СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Навчально-науковий інститут (факультет) інформаційних технологій та електроніки

Кафедра інформаційних технологій та програмування

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної випускної роботи

освітній ступінь бакалавр
спеціальність 121 „Інженерія програмного забезпечення“

на тему „Програмний комплекс для передачі даних зашифрованим каналом зв’язку на основі алгоритму DES шифрування“

Виконав: студент групи ІПЗ-20д _____ О.В. Тимашов
(підпис) (ініціали і прізвище)

Керівник _____ Д.В. Ратов
(підпис) (ініціали і прізвище)

Завідувач кафедри _____ О.І. Захожай
(підпис) (ініціали і прізвище)

Рецензент__О.І. Захожай_____

Київ - 2024

СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Навчально-науковий інститут (факультет) інформаційних технологій та електроніки

Кафедра інформаційних технологій та програмування

Освітній ступінь бакалавр
спеціальність 121 „Інженер програмного забезпечення“
(шифр і назва спеціальності)

ЗАТВЕРДЖУЮ

Завідувач кафедри
_____Захожай О.І.
“____”_____2024 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ ВИПУСКНУ РОБОТУ СТУДЕНТУ

_____ Тимашов Олександр В’ячеславович_____
(прізвище, ім’я, по батькові)

1. Тема роботи __Програмний комплекс для передачі даних зашифрованим каналом зв’язку на основі алгоритму DES шифрування_____
Керівник роботи_____доцент Ратов Денис Валентинович,к.т.н., доцент,_____
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджений наказом університету від “06” __травня____2024 року №171/15.15-С

2. Строк подання студентом роботи__10.06.2024_____
3. Вихідні дані до роботи ___Об’єктом даної роботи є створення програмного комплексу для передачі даних зашифрованим каналом зв’язку_____
_____
_____
_____
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Вступ. Аналітичний огляд. Алгоритм DES шифрування та його властивості. Процес розробки та його результат. Висновок. Перелік використаних джерел._____
_____
_____
_____
_____
5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслеників)
_____
_____
_____

6. Консультанти розділів роботи

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

7. Дата видачі завдання:___30.03.2024_____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання кваліфікаційної випускної роботи	Строк виконання етапів	Примітка
	Одержання завдання на виконання роботи	30.03.24	виконано
	Укладення і погодження з керівником плану і етапів виконання роботи	07.04.24	виконано
	Узагальнення даних літературних джерел, укладання розділу «Аналіз предметної галузі»	20.04.24	виконано
	Аналіз шляхів виконання завдання. Вибір і погодження керівником оптимального шляху	01.05.24	виконано
	Укладання та тестування програмного продукту	05.05.24	виконано
	Укладання та погоджування пояснювальної записки з керівником	10.05.24	виконано
	Здача готової записки на кафедру	10.06.24	виконано
	Укладання доповіді і презентації	14.06.24	виконано

Студент _____ _____О.В. Тимашов_____
(підпис) (ініціали і прізвище)

Керівник роботи _____ _____Д.В. Ратов_____
(підпис) (ініціали і прізвище)

ЛИСТ ПОГОДЖЕННЯ І ОЦІНЮВАННЯ
дипломної роботи студента гр. ІПЗ-20д Тимашов О.В.

Науковий керівник

Доцент, к.т.н. _____ Ратов Д.В.

Оцінка наукового керівника: _____

Рецензент Захожай О.І., СНУ ім. В.Даля, зав.каф.ІТП

ПІБ, місто роботи, посада

Оцінка рецензента: _____

Кінцева оцінка за результатами захисту:

_____

Голова ЕК

_____ Меняйленко О.С.

підпис

РЕФЕРАТ

Робота містить: 65 сторінок основного тексту, 20 сторінок додатків, 23 рисунків, 14 використаних джерел, 10 таблиць.

Об'єкт дослідження – розробка, реалізація та тестування програмного забезпечення, яке дозволяє користувачам безпечно обмінюватися інформацією, використовуючи захищений канал зв'язку засобами шифрування за допомогою алгоритму шифрування DES.

Предмет дослідження – процес розробки програми передачі даних по зашифрованому каналу зв'язку, використовуючи алгоритм шифрування DES. Це включає вивчення всіх етапів створення додатку: починаючи з аналізу вимог і проектування архітектури додатка, розробки коду, тестування і налагодження, а також документування всього процесу. Основна увага приділяється реалізації алгоритму DES усередині додатка та його інтеграції з каналом зв'язку для забезпечення безпечної передачі даних.

Мета роботи – створення програми, яка дозволяє користувачам обмінюватись повідомленнями в локальній мережі.

При розробці цієї програми для операційної системи Windows необхідно вирішити такі задачі:

• Проаналізувати класифікацію криптографічних алгоритмів

• Реалізувати передачу даних через локальну мережу

• Розробити інтерфейс користувача

• Реалізувати алгоритм DES для шифрування та дешифрування даних

• Розглянути структуру, режими роботи та криптостійкість алгоритмів

Технологія розробки – код мовою програмування C#.

Результатом роботи є працюючий клієнт-серверний чат із шифруванням даних.

Ключові слова: АЛГОРИТМ DES, ШИФРУВАННЯ, КЛЮЧ ШИФРУВАННЯ, КРИПТОСТІЙКІСТЬ, ПЕРЕДАЧА ДАНИХ, БЕЗПЕКА ДАНИХ.

ВСТУП

Актуальність досліджень. У сучасному цифровому світі, де кількість даних, що передаються щодня, зростає в геометричній прогресії, забезпечення їх безпеки є надзвичайно важливим завданням. Незважаючи на те, що алгоритм DES був випущений у 1970-х роках, він досі залишається одним із найпопулярніших та найефективніших алгоритмів шифрування. Розробка програмного забезпечення для передачі даних з використанням цього алгоритму є актуальним завданням, яке спрямовано на забезпечення конфіденційності та безпеки обміну інформацією.

Об’єкт дослідження: Криптографічний алгоритм DES.

Предмет дослідження: Розробка програмного забезпечення для передачі даних по зашифрованому каналу зв’язку у локальній мережі на основі алгоритму шифрування DES.

Мета дослідження: Метою даної роботи є розробка програми, що забезпечує зашифровану передачу даних у локальній мережі за допомогою DES шифрування.

Завдання дослідження:

1. Проаналізувати теоретичні відомості про алгоритм DES.
2. Здійснити програмну реалізацію алгоритму DES шифрування.
3. Провести тестування розробленої програми для підтвердження її працездатності.

РОЗДІЛ 1. АНАЛІТИЧНИЙ ОГЛЯД

Засоби та системи криптографічного захисту інформації грають важливу роль у сучасних комп'ютерних інформаційних системах що використовуються у сфері фінансової і комерційної діяльності. Сучасна криптологія базується на багатьох математичних дисциплінах таких як: лінійна алгебра, теорія груп, напівгруп, автоматов, математичний аналіз, теорія дискретних функцій, теорія чисел, комбінаторний аналіз, теорія ймовірностей, теорія інформації, теорія складності обчислень.

Знання основ криптографічних методів захисту інформації є невід'ємною складовою культури сучасної людини, яка стикається практично кожен день застосуванням сучасних мережевих технологій. В даний час криптографічні функції застосовуються для вирішення наступних задач захисту інформації:

1. Забезпечення конфіденційності інформації.
2. Забезпечення цілісності інформації.
3. Автентифікації інформації.
4. Посвідчення авторства стосовно повідомлення чи документа
5. Забезпечення неналежності інформації

Метою цього аналітичного огляду є оцінка проблем і майбутніх перспектив розвитку криптографічних алгоритмів, а також порівняння та аналіз алгоритмів, які можуть конкурувати з відомим алгоритмом DES. Також у цьому огляді ми досліджуємо поточні виклики сучасних криптографічних алгоритмів, включаючи проблеми безпеки та потребу в постійному вдосконаленні.

1.1. Класифікація криптографічних алгоритмів

Існує кілька моделей класифікації криптографічних алгоритмів, кожна з яких базується на групі характеристик. Таким чином, один і той же алгоритм «проходив» відразу кілька схем, потрапляючи в одну з підгруп в кожній з них. Залежно від наявності або відсутності ключа алгоритми кодування поділяються на шифрувальні

та криптографічні. Залежно від збігу ключів шифрування та дешифрування – симетричні та асиметричні. Залежно від типу використовуваних перетворень - підстановка і перестановка. Залежно від розміру зашифрованого блоку - потокове та блочне шифрування.

Одна з них поділяє криптографічні алгоритми залежно від числа ключів, що застосовуються у конкретному алгоритмі. Класифікація по цьому критерію представлена на рисунку 1.1.

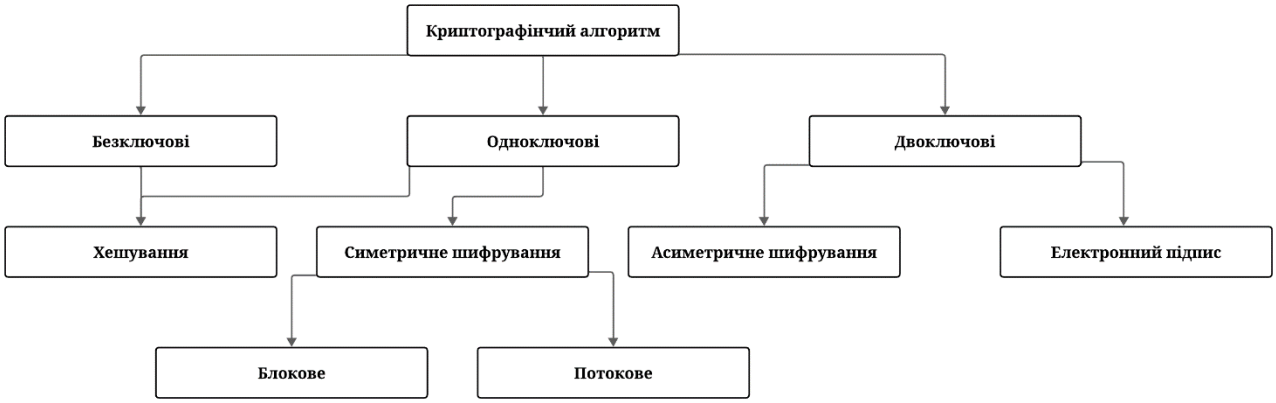

Рисунок 1.1. – Класифікація криптографічних алгоритмів за кількістю ключів

Безключові алгоритми – не використовують ключі в обчисленнях ніяких ключів.

Одноключові алгоритми – працюють з одним ключовим параметром (секретним ключом)

Двох ключові – на різних стадіях роботи в них застосовуються два ключових параметра: секретний та відкритий ключ.[1]

Хешування – це метод криптозахисту, що представляє собою контрольне перетворення інформації: з даних необмеженого розміру шляхом виконання криптографічних перетворень обчислюється хеш-значення фіксованої довжини, однозначно відповідне вихідним даним. Фактично це контрольне підсумовування даних, яке може виконуватися як за участю ключа, так і без нього. Такі функції мають досить широке застосування в області захисту комп'ютерної інформації:

1. для підтвердження цілісності будь-яких даних в тих випадках, коли використання електронного підпису неможливо (наприклад, через велику ресурсоємності) або є надлишковим;

2. в схемах електронного підпису – підписується зазвичай хеш даних, а не всі дані цілком;

3. в різних схемах аутентифікації користувачів.

Інша класифікація поділяє криптоалгоритми в залежності від типу виконаних перетворень над вхідним текстом. Класифікація згідно з цим критерієм наведена на рисунку 1.2.

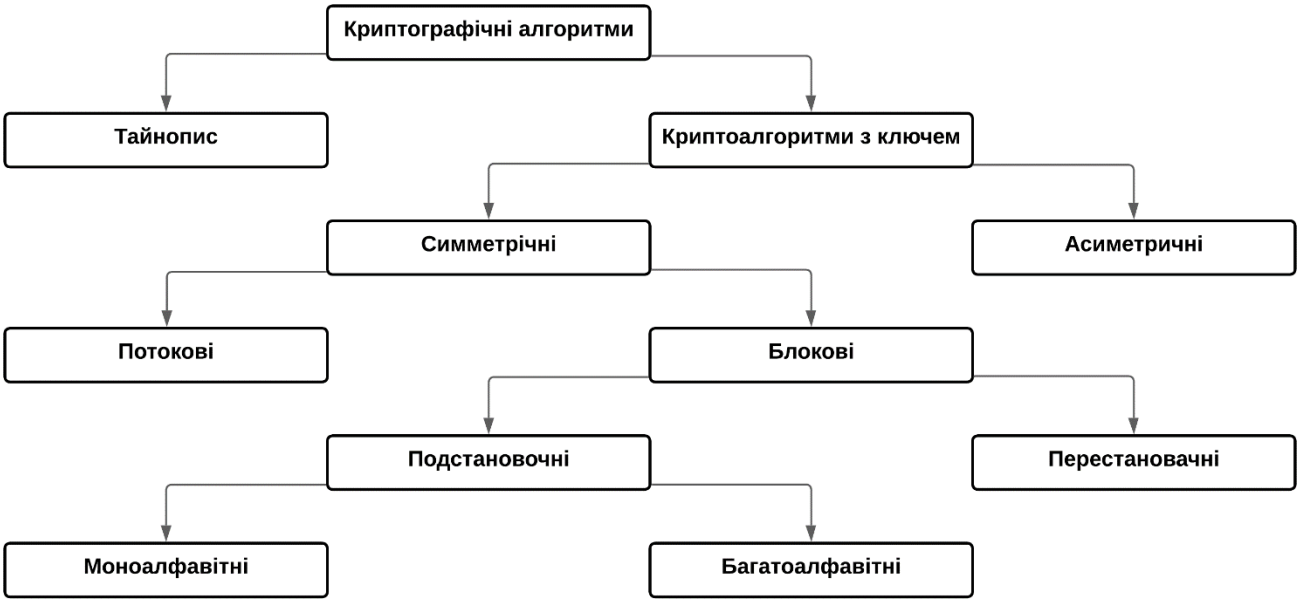

Рисунок 1.2 – Класифікація криптографічних алгоритмів за типом перетворення.

Тайнопис передбачає, що відправник і одержувач роблять над повідомленням перетворення, відомі лише їм двом. Стороннім особам невідомі виконувані алгоритмом зміни над відкритим текстом, що є гарантією нерозкриття даних на етапі аналізу.

На противагу тайнопису, криптоалгоритми з ключем побудовані тому принципі, що алгоритм на передані дані відомий всім стороннім особам, але залежить від деякого параметра, що тримається у секреті – «ключа», який відомий

лише двом особам, що у обміні інформацією. Основу такого підходу заклав голландський криптоаналізатор, який на початку XX століття запропонував шифрування за допомогою секретного ключа. Це означає, що криптоаналітику може бути відомо все про процес шифрування та декодування тексту, але не відомо, який саме ключ використовувався для цього процесу. Даний підхід обумовлений тим, що захищеність системи не повинна залежати від того, що не неможливо або складно замінити у випадку витоку секретної інформації.

Криптосистеми з ключем поділяються на симетричні та асиметричні. Відмінною рисою симетричних алгоритмів шифрування є наявність одного ключа шифрування (k на рисунку 1.3), який має бути відомий лише відправнику та одержувачу повідомлення. Відправник ключа k шифрує повідомлення, одержувач дешифрує отриманий шифротекст ключем k. Криптоаналітик може перехопити шифротекст Y, що передається відкритими каналами зв'язку, але, оскільки він не знає ключа, завдання розкриття шифротексту є дуже трудомістким. Принциповим моментом є необхідність наявності секретного каналу зв'язку між одержувачем і відправником передачі ключа шифрування без можливості його перехоплення криптоаналітиком. Модель симетричної системи показана на рисунку 1.3.

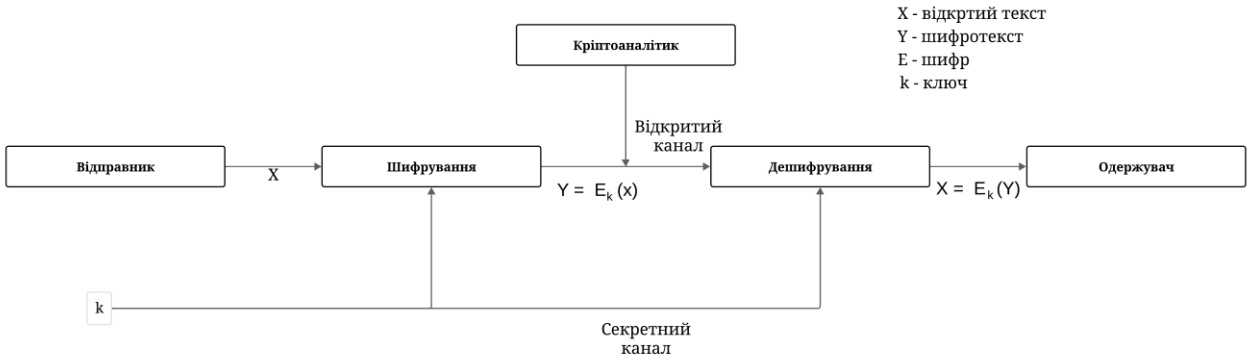

Рисунок 1.3 – Модель симетричного шифрування.

Відмінною особливістю асиметричних алгоритмів є наявність пари ключів шифрування: відкритого ключа шифрування (квід), який передається другій стороні по незахищеному каналу зв'язку і тому може бути відомий криптоаналітику, а також закритого ключа (ksak), який відомий лише одній людині (отримувачу повідомлення)

і тримається в секреті. Пара ключів має ту властивість, що повідомлення, зашифроване на одному з ключів, може бути розшифроване тільки на іншому ключі. Фактично це означає, що секретним каналом передачі на схемі (рис. 1.4) є напрямок «відправник-одержувач», оскільки повідомлення, зашифроване на відкритому ключі відправником, може дешифрувати своїм закритим ключем лише одержувач.

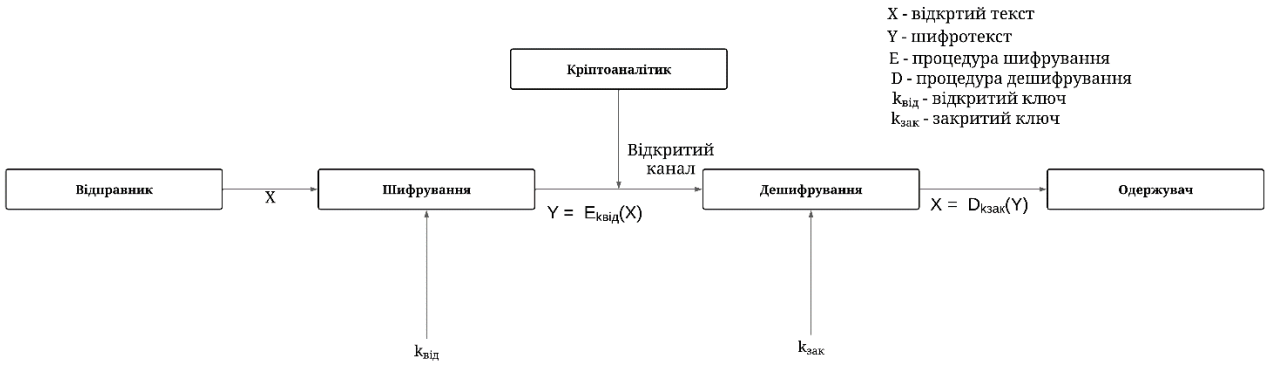

Рисунок 1.4 – Модель асиметричного шифрування.

Залежно від розміру блоку інформації, що шифрується, симетричні криптоалгоритми діляться на блокові і поточні шифри.

Блокове шифрування – у цьому випадку інформація розбивається на блоки фіксованої довжини (наприклад, 64 або 128 біт), після чого ці блоки по черзі шифруються. Причому у різних алгоритмах шифрування або навіть в різних режимах роботи одного і того же алгоритму блоки можуть шифруватися незалежно один від одного або "зі зчепленням" – коли результат зашифрування поточного блока даних залежить від значення попереднього блоку або від результату шифрування попереднього блоку.

Потокове шифрування – необхідно перш за все, в тих випадках коли інформацію неможливо розбити на блоки – потік даних, кожен символ якого бути зашифрований і відправлений куди-небудь, не чекаючи інших даних достатніх до формування блоку. Тому алгоритми потокового шифрування шифрують дані побітно або посимвольно. Деякі класифікації не поділяють блочне і потокове шифрування, вважаючи, що потокове шифрування – це шифрування блоків одиничної довжини [5].

Ще одним критерієм класифікації криптоалгоритмів є тип перетворень над блоками відкритого тексту. За цим критерієм криптоалгоритми поділяють на підстановочні та перестановочні (рисунку 1.2). У перестановочних шифрах блоки інформації не змінюються власними силами, але змінюється їхній порядок прямування, що робить інформацію недоступною сторонньому спостерігачеві. Підстановочні шифри змінюють блоки інформації за певними законами.

Поділ криптоалгоритмів на моноалфавітні та багатоалфавітні характерно для підстановочних шифрів. Моноалфавітні криптоалгоритми замінюють блок вхідного тексту (символ вхідного алфавіту) на той самий блок шифротекта (символ вихідного алфавіту). У багатоалфавітних шифрах одному й тому блоку вхідного тексту можуть відповідати різні блоки шифротексту, що суттєво ускладнює криптоаналіз.

За ступенем секретності криптоалгоритми діляться абсолютно стійкі і практично стійкі. Абсолютно стійкі шифри неможливо розкрити. На практиці цього можна досягти, тільки якщо розмір використовуваного ключа шифрування перевищує розмір повідомлення, що кодується, і при цьому ключ використовується одноразово. Практично стійким називається шифр, для якого не існує більш ефективного способу злому, крім повного перебору всіх можливих ключів шифрування [3].

1.2 Структура алгоритмів

Переважна більшість сучасних алгоритмів шифрування працюють вельми схожим образом: над шифруємим текстом виконується якесь перетворення за участю ключа шифрування, яке повторюється певне число разів (раундів). При цьому по виду повторюваного алгоритми шифрування прийнято ділити на кілька категорій.

1.2.1 Алгоритми на основі мережі Фейстеля

Мережа Фейстеля (Feistel network) має на увазі розбиття оброблюваного блоку даних на кілька субблоків, один з котрих оброблюється якоюсь функцією f та

накладається на один або декілька інших субблоків. На рисунку 1.5 приведена найбільш розповсюджена структура алгоритмів на основі мережі Фейстеля.

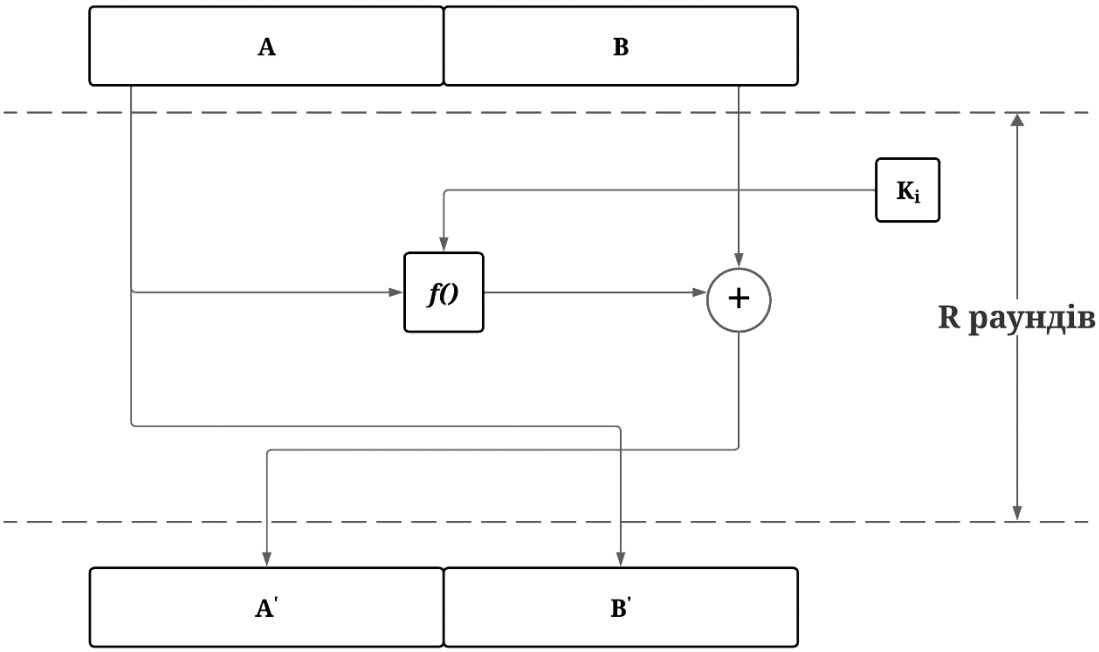

Рисунок 1.5 – Мережа Фейстеля.

Додатковий аргумент функції f позначений на рис 1.5 як K_i називається ключем раунду. Ключ раунду є результатом обробки ключа шифрування процедурою розширення ключа, завдання якої – отримання необхідної кількості ключів K_i з вихідного ключа шифрування відносно невеликого розміру. У найпростіших процедура розширення ключа просто розбиває ключ на кілька фрагментів, які по черзі використовуються у раундах шифрування.

Така структура алгоритмів шифрування отримала назву по імені одного с розробників алгоритмів шифрування Lucifer та розробленого на його основі алгоритму DES – Хорста Фейстеля.

На мережі Фейстеля засновано більшість сучасних алгоритмів шифрування завдяки безлічі переваг подібної структури, серед котрих варто відзначити наступні:

- Алгоритми на основі мережі Фейстеля можуть бути сконструйовані таким чином, що для шифрування и дешифрування може використовуватися один і той сам код лише в порядку застосування

ключів – різниця між цими операціями може лише складатися в порядку застосування ключів.

- Алгоритми на основі мережі Фейстеля є найбільш вивченими – таким алгоритмам присвячена величезна кількість криптоаналітичних досліджень, що є безперечною перевагою, як при розробці алгоритму, так і при його аналізу.
-

1.2.2 Алгоритми на основі підстановочно-перестановочних мереж

На відміну від мережі Фейстеля, SP – мережі (substitution-permutation network, підстановочно-перестановочна мережа) обробляють за один раунд цілком шифрований блок. Обробка даних зводиться в основному до замін (коли, наприклад, фрагмент вхідного значення замінюється іншим фрагментом відповідно з таблицею замін, яка може залежати від значення ключа K_i) і перестановок, що залежать від ключа. Спрощена схема показана на рисунку 1.6.

SP-мережі є набагато менш поширеними, ніж мережі Фейстеля. Як приклад SP-мереж можна привести алгоритми Serpent або SAFER+.

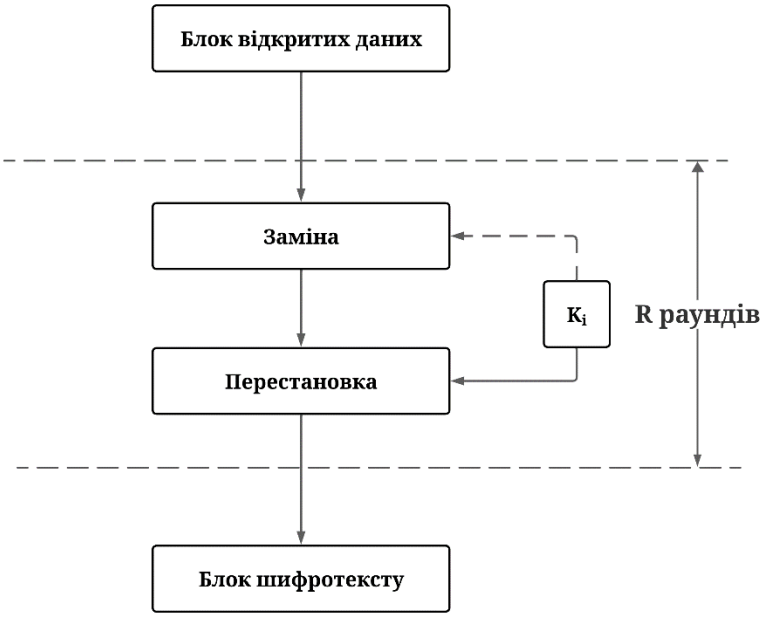

Рис. 1.6. SP-мережа.

1.2.3 Алгоритми зі структурою "квадрат"

Для структури "квадрат" (Square) характерно представлення блоку, що шифрується даних у вигляді двомірного байтового масиву. Криптографічні перетворення можуть виконуватися над окремими байтами масиву, також над його рядками та стовпцями. Недоліком алгоритмів зі структурою "квадрат" є їх недостатня вивченість.

1.2.4 Алгоритми з нестандартною структурою

Класифікувати всі можливі варіанти алгоритмів шифрування представляється складною задачею, тобто існують такі алгоритми, які неможливо зарахувати до жодного з перерахованих типів. Як приклад алгоритму з нестандартною структурою можна привести унікальний за своєю структурою алгоритм FROG, в кожному раунді якого за досить складними правилами виконується модифікація двох байтів даних, що шифруються.

Сурові межі між описаними вище структурами не визначені, тому досить часто зустрічаються алгоритми, які різними експертами зараховуються до різних структур. Наприклад, алгоритм CAST-256 відноситься його автором до SP-мережі, а багатьма експертами називається розширеною мережею Фейстеля.

1.3 Режими роботи алгоритмів

1980 році в США був прийнятий стандарт, що визначає режим роботи алгоритму DES-стандарту. Можна сказати що цей стандарт уточнював подробиці реалізації DES для різних застосувань.

У стандарті були визначені наступні режими роботи алгоритму DES:

- Електрона кодова книга ECB (Electronic Code Block);
- Зчеплення блоків шифру CBC (Cipher Block Chaining);
- Зворотний зв’язок по шифротексту CFB (Cipher Feed Back);
- Зворотній зв’язок по виходу OFB (Output Feed Back).

Ці режими фактично не прив’язані до конкретного алгоритму – фактично будь-які алгоритми блочного симетричного шифрування можуть бути використанні в даних режимах роботи

1.3.1 Електрона кодова книга (ECB)

Найбільш простим з режимів є ECB. Суть його полягає в тому, що кожен блок шифруємих даних "проганяється" через алгоритм шифрування окремо і незалежно від інших блоків:

$$C_i = E_k(M_i),$$

де:

- E_k функція шифрування на ключі k.
- C_i та M_i блоки відкритого і відповідні їм блоки шифротексту

Режим ECB поганий тим, що якщо відкритий текст містить якусь кількість блоків з однаковим вмістом (наприклад великий масив з ініціалізований нульовими або одиничними бітами) то і шифро текст буде містить таку ж кількість однакових блоків. Ще один недолік алгоритм шифрує дані тільки поблочно. При необхідності зашифрувати алгоритмом DES 8 бітів доведеться доповнити ці дані до 64- бітного розміру блоку, зашифрувати блок цілком, а при розшифруванні відкинути доповнюючи біти.

Перевага режиму ECB – простота його реалізації в порівнянні з іншими режимами.

1.3.2 Зчеплення блоків шифру (CBC)

Суттєво більш стійким є режим CBC. У цьому режиму перед шифруванням кожного I-го блоку шифруємих даних виконується його побітове додавання по модулю 2 з результатом шифрування попереднього (i-1)-го, блоку. Результат шифрування кожного блоку залежить не тільки від вмісту шифруваного блоку, але від всіх попередніх блоків відкритого тексту. При шифруванні першого блоку

відкритого тексту замість результату зашифровування попереднього блоку використовується вектор ініціалізації.

Варіюючи значення вектору ініціалізації, можна отримувати різні шифротексти для однакових відкритих текстів. При розшифруванні шифротексту, отриманого в режимі CBC, вектор ініціалізації повинен бути відомий. Це справедливо також і для режимів CFB і OFB.

Аналогічно попередньому режиму, дані, розмір яких меньше 64- бітного блоку, доведеться перед обробкою доповнювати до 64 бітів.

Режим CBC використовується безпосередньо для шифрування даних, у том числі в великих обсягах. Крім того, останній блок шифротексту може використовуватися для контролю цілісності інформації повідомлень, оскільки його значення залежить від вмісту всіх блоків відкритого тексту, вектору ініціалізації та ключа.

1.3.3 Зворотний зв'язок по шифротексту (CFB)

Режим CFB більш складний в реалізації, ніж два попередніх. Шифрування в цьому режимі виконується наступним чином.

1. Вектор ініціалізації записується в регістр 1.
2. 64- бітний вміст регістру шифрується та поміщається у регістр 2.
3. З регістра 2 вибираються L лівих бітів ($1 \leq L \leq 64$), які накладаються операцією XOR на L- бітний блок шифруємого тексту. Результат виконання цього кроку L- бітний блок шифротексту.
4. Вміст регістра 1 зсувається вліво на L бітів.
5. Регістр 1 доповнюється праворуч L- бітовим блоком шифротексту. При необхідності продовжити шифрування даних кроки 2-5 повторюються в циклі до обробки всіх зашифрованих даних.

Фактично в режимі CFB алгоритм шифрування на основі вектору ініціалізації, ключа шифрування і попередніх блоків шифротексту генерує псевдовипадкову послідовність, яка накладається на відкритий текст.

1.3.4 Зворотній зв’язок по виходу (OFB)

Режим шифрування OFB схожий на попередній:

1. Вектор ініціалізації записується в регістр 1.
2. Вміст регістра 1 зашифровується, результат поміщається в регістри 1 і 2
3. З регістра 2 вибираються L лівих бітів, які накладаються операцією XOR на L- бітний блок шифруємого тексту, в результаті виходить L- бітний блок шифротексту. При необхідності продовжити шифрування кроки 2-3 повторюються в циклі.

Відмінність від попереднього режиму лише в тому, що значення регістра 1 замінюється в циклі його ж зашифрованим вмістом. Звідси виникає важлива властивість режиму OFB – послідовність, що накладається на шифротекст залежить тільки від значення вектору ініціалізації і ключа. Тобто шифруючу послідовність можна згенерувати заздалегідь і, наприклад, використовувати її в періоди максимального навантаження на шифруючий комп’ютер або пристрій, заповнюючи послідовність у фоновому режимі.

Ще одна важлива відмінність режиму OFB від інших полягає у тому, що при виникненні помилку в одному біті шифротексту після розшифрування виникає помилка лише в одному біті розшифрованих даних, тоді як в інших режимах помилково розшифруються одни або два блоки.

1.4 Криптостійкість алгоритмів

Криптографічна стійкість – це здатність криптографічного алгоритму бути стійким до криптографічного аналізу. Стійким вважається алгоритм, який для успішної атаки вимагає від противника недосяжних обчислювальних ресурсів, недосяжного обсягу перехоплених відкритих і зашифрованих повідомлень чи ж такого часу розкриття, що по його закінченню захищена інформація буде вже не актуальна, і т. д. Здебільшого крипостійкість не можна математично довести, можна тільки довести уразливості криптографічного алгоритму [6].

Криптостійкість алгоритму можна визначати за допомогою спеціальних одиниць вимірювання стійкості, таких як:

- Витрати часу на злам ключа, включаючи розробку відповідної обчислювальної моделі.
- Необхідний обсяг пам’яті для зламування ключа.
- Вартісні витрати на злам ключа.
- Кількість необхідної енергії, що витрачається на злам ключа.
- Фізичний об’єм обчислювальної моделі для злама ключа

Час, протягом якого забезпечується збереження конфіденційності зашифрованої інформації, визначається сукупністю різних факторів, що включають у собі:

- Продуктивність обчислювальної моделі, що знаходиться в розпорядженні.
- Обсяг пам’яті обчислювальної моделі.
- Складність найкращого з відомих алгоритмів, що вирішує завдання розтину.
- Можливість отримання додаткової інформації про ключ, наприклад, наявність відкритих і відповідних зашифрованих текстів, можливість зашифровування або розшифрування спеціальним чином підібраних текстів [7].

Алгоритми в криптографії можна віднести до трьох типів за рівнем криптографічної стійкості:

I. Обчислювальна стійкість – це можливість потенційного криптоаналітичного розкриття шифру, коли при всіх заданих параметрах та ключах алгоритму шифрування на сучасної стадії розвитку криптографічного аналізу у зловмисника немає необхідних обчислювально-алгоритмічних та тимчасових ресурсів, щоб реалізувати розтин. Вважається, що якщо алгоритм, який здійснює розтин шифру, повинен виконати більше або порядку 280 операцій, шифр є обчислювально стійким. Не є обчислювально стійкими:

шифри зсуву, заміни, віженера. До обчислювально стійких шифрам відносяться DES, AES, RSA, шифр Ель-Гамалю;

II. Інформаційно-теоретична стійкість (або абсолютна стійкість) – це ситуація, коли криптоаналітик не здатний розкрити криптосистему ні теоретично, ні практично навіть якщо він має нескінченно великі обчислювально-алгоритмічні ресурси. Докази криптографічної стійкості моделі такого типу виводяться з теорії інформації;

III. Доказна стійкість – за якої доказ алгоритмічної стійкості криптографічної системи представляється як рішення цілком конкретної важко вирішуваної математичної проблеми, покладеної основою алгоритмічного коду [8].

Серед методів аналізу на криптографічну стійкість слід звернути увагу насамперед повний перебір всіх ключів (чи спосіб «грубою сили»), так як криптоаналітичне розтин даним методом можлив для всіх типів криптографічних алгоритмів, крім абсолютно стійких «по Шеннону». Для новоствореного алгоритму даний метод криптографічного аналізу часто є єдиним можливим. Методи їх аналізу значною мірою залежать від труднощів обчислення алгоритмічної складності коду, яка потім виражається в об'ємах часу, що витрачається, а також у грошах, необхідної продуктивності апаратури та її обчислювально-алгоритмічних ресурсах. Зокрема, алгоритмічний код можна позначити як криптографічно стійкий, якщо немає способу його «злому» значно швидше, ніж метод повного перебору всіх ключів. Криптографічні атаки, в основі яких лежить метод повного перебору або «грубої сили», є на даний момент найуніверсальніший спосіб злому алгоритмічного коду, але при цьому він є і найтривалішим за часом. Метод «грубою сили» є найкращим варіантом розкриття криптографічного алгоритму, якщо немає чи не вдається знайти вразливостей у системі шифрування, або у цій системі шифрування слабких місць немає. Якщо ж такі вразливості буде знайдено, то зловмисниками розробляється методики

криптографічного аналізу з урахуванням виявлених особливостей системи, що значно підвищує ймовірність розтину [7].

По-друге, для подальшого вивчення алгоритму з метою знаходження його слабкостей (вразливостей), потрібно оцінити його стійкість щодо інших відомих методів встановлення криптостійкості, таких як лінійний криптоаналіз, диференціальний криптоаналіз та більш специфічні, які повинні знизити існуючу стійкість [9].

Наприклад, для великої кількості симетричних шифрів є ненадійні ключі та S-блоки, використання яких зменшує криптографічну стійкість .

Важливо звернути увагу на те, що особливим методом здійснення криптографічної стійкості є атаки на реалізацію, які використовуються спеціально для конкретного програмно-апаратно-людського комплексу. Вони утворюють особливий тип атак, які націлені на вразливі місця в практично криптографічної експлуатації системи. Це істотно відрізняється від теоретичного криптографічного аналізу, так як атаки по стороннім каналам реалізують інформацію про фізичні процеси в обчислювальній техніці, які не схильні до розгляду в теоретичній основі криптографічного алгоритму.

Також важливо підкреслити значущість тривалої перевірки та відкритого обговорення криптографічних методів та алгоритмів. Тривалий аналіз та висока кваліфікація експертів щодо алгоритму та його реалізацій породжують впевненість у криптографічної стійкості, якщо спроби злому не дали результатів. Існують приклади, коли тривалий та уважний аналіз призводив до зменшення криптографічної стійкості нижче за прийнятний рівень (наприклад, у чорнових версіях FEAL). Відомо що недостатня перевірка (на думку багатьох криптографів - штучне ослаблення) алгоритму потокового шифрування A5/1 призвела до успішної атаки на нього.

1.5 Постановка задачі

В результаті виконання дипломного проекту повинен бути створена програма яка дозволяє користувачам обмінюватися повідомленнями у локальній мережі. Програма повинна бути зручною для користувача, проста в використанні, інтуїтивно зрозумілою. Програма призначається для функціонування в операційній системі Windows.

При розробці даної програми для операційної системи Windows потрібно вирішити наступні завдання:

- Проаналізувати класифікації криптографічних алгоритмів
- Реалізувати передачу даних через локальну мережу
- Розробити інтерфейс користувача
- Реалізувати алгоритм DES для шифрування и дешифрування даних в програмі
- Розглянути структуру, режими роботи та криптостійкість алгоритмів

РОЗДІЛ 2. СТАНДАРТ ШИФРУВАННЯ DES. АЛГОРИТМ ТА ЙОГО ОСНОВНІ ВЛАСТИВОСТІ

2.1 Основні характеристики та структура алгоритму DES

Алгоритм DES використовує комбінацію підстановок та перестановок. Він здійснює шифрування 64-бітових блоків даних за допомогою 64-бітового ключа, в якому значущими є 56 біт (інші 8 біт перевірочні біти для контролю на парність). Дешифрування в DES є операцією зворотною шифруванню і виконується шляхом повторення шифрування у зворотній послідовності. Узагальнену схему процесу шифрування в алгоритмі DES показано на рисунку 2.1. Процес шифрування полягає у початковій перестановці 64-бітівого блоку, шістнадцяти циклах шифрування і кінцевій перестановці бітів [11].

Рис. 2.1. Узагальнена схема DES шифрування.

Шифрування даних виконується наступним чином (рис. 2.2):

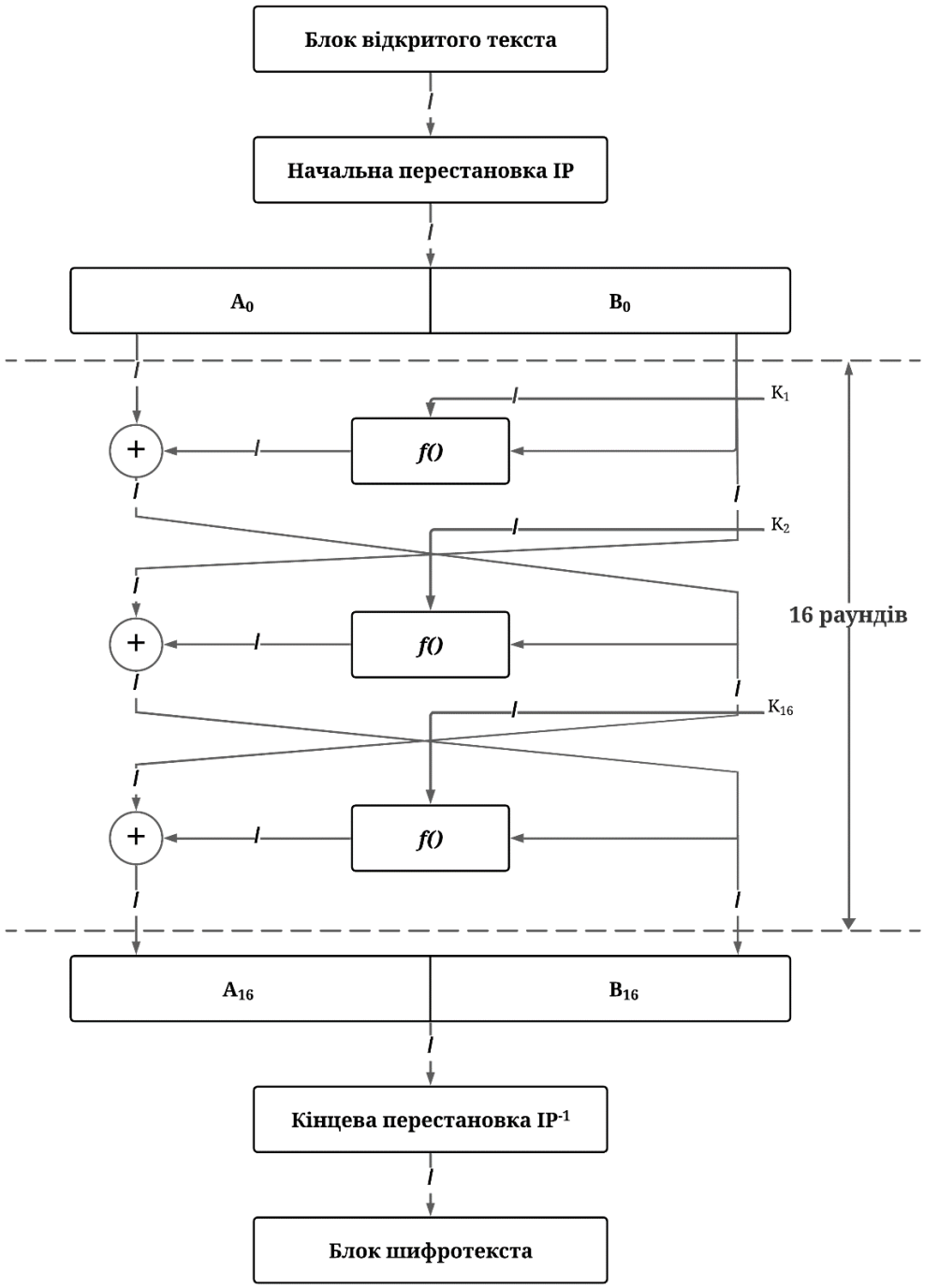

Рис. 2.2. Структура алгоритму DES

1. Над 64 бітним блоком даних проводиться початкова перестановка (IP) згідно з таблицею згідно з таблицею 2.1.

Таблиця трактується так: значення вхідного біта 58 (тут і далі всі біти нумеруються зліва направо, починаючи з 1-го) поміщається у вихідний біт 1, значення 50-го біта в біт 2 і т. д. [5]

Таблиця 2.1. Начальна перестановка IP

58	50	42	34	26	18	10	2	60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6	64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1	59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5	63	55	47	39	31	23	15	7

2. Результат попередньої операції ділиться на 2 субблоки по 32 біти (на рис. 2.2 позначені як A_0 і B_0) над якими здійснюються 16 раундів наступних перетворень:

$$A_i = B_{i-1};$$

$$B_i = A_{i-1} \oplus f(B_{i-1}, K_i),$$

де:

- i – номер поточного раунду;
- K – ключ раунду;
- $\oplus$ - побітова логічна операція «виключне або» (XOR)

Структура функції раунду f() наведено на рис 2.3. Ця функція виконується у декілька кроків.

- Над 32-бітним входом виконується розширювальна перестановка E (рис. 2.3). Ця операція розширює вхідне значення (32 біта) до 48-бітів для подальшого складання з ключем раунду. Функція розширення визначається таблицею 2.2. Таблиця трактується так: значення вхідного біта 32-го поміщається у вихідний біт 1, значення 1 біту в 2 біт і т. д. [5]

Таблиця 2.2. Функція розширення E

32	1	2	3	4	5

Таблиця 2.2. Функція розширення E

4	5	6	7	8	9
8	9	10	11	12	13
12	13	14	15	16	17
16	17	18	19	20	21
20	21	22	23	24	25
24	25	26	27	28	29
28	29	30	31	32	1

- Результат попереднього раунду складається з 48-бітним ключем раунду (K_i), який є результатом перетворень 64-бітового (K), операцією XOR.
- Результат додавання розбивається на 8 фрагментів по 6 бітів, кожен із яких «проганяється» через відповідну таблицю замін (S1…S8). Таблиці замін є зафіксованими та наведені у Додатку А. Кожна таблиця містить по 4 рядки, що містять по 16 значень від 0 до 15. Вхідне значення інтерпретується наступним чином: два крайні біти формують номер рядка (від 0 до 3), з якого вибирається число, що у стовпці, номер якого відповідає значенню чотирьох інших бітів входу. Наприклад, нехай на вхід матриці S_1 поступає 6-бітовий блок $B_1 = 100110 = b_1b_2b_3b_4b_5b_6$, то 2 крайніх біта $b_1b_6 = 10_{(2)} = 2_{(10)}$ вказує рядок з номером 2 матриці S_1, а 4-бітове число $b_2b_3b_4b_5 = 0011_{(2)} = 3_{(10)}$ вказує стовбець із номером 3 матриці S_1. Це означає, що у матриці S_1 блок B_1 вибирає елемент на перетині рядка з номером 2 і стовпця з номером 3, тобто елемент $8_{(10)} = 1000_{(2)}$. Сукупність 6-бітових блоків B_1, B_2, …, B_8 забезпечує вибір 4-бітового елемента в кожній з матриць S_1, S_2…, S_8. На останньому кроці 4 бітові значення, отриманні після виконання замін, об’єднуються, після чого над ними виконується операція P, що є простою перестановкою згідно з таблицею 2.3.

- На останньому кроці 4-бітні значення, отримані після виконання замін, об’єднуються, після чого нам ними виконується операція P, що є перестановкою згідно з таблицею 2.3. [5]

Таблиця 2.3. Перестановка P

16	7	20	21	29	12	28	17
1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9
19	13	30	6	22	11	4	25

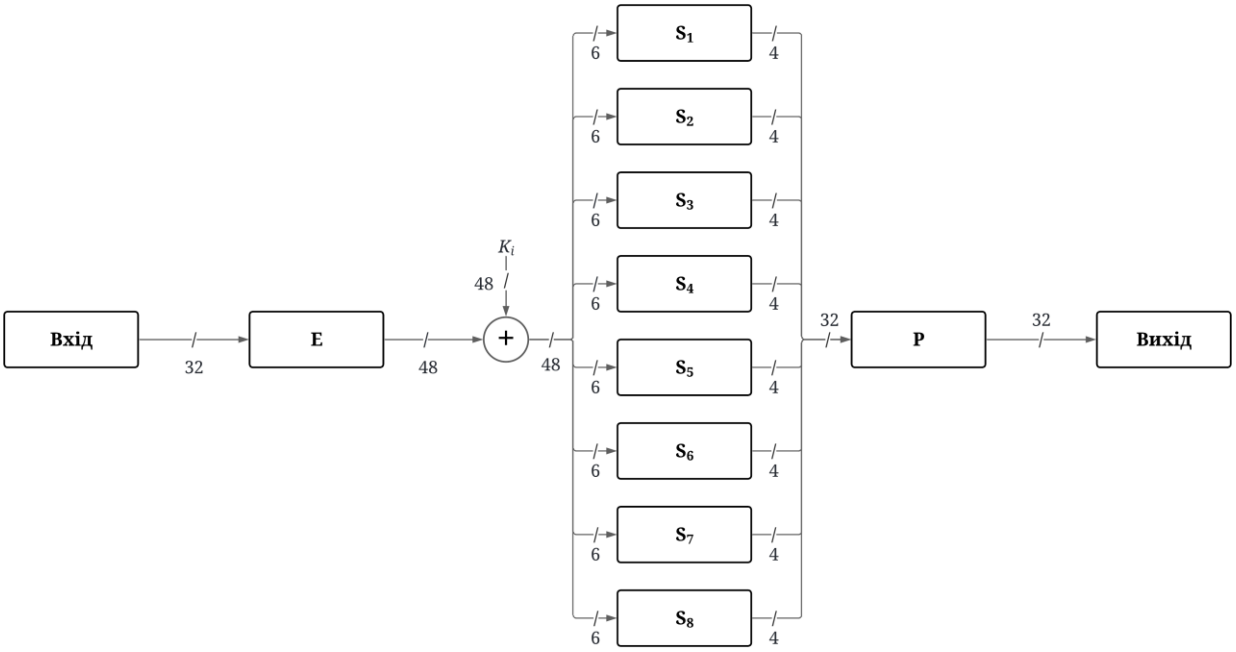

Рис 2.3. Функція *f()*

3. Отримані субблоки об’єднуються у 64-бітовий блок над, над яким виконується фінальна перестановка (IP^{-1}) даних згідно з таблицею 2.4. Фінальна перестановка (IP^{-1}) є інверсною по відношенню до початковою перестановки (IP), що виконується на 1 кроці. Результатом фінальної перестановки є блок зашифрованих даних.

Таблиця 2.4. Зворотня перестановка IP^{-1}

40	8	48	16	56	24	64	32	39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30	37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28	35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26	33	1	41	9	49	17	57	25

2.2 Процедура розширення ключа

Схема процедури розширення ключа показана на рисунку 2.4. Її завдання – формування 16 ключів раундів. Як неважко помітити, на кожній ітерації використовується нове значення ключа *K* (довжиною 48 бітів). Ключ *K* являє собою 64-бітовий блок з 8 бітами контролю на парність, які розташовані в позиціях 8, 16, 24, 32, 40, 48, 56, 64. Для видалення контрольних бітів та підготовки ключа до роботи використовується функція G (таблиця 2.5). Таблиця 2.5 поділена на дві частини. Результат перетворення G(*K*) розбивається на дві половини C_0 і D_0 по 28 біт кожна. Перші два рядки матриці G визначають, як вибираються біти послідовності C_0 (першим бітом буде біт 57 ключа шифру (*K*) потім 49 і т. д.). Останні два рядки матриці G визначають, як вибираються біти послідовності D_0. [5]

Таблиця 2.5. Функція G Початкова підготовка ключа.

57	49	41	33	25	17	9	1	58	50	42	34	26	18	C_0
10	2	59	51	43	35	27	19	11	3	60	52	44	36	
63	55	47	39	31	23	15	7	62	54	46	38	30	22	D_0
14	6	61	53	45	37	29	21	13	5	28	20	12	4	

Як бачимо з таблиці 2.5 для генерації C_0 і D_0 не використовуються біти 8,16, 24, 32, 40, 48, 56, 64 ключа шифру. Ці біти не впливають на шифрування і можуть служити для інших цілей (наприклад, контролю парності). Таким чином, ключ шифру є 56-бітовим насправді.

Після визначення C_0 і D_0 рекурсивно визначаються C_i і D_i, i= 1, 2, 3, …, 16. Для цього незалежно застосовується операція циклічного зсуву вліво на один або два біти в залежності від номеру кроку ітерації. Для раундів i= 1, 2, 9, 16 зсув відбувається на один біт, в решті раундів виконується зсув на два біти. Операція зсуву виконується для послідовностей C_i і D_i незалежно. Наприклад, послідовність C_3 виходить за допомогою циклічного зсуву вліво на два біти послідовності C_2, а послідовність D_3 за допомогою зсуву вліво на два біти послідовності D_2, C_{16} і D_{16} виходять із C_{15} і D_{15} за допомогою зсуву вліво на один біт.

Ключ K_i, що визначається на кожному кроку ітерації, є результатом вибору конкретних бітів з 56-бітової послідовності C_i D_i та їх перестановки. Іншими словами, ключ K_i = H(C_i D_i), де функція H визначається матрицею, що завершує обробку ключа (таблиця 2.6). [5]

Таблиця 2.6. Функція H фінальної обробки ключа.

14	17	11	24	1	5	3	28	15	6	21	10	23	19	12	4
26	8	16	7	27	20	13	2	41	52	31	37	47	55	30	40
51	45	33	48	44	49	39	56	34	53	46	42	50	36	29	32

Поєднання циклічного зсуву та фінальної перестановки ключа призводить до того, що в кожному з ключів раундів використовується унікальній набір бітів ключа шифрування

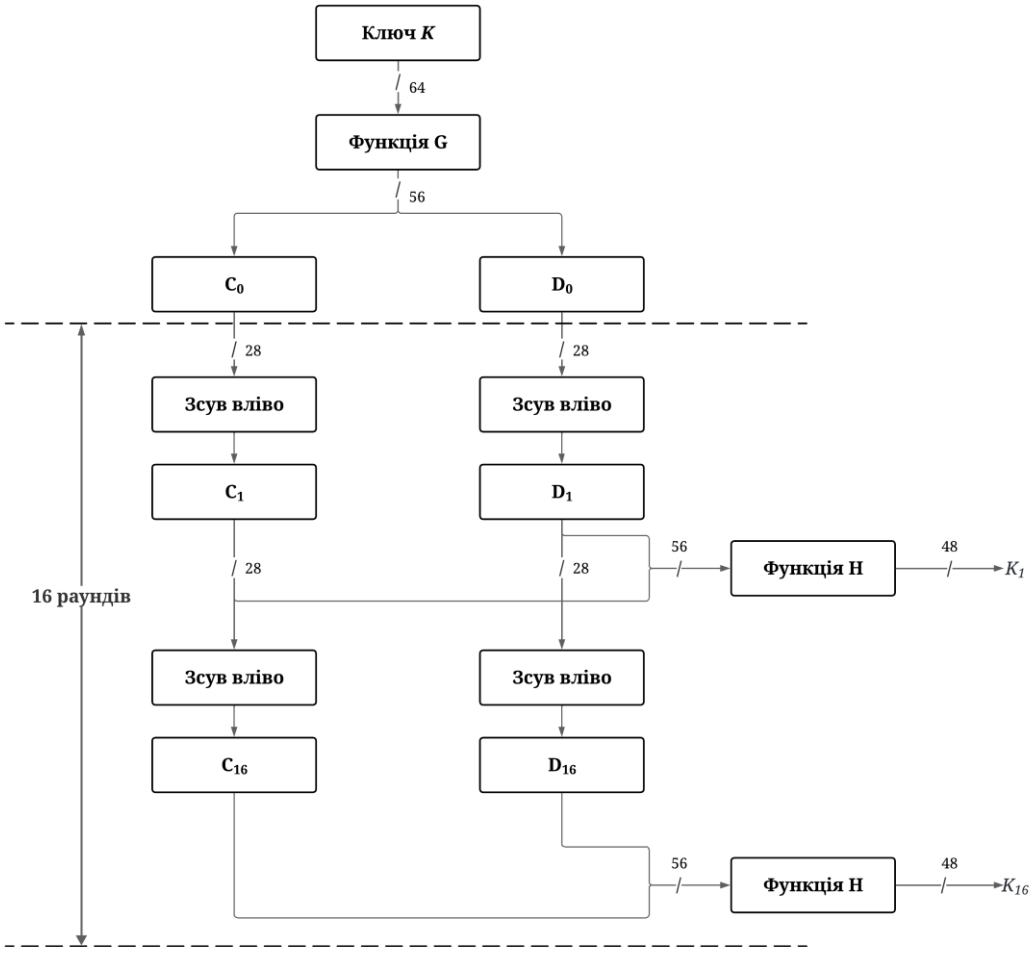

Рис 2.4. Процедура розширення ключа

При розшифруванні даних можна використовувати ту саму процедуру розширення ключа, але застосувати ключі раундів у зворотному порядку. Є й інший варіант: в кожному раунді процедури розширення ключа замість циклічного зсуву вліво виконувати циклічний зсув вправо, де 0 бітів зсув для першого раунду, 1 біт зсув для i= 2,9,16 і 2 біти зсув для інших раундів.

Варто сказати, що така можливість виконання розширення ключа вважається перевагою алгоритмів шифрування, оскільки в цьому випадку розширення ключа можна виконувати паралельно шифруванню і не витрачати пам’ять на зберігання ключів інших раундів, крім поточного.

2.3 Криптостійкість алгоритму DES

З початку використання алгоритму DES криптоаналіики всього світу докладали безліч зусиль з його злому. Фактично DES дав небачений досі поштовх розвитку криптоаналізу. Вийшли сотні праць, присвяченим різним методам криптоаналізу. Можна стверджувати, що саме завдяки DES з'явилися цілі напрями криптоаналізу, такі як:

- Лінійний криптоаналіз
- Диференціальний криптоаналіз
- Криптоаналіз на пов'язаних ключах (В роботі [14] було доказано, що DES невразливий до даного виду атак завдяки тому, що в описаній вище процедурі розширення ключа циклічний зсув виконується на різне число бітів у різних раундах)

Крім того, практично відразу після появи DES було виявлено наступні проблеми із ключами шифрування [12].

Через те, що початковий ключ змінюється при отримані підключення кожного етапу алгоритму, певні початкові ключи є слабкими. Початкове значення розщеплюється на дві половини, кожна з яких зсувається незалежно. Якщо всі біти кожної половини дорівнюють 0 або 1, то для всіх етапів алгоритму використовується один і той же ключ. Це може статися, якщо ключ складається з одних 1, з одних 0, або якщо одна половина ключа складається з одних 1, а інша з одних 0. Чотири слабких ключа показані в шістнадцятковому вигляді в таблиці 2.7. [12]

Таблиця 2.7. Слабкі ключи DES

Значення слабкого ключа с бітами парності (HEX)	Дійсний ключ
0101 0101 0101 0101	0000000 0000000
1F1F 1F1F 0E0E 0E0E	0000000 FFFFFFF
E0E0 E0E0 F1F1 F1F1	FFFFFFF 0000000
FEFE FEFE FEFE FEFE	FFFFFFF FFFFFFF

Крім того, деякі пари ключів при шифруванні переводять відкритий текст на ідентичний шифротекст. Іншими словами, один із ключів пари може розшифрувати повідомлення, зашифровані іншим ключем пари. Це відбувається через метод, який використовується DES для генерації підключів замість 16 різних підключів ці ключи генерують тільки да різних підключа. В алгоритмі кожен із цих підключів використовується вісім разів. Ці ключі, названі напівслабкими ключами. [12]

Таблиця 2.8. Напівслабкі пари ключів DES

01FE 01FE 01FE 01FE і FE01 FE01 FE01 FE01
1FE0 1FE0 0EF1 0EF1 і E01F E01F F10E F10E
01E0 01E0 01F1 01F1 і E001 E001 F101 F101
1FFE 1EEE 0EFE 0EFE і FE1F FE1F FE0E FE0E
011F 011F 010E 010E і 1F01 1F01 0E01 0E01
E0FE E0FE F1FE F1FE і FEE0 FEE0 FEE1 FEE1

48 ключів є «можливо слабкими», їх повний перелік наведено у [12, с 210]. Можливо, слабкі ключи при їх розширенні дають лише 4 різні ключі раундів, кожен з яких використовується при шифруванні по 4 рази.

Однак ці недоліки не є суттєвими, оскільки в програмній або апаратній реалізації DES досить просто заборонити використання проблемних ключів.

Досить багато криптоаналітичних досліджень було присвячено алгоритму DES зі зменшеною кількістю раундів. Вважається що кількість раундів будь-якого багатораундового шифрування можна зменшувати до певних меж, досягаючи оптимального співвідношення швидкість шифрування/криптостійкість. Однак алгоритм DES схильний до істотного зниження криптостійкості при зменшені кількості раундів.

2.3.1 Диференціальний аналіз.

Диференціальній аналіз працює з парами шифротекстів, між відкритими текстами яких існує певна різниця (difference). Метод аналізує еволюцію цих

відмінностей у процесі проходження відкритих текстів через етапи DES при шифруванні одним і тим же ключем. При аналізі алгоритмів текстів може бути визначена по-різному. Для DES різниця відкритих текстів M_1 та M_2 визначається як операція XOR між цими текстами:

$$\Delta = M_1 \oplus M_2.$$

Диференціальний криптоаналіз використовує безліч пар текстів з певною різницею, аналіз яких дозволяє виділити якийсь ключ (або його фрагмент), який є ключем однозначно, або з найбільшою (порівняно з іншими можливими ключам) ймовірністю.

Виконується такий аналіз в такий спосіб. Припустимо, є два відкритих тексти, які на вході в функцію *f* будь-якого раунду алгоритму мають різницю Δb (рис 2.5):

$$\Delta b = b_1 \oplus b_2.$$

Різниця $\Delta be = be_1 \oplus be_2$, тобто різниця після обробки b_1 та b_2 розширювальною перестановкою E, дуже легко визначити, оскілки ми знаємо як виконується ця перестановка (див. таблицю 2.2):

$$\Delta be = E(b_1) \oplus E(b_2) = E(b_1 \oplus b_2) = E(\Delta b).$$

Накладення фрагмента ключа операцією XOR взагалі не змінює різницю, тобто:

$$\Delta bk = \Delta be.$$

Аналогічно перетворенню E, легко обчислити різницю Δbp після перестановки P (см таблицю 2.3):

$$\Delta bp = P(bs_1) \oplus P(bs_2) = P(bs_1 \oplus bs_2) = P(\Delta bs).$$

Таким чином, єдиною операцією із виконуваних функцією *f*, що суттєво впливає на значення різниці, залишається таблична заміна. Значення Δbs залежить не лише від різниці Δbk, а й від конкретних bk_1 та bk_2. Ось тут і проявляється вплив накладенного попередньою операцію ключа шифрування.

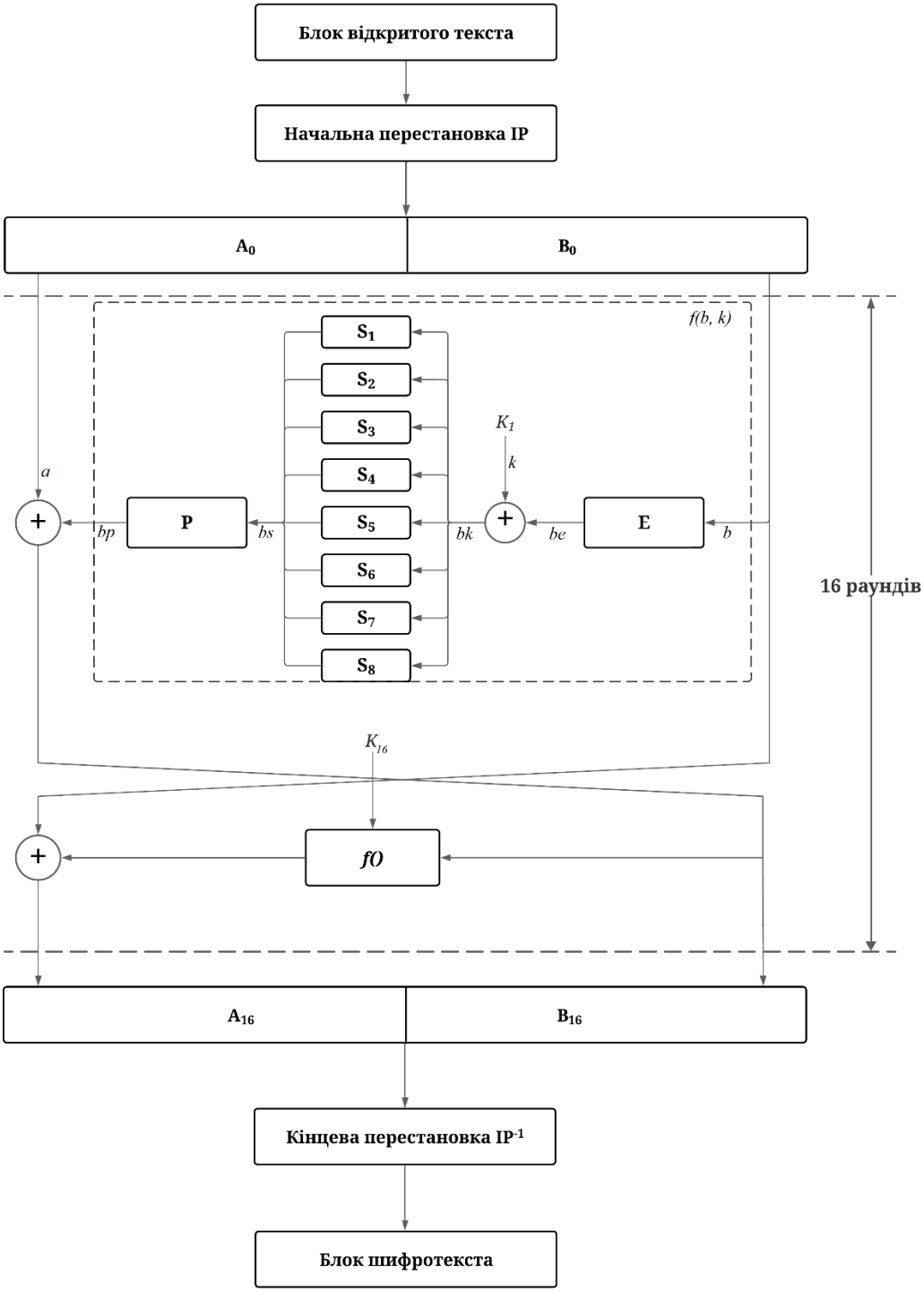

Рис 2.5. Розгорнута структура алгоритму DES

Як було зазначено вище, таблиці замін S змінюють 6-бітове вхідне значення на 4 бітне. Це означає, що будь-якій вхідній різниці $\Delta bk,n$ (де n – номер таблиці S) відповідає 2^6= 64 можливих пар вхідних значень (позначим їх як $\Delta bk_1,n$ та $\Delta bk_2,n$). Відповідно, будь-якій різниці $\Delta bs,n$ відповідає 2^4 = 16 можливих пар bs_1,n та bs_2,n.

Диференціальний криптоаналіз алгоритму DES експлуатує той факт, що, по-перше, кожне конкретне значення $\Delta bk,n$ призводить не до всіх можливих 16 значень $\Delta bs,n$, а по-друге, дані значення $\Delta bs,n$ мають дуже різну ймовірність .Біхам і Шамір як приклад розглядають таблицю S_1 та її вхідну різницю $\Delta bk,1 = 34$ (тут і далі значення різниць вказується у шістнадцятковій системі). Можливі значення $\Delta bs,1$ та їх ймовірності (в кількості значення пар $bk_1,1$ *та* $bk_2,1$ з 64 можливих, які призводять до значення $\Delta bs,1$) наведені у таблиці 2.9 [13].

Таблиця 2.9. Можливі вхідні дані для $\Delta bk,1 = 34$

Значення $\Delta bs,1$	Можливі вхідні дані	Ймовірність
1	03, 0F, 1E, 1F, 2A, 2B, 37, 3B	8/64
2	04, 05, 0E, 11, 12, 14, 1A, 1B, 20, 25, 26, 2E, 2F, 30, 31, 3A	16/64
3	01, 02, 15, 21, 35, 36	6/64
4	13, 27	2/64
7	00, 08, 0D, 17, 18, 1D, 23, 29, 2C, 34, 39, 3C	12/64
8	09, 0C, 19, 2D, 38, 3D	6/64
D	06, 10, 16, 1C, 22, 24, 28, 32	8/64
F	07, 0A, 0B, 33, 3E, 3F	6/64
Всього		64

Як за допомогою всього цього можна визначити ключ, розглянемо найпростіший приклад. Припустимо, що у розпорядженні криптоанлітика є пара текстів з $\Delta bk,1 = 34$. Крім того, у криптоаналітика є відповідні шифротексти (знову припустимо, що для їх зашифрування використовувався одноразундовий DES) с $\Delta bs,1$=4. Вхідні значення S_1 за таких умов це значення 13 і 27 ($bk_1,1$ =13*, а* $bk_2,1$ = 27, або навпаки). Виходить, що криптоаналітик знає значення be_1 та be_2 (оскільки йому відомі відкриті тексти), а також два варіанти значень $bk_1,1$ і $bk_2,1$. В результаті криптоаналітик може найпростішою операцію XOR обчислити 2 можливі варіанти перших шести бітів K_1. Вибрати з двох варіантів правильну допоможе друга пара

відкритих текстів, якщо така у нього є. Навіть якщо такої пари немає, криптоаналітик суттєво звузив галузь можливих значень ключа: у 25 разів. Зрозуміло, що однораундовий DES не єреальним об'єктом для криптоаналізу. Для розкриття алгоритмів з великою кількістю раундів використовують характеристики – такі різниці відкритих текстів, які з високою ймовірністю викликають певні різниці в шифротекстах. Як приклад розглянемо наступну трьохраундову характеристику алгоритму DES (рис. 2.6) [13]:

1. У першому раунді на вхід функції подаються два субблоки з різницею Δb = 60000000. Різниця підібрана таким чином, щоб після перестановки E (накладення фрагмента ключа, як було сказано вище, не впливає на різницю) вона була нульовою на вході всіх таблиць замін, крім S_1, на вході якої різниця $\Delta bk,1$ = 0C. Ця різниця забезпечує вихідну різницю $\Delta bs,1$=0E з ймовірністю 14/64, яка після перестановки P (з урахуванням нульової різниці на виході інших таблиць) призводить до різниці Δbp = 00808200. Як видно, ця різниця збігається з різницею необроблених субблоків, тобто Δa; в результаті накладення результату функції на необроблений субблок виходить нульова різниця на вході другого раунду.
2. Нульова різниця Δb на вході функції у другому раунді дає також нульову різницю на виході. Отже (рис. 2.6), різниця на вході функції *f* до третього раунду повністю еквівалентна такій у першому раунді.
3. Виходить, що різниці третього раунду та їх ймовірності аналогічні першому раунду та їх ймовірності аналогічні першому раунду. Таким чином, дана характеристика забезпечує після трьох раундів вихідну різницю Δ = 0080820060000000 (рівну вхідній різниці) з такою ймовірінстю:

$$p = \frac{14}{64} * 1 * \frac{14}{64} \approx 0{,}478.$$

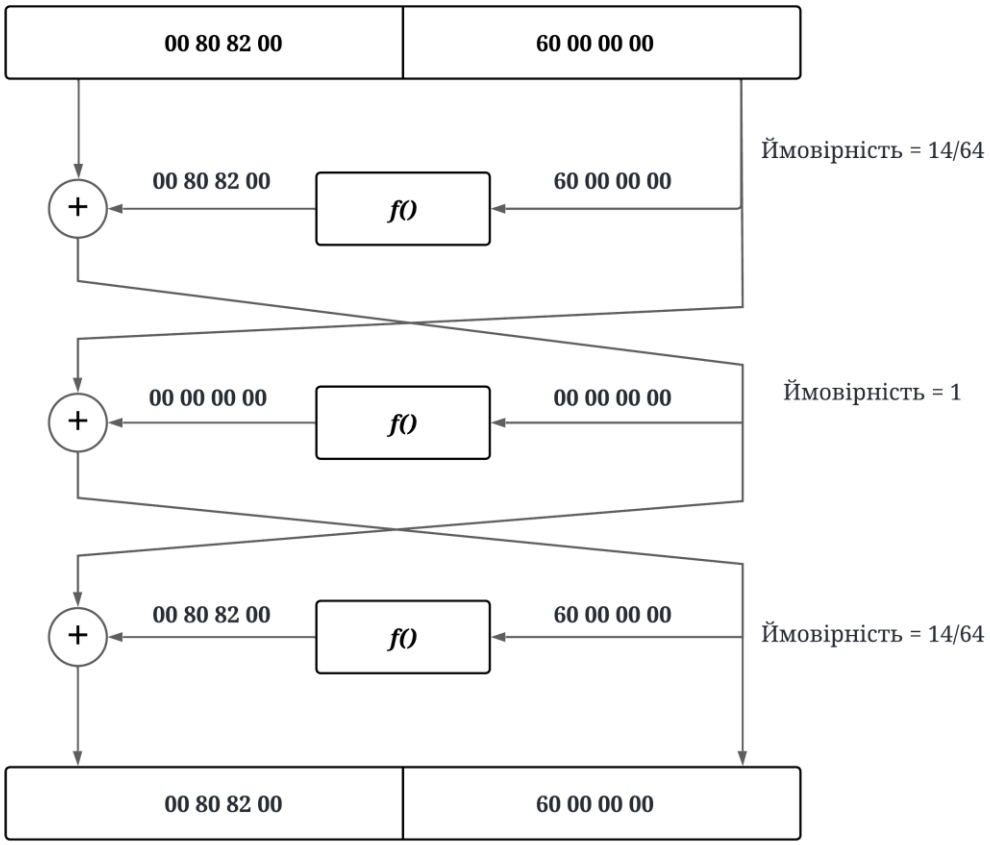

Рисунок 2.6. Трьохраундова характеристика

Характеристики використовуються наступним чином (на прикладі трираундового DES та наведеної вище характеристики):

1. Генерується необхідна кількість вибраних відкритих текстів та відповідних шифротекстів.
2. Формується таблиця, яка містить можливі варіанти шуканого фрагмента ключа шифрування (Як було показано вище для однораундового DES). Вірне значення має повторюватися для кожного аналізованого тексту, оскільки саме воно використовувалося для його зашифрування.

Отже, диференціальний розтин n-етапного DES дає 48 бітовий підключ, що використовується на етапі n, а 8 бітів ключа, що залишалися, виходять за допомогою грубого злому.

Але ряд помітних проблем все ж таки залишається. По-перше, поки ви не перейдете через деяки граничне значення, ймовірність успіху зневажливо мала. Тобто, поки не буде накопичена достатня кількість даних виділити правильне підключення з шуму неможливо. Крім того, такий розтин не практичен. Для збереження ймовірностей 2^{48} можливих ключів необхідно використовувати лічильники, і до того ж для розтину потрібно занадто багато даних.

В таблиці 2.10 проведено огляд кращих диференціальних розтинів DES з різною кількістю етапів. Перший стовпець містить кількість етапів. Елементи настпуних двох стовпців є кількість. Вибраних або відомих відкритих текстів, які повинні бути перевірені для розтину, а четвертий містить кількість дійсно проаналізованих відкритих текстів. В останньому стовпці наведено складність аналізу після виявлення необхідної пари. [12]

Таблиця 2.10. Розтин за допомогою диференціального криптоаналізу

Кількість етапів	Вибрані відкриті тексти	Відомі відкрити тексти	Проаналізовані відкриті тексти	Складність аналіза
8	2^{14}	2^{38}	4	29
9	2^{24}	2^{44}	2	2^{32}
10	2^{24}	2^{43}	2^{14}	2^{15}
11	2^{31}	2^{47}	2	2^{32}
12	2^{31}	2^{47}	2^{21}	2^{21}
13	2^{39}	2^{52}	2	2^{32}
14	2^{39}	2^{51}	2^{29}	2^{29}
15	2^{47}	2^{56}	27	2^{37}
16	2^{47}	2^{55}	2^{36}	2^{37}

Потрібно відзначити низку важливих моментів. По-перше, цей розтин значною мірою теоретичний. Великі вимоги до часу та обсяг даних, необхідних для виконання розтину за допомогою диференційного криптоаналізу, знаходяться майже для всіх поза межами досяжності. Щоб отримати потрібні дані для виконання такого

розкриття повного DES, доведеться майже три роки шифрувати потік вибраних шифротекстів 1.5 Мегабіт/с. По-друге, це насамперед розтин з вибраним відкритим текстом. Воно може бути перетворено на розтин з відомим відкритим текстом, але вам доведеться переглянути всі пари "відкритий текст/шифротекст" у пошуках корисних. У разі повного 16-етапного DES це робить трохи менш ефективним порівняно з грубою силою (розкриття диференціальнним криптоаналізом вимагає $2^{55.1}$ (у найкращому випадку 2^{37}) операцій, а розкриття грубою силою 2^{55}). Таким чином, правильно реалізований DES зберігає стійкість до диференціального криптоаналізу [12].

2.3.2 Лінійний криптоаналіз

Лінійний криптоаналіз винайшов японський криптолог Міцуру Мацуї (Mitsuru Matsui). Сенс лінійного криптоаналізу полягає у знаходженні співвідношень наступного виду:

$$P_{i1} \oplus P_{i2} \oplus ... \oplus P_{ia} \oplus C_{j1} \oplus C_{j2} \oplus ... \oplus C_{jb} = K_{k1} \oplus K_{k2} \oplus ... \oplus K_{kr}, \quad (2.1)$$

де P_n, C_n, K_n – n-ні біти відкритого тексту, шифротексту та ключа відповідно.

Для довільно вибраних бітів відкритого тексту, шифротексту та ключа ймовірність P справедливості такого співвідношення становить близько 1/2. В тому випадку, криптоаналітику вдається знайти такі біти, при яких ймовірність P помітно відрізняється від 1/2 цим співвідношенням можна скористатися для розкриття алгориму.

Так само, як і в диференціальному криптоаналізі, спочатку криптоаналітик знаходить якесь одноразундове співвідношення, потім намагається поширити його на кількість більшу кількість раундів, в результаті знаходить співвідношення для повнораундового варіанта алгоритму, що аналізується. Варто відмітити що, на відміну від диференціального криптоаналізу, існують алгоритми пошуку корисних співвідношень.

На рис 2.7 представлено найбільш ефективне одноразундове співвідношення для алгоритму DES. Дане співвідношення використовує ту властивість таблиці S_5,

що другий вхідний біт таблиці дорівнює результату застосування операції XOR над усіма чотирма вихідними бітами з ймовірністю 3/16 (тобто зсув 5/16 щодо ймовірності 1/2), тобто (застосовуючи нотацію, використану раніше при описі диференціалного криптоаналізу):

$$(bk_5)_2 = (bs_5)_1 \oplus (bs_5)_2 \oplus (bs_5)_3 \oplus (bs_5)_4,$$

що еквівалентно:

$$bk_{26} = bs_{17} \oplus bs_{18} \oplus bs_{19} \oplus bs_{20}. \quad (2.1)$$

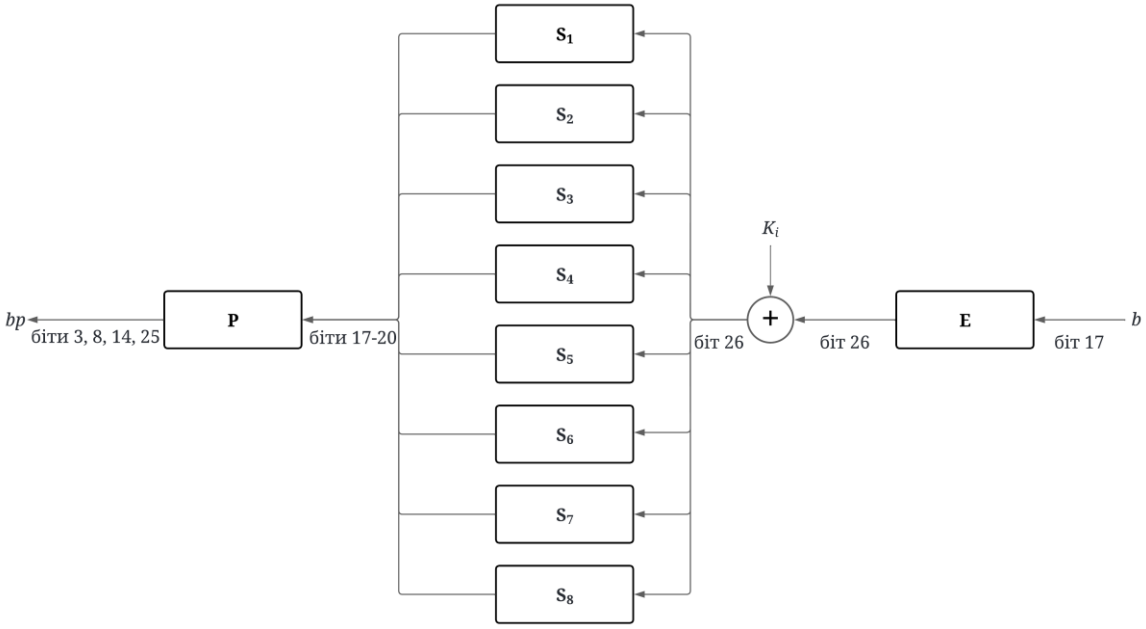

Рисунок 2.7. Найбільш ефективне одноразундове співвідношення для алгоритму DES

Подальше розширення цієї властивості таке:

1. Як показано на рисунку 2.7 співвідношення (2.1) поширюється на цілий раунд алгоритму; внаслідок чого (з урахуванням перестановок) виходить наступне одноразундове співвідношення:

   $$b_{17} \oplus bp_3 \oplus bp_8 \oplus bp_{14} \oplus bp_{25} = (K_i)_{26},$$

   де i – номер раунду.
2. Дуже схоже на диференціальний криптоаналіз співвідношення розповсюджується на кілька раундів. Приклад такого співвідношення показано

на рис 2.8 ((2.1) справедливе для тих бітів, номери яких вказані на рисунку); його ймовірність зміщена щодо 1/2 на 0.0061 [13]. А для повнораундового DES відоме співвідношення, що виконується з ймовірністю $1/2 + 2^{-24}$ [5].

3. З використанням максимально ефективного співвідношення виконується аналіз наявних пар «відкритий текст – шифртекст» з метою знайти найбільш ймовірні значення певних бітів ключа шифрування.

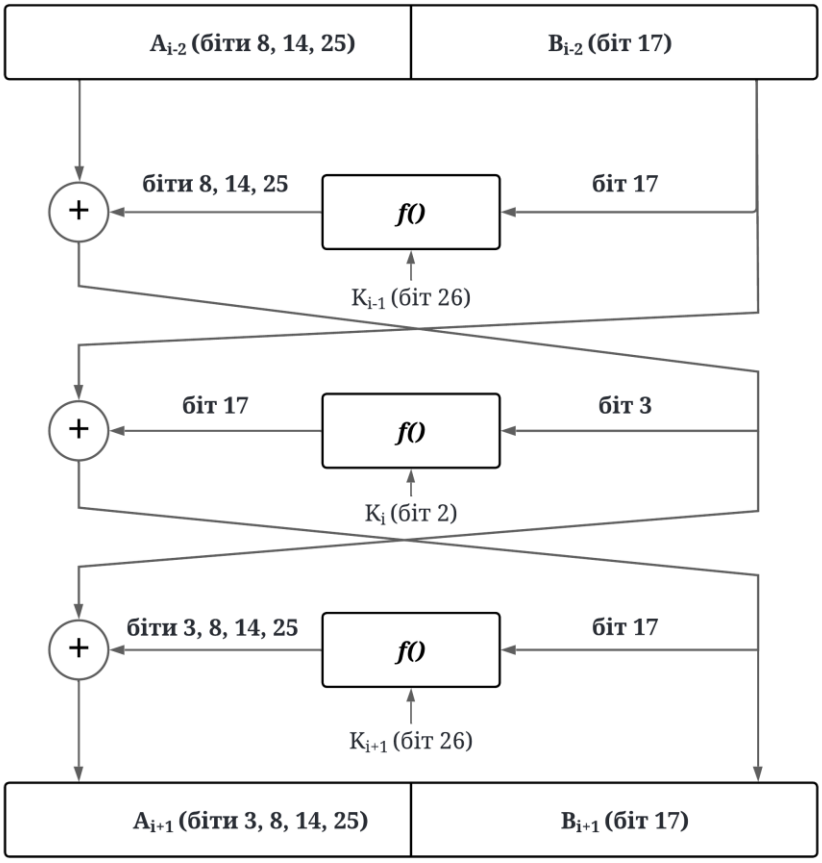

Рисунок 2.8. Приклад трьохраундового співвідношення

Базовий розтин має використовувати найкраще лінійне наближення для 16-етапного DES. Для нього потрібно 2^{47} відомих відкритих блоків, а результатом відкриття є 1 біт ключа. Це не дуже корисно. Якщо поміняти місцями відкритий текст і шифротекст і використати дешифрування разом із шифруванням можна отримаи 2 біти. Це ще не дуже корисно.

Існує ряд тонкощів. Використовуючи 14 етапне лінійне наближення для етапів з 2 по 15. Спробуємо вгадати 6 бітів підключа для S_5 першого та останнього етапів (всього, таким чином, 12 бітів ключа). Для ефективності виконуємо лінійний криптоаналіз паралельно 2^{12} разів і вибираємо правильний варіант, грунтуючись на ймовірностях. Це розкриває 12 бітів, а помінявши місцями відкритий текст та шифротекст ми отримаємо ще 13 бітів. Для отримання 30 бітів, що залишилися, використовуйте, вичерпний пошук [12].

Лінійний криптоаналіз сильно залежить від структури S-блоків, виявилося, що S-блоки DES не оптимізовані проти такого способу розтину. Справді, зміщення в S-блоках, вибраних для DES, знаходиться між 9 і 16 відсотками, що не забезпечує надійного захисту проти лінійного криптоаналізу.

РОЗДІЛ 3. АНАЛІЗ І РОЗРОБКА ТЕХНІЧНИХ ВИМОГ ДО СИСТЕМИ ВЗАЄМОДІЇ З ПЕРИФЕРІЙНИМИ ПРИСТРОЯМИ ПРИ ОБРОБЦІ ДАНИХ У СТАНДАРТІ DES.

3.1 Визначення вимог щодо інтерфейсів зв'язку з периферійними пристроями.

3.1.1 Сумісність інтерфейсів

Коли йдеться про сумісність інтерфейсів зв'язку з периферійними пристроями в контексті обробки даних за стандартом DES, важливо враховувати як апаратні, а й програмні аспекти.

З апаратної точки зору інтерфейси зв'язку повинні бути сумісні з використовуваними периферійними пристроями, що включає відповідність фізичним стандартам (наприклад, USB, Ethernet, Bluetooth).

З програмної точки зору важливо, щоб інтерфейси підтримували необхідні протоколи зв'язку передачі даних між системою, що реалізує стандарт DES, і периферійними пристроями. Це може містити стандартні протоколи передачі даних (наприклад, TCP/IP для мережних пристроїв), а також спеціалізовані протоколи, необхідні для взаємодії з конкретними типами пристроїв.

Крім того, сумісність інтерфейсів також може включати можливість роботи з різними операційними системами, що забезпечує універсальність і гнучкість у використанні системи в різних середовищах.

Таким чином, сумісність інтерфейсів зв'язку з периферійними пристроями грає ключову роль у забезпеченні ефективної та безпечної взаємодії між системою та зовнішніми пристроями при обробці даних за стандартом DES.

3.1.2 Розробка алгоритму передачі даних по локальній мережі:

1) Вибір мережевих протоколів: Розробнику слід визначити найбільш відповідні мережеві протоколи передачі даних у локальної мережі. Це може включати протоколи TCP/IP для забезпечення надійної та стійкої передачі даних.

2) Проектування архітектури мережі: необхідно спроектувати архітектуру мережі, визначити роль і конфігурацію кожного вузла мережі, і навіть налаштувати мережне устаткування забезпечення ефективної передачі.

3) Реалізація мережевої взаємодії: Розробник повинен реалізувати механізми передачі даних між комп'ютерами в локальній мережі. Це може включати створення сервера та клієнта для встановлення з'єднання та обміну даними.

4) Забезпечення безпеки мережі: важливим аспектом розробки передачі по локальній мережі є забезпечення безпеки. Це включає шифрування даних за допомогою алгоритму DES, а також реалізацію механізмів автентифікації та контролю доступу для запобігання несанкціонованого доступу до мережі та даних.

5) Тестування та налагодження: після реалізації передачі даних необхідно провести тестування та налагодження, щоб переконатися в коректності роботи системи та її здатності ефективно передавати дані по локальній мережі.

3.1.3 Розробки інтерфейсу користувача

Аналіз потреб користувачів: перед початком розробки необхідно провести аналіз потреб та очікування користувачів від інтерфейсу. Це дозволить визначити функціональні та ергономічні вимоги до інтерфейсу.

Проектування інтерфейсу: на основі аналізу потреб користувачів розробляється дизайн інтерфейсу, який має бути інтуїтивно зрозумілим та зручним у використанні. Важливо враховувати стандарти дизайну інтерфейсів і принципи зручності та ефективності.

Вибір елементів керування: розробнику слід вибрати відповідні елементи керування для взаємодії користувача з програмою. Це можуть включати кнопки,

поля введення, списки, що випадають, та інші елементи, що відповідають функціональності програми.

Реалізація інтерфейсу: після проектування інтерфейсу необхідно розпочати його реалізації. Це включає написання коду для створення графічних елементів та їх взаємодії з іншими компонентами програми.

Тестування інтерфейсу користувача: після завершення реалізації інтерфейсу потрібно провести тестування його роботи з урахуванням різних сценаріїв використання. Важливо переконатися, що інтерфейс функціонує коректна та відповідає потребам користувачів.

3.1.5 Забезпечення універсальності та довгострокової ефективності програми

Адаптація до різних типів пристроїв: Інтерфейси повинні бути спроектовані таким чином, щоб бути сумісними з різними типами периферійних пристроїв. Інтерфейс повинен підтримувати роботу з відповідними пристроями та їх специфічними характеристиками.

Підтримка різних форматів даних: Інтерфейси повинні забезпечувати можливість передачі та обробки різних форматів даних, залежно від вимог конкретних периферійних пристроїв. Наприклад, якщо програма обробляє зображення, інтерфейс повинен бути гнучким для роботи з різними типами зображень.

Можливість розширення функціональності: інтерфейс має бути спроектований з урахуванням можливості додавання нової функціональності у майбутньому. Це може включати додавання нових елементів керування, модулів або інтеграцію з новими периферійними пристроями без необхідності внесення істотних змін до основного коду програми.

Підтримка стандартів та протоколів: інтерфейси повинні бути гнучкими та розширюваними з точки зору підтримки різних стандартів та протоколів зв'язку.

Наприклад, якщо програма працює з мережними пристроями, інтерфейс повинен бути здатний адаптуватися до змін у мережевих стандартах та протоколах.

Документація та підтримка: важливо надати користувачам документацію та підтримку щодо розширення та гнучкості інтерфейсу. Це може включати посібники з розробки додаткових модулів або інструкції з налаштування інтерфейсу для роботи з новими пристроями.

3.2 Специфікації протоколів обміну даними між системою та периферійними пристроями.

Специфікації протоколів обміну даними між системою та периферійними пристроями визначають набір правил і форматів, які система та пристрої повинні дотримуватись при передачі інформації один одному.

Формат повідомлень. Специфікація протоколу визначає формати повідомлень, які можуть надсилатися між системою та пристроями. Це може включати структуру заголовків, полів даних і контрольних сум.

Команди та відповіді. протокол визначає список команд, які система може надсилати пристрою, а також очікувані відповіді пристроїв на ці команди. Наприклад, обмін повідомленнями між клієнтами з командами та відповідями може виглядати так:

1) Надсилання повідомлення від клієнта А до клієнта В:
   - Клієнт А формує повідомлення для надсилання клієнту В і надсилає його на сервер через своє з'єднання TCP.
   - Сервер отримує повідомлення від клієнта А і перенаправляє його клієнту В через його з'єднання TCP.
2) Отримання повідомлення клієнтом В:
   - Клієнт В отримує повідомлення від сервера через TCP-з'єднання.
3) Відповідь на отримане повідомлення:
   - Клієнт В формує повідомлення у відповідь і відправляє його на сервер через своє TCP-з'єднання.

- Сервер отримує відповідь від клієнта B і передає його клієнту A через його з'єднання TCP.

4) Підтвердження отримання повідомлення:

- Після того, як клієнт A отримав відповідь від клієнта B, він може надіслати підтвердження серверу через своє з'єднання TCP.
- Сервер може підтвердити повідомлення клієнту A.

Обробка помилок: специфікація протоколу також визначає, як система та пристрої повинні обробляти помилки під час передачі даних. Це може включати механізми виявлення помилок і методи відновлення після помилок.

Управління з'єднанням: деякі протоколи можуть включати механізми управління з'єднанням, такі як установка і розрив з'єднання між системою і пристроями.

Захист даних: специфікація протоколу може також включати заходи захисту даних, такі як шифрування та автентифікація, щоб забезпечити безпеку інформації, що передається.

3.3 Гарантії безпеки передачі даних між системою та периферійними пристроями під час використання стандарту DES.

DES забезпечує шифрування даних, що робить їх незрозумілими для сторонніх осіб, які мають ключа розшифровки. Це значно знижує ризик перехоплення та прослуховування конфіденційної інформації під час передачі через мережу.

Для використання DES потрібен обмін секретними ключами між системою та периферійними пристроями. Це забезпечує додатковий рівень безпеки, оскільки тільки учасники, які знають ключ, можуть розшифрувати передані дані.

DES як шифрує дані, а й забезпечує їх цілісність. Це означає, що дані не можуть бути змінені або пошкоджені під час передачі без виявлення.

Додаткові механізми автентифікації можуть бути реалізовані для підтвердження справжності та ідентифікації пристроїв та систем, забезпечуючи додатковий рівень безпеки під час передачі даних.

DES, хоч і має свої обмеження, залишається порівняно стійким до різних атак, особливо при правильному використанні та налаштуванні.

Крім того, стандарт DES надає деякі механізми захисту від певних видів атак, таких як атаки перебором ключа. DES використовує сильну криптографічну функцію, що ускладнює спроби злому шляхом перебору всіх можливих ключів. В даний час, з урахуванням зростання обчислювальної потужності комп'ютерів, стандарт DES стає вразливим для більш передових атак, таких як атаки методом грубої сили.

Хоча DES має деякі недоліки і вважається застарілим з точки зору криптографічної безпеки, він все ще може забезпечити прийнятний рівень захисту даних при передачі між системою та периферійними пристроями деяких сценаріях використання.

Тим не менш, із застосуванням адекватних запобіжних заходів, таких як використання досить довгих ключів і регулярне оновлення ключових матеріалів, DES може залишитися ефективним засобом забезпечення безпеки даних при обміні інформацією між системою і периферійними пристроями. Крім того, DES все ще широко використовується в деяких галузях, де потрібне стандартизоване шифрування даних, наприклад, банківській сфері або військової промисловості.

Крім цього, стандарт DES може бути комбінований з іншими механізмами захисту, такими як автентифікація та контроль доступу для створення більш надійних систем обміну даними. В цілому, хоча стандарт DES має свої обмеження та послаблення, він все ще залишається важливим інструментом для забезпечення безпеки даних у мережевих та системних програмах, особливо при правильному використанні та налаштуванні.

3.4 Проектування інтерфейсів взаємодії з периферійними пристроями, сумісними з DES

Для реалізації TCP/IP клієнт-серверного чату з використанням алгоритмів DES-шифрування необхідно розробити інтерфейси взаємодії з периферійними

пристроями, сумісними з DES. Ці інтерфейси повинні забезпечувати зручну та безпечну взаємодію користувачів із системою чату.

3.4.1 Вимоги до інтерфейсів

Зручність використання: Інтерфейси мають бути простими та зрозумілими для користувачів.

Безпека: Інтерфейси повинні забезпечувати конфіденційність та цілісність даних, що передаються між клієнтом та сервером.

Сумісність: Інтерфейси мають бути сумісні з DES-шифруванням.

3.4.2 Варіанти реалізації

Програмний інтерфейс: Інтерфейси можуть бути реалізовані програмно, використовуючи алгоритм блочного кодування DES-шифрування. Це дозволяє використовувати різні мови програмування та платформи.

Інтерфейси можуть бути реалізовані апаратно, використовуючи спеціальні чіпи DES-шифрування. Це забезпечує більш високу продуктивність та безпеку.

3.4.3 Вибір варіанта реалізації

Вибір варіантів реалізації залежить від конкретних вимог до системи чату. Якщо потрібна висока продуктивність та безпека, доцільно використовувати апаратний інтерфейс. Якщо ж потрібна простота та гнучкість, доцільно використовувати програмний інтерфейс.

3.4.4 Опис інтерфейсів

Інтерфейс клієнта:

Поле введення повідомлення: Користувач вводить повідомлення у текстове поле.

Кнопка надсилання: Користувач натискає кнопку надсилання, щоб надіслати повідомлення серверу.

Вікно чату: Вікно чату відображає повідомлення, надіслані та отримані від сервера.

Інтерфейс сервера:

Мережевий приймач: Сервер приймає повідомлення від клієнтів.

Шифратор DES: Сервер шифрує повідомлення за допомогою ключа DES.

Обробник повідомлень: Сервер обробляє повідомлення, наприклад, зберігає їх в історії чату.

Дешифратор DES: Сервер дешифрує повідомлення за допомогою ключа DES.

Відправник мережі: Сервер надсилає повідомлення клієнтам.

3.4.5 Схема взаємодії

- Користувач вводить повідомлення та натискає кнопку надсилання.
- Клієнтський інтерфейс шифрує повідомлення за допомогою DES-ключа.
- Зашифроване повідомлення надсилається серверу.
- Серверний інтерфейс приймає зашифроване повідомлення.
- Серверний інтерфейс дешифрує повідомлення за допомогою DES-ключа.
- Дешифроване повідомлення обробляється сервером.
- Сервер шифрує повідомлення за допомогою DES-ключа.
- Зашифроване повідомлення надсилається клієнту.
- Клієнтський інтерфейс приймає зашифроване повідомлення.
- Клієнтський інтерфейс дешифрує повідомлення за допомогою DES-ключа.
- Дешифроване повідомлення відображається у вікні чату.

3.5 Реалізація механізмів шифрування та дешифрування даних, що передаються між системою та периферійними пристроями

Для забезпечення конфіденційності та цілісності даних, що передаються між системою та периферійними пристроями, необхідно використовувати надійні методи шифрування. Алгоритм DES - один із найпоширеніших методів шифрування, який може бути використаний для цієї мети.

Механізми шифрування та дешифрування даних з використанням алгоритму DES можуть бути реалізовані як програмно, так і апаратно.

Ретельно вивчивши опис алгоритму DES, включаючи етапи шифрування та дешифрування, структуру блоків даних та таблиці заміни (S-Box). Вручну реалізуємо функції шифрування та дешифрування DES, використовуючи мову програмування C# або інші інструменти.

Реалізуємо сам алгоритм DES. Цей алгоритм заснований на серії підстановок та перестановок, які застосовуються до блоків даних. Він складається з декількох раундів шифрування, що включають операції підстановки і перестановки бітів, а також операції над ключом шифрування.

Ключі зберігаються у захищеному місці та доступні лише авторизованим особам.

Після генерації ключа шифрування слід розпочати шифрування даних. Дані, що передаються між системою та периферійними пристроями, будуть розбиті на блоки та зашифровані з використанням алгоритму DES та згенерованого ключа.

На стороні одержувача дані будуть дешифровані за допомогою того ж ключа шифрування та алгоритму DES. Це дозволить одержувачу відновити вихідні дані із зашифрованих блоків.

Складнощі реалізації:

Трудомісткість: ручна реалізація DES без бібліотек може бути трудомісткою та вимагати значних витрат часу.

Помилки: при ручній реалізації висока ймовірність помилок, які можуть призвести до уразливостей у системі безпеки.

Складність супроводу: підтримка та модифікація коду, написаного вручну, може бути складнішим завданням, ніж при використанні бібліотек.

Розбиваючи завдання більш дрібні подзадачи і реалізуємо їх поетапно, ретельно тестуючи кожен етап.

Документуємо код та алгоритми, щоб полегшити його розуміння та модифікацію в майбутньому.

РОЗДІЛ 4. РОЗРОБКА ДОДАТКУ ДЛЯ ПЕРЕДАЧІ ДАНИХ ПО ЗАШИФРОВАНОМУ КАНАЛУ ЗВ'ЯЗКУ

4.1 Архітектура програми передачі даних по зашифрованому каналу зв'язку

Під технологією розробки ПЗ розуміють оптимальний спосіб ведення розробки, який за певних умов забезпечить отримання кінцевого продукту із заздалегідь заданими властивостями.

У ході дослідження було структуровано існуючий матеріал і на його основі створено технологію розробки клієнт-серверних додатків.

Розробка здійснюється за наступним алгоритмом:

1. Вибір архітектури клієнт-сервера (технічна частина). На початку розробки необхідно вибрати між дворівневою та багаторівневою архітектурою. Вибір у цьому випадку залежить від кількості користувачів, які працюватимуть у даній системі, вартості обладнання та подальшого обслуговування системи (наприклад, доопрацювання функціоналу). Детальний опис цих архітектур описано у першому розділі.

2. Вибір мови програмування (програмна частина). Клієнт-сервер можна написати різними мовами. У різних мов присутні свої особливості реалізації мережевих завдань.

Вибір мови програмування для реалізації програмної частини клієнт-серверного чату залежить від низки факторів, таких як цільова платформа, існуючі навички команди розробників, вимоги до продуктивності, доступні інструменти та бібліотеки, а також особисті переваги. Прикладом мови програмування, яка може бути використана для створення клієнт-серверного чату, є C#.

C# - це потужна та зручна мова програмування, яка широко використовується для розробки додатків під платформу Windows. Він має великою стандартною бібліотекою класів, яка надає різні інструменти для роботи з мережею,

багатопоточністю, а також інтерфейсом користувача. Ось як можна було б використовувати C# для реалізації програмної частини клієнт-серверного чату:

Створення сервера: на серверній стороні можна використовувати C# для створення програми, яка слухає певний порт та приймає підключення від клієнтів. У C# існують засоби створення мережевих додатків, такі як класи TcpListener і TcpClient, які можуть бути використані для встановлення і управління з'єднаннями між сервером і клієнтами.

Створення клієнта: на клієнтській стороні можна використовувати C# для створення програми, яка встановлює з'єднання з сервером і обмінюється повідомленнями. Клієнтська програма може бути реалізована за допомогою класу TcpClient для встановлення з'єднання з сервером та надсилання повідомлень.

Робота з повідомленнями: для обміну повідомленнями між клієнтами та сервером можна використовувати простий протокол, такий як текстовий протокол, де повідомлення можуть бути подані у вигляді рядків. C# надає засоби для читання та запису даних через мережеві потоки, що дозволяє надсилати та отримувати повідомлення між клієнтом та сервером.

Інтерфейс користувача: для створення інтерфейсу користувача клієнта можна використовувати Windows Forms або WPF, які надають графічні елементи управління для створення зручного і інтуїтивно зрозумілого інтерфейсу.

3. Створення робочого прототипу.

Процес створення прототипу зазвичай складається з кроків:

1) Налаштування сервера:

Сервер прослуховуватиме певний порт і прийматиме підключення від клієнтів.

При з'єднанні з клієнтом сервер генерує випадковий ключ для шифрування DES і відправляє його клієнту.

Потім сервер приймає зашифровані повідомлення від клієнта, розшифровує їх за допомогою DES ключа і виводить їх на екран.

2) Налаштування клієнта:

Клієнт встановлює з'єднання з сервером та отримує ключ шифрування DES від сервера.

Після отримання ключа клієнт шифрує свої повідомлення з використанням DES перед відправкою на сервер.

Клієнт також розшифровує повідомлення з сервера за допомогою того ж ключа DES.

3) Використання алгоритму DES:

Для шифрування та розшифрування повідомлень на стороні сервера та клієнта використовується алгоритм DES.

Кожен байт даних, що надсилаються або приймаються через мережу, шифрується на стороні відправника та розшифровується на стороні одержувача з використанням того самого ключа DES.ч

4.2 Розробка клієнт-серверної програми

У цій частині розглянемо, структурну модель програми, розробку інтерфейсу користувача, а також як взаємодіє клієнт та сервер.

4.2.1 Вибір середовища та створення структурної моделі

Для створення клієнт-серверної програми було обрано дворівневу архітектуру та об'єктно-орієнтовану мову програмування C# т.к. він є відмінним вибором для розробки клієнт-серверних програм завдяки своїй універсальності, продуктивності та широкому набору бібліотек.

Середовище розробки (IDE) Visual Studio - це популярна IDE для C#, яка пропонує широкий спектр функцій для розробки, тестування та налагодження коду.

Фреймворки та бібліотеки:

.NET Framework: Забезпечує базовий набір класів та бібліотек для розробки C#-додатків.

Створення структурної моделі перед початком проекту має вирішальне значення. Побудова структурної моделі допомагає програмісту повністю зрозуміти вимоги проекту, включаючи функціональні можливості, основні компоненти системи та взаємозв'язки між ними. На основі структурної моделі можна визначити обсяг роботи, необхідні ресурси розподілити їх ефективно задля досягнення поставленої мети. Створення структурної моделі дозволяє виявити потенційні

проблеми чи помилки в архітектурі проекту ще до початку його реалізації, що дозволяє запобігти непорозумінням та спрощує подальшу розробку. Правильно розроблена структурна модель дозволяє оптимізувати процес розробки, зменшуючи час, що витрачається на пошук та виправлення помилок надалі, а також підвищуючи ефективність команди.

Структурна модель будувалась так що навіть після релізу можна відносно легко додавати новий функціонал. Модель зображена рисунку 4.1:

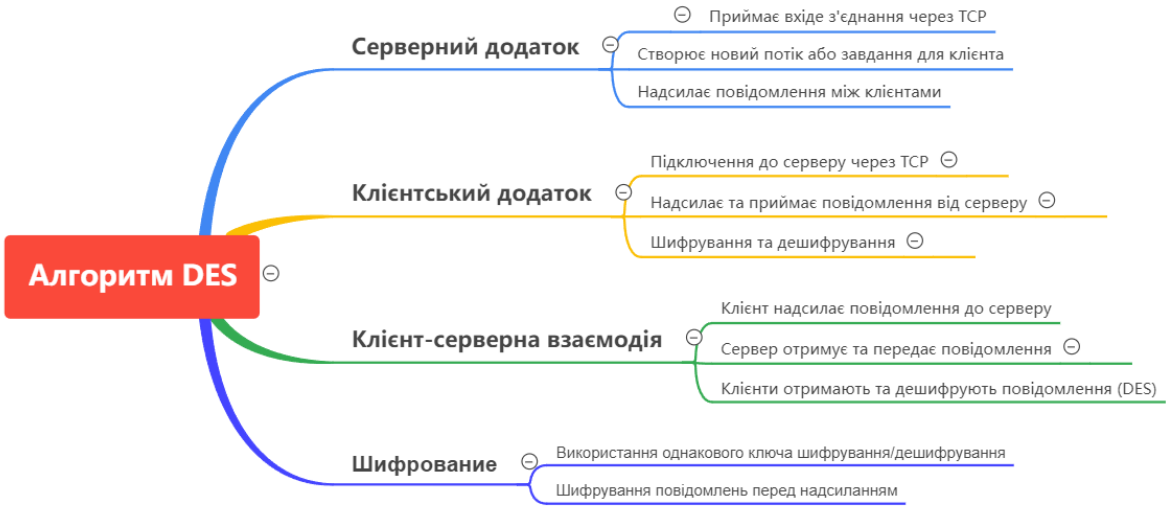

Рисунок 4.1. Структурна модель клієнт-серверного чату

4.2.2 Розробка інтерфейсу користувача для взаємодії з програмою

Цей крок включає розробку інтерфейсу користувача (UI) для взаємодії з клієнт-серверним чатом, використовуючи Windows Forms мовою програмування C#.

У IDE Visual Studio створюємо новий проект Windows Forms Application мовою C#.

Додаємо у головне вікно програми (форму) за допомогою Windows Forms Designer.

Розміщуємо на формі елементи керування, такі як текстове поле для введення повідомлень, кнопку "Надіслати", область для відображення чату і т. д. Використовуємо Toolbox для додавання елементів управління на форму, таких як RichTextBox для відображення повідомлень, TextBox для введення повідомлень,

ListBox для виведення підключених користувачів та 3 елементи Button для надсилання повідомлень та підключення та відключення до сервера.

Розміщуємо елементи управління на формі згідно з розробленим макетом інтерфейсу на рисунку 4.2.

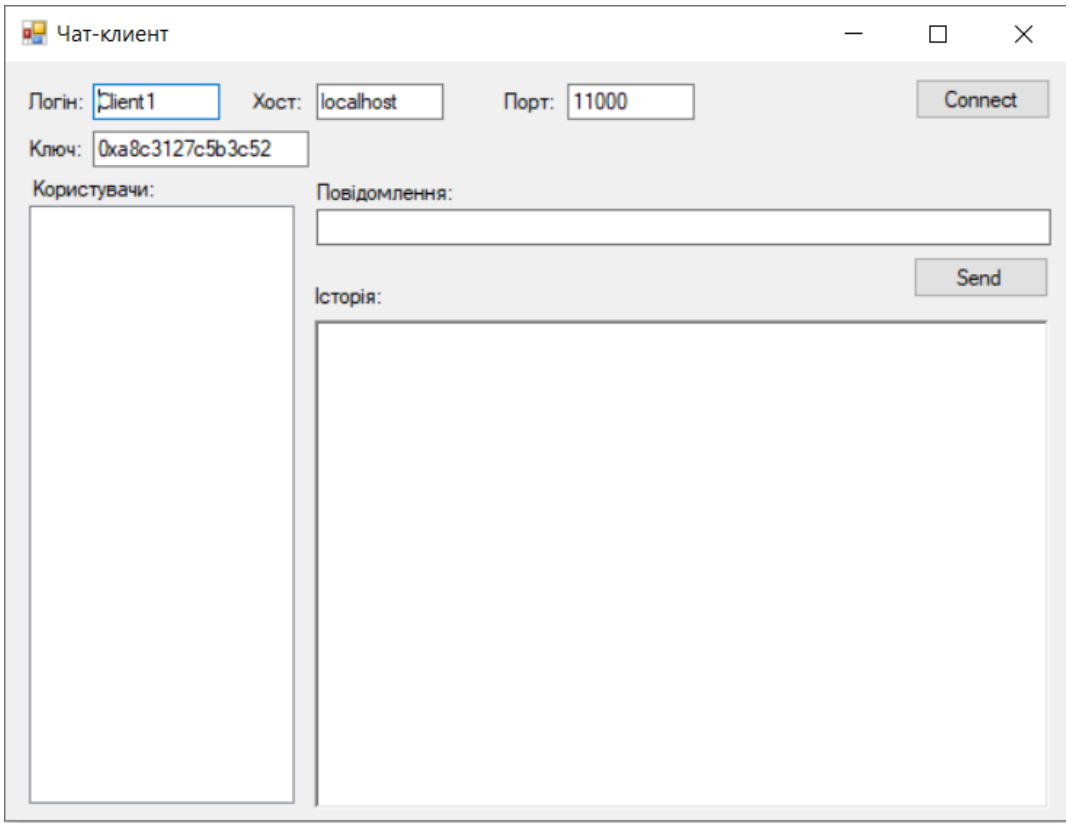

Рисунок 4.2. Макет інтерфейсу користувача

Прив'язка елементів керування до подій:

Додамо обробники подій для кнопки підключення до сервера, щоб обробляти дії користувача.

```
private void btnConnect_Click(object sender, EventArgs e)
    {
      //SendMessageFromSocket(11000);
      conn.Connect(tbLogin.Text, tbHost.Text, tbPort.Text);
      coder = new Des64(tbKey.Text);
      tbId.Text = conn.Addr;
    }
```

Напишемо код для обробки натискання кнопки, надсилання повідомлення на сервер.

```
        private void btnSend_Click(object sender, EventArgs e)
        {
            if (lbUsers.SelectedIndex < 0)
            String msg = tbMessage.Text;
            while ((Encoding.UTF8.GetByteCount(msg) % 8) != 0)
                msg += " ";
            byte[] crypt = Encoding.UTF8.GetBytes(msg);
```

Кодуємо набір байтів crypt за алгоритмом DES

```
            String secret_msg = Convert.ToBase64String(coder.Encrypt(crypt));
            conn.WriteString("7" + ((ChatUser)lbUsers.SelectedItem).id + "," +
secret_msg);
            rtbHist.AppendText("To " + ((ChatUser)lbUsers.SelectedItem).login +
": " + tbMessage.Text + "\n");
            tbMessage.Text = "";
        }
```

4.2.3 Розробка серверної частини

Код є основною частиною програми, що реалізує сервер для чату з використанням сокетів в C #.

Програма створює серверний сокет sListener, який прослуховує вхідні з'єднання.

```
Socket sListener = new Socket(AddressFamily.InterNetwork, SocketType.Stream,
ProtocolType.Tcp);
```

Встановлює локальні кінцеві точки

```
IPAddress ipAddress = IPAddress.Parse("127.0.0.1");
IPEndPoint ipEndPoint = new IPEndPoint(ipAddress, 11000);
```

Прив'язує сокет до локальної кінцевої точки і починає прослуховування вхідних з'єднань

```
sListener.Bind(ipEndPoint);
```

```
sListener.Listen(10); // Максимальна довжина черги запитів, що входять
Console.WriteLine("Очікування підключень...");
while (true)
    Приймає вхідні підключення
Socket handler = sListener.Accept();
        Обробляє клієнта, що підключився (створює нового об'єкта CUser)
CUser newUser = new CUser(NextUserID.ToString(), handler);
users.Add(newUser);
    Надсилає інформацію про поточних користувачів чату новому клієнту
string userListInfo = GetUsersListInfo();
newUser.WriteString(userListInfo);
Console.WriteLine($"Підключився новий клієнт:
{handler.RemoteEndPoint}");
    Зберігає новий сокет при успішному підключенні
CUser new_user = new CUser(NextUserID.ToString(), handler);
NextUserID++;
users.Add(new_user);
handler.SendTimeout = 100;
handler.ReceiveTimeout = 100;
String s_ip = ((IPEndPoint)handler.RemoteEndPoint).Address.ToString();
String s_port = ((IPEndPoint)handler.RemoteEndPoint).Port.ToString();
IPEndPoint ep = (IPEndPoint)handler.RemoteEndPoint;
   Надсилає ідентифікатор та дані інших користувачів чату
String dt = "2" + new_user.id.ToString() + ",";
foreach(CUser us in users)
{
        if (us == new_user)
            continue;
        dt += us.id.ToString() + "," + us.login + ",";
    }
```

```
//Console.Write($"Clients: {dt}\n");
new_user.WriteString(dt);
```

Основні методи та логіка:

• Метод `Main` - точка входу до програми.

• Встановлює локальну кінцеву точку сервера.

• Створює серверний сокет sListener та починає його прослуховування.

• Під час підключення нового клієнта створюється об'єкт CUser для представлення клієнта.

• Перевіряє та обробляє дані від кожного клієнта у циклі.

• Метод `ReadData` у класі `CUser` - читає дані із сокету клієнта.

• Використовує conn.Receive для отримання даних із сокету.

• Перетворює отримані байти у рядок та обробляє дані.

• Метод `WriteData` у класі `CUser` - надсилає дані клієнту.

• Формує повідомлення та надсилає його через сокет клієнта.

Цикл обробки підключень та даних:

• Цикл while (true) очікує на нові підключення.

• При підключенні нового клієнта створюється новий екземпляр CUser і додається до списку users.

• Далі відбувається безперервне читання та обробка даних від кожного клієнта у циклі foreach.

Обробка різних типів даних від клієнтів:

• data[0] == '1' - обробка логіну нового клієнта.

• data[0] == '7' - обробка нового повідомлення від клієнта.

Відключення та видалення клієнтів:

• При виявленні відключення клієнта видаляє його зі списку users та сповіщення інших клієнтів.

• У коді використовується неблокуючий режим (Blocking = false), що дозволяє уникнути блокування під час читання та надсилання даних.

• Обробка виключень (SocketException) при неприйнятих підключеннях або інших мережних помилках.

• Здійснюється передача повідомлень між клієнтами через сокети, використовуючи певні протоколи та формати повідомлень (наприклад, '1' для логіна, '7' для повідомлень тощо).

4.2.4 Розробка клієнтської частини

Створимо клас SConn.

Оголошуємо публічні поля: Addr, id, login для адреси сервера, ідентифікатора клієнта та імені користувача.

```
public String Addr;
public String id;
public String login;
public Socket conn;
public List<ChatUser> users;
```

Оголошуємо змінні для роботи з IP-адресами.

```
IPHostEntry ipHost;
IPAddress ipAddr;
```

- Оголошуємо масив байт bytes, змінну bytesRec для відстеження кількості прийнятих байт та рядок data для зберігання отриманих даних.

```
byte[] bytes = new byte[1024];
int bytesRec = 0;
String data;
```

- Конструктор класу, що ініціалізує список користувачів та встановлює conn у null.

```
public SConn()
{
  users = new List<ChatUser>();
  conn = null;
}
```

- Метод WriteString, який надсилає рядок на сервер у вигляді масиву байтів.

```
public void WriteString(String src)
{
  if (src.Length == 0) return;
  byte[] msg = Encoding.UTF8.GetBytes(src);
  conn.Send(msg);
}
```

- Метод GetUserLoginById, який повертає об'єкт користувача за його ID.

```
public ChatUser GetUserLoginById(String key_id)
{
  return users.Find(x => x.id == key_id);
}
```

- Метод Connect, що встановлює з'єднання із сервером, використовуючи ім'я користувача, хост та порт.

```
public void Connect(String name, String host, String port)
{
  ipHost = Dns.GetHostEntry(host);
  ipAddr = ipHost.AddressList[0];
  login = name;
  IPEndPoint ipEndPoint = new IPEndPoint(ipAddr, Int32.Parse(port));
  conn = new Socket(ipAddr.AddressFamily, SocketType.Stream,
ProtocolType.Tcp);
  conn.Connect(ipEndPoint);
  conn.SendTimeout = 100;
  conn.ReceiveTimeout = 100;

  IPEndPoint ep = (IPEndPoint)conn.LocalEndPoint;
  Addr = ep.ToString();

  byte[] msg = Encoding.UTF8.GetBytes("1" + login);
```

```
        conn.Send(msg);
    }
```

- Метод GetRequest для обробки вхідних запитів від сервера, таких як оновлення списку користувачів чату, дані про клієнта, що приєднався або відключився, а також отримання повідомлень від інших клієнтів.

```
    public void GetRequest(ListBox lst, RichTextBox hist, Des64 coder)
    {
        if (conn == null || conn.Available == 0) return;
        bytesRec = conn.Receive(bytes);
        data = Encoding.UTF8.GetString(bytes, 0, bytesRec);
        // Обробка різних типів запитів від сервера
        if (data[0] == '2')
            // Оновлення списку користувачів чату
            // (Див. Додаток Б)
        }
        else if (data[0] == '3')
        {
            // Дані про нового клієнта
            // (Див. Додаток Б)
        }
        else if (data[0] == '4')
        {
            // Дані про клієнта, що відключився
            // (Див. Додаток Б)
        }
        else if (data[0] == '8')
        {
            // Отримано повідомлення від іншого клієнта
            // (Див. Додаток Б)
```

```
    }
}
```

Повертаємося у Windows Forms і створюємо обробник події після натискання кнопки Connect (див. Рис 4.1). Тим самим ініціалізуємо з'єднання та передаємо в клас SConn дані про користувача та повідомлення:

Addr: Зберігає адресу сервера.

id: Ідентифікує клієнта.

login: Зберігає ім'я користувача клієнта.

conn: Об'єкт сокету для спілкування із сервером.

users: Список об'єктів ChatUser, які представляють інших користувачів у чаті.

ipHost, ipAddr: Використовуються для роботи з IP-адресами.

bytes, bytesRec, data: Змінні для роботи з масивами байт та отриманими даними.

З допомогою методу WriteString відправляємо рядкове повідомлення на сервер.

Метод GetRequest обробляє різні типи вхідних даних, що ідентифікуються першим символом отриманого повідомлення. Наприклад:

'2': Вказує на оновлення списку користувачів чату.

'3': Вказує на дані про нового клієнта, що приєднався.

'4': Вказує дані про звільнення клієнта.

'8': Вказує на отримання повідомлення з іншого клієнта.

Кожен випадок обробляє відповідну дію, таку як оновлення списку користувачів, додавання нового користувача, видалення користувача або відображення отриманих повідомлень в історії чату.

4.2.5 Написання DES-шифрування

Код реалізує основні кроки алгоритму DES, включаючи розширення ключа, перетворення Фейстеля і циклічні зрушення.

У додатку Б реалізується шифр DES (Data Encryption Standard) до роботи з повідомленнями довжиною 64 біта.

1. Des64 - клас, що представляє реалізацію DES шифрування. Він містить методи для шифрування та дешифрування повідомлень, а також допоміжні методи та поля.
2. key та message - масиви байтів для зберігання ключа та повідомлення відповідно.
3. key56 і key64 - масиви булевих значень для представлення ключа в 56 та 64 бітах відповідно.
4. k - масив масивів булевих значень зберігання 48-бітних ключів шифрування.
5. Методи to_bool та to_byte використовуються для перетворення масивів байтів у масиви булевих значень та навпаки.
6. Rearrange - метод виконання перестановки бітів масиву відповідно із заданою таблицею перестановки.
7. extend_key_56_64bit – метод для розширення 56-бітного ключа до 64 біт шляхом додавання контрольних сум.
8. shift_lcyc1 та shift_lcyc2 - методи для виконання циклічного зсуву вліво на один або два біти відповідно.
9. generate_keys – метод для генерації 16 ключів шифрування.
10. Feist - метод для виконання перетворення Фейстеля Він заснований на блоковому шифруванні, де блок даних розбивається на дві половини, а потім проходить через серію раундів (див. Додаток А), де одна половина піддається деяким операціям, що залежать від іншої половини і ключа шифрування.
11. Encrypt64 та Decrypt64 - методи для шифрування та дешифрування 64-бітних повідомлень відповідно.
12. Encrypt та Decrypt - методи для шифрування та дешифрування повідомлень довільної довжини, які автоматично розбивають повідомлення на блоки по 64 біти.
13. print_bytes і print_bits - допоміжні методи виведення масивів байтів і булевих значень в консоль.

4.2.6 Інтеграція DES - шифрування в клієнт-серверний чат

Повертаємося в Windows Forms і створюємо обробник події після натискання кнопки Send (див. рис. 4.1). Коли користувач натискає кнопку для надсилання повідомлення, виконується таке:

1. Перевіряється, чи вибраний користувач зі списку. Якщо ні, нічого не відбувається.
2. Отримуємо текст повідомлення із текстового поля tbMessage.
3. Виконується цикл, щоб доповнити повідомлення пробілами доти, доки його довжина буде кратна 8 байтам. Це робиться для відповідності до вимог алгоритму шифрування DES, який працює з блоками по 8 байт.
4. Повідомлення перетворюється на масив байтів за допомогою кодування UTF-8.
5. Масив байтів crypt шифрується алгоритмом DES і результат перетворюється на рядок у кодуванні Base64.
6. Формується рядок secret_msg, що містить зашифроване повідомлення.
7. Викликається метод WriteString об'єкта conn, щоб надіслати повідомлення клієнту, попередньо додавши символ 7, а потім ідентифікатор користувача та зашифроване повідомлення.
8. Додається інформація про надіслане повідомлення у вікно історії чату.
9. Очищається текстове поле tbMessage.

```
private void btnSend_Click(object sender, EventArgs e)
{
    if (lbUsers.SelectedIndex < 0)
        return;
    String msg = tbMessage.Text;
    while ((Encoding.UTF8.GetByteCount(msg) % 8) != 0)
        msg += " ";
    byte[] crypt = Encoding.UTF8.GetBytes(msg);
```

```
            // Кодируем набор байтов crypt по алгоритму DES
            String secret_msg = Convert.ToBase64String(coder.Encrypt(crypt));
            conn.WriteString("7" + ((ChatUser)lbUsers.SelectedItem).id + "," +
secret_msg);
            rtbHist.AppendText("To " + ((ChatUser)lbUsers.SelectedItem).login + ": " +
tbMessage.Text + "\n");
            tbMessage.Text = "";
        }
```

При отриманні повідомлення іншим клієнтом відбувається таке:

1. Перевірте перший символ повідомлення. Якщо він дорівнює '8', це означає, що отримане повідомлення є зашифрованим текстом.
2. Виймається номер клієнта із повідомлення, що знаходиться після символу '8' і до першої коми.
3. За отриманим номером клієнта (u_id) вилучається його логін зі списку користувачів.
4. Саме повідомлення отримується після коми.
5. Використовуючи об'єкт coder відбувається розшифровка отриманого зашифрованого повідомлення за допомогою алгоритму, ймовірно, DES. Для цього спочатку рядок msg декодується з формату Base64 масив байт, потім цей масив байт розшифровується і декодується назад в рядок UTF-8.
6. Розшифроване повідомлення додається до історії повідомлень із зазначенням імені відправника.

```
if (data[0] == '8')
{
    // Витягаємо номер клієнта
    int pos = data.IndexOf(',');
    String u_id = data.Substring(1, pos - 1);
    // Отримуємо login за номером клієнта
    String u_login = users.Find(x => x.id == u_id).login;
```

```
            // Вилучаємо саме повідомлення
            String msg = data.Substring(pos + 1);
            String msg1 = Encoding.UTF8.GetString(coder.Decrypt(Convert.FromBase64String(msg)));
            // Оновлюємо історію повідомлень
            hist.AppendText(u_login + ": " + msg1 + "\n");
        }
```

З боку сервера відбувається обробка нових повідомлень, які мають бути надіслані іншому клієнту.

1. Перевірте перший символ повідомлення користувача (user.data). Якщо він дорівнює '7', це означає, що це нове повідомлення, яке потрібно надіслати іншому клієнту.
2. Спочатку замінюється перший символ на '0', щоб визначити, що повідомлення було оброблено.
3. Потім отримується номер одержувача повідомлення, який знаходиться після першого символу і до першої коми.
4. Саме повідомлення отримується після коми.
5. Отримане повідомлення декодується з формату Base64 масив байт.
6. За номером одержувача (whom_id) знаходиться відповідний об'єкт клієнта у списку користувачів.
7. Формується рядок повідомлення (dt), що містить символ '8', ідентифікатор відправника (user.id) та зашифроване повідомлення.
8. Метод WriteString об'єкта us викликається надсилання повідомлення адресату.

Крім того, виводиться вміст повідомлення у вигляді рядка шістнадцяткових значень для налагоджувальних цілей за допомогою Console.Write().

4.3 Результат роботи програми

1. Сервер підключається до порту на хості і чекає на з'єднання з клієнтом;
2. Клієнт створює сокет та намагається з'єднати його з портом на хості;
3. Якщо створення сокету пройшло успішно, сервер переходить у режим очікування команд від клієнта;
4. Клієнт формує команду та передає її серверу, переходить у режим очікування відповіді;
5. Сервер приймає команду, виконує її та пересилає відповідь клієнту;
6. Клієнт обробляє отриману інформацію і виводить її користувачеві у зручному вигляді, на її сприйняття.
7. Поки клієнт або сервер не розірве з'єднання, дивіться пункт 4.

Додаток поєднує в собі два режими:

Режим сервера: запускає сервер на порту 11000. Являє собою консольне вікно зі списком клієнтів, що підключилися до нього. У списку відображається IP адреса, за якою клієнти мають можливість підключитися і повідомлення клієнтів у шифрованому вигляді;

Режим клієнта: відкривається вікно в якому потрібно внести найменування сервера (хоста) логін, який повинен складати від 5 до 15 латинських, російських символів або цифр, порт, що використовується сервером і ключ.

Під час створення сервера генерується симетричний ключ алгоритмом DES, цим ключем сервер шифрує всю інформацію, що передається клієнтам. Для отримання цієї інформації клієнт при підключенні до сервера генерує у себе парний ключ.

Активація сервера:

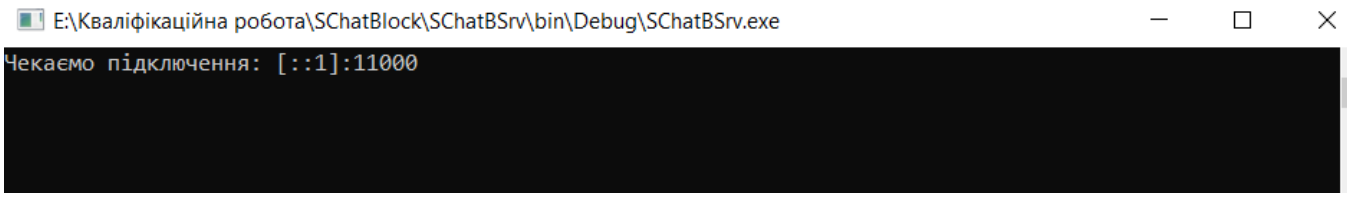

Рисунок 4.3. Старт серверу

Додавання нових користувачів:

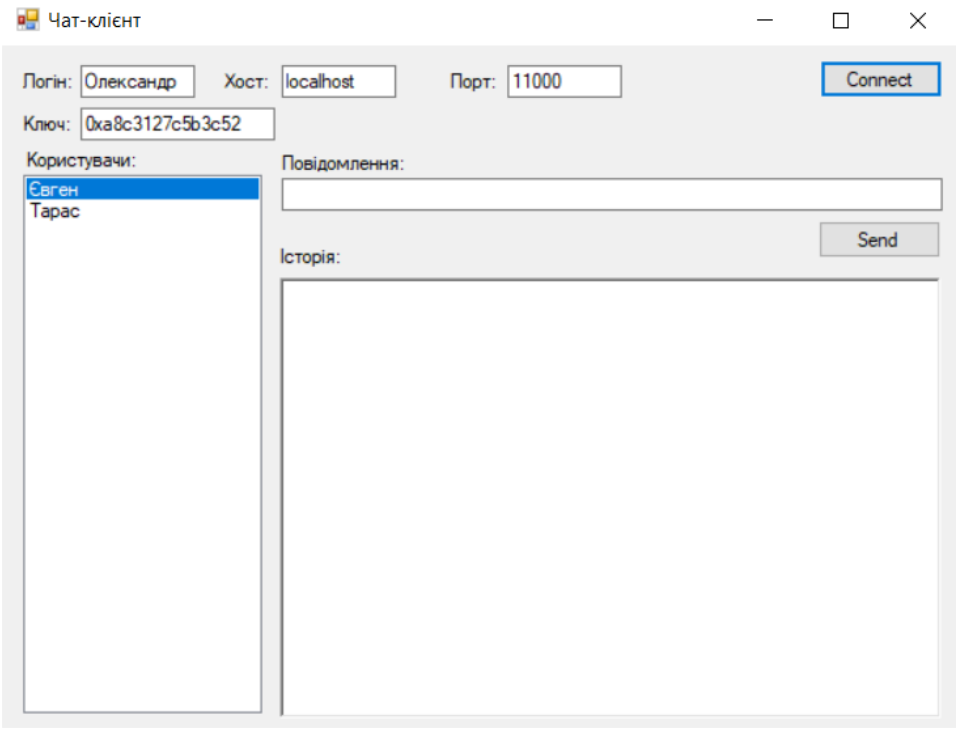

Рисунок 4.4. Додавання першого користувача

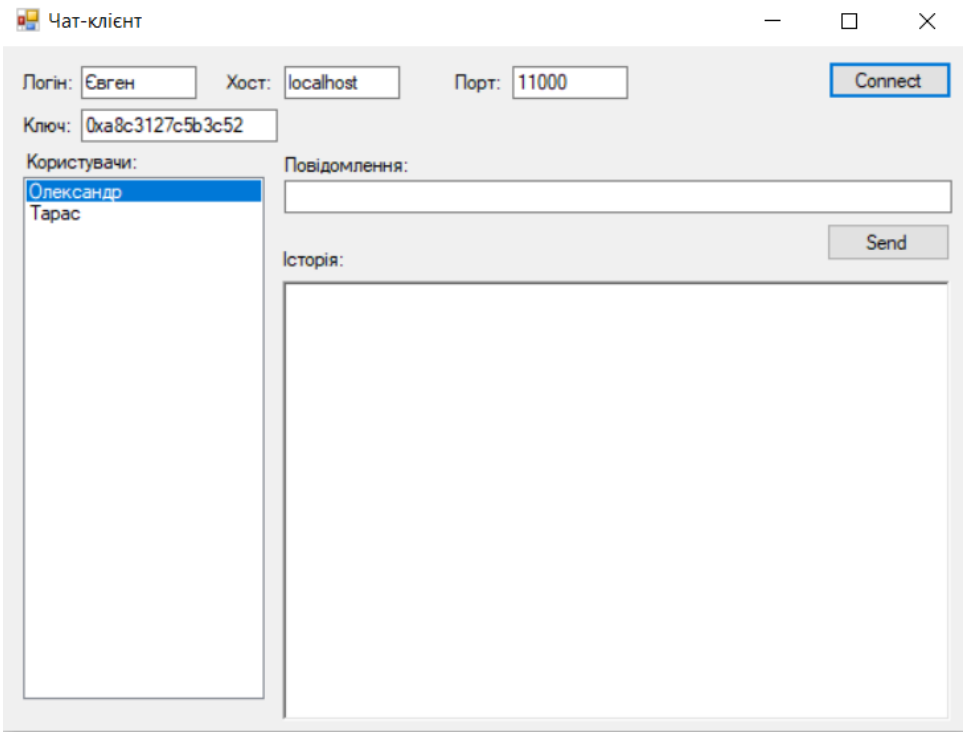

Рисунок 4.5. Додавання другого користувача

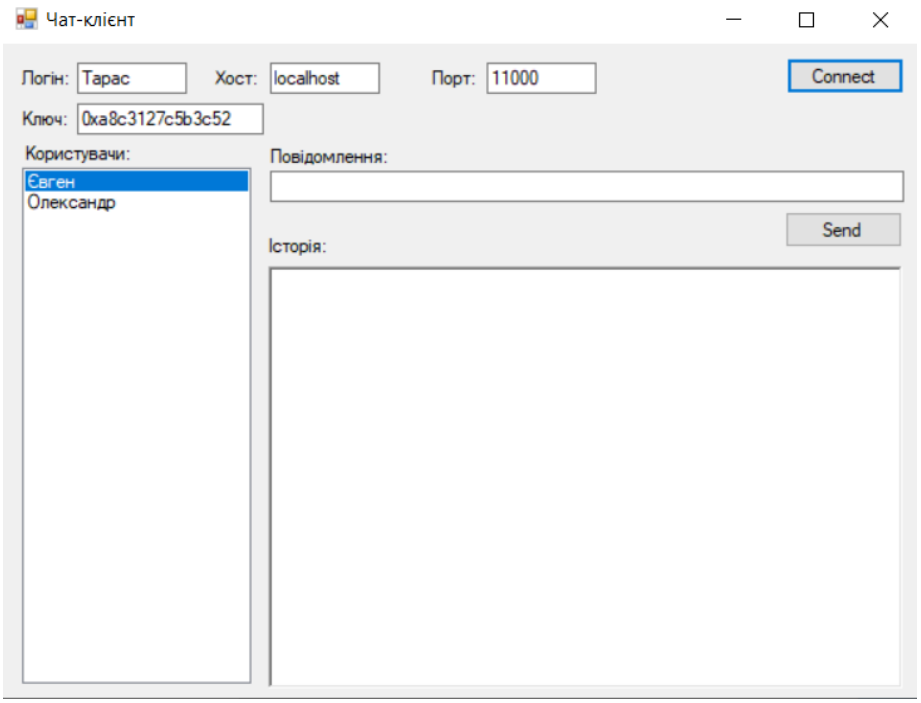

Рисунок 4.6. Додавання третього користувача

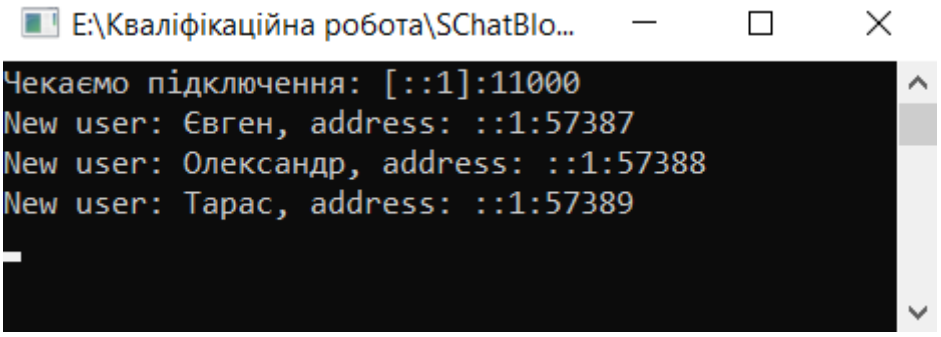

Рисунок 4.7. Виведення даних о підключених користувачах

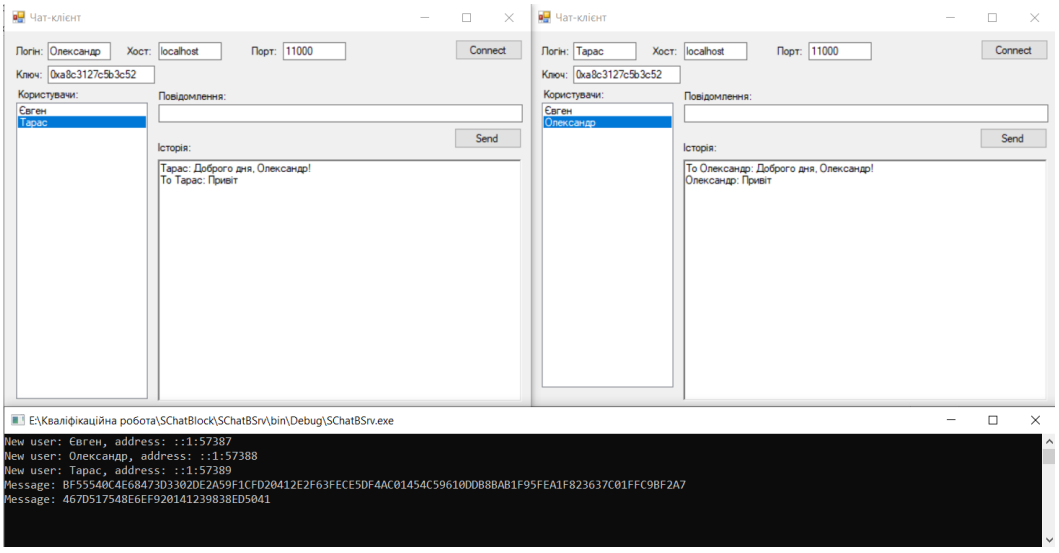

Рисунок 4.8. Результат відправки повідомлення

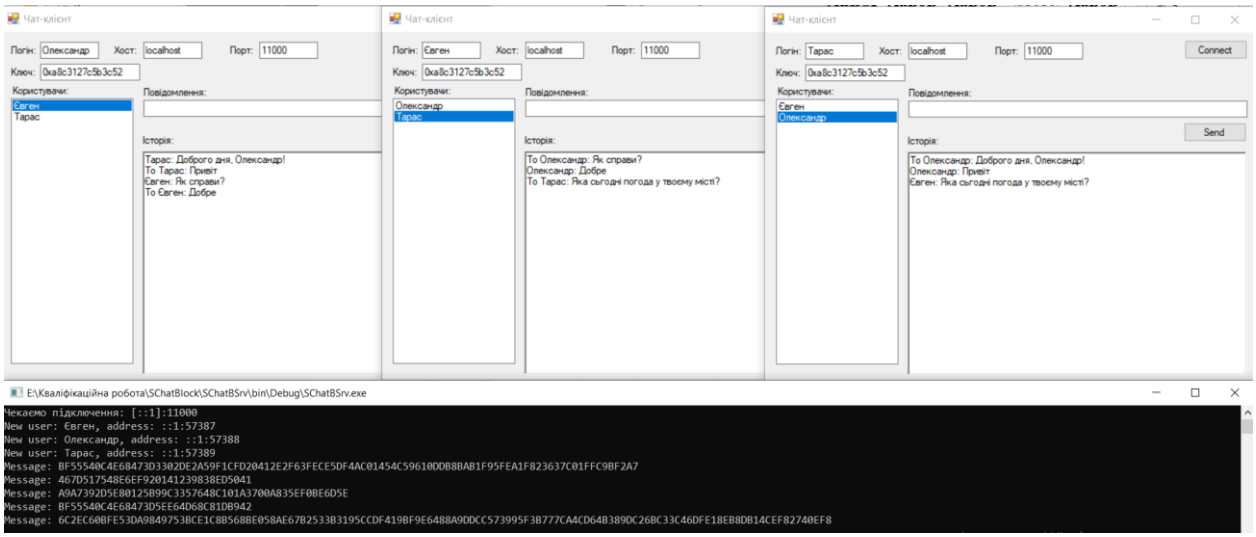

Рисунок 4.9 Результат відправки повідомлень між трьома користувачами

ВИСНОВКИ

Криптографічні методи використовуються як для захисту інформації так і для захищеної передачі даних через глобальні мережі. Ці методи включають в себе механізми шифрування, кодування, аутентифікацію користувачів, цифровий підпис, забезпечення цільності даних.

У рамках роботи я:

- Розробив програму, що забезпечує зашифровану передачу даних у локальній мережі з використанням алгоритму DES. Це дозволяє користувачам обмінюватися повідомленнями, забезпечуючи конфіденційність та безпеку даних.
- Вивчив та реалізував різні технічні аспекти, такі як передача даних по локальній мережі, розробка інтерфейсу користувача та реалізація алгоритму DES для шифрування та дешифрування даних.
- Провів аналіз криптографічних алгоритмів, і особливу увагу було приділено алгоритму DES. Розглянув його структура, режими роботи та криптостійкість, що дозволило вибрати його як основу для розробленої програми.

Розроблений додаток є практичним рішенням для забезпечення безпечної передачі даних у локальній мережі. Воно задовольняє вимогам зручності використання, простоти настроювання та високого ступеня захисту даних.

Надалі можна розглянути можливості розширення функціональності програми, запровадження додаткових криптографічних алгоритмів підвищення безпеки, і навіть адаптації докладання до роботи у інших операційних системах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. КЛАСИФІКАЦІЯ КРИПТОГРАФІЧНИХ АЛГОРИТМІВ [Електронний ресурс] – Режим доступу: https://cyberleninka.ru/article/n/klassifikatsiya-kriptograicheskih-algoritmov/viewer
2. Класифікація криптографічних алгоритмів | Your Private Network. Your Private Network | [Електронний ресурс] – Режим доступу: http://ypn.ru/228/classification-of-cryptographic-algorithms/
3. Класифікація криптографічних алгоритмів. - Інформатика, інформаційні технології. [Електронний ресурс] – Режим доступу: http://csaa.ru/klassifikacija-kriptograficheskih-algoritmov/
4. Ростовцев А.Г., Маховенко Е.Б. Теоретична криптографія. – СПб.: НПО «Професіонал», 2003. – 478 с
5. Панасенко С.П. Алгоритми шифрування. Спеціальний довідник. – СПб.: БХВ-Петербург, 2009. – 576 с.
6. Криптографічна стійкість [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Криптографічна_стійкість
7. Гатченко Н.А., Исаев А.С., Яковлев А.Д. Криптографічний захист інформації. – СПб:НИУ ИТМО, 2012. 142 с.
8. ПРАКТИЧНА КРИПТОЛОГІЯ ЛЕКЦІЯ 3. [Електронний ресурс] – Режим доступу: http://bit.nmu.org.ua/ua/student/metod/cryptology/лекция 3.pdf
9. Мао В. Сучасна криптографія: Теорія та практик. – Вильямс, 2005. 768 с
10. Шурховецкий Г.Н. Криптостійкість алгоритмів шифрування [Електронний ресурс] / Г.Н. Шурховецкий // Молода наука Сибіру: електрон. науковий журнал – 2018. – №2. – Режим доступу: http://mnv.irgups.ru/toma/22-2018,
11. Романцев Ю.В., Тимофеев П.А., Шаньгин В.Ф. Захист інформації в комп'ютерних системах та мережах. – М.: Радіо та зв'язок, 2001. – 376 с

12. Шнайер Б. Прикладна криптографія. Протоколи, алгоритми, вихідні тексти мовою С. – Пер. с англ.: М.: Видавництво ТРИУМФ, 2002. 816с.
13. Biham E., Shamir A. Differential cryptanalysis of DES-like cryptosystems. Journal of Cryptology. 1991. Vol.4, no.1. P.3–72. Режим доступу: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=5049bb7e0114edccd2841c1788a114567d040f3c
14. Biham E. New types of cryptanalytic attacks using related keys. Journal of Cryptology. 1994. Vol.7, no.4. P.229–246. Режим доступу: https://doi.org/10.1007/bf00203965
15. Data Encryption Standard [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Data_Encryption_Standard
16. Організація ком'ютерних мереж [Електронний ресурс] – Режим доступу: https://marytisna.blogspot.com/2017/12/blog-post_11.html
17. Transmission Control Protocol [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Transmission_Control_Protocol
18. Data encryption standard (DES) [Електронний ресурс] – Режим доступу: https://www.geeksforgeeks.org/data-encryption-standard-des-set-1/
19. Technische Universität Berlin [Електронний ресурс] – Режим доступу: https://page.math.tu-berlin.de/~kant/teaching/hess/krypto-ws2006/des.htm
20. Differential cryptanalysis [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Differential_cryptanalysis#:~:text=Differential%20cryptanalysis%20is%20a%20general,resultant%20difference%20at%20the%20output.

ДОДАТОК А

		Номер стовбця																
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	×
Номер рядка	0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7	S_1
	1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8	
	2	4	1	4	8	13	6	2	11	15	12	9	7	3	10	5	0	
	3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13	
	0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10	S_2
	1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5	
	2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15	
	3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9	
	0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8	S_3
	1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1	
	2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7	
	3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12	
	0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15	S_4
	1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9	
	2	10	5	9	0	12	11	7	13	15	1	3	14	5	10	14	7	
	3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14	
	0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9	S_5
	1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6	
	2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14	
	3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3	
	0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11	S_6
	1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8	
	2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	1	6	
	3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13	

		Номер стовбця																
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	×
Номер рядка	0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1	S_7
	1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6	
	2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2	
	3	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12	
	0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7	S_8
	1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2	
	2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8	
	3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11	

Таблиця замін S_1… S_8

ДОДАТОК Б

ChatUser.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ClientGUI
{
    class ChatUser
    {
        public String id;
        public String login;

        public ChatUser(String num, String name)
        {
            id = num;
            login = name;
        }

        public override string ToString()
        {
            return login;
        }
    }
}
```

Program.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace ClientGUI
{
    static class Program
    {
        /// <summary>
        /// Головна точка входу для програми.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new FMain());
        }
    }
}
```

SConn.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Net;
using System.Net.Sockets;
using System.Windows.Forms;
```

```
using DesEnc;

namespace ClientGUI
{
    class SConn
    {
        public String Addr;
        // Ідентифікатор клієнта (запитується при
        // з'єднання з сервером)
        public String id;
        // Ім'я користувача чату
        public String login;
        // Сокет для зв'язку із сервером
        Socket conn;
        // Список інших клієнтів
        public List<ChatUser> users;

        IPHostEntry ipHost;
        IPAddress ipAddr;

        byte[] bytes = new byte[1024];

        int bytesRec = 0;
        String data;

        public SConn()
        {
            users = new List<ChatUser>();
            conn = null;
        }

        public void WriteString(String src)
        {
            if (src.Length == 0) return;
            byte[] msg = Encoding.UTF8.GetBytes(src);
            conn.Send(msg);
        }

        public ChatUser GetUserLoginById(String key_id)
        {
            return users.Find(x => x.id == key_id);
        }

        public void Connect(String name, String host, String port)
        {
            ipHost = Dns.GetHostEntry(host);
            ipAddr = ipHost.AddressList[0];
            login = name;
            IPEndPoint ipEndPoint = new IPEndPoint(ipAddr, Int32.Parse(port));
            conn = new Socket(ipAddr.AddressFamily, SocketType.Stream, ProtocolType.Tcp);
            //conn.Blocking = false;

            // З'єднуємо сокет із віддаленою точкою
            conn.Connect(ipEndPoint);
            conn.SendTimeout = 100;
            conn.ReceiveTimeout = 100;

            IPEndPoint ep = (IPEndPoint)conn.LocalEndPoint;
            Addr = ep.ToString();

            // Передаємо серверу ім'я нового клієнта
            byte[] msg = Encoding.UTF8.GetBytes("1" + login);
            conn.Send(msg);
        }
```

```
public void GetRequest(ListBox lst, RichTextBox hist, Des64 coder)
{

    if (conn == null || conn.Available == 0) return;

    bytesRec = conn.Receive(bytes);
    data = Encoding.UTF8.GetString(bytes, 0, bytesRec);
    // Ідентифікатор клієнта та список учасників чату
    if (data[0] == '2')
    {
        // Витягаємо номер клієнта
        int pos = data.IndexOf(',');
        id = data.Substring(1, pos - 1);

        // Виймаємо номери та імена інших користувачів чату
        users.Clear();
        while (pos >= 0)
        {
            int start = pos + 1;
            pos = data.IndexOf(',', start);
            if (pos < 0)
                break;
            String u_id = data.Substring(start, pos - start);
            start = pos + 1;
            pos = data.IndexOf(',', start);
            String u_name = data.Substring(start, pos - start);
            users.Add(new ChatUser(u_id, u_name));
        }
        lst.BeginUpdate();
        lst.Items.Clear();
        foreach (var us in users)
        {
            lst.Items.Add(us);
        }
        if (lst.Items.Count > 0)
            lst.SelectedIndex = 0;
        lst.EndUpdate();
    }

    // Дані про нового клієнта
    if (data[0] == '3')
    {
        // Витягаємо номер клієнта
        int pos = data.IndexOf(',');
        String u_id = data.Substring(1, pos - 1);

        int ei = data.IndexOf(',', pos + 1);
        String u_name = data.Substring(pos + 1, ei - pos - 1);
        ChatUser us = new ChatUser(u_id, u_name);
        users.Add(us);

        lst.BeginUpdate();
        lst.Items.Add(us);
        if (lst.SelectedIndex < 0)
            lst.SelectedIndex = 0;
        lst.EndUpdate();
    }

    // Дані про клієнта, що відключився
    if (data[0] == '4')
    {
        // Витягаємо номер клієнта
        String u_id = data.Substring(1);
```

```
                // Знаходимо користувача у списку та видаляємо його
                lst.Items.Remove( users.Find(x => x.id == u_id) );
                if ((lst.Items.Count > 0) && (lst.SelectedIndex < 0))
                    lst.SelectedIndex = 0;
            }

            // Получено сообщение от другого клиента
            if (data[0] == '8')
            {
                // Извлекаем номер клиента
                int pos = data.IndexOf(',');
                String u_id = data.Substring(1, pos - 1);

                // Отримуємо login за номером клієнта
                String u_login = users.Find(x => x.id == u_id).login;


                // Вилучаємо саме повідомлення
                String msg = data.Substring(pos + 1);

                String msg1 =
Encoding.UTF8.GetString(coder.Decrypt(Convert.FromBase64String(msg)));

                // Оновлюємо історію повідомлень
                hist.AppendText(u_login + ": " + msg1 + "\n");
            }
        }
    }
}
```

Des64.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DesEnc
{
    public class Des64
    {
        public byte[] key;
        public byte[] message;

        // Оригінальний та розширений ключ у булевому вигляді
        public bool[] key56 = null;
        public bool[] key64 = null;

        // 48-бітові ключі шифрування
        public bool[][] k;

        // Вихідне повідомлення у формі бульова масиву
        public bool[] b_T = null;
        // Початкова перестановка у булевому вигляді
        public bool[] b_IP_T = null;

        public Des64(byte[] passw)
        {
            key = new byte[7];
            Array.Copy(passw, key, 7);
            k = new bool[16][];
        }

        // Пароль у вигляді рядкового запису шістнадцяткового числа
        public Des64(String passw)
```

```
{
    UInt64 pw16 = Convert.ToUInt64(passw, 16);
    key = new byte[7];
    //Array.Copy(passw, key, 7);
    for (int i = 0; i < 7; i++ )
    {
        key[i] = (byte)(pw16 & 0x00FF);
        pw16 >>= 8;
    }
    k = new bool[16][];
}

// Таблиця початкової перестановки
int[] IP = new int[]
{
    58, 50, 42, 34, 26, 18, 10, 2, 60, 52, 44, 36, 28, 20, 12, 4,
    62, 54, 46, 38, 30, 22, 14, 6, 64, 56, 48, 40, 32, 24, 16, 8,
    57, 49, 41, 33, 25, 17, 9, 1, 59, 51, 43, 35, 27, 19, 11, 3,
    61, 53, 45, 37, 29, 21, 13, 5, 63, 55, 47, 39, 31, 23, 15, 7
};

// Таблиця кінцевої перестановки
int[] IPm1 = new int[]
{
    40, 8, 48, 16, 56, 24, 64, 32, 39, 7, 47, 15, 55, 23, 63, 31,
    38, 6, 46, 14, 54, 22, 62, 30, 37, 5, 45, 13, 53, 21, 61, 29,
    36, 4, 44, 12, 52, 20, 60, 28, 35, 3, 43, 11, 51, 19, 59, 27,
    34, 2, 42, 10, 50, 18, 58, 26, 33, 1, 41, 9, 49, 17, 57, 25
};

// Перетворення байтів на булев масив
public bool[] to_bool(byte[] src)
{
    int len = src.Length;
    bool[] res = new bool[len * 8];
    byte cb;

    for(int i = 0; i < len; i++)
    {
        cb = src[i];
        for(int j = 0; j < 8; j++)
        {
            res[i * 8 + j] = ((cb & 0x80) != 0);
            cb <<= 1;
        }
    }

    return res;
}

// Перетворення булевого масиву на байти
public byte[] to_byte(bool[] src)
{
    int len = src.Length;
    byte[] res = new byte[len / 8];
    byte cb;

    for(int i = 0; i < res.Length; i++)
    {
        cb = 0;
        for(int j = 0; j < 8; j++)
        {
            cb <<= 1;
            if (src[i * 8 + j])
                cb |= 1;
        }
        res[i] = cb;
    }
```

```
    return res;
}

// Обчислення перестановки довжини len для src за таблицею tbl
public bool[] Rearrange(bool[] src, int[] tbl, int len)
{
    bool[] res = new bool[len];

    for(int i = 0; i < len; i++)
    {
        res[i] = src[tbl[i] - 1];
    }

    return res;
}

// Розширення 56-бітного ключа до 64 біт додаванням контрольних сум
public bool[] extend_key_56_64bit(bool[] src)
{
    bool[] res = new bool[64];

    int k = 0;
    for(int i = 0; i < 8; i++)
    {
        bool bit8x = true;
        for(int j = 0; j < 7; j++)
        {
            res[k++] = src[i * 7 + j];
            if (src[i * 7 + j])
                bit8x = !bit8x;
        }
        res[k++] = bit8x;
    }

    return res;
}

public bool[] shift_lcyc1(bool[] src)
{
    int len = src.Length;
    bool[] res = new bool[len];

    bool bit_l1 = src[0];
    for(int i = 0; i < len - 1; i++)
        res[i] = src[i + 1];
    res[len - 1] = bit_l1;

    return res;
}
public bool[] shift_lcyc2(bool[] src)
{
    int len = src.Length;
    bool[] res = new bool[len];

    bool bit_l1 = src[0];
    bool bit_l2 = src[1];
    for (int i = 0; i < len - 2; i++)
        res[i] = src[i + 2];
    res[len - 2] = bit_l1;
    res[len - 1] = bit_l2;

    return res;
}

// Генерація ключів шифрування
public void generate_keys()
{
```

```
    // Перетворимо вихідний ключ на булев масив
    key56 = to_bool(key);

    // Розширюємо ключ до 64 біт
    key64 = extend_key_56_64bit(key56);

    // Виконуємо перестановку для розширеного ключа
    int[] tbl_C0 = new int[] {
        57, 49, 41, 33, 25, 17, 9, 1, 58, 50, 42, 34, 26, 18,
        10, 2, 59, 51, 43, 35, 27, 19, 11, 3, 60, 52, 44, 36 };

    int[] tbl_D0 = new int[] {
        63, 55, 47, 39, 31, 23, 15, 7, 62, 54, 46, 38, 30, 22,
        14, 6, 61, 53, 45, 37, 29, 21, 13, 5, 28, 20, 12, 4 };

    bool[] Ci = Rearrange(key64, tbl_C0, 28);
    bool[] Di = Rearrange(key64, tbl_D0, 28);

    int[] tbl_k = new int[] {
        14, 17, 11, 24, 1, 5, 3, 28, 15, 6, 21, 10, 23, 19, 12, 4,
        26, 8, 16, 7, 27, 20, 13, 2, 41, 52, 31, 37, 47, 55, 30, 40,
        51, 45, 33, 48, 44, 49, 39, 56, 34, 53, 46, 42, 50, 36, 29, 32 };

    // Отримуємо ключі шифрування k1...k16
    int[] shifts = new int[16] { 1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1 };
    for (int i = 0; i < 16; i++)
    {
        // Обчислюємо нові вектори Ci, Di
        if (shifts[i] == 1)
        {
            Ci = shift_lcyc1(Ci);
            Di = shift_lcyc1(Di);
        }
        else
        {
            Ci = shift_lcyc2(Ci);
            Di = shift_lcyc2(Di);
        }

        // Формуємо об'єднаний вектор CiDi
        bool[] CD = new bool[56];
        Array.Copy(Ci, 0, CD, 0, 28);
        Array.Copy(Di, 0, CD, 28, 28);

        // Виконуємо перестановку (обчислюємо Ki)
        k[i] = Rearrange(CD, tbl_k, tbl_k.Length);
    }
}

// Перетворення Фейстеля
public bool[] Feist(bool[] Ri, bool[] ki)
{
    bool[] res = new bool[32];

    // Виконуємо розширення Ri до 48 біт
    int[] tbl_E = new int[] {
        32, 1, 2, 3, 4, 5, 4, 5, 6, 7, 8, 9, 8, 9, 10, 11,
        12, 13, 12, 13, 14, 15, 16, 17, 16, 17, 18, 19, 20, 21, 20, 21,
        22, 23, 24, 25, 24, 25, 26, 27, 28, 29, 28, 29, 30, 31, 32, 1 };
    bool[] ERi = Rearrange(Ri, tbl_E, tbl_E.Length);

    // Додавання по модулю 2 з ключем ki
    bool[] ERi_2_Ki = bXOR(ERi, ki);

    //print_bits(ERi_2_Ki, "Feist, ERi2Ki: ");

    int[,] S1 = new int[4, 16] {
        { 14, 4, 13, 1, 2, 15, 11, 8, 3, 10, 6, 12, 5, 9, 0, 7 },
```

```
    { 0, 15, 7, 4, 14, 2, 13, 1, 10, 6, 12, 11, 9, 5, 3, 8 },
    { 4, 1, 14, 8, 13, 6, 2, 11, 15, 12, 9, 7, 3, 10, 5, 0 },
    {15, 12, 8, 2, 4, 9, 1, 7, 5, 11, 3, 14, 10, 0, 6, 13 } };

int[,] S2 = new int[4, 16] {
    { 15, 1, 8, 14, 6, 11, 3, 4, 9, 7, 2, 13, 12, 0, 5, 10 },
    { 3, 13, 4, 7, 15, 2, 8, 14, 12, 0, 1, 10, 6, 9, 11, 5 },
    { 0, 14, 7, 11, 10, 4, 13, 1, 5, 8, 12, 6, 9, 3, 2, 15 },
    { 13, 8, 10, 1, 3, 15, 4, 2, 11, 6, 7, 12, 0, 5, 14, 9 } };

int[,] S3 = new int[4, 16] {
    { 10, 0, 9, 14, 6, 3, 15, 5, 1, 13, 12, 7, 11, 4, 2, 8 },
    { 13, 7, 0, 9, 3, 4, 6, 10, 2, 8, 5, 14, 12, 11, 15, 1 },
    { 13, 6, 4, 9, 8, 15, 3, 0, 11, 1, 2, 12, 5, 10, 14, 7 },
    { 1, 10, 13, 0, 6, 9, 8, 7, 4, 15, 14, 3, 11, 5, 2, 12 } };

int[,] S4 = new int[4, 16] {
    { 7, 13, 14, 3, 0, 6, 9, 10, 1, 2, 8, 5, 11, 12, 4, 15 },
    { 13, 8, 11, 5, 6, 15, 0, 3, 4, 7, 2, 12, 1, 10, 14, 9 },
    { 10, 6, 9, 0, 12, 11, 7, 13, 15, 1, 3, 14, 5, 2, 8, 4 },
    { 3, 15, 0, 6, 10, 1, 13, 8, 9, 4, 5, 11, 12, 7, 2, 14 } };

int[,] S5 = new int[4, 16] {
    { 2, 12, 4, 1, 7, 10, 11, 6, 8, 5, 3, 15, 13, 0, 14, 9 },
    { 14, 11, 2, 12, 4, 7, 13, 1, 5, 0, 15, 10, 3, 9, 8, 6 },
    { 4, 2, 1, 11, 10, 13, 7, 8, 15, 9, 12, 5, 6, 3, 0, 14 },
    { 11, 8, 12, 7, 1, 14, 2, 13, 6, 15, 0, 9, 10, 4, 5, 3 } };

int[,] S6 = new int[4, 16] {
    { 12, 1, 10, 15, 9, 2, 6, 8, 0, 13, 3, 4, 14, 7, 5, 11 },
    { 10, 15, 4, 2, 7, 12, 9, 5, 6, 1, 13, 14, 0, 11, 3, 8 },
    { 9, 14, 15, 5, 2, 8, 12, 3, 7, 0, 4, 10, 1, 13, 11, 6 },
    { 4, 3, 2, 12, 9, 5, 15, 10, 11, 14, 1, 7, 6, 0, 8, 13 } };

int[,] S7 = new int[4, 16] {
    { 4, 11, 2, 14, 15, 0, 8, 13, 3, 12, 9, 7, 5, 10, 6, 1 },
    { 13, 0, 11, 7, 4, 9, 1, 10, 14, 3, 5, 12, 2, 15, 8, 6 },
    { 1, 4, 11, 13, 12, 3, 7, 14, 10, 15, 6, 8, 0, 5, 9, 2 },
    { 6, 11, 13, 8, 1, 4, 10, 7, 9, 5, 0, 15, 14, 2, 3, 12 } };

int[,] S8 = new int[4, 16] {
    { 13, 2, 8, 4, 6, 15, 11, 1, 10, 9, 3, 14, 5, 0, 12, 7 },
    { 1, 15, 13, 8, 10, 3, 7, 4, 12, 5, 6, 11, 0, 14, 9, 2 },
    { 7, 11, 4, 1, 9, 12, 14, 2, 0, 6, 10, 13, 15, 3, 5, 8 },
    { 2, 1, 14, 7, 4, 10, 8, 13, 15, 12, 9, 0, 3, 5, 6, 11 } };

int [][,] Si = new int[][,] { S1, S2, S3, S4, S5, S6, S7, S8 };

// Перетворення блоків S1..S8 по 6 біт у кожному
int s_bit = 0;
int d_bit = 0;
for (int i = 0; i < 8; i++)
{
    int row = 0, col = 0;
    if (ERi_2_Ki[s_bit]) row += 2;
    if (ERi_2_Ki[s_bit + 5]) row += 1;

    if (ERi_2_Ki[s_bit + 4]) col += 1;
    if (ERi_2_Ki[s_bit + 3]) col += 2;
    if (ERi_2_Ki[s_bit + 2]) col += 4;
    if (ERi_2_Ki[s_bit + 1]) col += 8;

    // Обчислюємо B-штрих
    int Bs = Si[i][row, col];
    //Console.Write($"Bs{i}: {Bs}\n");

    res[d_bit] = ((Bs & 0x8) != 0);
    res[d_bit + 1] = ((Bs & 0x4) != 0);
```

```
            res[d_bit + 2] = ((Bs & 0x2) != 0);
            res[d_bit + 3] = ((Bs & 0x1) != 0);

            s_bit += 6;
            d_bit += 4;
        }

        //print_bits(Ri, res, "Feist(): ");

        // Виконуємо перестановку P
        int[] tbl_P = new int[] {
            16, 7, 20, 21, 29, 12, 28, 17,
            1, 15, 23, 26, 5, 18, 31, 10,
            2, 8, 24, 14, 32, 27, 3, 9,
            19, 13, 30, 6, 22, 11, 4, 25 };

        res = Rearrange(res, tbl_P, tbl_P.Length);

        return res;
    }

    // Додавання по модулю 2 булевих масивів
    public bool[] bXOR(bool[] a, bool[] b)
    {
        int len = a.Length;
        bool[] res = new bool[len];

        for (int i = 0; i < len; i++)
            res[i] = (a[i] ^ b[i]);
        return res;
    }

    // Кодування 64-розрядного повідомлення
    public byte[] Encrypt64(byte[] mess)
    {
        message = new byte[8];
        Array.Copy(mess, message, 8);

        //print_bytes(message, "Encrypt(): ");

        // Перетворимо вихідне повідомлення на булев масив
        b_T = to_bool(message);
        //print_bits(b_T, "Encrypt(): ");

        // Обчислюємо початкову перестановку
        b_IP_T = Rearrange(b_T, IP, b_T.Length);
        //print_bits(b_IP_T, "Encrypt(): ");

        // Генеруємо ключі
        generate_keys();

        //for(int i = 0; i < 16; i++)
        //    print_bits(k[i], "Key" + (i + 1).ToString() + ": ");

        // Розбиття блоку на ліву та праву частини
        bool[] L = new bool[32];
        bool[] R = new bool[32];
        Array.Copy(b_IP_T, 0, L, 0, 32);
        Array.Copy(b_IP_T, 32, R, 0, 32);

        bool[] Li = new bool[32];
        bool[] Ri = new bool[32];

        //print_bits(L, R, "Li,Ri: ");

        // 16 циклів перетворення Фейстеля
        for(int i = 0; i < 16; i++)
        {
```

```
            Array.Copy(R, Li, R.Length);
            Ri = bXOR(L, Feist(R, k[i]));
            Array.Copy(Li, L, Li.Length);
            Array.Copy(Ri, R, Ri.Length);

            //print_bits(L, R, "L,R-" + i.ToString() + ": ");
        }

        // Об'єднуємо блоки L та R
        bool[] crypt = new bool[64];
        Array.Copy(L, 0, crypt, 0, 32);
        Array.Copy(R, 0, crypt, 32, 32);

        //print_bits(crypt, "Encrypt(): ");

        // Виконуємо перестановку по IP^-1
        bool[] b_res = Rearrange(crypt, IPm1, crypt.Length);

        //print_bits(b_res, "Encrypt(): ");

        // Перетворюємо криптограму на масив байтів
        byte[] res = to_byte(b_res);
        //print_bytes(res, "Encrypt(): ");

        return res;
    }

    // Декодування 64-бітного повідомлення
    public byte[] Decrypt64(byte[] data)
    {
        // Перетворимо криптограму на булев масив
        bool[] bc_T = to_bool(data);

        // Обчислюємо початкову перестановку
        bool[] bc_IP_T = Rearrange(bc_T, IP, bc_T.Length);

        // Генеруємо ключі
        generate_keys();

        // Розбиття блоку на ліву та праву частини
        bool[] L = new bool[32];
        bool[] R = new bool[32];
        Array.Copy(bc_IP_T, 0, L, 0, 32);
        Array.Copy(bc_IP_T, 32, R, 0, 32);

        bool[] Li = new bool[32];
        bool[] Ri = new bool[32];

        // 16 циклів зворотного перетворення Фейстеля
        for (int i = 15; i >= 0; i--)
        {
            Array.Copy(L, Ri, L.Length);
            Li = bXOR(R, Feist(L, k[i]));
            Array.Copy(Li, L, Li.Length);
            Array.Copy(Ri, R, Ri.Length);
        }

        // Об'єднуємо блоки L та R
        bool[] text = new bool[64];
        Array.Copy(L, 0, text, 0, 32);
        Array.Copy(R, 0, text, 32, 32);

        // Виконуємо перестановку по IP^-1
        bool[] b_res = Rearrange(text, IPm1, text.Length);

        // Перетворюємо криптограму на масив байтів
        return to_byte(b_res);
    }
```

```
// Кодування повідомлення з довжиною кратною 64 біт
public byte[] Encrypt(byte[] mess)
{
    byte[] res = new byte[mess.Length];

    byte[] mess8 = new byte[8];
    for(int i = 0; i < mess.Length / 8; i++)
    {
        Array.Copy(mess, i * 8, mess8, 0, 8);
        Array.Copy(Encrypt64(mess8), 0, res, i * 8, 8);
    }

    return res;
}

// Декодування повідомлення з довжиною кратною 64 біт
public byte[] Decrypt(byte[] cr)
{
    byte[] res = new byte[cr.Length];

    byte[] mess8 = new byte[8];
    for (int i = 0; i < cr.Length / 8; i++)
    {
        Array.Copy(cr, i * 8, mess8, 0, 8);
        Array.Copy(Decrypt64(mess8), 0, res, i * 8, 8);
    }

    return res;
}
public static void print_bytes(byte[] ar, String pref)
{
    Console.Write(pref);
    for (int i = 0; i < ar.Length; i++)
        Console.Write($"{ar[i]} ");
    Console.Write("\n");
}
public static void print_bits(bool[] ar, String pref)
{
    Console.Write(pref);
    for (int i = 0; i < ar.Length; i++)
    {
        int j = 0;
        if (ar[i]) j = 1;
        Console.Write($"{j}");
    }
    Console.Write("\n");
}
public static void print_bits(bool[] ar1, bool[] ar2, String pref)
{
    Console.Write(pref);

    for (int i = 0; i < ar1.Length; i++)
    {
        int j = 0;
        if (ar1[i]) j = 1;
        Console.Write($"{j}");
    }
    Console.Write(" ");
    for (int i = 0; i < ar2.Length; i++)
    {
        int j = 0;
        if (ar2[i]) j = 1;
        Console.Write($"{j}");
    }
    Console.Write("\n");
}
```

```
    }
}
```

Program.cs:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using System.Net;
using System.Net.Sockets;

namespace SChatBSrv
{
    // Клієнт чата
    class CUser
    {
        // Унікальний ідентифікатор клієнта (задається сервером)
        public String id;
        // Ім'я користувача чату
        public String login;
        // Сокет для зв'язку з
        public Socket conn;
        // Отримані від клієнта дані
        byte[] bytes = new byte[1024];

        int bytesRec = 0;
        public String data;

        public bool disconnected = false;
        public CUser(String c_id, Socket socket)
        {
            id = c_id;
            conn = socket;
        }

        // Неблокуюче читання даних із сокету клієнта
        public void ReadData()
        {
            bytesRec = 0;
            if (conn.Available == 0) return;

            bytesRec = conn.Receive(bytes);

            data = Encoding.UTF8.GetString(bytes, 0, bytesRec);

            // Показуємо дані на консолі
            //Console.Write($"Клиент {id}: {data} \n");
            if(data[0] == '1')
            {
                login = data.Substring(1);
                //Console.Write($"Логин: {login}\n");
            }
        }

        public void WriteData()
        {
            if (bytesRec == 0) return;
            // Надсилаємо відповідь клієнту
            string reply = "Дякуємо за запит у " + data.Length.ToString()
                    + " символів";
            byte[] msg = Encoding.UTF8.GetBytes(reply);
            conn.Send(msg);
```

```
        }

        public void WriteString(String src)
        {
            if (src.Length == 0) return;
            byte[] msg = Encoding.UTF8.GetBytes(src);
            conn.Send(msg);
        }
    }
    class Program
    {
        static List<CUser> users = new List<CUser>(10);
        static Int32 NextUserID = 1;

        static void Main(string[] args)
        {
            Console.OutputEncoding = UTF8Encoding.UTF8;
            // Встановлюємо для сокету локальну кінцеву точку
            IPHostEntry ipHost = Dns.GetHostEntry("localhost");
            IPAddress ipAddr = ipHost.AddressList[0];
            IPEndPoint ipEndPoint = new IPEndPoint(ipAddr, 11000);
            //Console.Write($"IP: {ipAddr}\n");
            Console.Write($"Чекаємо підключення: {ipEndPoint}\n");

            //Byte[] bytes = ipAddr.GetAddressBytes();
            //for (int i = 0; i < bytes.Length; i++)
            //{
            //    Console.Write($"{bytes[i]} ");
            //}
            //Console.Write($"\n\n");

            // Створюємо сокет Tcp/Ip
            Socket sListener = new Socket(ipAddr.AddressFamily, SocketType.Stream,
ProtocolType.Tcp);
            sListener.Blocking = false;

            // Призначаємо сокет локальної кінцевої точки та слухаємо вхідні сокети
            try
            {
                sListener.Bind(ipEndPoint);
                sListener.Listen(10);

                // Починаємо слухати з'єднання
                while(true)
                {
                    //Console.WriteLine("Ожидаем соединение через порт {0}", ipEndPoint);

                    Socket handler = null;
                    try
                    {
                        // Спроба з'єднання (неблокуюча)
                        handler = sListener.Accept();
                    }
                    catch(SocketException ex)
                    {
                        //Console.WriteLine($"No incoming conn.. {ex.ErrorCode}");
                        //continue;
                        ;
                    }
                    if (handler != null)
                    {
                        // Зберегти новий сокет під час успішного підключення
                        CUser new_user = new CUser(NextUserID.ToString(), handler);
                        NextUserID++;
```

```
                        users.Add(new_user);
                        handler.SendTimeout = 100;
                        handler.ReceiveTimeout = 100;
                        //IPAddress.Parse(((IPEndPoint)s.LocalEndPoint).Address.ToString())
                        String s_ip =
((IPEndPoint)handler.RemoteEndPoint).Address.ToString();
                        String s_port =
((IPEndPoint)handler.RemoteEndPoint).Port.ToString();

                        IPEndPoint ep = (IPEndPoint)handler.RemoteEndPoint;
                        //Console.Write($"New client: {s_ip} - {s_port}\n");
                        //Console.Write($"New client: {ep.Address} - {ep.Port} -
{ep.ToString()}\n");

                        // Надіслати ідентифікатор та дані інших користувачів чату
                        String dt = "2" + new_user.id.ToString() + ",";
                        foreach(CUser us in users)
                        {
                            if (us == new_user)
                                continue;
                            dt += us.id.ToString() + "," + us.login + ",";
                        }
                        //Console.Write($"Clients: {dt}\n");
                        new_user.WriteString(dt);
                    }

                    foreach (var user in users)
                    {
                        bool BlockingState = user.conn.Blocking;
                        try
                        {
                            byte[] tmp = new byte[1];

                            user.conn.Blocking = false;
                            user.conn.Send(tmp, 0, 0);
                        }
                        catch (SocketException e)
                        {
                            // 10035 == WSAEWOULDBLOCK
                            if (!e.NativeErrorCode.Equals(10035))
                            {
                                //Console.Write($"Disconnected: {user.login}\n");
                                user.disconnected = true;
                            }
                        }
                        finally
                        {
                            user.conn.Blocking = BlockingState;
                        }
                        if (user.disconnected)
                        {
                            // Повідомити решту клієнтів про відключення
                            foreach(var us in users)
                            {
                                String dt = "4" + user.id;
                                if (us == user)
                                    continue;
                                us.WriteString(dt);
                            }
                            Console.Write($"User leave: {user.login}\n");
                            users.Remove(user);
                            break;
                        }
                        user.ReadData();
```

```
                        // Отримано логін від нового клієнта
                        if (user.data[0] == '1')
                        {
                            user.data = "0" + user.login;
                            String dt = "3" + user.id + "," + user.login + ",";

                            String s_ip =
((IPEndPoint)user.conn.RemoteEndPoint).Address.ToString();
                            String s_port =
((IPEndPoint)user.conn.RemoteEndPoint).Port.ToString();
                            Console.Write($"New user: {user.login}, address:
{s_ip}:{s_port}\n");

                            // Повідомити решту клієнтів про появу нового
                            foreach (var us in users)
                            {
                                if (us == user)
                                    continue;
                                us.WriteString(dt);
                            }
                        }

                        // Отримано нове повідомлення
                        if (user.data[0] == '7')
                        {
                            user.data = "0" + user.data.Substring(1);

                            // Виймаємо номер одержувача
                            int pos = user.data.IndexOf(',');
                            String whom_id = user.data.Substring(1, pos - 1);

                            // Вилучаємо саме повідомлення
                            String msg = user.data.Substring(pos + 1);

                            byte[] b_msg = Convert.FromBase64String(msg);

                            // Знаходимо клієнта за номером
                            CUser us = users.Find(x => x.id == whom_id);

                            // Надсилаємо повідомлення адресату
                            String dt = "8" + user.id + "," + msg;
                            us.WriteString(dt);
                            Console.Write($"Message:
{BitConverter.ToString(b_msg).Replace("-", "")}\n");
                            //for(int i = 0; i < b_msg.Length; i++)
                            //{
                            //    Console.Write($"{b_msg[i]:X}");
                            //    if ((i % 8) == 7)
                            //        Console.Write(" ");
                            //}
                        }

                        user.WriteData();
                        //if (user.data.IndexOf("<TheEnd>") > -1)
                        //{
                        //    Console.WriteLine("Сервер завершил соединение с клиентом.");
                        //}
                    }

                    //handler.Shutdown(SocketShutdown.Both);
                    //handler.Close();
                }
            }
```

```
            catch (Exception ex)
            {
                Console.WriteLine(ex.ToString());
            }
            finally
            {
                Console.ReadLine();
            }
        }
    }
}
```