МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра кібербезпеки

КАЛУШКА Максим Михайлович

Алгоритми гібридного шифрування / Algorithms of Hybrid Encryption

спеціальність: 125 – Кібербезпека та захист інформації
освітньо-професійна програма – Кібербезпека

Кваліфікаційна робота

Виконав студент групи КБм -21
М.М. Калушка

Науковий керівник
д.т.н., професор В.В. Яцків

Кваліфікаційну роботу допущено
до захисту:

«____» _____ ___2024 р.

Завідувач кафедри
_____ В.В.Яцків

ТЕРНОПІЛЬ – 2024

Факультет комп'ютерних інформаційних технологій
Кафедра кібербезпеки
Освітній ступінь «магістр»
спеціальність: 125 – Кібербезпека та захист інформації
освітньо-професійна програма – Кібербезпека

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ В.В.Яцків
«____» _____ 2024 року

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
КАЛУШКИ Максима Михайловича
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи:
Алгоритми гібридного шифрування /
Algorithms of Hybrid Encryption
керівник роботи д.т.н., професор В.В.Яцків
затверджені наказом по університету від 12 грудня 2023 року № 753
2. Строк подання студентом закінченої випускної кваліфікаційної роботи 12 грудня 2024 року.
3. Вихідні дані до кваліфікаційної роботи: завдання на випускну кваліфікаційну роботу студента, наукові статті, технічна література.
4. Основні питання, які потрібно розробити:
– провести аналіз методів шифрування даних;
– проаналізувати основні симетричні криптографічні алгоритми;
– проаналізувати основні асиметричні криптографічні алгоритми;
– дослідити сфери застосування симетричних та асиметричних алгоритмів;
– провести аналіз принципів роботи асиметричного та симетричного шифрування;
– провести аналіз принципів роботи гібридного шифрування;
– побудувати алгоритми шифрування даних та симетричного ключа;
– реалізувати модулі шифрування та дешифрування пакетів даних на основі запропонованих алгоритмів гібридного алгоритму шифрування.
5. Перелік графічного матеріалу у роботі:
– принцип шифрування та дешифрування;
– основні методи криптографії;
– схема асиметричного шифрування;
– схема симетричного шифрування;
– загальна схема гібридного шифрування;
– алгоритм шифрування пакету даних;
– алгоритм шифрування симетричного ключа;
– алгоритм дешифрування симетричного ключа;
– вікно модуля шифрування;
– вікно модуля дешифрування.

6. Консультанти розділів кваліфікаційної роботи

	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 12 грудня 2023 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строки виконання етапів кваліфікаційної роботи	Примітка
1	Основи шифрування та криптографії	12.2023 р. – 03.2024 р.	
2	Концепція та принципи гібридного шифрування	03.2024 р. – 06.2024 р.	
3	Практична реалізація гібридного шифрування	06.2024 р. – 11.2024 р.	

Студент _____ Максим КАЛУШКА
(підпис)

Керівник роботи _____ д.т.н., професор Василь ЯЦКІВ
(підпис)

АНОТАЦІЯ

Кваліфікаційна робота на тему «Алгоритми гібридного шифрування» на здобуття освітнього ступеня «Магістр» зі спеціальності 125 «Кібербезпека та захист інформації» освітньо-професійної програми «Кібербезпека» написана обсягом 73 сторінки і містить 24 рисунки, 5 таблиць та 61 джерело за переліком посилань.

Метою кваліфікаційної роботи є підвищення ефективності алгоритмів гібридного шифрування даних.

Методи досліджень. Для розв’язання поставлених задач у даній кваліфікаційній роботі використано: методи аналізу контролю та обмеження доступу, процеси шифрування та дешифрування.

Результати дослідження: проведено дослідження алгоритмів симетричного та асиметричного шифрування даних, запропоновано та реалізовано алгоритм гібридного шифрування на основі симетричного алгоритму Blowfish та асиметричного ECC.

Результати роботи можуть успішно застосовуватися при шифрованому обміні даними між користувачами в загальній мережі.

КЛЮЧОВІ СЛОВА: КІБЕРБЕЗПЕКА, ШИФРУВАННЯ ДАНИХ, ДЕШИФРУВАННЯ ДАНИХ, СИМЕТРИЧНІ АЛГОРИТМИ, АСИМЕТРИЧНІ АЛГОРИТМИ, ГІБРИДНЕ ШИФРУВАННЯ.

ABSTRACT

The graduate work on the topic «Algorithms of Hybrid Encryption» for Master's degree on speciality 125 " Cybersecurity and Information Protection " is written on 73 pages and contains 24 illustrations, 5 tables and 61 references.

The aim of graduate work is to investigate the principles of hybrid encryption and data encryption algorithms within it.

Research methods. To solve the tasks set in this qualification work, the following methods were used: analysis of access control and restriction, encryption, and decryption processes.

Results of the study. A study of symmetric and asymmetric data encryption algorithms was conducted, and a hybrid encryption algorithm based on the symmetric Blowfish algorithm and the asymmetric ECC algorithm was proposed and implemented.

The results of the work can be successfully applied for encrypted data exchange between users in a general network.

KEYWORDS: CYBERSECURITY, DATA ENCRYPTION, DATA DECRYPTION, SYMMETRIC ALGORITHMS, ASYMMETRIC ALGORITHMS, HYBRID ENCRYPTION.

.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ЦП – Центральний процесор.

RSA – Rivest–Shamir–Adleman.

AES – Advanced Encryption Standard.

DES – Data Encryption Standard.

RC4 – Rivest cipher 4.

IDEA – International Data Encryption Algorithm.

TDES – Triple DES.

DSA – Digital Signature Algorithm.

ECC – Elliptic Curve Cryptography.

NIST – National Institute of Standards and Technology.

OQS – Open Quantum Safe.

KEM – Key-encapsulation Mechanism.

MLWE – Module-LWE.

ВСТУП

З ростом обсягів обміну даними та комунікації через інтернет захист інформації від зловмисників стає вкрай важливим [1]. Для цього використовуються різні методи шифрування, але кожен із них має свої недоліки: симетричне - складність з передачею ключа, тоді як асиметричне має низьку швидкість та вимагає значних ресурсів.

Гібридне шифрування поєднує переваги симетричних і асиметричних алгоритмів, забезпечуючи як високу безпеку, так і ефективність при передачі даних [2]. У ньому симетричне шифрування використовується для швидкого шифрування великих обсягів даних, а сам симетричний ключ шифрується за допомогою відкритого ключа асиметричного алгоритму. Це гарантує належний рівень захисту під час передачі через незахищені канали або при зберіганні на публічних хмарних сервісах.

Наразі методи гібридного шифрування є одними з найефективніших способів захисту даних, проте із розвитком квантових досліджень існуючі рішення у цій сфері потребуватимуть додаткових досліджень та удосконалень [3].

Одною із ключових завдань для гібридного шифрування є захист масивів даних або файлів при пересиланні від одного абонента до іншого. Використання різноманітних методів гібридного шифрування дає змогу передавати інформацію через публічні канали обміну інформацією та уникати витоку важливої інформації.

Мета і завдання дослідження. Метою кваліфікаційної роботи є дослідження методів та алгоритмів гібридного шифрування даних.

Досягнення визначеної мети передбачає вирішення таких завдань:

– провести аналіз методів шифрування даних;

– проаналізувати основні симетричні криптографічні алгоритми;

– проаналізувати основні асиметричні криптографічні алгоритми;

– дослідити сфери застосування симетричних та асиметричних алгоритмів;

– провести аналіз принципів роботи асиметричного та симетричного шифрування;

– провести аналіз принципів роботи гібридного шифрування;

– побудувати алгоритми шифрування даних та симетричного ключа;

– реалізувати модулі шифрування та дешифрування пакетів даних на основі запропонованих алгоритмів гібридного алгоритму шифрування.

Об’єкт дослідження – процеси шифрування та дешифрування даних на стороні відправника та отримувача і обмін ключами.

Предмет дослідження – принципи та алгоритми гібридного шифрування та реалізація його складових.

Методи досліджень. Для розв’язання поставлених задач у даній кваліфікаційній роботі використано: методи аналізу контролю та обмеження доступу, процеси шифрування та дешифрування.

Наукова новизна одержаних результатів. Проведено дослідження алгоритмів симетричного та асиметричного шифрування даних, запропоновано та реалізовано алгоритм гібридного шифрування на основі симетричного алгоритму Blowfish та асиметричного ECC.

Практичне значення отриманих результатів. Реалізовано модулі відправника та отримувача на основі запропонованих у роботі алгоритмів та методів поєднання симетричного та асиметричного алгоритмів шифрування.

Публікації та апробація до магістерської роботи.

1. Калушка М., Кулина С. (2024). Схема та алгоритм гібридного шифрування. *Збірник матеріалів науково-практичної конференції молодих вчених, аспірантів та студентів «Кібербезпека та комп’ютерно-інтегровані технології» (КБКІТ-2024)*, Тернопіль, с. 100-103.

2. Калушка М., Кулина С. (2024). Концепція та принципи гібридного шифрування. *Кібербезпека державних інституцій та подолання кризових станів* : *матеріали ІІІ Міжнар. наук.-практ. конф. в 2 т. (Київ – Прага – Таллінн – Тернопіль)*, 14.11.2024 р. ІСЗЗІ КПІ ім. Ігоря Сікорського. С. 450.

1. ОСНОВИ ШИФРУВАННЯ ТА КРИПТОГРАФІЇ

1.1. Поняття криптографії та її класифікація

Зі збільшенням використання обміну даними та комунікації через Інтернет стає критично важливим захистити дані від потрапляння до рук зловмисників. Забезпечення цілісності та конфіденційності даних стає одним із ключових викликів для дослідників та фахівців у сфері кібербезпеки [2].

Це досягається за допомогою криптографії [4]. Криптографія - це наука та практика захисту інформації шляхом її перетворення в закодований вигляд, що робить її зрозумілою тільки для тих, хто має відповідні права доступу або ключі для розшифрування. Вона використовується для забезпечення конфіденційності, цілісності, аутентифікації та невідмовності даних.

Конфіденційність даних означає захист даних від несанкціонованого розкриття чи витоку. Цілісність даних - це ключова властивість даних, що гарантує їх точність, повноту, узгодженість і надійність у процесі зберігання, передачі та обробки. Вона забезпечує, щоб дані у цих процесах не були випадково або навмисно змінені, спотворені або пошкоджені та зберігали свою первісну цінність і коректність [5].

Автентифікація дозволяє підтверджувати, що джерело даних є надійним і інформація надходить саме від того, хто заявлений як відправник. Вона реалізується шляхом застосування цифрових підписів, сертифікатів та інших методів перевірки особистості. Умова невідновності даних забезпечує, що відправник не може заперечити про факт відправлення даних, а отримувач про факт отримання. Це зазвичай досягається за допомогою цифрових підписів або протоколів підтвердження передачі [6].

Контроль доступу дозволяє обмежити доступ до інформації відповідно до ролей і рівнів доступу користувачів. Це включає механізми багатофакторної аутентифікації, розподіл привілеїв та використання паролів.

Мета криптографії полягає в захисті критичних даних або документів на жорсткому диску або під час їх передачі через ненадійний канал зв'язку [4].

А під ненадійним каналом зв'язку мають на увазі такий канал через який передача даних може піддаватися втратам, пошкодженням, перехопленню чи зміні через сторонній вплив. У такому каналі можуть виникати різні помилки під час передачі інформації, а також ризики несанкціонованого доступу.

Відповідно використання криптографії дозволяє частково уникнути таких загроз або зменшити їх вплив. сам процес захисту інформації полягає в шифруванні та дешифруванні даних.

Шифрування даних – це процес захисту повідомлень шляхом перетворення відкритого тексту у шифротекст, тоді як зворотний процес отримання початкового значення даних з шифротексту називається дешифруванням. Шифрування та дешифрування здійснюється за допомогою ключів. На рисунку 1.1 показано принцип шифрування та дешифрування даних.

Рисунок 1.1 - Принцип шифрування та дешифрування

Як уже зазначалося раніше процес шифрування полягає у створенні шифротексту, відповідно для цього застосовують різноманітні алгоритми шифрування. Ключовим завданням алгоритмів шифрування є ускладнити процес дешифрування при відсутності ключа, що використовувався для шифрування даних. Відповідно чим складніший і досконаліший алгоритм шифрування, тим більше часу потребується для його зламу [7].

Основні методи криптографії включають шифрування (із симетричним та асиметричним ключами) та гешування [8].

Розподіл основних методів криптографії наведено на рисунку 1.2.

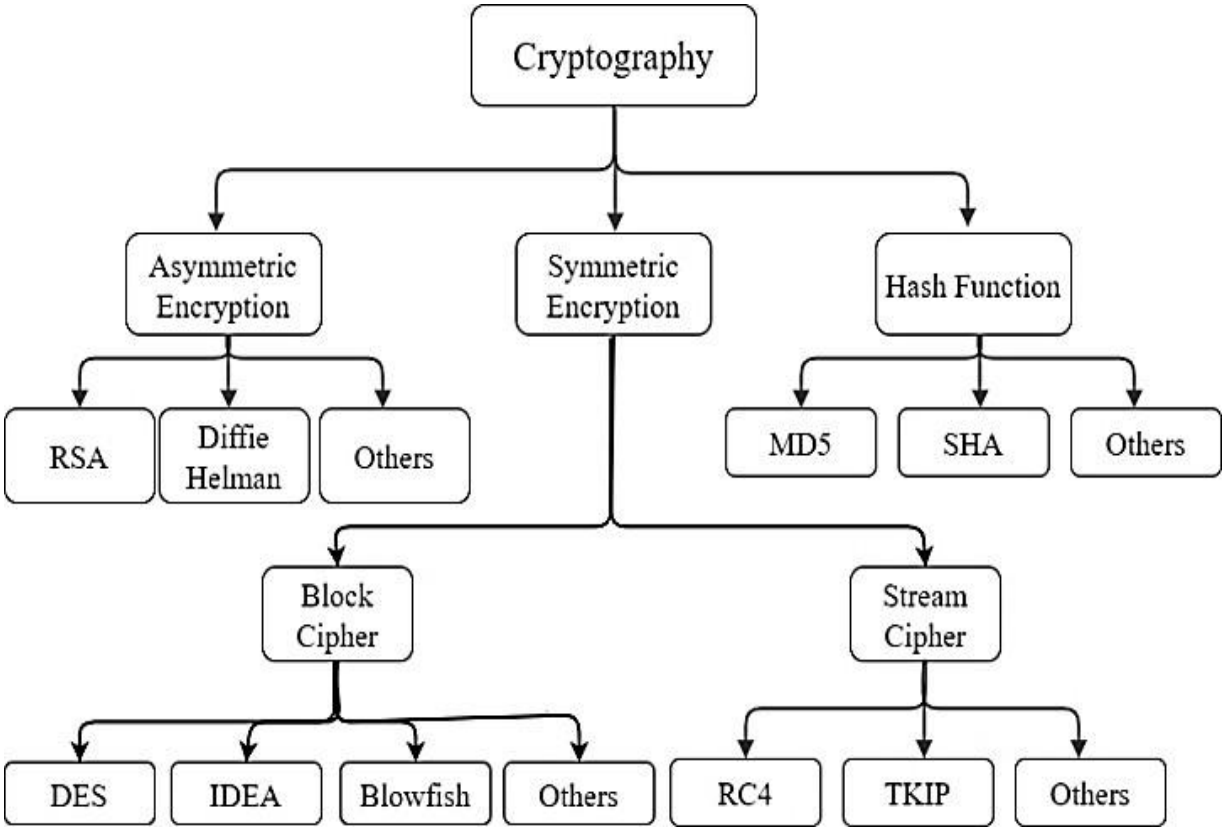

Рисунок 1.2 - Основні методи криптографії [8]

Як бачимо з рисунка 1.2 першим поширеним методом є асиметричне шифрування. Цей метод шифрування використовує пару математично пов'язаних ключів, які називають відкритий ключ та закритий ключ. Ця пара ключів дозволяє забезпечити високий рівень захисту при передачі даних та автентифікації, оскільки операції шифрування та розшифрування виконуються різними ключами. Відкритий ключ призначений для шифрування даних і доступний будь-кому, хто хоче надіслати зашифроване повідомлення отримувачу. Натомість закритий ключ використовується для розшифрування повідомлення і зберігається в таємниці, його має тільки отримувач (рис. 1.3).

Хоча асиметричні системи забезпечують вищий рівень безпеки, вони не підходять для файлів великого розміру. Це пояснюється тим, що швидкість шифрування значно повільніша у порівнянні з системами на основі симетричних ключів та мають вище споживання ЦП [9].

Найбільш поширеними алгоритмами асиметричного шифрування є RSA, Ель-Гамаля та Діффі-Геллмана [10].

Рисунок 1.3 - Процес асиметричного шифрування

Наступним і не менш поширеним методом шифрування є симетричне. У цьому методі як шифрування, так і дешифрування здійснюються на основі єдиного ключа, який ще називається приватним або секретним ключем [11].

Окрім цього алгоритми із симетричним ключем залежно від вхідних даних поділяються на два типи: блочні та потокові шифри. У системах на основі блочного шифру дані обробляються або шифруються на фіксованій групі бітів, що називається блоком, тоді як у системах на основі потокового шифру дані обробляються на потоці бітів. Найбільш поширеними алгоритмами симетричного шифрування є AES і DES з блочних та RC4 і Blowfish з потокових шифрів [12].

Процес симетричного шифрування зображено на рисунку 1.4.

Рисунок 1.4 - Процес симетричного шифрування

Основним недоліком симетричних алгоритмів є те, що для обміну секретним ключем між відправником і отримувачем потрібен окремий захищений канал [13].

Ну і останнім, третім методом застосування криптографії є гешування. Згідно визначення: «при гешуванні вхідне повідомлення переводиться в компактний бітовий рядок фіксованого розміру, який називається геш, а функції для його отримання називають геш-функціями. Такі функції є математичними алгоритмами, які відображають вхідне повідомлення довільного розміру в геш фіксованого розміру (дайджест повідомлення)» [14].

На рисунку 1.5 представлена загальна концепція геш-функції.

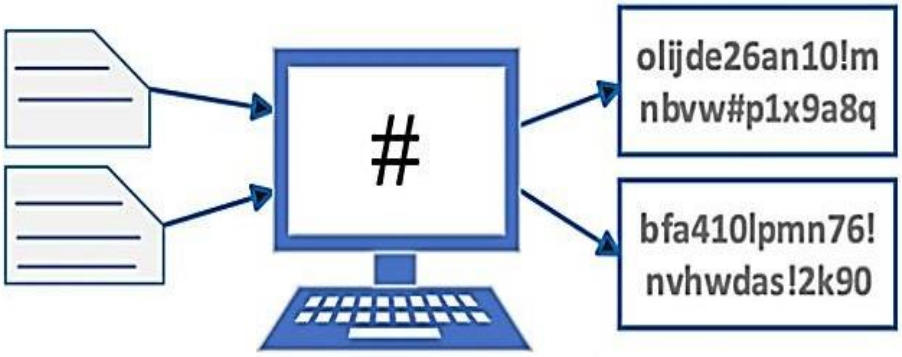

Рисунок 1.5 - Процес гешування файлу

Геш-функція виконує перетворення початкового значення в геш і забезпечує, що навіть невелика зміна вхідних даних призведе до зовсім іншого результату. Ця властивість ще називають лавинним ефектом.

Проте, головна властивість геш-значень – незворотність, тобто, знаючи лише геш, неможливо відновити оригінальні дані. Це робить гешування корисним для захисту паролів та інших конфіденційних даних [15].

Іншою ключовою властивістю процесу гешування є фіксована довжина результату, тобто незалежно від розміру вхідних даних, геш-функція завжди видає результат однакової довжини (наприклад, SHA-256 завжди повертає геш довжиною 256 біт).

Ну і останньою властивістю яку варто назвати є відсутність колізій. Вона означає, що різні вхідні дані повинні породжувати різні геші. Ідеальною є така геш-функція, яка унеможливлює колізії (ситуації, коли різні вхідні дані створюють однаковий геш). На практиці, через кінцеву кількість можливих гешів, повністю уникнути колізій неможливо, але криптографічні геш-функції

розроблені таким чином, щоб колізії виникали дуже рідко і були обчислювально нездійсненними для знаходження [16].

Завдяки своїм можливостям геш-функції використовують для захисту паролів, перевірки цілісності даних та цифрових підписів. Проте вони мають і свої недоліки ключовим з яких є вразливість до атак "грубої сили" та райдужних таблиць - спеціальний варіант таблиць пошуку для відновлення початкового значення криптографічних геш-функцій.

Останнім часом окремо виділяють цифрові підписи, які по своєму принципу схожі на асиметричне шифрування. Цифровий підпис - це математичний методи або алгоритм, який використовується для перевірки автентичності та цілісності інформації або повідомлень, наприклад електронної пошти, транзакції кредитної картки або цифрового документа. Він діє як електронний відбиток пальця для однозначної ідентифікації користувачів і захисту даних користувачів. Використання цифрового підпису гарантує, що повідомлення або документ не було змінено з моменту його підписання. Це робиться шляхом застосування гешування до документа або повідомлення, а потім шифрування документа за допомогою секретного ключа відправника. Цифрові підписи використовують інфраструктуру відкритих ключів (PKI) для посилення безпеки. PKI представляє політику та стандарти, які підтримують розповсюдження відкритих ключів і підтвердження особи фізичних або юридичних осіб за допомогою цифрових сертифікатів [17].

1.2. Основні криптографічні алгоритми

1.2.1. Симетричні алгоритми шифрування

Як вже зазначалося вище симетричні алгоритми шифрування використовують однаковий ключ як для шифрування так і для розшифрування даних. «Одним із перших симетричних алгоритмів вважають шифр Цезаря. Згідно джерел: «він є одним з найпростіших симетричних методів шифрування блочного типу, а отже, його легко зламати. Римський правитель Юлій Цезар створив і використовував цю схему шифрування для надсилання військових

наказів своїм легіонам. Його основою є шифр підстановки, де ключі шифрування та дешифрування однакові. Відповідно Ключі є цілими числами, і найбільш часто використовуваним цілим числом було 3» [18]. У процесі шифрування кожен символ алфавіту зсувається праворуч або ліворуч на ключ-значення, як показано на рисунку 1.6.

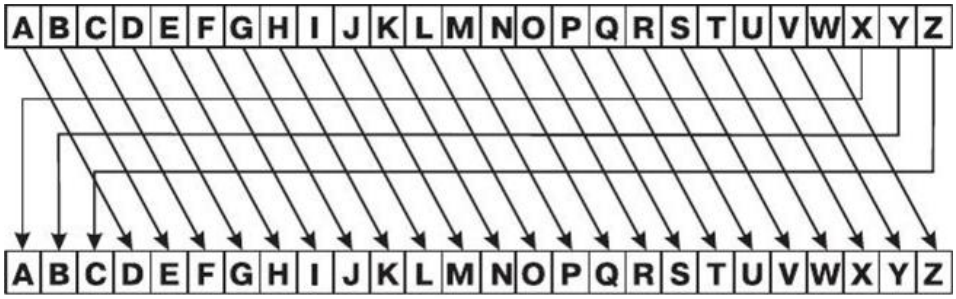

Рисунок 1.6 - Шифр Цезаря [18]

Звичайно в даний час такий алгоритм шифрування застосовується лише в ознайомчих цілях та для навчання базових принципів криптографії.

Основою симетричного шифрування є алгоритм DES [19]. Він є одним із основних симетричних алгоритмів блочного шифрування, який приймає звичайні тексти як блоки, кожен з яких містить 64 біти, і перетворює зашифровані тексти, використовуючи ключі з 64 біти. З цих 64 бітів 8 бітів ключа використовуються для непарності, яка не враховуватиметься в довжині ключа. Тому існує 256 можливих способів знайти правильний ключ.

Алгоритм DES виконує дві перестановки (початкову перестановку та кінцеву перестановку) і 16 кроків обробки, кожен з яких називається раундом, і для кожного раунду використовується інший ключ. DES базується на двох криптографічних операціях: підстановці та транспозиції. У кожному раунді DES виконуються деякі заміни та транспозиції [20].

Алгоритм роботи DES пояснюється на рисунку 1.7.

Згідно алгоритму зображеного на рисунку 1.7 перед початком першого раунду до звичайного тексту застосовується початкова перестановка.

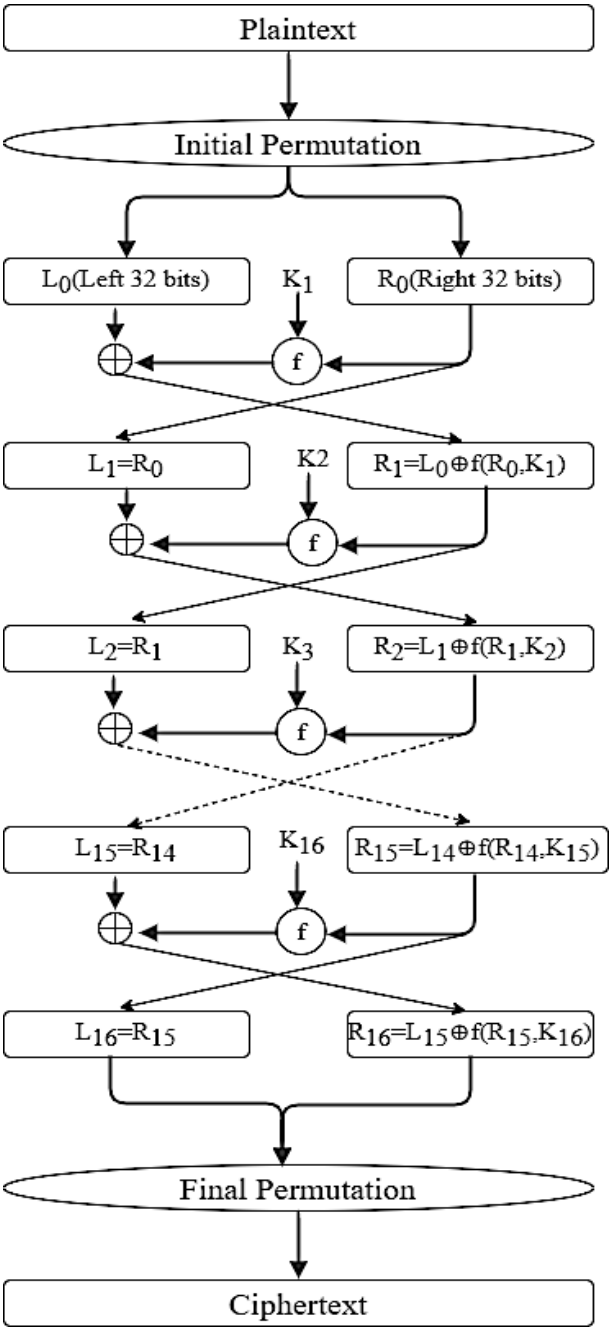

Рисунок 1.7 - Алгоритм DES [20]

Наприклад, початкова перестановка замінює перший біт звичайного тексту на 58-й біт, а другий — на 50-й біт і так далі. Отриманий переставлений блок ділиться на дві половини, обидві мають 32 біти, і кожна з них проходить через 16 раундів процесів шифрування. Остаточна перестановка застосовується до комбінованого блоку, щоб отримати зашифрований текст.

Щоб підвищити безпеку застосування алгоритму DES було запропоновано алгоритм Triple-DES (TDES). Як видно з назви, «TDES застосовує DES тричі до кожного блоку даних, щоб підвищити безпеку

зашифрованих даних. Оскільки безпека TDES втричі вища, ніж DES, то він вважається кращим, ніж DES. Однак він займає значно більшу кількість часу в порівнянні з попередником. TDES працює так само, як і DES, у циклі довжиною 3. Спочатку оригінальний звичайний текст шифрується одним ключем, отриманий зашифрований текст знову шифрується іншим ключем і, нарешті, знову виконується третім ключем» [21].

IDEA – це симетричний блоковий алгоритм шифрування, розроблений у 1990 році Джеймсом Массі та Ха Шаоцзюнем. Їхньою метою було створення алгоритму, який би забезпечував високу надійність та стійкість до відомих атак, уникаючи при цьому патентованих обмежень, як у DES. Завдяки чому він є популярним у різних додатках для захисту даних [21].

IDEA обробляє 64-бітні блоки даних за допомогою 128-бітного ключа. Кожен 64-розрядний блок даних розділений на чотири рівні підблоки, кожен розміром 16 біт. Кожен із цих підблоків проходить вісім раундів повторюваних послідовностей операцій і одну фазу вихідного перетворення. Для кожного циклу роботи ця система потребує шести унікальних ключів, які генеруються з 128-бітного оригінального ключа. Результати кожного раунду даються як вхідні дані для наступного раунду, за винятком восьмого раунду. Вихідні дані восьмого раунду передаються на фазу вихідних перетворень, яка виконує лише арифметичні операції, і їй потрібні чотири ключі. Фаза вихідного перетворення створює остаточний ключ шифру. Весь процес шифрування вимагає 52 ключів.

Процес шифрування алгоритмом IDEA зображено на рисунку 1.8.

Проте з часом алгоритми DES, TDES та IDEA були замінені новим алгоритмом - AES [22]. Цей сучасний симетричний алгоритм шифрування був прийнятий як стандарт шифрування урядом США в 2001 році і є одним із найнадійніших методів для захисту конфіденційної інформації завдяки своїй стійкості та ефективності. Він шифрує та розшифровує шифрує дані блоками фіксованого розміру — 128 біт (16 байтів). Якщо повідомлення довше, воно розбивається на блоки; якщо коротше, застосовується доповнення.

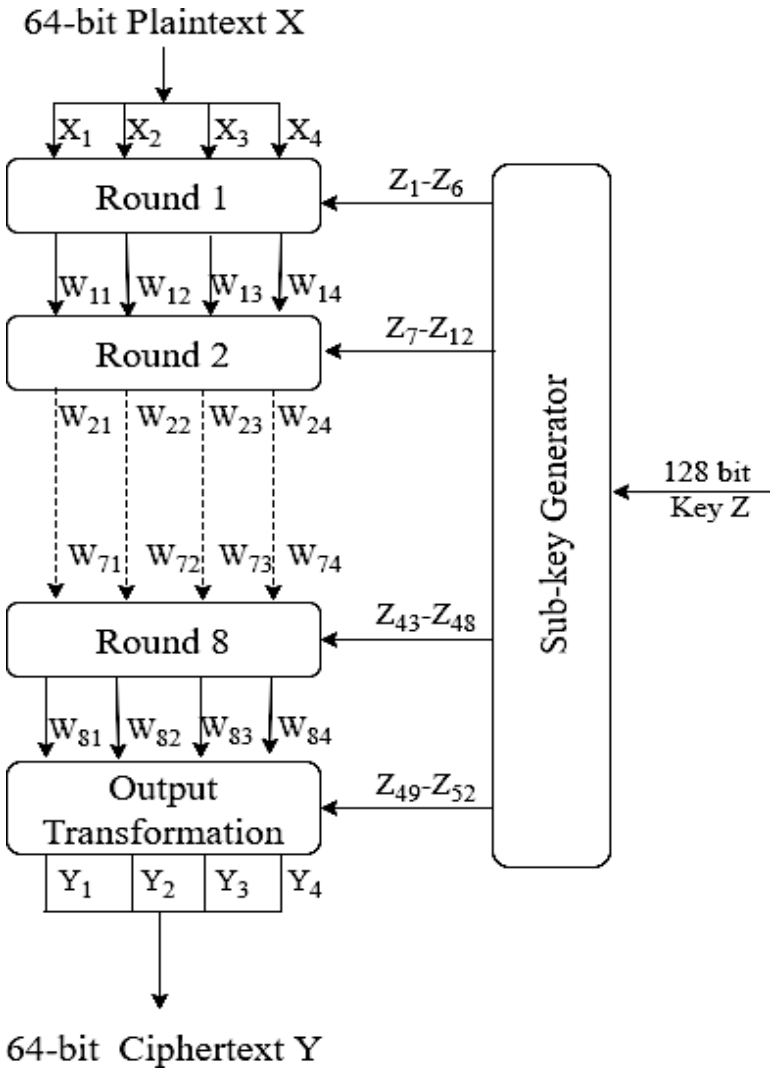

Рисунок 1.8 - Алгоритм IDEA [21]

Залежно від вибору розміру ключа, 128 біт, 196 біт або 256 біт, AES застосовує 10, 12 або 14 циклів шифрування. Чим більший розмір ключа, тим більш захищеним є шифрування, хоча й збільшується час обчислення [22].

Кожен раунд містить чотири операції:

- заміна байту на іншим згідно з таблицею;

- зсув рядків блоку на певну кількість позицій;

- змішування байтів у кожному стовпчику блоку шляхом множення їх на визначені коефіцієнти;

- додавання ключа раунду за допомогою операції XOR.

Однак операція змішування байтів у кожному стовпчику не виконується в останньому циклі. У кожному раунді шифрування використовуються окремі раундові ключі, згенеровані з заданого ключа шифру. Дані, що підлягають

шифруванню, розбиваються на блоки. Кожен блок представлений як масив даних, який відомий як масив стану[23].

Процес шифрування AES показано на рисунку 1.9.

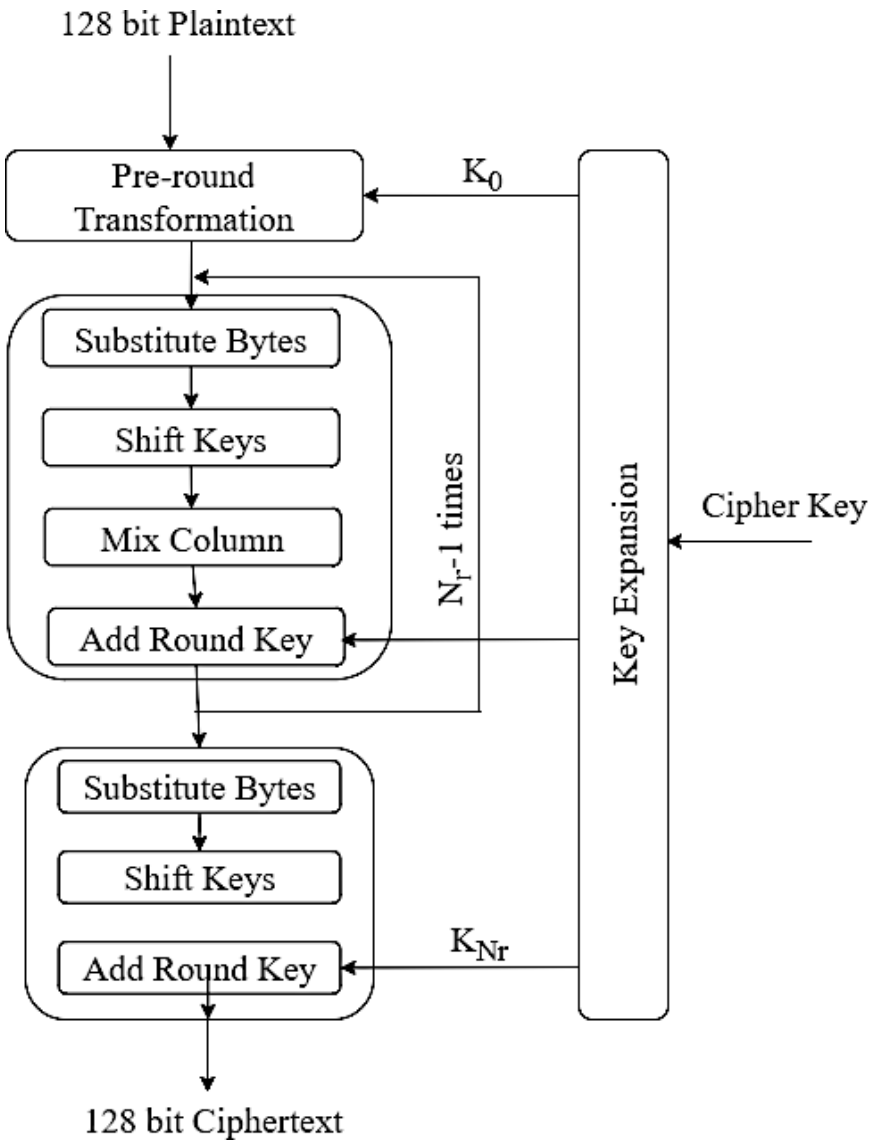

Рисунок 1.9 - Алгоритм AES [23]

За умови, що вам необхідно змінювати довжину ключа під час шифрування, то рекомендують використовувати алгоритм шифрування BlowFish. Це алгоритм шифрування на основі блочного шифру, довжина ключа якого варіюється від 32 до 448 біт. Кожен блок обробляє 64 біти даних [24].

BlowFish шифрує дані за допомогою 16 раундів операцій. У кожному раунді дані проходять в залежності від ключа перестановку в P-блоці та заміну в S-блоці. Кожен S-блок містить 32 біти даних (рис. 1.10).

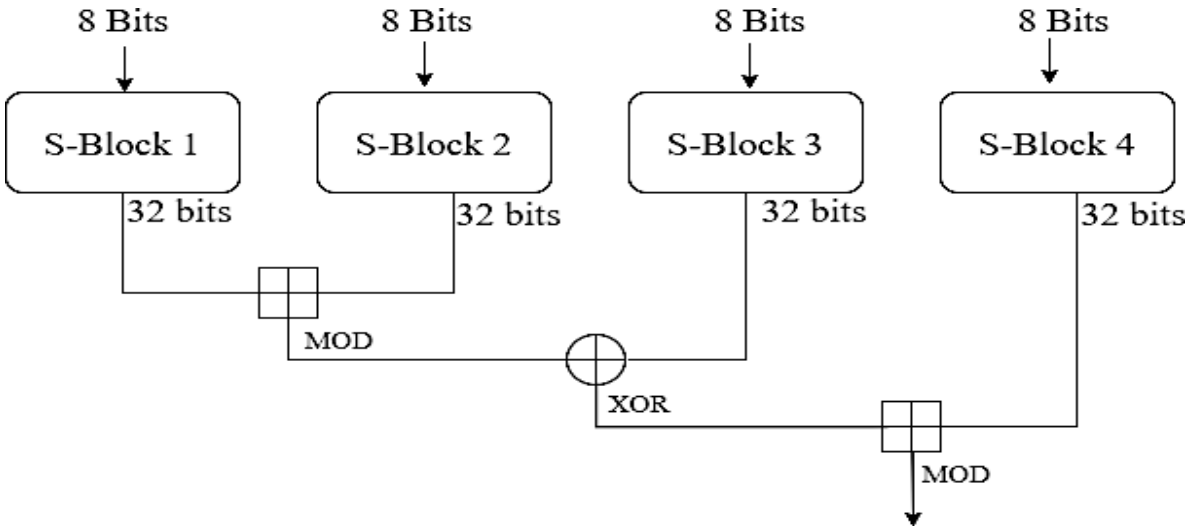

Рисунок 1.10 - Алгоритм BlowFish [24]

На рисунку 1.10 показано функцію BlowFish, яка розділяє 32-бітні дані на чотири чверті; кожен з яких містить 8 біт. Ці чверті будуть входами для S-блоку. У S-блоках виконуються операції XOR і Modulo 232, щоб отримати зашифровані дані. Для розшифровки даних виконується зворотний процес.

Blowfish є одним із найшвидших і найменших блокових шифрів, які зараз використовуються, перетворюючи дані на зашифрований текст за допомогою симетричного ключа шифрування. Blowfish все ще широко використовується протягом трьох десятиліть після того, як він був спочатку створений, оскільки він надає такі переваги:

- Процес шифрування даних ефективний на великих мікропроцесорах.
- Незважаючи на складну фазу ініціалізації перед шифруванням, він набагато швидший і ефективніший, ніж алгоритми DES і IDEA.
- Він також забезпечує широку безпеку програмного забезпечення та програм на основі Java, безпечний доступ для інструментів резервного копіювання.
- Підтримує безпечну автентифікацію користувача для віддаленого доступу.

Отже, алгоритм Blowfish є оптимальним рішенням для роботи з невеликими фрагментами даних.

1.2.2. Асиметричні алгоритми шифрування

На відміну від симетричних алгоритмів асиметричні алгоритми шифрування, також відомі як алгоритми з відкритим ключем, використовують два різні ключі для шифрування і розшифрування - публічний і приватний.

Цей підхід кардинально відрізняється від симетричного шифрування, де один ключ використовується для обох процесів. У асиметричному шифруванні публічний ключ є загальнодоступним для всіх, тоді як приватний ключ залишається в таємниці у власника.

Основною перевагою шифрування даного типу є те, що повідомлення, зашифроване публічним ключем, може розшифрувати лише власник відповідного приватного ключа, і навпаки.

Це робить більш ефективними основні функції криптографічних систем – конфіденційність, аутентифікацію та цілісність. Перша забезпечується тим, що відправник шифрує повідомлення публічним ключем отримувача. Це гарантує, що тільки власник відповідного приватного ключа зможе його розшифрувати.

Автентифікація та цілісність забезпечується завдяки тому, що відправник шифрує дані своїм приватним ключем, і будь-хто зможе розшифрувати їх публічним ключем відправника, що підтверджує, що повідомлення справді надійшло від власника приватного ключа.

Одним із основних асиметричних криптографічних алгоритмів є RSA. Він розроблений у 1977 році Роном Рівестом, Аді Шаміром та Леонардом Адлеманом і є одним з найпопулярніших та найбільш використовуваних алгоритмів у сучасній криптографії. Алгоритм RSA базується на математичних операціях з простими числами та складності факторизації добутків великих простих чисел. Для генерації пари ключів необхідно виконати наступні кроки:

1. Генерація простих чисел. Для роботи системи вибираються два великі прості числа p та q.

2. Обчислення модуля n. На наступному кроці обчислюють добуток $n = p \cdot q$. Значення n є частиною як відкритого, так і закритого ключа.

3. Обчислення функції Ейлера $\varphi(n)$. Цей крок полягає у обчисленні значення функції Ейлера $\varphi(n) = (p-1)(q-1)$.

4. Вибір відкритого експонента e. Наступним обирається число e, яке є взаємно простим з $\varphi(n)$.

5. Обчислення закритого експонента. Закрита експонента d є оберненим до e по модулю $\varphi(n)$, тобто $e \cdot d \equiv 1(mod\ \varphi(n))$.

Утворена у результаті розрахунків пара (n, e) є відкритим ключем, який доступний усім, а пара (n, d) є закритим ключем, який зберігається в секреті та необхідний для дешифрування повідомлень.

Завдяки свої властивостям алгоритм RSA вважається безпечним, оскільки розбиття на прості множники великого числа (факторизація) є обчислювально складною задачею і складність підвищується зі збільшенням розміру ключа.

При цьому рекомендований розмір ключа для сучасного рівня безпеки складає 2048 біт або більше. Звичайно ж чим більший розмір ключа, тим вищий рівень захисту, проте це знижує швидкість обробки даних.

Саме тому, коли згадують про RSA, то кажуть що він є повільнішим, ніж симетричні алгоритми, тому його використовують в основному для шифрування даних, як при симетричних ключів у гібридному шифруванні.

RSA залишається важливою технологією в сучасній криптографії, хоча його часто комбінують із симетричними алгоритмами в гібридних системах шифрування, щоб підвищити ефективність і швидкість шифрування.

Іншим поширеним асиметричним криптографічним алгоритмом, розробленим у 1985 році Тахером Ель-Гамалом є алгоритм Ель-Гамаля. Він базується на складності задачі дискретного логарифмування в скінченних групах, що робить його досить надійним для шифрування даних та цифрових підписів. Завдяки своїй стійкості, він став одним з основних асиметричних алгоритмів і використовується у багатьох криптографічних протоколах.

Алгоритм Ель-Гамаля працює з елементами циклічної групи, визначеної модулем p, де p — велике просте число, та основою g, яка є генератором групи. Умовно весь процес роботи алгоритму Ель-Гамаля можна розділити на три етапи: генерація ключів, шифрування та розшифровування.

На першому етапі здійснюється генерація ключів за наступним алгоритмом:

- Вибирається велике просте число p і генератор g.
- Вибирається випадкове приватне число x, яке буде приватним ключем.
- Обчислюється публічний ключ y.
- Пара (p, g, y) є публічним ключем, а x — приватним ключем.

Під час етапу шифрування застосовується наступний алгоритм:

- Для шифрування повідомлення m (представленого у вигляді числа менше p), відправник вибирає випадкове ціле число k, яке буде одноразовим секретом.
- Відправник обчислює значення: $c_1 = g^k \ mod \ p$.
- Шифрує повідомлення m за допомогою публічного ключа y: $c_2 = m \cdot y^k \ mod \ p$.
- Пара (c_1,c_2) є зашифрованим повідомленням, яке відправляється отримувачу.

Для розшифровування необхідно виконати наступний алгоритм:

- Отримувач, маючи приватний ключ x, розшифровує повідомлення, обчисливши значення s: $s = c_1^x \ mod \ p$.
- Отримувач відновлює повідомлення m шляхом обчислення: $m = c_2 \cdot s^{-1} \ mod \ p$.

де s^{-1} мультиплікативне обернене значення s по модулю p.

Перевагами алгоритму Ель-Гамаля можна назвати високу стійкість до атак та складні при декодуванні, проте у нього є і недоліки. Основними недоліками даного алгоритму вважається його низька швидкість та надлишковість, зашифроване повідомлення вдвічі більше за оригінал, оскільки складається з двох компонентів.

Також до асиметричних ключів відносять «алгоритм Діффі-Хеллмана на еліптичних кривих. Він дозволяє двом сторонам, які мають пари відкритий/закритий ключ на еліптичних кривих, отримати загальний секретний ключ, використовуючи незахищений від прослуховування канал зв'язку» [25]. Цей секретний ключ може бути використаний як для шифрування подальшого

обміну, так і для формування нового ключа, який потім може використовуватися для подальшого обміну інформацією за допомогою алгоритмів симетричного шифрування.

Алгоритм Діффі-Хеллмана складається з трьох кроків:

1) вибір загальних параметрів;
2) генерація приватних і публічних значень;
3) обчислення спільного секрету.

На першому кроці публічно обирається велике просте число p і база g, яка є генератором групи модулю p. Значення p та g можуть бути відомі всім.

При генерації приватних і публічних значень кожна зі сторін обирає свій приватний ключ. Нехай сторона A вибирає приватний ключ a, а сторона B вибирає приватний ключ b. Сторони обчислюють свої публічні ключі:

- Сторона А обчислює $A = g^a \ mod \ p$,
- Сторона В обчислює $B = g^b \ mod \ p$.

Після чого сторони обмінюються публічними ключами A і B через відкритий канал.

На третьому кроці кожна із сторін маючи свій приватний ключ і наданий протилежною стороною публічний ключ обчислює спільний секрет:

$K = B^a \ mod \ p$ – для сторони А і $K = A^b \ mod \ p$ – для сторони В.

Таким чином ключ К використовується в подальшому для симетричного шифрування даних, що передаються між сторонами [26].

1.3. Дослідження сфери застосування симетричних та асиметричних алгоритмів

Існує безліч доступних алгоритмів шифрування для забезпечення приватності даних і конфіденційності. Будь-який алгоритм шифрування захистить дані; однак вибір відповідного алгоритму залежить від кількох факторів, таких як показники продуктивності, специфікації системи, складність і рівень забезпеченої безпеки.

Протягом останніх років не одноразово проводилися дослідження ефективності тих чи інших алгоритмів. Так наприклад, у [27] провели порівняльне дослідження наявних на даний момент алгоритмів шифрування, таких як AES, DES, TDES, DSA, RSA, ECC, EEE та RC4, на основі їхньої безпеки, розміру ключа, складності та часу. На основі цього дослідження AES, BlowFish, RC4, E-DES і TDES є найшвидшими алгоритмами з точки зору шифрування, часу, швидкості та гнучкості. Вони дійшли висновку, що AES є найнадійнішим алгоритмом з точки зору швидкості шифрування, складності декодування, довжини ключа, безпеки, а також гнучкості.

У [28] проаналізували продуктивність чотирьох алгоритмів блокового шифрування, таких як AES, DES, TDES і E-DES, на основі швидкості, розміру блоку та розміру ключа. Вони прийшли до висновку, що E-DES перевершує всі інші три моделі на основі вхідних файлів і експериментальних результатів. Він шифрує/дешифрує дані швидше, ніж інші алгоритми. У порівнянні з DES E-DES продемонстрував покращення у двох сферах; більш проста реалізація та важливіші ключові та вхідні блоки для забезпечення безпеки.

У [29] вчені пояснили різні доступні методи шифрування та дешифрування та порівняв їх з точки зору розробки, кількості раундів, довжини ключа, розміру блоку, знайдених атак, рівня безпеки, можливих ключів, часу, необхідного для перевірки всіх можливих ключів тощо. провели порівняння традиційних і гібридних методів шифрування, таких як DSA-RSA, AES-RC4, RC4-AES-SERPENT, SERPENT-RC4 і AES-ECC. Вони дійшли висновку, що AESECC зменшив час і простір, а гібридний алгоритм DSA-RSA мав кращу продуктивність і пропускну здатність.

А у [30] порівняли та проаналізували різні алгоритми шифрування даних як у симетричних, так і в асиметричних категоріях, щоб знайти найкращий алгоритм. Вони порівнювали алгоритми на основі розробки, довжини ключа, кількості раундів, необхідних для шифрування та дешифрування, розміру блоку, різних типів знайдених атак, рівня безпеки та швидкості шифрування. У своєму дослідженні вони помітили, що міцність кожного алгоритму може визначатися керуванням ключами, типом криптографії, кількістю ключів,

кількістю бітів, які використовуються в ключі тощо. Вони дійшли висновку, що BlowFish і ECC мають кращі результати продуктивності. Вони також заявили, що на той час не було повідомлено про успішну атаку на BlowFish, тоді як ECC була успішно атакована.

У [31] провели аналіз різних алгоритмів шифрування, таких як DES, TDES, AES, RSA та BlowFish, на основі їх продуктивності, слабких і сильних сторін. У своєму дослідженні вони дійшли висновку, що BlowFish кращий з точки зору використання пам'яті, часу та пом'якшення атак. Однак, якщо конфіденційність і цілісність є проблемами, тоді AES стає кращим вибором.

У [32] провели порівняння симетричних ключових алгоритмів як у локальній системі, так і в хмарній системі. Для двох систем вони реалізували чотири симетричні ключові алгоритми: AES, DES, BlowFish і DESede. Локальна реалізація була виконана за допомогою Java на eclipse, а хмарна реалізація використовувала eclipse-SDK і Google App Engine. Для їхньої оцінки використовувалися вхідні дані розміром 10 КБ, 13 КБ, 39 КБ і 56 КБ. Вони виявили, що коефіцієнт прискорення DES і BlowFish відображає невелику зміну зі збільшенням розміру вхідних даних, тоді як для AES він зменшується. Вони відзначили, що DESede вимагав більше часу, а BlowFish був найнижчим за часом. У їхньому висновку зазначено, що з точки зору продуктивності BlowFish, AES і DES є кращими алгоритмами, а AES продемонстрував високу безпеку з найменшими витратами часу.

У [33] розроблено легку криптосистему, яку називають простим і високонадійним алгоритмом дешифрування шифрування для зберігання даних у хмарних обчисленнях. Їхня система заснована на алгоритмі шифрування IDEA, і її продуктивність порівнювалася з AES, DES і LED. Запропонований алгоритм працює краще, ніж AES і LED, однак, його продуктивність була трохи повільнішою, ніж DES.

В [34] вчені провели теоретичне дослідження чотирьох популярних симетричних алгоритмів, таких як DES, TDES, AES та BlowFish. Вони порівнювали ці алгоритми на основі різних факторів, таких як швидкість, розмір блоку, захист від атак, конфіденційність, пропускна спроможність,

енергоспоживання, розмір ключа тощо. Згідно з їхнім дослідженням, BlowFish мав кращу продуктивність, враховуючи час шифрування, час дешифрування та пропускну здатність. Вони також дійшли висновку, що TDES був найнижчим з точки зору продуктивності.

У [35] проаналізовано різні симетричні ключові алгоритми, такі як AES, DES, TDES, BlowFish, RC4 і RC6, щодо безпеки, продуктивності, часу обробки та кількості раундів. Результати показали, що BlowFish забезпечив більшу конфіденційність і безпеку при передачі даних по небезпечному каналу, якщо збільшити розмір ключа зі 128 до 448.

У [36] вчені провели порівняння продуктивності між DES, TDES, AES, RC2, RC6 і BlowFish. Вони оцінювали продуктивність цих алгоритмів на основі довжини ключа, методу кодування, типу даних і розміру пакета. Вони виявили, що методи кодування не впливають на процеси шифрування чи дешифрування алгоритмів. BlowFish перевершив інші алгоритми, коли розмір пакета змінено. Крім того, вони показали, що RC2 мав низьку продуктивність і пропускну здатність порівняно з іншими алгоритмами. RC2, RC6 і BlowFish зіткнулися зі значним недоліком порівняно з іншими алгоритмами, коли тип вхідних даних змінився з тексту на зображення. Вони також дійшли висновку, що більша довжина ключа вплине як на енергоспоживання, так і на споживання часу.

У [37] вчені проаналізували DES, TDES, AES, BlowFish, Twofish, Threefish, RC2, RC4, RC5 та RC6 на основі пропускної здатності, масштабованості, безпеки, використання пам’яті, енергоспоживання, швидкості та гнучкості. Їх результати показують, що BlowFish був найефективнішим алгоритмом з точки зору безпеки, гнучкості, використання пам’яті, а також продуктивності.

А у [38] вчені проаналізували DES, TDES і RSA на основі рівня безпеки, часу, необхідного для шифрування/дешифрування, і пропускної здатності. Продуктивність алгоритмів залежить від розміру вхідних даних. Вони резюмували, що швидкість і пропускна здатність DES кращі, ніж у TDES. Крім того, DES показав меншу потужність, ніж TDES і RSA. TDES забезпечив

більшу конфіденційність і масштабованість у цілому, але порівняно з DES і RSA; він споживав більше енергії з меншою пропускною здатністю.

У [39] впровадили та порівняли DES, TDES, AES і BlowFish за допомогою Java та оцінили їх продуктивність на основі різних типів і розмірів вхідних даних, швидкості виконання та різних апаратних платформ. Виходячи з їх порівняння, BlowFish мав кращу продуктивність, ніж інші. Вони ранжували ці алгоритми на основі часу виконання: BlowFish (найшвидший), DES, AES, TDES (найповільніший). Швидкість виконання алгоритмів на основі блочного шифру зростала при збільшенні розміру блоків і зменшенні розміру ключа. Однак в алгоритмах потокового шифрування швидкість зменшується при збільшенні розміру блоку. Вони також дійшли висновку, що безпека, яку забезпечує алгоритм, зростає з кількістю раундів шифрування, хоча це сповільнює швидкість алгоритму.

У [40] автор представив дослідження більшості методів захисту конфіденційності, запропонованих у літературі для хмарних систем. У роботі систематизовано різні методи, доступні в літературі для хмарних систем. Опитування виявило кілька методів, які підпадають під шифрування на основі атрибутів (ABE), шифрування на основі атрибутів ключової політики (KP-ABE), (KPABE), шифрування на основі атрибутів політики шифрованого тексту (CPABE) та багато інших методів. Опитування висвітлює поточні проблеми, пов’язані з кількома запропонованими технологіями захисту для хмари. Перелічені основні проблеми: довіра, контроль доступу та шифрування.

Враховуючи всі вище зазначені дослідження для забезпечення необхідного рівня захисту інформації необхідно поєднати декілька алгоритмів шифрування в один процес. Такі можливості дає використання гібридного шифрування. воно використовує переваги як симетричного, так і асиметричного шифрування, що забезпечує високу безпеку та ефективність передачі даних.

2. КОНЦЕПЦІЯ ТА ПРИНЦИПИ ГІБРИДНОГО ШИФРУВАННЯ

2.1. Принципи роботи асиметричного та симетричного шифрування

Ключовим при роботі з шифруванням даних є розуміння принципів та алгоритмів їх роботи, тому для обґрунтування доцільності гібридного шифрування даних необхідно проаналізувати принципи реалізації асиметричного та симетричного шифрування даних.

Основною перевагою асиметричного шифрування є безпечний обмін ключами. Завдяки своїм властивостям асиметричне шифрування дає можливість передавати публічний ключ відкритими каналами, не ризикуючи компрометацією приватного ключа [41].

Відповідно абонент, що отримав відкритий ключ іншої сторони може зашифрувати будь які дані та переслати їх отримувачу. Використовуючи дві пари асиметричних ключів користувачі можуть без проблем обмінюватись секретною інформацією без загрози її витоку (рис. 2.1).

Рисунок 2.1 - Схема асиметричного шифрування

Представлена на рисунку 2.1 схема відображає процес обміну даними між двома користувачами із застосуванням виключно асиметричного шифрування.

Відправник створює свою пару асиметричних ключів – відкритий та закритий. Для процесу шифрування даних *M*, він отримує по відкритому каналу зв'язку або через хмарне сховище відкритий ключ одержувача. Одержувач заздалегідь надсилає свій відкритий ключ у відповідний канал або сховище, оскільки їх поширення не загрожує витоку інформації.

На наступному кроці зашифровані ключем K_{2_1} дані *C* (шифротекст) пересилаються у канал зв'язку або можуть бути розміщені на доступному багатьом користувачам хмарному сховищі.

Отримувач підключившись до того ж самого каналу обміну даними або мережевого сховища завантажує собі зашифровані дані *C* і розшифровує їх за допомогою приватного ключа K_{2_2}. У випадку, якщо необхідно відіслати відповідь отримувач завантажує собі відкритий ключ відправника K_{1_1} і шифрує інформацію за його допомогою.

Зловмисник маючи доступ до такого каналу або сховище отримає відкритий ключ K_{2_1} та зашифровані дані *C*, проте, не зможе нічого з цим зробити.

В цьому і полягає основна перевага асиметричних методів шифрування перед симетричними, вся інформація може передаватись відкритими каналами без додаткових умов [42].

Проте асиметричне шифрування має і свої недоліки, а саме:

- Низька швидкість. Асиметричні алгоритми, є обчислювально інтенсивними та значно повільнішими за симетричні алгоритми, що особливо помітно при шифруванні великих обсягів даних. Через це асиметричне шифрування зазвичай використовується лише для обміну ключами або для захисту коротких даних (наприклад, цифрових підписів), після чого сторони переходять до симетричного шифрування.
- Вимоги до обчислювальних ресурсів. Для надійності асиметричного шифрування необхідно використовувати ключі великої довжини (наприклад, 2048 або 4096 біт для RSA). Це потребує значних обчислювальних ресурсів і

може бути важким для малопотужних або обмежених за ресурсами пристроїв, таких як мобільні чи IoT-пристрої.

- Уразливість до квантових комп'ютерів. Сучасні асиметричні алгоритми побудовані на математичних задачах, які можуть бути відносно шидко розв'язані квантовими комп'ютерами (наприклад, факторизація цілих чисел). І хоча квантові комп’ютери ще не досягли рівня, необхідного для реального зламу сучасних алгоритмів, їх потенційна загроза є важливим фактором проти вибору асиметричних принципів шифрування.
- Складність управління ключами. Асиметричне шифрування вимагає управління двома типами ключів – публічним та приватним. Безпека залежить від надійного зберігання приватного ключа, адже його компрометація дозволить зловмисникам декодувати зашифровану інформацію або підробляти цифрові підписи.
- Ризик компрометації відкритого ключа. Хоча відкритий ключ не є секретним, якщо він потрапить до зловмисників із підробленою асоціацією до конкретного користувача, зловмисники зможуть перехоплювати та дешифрувати дані, призначені для цього користувача. Це можливо через атаки типу «людина посередині» (MITM), якщо публічні ключі не верифіковані належним чином.
- Відносна складність у розумінні та впровадженні. Асиметричні алгоритми складніші за симетричні, як з математичної точки зору, так і з погляду впровадження в програми та системи. Це може призводити до помилок у реалізації та бути джерелом вразливостей, якщо не дотримуватися всіх правил і рекомендацій.

Тому ключовим при використанні асиметричного шифрування є пошук шляхів компенсації недоліків даної системи шифрування [43].

На відміну від асиметричного симетричне шифрування даних здійснюється одним і тим самим ключем, тому виникають додаткові як переваги його застосування так і недоліки (рис. 2.2).

Рисунок 2.2 - Схема симетричного шифрування

Представлена на рисунку 2.2 схема відображає процес обміну даними між двома користувачами із застосуванням виключно симетричного шифрування, а сам алгоритм можна представити наступним чином:

Крок 1 схожий на асиметричний алгоритм, на ньому генерується ключ для шифрування даних.

На кроці 2 дані *M* шифруються за допомогою ключа *K* симетричного алгоритма шифрування.

На кроці 3 зашифровані дані *C* надсилаються по незахищеному каналі даних або завантажуються на мережеве або хмарне сховище.

Після чого виникає ключова відмінність між симетричним і асиметричним алгоритмом, а саме необхідність передати симетричний ключ. Тому на кроці 4 формується захищений канал передачі даних для надсилання симетричного ключа отримувачу.

На останньому, 5 кроці отримувач використавши отриманий симетричний ключ *K* розшифровує дані *C* та отримує необхідні йому дані *M*.

Відповідно саме виконання процедури розшифрування значно швидше ніж при асиметричному кодуванні [44]. Проте виникає багато проблем із симетричними алгоритмами шифрування, а саме:

- Проблема обміну ключами. Обидві сторони, які хочуть спілкуватися зашифрованим каналом, повинні мати спільний секретний ключ. Передача цього ключа через незахищені канали небезпечна, оскільки існує ризик перехоплення ключа третіми сторонами. Саме ця проблема стала основним стимулом до створення асиметричного шифрування.
- Масштабованість. При збільшенні кількості користувачів однієї системи симетричне шифрування стає все менш зручним. Для кожної пари користувачів потрібен окремий ключ, відповідно у системі з n користувачів потрібно $n(n-1)/2$ ключів для забезпечення зв'язку між усіма парами користувачів, що стає складним і некомфортним для використання.
- Ризик компрометації ключа. У випадку, коли один із ключів перехоплено чи зламано, усі повідомлення, що зашифровані таким ключем, стають вразливими. Симетричне шифрування не передбачає автоматичного оновлення ключів, тому компрометація ключа вимагає його заміни у всіх учасників, що є технічно складним завданням.
- Відсутність підтвердження автентичності. Симетричні алгоритми шифрування зазвичай не підтримують підтвердження автентичності. Іншими словами, немає способу впевнитися, що повідомлення дійсно було надіслане певним користувачем, оскільки обидві сторони мають один і той самий ключ.
- Складнощі з управлінням ключами. Симетричні системи потребують надійного управління ключами, зокрема захищених каналів для їх розповсюдження, зберігання та оновлення.

Враховуючи вище зазначені недоліки, симетричні системи на даний час також не задовольняють всіх вимог що виникають при передачі даних [45].

2.2. Схема та алгоритм гібридного шифрування

На основі наведених раніше властивостей та недоліків асиметричних та симетричних методів шифрування можна провести їх порівняння (табл. 2.1).

Таблиця 2.1

Порівняння симетричного та асиметричного шифрування

	Симетричне шифрування	Асиметричне шифрування
Кількість ключів	1	2
Довжина ключа	128 – 256	1024 – 4096
Швидкість шифрування	Висока, через використання єдиного ключа	Низька, через використання різних ключів
Продуктивність	Висока, оскільки потребує невелику обчислювальну потужність через простий алгоритм шифрування	Низька, потребує більше потужності обчислювання через більш складні алгоритми шифрування
Сфера застосування	Захист даних на дисках і пристроях зберігання; шифрування даних при передачі; захист конфіденційної інформації в додатках; зберігання паролів; шифрування у фінансових транзакціях	Захист комунікацій (TLS/SSL); цифрові підписи; обмін ключами; електронна пошта (PGP, S/MIME); захист даних у хмарі
Рівень захисту	Середній	Високий

Як видно з таблиці 2.1 кожен із методів має свої переваги і недоліки та зазвичай нездатний повністю задовольнити всі вимоги що виникають у процесі їх застосування.

Саме тому останнім часом широко застосовується гібридне шифрування. Воно використовує переваги як симетричного, так і асиметричного шифрування, що забезпечує високу безпеку та ефективність передачі даних. Ключовою властивістю гібридного шифрування є поєднання швидкості та зручності асиметричного шифрування з ефективністю симетричного [46].

У пропонованій схемі гібридного шифрування використовуються сильні сторони обох методів, а саме: асиметричне шифрування для безпечного обміну ключами та симетричне шифрування для швидкого та ефективного шифрування великих обсягів даних (рис. 2.3).

Рисунок 2.3 - Загальна схема гібридного шифрування

Відповідно алгоритм для гібридного шифрування складатиметься з наступних кроків:

Крок 1. Генерація симетричного ключа. Відправник створює випадковий симетричний ключ K, який буде використаний для шифрування даних.

Крок 2. Шифрування даних симетричним ключем. На поточному кроці відправник шифрує дані M за допомогою симетричного ключа K, отримуючи у результаті зашифровані дані C.

Крок 3. Шифрування симетричного ключа асиметричним алгоритмом. Щоб захистити симетричний ключ K, відправник шифрує його за допомогою публічного асиметричного ключа отримувача, отримуючи зашифрований ключ E_K.

Крок 4. Надсилання зашифрованих даних. На поточному кроці відбувається надсилання зашифрованих даних C та зашифрованого симетричного ключа E_K.

Крок 5. Розшифрування симетричного ключа. Отримувач, маючи свій приватний ключ, який є парним до публічного ключа, використаного для шифрування симетричного ключа на кроці 3, розшифровує зашифрований

симетричний ключ E_K, отримуючи у результаті оригінальний симетричний ключ K.

Крок 6. Розшифровування даних. Отримувач, маючи оригінальний симетричний ключ K, використовує його для розшифровування зашифрованих даних C, отримуючи в результаті оригінальні дані M.

Як бачимо запропонований алгоритм поєднує у собі переваги симетричного та асиметричного шифрування, а саме:

- використовує симетричний ключ для шифрування даних, що збільшує як швидкість шифрування, так і швидкість дешифрування даних;
- використовує загальний канал для пересилання шифрованих даних та не потребує окремого захищеного каналу для пересилання ключа кодування;
- шифрує симетричний ключ за допомогою публічного ключа одержувача, що забезпечує достатній рівень захисту та дозволяє пересилати його відкритим каналом зв'язку.

Отже, як видно із запропонованого алгоритму він повинен складатися і з симетричного і з асиметричного алгоритмів шифрування.

Як зазначалося в першому розділі симетричні алгоритми шифрування мають декілька поширених варіантів, серед яких найвідоміші – AES, DES, TDES та Blowfish.

2.3. Вибір симетричного алгоритму

Ключовими параметрами при порівнянні алгоритмів є довжина ключа, швидкість, рівень безпеки та поширення [47].

В таблиці 2.2 наведено порівняння вище згаданих симетричних алгоритмів.

Таблиця 2.2

Порівняння найвідоміших симетричних алгоритмів шифрування

Алгоритм	Довжина ключа, біт	Швидкість	Рівень безпеки	Поширення
DES	56	Висока	Низький	Не застосовується
TDES	112, 168	Низька	Низький	Не застосовується
AES	128, 192, 256	Висока	Високий	Сучасний стандарт
Blowfish	32-448	Висока	Високий	Сучасний стандарт
Twofish	<256	Середня	Середній	Сучасний стандарт

Як видно з таблиці 2.2 DES та TDES вийшли з ужитку і в даний час нові криптографічні системи на їх основі вже не розробляються. AES, Blowfish та Twofish є сучасними стандартами, які широко застосовуються у криптографічних системах.

AES працює з блоками даних розміром 128 біти та підтримує ключі довжиною 128, 192 та 256 біт [12]. Він широко використовується для забезпечення конфіденційності даних у багатьох практичних реалізаціях, оскільки він забезпечує високий рівень безпеки та відносно високу продуктивність. Основні сфери використання AES включають захист інтернет-з'єднань (SSL/TLS), Wi-Fi мереж, VPN-з'єднань, а також шифрування дисків і хмарних даних. Завдяки цьому AES став ключовим стандартом шифрування, широко застосовуваним для забезпечення конфіденційності у різних сферах — від захисту мережевих з'єднань до безпеки даних на мобільних пристроях.

Blowfish працює з блоками даних розміром 64 біти та підтримує ключі змінної довжини від 32 до 448 біт, що забезпечує додаткову гнучкість у налаштуванні рівня безпеки [24]. Його часто використовують для гешування паролів у поєднанні з функцією bcrypt, яка є модифікацією Blowfish.

Bcrypt використовує Blowfish для шифрування та спеціальну структуру, що включає сіль і параметр обчислювальної складності. Це робить bcrypt стійким до атак підбору паролів і дозволяє адаптувати гешування до зростаючої обчислювальної потужності. Проте ключовою сферою застосування Blowfish є захист даних у вбудованих системах та пристроях з обмеженими ресурсами.

Blowfish завдяки своїй швидкодії і невеликим вимогам до пам’яті може використовуватися для шифрування критичних даних у вбудованих системах, таких як сенсори або IoT-пристрої.

Twofish працює з блоками даних розміром 128 біт і підтримує ключі довжиною до 256 біт, що дозволяє гнучкіше налаштовувати рівень безпеки у залежності від вимог [48].

Оскільки він має відкритий код, Twofish є популярним вибором у багатьох безкоштовних програмах і для шифрування конфіденційних даних у випадках, коли AES не є обов’язковим стандартом. Основною сферою застосування цього алгоритму є шифрування даних на носіях та в хмарах, а також різноманітне програмне забезпечення з відкритим кодом. Twofish забезпечує високу криптостійкість, гнучкість, швидкість та ефективність і залишається хорошою альтернативою алгоритму AES.

Незважаючи на ключові переваги AES, він є менш гнучким чим Blowfish та Twofish, що є критичним при роботі з вбудованими системами, такими як розумний будинок, системи сенсорів та багато інших, тому його використання в рамках роботи є недоцільним.

Обираючи між Blowfish та Twofish варто зазначити, що обидва алгоритми створені Брюсом Шнайєром, але з різними характеристиками та сферою застосування. Хоча Twofish вважається вдосконаленою версією Blowfish і має певні переваги, є також кілька аспектів, у яких Blowfish є більш вигідним:

- Складніша реалізація та налаштування. Twofish є більш складним для реалізації, ніж Blowfish, що може бути проблемою для розробників, які бажають інтегрувати алгоритм у свої додатки без значних витрат на розробку. Blowfish має відносно просту архітектуру, що спрощує його впровадження, особливо для розробників, які не мають великого досвіду у криптографії. Це робить Blowfish зручнішим вибором для швидкої інтеграції у програми з відкритим кодом або для програм, де потрібен базовий рівень безпеки.
- Повільніша робота на деяких платформах. Через більшу складність алгоритму Twofish може працювати повільніше на деяких пристроях з обмеженими ресурсами, зокрема на вбудованих системах. Хоча Twofish

оптимізований для високопродуктивних систем і може працювати швидше на сучасних процесорах, він менш ефективний на пристроях зі слабким апаратним забезпеченням, де простіший Blowfish демонструє кращу швидкодію. Таким чином, для швидкодії на пристроях з обмеженими ресурсами Blowfish є вигіднішим вибором.

- Перевага Blowfish у швидкості при меншій довжині ключа. Blowfish працює швидше за Twofish, коли використовується короткий ключ (до 128 біт) і коли захист не потребує найвищого рівня безпеки. Blowfish має меншу кількість раундів шифрування порівняно з Twofish, що зменшує час обробки даних. У програмах, де швидкість є критично важливою, а дані не потребують високої стійкості до атак (наприклад, у випадкових сесіях), Blowfish може бути вигіднішим за Twofish.

- Проблеми з апаратним прискоренням. Twofish не має такої ж підтримки апаратного прискорення, як AES, що є більш вагомим у певних випадках, але це також ставить Twofish у невигідне становище порівняно з Blowfish у програмах, де відсутня необхідність підтримки апаратного прискорення. Blowfish часто застосовується там, де апаратне прискорення не має значення (наприклад, в IoT-пристроях і вбудованих системах), і забезпечує необхідний рівень продуктивності на програмному рівні.

- Складна структура S-блоків. Twofish використовує складнішу структуру S-блоків, які мають псевдовипадкову природу і змінюються на основі ключа. Це збільшує час ініціалізації та підготовки до роботи з алгоритмом, що може бути недоліком для додатків, які повинні швидко починати шифрування. Blowfish, з іншого боку, має простішу структуру S-блоків, що дозволяє розпочати обробку даних швидше.

- Обмежена підтримка в програмному забезпеченні та стандартних бібліотеках. Twofish не набув такої популярності, як Blowfish, і тому має обмежену підтримку у стандартних криптографічних бібліотеках та програмному забезпеченні. Blowfish доступний у багатьох бібліотеках і є підтримуваним у більшості мов програмування та криптографічних платформ, тоді як Twofish часто потребує додаткових зусиль для інтеграції.

У порівнянні з Twofish Blowfish виграє за рахунок простішої реалізації, швидшої роботи на обмежених пристроях, меншого навантаження на систему при коротших ключах і кращої підтримки у багатьох стандартних бібліотеках. Це робить Blowfish кращим вибором для програм, де простота реалізації, швидкодія і базова безпека є ключовими вимогами, а значить краще підходить для інтеграції як складова системи гібридного шифрування.

2.4. Вибір асиметричного алгоритму

2.4.1. Порівняння найвідоміших асиметричних алгоритмів шифрування

Іншим необхідним фрагментом гібридного шифрування є асинхронний алгоритм шифрування, що використовується для шифрування синхронного ключа при передачі через відкритий канал зв’язку.

Як же зазначалося раніше асиметричні алгоритми шифрування використовують пару ключів — публічний та приватний. Існує кілька найвідоміших алгоритмів асиметричного шифрування, особливості, переваги та недоліки яких представлено в таблиці 2.3.

У порівнянні з симетричними алгоритмами усі перелічені асиметричні алгоритми забезпечують високий рівень безпеки при використанні ключів рекомендованої довжини, тому відповідну характеристику в таблиці представляти не потрібно.

RSA є «одним із найбільш популярних асиметричних алгоритмів шифрування, який використовується для забезпечення безпеки передачі даних, автентифікації та цифрових підписів» [31]. Його застосовують всюди, як для захисту інтернет-з'єднань (HTTPS / TLS), так і для цифрових підписів та систем електронної пошти. RSA забезпечує високий рівень безпеки, гнучкість у застосуванні та широко підтримується в криптографічних стандартах і бібліотеках, що робить його загальнодоступним та універсальним.

Таблиця 2.3

Порівняння найвідоміших асиметричних алгоритмів шифрування

Алгоритм	Довжина ключа, біт	Швидкість	Складність реалізації	Підтримка стандартами	Сфера використання
RSA	2048-4096	Середня	Середня	Висока	Підписи, шифрування
ECC	224-521	Висока	Висока	Висока	Підписи, шифрування
DSA	1024-3072 біт	Висока	Середня	Середня	Цифрові підписи
Ель-Гамаля	2048	Низька	Середня	Середня	Підписи, шифрування
На основі ґраток	Залежно від типу	Низька	Середня	Низька	Пост-квантова криптографія

Проте, RSA має і деякі недоліки, а саме великий розмір ключів, що може знижує швидкість роботи алгоритму та низьку швидкість в порівнянні з іншими асиметричними алгоритмами.

Еліптична криптографія (ECC) є сучасним методом асиметричного шифрування, який використовує математичні властивості еліптичних кривих для створення надійного захисту [49]. Основною перевагою ECC є можливість досягти високого рівня безпеки з меншими розмірами ключів порівняно з RSA, що робить ECC особливо привабливим для обмежених за ресурсами пристроїв, таких як мобільні пристрої та IoT. Вона широко використовується, як для захисту інтернет-з'єднань (HTTPS / TLS), так і для цифрових підписів (ECDSA), мобільних платежів, криптовалют та інтернету речей. ECC забезпечує еквівалентний рівень безпеки з коротшими ключами порівняно з RSA.

Наприклад, 256-бітний ключ ECC відповідає рівню безпеки 3072-бітного ключа RSA, завдяки чому збільшується швидкість та економиться пам'ять та потужність процесорів, що особливо актуально для мобільних пристроїв і IoT. ECC має чимало переваг, однак при її застосуванні в складних і критичних системах важливо враховувати й недоліки. До них належать: складність впровадження, патентні обмеження і вимоги ліцензування, чутливість до

побічних атак, недостатній захист від квантових обчислень, проблеми зі стандартизацією, потреба в глибоких криптографічних знаннях для налаштування, відносна новизна технології та можливості апаратної реалізації.

Алгоритм цифрового підпису (DSA) широко використовується для створення цифрових підписів і забезпечення аутентичності даних [50]. Його основна функція – перевірка того, що повідомлення або документ справді походить від вказаного відправника і не був змінений. Загалом DSA застосовується виключно для забезпечення захищеного обміну даними і перевірки автентичності, особливо в системах, що потребують високого рівня безпеки та захисту від несанкціонованих змін. Як і зрозуміло з назви, основним застосуванням є цифровий підпис документів, відповідно не може застосовуватись як один із елементів гібридного шифрування.

Алгоритм Ель-Гамаля детально розглядався у пункті 1.2.2, де було описано його алгоритм а також ключові переваги та недоліки. Його використання у якості одного із компонентів системи гібридного шифрування цілком обґрунтоване незважаючи на подвійну надлишковість при шифруванні. Такий недолік не є ключовим за умови передачі в мережу виключно зашифрованого симетричного ключа, а висока складність дешифрування та висока стійкість є необхідними умовами для захисту ключа при передачі.

Криптографічні алгоритми шифрування на основі ґраток останнім часом набувають все більшого розвитку, оскільки відносяться до пост-квантової криптографії [51]. Квантові комп'ютери потенційно можуть зламати сучасні алгоритми, такі як RSA та ECC, за допомогою ефективних методів факторизації й дискретного логарифмування.

Алгоритми шифрування на основі ґраток ґрунтуються на складності розв’язання задач, пов’язаних з ґратками, таких як проблема найкоротшого вектора (SVP) або найближчого вектора (CVP). Завдяки своїм властивостям ці алгоритми вважаються перспективними в епоху постквантової криптографії. Також, алгоритми на основі ґраток входять до стандартів постквантової криптографії, запропонованих NIST. Ключовими перевагами алгоритму на основі ґраток можна назвати квантову стійкість, гнучкість та масштабованість.

Проте є і певні недоліки до яких відносяться великий розмір ключів, висока обчислювальна складність, не достатня поширеність та вразливість до спеціалізованих атак.

Як уже зазначалося раніше, планований гібридний алгоритм використовуватиме асиметричний алгоритм тільки для шифрування симетричного ключа, тому великий розмір ключів не є недоліком для планованої системи.

Те саме відноситься і до наступного ключового недоліку – високої обчислювальної складності. Складність при обчисленні лише симетричного ключа є несуттєвою в порівнянні з об'ємом даних що будуть надсилатися.

Наступним з озвучених недоліків є низька поширеність алгоритму, що призводить до складності його реалізації та експлуатації. При впровадженні таких алгоритмів можуть виникати проблеми, що не мають рішення у спільнотах. Спеціалізовані атаки на криптографічні алгоритми, які використовують ґратки, часто спрямовані на розкриття приватної інформації за допомогою аналізу структури ґраток або її математичних властивостей. Основні типи спеціалізованих атак на криптографічні алгоритми, які використовують ґратки представлено в таблиці 2.4.

Таблиця 2.4

Аналіз атак на криптографічні алгоритми, які використовують ґратки

№ з/п	Тип атаки	Опис	Методи	Захист
1	2	3	4	5
1	Атака на найкоротший вектор (SVP)	Пошук найкоротшого вектора у ґратках для розкриття секретних даних	- Алгоритм LLL (Lenstra-Lenstra-Lovász) для низьких розмірностей. - Алгоритм BKZ для високих.	- Використання великих розмірностей ґраток. - Застосування сильного шуму.
2	Атака на найближчий вектор (CVP)	Відновлення найближчого вектора до заданої точки у ґратках для розкриття ключів.	- Євклідові проєкції. - Перебір можливих варіантів.	- Збільшення розмірності ґраток. - Зміцнення базису.

Прод. табл 2.4

1	2	3	4	5
3	Атака на проблему навчання з помилками (LWE)	Відновлення секретного ключа через зменшення впливу або видалення шуму.	- Аналіз шуму. - Гібридні методи (поєднання ґраткових атак і перебору).	- Використання великих параметрів шуму. - Налаштування параметрів за стандартами.
4	Атака через обмежені параметри	Використання слабких параметрів (малі розміри ґраток, слабкий базис).	- Генерація альтернативних ґраток. - Використання залежностей між параметрами.	- Забезпечення високої розмір-ності ґраток. - Оптимізація вибору базису.
5	Побічні атаки (Side-Channel Attacks)	Використання витоків інформації з реалізації (час, енергія, випромінювання).	- Аналіз часу обчислень. - Аналіз спожи-вання енергії. - Атаки на основі шуму.	- Маскування даних. - Захист апарат-ної реалізації. - Випадкові зміщення часу.
6	Декодування Бабая (Babai’s Decoding)	Використання алгоритму декодування Бабая для наближеного розв'язання CVP.	- Наближення координат у базисі ґраток. - Аналіз слабких базисів.	- Використання сильних базисів. - Додавання значного шуму.
7	Атака на кільцеву структуру (Ring-LWE)	Використання специфічних властивостей кільця у схемах, заснованих на Ring-LWE.	- Аналіз математичних залежностей кільця. - Використання слабких параметрів кільця.	- Перехід до загальнішої форми LWE без використання кільця. - Підвищення параметрів шуму.

Зазначені в таблиці 2.4 атаки можуть бути дуже потужними, але складність їх реалізації різко зростає із збільшенням розмірності ґраток та якісною генерацією шуму. Щоб забезпечити необхідний рівень захисту криптосистем на основі ґраток, слід дотримуватися сучасних стандартів параметрів та захищати реалізації від побічних атак.

2.4.2. Постквантові асиметричні алгоритми

З розвитком квантових комп’ютерів, які завдяки складнішим алгоритмам, таким як Шора (для задач факторизації) або Гровера (для завдань пошуку), здатні ефективно ламати більшість класичних криптографічних алгоритмів,

таких як RSA, ECC та DSA виникає потреба в нових криптографічних алгоритмах [52]. Постквантова криптографія, або квантово-стійка криптографія, займається створенням алгоритмів та систем шифрування, що забезпечать захист як від класичних, так і від квантових обчислювальних атак. Необхідність у таких рішеннях виникає через загрозу, яку становлять квантові комп'ютери для сучасних криптографічних методів. Вони базуються на складних математичних задачах, які важко вирішуються за допомогою традиційних цифрових технологій.

На сьогодні існує кілька видів квантово-стійких алгоритмів, які можуть слугувати основою для створення криптографічних систем [51].

Перший, це згадувана раніше криптографія на основі ґраток. Криптографічні системи на основі ґраток зазвичай відзначаються простотою алгоритмів, високою ефективністю та можливістю паралельного виконання. Алгоритми цього типу використовують базові математичні операції, такі як множення матриць і модульна арифметика, що дозволяє легко реалізувати їх на апаратному рівні та оптимізувати.

Іншим, досліджуваним міжнародною спільнотою, видом є мультиваріативна криптографія або як її ще називають криптографія на основі багатоваріантних поліноміальних рівнянь. Цей напрямок криптографії базується на задачах, що пов’язані із розв’язанням систем багатоваріантних нелінійних рівнянь над скінченними полями. Цей підхід вважається перспективним для захисту від квантових комп’ютерів, оскільки такі задачі залишаються складними навіть для квантових алгоритмів. Встановлено, що розв’язання багатовимірних поліноміальних рівнянь є NP-складною задачею. Однак методи MC вразливі до алгебраїчних, диференціальних атак та атак, що використовують базиси Грьобнера. Через це мультиваріативна криптографія не набула широкого застосування на практиці [53].

Третім, поширеним напрямом є криптографія на основі гешування. Цей криптографічний напрямок використовує властивості криптографічних геш-функцій для створення стійких схем шифрування та цифрових підписів. Основною перевагою цих алгоритмів є їх стійкість до атак квантових

комп’ютерів. Попри переваги, криптографія на основі гешування має свої недоліки. По-перше, її безпека може бути під загрозою, якщо зловмисникам вдасться знайти колізії в геш-функції. По-друге, для забезпечення стійкості до атак потрібно використовувати ключі значної довжини. Це зумовлено тим, що алгоритм Гровера може зменшити складність зламу геш-функції до $O(2^{n/2})$, де n - довжина ключа, у той час як класичні алгоритми мають експоненційну складність. Проте цей підхід є обмежено ефективним для коротких ключів. Збільшення розміру ключа, у свою чергу, уповільнює обробку даних та підвищує вимоги до необхідної пам’яті.

Інші відомі напрями постквантової криптографії, такі як криптографія на основі кодів коригування помилок та криптографія на основі лізогенії потребують значно більших витрат та потужностей при реалізації [54]. Згідно рішення Національного інститут Стандартів та Технологій (NIST) визначено п’ять рівнів безпеки постквантових крипто алгоритмів[55]:

1. Алгоритм повинен бути настільки ж складним для компрометації, як і процес підбору ключів у симетричному шифрі з довжиною ключа 128 біт;
2. Алгоритм має відповідати рівню стійкості 256-бітної геш-функції;
3. Алгоритм повинен забезпечувати стійкість, еквівалентну симетричному шифру з ключем довжиною 192 біти;
4. Алгоритм має відповідати рівню захисту 384-бітної геш-функції;
5. На найвищому рівні алгоритм має гарантувати захист, еквівалентний симетричному шифру з ключем довжиною 256 біт.

Аналіз продуктивності квантово-стійких алгоритмів здійснюється в рамках проекту OQS, який займається розробкою та тестуванням таких криптоалгоритмів. Головною метою OQS є сприяння стандартизації постквантової криптографії за версією NIST для механізмів захищеної передачі KEM ключів та схем цифрового підпису.

Користувачам зазвичай не потрібно, щоб алгоритм відповідав усім п’яти рівням безпеки. Для функціонування повноцінної системи достатньо дотримання двох або трьох із них. Наприклад, алгоритм CRYSTALS-KYBER забезпечує відповідність рівням 1, 3 та 5. Його безпека ґрунтується на

складності розв’язання задачі навчання з помилками на модульній ґратці (MLWE). Алгоритм характеризується високою швидкістю обчислень (може генерувати від 9 тисяч до 90 тисяч пар ключів за секунду, залежно від обраного рівня безпеки) і компактністю ключів: відкриті ключі мають розмір від 800 до 1568 байтів, а закриті — від 1632 до 3168 байтів [56].

У 1996 році був розроблений алгоритм NTRU, основою якого є складність розв'язання проблеми кільцевого навчання із помилками [57]. Він стійкий до атак на основі алгоритму Шора та має короткий ключ, в якому співвідношення довжин відкрито та закритого ключів рівне 1/~1.3.

Іншим пост квантовим алгоритмом на основі ґраток є SABER [58]. Він ґрунтується на складності вирішення проблеми модульного навчання із округленням і є простим та ефективним алгоритмом, що пропонує три рівня безпеки згідно вимог NIST та потреб користувачів:

1. LightSABER, який забезпечує постквантовий рівень безпеки, еквівалентний AES-128, відповідаючи 1-му рівню стандарту NIST;
2. SABER, який гарантує постквантовий рівень безпеки, аналогічний AES-192, відповідаючи 3-му рівню стандарту NIST;
3. FireSABER, який забезпечує постквантовий рівень безпеки, порівнянний із AES-256, відповідаючи 5-му рівню стандарту NIST.

Також, серед оголошених NIST алгоритмів постквантового шифрування присутні FrodoKEM, CRYSTALS-DILITHIUM та FALCON. Вони всі були оголошені у 3-му раунді та належить до сімейства криптографії на основі ґраток. Проте їх реалізація значно складніша та потребує більших затрат.

У 2019 році було заявлено про те, що IBM створили квантовий комп’ютер з процесором на 27 кубітів, а у 2021 році – повідомили про потужність 127 кубітів. Такий прогрес є дуже швидким, тому NIST враховує це при затверджені криптографічних алгоритмів.

2.5. Загрози для існуючих крипто алгоритмів

Квантові алгоритми здатні зламувати існуючі криптографічні системи, які ґрунтуються на обчислювальній складності відповідних математичних задач. Наприклад, RSA ґрунтується на складності розкладання великого цілого числа на прості множники. Квантовий алгоритм Шора дозволяє ефективно виконувати цю операцію, що робить RSA вразливою до атак такого типу.

Алгоритм Шора це квантовий метод, призначений для розкладання великих цілих чисел на прості множники, такий процес ще називають факторизацією. Вона є складною задачею для класичних комп'ютерів та слугує основою криптостійкості багатьох алгоритмів наприклад, таких як RSA [59].

Алгоритм Шора використовує квантову суперпозицію та інтерференцію для визначення множників великого числа. Процес починається з переведення вхідного числа у стан суперпозиції, у якому система перебуває у всіх можливих станах одночасно з певними ймовірностями. Далі застосовується модульна функція піднесення до степеня та квантова інтерференція, яка посилює правильні відповіді та усуває хибні. На останньому етапі класичні алгоритми аналізують періодичність отриманих значень, що дозволяє визначити множники числа.

Як приклад, розглянемо атаку на RSA, де модуль N є добутком двох великих простих чисел p та q. Злам RSA передбачає факторизацію N, що дозволяє обчислити значення Ейлера $\phi(n)$ і, зрештою, визначити приватний ключ на основі публічного. Алгоритм Шора, що шукає ціле число d, на яке ділиться N матиме наступний вигляд:

1. Обирається випадкове число $r<N$, де r та N взаємно прості.
2. Квантовий комп'ютер визначає невідомий період p для функції $f_{r,N}(x) = r^x mod\ N$.
3. Якщо p - непарне, то перший крок повторюємо, якщо p парне, переходимо до наступного.
4. Використовуємо співвідношення $(r^{\frac{p}{2}} - 1) \cdot (r^{\frac{p}{2}} + 1) = 0\ mod\ N$.

5. Якщо $(r^{\frac{p}{2}} + 1) = 0 \, mod \, N$, повторюємо з початку, інакше переходимо до наступного кроку.
6. Обчислюємо $d = gcd(r^{\frac{p}{2}} - 1, N)$, де gcd — функція для пошуку найбільшого спільного дільника, як результат отримуємо один із множників N.

Алгоритм Шора дозволяє виконати факторизацію числа N за час $O(log^3 N)$, використовуючи $O(log N)$, кубітів. Він ефективний для великих значень N, оскільки його складність логарифмічна, на відміну від експоненціальної у класичних алгоритмів. З появою квантових комп'ютерів із тисячами кубітів стане можливим зламування криптографічних систем із відкритим ключем, такі як RSA.

Хоча алгоритм Шора має значно вищу швидкість у порівнянні з класичними методами, його реалізація потребує великої кількості кубітів, що наразі перевищує можливості сучасних квантових комп'ютерів.

Іншою загрозою для сучасних криптографічних алгоритмів є алгоритм Гровера. Він здатен значно прискорити процес пошуку в несортованих базах даних. Цей алгоритм виконує пошук за час O($\sqrt{N}$), де N — розмір бази даних та може бути застосований для швидкого знаходження значення геш-функції, що відповідає заданому вхідному повідомленню. Це становить потенційну загрозу для криптографічних геш-функцій, які забезпечують цілісність даних, оскільки зловмисники можуть використати його для спроб зламу.

Атаки на геш-функції зазвичай базуються на методі перебору, який може бути вкрай трудомістким, особливо за умови використання геш-функцій із великим розміром дайджесту. Наприклад, для зламу першої половини (128 біт) результату SHA-256 знадобилося понад 7 місяців і близько 40 МВт електроенергії, використовуючи два біткоїн-майнери та суперкомп'ютер «Blue Gene/Q» [60].

Якщо геш-функція застосовується для збереження паролів, зловмисники можуть намагатися підібрати пароль, перебираючи всі можливі комбінації значень. У разі, якщо геш-функція забезпечує захист від змінення пакетів

даних, зловмисники можуть скористатися алгоритмом Гровера для пошуку значення геш-функції, яке дозволить модифікувати пакети даних без порушення цілісності гешу.

Алгоритм Гровера може бути використаний для пошуку колізій у геш-функціях, тобто для знаходження двох різних повідомлень, які дають однакове геш-значення. Він дозволяє шукати в просторі можливих повідомлень, які створюють задане геш-значення, за час $O(\sqrt{N})$, тоді як класичні алгоритми вимагають часу $O(N)$, де N — кількість можливих повідомлень. Наприклад, геш-функція SHA-256 генерує 2^{256} можливих повідомлень, на що витрачає час $O(2^{128})$. Квантовий комп'ютер, що використовує алгоритм Гровера, за ідеальних умов потребував би приблизно 2^{128} оцінок геш-функції, щоб знайти початкове повідомлення від якого отримано задане геш-значення, що значно ефективніше у порівнянні з класичним підходом, який вимагає 2^{256} оцінок. Можна сказати, що квантовий комп'ютер міг би зламати SHA-256 за ≈4 місяці, що є значним прискоренням у порівнянні з класичними підходами, які потребують близько 14 місяців. Однак навіть таке прискорення може виявитися недостатнім для практичного зламу великих геш-функцій.

Таким чином, алгоритм Гровера може бути використаний для подолання колізійної стійкості геш-функцій. Проте і цю вразливість можна зменшити за рахунок:

- використання геш-функцій з більшими вихідними дайджестами (що збільшує простір пошуку і ускладнює атаку алгоритму Гровера),
- застосування постквантових криптографічних алгоритмів,
- впровадження квантово стійких схем підпису.

Квантовий підхід може становити загрозу для криптографічних геш-функцій, таких як MD5, SHA-1, SHA-2 і SHA-3, проте для геш-функцій із великим розміром дайджесту, таких як SHA-512, теоретично потрібно 2^{256} операцій для зламу за допомогою алгоритму Гровера, що є недосяжним як для класичних, так і для сучасних квантових комп'ютерів.

3. ПРАКТИЧНА РЕАЛІЗАЦІЯ ГІБРИДНОГО ШИФРУВАННЯ

3.1. Алгоритми гібридного шифрування

У пункті 2.2 даної роботи було представлено схему гібридного шифрування, яка поєднує швидкість симетричного шифрування та метод безпечного обміну ключами на основі асиметричного шифрування даних.

Як зазначалося раніше симетричне шифрування завдяки використанню одного і того ж ключа є значно ефективнішим та швидшим при шифруванні даних не залежно від їх розміру.

Алгоритм шифрування можна зобразити наступним чином (рис. 3.1):

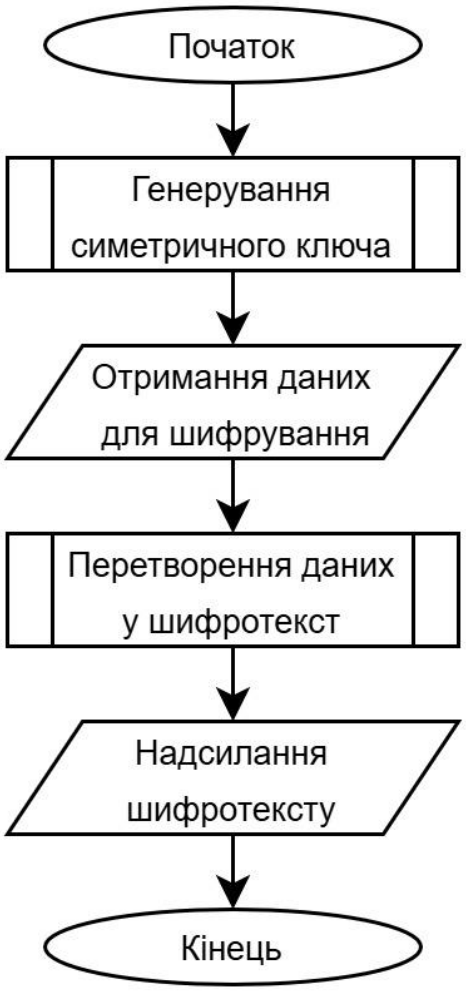

Рисунок 3.1 – Алгоритм шифрування пакетів даних

Як видно з рисунку 3.1 алгоритм шифрування є лінійним процесом, результатом якого буде отриманий шифротекст, готовий для надсилання по мережі або для зберігання на віддаленому хмарному чи локальному сховищі. Завдяки лінійності даного процесу він буде виглядати однаково не залежно від типу шифрування що застосовується, але однією із ключових проблем при симетричному шифруванні є метод передачі симетричного ключа.

Як зазначалося раніше в гібридних методах шифрування для цього застосовують асиметричну складову. Ключовою властивістю асиметричних алгоритмів є те, що вони складаються з двох ключів один із яких є публічним та може вільно передаватися до відправника будь яким способом.

При отриманні публічного ключа відправник згідно пропонованого алгоритму (пункт 2.2), повинен зашифрувати симетричний ключ (рис. 3.2).

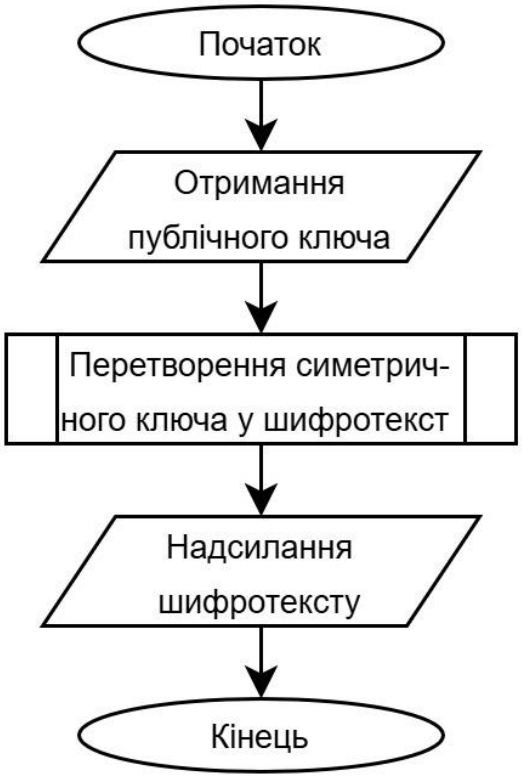

Рисунок 3.2 – Алгоритм шифрування симетричного ключа

У результаті виконання даного алгоритму симетричний ключ буде зашифрований із застосуванням публічного ключа отримувача та надісланий йому. У випадку, якщо зловмисник зміг перехопити даний пакет даних,

одержана інформація не дасть йому ніякої користі, оскільки згідно властивостей асиметричних алгоритмів для розшифрування повідомлення необхідний приватний ключ.

Можна зазначити, що ключовою перевагою запропонованого алгоритму є те, що для передачі можуть застосовуватися будь які канали зв'язку. При одержані обох повідомлень (шифротексту масиву даних та симетричного ключа) отримувач використовує приватний ключ для одержання симетричного ключа до масиву даних (рис. 3.3).

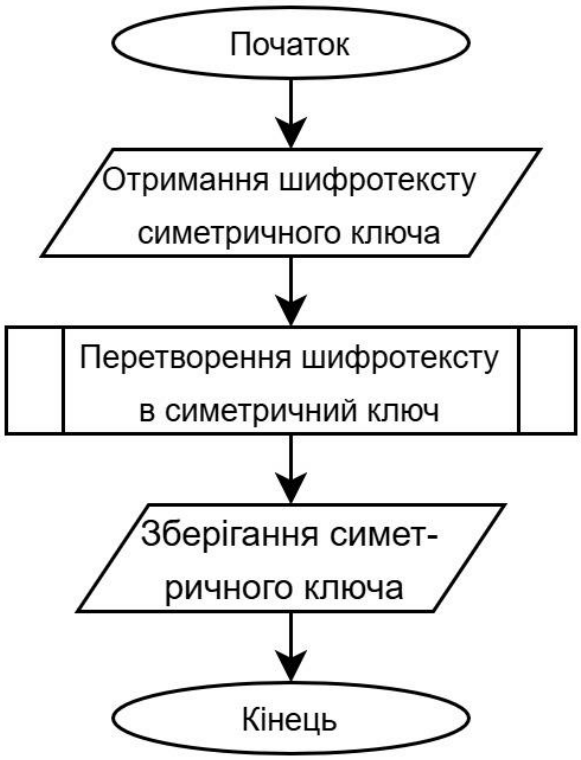

Рисунок 3.3 – Алгоритм дешифрування симетричного ключа

Відповідно у результаті виконання алгоритму одержувач отримає симетричний ключ, що був використаний для шифрування масиву даних. Суттєвою перевагою даного методу в конкретній ситуації є те, що кожен із масивів даних що надсилається може бути зашифрований іншим симетричним ключем. Це убезпечує користувачів від випадкової компрометації симетричного ключа. Якщо у результаті роботи відправника чи одержувача відбулося випадкове розголошення симетричного ключа, то до рук

зловмисників може попасти лише один пакет даних. Відповідно у залежності від конфіденційності інформації що надсилається допускається як надсилання кожного пакета інформації зашифрованим одним симетричним ключем, так і шифрування кожного пакету з допомогою іншого симетричного ключа.

Даний метод особливо ефективний при атаках типу Man-in-the-Midle (людина в середині). У результаті кібератак такого типу зловмисник таємно перехоплює і замінює спілкування між двома сторонами, які вважають, що взаємодіють напряму. Пропонований метод дозволяє з легкістю зауважити атаки такого типу, оскільки достатньо одноразового підтвердження, що ключ отриманий відправником належить справді отримувачу і внести будь які зміни в обмін даними значно складніше.

Згідно сучасних досліджень [61] навіть сучасним квантовим алгоритмам, таким як алгоритм Шора, необхідно приблизно 4 місяці для злому алгоритму SHA-256. Враховуючи сучасні об’єми інформації що обробляється та передається в мережі, а також швидкість старіння інформації та втрати нею цінності такий термін для дешифрування вже є не актуальним у більшості випадків.

Тому на основі проведених у пункті (2.3) досліджень для використання у гібридному методі шифрування пропонується використання симетричний алгоритм Blowfish, який забезпечує високу швидкість обробки даних та ефективно шифрувати великі файли або потоки даних з невеликим обчислювальним навантаженням. Blowfish має фіксований розмір блоку 64 біти та ключі змінної довжини (від 32 до 448 біт), що забезпечує гнучкість і надійність.

Враховуючи наведені у роботі дані для ефективної реалізації сегменту асиметричного шифрування цілком достатньо та ефективно буде застосовувати алгоритм на основі еліптичних кривих.

3.2. Реалізація симетричного фрагменту алгоритму

На рисунку 3.1 було представлено алгоритм шифрування, першим кроком якого є генерування симетричного ключа та його зберігання для подальшого шифрування масивів даних. Для реалізації цього алгоритму було обрано мову Python. Відповідно сама процедура генерування та зберігання матиме наступний вигляд:

```
def generate_key_and_save(key_length=16):
    """Генерує ключ для Blowfish і зберігає його у файл."""
    if not (4 <= key_length <= 56):
        raise ValueError("Довжина ключа повинна бути від 4 до 56 байт.")
        # Генеруємо ключ
    key = get_random_bytes(key_length)
        # Формуємо назву файлу з поточною датою і часом
    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
    key_filename = f"KEY_{timestamp}.key"
```

У результаті виконання даної процедури ми отримали файл із назвою "KEY_20241127_144159.key", який міститиме випадково згенерований ключ, який може мати заздалегідь визначену довжину. Довжина ключа безпосередньо визначає кількість можливих комбінацій, які потрібно перевірити для злому даних методом brute-force. Згідно результатів практичного дослідження, що наведені в таблиці 3.1, довжина ключа не впливає на швидкість шифрування та розшифровування даних.

Було проведено дослідження з файлом розміром 130 Мб, для якого було визначено швидкість шифрування та дешифрування в залежності від довжини ключа. Для визначення відхилення було обчислено середнє значення, яке становило 2,101659929 с для процесу шифрування та 1,842848143 с для дешифрування. Як видно з таблиці 3.1 відхилення від середнього значення що при процесі шифрування, що при дешифруванні не перевищує 5%, що можна вважати статистичною помилкою, а отже варто використовувати ключ максимальної довжини.

Таблиця 3.1

Швидкість виконання операцій симетричним алгоритмом Blowfish

Довжина ключа, біт	Шифрування файлу, с	Відхилення від середнього, %	Дешифрування файлу, с	Відхилення від середнього, %
4	2,062644	-4,14	1,84407	0,07
8	2,17645	1,15	1,879106	1,97
12	2,111131	-1,88	1,856001	0,71
16	2,039007	-5,24	1,894996	2,83
20	2,221521	3,25	1,894598	2,81
24	2,159994	0,39	1,887704	2,43
28	2,139989	-0,54	1,804991	-2,05
32	2,070998	-3,75	1,779963	-3,41
36	2,077	-3,47	1,892791	2,71
40	2,074906	-3,57	1,802587	-2,18
44	2,101826	-2,32	1,807203	-1,93
48	2,090188	-2,86	1,857404	0,79
52	2,058584	-4,33	1,816917	-1,41
56	2,039001	-5,24	1,781543	-3,33

Наступним кроком алгоритму шифрування є робота з масивом даних для шифрування, зазвичай така робота полягає у відкритті файлів та роботі з їх вмістом. Для цього потрібно запустити наступну процедуру:

```
def read_binary_file(file_path):
    try:
        with open(file_path, 'rb') as f:
            content = f.read()
        return content
```

Вміст відкритого файлу зчитується блоками розміром 64 байти, шифрується згідно алгоритму Blowfish та зберігається в подальшому в файл для пересилання:

```
def encrypt_file_content(input_file, output_file, key):
    try:
        with open(input_file, 'rb') as f:
```

```
        data = f.read()  # Зчитування вмісту файлу
    cipher = Blowfish.new(key, Blowfish.MODE_CBC)
    iv = cipher.iv  # Ініціалізаційний вектор
    padded_data = pad(data, Blowfish.block_size)
    ciphertext = cipher.encrypt(padded_data)
    with open(output_file, 'wb') as f:
        f.write(iv + ciphertext)
    print(f"Файл '{input_file}' успішно зашифровано та збережено у '{output_file}'.")
```

У результаті ми отримаємо готовий до надсилання файл вміст якого перетворений на шифротекст. Отримувачу для отримання доступу до вмісту початкового файлу необхідно мати семетричний ключ.

3.3. Реалізація асиметричного фрагменту алгоритму

Як зазначалося раніше, асиметричний алгоритм застосовується для шифрування і розшифрування симетричного ключа в процесі передач його до відправника для чого він повинен згенерувати відповідні ключі:

```
from cryptography.hazmat.primitives.asymmetric import ec
from cryptography.hazmat.primitives import serialization
def generate_ecc_key_pair():
    private_key = ec.generate_private_key(ec.SECP256R1())  # Використовується крива SECP256R1
    public_key = private_key.public_key()
    private_key_pem = private_key.private_bytes(
        encoding=serialization.Encoding.PEM,
        format=serialization.PrivateFormat.PKCS8,
        encryption_algorithm=serialization.NoEncryption()  # Без паролю
    )
    public_key_pem = public_key.public_bytes(
        encoding=serialization.Encoding.PEM,
        format=serialization.PublicFormat.SubjectPublicKeyInfo
    )
    with open("ecc_private_key.pem", "wb") as private_file:
        private_file.write(private_key_pem)
```

```
    with open("ecc_public_key.pem", "wb") as public_file:
        public_file.write(public_key_pem)
    print("ECC ключі успішно згенеровано та збережено:")
    print("- Приватний ключ: ecc_private_key.pem")
    print("- Публічний ключ: ecc_public_key.pem")
if __name__ == "__main__":
    generate_ecc_key_pair()
```

У результаті виконання даної процедури генерується набір ключів – відкритий та закритий. Відкритий передається у вільний доступ, а закритий зберігається для подальшого розшифрування даних.

Отримавши публічний ключ відправник застосовує його для шифрування симетричного ключа. Процедура шифрування симетричного ключа матиме наступний вигляд:

```
def encrypt_symmetric_key(sym_key_file, output_file, public_key_file):
    try:
        with open(sym_key_file, "rb") as f:
            sym_key = f.read()
        with open(public_key_file, "rb") as f:
            public_key = serialization.load_pem_public_key(f.read())
        ephemeral_private_key = ec.generate_private_key(ec.SECP256R1())
        shared_key = ephemeral_private_key.exchange(ec.ECDH(), public_key)
        encryption_key = HKDF(
            algorithm=hashes.SHA256(),
            length=32,
            salt=None,
            info=b"encryption key",
        ).
derive(shared_key)
        encrypted_sym_key = bytes(a ^ b for a, b in zip(sym_key, encryption_key[:len(sym_key)]))
        ephemeral_public_key = ephemeral_private_key.public_key()
        ephemeral_public_key_bytes = ephemeral_public_key.public_bytes(
            encoding=serialization.Encoding.PEM,
            format=serialization.PublicFormat.SubjectPublicKeyInfo,
        )
        with open(output_file, "wb") as f:
```

```
            f.write(ephemeral_public_key_bytes)  # Епізодичний публічний ключ
            f.write(encrypted_sym_key)  # Зашифрований симетричний ключ
        print(f"Симетричний ключ успішно зашифровано та збережено у '{output_file}'.")
    except Exception as e:
        print(f"Помилка: {e}")
```

Запис в код проекту даної процедури дозволяє в подальшому отримати зашифрований асиметричним алгоритмом ECC симетричний ключ Blowfish звернувшись до неї наступним чином:

```
if __name__ == "__main__":
    sym_key_file = "KEY_20241127_144159.key"
    output_file = "encrypted_symmetric_key.ecc"
    public_key_file = "ecc_public_key.pem"
    encrypt_symmetric_key(sym_key_file, output_file, public_key_file)
```

Схожим чином реалізована і процедура дешифрування:

```
def decrypt_symmetric_key(encrypted_file, private_key_file, output_key_file):
    try:
        with open(private_key_file, "rb") as f:
            private_key = serialization.load_pem_private_key(f.read(), password=None)
        with open(encrypted_file, "rb") as f:
            # Епізодичний публічний ключ
            ephemeral_public_key = serialization.load_pem_public_key(f.read())
            encrypted_sym_key = f.read()
        shared_key = private_key.exchange(ec.ECDH(), ephemeral_public_key)
        decryption_key = HKDF(
            algorithm=hashes.SHA256(),
            length=32,
            salt=None,
            info=b"encryption key",
        ).
derive(shared_key)
        decrypted_sym_key = bytes(a ^ b for a, b in zip(encrypted_sym_key, decryption_key[:len(encrypted_sym_key)]))
```

```
            with open(output_key_file, "wb") as f:
                f.write(decrypted_sym_key)
            print(f"Симетричний ключ успішно розшифровано та збережено у '{output_key_file}'.")
        except Exception as e:
            print(f"Помилка: {e}")
```

Виклик такої процедури дозволяє витягнути зашифрований симетричний ключ для подальшого розшифрування файлів з даними. Сам виклик матиме наступний вигляд:

```
if __name__ == "__main__":
    encrypted_file = "encrypted_symmetric_key.ecc
    private_key_file = "ecc_private_key.pem"
    output_key_file = "decrypted_symmetric_key.key"
    decrypt_symmetric_key(encrypted_file, private_key_file, output_key_file)
```

Відповідно вище наведені процедури є ключовими для роботи програмної реалізації гібридного алгоритму на основі симетричного алгоритму Blowfish та асиметричного ECC.

3.4. Практична реалізація гібридного алгоритму

Як зазначалося в пункті 3.2 для реалізації гібридного шифрування було обрано мову Python. За допомогою своїх бібліотек, таких як *tkinter*, Python дозволяє реалізовувати проект у вигляді графічного інтерфейсу, а спеціалізовані бібліотеки так як *cryptography*, дозволять використовувати готові рішення для реалізації більшості криптографічних примітивів таких як шифрування, дешифрування, генерування ключів та інші.

Практична реалізація повинна містити два модулі програми. Одня для відправника інша для одержувача. Відправник повинен мати можливість відкрити файл з публічним асиметричним ключом, згенерувати симетричний ключ для поточного сеансу надсилання даних, вибрати файл для шифрування

та переглянути прогрес шифрування, якщо файл дуже великий і необхідно почекати.

3.4.1. Практична реалізація модуля відправника

Код програмної реалізації модуля відправника представлено у додатку А, а вікно програми представлено на рисунку 3.4.

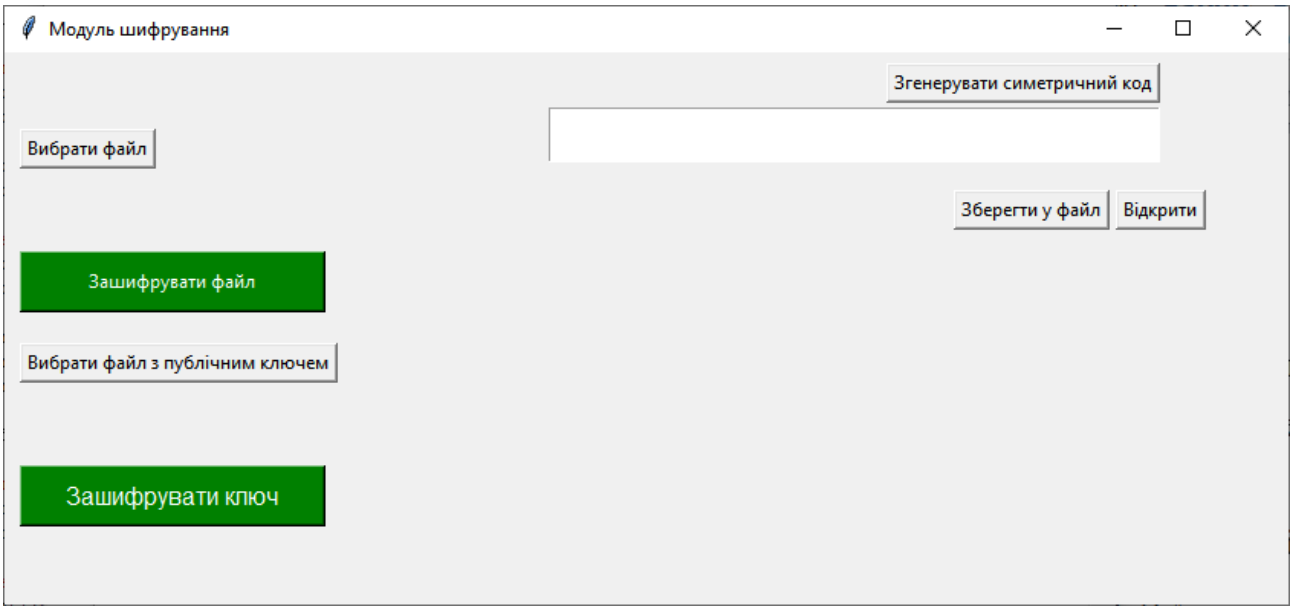

Рисунок 3.4 – Вікно модуля шифрування

Вікно програми можна умовно розділити на дві частини. В правій частині знаходиться блок, що відповідає за роботу з симетричним ключем на стороні відправника. Відправник має можливість згенерувати новий симетричний ключ, переглянути сгенерований ключ, зберегти його у файл та відкрити файл, який містить ключ що використовувався раніше (рис. 3.5).

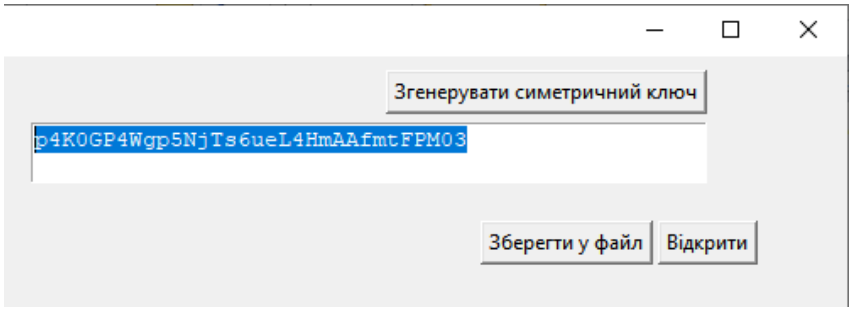

Рисунок 3.5 – Згенерований симетричний ключ

Як зазначалося раніше цей ключ використовується для шифрування файлу. Весь функціонал роботи програми з файлами є в лівій стороні вікна. Для роботи з файлом в першу чергу потрібно його вибрати (рис. 3.6).

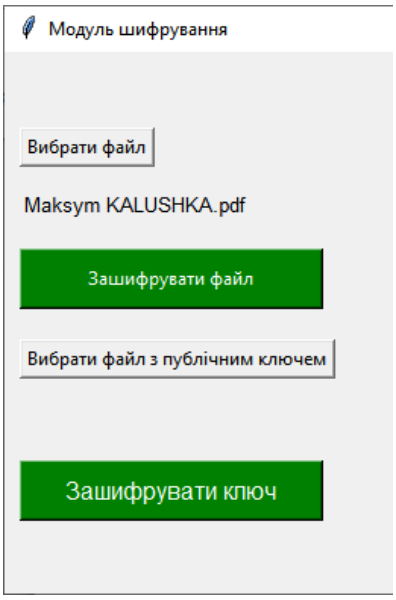

Рисунок 3.6 – Вибраний файл

Як видно на рисунку 3.6 назву обраного файлу видно у відповідному полі, що зменшує ймовірність шифрування не відповідного або помилкового файлу. Зашифрувавши файл симетричним алгоритмом його можна передати будь яким доступним каналом до одержувача.

Для забезпечення необхідного рівня захисту симетричний ключ необхідно передатавати або захищеним каналом, або застосовувати асиметричний публічній ключ для шифрування. У пропонованій реалізації необхідно вибрати файл з публічним ключем асиметричного алгоритму ECC для шифрування симетричного ключа.

3.4.2. Практична реалізація модуля отримувача

Код програмної реалізації модуля отримувача представлено у додатку Б, а вікно програми представлено на рисунку 3.7.

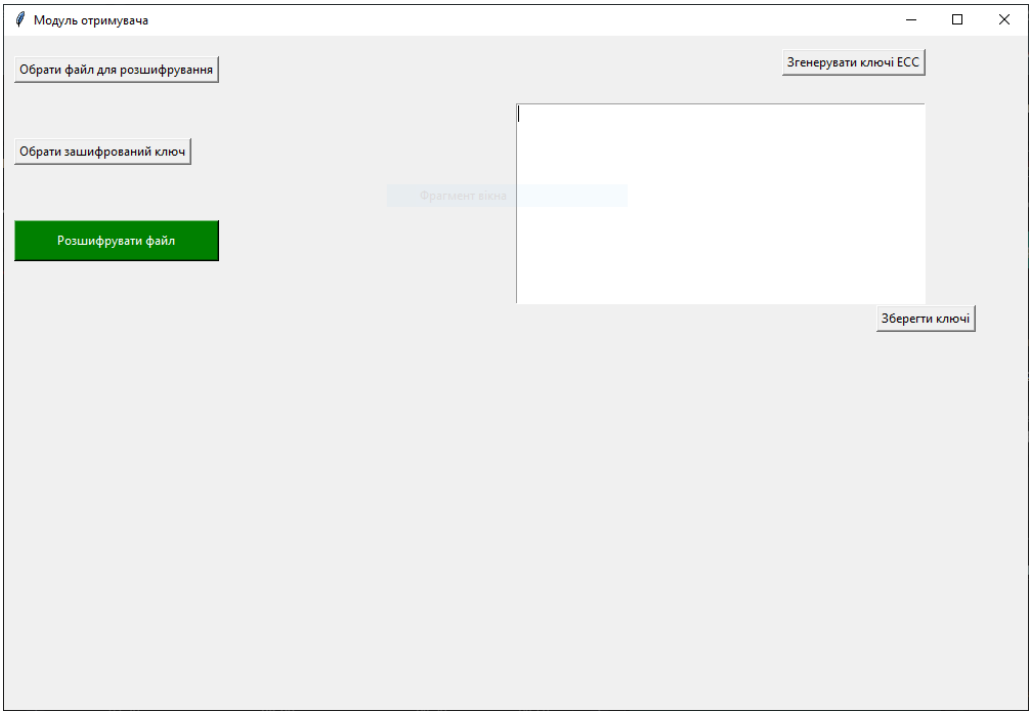

Рисунок 3.7 – Вікно модуля дешифрування

Згідно озвучених раніш завдань програмний модуль дешифрування знаходиться на стороні одержувача і його функціонал повинен давати змогу дешифрувати симетричний ключ та сам вміст файлу.

Не менш важливим для отримувача є можливість вибрати попередній приватний ключ ECC для відкриття надісланих раніше файлів (рис. 3.8).

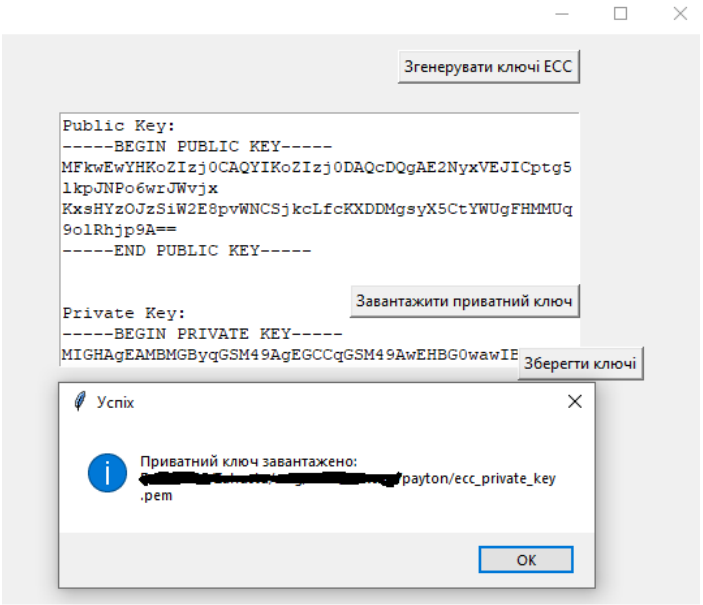

Рисунок 3.8 – Вибір попереднього ключа

Програма сама пакує ключі у файли спеціального розширення *.pem, які є спеціалізованими файлами призначеними для їх зберігання перед надсиланням. В подальшому згенерований або відкритий ключ використовується для роботи в правій частині вікна, де відбувається робота з шифротекстом та зашифрованим ключем (рис. 3.9).

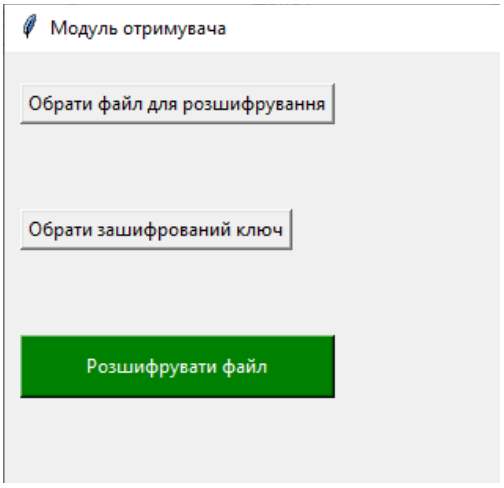

Рисунок 3.9 – Вибір файлів даних та ключа

Користувач за допомогою кнопок обирає надіслані йому через публічну мережу файли з симетричним ключем та зашифрованими даними. Інформація про це буде відображатися в спеціальному полі зразу під кнопкою (рис. 3.10).

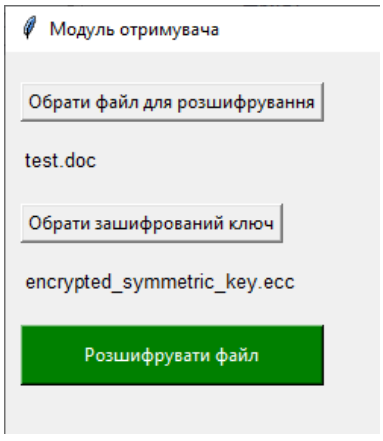

Рисунок 3.10 – Вибрані файли даних та ключа

Після натискання кнопки «Розшифрувати файл» результат розшифрування зберігається у новий файл із суфіксом *_decrypted*. Наприклад,

якщо вхідний файл називається *test.doc*, то розшифрований файл називатиметься *test_decrypted.doc* (рис. 3.11).

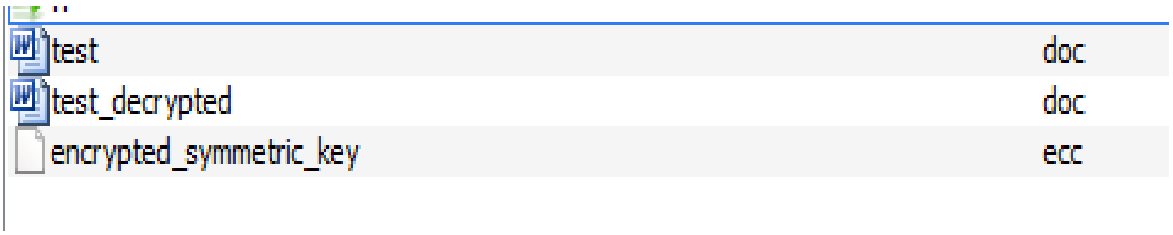

Рисунок 3.11 – Файли шифротексту, ключа та розшифрованих даних

В пропонованій структурі гібридного шифрування дозволяється застосування загальних каналів зв’язку між користувачами для пересилання публічного ключа ECC, шифротексту та зашифрованого симетричного ключа. Перехоплення файлів з таким вмістом не загрожує витоку інформації при отриманні їх зловмисником.

Єдиним вразливим місцем у такій системі буде тільки надійність місця для зберігання приватного ключа ECC, оскільки саме його витік загрожує втраті інформації.

ВИСНОВКИ

Проведено аналіз ключових методів шифрування даних, криптографії та їх класифікацію, розглянуто основні криптографічні алгоритми, а саме симетричні та асиметричні. Досліджено сферу застосування криптографічних алгоритмів та проблеми, що виникають при пересиланні даних в публічних мережах.

Проведено дослідження принципів роботи асиметричних та симетричних алгоритмів шифрування даних. Проведено аналіз принципів роботи гібридного шифрування та його складових. Запропоновано оптимальну симетричну та асиметричну складові, Обргунтовано використання блочного симетричного алгоритм Blowfish та асиметричного алгоритм ECC на основі еліптичних кривих.

Побудовано алгоритми шифрування пакетів даних та симетричного ключа перед передачею в публічній мережі, а також алгоритм дешифрування симетричного ключа на боці отримувача.

Приведено практичну реалізацію модулів шифрування та дешифрування пакетів даних на основі запропонованих алгоритмів гібридного алгоритму шифрування. Запропонована система дозволяє здійснювати генерацію симетричного ключа та зберігання його у файл; шифрувати файли даних та симетричний ключа на стороні відправника та генерувати асиметричний набір ключі та здійснювати обернені операції на стороні отримувача.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

4. Калушка М., Кулина С. (2024). Схема та алгоритм гібридного шифрування. *Збірник матеріалів науково-практичної конференції молодих вчених, аспірантів та студентів «Кібербезпека та комп'ютерно-інтегровані технології» (КБКІТ-2024)*, Тернопіль, с. 100-103.
5. Калушка М., Кулина С. (2024). Концепція та принципи гібридного шифрування. *Кібербезпека державних інституцій та подолання кризових станів* : *матеріали ІІІ Міжнар. наук.-практ. конф. в 2 т. (Київ – Прага – Таллінн – Тернопіль)*, 14.11.2024 р. ІСЗЗІ КПІ ім. Ігоря Сікорського. С. 450.
6. Portmann, C., & Renner, R. (2022). Security in quantum cryptography. *Reviews of Modern Physics*, *94* (2), 025008.
7. Al Busafi, S., & Kumar, B. (2020, December). Review and analysis of cryptography techniques. In *2020 9th International Conference System Modeling and Advancement in Research Trends (SMART)* pp. 323-327.
8. Deineka, O., & Harasymchuk, O. (2023). Дослідження проблем класифікації та безпечного зберігання даних. *Ukrainian Scientific Journal of Information Security*, *29*(3), 147-153.
9. Yulianto, B. D., Handoko, L. B., Rachmawanto, E. H., & Soeleman, M. A. (2022, September). Digital Certificate Authentication with Three-Level Cryptography (SHA-256, DSA, 3DES). In *2022 International Seminar on Application for Technology of Information and Communication (iSemantic)* pp. 343-350.
10. Дробиш, Р. С., & Люта, М. В. (2021). Аналіз існуючих методів захисту та шифрування інформації. In *Інноватика в освіті, науці та бізнесі: виклики та можливості*. *Київський національний університет технологій та дизайну*, 251.
11. Salami, Y., Khajevand, V., & Zeinali, E. (2023). Cryptographic algorithms: a review of the literature, weaknesses and open challenges. *J. Comput. Robot*, *16*(2), 46-56.
12. Лобода, Ю. Г., & Орлова, О. Ю. (2013). Використання криптографічних засобів захисту інформаційних ресурсів. *Наукові праці [Одеської національної академії харчових технологій]*, (44 (1)), 266-270.

13. Горбенко, І. Д., & Харламб, М. І. (2014). Порівняльний аналіз стандартизованих версій криптоалгоритму ECIES. *Прикладная радиоэлектроника*, 13№ 3, 333-336.
14. Stallings, W. (2016). Cryptography and Network Security: Principles and Practice (7th Edition).
15. Korobeinikova, T., & Kopach, A. (2024). Сучасні алгоритми шифрування: детальний аналіз та перспективи розвитку. *SWorldJournal*, (25-01), 89-95.
16. Летенко, Ю. О., Рябухо, О. М., & Турка, Т. В. (2015). Протоколи розподілу та узгодження ключа, *Збірник наукових праць фізико-математичного факультету ДДПУ*, 30-37.
17. Яремчук, Ю. Є., & Грицак, А. В. (2022). Удосконалення методу побудови криптостійких функцій гешування. *Тези доповідей*, 69.
18. Горбенко, І. Д., Бойко, А. В., & Герцог, А. М. (2009). Порівняння перспективних швидкодіючих функцій гешування. *Прикладная радиоэлектроника : науч.-техн. журн.* ХНУРЭ. Т. 8, № 3, 321–326.
19. Matov, O. Y., & Vasylenko, V. S. (2016). Захист інформаційних об'єктів від навмисних колізій контрольних ознак у завадостійких кодах. *Реєстрація, зберігання і обробка даних*, *18*(2), 31-39.
20. Al-Husainy, M. A. F., Al-Shargabi, B., & Aljawarneh, S. (2021). Lightweight cryptography system for IoT devices using DNA. *Computers and Electrical Engineering*, *95*, 107418.
21. F. Zhang, Z.-y. Liang, B.-l. Yang, X.-j. Zhao, S.-z. Guo, and K. Ren. (2018). Survey of design and security evaluation of authenticated encryption algorithms in the caesar competition. *Frontiers of Information Technology & Electronic Engineering*. Vol. 19, no. 12, pp. 1475–1499.
22. W. Stallings (2017). The principles and practice of cryptography and network security 7th edition. *Pearson Education*, vol. 20, no. 1, p. 7.
23. W. Tuchman. (1997). A Brief History of the Data Encryption Standard. *USA: ACM Press Addison-Wesley Publishing Co.* Ch. 16, pp. 275–280.
24. R. P. Adhie, Y. Hutama, A. S. Ahmar, M. Setiawan et al. (2018). Implementation cryptography data encryption standard (des) and triple data encryption standard

(3des) method in communication system based near field communication (nfc). *in Journal of Physics: Conference Series*, vol. 954, no. 1. 012009.

25. J. Katz and Y. Lindell, Introduction to Modern Cryptography, ser. Chapman & Hall. (2014). CRC Cryptography and Network Security Series. *CRC Press*.
26. N. Advani, C. Rathod, and A. M. Gonsai. (2019). Comparative study of various cryptographic algorithms used for text, image, and video. *In Emerging Trends in Expert Applications and Security. Springer*. pp. 393–399.
27. Premkumar, P. and Shanthi, D. (2016). Block Level Data Integrity Assurance Using Matrix Dialing Method towards High Performance Data Security on Cloud Storage. *Circuits and Systems*, **7**, 3626-3644. doi: 10.4236/cs.2016.711307.
28. Elaine Barker, Lily Chen, Allen Roginsky, Miles Smid. (2013) Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography. *National Institute of Standards and Technology*.
29. Завгородній, В. В., Завгородня, Г. А., Березінський, Ю. С., & Березінська, І. П. (2023). Реалізація криптостійкого алгоритму із простою процедурою шифрування та дешифрування на основі еліптичних кривих. *Таврійський науковий вісник. Серія: Технічні науки*, (3), 13-20.
30. O. Abood and S. Guirguis. (2018). A survey on cryptography algorithms. *International Journal of Scientific and Research Publications*, vol. 8, pp. 410–415.
31. C. Riman and P. E. Abi-Char. (2015). Comparative analysis of block cipher-based encryption algorithms: A survey. I*nformation Security and Computer Fraud*, vol. 3, no. 1, pp. 1–7.
32. P. Dixit, A. K. Gupta, M. C. Trivedi, and V. K. Yadav. (2018). Traditional and hybrid encryption techniques: a survey. *In Networking Communication and Data Knowledge Engineering. Springer*. pp. 239–248.
33. R. Bhanot and R. Hans. (2015). A review and comparative analysis of various encryption algorithms. *International Journal of Security and Its Applications*. Vol. 9, no. 4, pp. 289–306,.
34. M. N. A. Wahid, A. Ali, B. Esparham, and M. Marwan. (2018). A comparison of cryptographic algorithms: Des, 3des, aes, rsa and BlowFish for guessing attacks prevention. *J Comp Sci Appl Inform Technol*. vol. 3, no. 2, pp. 1–7.

35. G. Kaur and M. Mahajan. (2013). Evaluation and comparison of symmetric key algorithms. *International Journal of Science, Engineering and Technology Research (IJSETR)*. Vol. 2, no. 10, pp. 1960–1962.
36. A. Y. Hendi, M. O. Dwairi, Z. A. Al-Qadi, and M. S. Soliman. (2019). A novel simple and highly secure method for data encryption-decryption. *International Journal of Communication Networks and Information Security*. Vol. 11, pp. 232-238.
37. N. Tyagi and A. Ganpati. (2014). Comparative analysis of symmetric key encryption algorithms. I*nternational Journal of Advanced Research in Computer Science and Software Engineering*. Vol. 4, no. 8, pp. 63–70.
38. P. Princy. (2015). A comparison of symmetric key algorithms des, aes, BlowFish, rc4, rc6: A survey. *International Journal of Computer Science & Engineering Technology (IJCSET)*. Vol. 6, no. 5, pp. 328–331.
39. M. Mathur and A. Kesarwani. (2013). Comparison between des, 3des, rc2, rc6, BlowFish and aes. *In Proceedings of National Conference on New Horizons in ITNCNHIT*. Vol. 3, , pp. 143–148.
40. P. Nema and M.A.Rizvi. (2015). Critical analysis of various symmetric key cryptographic algorithms. *International Journal on Recent and Innovation Trends in Computing and Communication*. Vol. 3, no. 6, pp. 4301–4306.
41. M. Marwaha, R. K. Bedi, A. Singh, and T. Singh. (2013). Comparative analysis of cryptographic algorithms. *International Journal of Advanced Engineering Technology*. pp. 16–18.
42. A. Nadeem and M. Y. Javed. (2005). A performance comparison of data encryption algorithms. *International Conference on Information and Communication Technologies*. pp. 84–89.
43. P. J. Sun. (2019). Privacy protection and data security in cloud computing: a survey, challenges, and solutions. *IEEE Access*. Vol. 7, pp. 147 420-147 452.
44. Квєтний, Р. Н., & Титарчук, Є. О. (2017). Аналіз криптостійкості частково гомоморфного алгоритму шифрування на основі еліптичних кривих. *Інформаційні технології та комп'ютерна інженерія. № 1, 83-86.*
45. Мінгальова, Ю. І. (2013). Огляд методів симетричного й асиметричного шифрування даних. *Магістратура в умовах євроінтеграційних процесів*

вищої школи: збірник наукових праць. 366-368.

46. Воробйов, В. Г. (2017). Теоретичні основи побудови гібридних криптографічних систем захисту інформації. *Vědecké pokrok na přelomu tysyachaletyích Věd-2017*, 70.
47. Chang, X., Li, W., Yan, A., Tsang, P. W. M., & Poon, T. C. (2022). Asymmetric cryptosystem based on optical scanning cryptography and elliptic curve algorithm. *Scientific Reports*, *12*(1), 7722.
48. Tütüncü, K., & Çataltaş, Ö. (2021). Compensation of degradation, security, and capacity of LSB substitution methods by a new proposed hybrid n-LSB approach. *Computer Science and Information Systems*, *18* (4), 1311-1332.
49. Alarifi, A., Sankar, S., Altameem, T., Jithin, K. C., Amoon, M., & El-Shafai, W. (2020). A novel hybrid cryptosystem for secure streaming of high efficiency H. 265 compressed videos in IoT multimedia applications. *IEEE Access*, *8*, 128548.
50. Горбенко, І. Д., Качко, О. Г., Єсіна, М. В., & Пономар, В. А. (2022). Порівняльна характеристика алгоритмів інкапсуляції ключів Crystals-Kyber та Скеля (ДСТУ 8961 2019). *Радіотехніка : Всеукр. міжвід. наук.-техн. зб*. – Вип. 210. с. 7–21. DOI: 10.30837/rt.2022.3.210.01.
51. Haq, T. U., Shah, T., Siddiqui, G. F., Iqbal, M. Z., Hameed, I. A., & Jamil, H. (2021). Improved twofish algorithm: a digital image enciphering application. *IEEE Access*, *9*, 76518-76530.
52. Ullah, S., Zheng, J., Din, N., Hussain, M. T., Ullah, F., & Yousaf, M. (2023). Elliptic Curve Cryptography; Applications, challenges, recent advances, and future trends: A comprehensive survey. *Computer Science Review*, *47*, 100530.
53. Al-Absi, M. A., Abdullaev, A., Al-Absi, A. A., Sain, M., & Lee, H. J. (2020). Cryptography Survey of DSS and DSA. In *Advances in Materials and Manufacturing Engineering: Proceedings of ICAMME 2019*. pp. 661-669.
54. L. Balamurugan, C., Singh, K., Ganesan, G., & Rajarajan, M. (2021). Post-quantum and code-based cryptography - some prospective research directions. *Cryptography*, *5*(4), 38.
55. Новиков, Д. Полторак В. (2023). Технології постквантової криптографії. *Адаптивні системи автоматичного управління: міжвідомчий науково-*

технічний збірник. № 1 (42). С. 171-183.

56. Hashimoto Y., Takagi T., Sakurai K. General. (2011). Fault Attacks on Multivariate Public Key Cryptosystems. Post-Quantum Cryptography. *Taiwan. Taipei: Springer, Berlin, Heidelberg*. с.1-18.
57. Новиков, Д. Полторак В. (2023). Технології постквантової криптографії. *Адаптивні системи автоматичного управління: міжвідомчий науково-технічний збірник*. № 1 (42). с. 171-183.
58. Post-Quantum Cryptography Project – Security (Evaluation Criteria) [Електронний ресурс] – Режим доступу до ресурсу: https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum cryptography-standardization/evaluation-criteria/security-(evaluation-criteria)
59. K. Boudgoust та ін. (2020). Towards Classical Hardness of Module-LWE: The Linear Rank Case. *Advances in Cryptology – ASIACRYPT 2020*. South Korea. Daejeon: Springer, Cham,. с.289-317.
60. Lyubashevsky, V., Peikert, C., Regev, O. (2013). A Toolkit for Ring-LWE Cryptography. *In: Johansson, T., Nguyen, P.Q. (eds) Advances in Cryptology – EUROCRYPT 2013*. *Lecture Notes in Computer Science*, vol 7881.
61. Akleylek S. K. Seyhan. (2022). Module learning with rounding based key agreement scheme with modified reconciliation. *Computer Standards & Interfaces*. Vol 79.
62. Shor, P. (1997). Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM Journal on Computing*. №26.с. 1484–1509
63. Mellila Bouam, Charles Bouillaguet, Claire Delaplace, Camille Noûs. (2021). Brute-Force Cryptanalysis with Aging Hardware: Controlling Half the Output of SHA-256. Режим доступу до ресурсу: https://hal.science/hal-02306904v2
64. Gorbenko, Y., Yesina, M., Ponomar, V., Gorbenko, I., & Kapt'ol, E. (2023). Scientific and methodological bases of analysis, evaluation and results of comparison of existing and promising (post-quantum) asymmetric cryptographic primitives of electronic signature, protocols of asymmetric encryption and key encapsulation protocols. *Radiotekhnika*, *1*(212), 42–65.

ДОДАТОК А

Код програмної реалізації модуля відправника

```
import tkinter as tk
from tkinter import filedialog
from tkinter import messagebox
import random
import string
import os
def generate_symmetric_key():
    key = ''.join(random.choices(string.ascii_letters + string.digits, k=32))
    key_field.delete("1.0", tk.END)
    key_field.insert(tk.END, key)
def save_symmetric_key():
    key = key_field.get("1.0", tk.END).strip()
    if not key:
        messagebox.showwarning("Попередження", "Немає ключа для збереження.")
        return
    save_path = filedialog.asksaveasfilename(
        title="Зберегти симетричний ключ",
        defaultextension=".txt",
        filetypes=[("Text Files", "*.txt"), ("All Files", "*.*")]
    )    if save_path:
        with open(save_path, "w", encoding="utf-8") as file:
            file.write(key)
        messagebox.showinfo("Успіх", f"Ключ збережено у файл:\n{save_path}")
def open_symmetric_key():
    file_path = filedialog.askopenfilename(
        title="Вибрати файл із симетричним ключем",
        filetypes=[("Text Files", "*.txt"), ("All Files", "*.*")]
    )    if file_path:
        with open(file_path, "r", encoding="utf-8") as file:
            key = file.read().strip()
        key_field.delete("1.0", tk.END)
        key_field.insert(tk.END, key)
def browse_file():
    file_path = filedialog.askopenfilename(title="Вибрати файл для шифрування")
```

```
    if file_path:
        file_name_var.set(os.path.basename(file_path))
        data_file_path_var.set(file_path)
def browse_public_key():
    file_path = filedialog.askopenfilename(
        title="Вибрати файл з публічним ключем ECC",
        filetypes=[("Text Files", "*.txt"), ("All Files", "*.*")]
    )
    if file_path:
        ecc_key_name_var.set(os.path.basename(file_path))
        ecc_key_path_var.set(file_path)
def encrypt_with_symmetric_key():
    data_file = data_file_path_var.get()
    symmetric_key = key_field.get("1.0", tk.END).strip()
    if not data_file or not symmetric_key:
        messagebox.showwarning("Помилка", "Необхідно вибрати файл і згенерувати
симетричний ключ.")
        return
    encrypted_file = f"{os.path.splitext(data_file)[0]}_symmetric_encrypted.txt"
    try:
        with open(data_file, "rb") as file:
            data = file.read()
        encrypted_data = f"SYMMETRIC_ENCRYPTED({data.hex()}) WITH KEY({symmetric_key})"
        with open(encrypted_file, "w", encoding="utf-8") as file:
            file.write(encrypted_data)
        messagebox.showinfo("Успіх", f"Файл зашифровано симетричним
ключем:\n{encrypted_file}")
    except Exception as e:
        messagebox.showerror("Помилка", f"Не вдалося зашифрувати файл:\n{str(e)}")
def encrypt_symmetric_key_with_ecc():
    ecc_key_file = ecc_key_path_var.get()
    symmetric_key = key_field.get("1.0", tk.END).strip()
    if not ecc_key_file or not symmetric_key:
        messagebox.showwarning("Помилка", "Необхідно вибрати публічний ключ ECC і
згенерувати симетричний ключ.")
        return
    encrypted_key_file = f"SymmetricKey_encrypted_with_ECC.txt"
    encrypted_key = f"ECC_ENCRYPTED({symmetric_key})"
```

```
    with open(encrypted_key_file, "w", encoding="utf-8") as file:
        file.write(encrypted_key)
    messagebox.showinfo("Успіх", f"Симетричний ключ зашифровано за допомогою
ECC:\n{encrypted_key_file}")
root = tk.Tk()
root.title("Модуль шифрування")
root.geometry("1000x600")
generate_button = tk.Button(root, text="Згенерувати симетричний код",
command=generate_symmetric_key)
generate_button.place(relx=0.9, rely=0.02, anchor="ne")
key_field = tk.Text(root, height=2, wrap=tk.WORD)
key_field.place(relx=0.9, rely=0.1, width=400, anchor="ne")
save_button = tk.Button(root, text="Зберегти у файл", command=save_symmetric_key)
save_button.place(relx=0.8, rely=0.25, anchor="n")
open_button = tk.Button(root, text="Відкрити", command=open_symmetric_key)
open_button.place(relx=0.9, rely=0.25, anchor="n")
file_button = tk.Button(root, text="Вибрати файл", command=browse_file)
file_button.place(x=10, y=50)
file_name_var = tk.StringVar()
file_name_label = tk.Label(root, textvariable=file_name_var, font=("Arial", 10), anchor="w")
file_name_label.place(x=10, y=90, width=400)
data_file_path_var = tk.StringVar()
encrypt_file_button = tk.Button(root, text="Зашифрувати файл",
command=encrypt_with_symmetric_key, bg="green", fg="white")
encrypt_file_button.place(x=10, y=130, width=200, height=40)
ecc_button = tk.Button(root, text="Вибрати файл з публічним ключем",
command=browse_public_key)
ecc_button.place(x=10, y=190)
ecc_key_name_var = tk.StringVar()
ecc_key_label = tk.Label(root, textvariable=ecc_key_name_var, font=("Arial", 10), anchor="w")
ecc_key_label.place(x=10, y=230, width=400)
ecc_key_path_var = tk.StringVar()
encrypt_ecc_button = tk.Button(root, text="Зашифрувати ключ", font=("Arial", 12), bg="green",
fg="white", command=encrypt_symmetric_key_with_ecc)
encrypt_ecc_button.place(x=10, y=270, width=200, height=40)
root.mainloop()
```

ДОДАТОК Б

Код програмної реалізації модуля отримувача

```
import tkinter as tk
from tkinter import filedialog, messagebox
from cryptography.hazmat.primitives.asymmetric import ec
from cryptography.hazmat.primitives import serialization
from datetime import datetime
import os
def generate_ecc_keys():
    global private_key, public_key
    private_key = ec.generate_private_key(ec.SECP256R1())
    public_key = private_key.public_key()
    private_key_pem = private_key.private_bytes(
        encoding=serialization.Encoding.PEM,
        format=serialization.PrivateFormat.PKCS8,
        encryption_algorithm=serialization.NoEncryption(),
    )
    public_key_pem = public_key.public_bytes(
        encoding=serialization.Encoding.PEM,
        format=serialization.PublicFormat.SubjectPublicKeyInfo,
    )
    key_field.delete("1.0", tk.END)
    key_field.insert(tk.END, f"Public Key:\n{public_key_pem.decode()}\n\nPrivate
Key:\n{private_key_pem.decode()}")
def save_ecc_keys():
    """Зберігає ключі ECC у файл з автоматичною назвою."""
    if private_key is None or public_key is None:
        messagebox.showwarning("Попередження", "Ключі ще не згенеровані.")
        return
    try:
        private_key_pem = private_key.private_bytes(
            encoding=serialization.Encoding.PEM,
            format=serialization.PrivateFormat.PKCS8,
            encryption_algorithm=serialization.NoEncryption(),
        )
        public_key_pem = public_key.public_bytes(
```

```
            encoding=serialization.Encoding.PEM,
            format=serialization.PublicFormat.SubjectPublicKeyInfo,
        )
        timestamp = datetime.now().strftime("%Y-%m-%d_%H-%M-%S")
        desktop_path = os.path.join(os.path.join(os.environ["USERPROFILE"]), "Desktop")
        private_file_name = f"Private_Key_{timestamp}.pem"
        public_file_name = f"Public_Key_{timestamp}.pem"
        private_file_path = os.path.join(desktop_path, private_file_name)
        public_file_path = os.path.join(desktop_path, public_file_name)
        with open(private_file_path, "wb") as priv_file:
            priv_file.write(private_key_pem)
        with open(public_file_path, "wb") as pub_file:
            pub_file.write(public_key_pem)
        messagebox.showinfo(
            "Успіх",
            f"Ключі ECC збережено:\n{private_file_path}\n{public_file_path}",
        )
    except Exception as e:
        messagebox.showerror("Помилка", f"Не вдалося зберегти ключі:\n{str(e)}")
def load_ecc_private_key():
    global private_key, public_key
    file_path = filedialog.askopenfilename(
        title="Виберіть файл з приватним ключем", filetypes=[("PEM файли", "*.pem")]
    )
    if not file_path:
        return
    try:
        with open(file_path, "rb") as file:
            private_key = serialization.load_pem_private_key(file.read(), password=None)
        public_key = private_key.public_key()
        private_key_pem = private_key.private_bytes(
            encoding=serialization.Encoding.PEM,
            format=serialization.PrivateFormat.PKCS8,
            encryption_algorithm=serialization.NoEncryption(),
        )
        public_key_pem = public_key.public_bytes(
            encoding=serialization.Encoding.PEM,
            format=serialization.PublicFormat.SubjectPublicKeyInfo,
```

```
        )
        key_field.delete("1.0", tk.END)
        key_field.insert(tk.END, f"Public Key:\n{public_key_pem.decode()}\n\nPrivate
Key:\n{private_key_pem.decode()}")
        messagebox.showinfo("Успіх", f"Приватний ключ завантажено:\n{file_path}")
    except Exception as e:
        messagebox.showerror("Помилка", f"Не вдалося завантажити ключ:\n{str(e)}")
def browse_encrypted_file():
    """Вибір файлу для розшифрування."""
    file_path = filedialog.askopenfilename(title="Вибрати файл для розшифрування")
    if file_path:
        encrypted_file_var.set(os.path.basename(file_path))
        encrypted_file_path_var.set(file_path)
def browse_encrypted_key():
    file_path = filedialog.askopenfilename(title="Вибрати зашифрований ключ")
    if file_path:
        encrypted_key_var.set(os.path.basename(file_path))
        encrypted_key_path_var.set(file_path)
def decrypt_file():
    encrypted_file = encrypted_file_path_var.get()
    encrypted_key_file = encrypted_key_path_var.get()
    if not encrypted_file or not encrypted_key_file:
        messagebox.showwarning("Помилка", "Необхідно вибрати файл і зашифрований ключ.")
        return
    decrypted_file = f"{os.path.splitext(encrypted_file)[0]}_decrypted.txt"
    try:
        with open(encrypted_file, "r", encoding="utf-8") as file:
            encrypted_data = file.read()
        with open(encrypted_key_file, "r", encoding="utf-8") as file:
            encrypted_key = file.read()
        decrypted_data = f"DECRYPTED({encrypted_data}) USING SYMMETRIC KEY FROM
({encrypted_key})"
        with open(decrypted_file, "w", encoding="utf-8") as file:
          file.write(decrypted_data)
        messagebox.showinfo("Успіх", f"Файл розшифровано:\n{decrypted_file}")
    except Exception as e:
        messagebox.showerror("Помилка", f"Не вдалося розшифрувати файл:\n{str(e)}")
private_key = None
```

```
public_key = None
root = tk.Tk()
root.title("Модуль отримувача")
root.geometry("1000x600")
browse_file_button = tk.Button(root, text="Обрати файл для розшифрування",
command=browse_encrypted_file)
browse_file_button.place(x=10, y=20)
encrypted_file_var = tk.StringVar()
encrypted_file_label = tk.Label(root, textvariable=encrypted_file_var, font=("Arial", 10),
anchor="w")
encrypted_file_label.place(x=10, y=60, width=400)
encrypted_file_path_var = tk.StringVar()
browse_key_button = tk.Button(root, text="Обрати зашифрований ключ",
command=browse_encrypted_key)
browse_key_button.place(x=10, y=100)
encrypted_key_var = tk.StringVar()
encrypted_key_label = tk.Label(root, textvariable=encrypted_key_var, font=("Arial", 10),
anchor="w")
encrypted_key_label.place(x=10, y=140, width=400)
encrypted_key_path_var = tk.StringVar()
decrypt_button = tk.Button(root, text="Розшифрувати файл", command=decrypt_file,
bg="green", fg="white")
decrypt_button.place(x=10, y=180, width=200, height=40)
key_field = tk.Text(root, height=12, wrap=tk.WORD)
key_field.place(relx=0.9, rely=0.1, width=400, anchor="ne")
generate_button = tk.Button(root, text="Згенерувати ключі ECC", command=generate_ecc_keys)
generate_button.place(relx=0.9, rely=0.02, anchor="ne")
load_key_button = tk.Button(root, text="Завантажити приватний ключ",
command=load_ecc_private_key)
load_key_button.place(relx=0.9, rely=0.32, anchor="ne")
save_button = tk.Button(root, text="Зберегти ключі", command=save_ecc_keys)
save_button.place(relx=0.9, rely=0.4, anchor="n")
root.mainloop()
```

ДОДАТОК В

КОПІЇ ПУБЛІКАЦІЙ

КІБЕРБЕЗПЕКА ТА КОМП'ЮТЕРНО-ІНТЕГРОВАНІ ТЕХНОЛОГІЇ

2024

науково-практична конференція молодих вчених, аспірантів та студентів



ЗАХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ПОЛІТЕХНІКА»
ГАЛИЦЬКИЙ ФАХОВИЙ КОЛЕДЖ ІМ. В'ЯЧЕСЛАВА ЧОРНОВОЛА

КІБЕРБЕЗПЕКА ТА КОМП'ЮТЕРНО-ІНТЕГРОВАНІ ТЕХНОЛОГІЇ (КБКІТ – 2024)

науково-практична конференція
молодих вчених, аспірантів та студентів

26–28 серпня 2024
Тернопіль

Збірник матеріалів науково-практичної конференції молодих вчених, аспірантів та студентів «Кібербезпека та комп'ютерно-інтегровані технології» (КБКІТ - 2024), Тернопіль, 2024. - 160 с.

Адреса редакції:
Західноукраїнський національний університет, кафедра кібербезпеки,
вул. Олени Теліги 8, м. Тернопіль 46003
Контакти:
*e-mail:*conferencekb@gmail.com

БЕЗПЕКА ІНТЕРНЕТ РЕЧЕЙ

КРИПТОГРАФІЧНІ МЕТОДИ ЗАХИСТУ ІНФОРМАЦІЇ

УДК 343.9

Максим КАЛУШКА, Сергій КУЛИНА

Західноукраїнський національний університет

СХЕМА ТА АЛГОРИТМ ГІБРИДНОГО ШИФРУВАННЯ

Вступ. Зі збільшенням використання обміну даними та комунікації через Інтернет стає критично важливим захистити дані від потрапляння до рук зловмисників. Забезпечення цілісності та конфіденційності даних стає одним із ключових викликів для дослідників та фахівців у сфері кібербезпеки [1].

Мета: розробка схеми та алгоритму гібридного шифрування та подальше його впровадження в системах обробки даних.

1. Роль криптографії у кібербезпеці

Забезпечення захисту даних стає одним із ключових викликів для дослідників та фахівців у сфері кібербезпеки. Існує багато методів забезпечення ключових засад та принципів кібербезпеки, найпоширенішим з яких є застосування методів криптографії. Їх використання дає змогу забезпечити необхідний рівень конфіденційності, цілісності та невідмовності даних [2]. Що в свою чергу досягається захистом критичних даних або документів на жорсткому диску або під час їх передачі через ненадійний канал зв'язку. Дані у цьому процесі можуть бути випадково або навмисно змінені, спотворені або пошкоджені. Використання криптографії дозволяє частково уникнути таких загроз або зменшити їх вплив, а сам процес захисту інформації полягає в шифруванні та дешифруванні даних. Шифрування та дешифрування здійснюється за допомогою ключів, а сам процес поділяють на шифрування з симетричним ключем та шифрування з асиметричним ключем.

2. Симетричний алгоритм шифрування

У симетричному методі і шифрування і дешифрування здійснюються на основі єдиного ключа, який ще називається приватним або секретним ключем. Алгоритми із симетричним ключем залежно від вхідних даних поділяються на два типи: блочні та потокові шифри. У системах на основі блочного шифру дані обробляються або шифруються на фіксованій групі бітів, що називається блоком, тоді як у системах на основі потокового шифру дані обробляються у вигляді потоку бітів. Найбільш поширеними алгоритмами симетричного шифрування є AES і DES з блочних та RC4 і Blowfish з потокових шифрів [3].

Симетричне шифрування даних здійснюється одним і тим самим ключем, тому виникають додаткові недоліки його застосування:

- Проблема обміну ключами.
- Масштабованість.
- Ризик компрометації ключа.
- Відсутність підтвердження автентичності.
- Складнощі з управлінням ключами.

Представлена на рисунку 1 схема відображає процес обміну даними між двома користувачами із застосуванням симетричного шифрування даних.

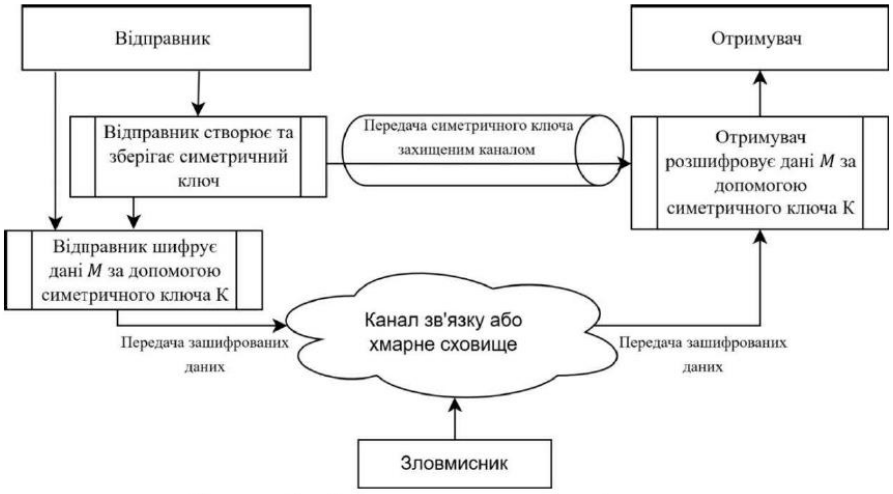

Рисунок 1 - Схема симетричного шифрування

2. Асиметричний алгоритм шифрування

Асиметричне шифрування використовує пару математично пов'язаних ключів, які називають відкритий ключ (public key) та закритий ключ (private key). Ця пара ключів дозволяє забезпечити високий рівень захисту при передачі даних та автентифікації, оскільки операції шифрування та розшифрування виконуються різними ключами. Відкритий ключ призначений для шифрування даних і доступний будь-кому, хто хоче надіслати зашифроване повідомлення отримувачу. Натомість закритий ключ використовується для розшифрування повідомлення і зберігається в таємниці, його має тільки отримувач (рисунок 2).

Рисунок 2 - Схема асиметричного шифрування

Основною перевагою асиметричного шифрування є безпечний обмін ключами. Завдяки своїм властивостям асиметричне шифрування дає можливість передавати публічний ключ відкритими каналами, не ризикуючи компрометацією приватного ключа [3].

Відповідно абонент, що отримав відкритий ключ іншої сторони може

зашифрувати будь які дані та переслати їх отримувачу. Використовуючи дві пари асиметричних ключів користувачі можуть без проблем обмінюватись секретною інформацією без загрози її витоку.

Представлена на рисунку 2.1 схема відображає процес обміну даними між двома користувачами із застосуванням виключно асиметричного шифрування.

Проте асиметричне шифрування має і свої недоліки, а саме:

- Низька швидкість.
- Вимоги до обчислювальних ресурсів.
- Уразливість до квантових комп'ютерів.
- Складність управління ключами.
- Ризик компрометації відкритого ключа.
- Відносна складність у розумінні та впровадженні.

Тому ключовим при використанні асиметричного шифрування є пошук шляхів компенсації недоліків даної системи шифрування [4].

3. Гібридний алгоритм шифрування

Останнім часом широко застосовується гібридне шифрування. Воно використовує переваги як симетричного, так і асиметричного шифрування, що забезпечує високу безпеку та ефективність передачі даних. Ключовою властивістю гібридного шифрування є поєднання швидкості та зручності асиметричного методу з ефективністю симетричного.

У пропонованій схемі гібридного шифрування використовуються сильні сторони обох методів, а саме: асиметричне шифрування для безпечного обміну ключами та симетричне шифрування для швидкого та ефективного шифрування великих обсягів даних (рисунок 3).

Рисунок 3 - Загальна схема гібридного шифрування

Відповідно алгоритм для гібридного шифрування складатиметься з наступних кроків:

Крок 1. Генерація симетричного ключа. Відправник створює випадковий симетричний ключ K, який буде використаний для шифрування даних.

Крок 2. Шифрування даних симетричним ключем. На поточному кроці відправник шифрує дані M за допомогою симетричного ключа K, отримуючи у результаті зашифровані дані C.

Крок 3. Шифрування симетричного ключа асиметричним алгоритмом. Щоб захистити симетричний ключ K, відправник шифрує його за допомогою публічного асиметричного ключа отримувача, отримуючи зашифрований ключ E_K.

Крок 4. Надсилання зашифрованих даних. На поточному кроці відбувається надсилання зашифрованих даних C та зашифрованого симетричного ключа E_K.

Крок 5. Розшифрування симетричного ключа. Отримувач, маючи свій приватний ключ, який є парним до публічного ключа, використаного для шифрування симетричного ключа на кроці 3, розшифровує зашифрований симетричний ключ E_K, отримуючи у результаті оригінальний симетричний ключ K.

Крок 6. Розшифровування даних. Отримувач, маючи оригінальний симетричний ключ K, використовує його для розшифровування зашифрованих даних C, отримуючи в результаті оригінальні дані M.

Як бачимо запропонований алгоритм поєднує у собі переваги симетричного та асиметричного шифрування, а саме:

- використовує симетричний ключ для шифрування даних, що збільшує як швидкість шифрування, так і швидкість дешифрування даних;
- використовує загальний канал для пересилання шифрованих даних та не потребує окремого захищеного каналу для пересилання ключа кодування;
- шифрує симетричний ключ за допомогою публічного ключа одержувача, що забезпечує достатній рівень захисту та дозволяє пересилати його відкритим каналом зв'язку.

Висновок. Отже, використання методів гібридного шифрування значно підвищує захищеність та стійкість даних при передачі між абонентами або при зберіганні на віддалених хмарних сховищах. Це досягається завдяки поєднанню переваг симетричних та асиметричних алгоритмів шифрування, а вибір самих алгоритмів залежить від завдання, яке ставить перед собою власних системи.

Перелік використаних джерел.

1. Бондаренко, Т., Шкітов, А., Шаповал, В., Шевага, В., Нещерет, І., Цикало, Ю., Лазута, Р. (2024). Мобільні особливості кібербезпеки щодо штучного інтелекту у повоєнній україні: виклики та стратегії.-Наука і техніка сьогодні, (4 (32)).

1. 2 Лубко, Д. В., Мірошниченко, М. Ю. (2024). Механізми безпеки інформації та їх виклики. Науковий вісник Таврійського державного агротехнологічного університету, 14(2).

2. Савченко, Я., Чмиренко, О. (2023). Криптографічні та стеганографічні засоби захисту інформації. Актуальні питання забезпечення кібербезпеки та захисту інформації, 94.

3. Філіп'єва, М. В., Гвоздецька, К. П. (2021). Порівняння симетричного і асиметричного шифрування. Міжнародна наукова інтернет-конференція Інформаційне суспільство: технологічні, економічні та технічні аспекти становлення, 71.

CERTIFICATE

OF CONFERENCE PARTICIPANT

Maksym KALUSHKA

took part in the III International Scientific and Practical Conference
"CYBERSECURITY OF STATE INSTITUTIONS AND CRISIS MANAGEMENT"
(0.2 ECTS credits)

14 November 2024

Head of Institute of Special Communications and Information Protection of the National Technical University of Ukraine "Igor Sikorsky Kyiv Polytechnic Institute"

Brigadier General, PhD, Professor **Oleksandr PUCHKOV**

Максим КАЛУШКА;
Сергій КУЛИНА, доктор PHD з кібербезпеки, ст. викладач

КОНЦЕПЦІЯ ТА ПРИНЦИПИ ГІБРИДНОГО ШИФРУВАННЯ

Анотація. У даній роботі описуються принципи реалізації гібридного шифрування. Гібридне шифрування вирішує проблему поширення ключів та забезпечує високу швидкість шифрування даних.

Summary. This work describes the principles of hybrid encryption implementation. Hybrid encryption addresses the key distribution problem and ensures high data encryption speed.

Ключові слова: симетричне шифрування, асиметричне шифрування, гібридне шифрування, відкритий і закритий ключі.

Зі збільшенням використання обміну даними та комунікації через Інтернет стає критично важливим захистити дані від потрапляння до рук зловмисників. Це досягається за допомогою різноманітних типів шифрування, проте кожен із них має свої недоліки: симетричне - складність з передачею ключа, асиметричне – низьку швидкість та високу затрату ресурсів.

Гібридне шифрування використовує переваги як симетричного, так і асиметричного шифрування, що забезпечує високу безпеку та ефективність передачі даних. Використовуючи в якості одного із компонентів симетричне шифрування ми досягаємо необхідної швидкості при шифруванні великих об’ємів даних. У свою чергу шифрування симетричного ключа за допомогою відкритого ключа асиметричного алгоритму надає необхідний рівень захисту при передачі по незахищених каналах передачі даних або при зберіганні на публічних хмарних сховищах.

Наразі методи гібридного шифрування є одними з найефективніших способів захисту даних, проте із розвитком квантових досліджень існуючі рішення у цій сфері потребуватимуть додаткових досліджень та удосконалень.

Висновки. У роботі запропоновано принцип та алгоритм виконання гібридного шифрування даних. Описано послідовність кроків необхідних для шифрування симетричного ключа та передачі його отримувачу.