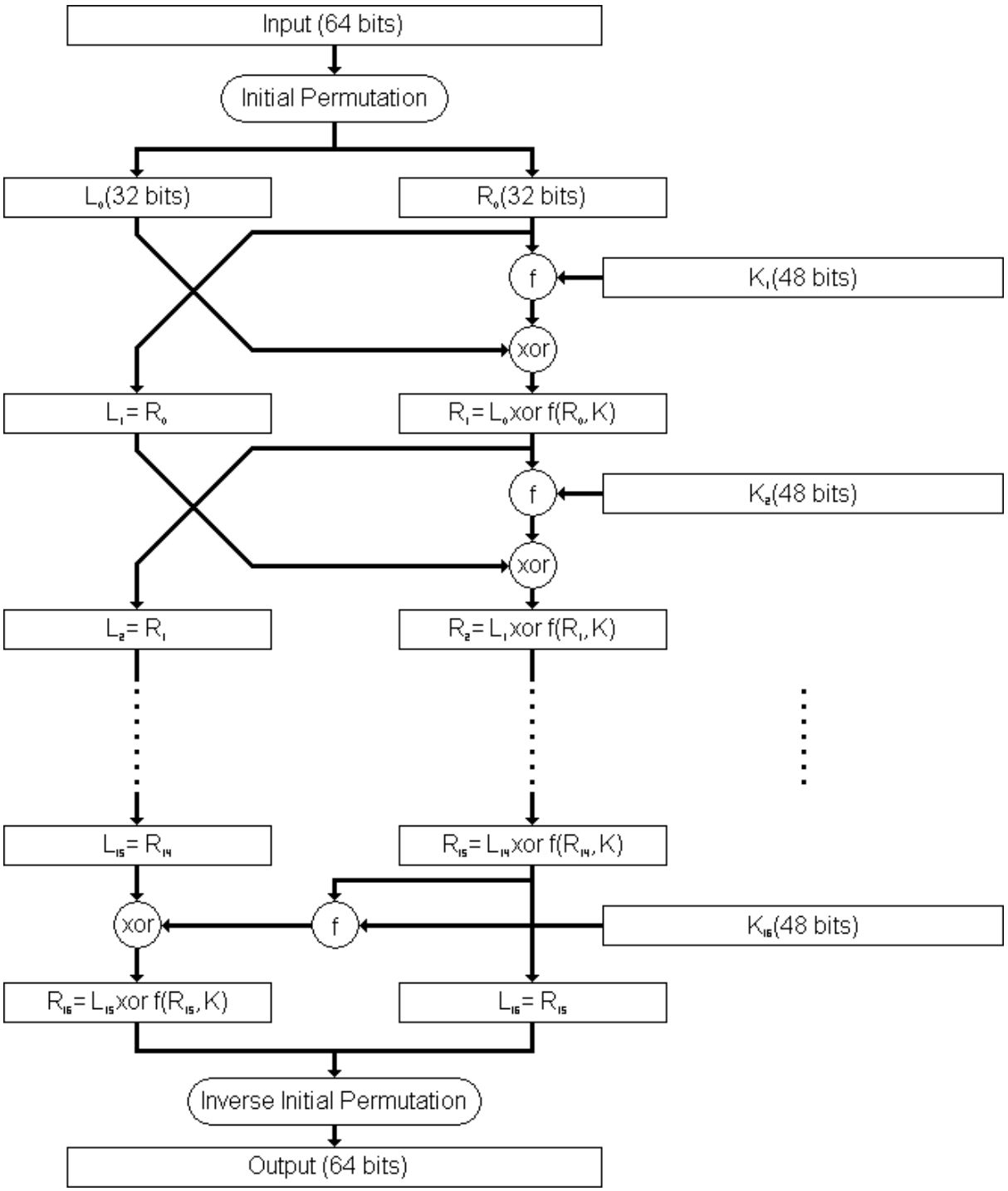

Рисунок 2.5 – Процедура шифрування DES

У своєму процесі шифрування DES використовує ключ завдовжки 56 біт як для шифрування, так і для розшифрування. Кожен блок даних розміром 64 біти проходить 16 раундів шифрування. На кожному раунді використовується функція F. DES має такі режими роботи: ECB, CBC, CFB та OFB [8]. Сила шифрування прямо пов'язана з розміром ключа, і ключі завдовжки 56 біт стали занадто малими в порівнянні з обчислювальною потужністю сучасних комп'ютерів. Тому NIST відчула потребу у

новому та більш безпечному алгоритмі шифрування даних в цій галузі. Стандарт шифрування даних був офіційно вилучений у травні 2005 року.

Незважаючи на те, що було виявлено низьку стійкість до атак, за винятком невеликого розміру ключа, який надає меншу безпеку. Єдину успішну атаку на DES було здійснено методом "brute force". Ще одним його слабким місцем є швидкість шифрування, яка є дуже повільною.

2.3.2 Triple Data Encryption Standard

У криптографії, 3DES є загальною назвою симетричного блокового шифру TDEA, який застосовує алгоритм шифрування DES три рази до кожного блоку даних. Також у 1978 році компанією IBM було запропоновано 3DES як заміну DES. Отже, 3DES це просто симетричний алгоритм шифрування DES, який застосовується три рази до одних і тих же даних. Трійний DES також називають T-DES (рис. 2.6). Він використовує простий алгоритм шифрування DES три рази для підвищення безпеки зашифрованого тексту [10].

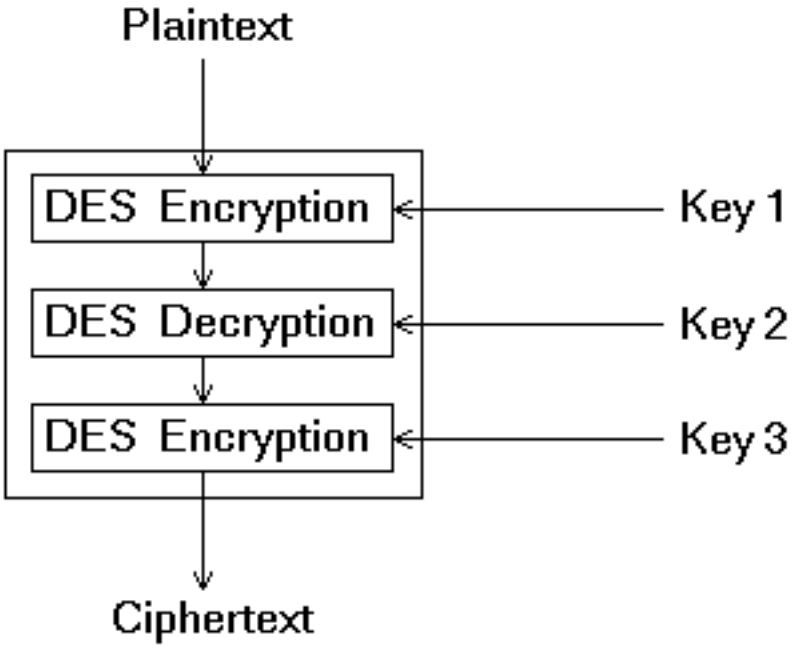

Рисунок 2.6 – Структура 3DES

У цьому методі дані шифруються тричі за допомогою DES. Таким чином, шифрування стає більш міцним і складнішим для розшифрування. Triple DES базується на блочному шифрі, який використовує 48 раундів (втричі більше, ніж у DES), та має довжину ключа 168 біт. Для шифрування 3-DES також використовується розмір блоку 64 біта [11]. Існують такі конфігурації:

– DES-EDE3 – шифрування, дешифрування та шифрування з використанням 3 унікальних ключів (Key1, Key2, Key3), які згадані вище.

– DES-EEE3 – блок даних шифрується, потім шифрується знову з використанням іншого ключа, і нарешті шифрується ще раз з використанням іншого ключа, використовуючи в загальному 3 унікальні ключі.

– DES-EDE2 – використовуються лише два ключі, при цьому перше і останнє шифрування виконується за допомогою точно такого ж ключа.

– DES-EEE2 – нарешті, тут також використовуються два ключі, перше і останнє шифрування виконуються за допомогою одного й того ж ключа.

3DES - це підхід про повторне використання алгоритму шифрування DES, але з трьома різними ключами. Вважається, що 3DES є безпечним принаймні до 2^{112} стійкості, але він повільний, особливо в обчисленнях програмного забезпечення [12]. 3-DES також забезпечує достатню безпеку. Тому користувачам потрібен наступник 3-DES.

Основною перевагою Triple DES є те, що він в три рази стійкіший (оскільки це поєднання трьох алгоритмів DES з різними ключами на кожному рівні) ніж DES, тому він віддається переважно над простим алгоритмом шифрування DES. Він забезпечує достатню безпеку для даних, але він не є найкращим, оскільки він споживає багато часу і швидкість його шифрування також менше, ніж у алгоритму шифрування DES.

2.3.3 RSA

Алгоритм RSA є найважливішою системою криптографії з відкритим ключем. Це найбільш відома та широко використовувана схема з відкритим ключем. Вона

використовує великі цілі числа розміром 1024 біта. У нього є лише один раунд шифрування. Це асиметричний блочний шифр. RSA - це алгоритм, який використовують сучасні комп'ютери для шифрування та дешифрування повідомлень. RSA - це асиметричний криптографічний алгоритм. Асиметричний означає, що в процесі шифрування та дешифрування використовуються два різні ключі (рис. 2.7).

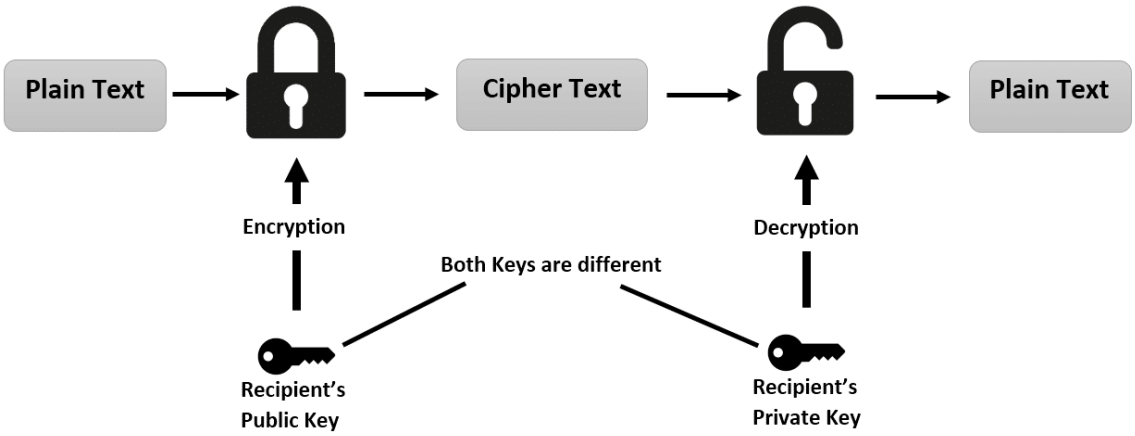

Рисунок 2.7 – RSA Алгоритм

Це також називається криптографією з відкритим ключем, оскільки один з ключів може бути розповсюдженим серед усіх, а інший ключ має бути збереженим приватним. Вона базується на проблемі факторизації. RSA - це абревіатура від Рона Рівеста, Аді Шамира та Леонарда Адлемана, які розробили і описали його у 1978 році [11]. Користувач RSA створює та публікує добуток двох великих простих чисел ($P * Q$) разом з допоміжним значенням (I) як їхній публічний ключ. Прості множники ($P * Q$) мають бути збережені в таємниці. Будь-хто може використовувати публічний ключ для шифрування повідомлення, але з використанням поточних методів, якщо публічний ключ достатньо великий, тільки той, хто знає прості множники, може дешифрувати повідомлення. Алгоритм RSA може бути використаний як для шифрування з відкритим ключем, так і для цифрового підпису. Його безпека базується на складності факторизації великих цілих чисел.

Наступний алгоритм використовується в RSA:

– Виберіть p та q

– Обчисліть $n = (p * qn)$

– Обчисліть $\varphi(n) = (p-1) * (q-1)\ 1 < e < \varphi(n)$

– Виберіть e таке, що $1 < e < \varphi(n)$ та e та n є взаємно простими.

– Обчисліть значення d таке, що $d * e\ \%\ \varphi(n) = 1$.

– Публічний ключ - (e, n).

– Приватний ключ - (d, n).

– Для шифрування $C = me(mod\ n)$, а для дешифрування $m = cd(mod\ n)$.

Отже, за допомогою вище зазначеного алгоритму відкритий текст перетворюється в зашифрований вигляд або шифротекст, а потім розшифровується з шифротексту до відкритого тексту.

У криптографічному алгоритмі RSA головним недоліком є швидкість шифрування. Для шифрування даних витрачається багато часу. Насправді, це недолік асиметричних ключових алгоритмів через використання двох асиметричних ключів. Він забезпечує хороший рівень безпеки, але повільний для шифрування файлів. Ще одною загрозою в цьому алгоритмі є підробка ключа на етапі дешифрування, тому секретний ключ повинен бути приватним та вірним, щоб успішно зашифрувати дані.

2.2.4 AES

У 1997 році NIST оголосив про ініціативу з вибору наступника DES. У 2001 році був обраний Advanced Encryption Standard як заміну DES та 3DES. AES був розроблений Вінсентом Рейменом та Джоан Даєманом у 2001 році. AES є симетричним блочним шифром, який використовується урядом США для захисту класифікованої інформації та виконується в програмному та апаратному забезпеченні по всьому світу для шифрування чутливої інформації. Фактично AES складається з трьох блочних шифрів: AES-128, AES-192 та AES-256. Кожен з цих шифрів шифрує та розшифровує дані блоками по 128 біт, використовуючи криптографічні ключі відповідно по 128, 192 та 256 біт (рис. 2.8). У Advanced Encryption Standard є 10 раундів для 128-бітних ключів, 12 раундів для 192-бітних ключів та 14 раундів для 256-бітних ключів [13].

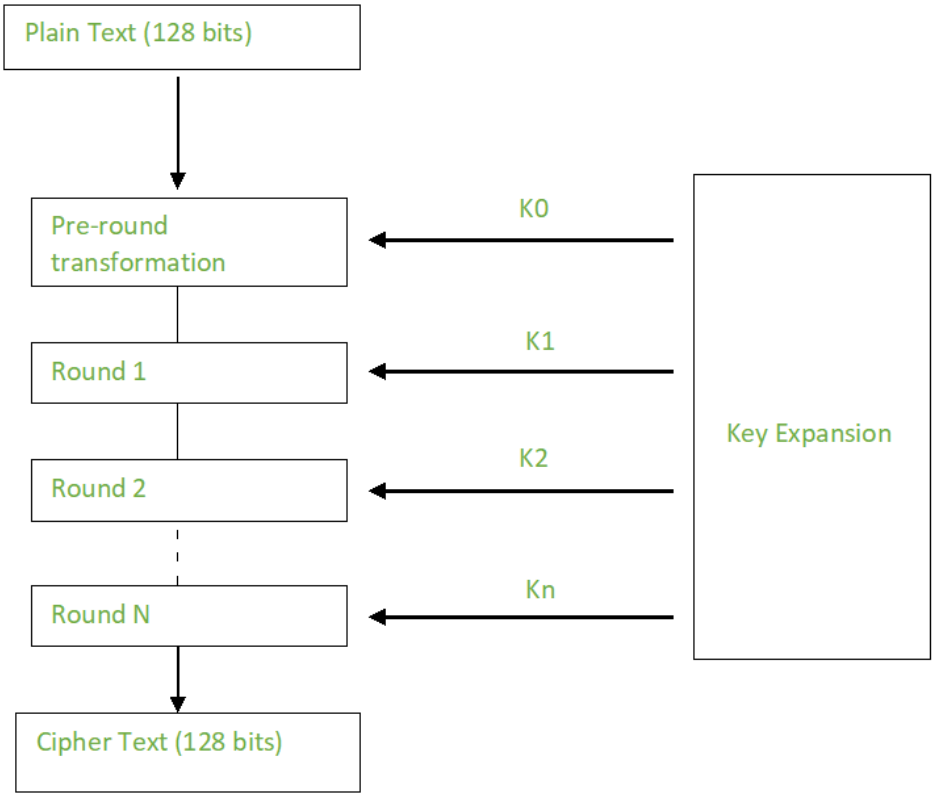

Рисунок 2.8 – AES Алгоритм

У кожному випадку всі інші раунди ідентичні, окрім останнього раунду. Кожен раунд у процесі шифрування далі складається з декількох кроків для завершення кожного раунду до n. Кожен раунд має чотири етапи: заміна байтів, зсув рядків, змішування стовпців та додавання ключа раунду (рис. 2.9).

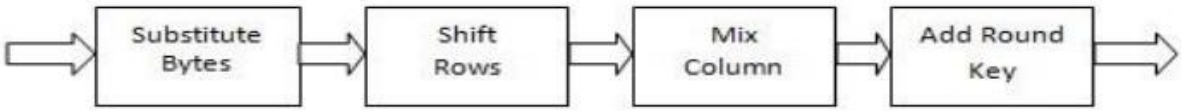

Рисунок 2.9 – Схема раундів AES

– Раунд заміни. У цьому кроці зашифровані байти замінюються один за одним за допомогою таблиці S-блоків під час прямого процесу шифрування.

– Зсув рядків. У цьому кроці рядки стану масиву зсуваються під час прямого процесу (за допомогою S-блоку).

– Змішування колонок. Змішування колонок для змішування окремих байтів у кожній колонці під час прямого процесу.

– Додавання раундового ключа. У цьому кроці до виходу з попереднього кроку додається раундовий ключ під час прямого процесу. Цей крок відрізняється від інших через розмір ключа.

У процесі шифрування AES використовуються різні раундові ключі. Ці ключі застосовуються разом з іншими математичними операціями до масиву даних. Ці дані присутні в блоках певного розміру. Цей масив називається масивом стану. Цей процес шифрування включає наступні процеси:

– Спочатку виведіть різні раундові ключі з ключа шифрування.

– Ініціалізуйте масив стану даними блоку або текстом.

– Почніть з початкового масиву стану, додавши раундовий ключ.

– Виконайте процес маніпулювання станом протягом дев'яти раундів.

– Після десятого раунду маніпулювання ми отримаємо кінцевий вихід у вигляді шифротексту.

Дотримуючись вищезазначеного процесу, ми отримуємо кінцевий зашифрований текст або шифротекст.

2.4 Порівняння та аналіз алгоритмів для криптографічного захисту інформації

У цьому підрозділі для оцінки та порівняння ефективності шифрування використовуються критерії, що були визначені у попередньому підрозділі [13]. А саме:

– Надійність безпеки – цей фактор пов’язаний із досягненням захищенності шифрування від підбору «brute force», таймінг-атаки та різних криптоаналітик відкритого тексту.

– Довжина та тип використовуваного ключа – це фактор, який стосується безпеки методу та визначає тип методу шифрування.

– Структура алгоритму – якщо алгоритм має структуру Фейстеля, де перестановка і операція заміни реалізовується на півблоку в кожному раунді.

– Час впровадження та швидкість алгоритму – цей фактор визначає час, необхідний для впровадження всіх операцій алгоритмів і пов'язаний зі складністю алгоритму і швидкістю процесора.

– Складність і вартість – цей фактор пов’язаний з типом процесу та розміром пам’яті, що використовується для реалізації алгоритм шифрування.

– Лавинний ефект – це властивість алгоритму, якщо стається невелика зміна у вхідному тексту або у ключі, то це викликає значні зміни в зашифрованому тексті [14].

Розглянемо результати порівняння для оцінки ефективності симетричного та асиметричного алгоритмів шифрування за різними параметрами, які впливають на вибір шифрування. Час вимірюється як високий та низький. Швидкість визначається такими термінами, як швидка, повільна та помірна. Визначаємо структуру алгоритму в “Feistel” і “non Feistel”. Довжина ключа вимірюється в використаних бітах. Лавинний ефект оцінюється як сильний. Енергоспоживання (використана пам'ять) визначається як висока або низька. Результати аналізу наведено нижче (табл. 2.1):

Таблиця 2.1

Порівняння методів шифрування

Критерій	RSA	AES	DES	3DES
Тип шифрування	Асиметричне шифрування	Симетричне шифрування	Симетричне шифрування	Симетричне шифрування
Стійкість безпеки проти атак	Атака по часу	Груба сила, статистична атака, диференціальна та лінійна атака	Груба сила, вразливий до лінійної атаки	Груба сила, атака посередника
Довжина ключа	Велике просте число> 1024 біт	128, 192 або 256 біт	64 біти	129 біт

Критерій	RSA	AES	DES	3DES
Тип ключа	відкритий ключ і закритий ключ, один для шифрування і один для дешифрування	один закритий ключ для шифрування та дешифрування	один закритий ключ для шифрування та дешифрування	один закритий ключ для шифру та дешифру
Структура алгоритму	“non Feistel”; немає раундів	“non Feistel”; 10,12,14 раундів	“Feistel”; 16 раундів	“Feistel”; 16 раундів
Швидкість алгоритму	Висока	Висока	Висока	Висока
Використання пам’яті	Високе	Менше ніж DES	Більше ніж AES	Більше ніж DES
Лавинний ефект	Менше ніж для AES	Сильніше ніж для DES	Сильний, менше ніж для AES	Сильний, менше ніж для AES
Показники міцності	Велике просте число в генерації ключів	Великий розмір ключа і блоку; Кількість раундів і складність операції шифрування та процесу генерації ключів	Кількість раундів; Плутанина та поширення відповідно до використання S-box, P-box	Великий розмір ключа; Кількість раундів; Плутанина та поширення відповідно до використання S-box, P-box

З таблиці вище видно, що AES є більш безпечним, ніж DES, відповідно до великого розміру ключа та складності операції шифрування в кожному раунді, а також складності генерації ключа. Також кількість раундів підвищує безпеку алгоритму шифрування. Оскільки споживання пам’яті зменшилося, швидкість реалізації алгоритму зросла, тому AES вважається найкращим рішенням серед алгоритмів симетричного шифрування, оскільки він характеризується простотою, де його можна легко реалізувати за допомогою дешевих процесорів і мінімального

обсягу пам’яті. Серед асиметричних алгоритмів шифрування алгоритм RSA забезпечує більшу безпеку, оскільки він використовує факторізацію великого простого числа для генерації ключів. Таким чином, алгоритм RSA визнано найкращим рішенням серед алгоритмів асиметричного шифрування.

2.5 Порівняння варіацій RSA

З появою системи шифрування RSA у 1978 році, прості числа відіграли важливу роль у забезпеченні безпечної передачі конфіденційної інформації мережевими каналами. RSA базується на збільшенні двох зазвичай величезних простих чисел. RSA використовує два ключі - публічний та приватний. Повідомлення спочатку перетворюється в числовий формат, який можна представити у вигляді послідовності бітів. Потім цей числовий текст шифрується з використанням публічного ключа. Коли повідомлення потрібно розшифрувати, це робиться за допомогою приватного ключа [15]. Факторизація простих чисел є дуже складною і фреймворк RSA використовує цю багату властивість. У будь-якому випадку, використання надзвичайно великих простих чисел для RSA збільшує обчислювальний час при шифруванні та дешифруванні даних, який повинен бути скоригований для безперервних застосувань. Враховуючи це обмеження, зловмисники зосереджуються на тому, щоб зламати фреймворк RSA, знайшовши вправні стратегії для факторизації простих чисел [16].

RSA є одним з найбільш поширених алгоритмів криптографії. Він використовується для цифрового підпису, де він генерує електронний підпис, що підтверджує автентичність повідомлення та ідентифікує автора. Це використовується в багатьох інтернет-протоколах, включаючи HTTPS, SSL та TLS. RSA також використовується для безпечного з'єднання з віддаленим сервером через SSH-протокол. У цьому випадку, RSA використовується для автентифікації клієнта та сервера для забезпечення безпеки обміну даними між ними.

Крім того, RSA має широке застосування в різних сферах, таких як електронна комерція, банківські операції та забезпечення інтернет-безпеки. Існує кілька варіантів

RSA з різними розмірами ключів, які впливають на швидкість шифрування та безпеку.

Одним з головних переваг RSA є те, що він використовує відкриті ключі, що дозволяє безпечно передавати їх по мережі без ризикує викриття приватного ключа. Крім того, RSA дуже ефективний для шифрування даних невеликого обсягу, таких як ключі шифрування та підписи.

Однак, RSA також має деякі недоліки та обмеження. Зокрема, його використання може бути повільнішим порівняно з іншими алгоритмами шифрування, такими як AES. Крім того, RSA вимагає більшої довжини ключа для забезпечення високого рівня безпеки, що може стати проблемою для використання в деяких обмежених середовищах.

У цілому, RSA є одним з ключових алгоритмів криптографії, що забезпечує високий рівень безпеки та конфіденційності для передачі конфіденційної інформації та цифрового підпису. Його застосування широко поширене в Інтернеті та комп'ютерних мережах, але водночас його безпека та ефективність постійно вдосконалюються для забезпечення найвищого рівня захисту [17,18].

2.5.1 RSA

Рівестом, Шаміром та Адлеманом була створена перша версія алгоритму RSA, що дозволяє шифрувати інформацію через відкритий канал за допомогою двох ключів - публічного та приватного. Публічний ключ можна передавати відкрито, але розшифрувати дані можна лише за допомогою приватного ключа. Хоча базовий алгоритм мав багато варіацій, його безпека залежала від обчислювальних ресурсів. Було складно атакувати або оновлювати його алгоритми. Однак на сьогодні RSA було атаковано багатьма способами, такими як підбір ключа грубою силою, математичні атаки, атаки на час та атаки на вміст обраного шифру [19].

2.5.2 Efficient RSA

Абуд та Джаббар запропонували варіант RSA під назвою "Ефективний RSA". Цей варіант використовує концепцію лінійного групування, де h - порядок загальної лінійної групи, значення якої вибираються випадковим чином з кільця натуральних чисел за $mod(n)$, а n є результатом двох величезних простих чисел. Число, яке є спільним простим з n, утворює лінійну групу множення за модулем n з порядком $g(n)$. Обернені квадратні матриці рангу h і натуральне число за модулем n утворюють подібну групу того ж порядку, і це не видно на загальному плані [20]. У загальному плані, де n є результатом двох конкретних простих чисел, ми можемо знайти порядок для цієї групи за допомогою гіпотези супроводу:

$$g = (p^h - 1)(p^{h-p}) \dots (p^h - p^{h-1}) + (q^h - 1)(q^{h-p}) \dots (q^h - p^{h-1}), \quad (2.1)$$

де g - лінійна група матриць $h \times h$ над натуральними числами.

2.5.3 Dependent RSA

Пойнтшеваль запропонував "Залежний RSA" для забезпечення семантичної безпеки для оригінальної проблеми RSA. Криптосистема є семантично безпечною проти вибраних атак на відкритий текст у стандартній моделі, порівняно з іншою проблемою (проблема інверсії), з коефіцієнтом шифрування в кілька разів швидшим, ніж у Ель-Гамаля (при порівнянних рівнях безпеки. RSA-1024 біт проти Ель-Гамаля-512 біт) [21].

Генерація ключа:

1. $n = p * q$, де p і q надзвичайно великі прості числа.
2. $\gcd\big(e, \varphi(N)\big) = 1$, де e є відносно простим до $\varphi(N)$.
3. h: $Z_N \times Z_N \to \{0,1\}^l$, де h - лінійна група або хеш-функція.
4. Публічний ключ - це пара (n, e).
5. Приватний ключ $d = e^{-1} mod\ \varphi(N)$.

2.5.4 Shared RSA

Схема RSA зі спільним доступом - це місце, де процедура декодування розділяється між різними комп'ютерами і розшифровується паралельно. Якщо прості множники n відомі k системам. Тоді кожна система P_i має лише $< p_i, q_i, d_i >$ частину ключа [22]. Крім того, повинні бути виконані чотири умови, що супроводжують цю задачу:

1.$p = p_1 + p_2 + \cdots p_k$, де p – велике просте число.

2. q = $q_1 + q_2 + \cdots q_k$, де q – велике просте число

3. $e * d$ співпадає з 1 $mod\ \varphi(n)$.

4. $d = d_1 + d_2 + \cdots d_k$, де d – приватний ключ.

2.5.5 Carmichael RSA

Функції Кармайкла визначаються як

$$a^m \equiv 1(mod\ n) \tag{2.2}$$

де n - натуральне число, а m - найменше натуральне число, визначене функцією $\lambda(n) = lcm(p-1, q-1)$ в теорії чисел і для кожного цілого числа a від 1 до n яке є спільним простим з n є функцією Кармайкла. Де Вріс запропонував варіацію з використанням функції Кармайкла, яка позначається $\lambda(n)$, де $n = p * q$, p і q - великі прості числа і характеризується як $\lambda(n) = lcm(p-1, q-1)$. $\lambda(n)|\varphi(n)$ оскільки як функція Кармайкла $\lambda(n)$, так і функція Ейлера $\varphi(n)$ мають спільні дільники [23]. Відтепер будь-які мислимі набори ключів, отримані за допомогою функції Ейлера, можуть бути додатково отримані за допомогою роботи Кармайкла, але підстановка не є дійсною. Цей варіант визначається як (e, d) . Де $n = p * q$ можна записати як $\varphi(\lambda(n))$.

2.5.6 Multi Prime RSA

Іншим варіантом RSA є Multi Prime RSA, який використовує декілька простих чисел для генерації приватного ключа. Розмір окремих простих чисел невеликий, щоб зменшити обчислювальну складність, але результуюче просте число після перемноження цих окремих простих чисел досить велике, тому факторизація стає складною [24,25].

Генерація ключа:

1. Нехай $p_1, p_2, p_3, \dots p_n$ – різні прості числа.
2. $N = p_1 * p_2 * p_3 * \dots p_n$.
3. $\varphi(n) = (p_{1-1}) * (p_2 - 1) * (p_3 - 1) * \dots (p_n - 1)$.
4. Публічний ключ e - $\gcd(e, \varphi(n)) = 1$.
5. Приватний ключ - d, який є оберненим до n.

2.5.7 Multi Power RSA

Якщо дотримуватись якогось твердження, наприклад, що правильна структура-це те, що посилює безпеку як компонент довжини біта n, то в цьому випадку $n = p * q$, ймовірно, є правильною структурою. Тим не менш, безпека не є основною характеристикою криптосистеми. Клієнти думають про обчислювальну ефективність, консервативність, захист від атак з побічних каналів і ліцензійні претензії на інновації. Отже, може бути місце для виборчих структур ключів. Без сумніву, були запропоновані різні варіанти на противагу $n = p * q$. Спосіб, яким можна використовувати $= p_1^{e^1} * p_2^{e^2} * \dots p_l^{e^l}$, згадується в патенті RSA Коллінза і Сабіна. Різні пропозиції в письмових роботах, як правило, зосереджуються на підвищенні ефективності [26].

Генерація ключа:

1. Нехай p, q – два великих простих числа.
2. $N = p * p * p * p \dots n$ разів $* q$.
3. $\varphi(n) = (p - 1) * (q - 1)$.

4.Публічний ключ e - $\gcd(e, \varphi(n)) = 1$.

5.Приватний ключ d, який є оберненим до n.

2.5.8 Common Prime RSA

За версією Сарки, Майтри та Хінека проблема RSA полягає в його обчислювальній складності, якщо зменшити розмір ключів, то RSA може бути легко зламаний. При цьому згідно Рівесту, Шаміру, Адлеману приватне відображення $d > N^{0.292}$, де N є множенням 2 простих чисел [27].

Алгоритм не може бути зламаний, якщо приватний показник менший за $N^{0.25}$, якщо прості числа є звичайними простими. g - велике просте число, нехай $p = 2 * g * a + 1$ і $q = 2 * g * b + 1$ - прості числа, такі, що $\gcd(a, b) = 1$ і $h = 2 * g * a * b + a + b$ також є простим. Перше обмеження гарантує, що $\gcd(p - 1, q - 1) = 2 * g$, а друге гарантує, що $\frac{p*q-1}{2} = gh$, що приблизно дорівнює $n = p * q$, p і q, які задовольняють наведені вище властивості, називаються спільними простими [28, 29].

2.5.9 Chinese Reminder Theorem RSA

У цьому варіанті Вулансарі обчислення модулярного піднесення до ступеня при розшифровці можна зменшити в 0.5 рази при максимальному використанні CRT [27].

Генерація ключа:

1. p і q – два відносно великих простих числа , такі, що $p \neq q$.

2. $N = p * q$ і $\varphi(N) = (p - 1) * (q - 1)$.

3. Компонент публічного ключа e такий, що $1 < e < \varphi(N)\ with \gcd(e, \varphi(N)) = 1$.

4. Компонент приватного ключа $d = e^{-1} mod\ \varphi(N)$.

5. $d_p = d\ mod\ (p - 1)$ і $d_Q = d\ mod\ (p - 1)$

6. $q_{Inv} = q^{-1} mod\ p$

7. Відкриті ключі - (e, n) . Та приватні ключі - $(d_p, d_Q, q_{Inv}, p, q)$

2.6 Результати та аналіз

Було обрано python як мову сценаріїв, оскільки вона має багаті функціональні можливості для отримання точних результатів моделювання, а також містить багаті бібліотеки, такі як Gmpy2 та Crypto, для ефективного зворотного модулярного обчислення та генерації випадкових простих чисел.

1. RSA

Генерація ключа $d = e^{-1} mod\ \varphi(n)$

Шифрування $C = P^e mod\ n$

Розшифрування $P = C^d mod\ n$

2. Efficient RSA

Генерація ключа $d = (p^h - 1) \dots (p^h - p^{h-1}) + (q^h - 1) \dots (q^h - p^{h-1})$

Шифрування $C = P^e mod\ n$

Розшифрування $P = C^d mod\ n$

3. Dependent RSA

Генерація ключа $d = e^{-1} mod\ \varphi(n)$

Шифрування $C_1 = K^e mod\ n$

$$C_2 = M * ((K+1)^e mod\ n \tag{2.3}$$

Розшифрування $K = C_1^d mod\ n$

$$P = C_2/((K+1)^d mod\ n \tag{2.4}$$

4. Shared RSA

Генерація ключа $d = d_1 + \cdots + d_{1k}$

Шифрування $C = P^e mod\ n$

Розшифрування $m_i = c^{di} mod\ n$

$$P = p_1 * p_2 * \dots * p_k \tag{2.5}$$

5. Carmichael RSA

Генерація ключа $\lambda(n) = lcm(p-1, q-1)$

$$d = \phi(n) \backslash \lambda(n) \tag{2.6}$$

Шифрування $C = P^e mod\ n$

Розшифрування $P = C^d mod\ n$

6. Multi Prime RSA

Генерація ключа $\phi(n) = (p_1 - 1) * \ldots * (p_n - 1)$

$$d = e^{-1} mod\ \phi(n) \tag{2.7}$$

Шифрування $C = P^e mod\ n$

Розшифрування $P = C^d mod\ n$

7. Multi Power RSA

Генерація ключа $\phi(n) = p^{r-1} * (p-1) * q^{s-1} * (q-1)$

$$d = e^{-1} mod\ \phi(n) \tag{2.8}$$

Шифрування $C = P^e mod\ n$

Розшифрування $P = C^d mod\ n$

8. Common Prime RSA

Генерація ключа $p = 2 * g * a + 1$

$$q = 2 * g * b + 1 \tag{2.9}$$

$$\gcd(a, b) = 1 \tag{2.10}$$

$$h = 2 * g * a * b + a + b \tag{2.11}$$

$$\gcd(p-1, q-1) = 2 * g \tag{2.12}$$

$$\frac{p * q - 1}{2} = gh \tag{2.13}$$

$$\phi(n) = (p-1) * (q-1) \tag{2.14}$$

$$d = e^{-1} mod\ \phi(n) \tag{2.15}$$

Шифрування $C = P^e mod\ n$

Розшифрування $P = C^d mod\ n$

9. Chinese Remainder Theorem RSA

Генерація ключа $d = e^{-1}\ mod\ \phi(n)$

$$d_P = d\ mod\ (p-1) \tag{2.16}$$

$$d_Q = d\ mod\ (q-1) \tag{2.17}$$

$$q_{Inv} = q^{-1}\ mod\ p \tag{2.18}$$

$$d = (d_{P,} d_Q, q_{Inv}, p, q) \tag{2.19}$$

Шифрування $C = P^e mod\ n$

Розшифрування $P_1 = C^{dP}\ mod\ p$

$$P_2 = C^{dQ}\ mod\ q \tag{2.20}$$

$$h = q_{Inv} * (P_1 - P_2)\ mod\ p \tag{2.21}$$

$$P = P_2 + h * q \tag{2.22}$$

Вище окреслено та розглянуто всі варіації алгоритмів RSA щодо процедури генерації ключів, процедури шифрування та процедури дешифрування.

Як показано на рис 2.10, симуляція візуалізує поведінку Multi Power RSA, Multi Prime RSA, Chinese Remainder Theorem RSA, RSA, Charmichael RSA та Dependent RSA.

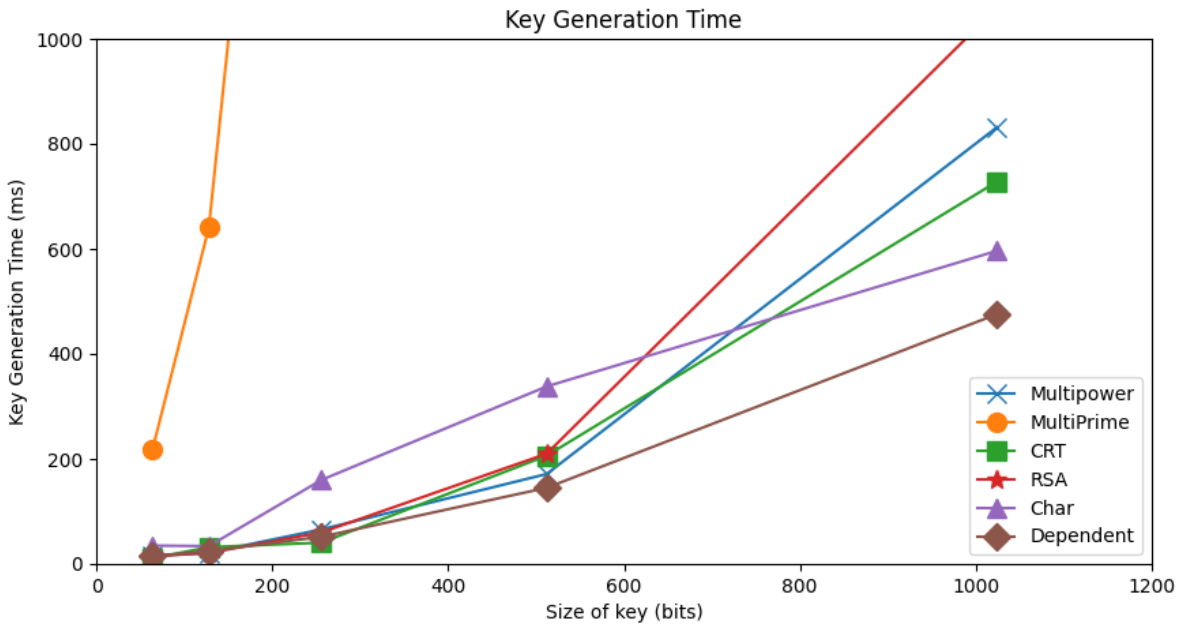

Рисунок 2.10 - Аналіз генерації приватних ключів

Для кожного варіанту генерація простих чисел відрізняється, тому результати генерації приватних ключів повністю базуються на випадкових простих числах, згенерованих під час симуляції. З рис 2.10 можна зробити висновок, що Multi Prime RSA та Multi Power RSA займають найбільше часу для генерації ключів, оскільки генерація простих чисел є дорогою операцією. У Multi Prime RSA використовується $n/2$ простих чисел, де n - розмір ключа в бітах, тоді як у Multi Power RSA потужність простих чисел p і q збільшується на $n/2$, де n - розмір ключа в бітах. Common prime RSA є ще одним варіантом, який має найбільший час генерації ключа серед усіх варіантів, що пояснюється багатогранністю генерації простих чисел. Також з рис 2.10 видно, що найменше часу на генерацію простих чисел витрачає Dependent RSA. Щоб бути послідовними, розрахуємо зворотній для приватного ключа як інтегральний фактор, тоді Charmichael RSA має найменший час генерації ключа завдяки

використанню функцій Кармайкла, де розмір чисел Кармайкла не такий великий, як числа Ейлера (табл. 2.2).

Таблиця 2.2

Генерація ключів в Common Prime RSA

Prime Size	Time (ms)
10	27
20	500
40	13576
60	111287
80	553793

У випадку простих чисел час генерації ключа RSA обчислюється окремо для ключів невеликого розміру, оскільки існує декілька множників для генерації простих чисел, що робить алгоритм повільнішим порівняно з іншими варіантами. Якщо існує ефективний алгоритм для генерації простих чисел, то можна зменшити кількість бітів у ключі за рахунок того, що факторизація до простих чисел є відносно складнішою, ніж до не простих.

Як зазначено вище, всі алгоритми, крім Dependent RSA, використовують однакову формулу шифрування, але під час моделювання, було виявлено, що час, необхідний для шифрування даних, не однаковий. Кожен варіант використовує 16 КБ даних для шифрування звичайного тексту, за винятком Multi power RSA та Multi prime RSA, де для шифрування використовується просте повідомлення "Hello". З графіка можна зробити декілька спостережень, які наведені нижче (рис. 2.11):

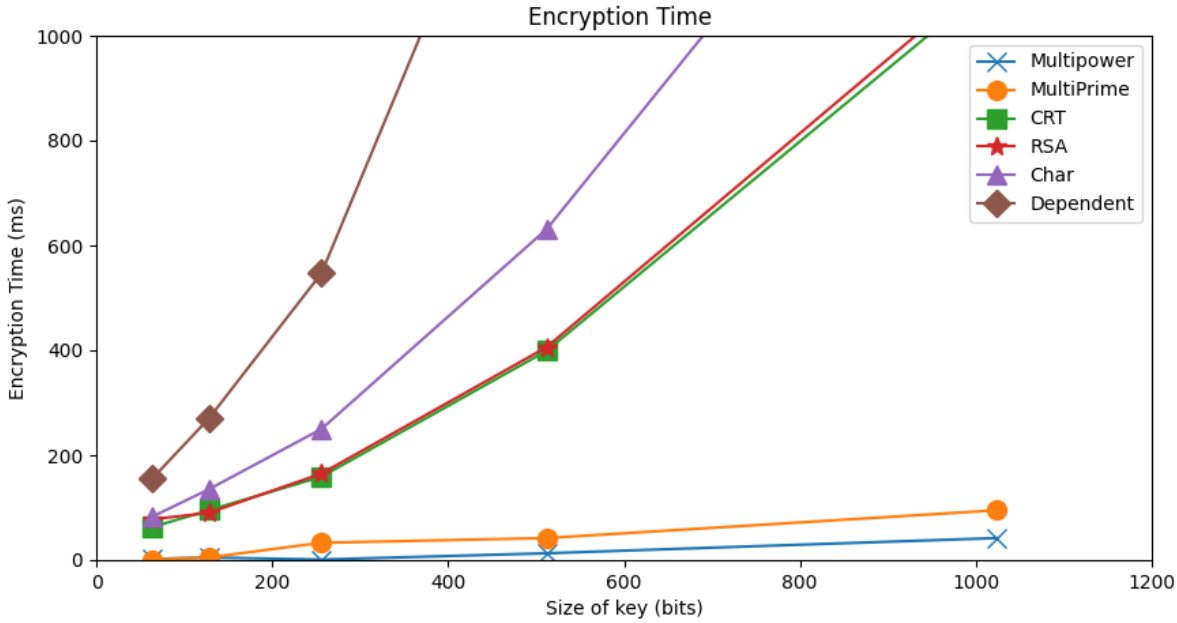

Рисунок 2.11 - Аналіз часу шифрування

1. Час шифрування 16 КБ даних для Multi prime та Multi power RSA буде високим у порівнянні з будь-яким іншим алгоритмом.

2. Час шифрування алгоритмів Carmichael RSA та Dependent RSA більший, ніж загального RSA, а час генерації ключів менший, ніж у загального RSA.

3. Chinese Remainder Theorem RSA працює так само, як і загальний алгоритм RSA, але час генерації ключа все ще кращий, ніж у загального алгоритму RSA, при кращій безпеці.

4. Shared RSA, який не включений в графік, буде найкращим алгоритмом для шифрування даних, оскільки обчислення можуть бути розбиті і змодельовані паралельно в залежності від кількості використовуваних машин.

Як зазначено на рисунку (рис. 2.12), всі алгоритми, окрім Dependent RSA, Shared RSA та CRT RSA, використовують однакову формулу розшифрування, але під час моделювання, як показано на рис.2.13, було виявлено, що час, який витрачається на розшифрування, не є однаковим. Кожен варіант використовує 16 КБ даних для розшифрування простого тексту, за винятком Multi power RSA та Multi prime RSA, де для розшифрування використовується просте повідомлення Hello. З графіка можна зробити декілька спостережень, які наведені нижче:

1. Час розшифрування 16 КБ даних для алгоритмів Multi prime та Multi power RSA буде високим у порівнянні з будь-яким іншим алгоритмом.

2. Час декодування алгоритмів Carmichael RSA та Dependent RSA більший, ніж загального RSA, в той час як час генерації ключів менший у порівнянні з загальним RSA.

3. CRT RSA перевершує загальний алгоритм RSA за часом генерації ключа, часом шифрування, часом розшифрування, а також за рівнем безпеки.

4. Розподілений RSA, який не включений в графік, буде найкращим алгоритмом для розшифрування даних, оскільки обчислення можуть бути розбиті і змодельовані паралельно в залежності від кількості використовуваних машин [15].

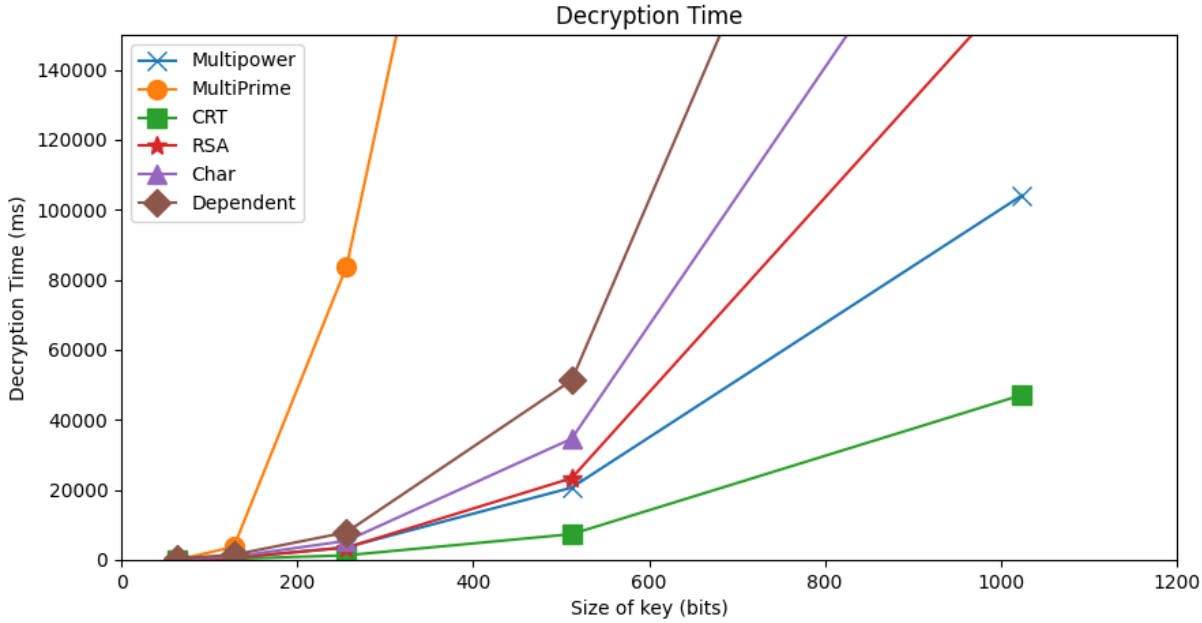

Рисунок 2.12 - Аналіз часу розшифрування

Висновки за розділом 2

В другому розділі було детально проаналізовано принципи криптографічного захисту інформації, показники для оцінки криптографічного захисту інформації. Також було ретельно порівняно алгоритми криптографічного захисту інформації за різними параметрами. Було проведено порівняльне дослідження між різновидами RSA за п'ятьма компонентами: довжина ключа, генерація простого числа, генерація закритого

ключа, кодування та декодування. Було виявлено, що кожна варіація алгоритму має кілька переваг та кілька недоліків. А деякі варіації, такі як Multi Prime RSA і Multi Power RSA, не можуть бути використані, коли розмір повідомлення занадто великий. CRT RSA може бути використана як замінник загального RSA, оскільки вона перевершує загальний RSA в багатьох ситуаціях. Основним параметром безпеки сучасних криптографічних алгоритмів є довжина ключа. Великий ключ забезпечить високий рівень безпеки, але операції також займатимуть багато часу і споживатимуть багато пам'яті. Але ця довжина, ймовірно, рано чи пізно буде занадто короткою. У нинішній час потрібні алгоритми, які з меншим розміром ключа забезпечують максимальну безпеку і стійкість для фреймворків, де обчислювальна потужність є надзвичайно низькою. Тому в поточній ситуації потрібна більш домінуюча варіація алгоритму RSA.

РОЗДІЛ 3
ПРОГРАМНА РЕАЛІЗАЦІЯ ВЕБ ЗАСТОСУНКУ

3.1 Засоби створення програмного забезпечення

Засоби створення програмного забезпечення є різноманітними і залежать від конкретних потреб і вимог проекту. Однак, загальною категорією засобів можна виділити наступні:

– Інтегровані середовища розробки. Це комплексні розробницькі середовища, які надають засоби для написання, тестування, налагодження та відлагодження коду. Вони часто мають розширення для різних мов програмування і забезпечують роботу з проектами, управління версіями, автодоповненням коду, рефакторингом та іншими корисними функціями.

– Текстові редактори. Це базові редактори, які дозволяють редагувати код у текстовому форматі. Вони можуть надавати підсвічування синтаксису, автодоповнення, вбудовані термінали і інші корисні функції.

– Компілятори та інтерпретатори. Компілятори перетворюють вихідний код у виконуваний формат, тоді як інтерпретатори виконують код без попередньої компіляції. Вони дозволяють перевірити синтаксичні помилки та виконати програму.

– Фреймворки. Набір бібліотек, інструментів і шаблонів, що надають розробникам загальну структуру та функціональність для певного виду програмного забезпечення. Вони дозволяють прискорити процес розробки і забезпечити кращу організацію коду. Приклади

3.1.1 Середовище розробки PyCharm

PyCharm - це інтегроване середовище для розробки на мові Python, розроблене компанією JetBrains [30]. Даний редактор вихідного коду включає в себе декілька функцій, а саме - відладчик, інструменти для роботи з системами контролю версій, також засоби перевірки синтаксису та рефракторингу.

Було обрано PyCharm через його переваги. Підтримка великої кількості фреймворків. PyCharm підтримує більшість популярних фреймворків Python, таких як Flask, Django, Pyramid, CherryPy та інші. Це дає можливість розробляти веб-додатки та інші проекти без необхідності вручну налаштовувати інфраструктуру.

Основними перевагами через які було обрано PyCharm є:

1. Інтеграція з Git та іншими системами контролю версій. PyCharm має вбудовану підтримку систем контролю версій, що дозволяє легко працювати з кодом, використовуючи Git, SVN, Mercurial та інші системи [31].

2. Зручний редактор коду. PyCharm має багато функцій, які полегшують редагування коду, такі як автодоповнення, вбудований дебагер, підсвічування синтаксису та інші.

3. Підтримка віртуальних середовищ. PyCharm має вбудовану підтримку віртуальних середовищ, що дозволяє встановлювати та використовувати різні версії Python та пакетів безпосередньо зі свого інтерфейсу.

4. Підтримка тестування. PyCharm має вбудовану підтримку тестування, яка дозволяє легко створювати, запускати та аналізувати тестові набори без необхідності встановлювати додаткові інструменти.

5. Зручний візуальний редактор. PyCharm дозволяє швидко створювати та редагувати файли, використовуючи вбудований візуальний редактор, який підтримує різні типи файлів, такі як HTML, CSS.

3.1.2 GitHub

GitHub - це веб-платформа для зберігання, управління та спільної роботи з проектами з використанням системи контролю версій Git [32].

Основні переваги GitHub:

1. Зберігання та контроль версій коду. GitHub дозволяє зберігати код в репозиторії, зберігаючи копії кожного коміту. Це дозволяє відстежувати зміни, повертатись до попередніх версій і вносити зміни відносно них.

2. Спільна робота. GitHub дозволяє створювати спільноти розробників, що дозволяє спільно працювати над проектами, робити пропозиції, вносити зміни та переглядати їх.

3. Інструменти для спільної роботи. GitHub надає користувачам різноманітні інструменти для спільної роботи, такі як інструменти для збірки та тестування, інструменти для створення документації та інші.

4. Інтеграція з іншими інструментами. GitHub можна інтегрувати з багатьма іншими інструментами розробки, такими як системи збірки, системи тестування, інструменти для автоматичного розгортання, інструменти для створення документації та інші.

5. Надійність та безпека. GitHub забезпечує надійне зберігання даних і захист від несанкціонованого доступу.

6. Безкоштовний доступ для відкритих проектів. Для відкритих проектів доступ до сервісу GitHub є безкоштовним.

3.1.3 Фреймворк Flask

Flask - це легкий та мінімалістичний веб-фреймворк для мови програмування Python, який дозволяє створювати веб-додатки та API з мінімальними зусиллями. Flask пропонує простий та зрозумілий API, який дозволяє розробникам швидко та ефективно створювати веб-додатки з нуля або розширювати існуючі додатки. Flask має відкритий код та має велику та дружню спільноту користувачів, що робить його більш доступним для розробки та підтримки веб-додатків [33].

Основні переваги Flask:

1. Легкість використання та навчання. Flask має просту та зрозумілу структуру, що дозволяє розробникам швидко навчитися роботі з ним та створювати веб-додатки без складнощів.

2. Мінімалістичність. Flask має мінімальний набір функцій та залежностей, що дозволяє розробникам самостійно вибирати та встановлювати додаткові компоненти, не перевантажуючи додаток непотрібними функціями.

3. Гнучкість. Flask не накладає жорстких обмежень на структуру та архітектуру веб-додатків, що дозволяє розробникам самостійно обирати підхід до створення додатку та реалізовувати свої ідеї.

4. Підтримка розширень. Flask пропонує велику кількість розширень та інтеграцій з іншими технологіями, що дозволяє розширювати можливості додатку та спрощувати роботу з ним.

5. Відкритий код та активна спільнота користувачів.

3.1.4 Heroku

Heroku - це хмарна платформа, яка надає можливість розгортання та масштабування веб-додатків, написаних на різних мовах програмування, включаючи Python. Для розгортання додатків на Heroku можна використовувати різні стеки, такі як Flask, Django, Pyramid і т.д [34].

Основні переваги Heroku як сервісу для розгортування Python додатків:

1. Простота використання. Heroku надає простий та зручний інтерфейс для розгортання додатків, що дозволяє швидко розпочати роботу з платформою.

2. Масштабованість. Heroku може масштабувати додатки автоматично в залежності від навантаження, що дозволяє забезпечити стабільну роботу додатків навіть при великому навантаженні.

3. Інтеграція з GitHub. Heroku дозволяє підключити репозиторій на GitHub та автоматично розгорнути додаток при кожному коміті до гілки master.

4. Підтримка різних мов та середовищ. Heroku підтримує різні мови програмування та середовища, що дозволяє розгортати різноманітні додатки.

5. Налаштування середовища. Heroku надає можливість налаштувати середовище для розгортання додатків, що дозволяє забезпечити їхню стабільну роботу.

6. Безкоштовний тариф. Heroku надає безкоштовний тариф для розгортання додатків, що дозволяє випробувати платформу без великих витрат.

3.2 Опис програмної реалізації

Під час розробки та проектування проекту використовувалися різноманітні популярні технології та засоби розробки. Основні частини програмного застосунку:

- налаштування середовища на Heroku;
- створення резозиторію на GitHub та налаштування автоматичного розгортання на Heroku;
- створення програмного застосунку за допомогою фреймворку Flask;
- розміщення коду у репозиторії GitHub за допомогою системи контролю версій GIT;
- налагодження клієнт-сервер з'єднання за допомогою REST API.

Даний застосунок складається з декількох компонентів (рис. 3.1):

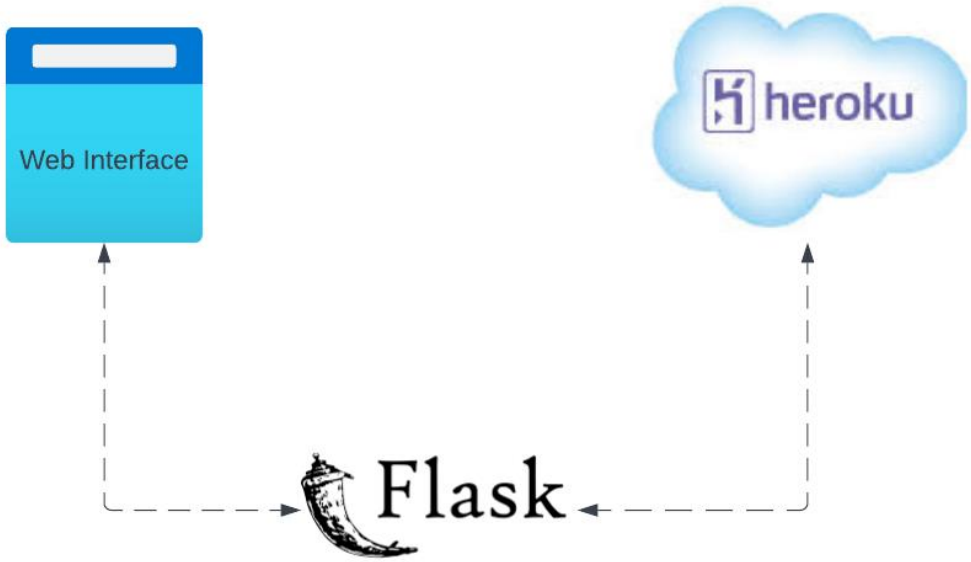

Рисунок 3.1 – Високорівневе представлення компонентів Python проекту

На цій діаграмі Web Interface (або Web Browser) є клієнтською стороною додатку, яка взаємодіє з Flask за допомогою HTTP запитів. Flask, у свою чергу, є серверним фреймворком, який отримує запити від Web Interface та генерує відповіді на них. Heroku- це платформа хостингу, яка дозволяє розміщувати та запускати веб-додатки, створені з використанням Flask та інших технологій.

Така архітектура може бути використана для створення та розгортання простих веб-додатків на Heroku. Наприклад, у разі використання Flask можна створити додаток, який надає користувачам можливість редагувати та зберігати свої замітки в базі даних. Flask дозволяє легко обробляти HTTP запити та взаємодіяти з базою даних. Після створення додатку його можна розмістити на Heroku, щоб забезпечити доступ до нього з Інтернету та додаткові можливості, такі як автоматична масштабованість та бекапи.

3.3 Структура проекту

Як відомо, від структури проекту залежить якість та швидкість розробки будь-якого додатку (табл. 3.1). Тому структуру проекту було організовано наступним чином (рис. 3.2):

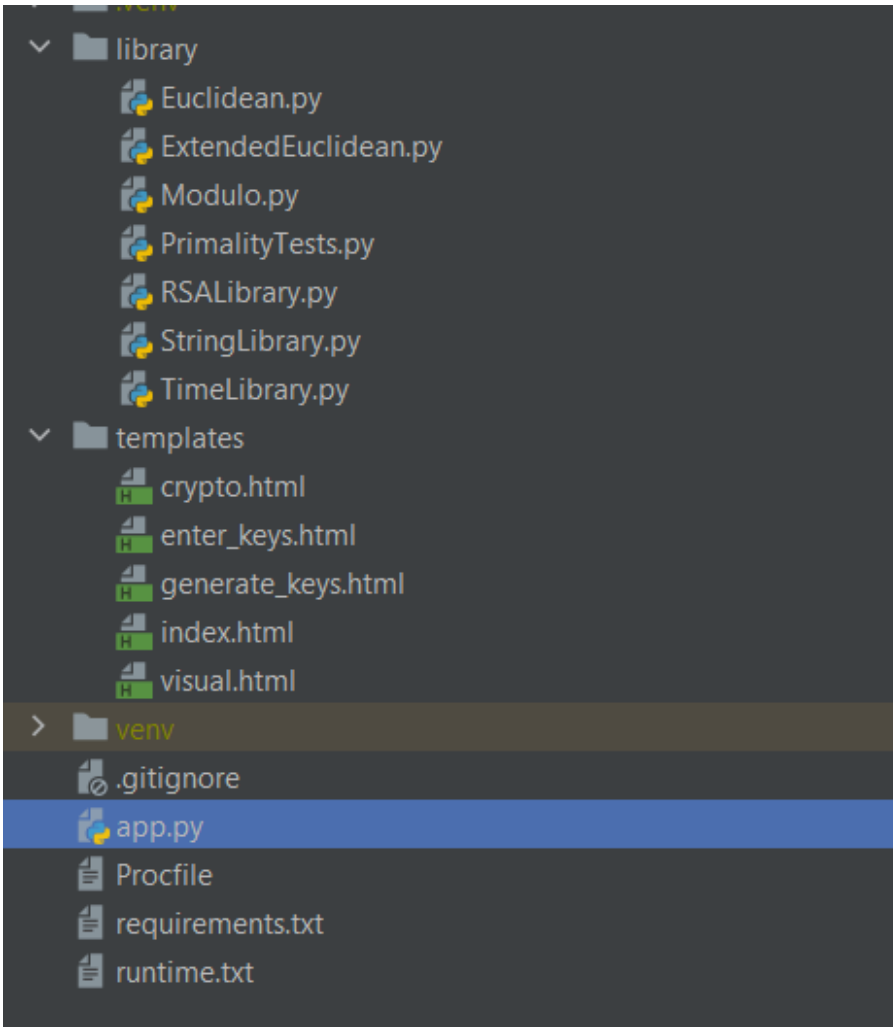

Рисунок 3.2 - Скріншот папки проекту у PyCharm

Таблиця 3.1

Папки та файли проекту

Об’єкт	Опис
Library	Це папка з модулями логіки застосунку
Templates	Папка з шаблонами веб сторінок
app.py	Основний файл проєкту, де зберігається маршрутизація проекту
Procfile	текствовий файл Heroku, котрий вказує яка команда запускає застосунок
requirements.txt	Файл з переліком бібліотек та їх версій, шо потрібні для роботи застосунку
runtime.txt	в цьому файлі визначається версія Python, котру слід встановити на сервер

3.4 RESTful веб-сервіс Flask

Зв'язка між клієнтом та сервером у сучасному веб-додатку відбувається за допомогою веб-сервісів, які забезпечують можливість обміну даними між ними. RESTful архітектура - одна з найпоширеніших архітектур для розробки веб-сервісів. У цьому підрозділі було розглянуто, як створити RESTful веб-сервіс з використанням Flask, який є одним з найпопулярніших веб-фреймворків для Python.

REST - це архітектурний стиль для розробки веб-сервісів, що базується на принципах HTTP-протоколу. RESTful веб-сервіси забезпечують доступ до ресурсів (наприклад, даних) за допомогою HTTP-методів, таких як GET, POST, PUT та DELETE (рис. 3.3).

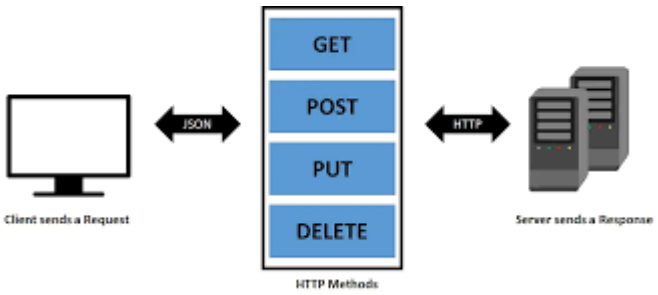

Рисунок 3.3 – Представлення RESTful

У RESTful веб-сервісах ресурси ідентифікуються за допомогою URI, а не за допомогою параметрів запиту, як у традиційних веб-додатках. Крім того, RESTful веб-сервіси повинні бути безстанними, що означає, що сервер не зберігає стан клієнта між запитами.

У Flask, декоратор @app.route() використовується для визначення URI та HTTP-методів, які будуть використовуватися для кожного ресурсу, доступного в веб-сервісі (рис. 3.4).

```
@app.route('/crypto', methods=['GET', 'POST'])
def crypto():
    if n <= 0 or (e <= 0 and d <= 0):
        return index()
    context = {
        'n': n,
        'e': e,
        'd': d,
        'p': p,
        'q': q,
    }
    return render_template('crypto.html', **context)
```

Рисунок 3.4 – Скріншот коду проекта, як приклад декоратора з GET, POST.

За допомогою методів HTTP, які відповідають різним типам запитів (GET, POST, PUT, DELETE), можна отримувати, додавати, оновлювати та видаляти ресурси з веб-сервісу.

Основні принципи RESTful веб-сервісу на Flask включають:

1. Використання URI для ідентифікації ресурсів.

2. Використання HTTP-методів для операцій з ресурсами.

3. Використання статус-кодів HTTP для позначення результатів операцій з ресурсами.

4. Надання ресурсів у форматі, що підтримується клієнтом (JSON, XML тощо).

5. Зберігання стану клієнта на клієнті, а не на сервері.

6. Відсутність стану на сервері (stateless), що означає, що кожен запит обробляється окремо.

7. Використання кешування, щоб зменшити навантаження на сервер.

3.5 Архітектура Flask додатку

Розглянемо архітектуру Flask додатку з компонентами Web Browser, Routers, Functions та Templates на діаграмі компонентів (рис. 3.5).

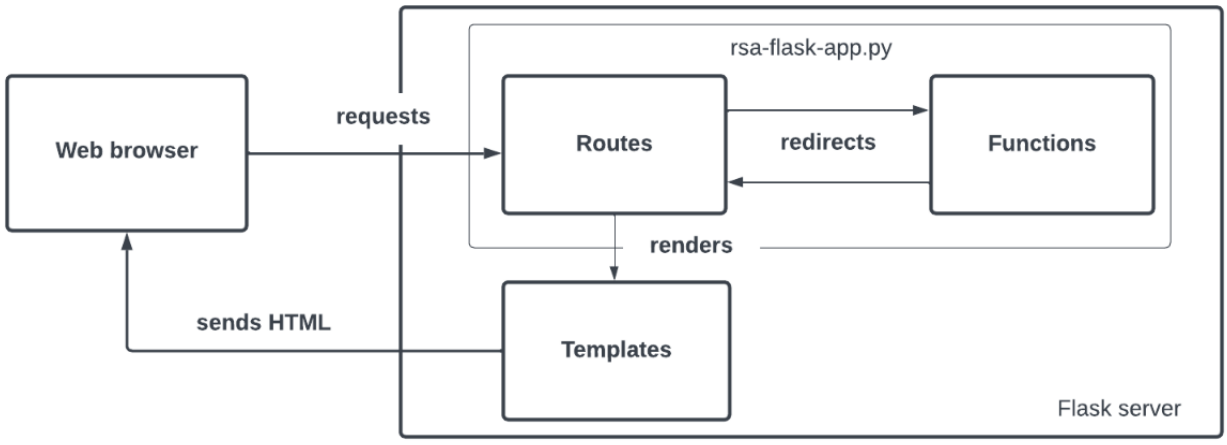

Рисунок 3.5 – Діаграма компонентів додатку

На діаграмі компонентів можна виділити чотири компоненти: Web Browser, Routers, Functions та Templates. Web Browser є клієнтською стороною додатку та надсилає запити до веб-додатку. Routers відповідає за маршрутизацію запитів та передачу кожного запиту до відповідної функції на Flask Server. Функції містять логіку додатку та генерують відповідь на запит. Templates відповідає за візуальне відображення даних від Flask Server на стороні клієнта (табл. 3.2):

.

Таблиця 3.2

Компоненти та їх застосування

Компонент	Його застосування
Web Browser	Це клієнтська сторона додатку, яка відображає веб-сторінки та надсилає запити до сервера. Браузери зазвичай використовують протокол HTTP або HTTPS для звернення до сервера та передачі запитів
Routers	Routers відповідає за маршрутизацію запитів та передачу їх до відповідних функцій на Flask Server. Роутери визначають, який URL-адрес буде відповідати кожній функції, що викликається в додатку. Для визначення маршрутів Flask використовує декоратори.
Functions	Функції містять логіку додатку та генерують відповідь на запит. Функції можуть приймати дані від клієнта та передавати їх до шаблонів для відображення на стороні клієнта. Flask використовує вбудований шаблонізатор Jinja2 для відображення даних на веб-сторінках.
Templates	Templates відповідає за візуальне відображення даних від Flask Server на стороні клієнта. Шаблони зазвичай містять HTML-код та директиви Jinja2 для відображення даних, переданих з функцій. Flask дозволяє використовувати різні шаблони для різних маршрутів.

Компонент	Його застосування
Flask Server	Flask Server відповідає за обробку запитів від клієнтів, маршрутизацію цих запитів до відповідних функцій, виконання логіки додатку та генерацію відповідей на запити. Flask Server також відповідає за збереження та управління станом додатку. В загальному, Flask Server є центральною частиною додатку, яка обслуговує запити від клієнтів та взаємодіє з іншими компонентами, такими як бази даних, інші сервіси, зовнішні API тощо. Flask Server використовує фреймворк Flask для реалізації веб-додатку та допоміжних бібліотек та інструментів для розробки.

Загалом, архітектура Flask додатку досить проста та добре структурована, що робить його легко зрозумілим та розширюваним. Flask дозволяє легко створювати RESTful API та веб-додатки, які можна розгорнути на різних платформах. З використанням вбудованого сервера Flask, додаток може бути запущений локально для тестування та розробки, або можна використовувати різні сервери розгортання, такі як Gunicorn або WSGI.

Усі компоненти архітектури Flask додатку працюють разом, щоб забезпечити функціональність веб-додатку. Web Browser надсилає запити до Routers, який знаходить відповідну функцію на Flask Server та передає їй запит. Функції генерують відповідь та передають її до Templates для відображення на стороні клієнта. Ця взаємодія між компонентами дозволяє Flask додатку працювати як повноцінний веб-сервіс.

3.6 Проектування мовою UML та опис діаграми послідовностей

Проектування програмного забезпечення може бути досить складним процесом, який вимагає детального планування та розуміння взаємодії різних

компонентів системи. Одним із інструментів, що допомагають у процесі проектування є мова UML, яка дозволяє створювати діаграми, що демонструють взаємодію між компонентами системи.

Діаграма послідовностей є однією з найбільш поширених діаграм в мові UML. Ця діаграма відображає взаємодію між об'єктами системи на основі послідовності викликів методів між ними. Діаграма послідовностей дозволяє показати послідовність дій, які відбуваються між об'єктами, а також виклики методів, що відбуваються між ними.

Давайте розглянемо діаграму послідовностей, яка демонструє взаємодію між користувачем, Flask додатком та модулем шифрування/дешифрування (рис. 3.6).

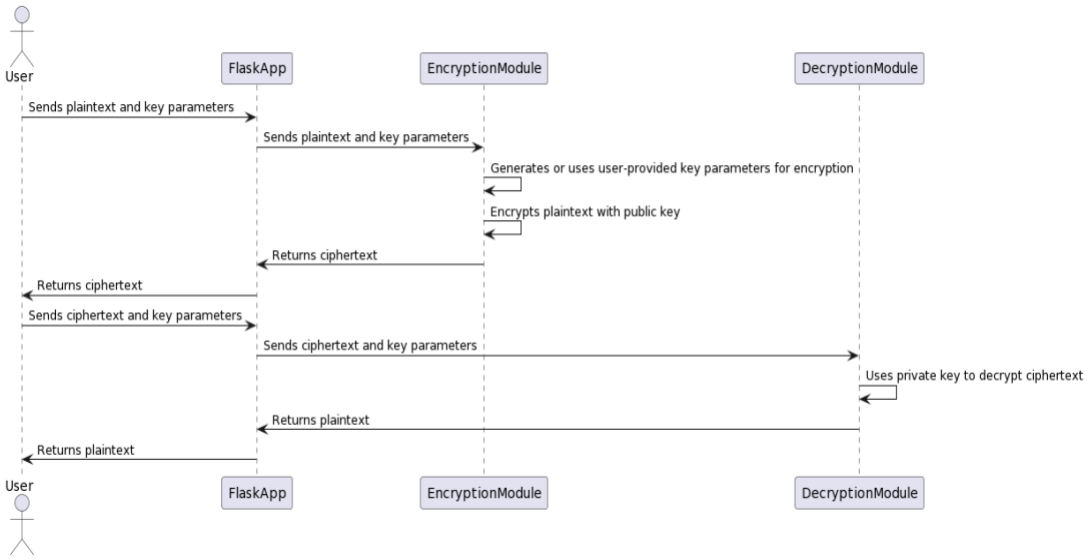

Рисунок 3.6 – Діаграма послідовностей у нотації UML

На діаграмі зображено взаємодію між трьома об'єктами: користувачем, Flask додатком та модулем шифрування/дешифрування. Користувач відправляє текст та ключ шифрування до Flask додатку через веб-інтерфейс. Далі Flask додаток передає ці дані до модуля шифрування/дешифрування. Продовжуючи про тему проектування мовою UML, діаграма послідовностей є однією з основних діаграм, яка використовується для моделювання взаємодії між об'єктами в системі. Діаграма послідовностей показує послідовність повідомлень, які передаються між об'єктами системи, що допомагає візуалізувати процес взаємодії між ними.

Ця діаграма відображає процес шифрування та дешифрування повідомлень в системі за допомогою Flask додатку.

На діаграмі є два актори: User та FlaskApp. User є користувачем системи, який відправляє повідомлення на FlaskApp, щоб його зашифрувати та відправити. FlaskApp є додатком, який отримує повідомлення від користувача, шифрує його та повертає зашифроване повідомлення користувачеві.

У процесі шифрування повідомлення, User відправляє plaintext та key parameters на FlaskApp. FlaskApp передає ці параметри на EncryptionModule, який генерує ключі для шифрування. Потім EncryptionModule використовує публічний ключ для шифрування повідомлення та повертає зашифрований текст на FlaskApp. FlaskApp повертає зашифрований текст користувачеві.

У процесі дешифрування повідомлення, User відправляє зашифрований текст та key parameters на FlaskApp. FlaskApp передає ці параметри на DecryptionModule, який використовує приватний ключ для дешифрування повідомлення та повертає plaintext на FlaskApp. FlaskApp повертає plaintext користувачеві.

Ця діаграма показує взаємодію між різними компонентами системи в процесі шифрування та дешифрування повідомлень.

Діаграма послідовностей є дуже корисним інструментом для проектування програмного забезпечення з орієнтацією на об'єкти, оскільки вона показує взаємодію між об'єктами в часовому порядку. Це дозволяє програмістам відобразити послідовність дій, необхідних для виконання певного завдання, і виявити потенційні помилки в алгоритмах.

У діаграмі послідовностей, запити від користувача та відповіді від сервера відображаються як стрілки між актором та об'єктами. У даному випадку, актором є користувач, який взаємодіє з Flask додатком, надсилаючи параметри для шифрування та розшифрування. Flask додаток відповідає на запити користувача, взаємодіючи з модулем шифрування та розшифрування.

Діаграма показує, що користувач надсилає текстові параметри та параметри ключа на Flask додаток. Flask додаток відправляє ці параметри модулю шифрування та розшифрування, який генерує або використовує ключ для шифрування переданого

тексту. Після шифрування, модуль повертає зашифрований текст на Flask додаток, який повертає його користувачеві.

Далі, користувач надсилає зашифрований текст та параметри ключа на Flask додаток для розшифрування. Flask додаток відправляє ці параметри модулю розшифрування, який використовує приватний ключ для розшифрування переданого тексту та повертає його на Flask додаток. Після цього, Flask додаток повертає розшифрований текст користувачеві.

3.7 Розгортання Flask додатку на платформі Heroku

Розглянемо процес розгортування Python проєкту на платформі Heroku за допомгою загальної діаграми (рис. 3.7).

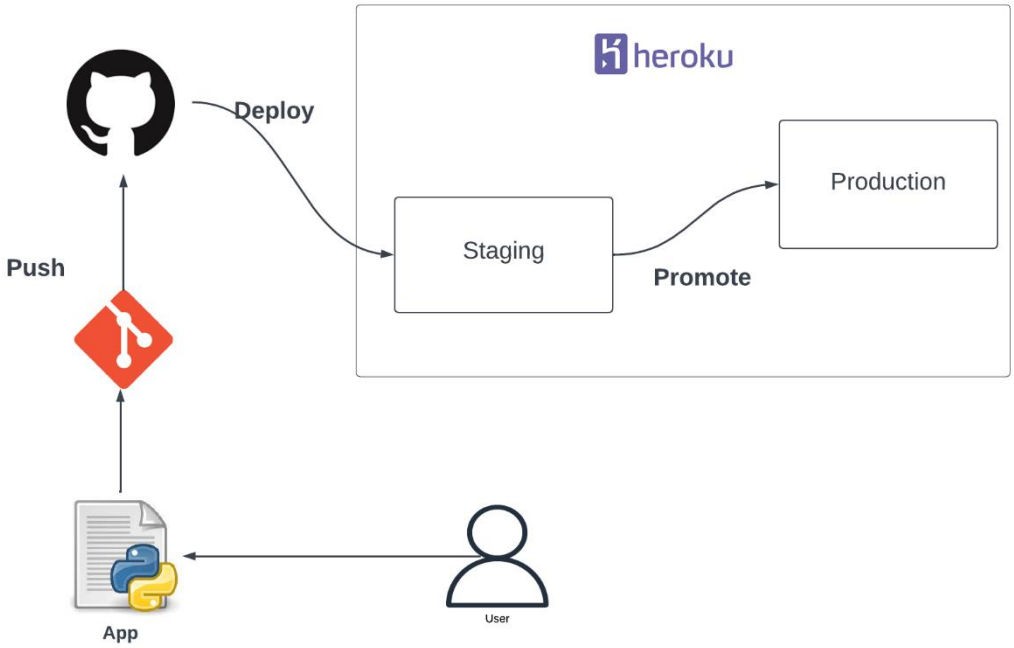

Рисунок 3.7 - Послідовність розгортування проекту

Основний процес розгортування Flask додатку на Heroku можна описати таким чином:

1. Створення акаунту на Heroku.
2. Ініціалізація git репозиторію в кореневій директорії проекту.

3. Створення requirements.txt файлу з переліком усіх необхідних залежностей та runtime.txt файлу з вказанням версії Python.

4. Створення файлу Procfile, в якому вказується команда для запуску додатку на Heroku.

5. Коміт та пуш проекту в GitHub репозиторій.

6. Створення нового додатку на Heroku через веб-інтерфейс.

7. Налаштування змінних середовища для додатку на Heroku через веб-інтерфейс.

8. Підключення до GitHub репозиторію веб-інтерфейсом Heroku.

9. Деплой додатку на Heroku через веб-інтерфейс (рис. 3.8).

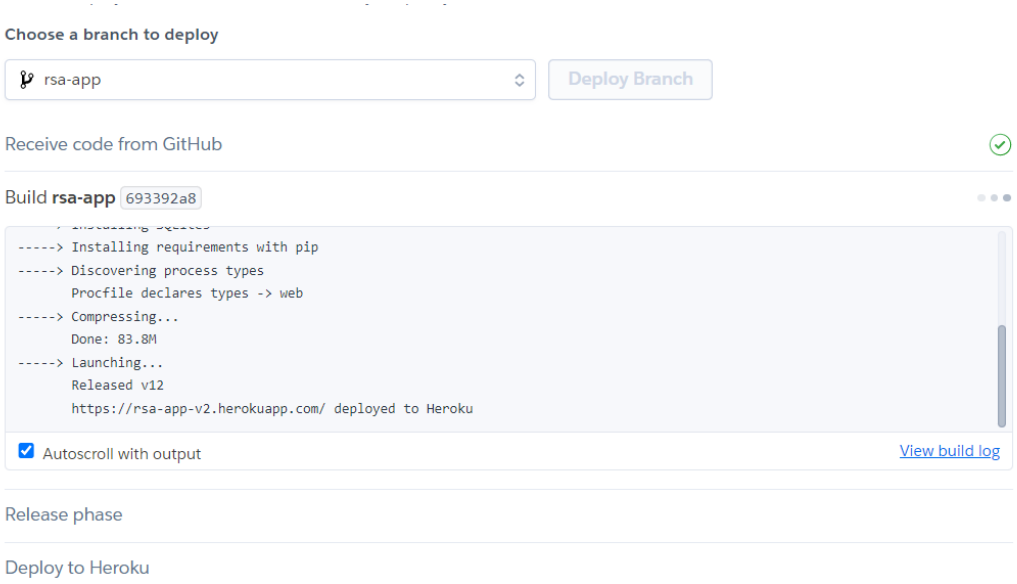

Рисунок 3.8 - Скріншот з Heroku

10. Перевірка статусу деплою та логів додатку через веб-інтерфейс(рис. 3.9).

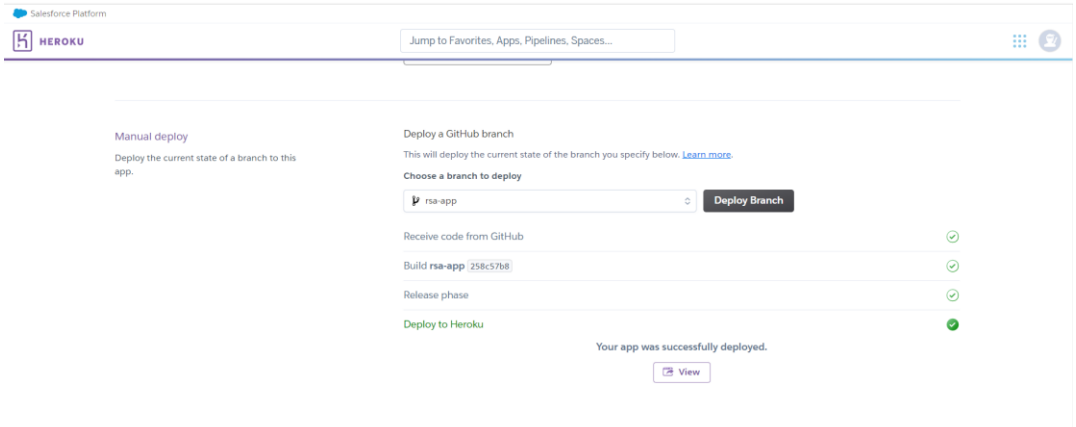

Рисунок 3.9 - Скріншот з Heroku

Якщо деплой успішний, додаток готовий до використання за адресою, яку можна знайти в веб-інтерфейсі Heroku (рис. 3.10).

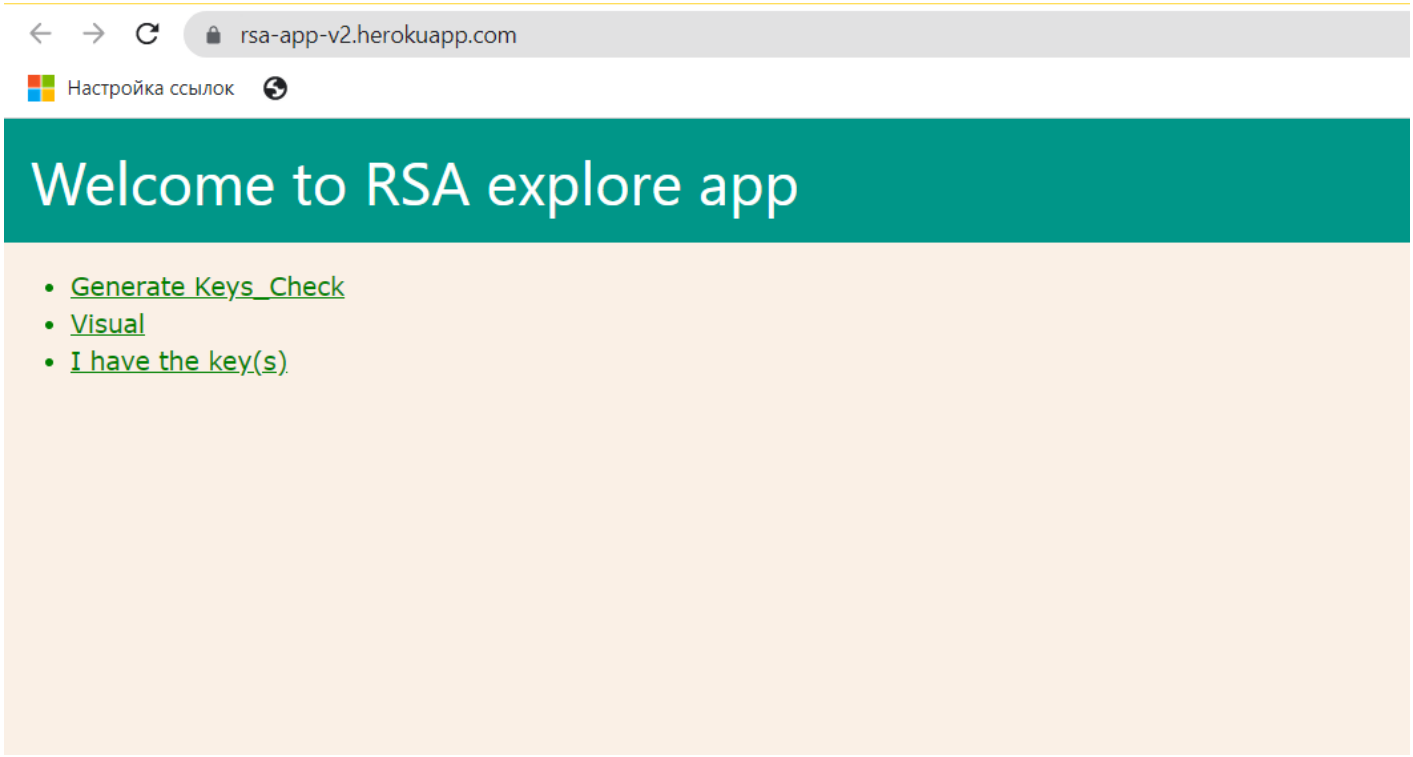

Рисунок 3.10 – Головна сторінка додатку

3.7.1 Валідація та верифікація роботи додатку

Для огляду та перевірки логіки у додатку (FlaskApp) створено два сценарії використання - шифрування повідомлення та отримання розшифрованого повідомлення.

1. Шифрування повідомлення

Актор: Користувач

Опис: Цей use case описує сценарій, в якому користувач шифрує повідомлення, використовуючи FlaskApp та EncryptionModule.

Передумови: FlaskApp та EncryptionModule встановлені та запущені.

Наслідки: Текстове повідомлення шифрується та повертається користувачеві.

Основні кроки:

– Користувач генерує або додає свої значення параметрів для ключів.

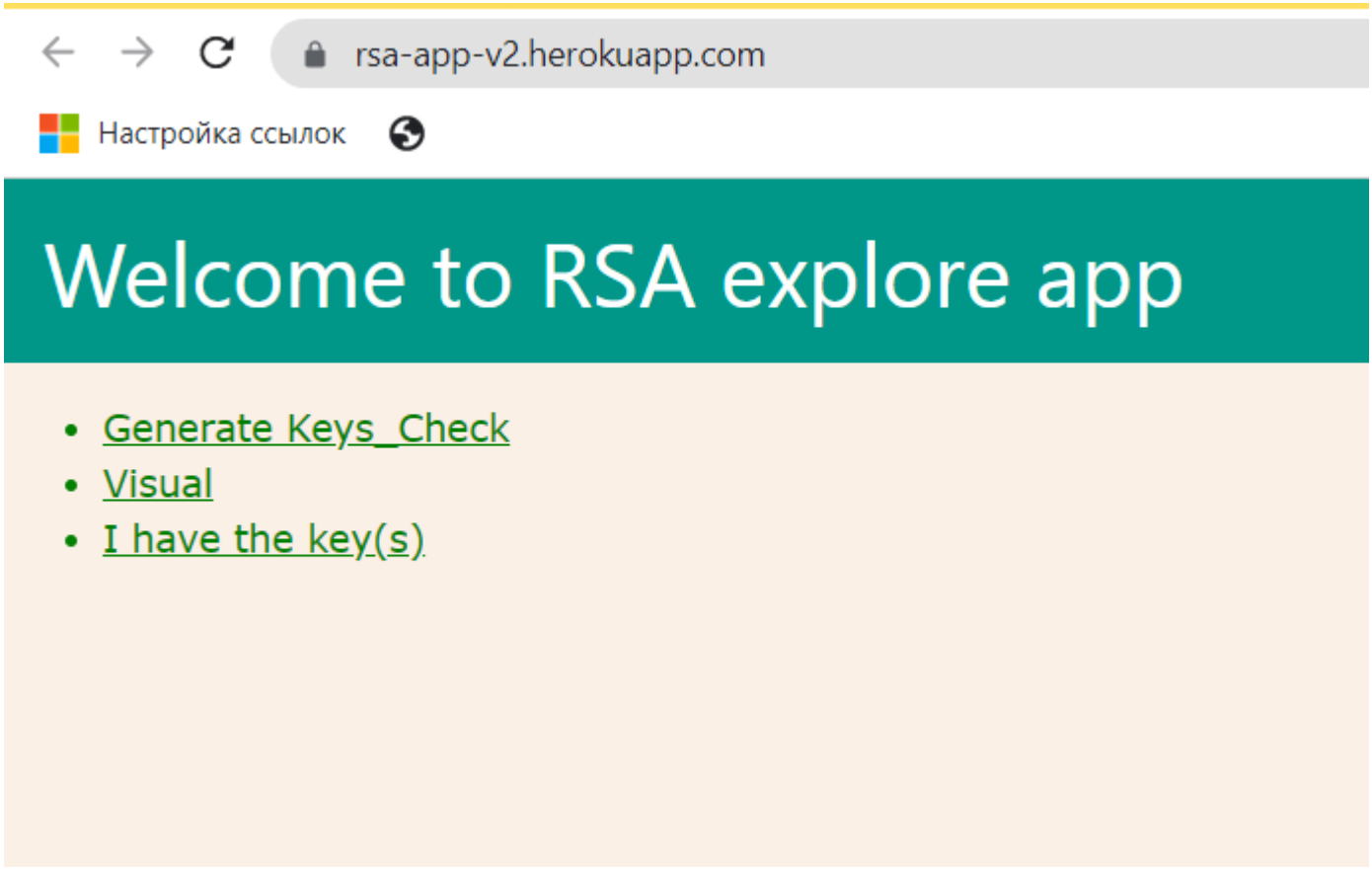

Рисунок 3.11 – Головний сторінка додатку

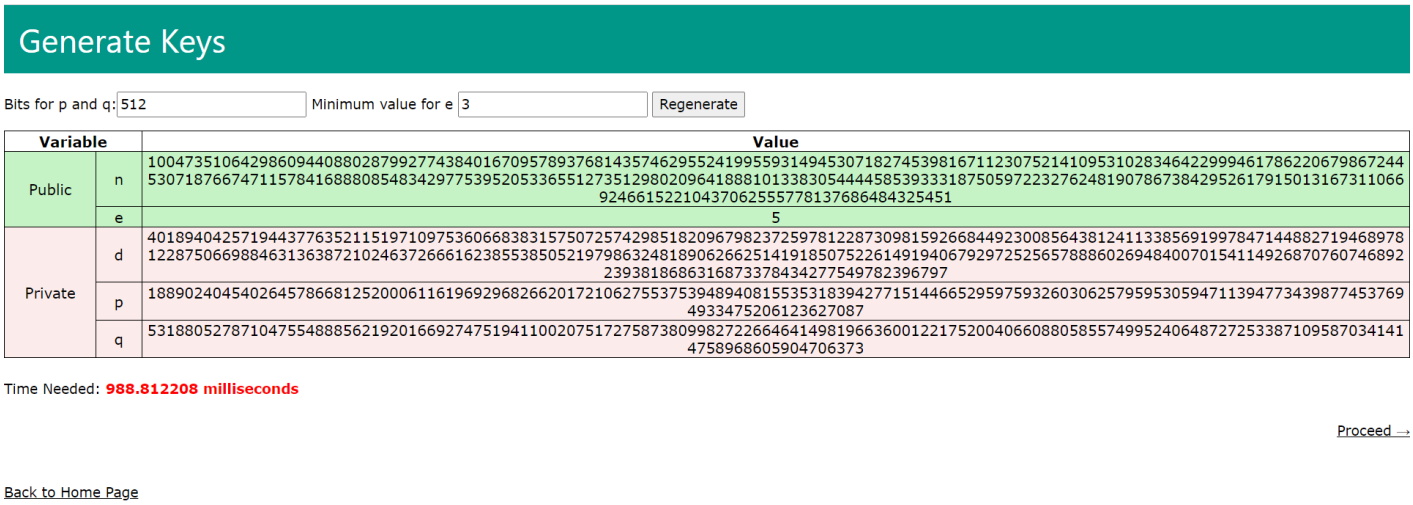

Рисунок 3.12 – Сторінка на якій користувач може згенерувати параметри для створення ключів

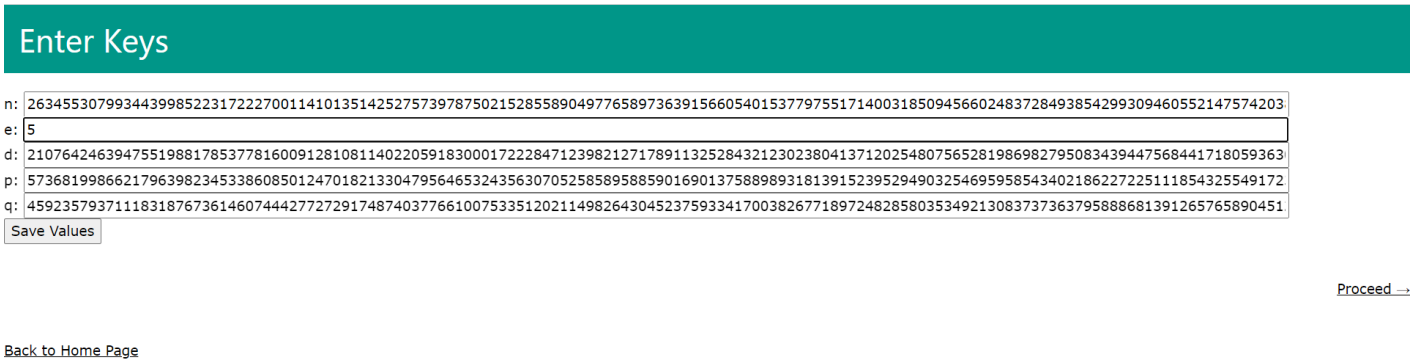

Рисунок 3.13 – Сторінка на якій користувач може самостійно ввести параметри

– Користувач відправляє повідомлення до FlaskApp.

– FlaskApp відправляє повідомлення та параметри ключів до EncryptionModule.

– EncryptionModule використовує згенеровані або надані користувачем параметри ключів для шифрування.

– EncryptionModule шифрує повідомлення з використанням публічного ключа.

– EncryptionModule повертає зашифроване повідомлення до FlaskApp.

– FlaskApp повертає зашифроване повідомлення користувачеві.

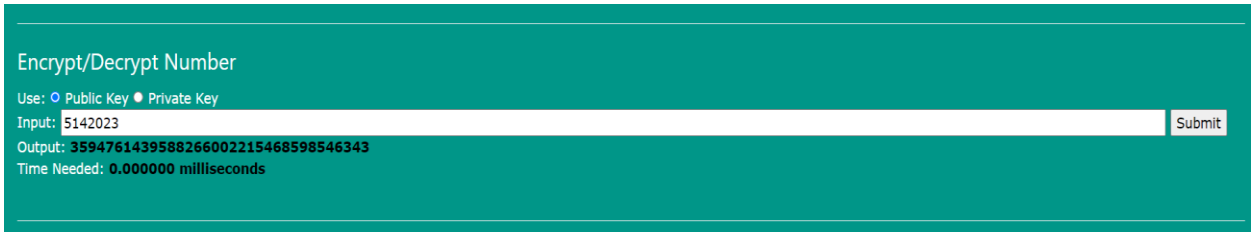

Рисунок 3.14 – Користувач зашифрував повідомлення за допомогою Public Key

2. Дешифрування повідомлення

Назва: Користувач розшифровує повідомлення

Актор: Користувач

Опис: Цей use case описує сценарій, в якому користувач розшифровує криптограму, використовуючи FlaskApp та DecryptionModule.

Передумови: FlaskApp та DecryptionModule встановлені та запущені.

Наслідки: Зашифроване повідомлення розшифровується та повертається користувачеві.

Основний потік:

– Користувач відправляє криптограму та ключові параметри до FlaskApp.

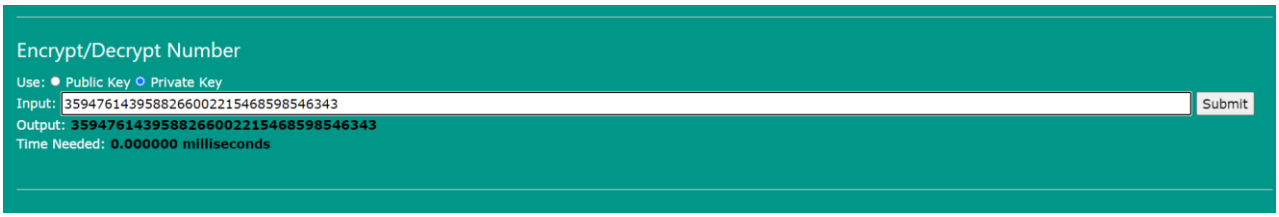

Рисунок 3.15 – Користувач ввів зашифроване повідомлення та обрав Private Key

– FlaskApp відправляє зашифроване повідомлення та параметри ключів до DecryptionModule.

– DecryptionModule використовує приватний ключ для розшифрування криптограми.

– DecryptionModule повертає розшифроване повідомлення до FlaskApp.

– FlaskApp повертає розшифроване повідомлення користувачеві.

Рисунок 3.16 – Користувач розшифрував зашифроване повідомлення за допомогою Private Key

3.7.2 Огляд функціоналу додатку

1. Користувач може зашифрувати та розшифрувати чисельне повідомлення, а також порівняти результати дешифрування RSA з розробленим CRT RSA. Як бачимо, CRT RSA має менший час роботи (рис. 3.17).

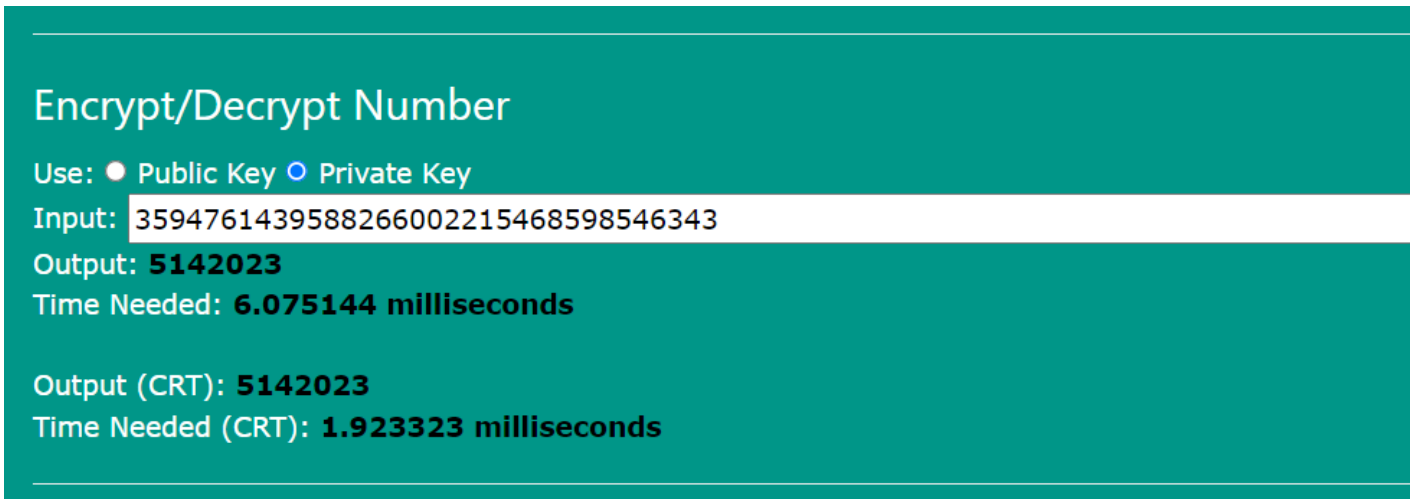

Рисунок 3.17 – Скріншот з Python додатку

3. Користувач може зашифрувати текстове повідомлення (рис. 3.18).

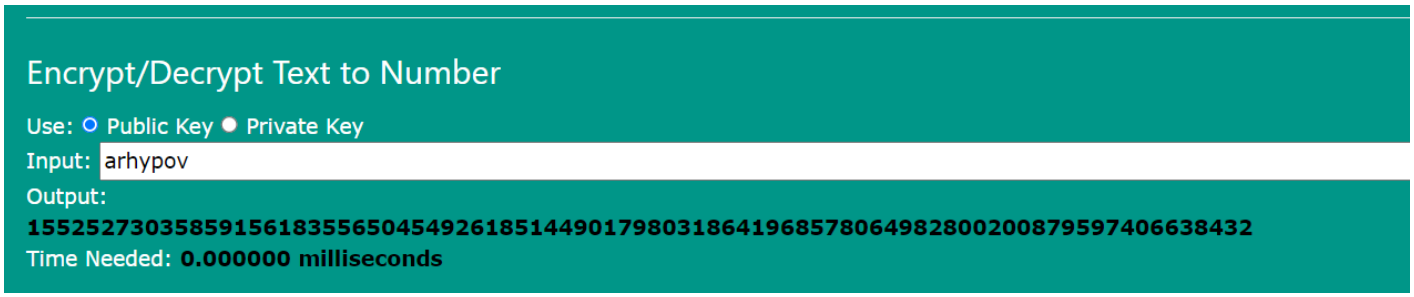

Рисунок 3.18 – Скріншот з Python додатку

4. Користувач може розшифрувати зашифроване повідомлення (рис. 3.19).

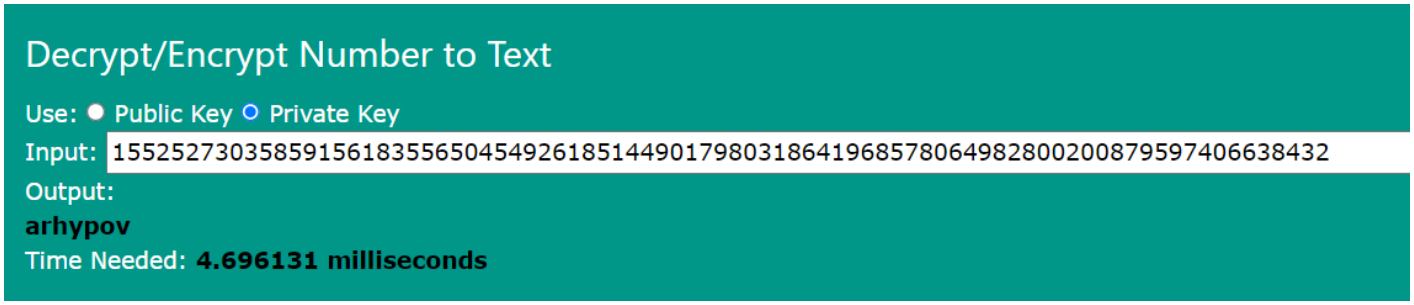

Рисунок 3.19 – Скріншот з Python додатку

Висновки за розділом 3

В даному розділі розглянуто представлення програмного засобу, його роботу та процес розгортання на хмарній платформі. Було розглянуто та проаналізовано засоби та підходи для розробки веб-додатку. Запропоновано до використання середовище розробки PyCharm, платформу для зберігання коду GitHub, фреймворк Flask та

хмарну платформу для хостінгу Heroku. Формалізовано архітектуру та взаємодію компонентів веб-додатку, розглянуто сценарії використання програми. Проведено тестування веб-додатку на працездатність.

ВИСНОВКИ

У кваліфікаційній роботі одержано наступні теоретичні та практичні результати:

1) Розглянуто та виконано порівняльний аналіз існуючих методів та підходів криптографічного захисту інформації на ОІД. Встановлено, що застосування криптографічних методів захисту допомагає забезпечити безпеку та конфіденційність інформації на об'єктах інформаційної діяльності, зменшити ризики несанкціонованого доступу та витоку даних. У зв'язку зі стрімким розвитком технологій та збільшенням кількості атак на інформаційні системи. Виникає великий попит на використання різних методів криптографічного шифрування, що дозволяють захистити інформаційні системи.

2) Було встановлено актуальність формалізації та створення веб-додатку для шифрування та дешифрування повідомлень.

3) Було протестовано різновиди RSA за п'ятьма характеристиками: довжина ключа, генерація простого числа, генерація закритого ключа, кодування та декодування. Основним параметром безпеки сучасних криптографічних алгоритмів є довжина ключа. Великий ключ забезпечить високий рівень безпеки, але операції також займатимуть багато часу і споживатимуть багато пам'яті. Але ця довжина, ймовірно, рано чи пізно буде занадто короткою. У нинішній час потрібні алгоритми, які з меншим розміром ключа забезпечують максимальну безпеку і стійкість для фреймворків, де обчислювальна потужність є надзвичайно низькою. Тому в поточній ситуації потрібна більш домінуюча варіація алгоритму RSA.

4) Запропоновано створення веб-додатку для шифрування та дешифрування повідомлень.

5) Проведено імплементацію математичного та програмного забезпечення для веб-додатку.

6) Виконано валідацію розробленого програмного забезпечення веб-додатку.

7) Визначено та запропоновано шляхи подальших досліджень криптографічного захисту інформації на ОІД:

- Верифікація та валідація математичного та програмного забезпечення веб-додатку в реальному середовищі використання.
- Розробка наступного покоління математичного та програмного забезпечення для покращення швидкості та надійності криптографічного алгоритму.
- Покращення та розширення компонентів веб-додатку. Для цього необхідно провести дослідження метрик для порівняння криптографічних алгоритмів та їх модифікацій, можливостей додавання нових модифікацій криптографічних алгоритмів для покращення роботи застосунку.

Мету роботи досягнуто, поставлені задачі виконано.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Богуш В.М. Захист інформації в телекомунікаційних системах та мережах. Конспект лекцій – К.: ДУІКТ. 2004. – 350 с.

2. Павлов І.М. Модель процесу роботи комплексної системи захисту інформації в спеціальних інформаційно – телекомунікаційних системах та вимоги до неї по захищеності інформації // Збірник наукових праць ВІТІ НТУУ «КПІ». – Київ, 2006. – № 3. – С. 82 – 91.

3. Толюпа С. В., Іванова О. М., Демченко І. О. Підходи до проектування та оцінки ефективності системи захисту інформації в автоматизованих системах обробки та передачі даних //Сучасний захист інформації. – 2013. – №. 1. – С. 25-30.

4. Порядок проведення робіт із створення комплексної системи захисту інформації в інформаційно – телекомунікаційної системі // НД ТЗІ 3.7 – 003 – 05. – Київ, 2005. – 35 с.

5. Cheng Z. et al. Algorithm for elliptic curve Diffie-Hellman key exchange based on DNA tile self-assembly //Journal of Computational and Theoretical Nanoscience. – 2010. – Т. 7. – №. 5. – С. 856-861.

6. Coppersmith D. The Data Encryption Standard (DES) and its strength against attacks //IBM journal of research and development. – 1994. – Т. 38. – №. 3. – С. 243-250.

7. Ko M., Osei-Bryson K. M., Dorantes C. Investigating the impact of publicly announced information security breaches on three performance indicators of the breached firms //Information Resources Management Journal (IRMJ). – 2009. – Т. 22. – №. 2. – С. 1-21.

8. ENISA, “Securing personal data. Recommended cryptographic measures. 2013. [Електрониий ресурс]. – Режим доступу: https://www.enisa.europa.eu/publications/recommended-cryptographic-measures-securing personal-data.

9. Rubinstein-Salzedo S. Cryptography. – Cham, Switzerland : Springer, 2018. – Т. 260.

10. Stallings W. Data and computer communications. – Pearson Education India, 2007.

11. Shanta J. V. Evaluating the performance of symmetric key algorithms: AES (advanced encryption standard) and DES (data encryption standard) //IJCEM International Journal of Computational Engineering & Management. – 2012. – Т. 15. – №. 4. – С. 43-49.

12. Marwaha M. et al. Comparative analysis of cryptographic algorithms //Int J Adv Engg Tech/IV/III/July-Sept. – 2013. – Т. 16. – С. 18.

13. Patil P. et al. A comprehensive evaluation of cryptographic algorithms: DES, 3DES, AES, RSA and Blowfish //Procedia Computer Science. – 2016. – Т. 78. – С. 617-624.

14. Sanap S. D., More V. Performance analysis of encryption techniques based on avalanche effect and strict avalanche criterion //2021 3rd International Conference on Signal Processing and Communication (ICPSC). – IEEE, 2021. – С. 676-679.

15. Архипов Я. Порівняльне дослідження алгоритму RSA та його різновидів/ Я. Архипов, С. Толюпа // Проблеми кібербезпеки інформаційно-телекомунікаційних систем: Збірник матеріалів доповідей та тез; м. Київ, 27 квітня 2023 року; Київський національний університет імені Тараса Шевченка / Редкол.: В.В. Ільченко, д.ф-м.н., проф., (голова); та ін. – К.: ВПЦ "Київський університет", 2023. – 166 с. (с.86-87).

16. Popescul D. The confidentiality–integrity–accessibility triad into the knowledge security. A reassessment from the point of view of the knowledge contribution to innovation //Proceedings of the 16th international business information management association conference (innovation and knowledge management, a global competitive advantage). – 2011. – С. 1338-1345.

17. Rivest R. L., Shamir A., Adleman L. A method for obtaining digital signatures and public-key cryptosystems //Communications of the ACM. – 1978. – Т. 21. – №. 2. – С. 120-126.

18. Overmars A., Venkatraman S. A fast factorisation of semi-primes using sum of squares //Mathematical and Computational Applications. – 2019. – Т. 24. – №. 2. – С. 62.

19. Shor P. W. Algorithms for quantum computation: discrete logarithms and factoring //Proceedings 35th annual symposium on foundations of computer science. – Ieee, 1994. – С. 124-134.

20. Aboud S. J. et al. An efficient RSA public key encryption scheme //Fifth International Conference on Information Technology: New Generations (itng 2008). – IEEE, 2008. – С. 127-130.

21. Pointcheval D. New public key cryptosystems based on the dependent-RSA problems //Advances in Cryptology—EUROCRYPT'99: International Conference on the Theory and Application of Cryptographic Techniques Prague, Czech Republic, May 2–6, 1999 Proceedings 18. – Springer Berlin Heidelberg, 1999. – С. 239-254.

22. Boneh D., Franklin M. Efficient generation of shared RSA keys //Advances in Cryptology—CRYPTO'97: 17th Annual International Cryptology Conference Santa Barbara, California, USA August 17–21, 1997 Proceedings 17. – Springer Berlin Heidelberg, 1997. – С. 425-439.

23. De Vries A. The ray attack, an inefficient trial to break RSA cryptosystems //arXiv preprint cs/0307029. – 2003.

24. Hinek M. J., Low M. K., Teske E. On some attacks on multi-prime RSA //Selected Areas in Cryptography: 9th Annual International Workshop, SAC 2002 St. John's, Newfoundland, Canada, August 15–16, 2002 Revised Papers 9. – Springer Berlin Heidelberg, 2003. – С. 385-404.

25. Garg D., Verma S. Improvement over public key cryptographic algorithm //2009 IEEE International Advance computing conference. – IEEE, 2009. – С. 734-739.

26. Rathod U., Sreenivas S., Chandavarkar B. R. Comparative Study Between RSA Algorithm and Its Variants: Inception to Date //ICCCE 2020: Proceedings of the 3rd International Conference on Communications and Cyber Physical Engineering. – Singapore : Springer Singapore, 2020. – С. 139-149.

27. Wulansari D., Muslim M. A., Sugiharti E. Implementation of RSA algorithm with chinese remainder theorem for modulus n 1024 bit and 4096 bit //International Journal of Computer Science and Security. – 2016. – Т. 10. – №. 5. – С. 186-194.

28. Sarkar S., Maitra S. Cryptanalytic results on ‘dual crt’and ‘common prime’RSA //Designs, codes and cryptography. – 2013. – Т. 66. – С. 157-174.

29. Itoh K., Kunihiro N., Kurosawa K. Small secret key attack on a variant of RSA (due to Takagi) //Topics in Cryptology–CT-RSA 2008: The Cryptographers’ Track at the RSA Conference 2008, San Francisco, CA, USA, April 8-11, 2008. Proceedings. – Springer Berlin Heidelberg, 2008. – С. 387-406.

30. PyCharm. Документація. [Електронний ресурс]. – Режим доступу: https://www.jetbrains.com/help/pycharm/python.html.

31. Git. Документація. [Електронний ресурс]. – Режим доступу: https://git-scm.com/doc.

32. GitHub. Документація. [Електронний ресурс]. – Режим доступу: https://docs.github.com/en.

33. Flask. Документація. [Електронний ресурс]. – Режим доступу: https://flask.palletsprojects.com/en/2.3.x/.

34. Heroku. Документація. [Електронний ресурс]. – Режим доступу: https://devcenter.heroku.com/categories/reference.

ДОДАТОК А

App.py

```
#!/usr/bin/env python3
import json
import sys
import pandas as pd
import json
import plotly
import plotly.express as px
from flask import Flask, render_template, request
import plotly.graph_objs as go
from library.TimeLibrary import timed_function_beautified
sys.path.append('/library')
import library.RSALibrary as RSALibrary
from library.TimeLibrary import timed_function_beautified
app = Flask(__name__)
n, e, d, p, q = 0, 0, 0, 0, 0
bits = 512
min_e = 3
@app.route('/')
def index():
    global n, e, d, min_e, bits
    context = {
        'n': n,
        'e': e,
        'd': d,
        'min_e': min_e,
        'bits': bits
```

```
    }
    return render_template('index.html', **context)
@app.route('/genkeys_pre')
def generate_keys_pre():
    global n, e, d, min_e, bits, p, q
    min_e_param = request.args.get('min_e', '').strip()
    bits_param = request.args.get('bits', '').strip()
    if min_e_param:
        min_e = int(min_e_param)
    if bits_param:
        bits = int(bits_param)
    # (n, e, d), time_needed = timed_function_beautified(RSALibrary.generate_keys, bits,
min_e)
    context = {
        'n': n,
        'e': e,
        'd': d,
        'p': p,
        'q': q,
        'min_e': min_e,
        'bits': bits
    }
    return render_template('generate_keys.html', **context)
@app.route('/genkeys')
def generate_keys():
    global n, e, d, min_e, bits, p, q
    min_e_param = request.args.get('min_e', '').strip()
    bits_param = request.args.get('bits', '').strip()
    if min_e_param:
        min_e = int(min_e_param)
```

```
    if bits_param:
        bits = int(bits_param)
    if bits > 2048:
        raise Exception("Max bits is 2048")
    (n, e, d, p, q), time_needed = timed_function_beautified(RSALibrary.generate_keys, bits,
min_e, True)
    context = {
        'n': n,
        'e': e,
        'd': d,
        'p': p,
        'q': q,
        'min_e': min_e,
        'time_needed': time_needed,
        'bits': bits
    }
    return render_template('generate_keys.html', **context)
@app.route('/enterkeys', methods=['GET', 'POST'])
def enter_keys():
    global n, e, d, min_e, bits, p, q
    saved = False
    if request.method == 'POST':
        n = int(request.form.get('n'))
        e = int(request.form.get('e'))
        d = int(request.form.get('d'))
        p = int(request.form.get('p'))
        q = int(request.form.get('q'))
        saved = True
    context = {
        'n': n,
```

```
        'e': e,
        'd': d,
        'p': p,
        'q': q,
        'saved': saved
    }
    return render_template('enter_keys.html', **context)
@app.route('/crypto', methods=['GET', 'POST'])
def crypto():
    if n <= 0 or (e <= 0 and d <= 0):
        return index()
    context = {
        'n': n,
        'e': e,
        'd': d,
        'p': p,
        'q': q,
    }
    return render_template('crypto.html', **context)
@app.route('/crypto_number', methods=['GET', 'POST'])
def crypto_number():
    if request.method != 'POST':
        return crypto()
    input = int(request.form.get('input')) if request.form.get('input') else 0
    keytype = request.form.get('keytype')
    number_output_crt, time_needed_crt = None, None
    if keytype != 'private':
        keytype = 'public'
        # number_output = RSALibrary.encrypt_number(n, e, input)
        number_output,                                    time_needed                                    =
```

```
timed_function_beautified(RSALibrary.encrypt_number, n, e, input)
    else:
        number_output, time_needed = timed_function_beautified(RSALibrary.encrypt_number, n, d, input)
        if p > 0 and q > 0:
            number_output_crt, time_needed_crt = timed_function_beautified(RSALibrary.decrypt_number_crt, d, p, q,  input)
    context = {
        'n': n,
        'e': e,
        'd': d,
        'p': p,
        'q': q,
        'input': input,
        'number_output': number_output,
        'time_needed': time_needed,
        'number_output_crt': number_output_crt,
        'time_needed_crt': time_needed_crt,
        'keytype': keytype
    }
    return render_template('crypto.html', **context)
@app.route('/crypto_text', methods=['GET', 'POST'])
def crypto_text():
    if request.method != 'POST':
        return crypto()
    input_text = request.form.get('input_text')
    keytype = request.form.get('keytype')
    if keytype != 'private':
        keytype = 'public'
        text_output, time_needed = timed_function_beautified(RSALibrary.encrypt_text_v2,
```

```
n, e, input_text)
    else:
        text_output, time_needed = timed_function_beautified(RSALibrary.encrypt_text_v2,
n, d, input_text)
    context = {
        'n': n,
        'e': e,
        'd': d,
        'p': p,
        'q': q,
        'input_text': input_text,
        'text_output': text_output,
        'time_needed': time_needed,
        'keytype': keytype
    }
    return render_template('crypto.html', **context)
@app.route('/crypto_text_dec', methods=['GET', 'POST'])
def crypto_text_dec():
    if request.method != 'POST':
        return crypto()
    input_text_dec = int(request.form.get('input_text_dec')) if
request.form.get('input_text_dec') else 0
    keytype = request.form.get('keytype')
    print(type(keytype))
    if keytype != 'private':
        keytype = 'public'
        text_output_dec, time_needed =
timed_function_beautified(RSALibrary.decrypt_text_v2, n, e, input_text_dec)
    else:
        text_output_dec, time_needed =
```

```
timed_function_beautified(RSALibrary.decrypt_text_v2, n, d, input_text_dec)
    context = {
        'n': n,
        'e': e,
        'd': d,
        'p': p,
        'q': q,
        'input_text_dec': input_text_dec,
        'text_output_dec': text_output_dec,
        'time_needed': time_needed,
        'keytype': keytype
    }
    return render_template('crypto.html', **context)
@app.route('/visual')
def visual():
    x = [64, 128, 256, 512, 1024]
    multipower_enctime = [1.999855042, 5.000352859, 1.002311707, 13.0007267, 42.00315475]
    multiprime_enctime = [0.999689102, 5.01203537, 33.00285339, 42.0024395, 95.00741959]
    crt_enctime = [63.00497055, 95.00646591, 159.0106487, 399.030447, 1109.083176]
    rsa_enctime = [78.00459862, 90.00706673, 165.0128365, 406.0301781, 1131.083965]
    char_enctime = [83.00614357, 135.0224018, 250.0200272, 631.0470104, 1693.109989]
    dependent_enctime = [156.0130119, 270.0023651, 547.0211506, 1577.118874, 5294.175386]
    df_enc = pd.DataFrame({
        'Size of Keys (bits)': x,
        'MultiPower RSA': multipower_enctime,
        'MultiPrime RSA': multiprime_enctime,
        'CRT RSA': crt_enctime,
```

```
        'RSA': rsa_enctime,
        'Carmichael RSA': char_enctime,
        'Dependent RSA': dependent_enctime
    })
    fig_enc = px.line(df_enc, x='Size of Keys (bits)', y=df_enc.columns[1:], title='Encryption
Time',
                      labels={'value': 'Time (ms)'})
    fig_enc.update_layout(xaxis_range=[0, 1200], yaxis_range=[0, 1000])
    fig_enc = fig_enc.to_html(full_html=False)
    multipower_dectime = [79.00309563, 622.0452785, 3644.270182, 20661.53336,
104141.7747]
    multiprime_dectime = [59.00239944, 3774.280071, 83714.70499, 381739.8643,
999999]
    crt_dectime = [87.00656891, 319.0245628, 1334.099293, 7375.546932, 47142.4973]
    rsa_dectime = [142.0109272, 635.0471973, 3539.262056, 23362.73408, 165806.3066]
    char_dectime = [214.0161991, 962.0711803, 5459.403515, 34631.56891, 223170.254]
    dependent_dectime = [414.0496254, 1490.129471, 7913.590431, 51314.013,
350634.3005]
    df_dec = pd.DataFrame({
        'Size of Keys (bits)': x,
        'MultiPower RSA': multipower_dectime,
        'MultiPrime RSA': multiprime_dectime,
        'CRT RSA': crt_dectime,
        'RSA': rsa_dectime,
        'Carmichael RSA': char_dectime,
        'Dependent RSA': dependent_dectime
    })
    fig_dec = px.line(df_dec, x='Size of Keys (bits)', y=df_dec.columns[1:], title='Decryption
Time',
                      labels={'value': 'Time (ms)'})
```

```
    fig_dec.update_layout(xaxis_range=[0, 1200], yaxis_range=[0, 150000])
    fig_dec = fig_dec.to_html(full_html=False)
    multipower_keygen = [15.0012, 21.40101, 65.8048, 171.227, 831.6657]
    multiprime_keygen = [218.75, 640.625, 2703.125, 6250, 169328.125]
    crt_keygen = [11.00063324, 32.00221062, 40.625, 206.12375, 728.125]
    rsa_keygen = [15.00201225, 20.99990845, 60.1039, 209.91401, 1059.077978]
    char_keygen = [35.00127792, 34.00063515, 160.0039005, 338.023901, 596.3503]
    dependent_keygen = [15.0008, 24.5006, 51.8034, 145.01, 475.7341]
    df_keygen = pd.DataFrame({
        'Size of Keys (bits)': x,
        'MultiPower RSA': multipower_keygen,
        'MultiPrime RSA': multiprime_keygen,
        'CRT RSA': crt_keygen,
        'RSA': rsa_keygen,
        'Carmichael RSA': char_keygen,
        'Dependent RSA': dependent_keygen
    })
    fig_keygen = px.line(df_keygen, x='Size of Keys (bits)', y=df_keygen.columns[1:],
title='Key Generation Time',
                         labels={'value': 'Time (ms)'})
    fig_keygen.update_layout(xaxis_range=[0, 1200], yaxis_range=[0, 1000])
    fig_keygen = fig_keygen.to_html(full_html=False)
    return render_template('visual.html', fig_enc=fig_enc, fig_dec=fig_dec,
fig_keygen=fig_keygen)
if __name__ == '__main__':
    # run() method of Flask class runs the application
    # on the local development server.
    app.run()
```

Library/RSALibrary.py

```
import random
```

```
import library.Euclidean as Euclidean
import library.PrimalityTests as PrimalityTests
import library.Modulo as Modulo
import library.ExtendedEuclidean as ExtendedEuclidean
import math
import library.StringLibrary as StringLibrary
import library.TimeLibrary as TimeLibrary
# following miller-rabin test
def is_primary(n):
    l = [2, 3]
    if n < 2:
        return False
    if n in [2,3,5,7,11]:
        return True
    bits = math.ceil(math.log2(n))
    if bits < 2:
        return False
    l.append(random.getrandbits(bits // 2))
    l.append(random.getrandbits(bits // 2))
    l.append(random.getrandbits(bits // 2))
    for a in l:
        if PrimalityTests.is_composite_miller_rabin(n, a):
            return False
    return True
def generate_random_primary(bits=512):
    max_iter = 5000
    while True:
        rnd = random.getrandbits(bits)
        if is_primary(rnd):
            return rnd
```

```
        max_iter -= 1
        if max_iter <= 0:
            raise Exception("Cannot find a primary number")
def generate_public_key_e(phi, min_value=3):
    if min_value % 2 == 0:
        min_value += 1
    for i in range(min_value, phi, 2):
        if Euclidean.gcd(i, phi) == 1:
            return i
    raise Exception("Couldn't find E!")
def compute_private_key_d(e, phi):
    g, x, y = ExtendedEuclidean.extended_euclidean(e, phi)
    d = x % phi
    return d
def generate_keys(bits=512, min_e_value=3, return_p_q=False):
    p = generate_random_primary(bits)
    q = generate_random_primary(bits)
    n = p * q
    phi = (p - 1) * (q - 1)
    e = generate_public_key_e(phi, min_value=min_e_value)
    d = compute_private_key_d(e, phi)
    if return_p_q:
        return n, e, d, p, q
    return n, e, d
def encrypt_number(n, e, number):
    if number >= n or number < 0:
        raise Exception("Number should be greater than 0 and less than N")
    return Modulo.pow_mod_binary(number, e, n)
def decrypt_number(n, d, cipher_number):
    return encrypt_number(n, d, cipher_number)
```

```
def decrypt_number_crt(d, p, q, cipher_number):
    n = p * q
    mp = Modulo.pow_mod_binary(cipher_number, d % (p - 1), p)
    mq = Modulo.pow_mod_binary(cipher_number, d % (q - 1), q)
    g, yp, yq = ExtendedEuclidean.extended_euclidean(p, q)
    m = (Modulo.mult_mod([mp, yq, q], n) + Modulo.mult_mod([mq, yp, p], n)) % n
    return m
def encrypt_text(n, e, text):
    l = []
    for c in text:
        l.append(encrypt_number(n, e, ord(c)))
    return l
def decrypt_text(n, d, cipher_text_list):
    s = ""
    for cipher_number in cipher_text_list:
        o = decrypt_number(n, d, cipher_number)
        s += chr(o)
    return s
def encrypt_text_to_text(n, e, text, sep='|'):
    l = encrypt_text(n, e, text)
    l = [str(n) for n in l]
    return sep.join(l)
def decrypt_text_from_text(n, d, cipher_text, sep='|'):
    l = cipher_text.split(sep)
    l = [int(s) for s in l]
    return decrypt_text(n, d, l)
def encrypt_text_v2(n, e, text):
    if len(text) >= math.log2(n) / 8:  # length of the text is more than the number of bytes in
N
        raise Exception("Text is longer than n({})".format(n))
```

```
    plain = StringLibrary.text_to_integer(text)
    return encrypt_number(n, e, plain)
def decrypt_text_v2(n, d, cipher_number):
    if cipher_number >= n:
        raise Exception("Cipher number({}) is longer than n({})".format(cipher_number, n))
    plain = decrypt_number(n, d, cipher_number)
    return StringLibrary.integer_to_text(plain)
def test():
    print("Generating keys...")
    n, e, d, p, q = generate_keys(bits=512, min_e_value=3, return_p_q=True)
    print("n=", n)
    print("e=", e)
    print("d=", d)
    print('-' * 20)
    print("Encrypting a number:")
    number = random.getrandbits(8)
    print("Number to send:", number)
    cipher_number = encrypt_number(n, e, number)
    print("Cipher:", cipher_number)
    decrypted = decrypt_number(n, d, cipher_number)
    print("Decrypted:", decrypted)
    decrypted_crt = decrypt_number_crt(d, p, q, cipher_number)
    print("Decrypted using CRT:", decrypted_crt)
    print('-' * 20)
    print("Normal vs CRT performance:")
    try:
        ts_normal = 0
        ts_crt = 0
        for i in range(300, 1000):
            _, t = TimeLibrary.timed_function(decrypt_number, n, d, cipher_number)
```

```
        ts_normal += t
        _, t = TimeLibrary.timed_function(decrypt_number_crt, d, p, q, cipher_number)
        ts_crt += t
    print("Total time for normal:", TimeLibrary.beautify_time(ts_normal))
    print("Total time for CRT:", TimeLibrary.beautify_time(ts_crt))
    print("Rate: %f times" % (ts_normal / ts_crt))
except:
    print("Failed!")
    pass
print('-' * 20)
print("Encrypting a text:")
text = "This is a test message..."
print("Text to send:", text)
cipher_text = encrypt_text(n, e, text)
print("Cipher Text:", cipher_text)
decrypted = decrypt_text(n, d, cipher_text)
print("Decrypted Text:", decrypted)
print('-' * 20)
print("Encrypting a text to text:")
text = "Another text message!"
print("Text to send:", text)
cipher_text = encrypt_text_to_text(n, e, text)
print("Cipher Text:", cipher_text)
decrypted = decrypt_text_from_text(n, d, cipher_text)
print("Decrypted Text:", decrypted)
print('-' * 20)
print("Encrypting a text to number (v2):")
text = "This third text is going to be encrypted to a single number."
print("Text to send:", text)
cipher_number = encrypt_text_v2(n, e, text)
```

```
    print("Cipher Number:", cipher_number)
    decrypted = decrypt_text_v2(n, d, cipher_number)
    print("Decrypted text:", decrypted)
####################
if __name__ == "__main__":
    test()
```

Templates/crypto.html

```
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet" href="https://www.w3schools.com/w3css/4/w3.css">
    <style>
    table {
        width: 100%;
        border: solid 1px;
        border-collapse: collapse;
        word-break: break-word;
        text-align: center;
        padding: 20px;
    }
    td,th {
        border: solid 1px;
    }
    .result, .time{
        width: 100%;
        word-break: break-all;
        font-weight: bold;
     }
     .time{
        color: black;
```

```
    }
    .result{
       color: black;
    }
    .public{
       background: #c5f3c5;
   }
   .private{
       background: #fbebeb;
   }
   </style>
</head>
<body>
<div class="w3-container w3-teal">
  <h1>Encryption</h1>
<!--<table style="width: 100%">
   <tr>
       <th style="min-width: 150px" colspan="2">Variable</th>
       <th>Value</th>
   </tr>
   <tr class="public">
       <td rowspan="2" style="width: 100px">Public</td>
       <td>n</td>
       <td>{{n}}</td>
   </tr>
   <tr class="public">
       <td>e</td>
       <td>{{e}}</td>
   </tr>
   <tr class="private">
```

```
        <td rowspan="3" style="min-width: 50px">Private</td>
        <td>d</td>
        <td>{{d}}</td>
    </tr>
    <tr class="private">
        <td>p</td>
        <td>{{p}}</td>
    </tr>
    <tr class="private">
        <td>q</td>
        <td>{{q}}</td>
    </tr>
</table> -->
<br/>
<hr/>
<h3>Encrypt/Decrypt Number</h3>
<form method="POST" action="{{ url_for('crypto_number') }}">
    Use: <input type="radio" value="public" name="keytype" {% if not keytype or keytype=="public" %} checked {%endif%}>
    Public Key
    <input type="radio" value="private" name="keytype" {% if keytype and keytype=="private" %} checked {%endif%}>
    Private Key
    <br/>
    Input: <input type="input" name="input" value="{{input}}" required style="min-width: 90%">
    <input type="submit" value="Submit"><br/>
    Output: <span class="result">{{ number_output }}</span><br/>
    {% if time_needed and number_output%}
    Time Needed: <span class="time">{{ time_needed }}</span><br/>
```

```
    {% endif %}
    <br />
    {% if number_output_crt %}
    Output (CRT): <span class="result">{{ number_output }}</span><br/>
    Time Needed (CRT): <span class="time">{{ time_needed_crt }}</span><br/>
    {% endif %}
</form>
<hr/>
<h3>Encrypt/Decrypt Text to Number</h3>
<form method="POST" action="{{ url_for('crypto_text') }}">
    Use: <input type="radio" value="public" name="keytype" {% if not keytype or keytype=="public" %} checked {%endif%}>
    Public Key
    <input type="radio" value="private" name="keytype" {% if keytype and keytype=="private" %} checked {%endif%}>
    Private Key
    <br/>
    Input: <input type="text" name="input_text" value="{{input_text}}" required style="min-width: 90%">
    <input type="submit" value="Submit"><br>
    Output:
    <div class="result">{{ text_output }}</div>
    {% if time_needed and text_output %}
    Time Needed: <span class="time">{{ time_needed }}</span><br/>
    {% endif %}
</form>
<h3>Decrypt/Encrypt Number to Text</h3>
<form method="POST" action="{{ url_for('crypto_text_dec') }}">
    Use: <input type="radio" value="public" name="keytype" {% if not keytype or keytype=="public" %} checked {%endif%}>
```

```
    Public Key
    <input type="radio" value="private" name="keytype" {% if keytype and keytype=="private" %} checked {%endif%}>
    Private Key
    <br/>
    Input: <input type="input" name="input_text_dec" value="{{input_text_dec}}" required style="min-width: 90%">
    <input type="submit" value="Submit"><br>
    Output:
    <div class="result">{{ text_output_dec }}</div>
    {% if time_needed and text_output_dec %}
    Time Needed: <span class="time">{{ time_needed }}</span><br/>
    {% endif %}
</form>
<br/>
<br/>
<br/>
<div>
    {% if n > 0 and (e > 0 or d > 0) %}
    <a href="{{ url_for('index') }}">Back to Home Page</a>
    {% endif %}
</div>
</body>
</html>
```

Templates/enter_keys.html

```
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet" href="https://www.w3schools.com/w3css/4/w3.css">
    <style>
```

```
    table {
        width: 100%;
        border: solid 1px;
        border-collapse: collapse;
        word-break: break-word;
        text-align: center;
        padding: 20px;
    }
    td {
        border: solid 1px;
    }
    </style>
</head>
<body>
<div class="w3-container w3-teal">
  <h1>Enter Keys</h1>
</div>
<form style="margin-top: 20px" method="POST">
    n: <input value="{{n}}" name="n" style="width: 90%"><br/>
    e: <input value="{{e}}" name="e" style="width: 90%"><br/>
    d: <input value="{{d}}" name="d" style="width: 90%"><br/>
    p: <input value="{{p}}" name="p" style="width: 90%"><br/>
    q: <input value="{{q}}" name="q" style="width: 90%"><br/>
    <input type="submit" value="Save Values">
</form>
    {% if saved %}
    Saved!
    {% endif %}
<br />
<div>
```

```
    {% if n > 0 and (e > 0 or d > 0) %}
    <a style="float: right" href="{{ url_for('crypto') }}">Proceed &rarr;</a>
    {% endif %}
</div>
<br />
<br />
<br />
<div>
    <a href="{{ url_for('index') }}">Back to Home Page</a>
</div>
</body>
</html>
```

Templates/generate_keys.py

```
<html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1">
<link rel="stylesheet" href="https://www.w3schools.com/w3css/4/w3.css">
    <style>
    table {
        width: 100%;
        border: solid 1px;
        border-collapse: collapse;
        word-break: break-word;
        text-align: center;
        padding: 20px;
    }
    td,th {
        border: solid 1px;
    }
    .time{
```

```
        width: 100%;
        word-break: break-all;
        font-weight: bold;
      }
      .time{
        color: red;
      }
    .public{
        background: #c5f3c5;
    }
    .private{
        background: #fbebeb;
    }
    </style>
</head>
<body>
<div class="w3-container w3-teal">
  <h1>Generate Keys</h1>
</div>
<form style="margin-top: 20px" action="{{url_for('generate_keys')}}">
    Bits for p and q:<input value="{{bits}}" name="bits">
    Minimum value for e <input value="{{min_e}}" name="min_e">
    <input type="submit" value="Regenerate">
</form>
<table style="width: 100%">
    <tr>
        <th style="min-width: 150px" colspan="2">Variable</th>
        <th>Value</th>
    </tr>
    <tr class="public">
```

```
        <td rowspan="2" style="width: 100px">Public</td>
        <td>n</td>
        <td>{{n}}</td>
    </tr>
    <tr class="public">
        <td>e</td>
        <td>{{e}}</td>
    </tr>
    <tr class="private">
        <td rowspan="3" style="min-width: 50px">Private</td>
        <td>d</td>
        <td>{{d}}</td>
    </tr>
    <tr class="private">
        <td>p</td>
        <td>{{p}}</td>
    </tr>
    <tr class="private">
        <td>q</td>
        <td>{{q}}</td>
    </tr>
</table>
<br/>
{% if time_needed %}
Time Needed: <span class="time">{{ time_needed }}</span><br />
{% endif %}
<br/>
<div>
    {% if n > 0 and (e > 0 or d > 0) %}
    <a style="float: right" href="{{ url_for('crypto') }}">Proceed &rarr;</a>
```

```
    {% endif %}
</div>
<br/>
<br/>
<br/>
<div>
    {% if n > 0 and (e > 0 or d > 0) %}
    <a href="{{ url_for('index') }}">Back to Home Page</a>
    {% endif %}
</div>
</body>
</html>
```