МІЖНАРОДНИЙ ГУМАНІТАРНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, програмної інженерії та комп'ютерних наук

Кафедра комп'ютерних наук

Пояснювальна записка

до кваліфікаційної роботи
першого (бакалаврського) рівня

на тему **РОЗРОБЛЕННЯ СИМЕТРИЧНОЇ КРИПТОГРАФІЧНОЇ СИСТЕМИ З ВИКОРИСТАННЯМ НЕЛІНІЙНИХ ОПЕРАЦІЙ**

Виконав: студент 4 курсу, групи КІ-4
спеціальності
123 Комп'ютерна інженерія
Бадрі Аймен

Керівник Мазур Г. Д.

Рецензент Йона Л. Г.

Одеса – 2023 р.

ДОВІДКА

кафедри КН про виконану бакалаврську роботу
студента 4 курсу ФКПІ та КН групи КІ-4

Бадрі Аймен

на тему Розроблення симетричної криптографічної системи з використанням нелінійних операцій

Висновок нормоконтролера пояснювальна записка до кваліфікаційної роботи виконана з незначними порушеннями ДСТУ та вимог внутрішнього положення КПІ, які не впливають

Нормоконтролер

викл. каф ТІІ		29.06.23	І.В Китишина
(науковий ступінь, вчене звання, посада)		(підпис, дата)	(і. б. прізвище)

Висновок відповідального за наявність плагіату згідно з сертифікатом ID 101532 7939 унікальність роботи підтверджено

Відповідальна особа	викл. каф ТІІ	29.06.23	І.В Китишина
	(науковий ступінь, вчене звання, посада)	(підпис, дата)	(і. б. прізвище)

Попередня експертиза (захист) бакалаврської роботи
(бакалаврської роботи чи магістерської роботи)

студ. Бадрій Аймен (прізвище і.б.) проведена "29" 06 2023 р.

Висновки Виконана робота відповідає завданню, є всі проведених розрахунків підтверджує добрий кваліфікаційний рівень випускника. Метою роботи здійснено розробку програмної реалізації симетричної криптографічної системи. В роботі розроблено алгоритм криптографічної системи, що передбачає використання нелінійних перетворень за допомогою псевдовипадкових послідовностей. Написано текст програми, що реалізує такий алгоритм та проведено тестування. Робота бакалавра відповідає вимогам стандартам та рекомендується до захисту в ЕК.

Члени комісії

(підпис)	к.т.н., доц. Соловська І.М.
	(науковий ступінь, вчене звання, посада, прізвище і.б.)
(підпис)	к.т.н. доц. Григор'єва Т.Ю.
	(науковий ступінь, вчене звання, посада, прізвище і.б.)
(підпис)	к.т.н, доц. Песа І.Г.
	(науковий ступінь, вчене звання, посада, прізвище і.б.)

МІЖНАРОДНИЙ ГУМАНІТАРНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, програмної інженерії та комп'ютерних наук
Кафедра комп'ютерних наук
Освітній ступінь бакалавр
Галузь знань 12 Інформаційні технології
Спеціальність 123 Комп'ютерна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри КН
к.т.н., доц.
_____І.М. Соловська
"7" 04 2023 року

З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ РОБОТУ

Бадрі Аймену

1. Тема роботи: Розроблення симетричної криптографічної системи з використанням нелінійних операцій
керівник роботи ст. викладач Мазур Тетяна Олександрівна
затверджені наказом закладу вищої освіти від 07.квітня 2023 р. № 587

2. Строк подання студентом роботи 10.06.2023 р.

3. Вихідні дані до роботи:
1) провести аналіз існуючих способів шифрування інформації, що використовуються в сучасних симетричних криптографічних системах;
2) розглянути способи криптографічних перетворювань, що використовуються в симетричних криптографічних системах та провести оцінку їх впливу на стійкість таких систем
4. Зміст розрахунково-пояснювальної записки
Розділ 1: Аналіз сучасних криптографічних систем
Розділ 2: Принципи побудови криптосистем
Розділ 3: Розробка алгоритму криптографічної системи

5. Перелік графічного матеріалу (з зазначенням обов'язкових креслень)
Слайд 1 – Класифікація алгоритмів шифрування
Слайд 2 – Аналіз симетричних алгоритмів шифрування

Слайд 3 – Порівняльна характеристика симетричних алгоритмів шифрування
Слайд 4 – Схема алгоритму DES
Слайд 5 – Режимі блокового шифрування
Слайд 6 – Константи для лінійних конгруентних генераторів
Слайд 7 – Фреймворк Anaconda Navigator 2.4.0
Слайд 8 – Текст програми модуля ok_exe()
Слайд 9 – Текст програми модуля functions of mixing
Слайд 10 – Текст програми модуля switching of numbers
Слайд 11 – Використання нелінійних перетворень в програмі
Слайд 12 – Результат виконання програми

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Завдання видав	Завдання прийняв

7. Дата видачі завдання 25.11.2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Строк виконання етапів роботи	Примітка
1	Вступ	10.12.2022	
2	1 Аналіз сучасних криптографічних систем	25.01.2023	
3	2 Принципи побудови криптосистем	20.02.2023	
4	3 Розробка алгоритму криптографічної системи	29.05.2023	
6	Висновки та рекомендації	05.06.2023	
7	Перелік джерел посилання	06.06.2023	
8	Додаток А	07.06.2023	
9	Додаток Б	08.06.2023	

Студент _____ Бадрі Аймен
(підпис)

Керівник роботи _____ Мазур Г. Д.
(підпис)

ВІДГУК КЕРІВНИКА

на бакалаврську роботу студента Бадрі Аймена.

на тему: «Розроблення симетричної криптографічної системи з використанням нелінійних операцій»

Бакалаврська робота студента Бадрі Аймена присвячена розробці симетричної криптографічної системи з використанням нелінійних операцій. З розвитком і постійним впровадженням комп'ютерних комунікацій у різні сфери людської діяльності (наукові дослідження, електронна комерція, банківська справа і т.д.) усе більш гостро встає питання безпечного обміну даними через незахищені канали передачі. Одним із ефективних засобів захисту інформаціїзалишається її криптографічне перетворення.

У роботі розглянуті питання, пов'язані з побудовою симетричних криптографічних систем. Проведений аналіз основних принципів, які лежать в основі побудови симетричних систем, а також математичний апарат, на якому будуються ці принципи.

У роботі був запропонований алгоритм криптографічного перетворення. Його цінністю є простота процедур шифрування й, як наслідок більша швидкість шифрування. Студентом був розроблений програмний продукт, який реалізує запропонований алгоритм та проведено тестування. Текст програми написаний на мові Python 3 з використанням фреймворка Anaconda Navigator. Зміст пояснювальної записки написано ясно й грамотно. Під час оформлення роботи використовувалися комп'ютерні технології.

Завдання на бакалаврську роботу виконано. Під час виконання роботи студент Бадрі Аймен. показав уміння користуватися навчальною технічною літературою та навиками програмування.

Робота студента Бадрі Аймена відповідає вимогам щодо кваліфікаційних робіт бакалаврського рівня та заслуговує оцінки «добре».

Студент Бадрі Аймен заслуговує присвоєння кваліфікації бакалавр з комп'ютерної інженерії за заявленою спеціальністю 123 Комп'ютерна інженерія.

Керівник
ст. викл., кафедри комп'ютерної інженерії
та інноваційних технологій Г. Д. Мазур

РЕЦЕНЗІЯ

на бакалаврську роботу студента Бадрі Аймена.

з теми: «Розроблення симетричної криптографічної системи з використанням нелінійних операцій»

Бакалаврська робота виконана на 31 с. текстової частини та містить відповідні розділи згідно з завданням на бакалаврську роботу. Робота складається з трьох основних розділів, вступу, висновків та рекомендації, переліку джерел посилання та двох додатків. У вступі студент обґрунтовує вибір теми та її актуальність. Тема роботи актуальна, стосується сучасних проблем розробки симетричних криптографічних систем та підтримання високого рівня захисту конфіденційних даних, які передаються по відкритим каналам зв'язку.

За результатами проведеного аналізу студ. Бадрі Аймена було зроблено висновок, про доцільність побудови блокового алгоритму, що включає нелінійні операції (були обрані складні перестановки) над окремими частинами тексту. Такі криптографічні перетворення в значній мірі підвищують криптографічну стійкість алгоритму, оскільки дешифрування не передбачає можливості істотного зменшення ключового простору. Розроблений у роботі симетричний алгоритм шифрування заслуговує уваги і може бути рекомендовано до впровадження.

Текстова частина бакалаврської роботи викладена послідовно, чітко, технічно грамотно. Робота виконана відповідно до завдання.

До недоліків роботи варто віднести:

- відсутні пояснення про розподіл ключів по каналу зв'язку;
- немає суттєвих пояснень, щодо процесу розшифрування даних в алгоритмі.

Зазначені недоліки суттєво не знижують якості виконаної роботи.

Робота студента Бадрі Аймена відповідає вимогам щодо кваліфікаційних робіт бакалаврського рівня та заслуговує оцінки «добре».

Студент Бадрі Аймен заслуговує присвоєння кваліфікації бакалавр з комп'ютерної інженерії за заявленою спеціальністю 123 Комп'ютерна інженерія.

Рецензент
к.т.н., доцент кафедри кібербезпеки
та технічного захисту інформації
Державного університету інтелектуальних
технологій і зв'язку Л. Г. Йона

UNICHECK
by Turnitin

Ім'я користувача:
Анна Серединко

Дата перевірки:
23.06.2023 14:26:46 EEST

Дата звіту:
23.06.2023 14:27:40 EEST

ID перевірки:
1015683843

Тип перевірки:
Doc vs Internet + Library

ID користувача:
100001433

Назва документа: Бадрі Аймен

Кількість сторінок: 48 Кількість слів: 7347 Кількість символів: 54217 Розмір файлу: 1.30 MB ID файлу: 1015327939

Виявлено модифікації тексту (можуть впливати на відсоток схожості)

34.9%
Схожість

Найбільша схожість: 5.43% з Інтернет-джерелом (http://bdpu.org:8080/bitstream/123456789/443/1/Antonenko%20O.%20

34.2% Джерела з Інтернету	909	Сторінка 50
3.99% Джерела з Бібліотеки	26	Сторінка 56

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	7
Підозріле форматування	12 сторінок

РЕФЕРАТ

Текстова частина бакалаврської роботи: 31 с., 26 рисунків, 4 таблиці,2 додатка, 12 джерел.

ЗАШИФРУВАННЯ, РОЗШИФРУВАННЯ, ГАМУВАННЯ, КЛЮЧ, КРИПТОГРАФІЧНА СИСТЕМА, ШИФРТЕКСТ, РОЗСІЮВАННЯ, ПЕРЕМІШУВАННЯ

Об'єкт дослідження – симетричні алгоритми шифрування.

Мета роботи – розробка програмної реалізації симетричної криптографічної системи.

Метод дослідження – програмний, аналітичний з використання комп'ютерних технологій.

У бакалаврській роботі проведено аналіз існуючих способів шифрування інформації, що використовуються в сучасних симетричних криптографічних системах, також були розглянуті способи криптографічних перетворювань та проведено оцінку їх впливу на стійкість таких систем. В роботі розроблено алгоритм криптографічної системи, що передбачає використання нелінійних перетворень за допомогою псевдовипадкових послідовностей, написано текст програми, що реалізує такий алгоритм та проведено тестування.

Referat

The text part of the bachelor thesis: 31 pp., 26 figures, 4 tables, 2 appendices, 12 sources.

ENCRYPTION, DECRYPTATION, HAMMERING, KEY, CRYPTOGRAPHIC SYSTEM, CIPHERTEXT, SCATTERING, SHUFFLING

The object of research is symmetric encryption algorithms.
The purpose of the work is to develop a software implementation of a symmetric cryptographic system.
The research method is programmatic, analytical with the use of computer technologies.
The bachelor's work analyzed the existing methods of information encryption used in modern symmetric cryptographic systems, also considered the methods of cryptographic transformations and evaluated their impact on the stability of such systems. In the work, an algorithm of a cryptographic system, which involves the use of non-linear transformations using pseudorandom sequences, was developed, the text of the program implementing such an algorithm was written, and testing was carried out.

ЗМІСТ

ВСТУП

Стрімкий розвиток засобів обчислювальної техніки і відкритих мереж, сучасні методи накопичення, обробки і передачі інформації сприяли появі загроз, пов'язаних з можливістю втрати, розкриття, модифікації даних, що належать кінцевим користувачам. У зв'язку з цим постійно розширюється як в кількісному, так і в якісному відношенні коло завдань, що вирішуються в області інформаційної безпеки. Під інформаційною безпекою слід розуміти стан захищеності опрацьованих даних, що зберігаються і передаються в інформаційно-комунікаційних системах, від незаконного ознайомлення, перетворення і знищення, а також стан захищеності інформаційних ресурсів від дій, направлених на порушення їх працездатності.

Основу забезпечення інформаційної безпеки в інформаційно-комунікаційних системах складають криптографічні методи і засоби захисту інформації.

Науково-технічна революція останнім часом прийняла грандіозні масштаби в області інформатизації суспільства на базі сучасних засобів обчислювальної техніки, зв'язку, а також сучасних методів автоматизованої обробки інформації. Застосування цих засобів і методів прийняло загальний характер, а створювані при цьому інформаційно-обчислювальні системи і мережі стають глобальними як в сенсі територіальної розподіленості, так і в сенсі широти обхвату в рамках єдиних технологій процесів збору, передачі, накопичення, зберігання, пошуку, переробки інформації і видачі її для використання. Інформація в сучасному суспільстві – одна з найцінніших речей в житті, що вимагає захисту від несанкціонованого доступу осіб що не мають до неї доступу.

Сучасні блокові шифри, що працюють у потокових режимах, забезпечують швидкість шифрування до декількох Гб/с і, що немаловажне, мають доведені формально властивості безпеки. Крім того, міжнародним співтовариством проведена значна робота зі стандартизації блокових шифрів: AES, ДСТУ 7624:2014, ISO/IEC 18033. За останній час стандартизовані деякі режими шифрування, що забезпечують конфіденційність CTR (Counter Mode), автентичність OMAC (One-key Media Access Control), конфіденційність і цілісність повідомлень одночасно CCM, GCM.

Об'єктом дослідження у бакалаврської роботи є симетричні алгоритми шифрування.

Метою бакалаврської роботи є:

а) провести аналіз існуючих способів шифрування інформації, що використовуються в сучасних симетричних криптографічних системах;

б) розглянути способи криптографічних перетворювань, що використовуються в симетричних криптографічних системах;

в) розробити симетричний алгоритм шифрування, що передбачає використання нелінійних операцій.

1 АНАЛІЗ СУЧАСНИХ КРИПТОГРАФІЧНИХ СИСТЕМ

1.1 Основні відомості

Криптографія – це наука про способи перетворювання (шифрування) інформації з метою її захисту від неправочинних користувачів.

Криптографічна система – це система, реалізована програмно, апаратно або програмно-апаратно і здійснює криптографічне перетворення інформації з метою її захисту [1].

Більшість засобів захисту інформації базується на використанні криптографічних шифрів та процедур зашифрування та розшифрування. Відповідно до ДСТУ 7624:2014 під шифром розуміють сукупність зворотних перетворень безлічі відкритих даних на безліч зашифрованих даних, що задаються ключем і алгоритмом криптографічного перетворення.

Ключ – це правило, згідно з яким здійснюються зашифровування та розшифровування текстів. Надійність алгоритму шифрування залежить від довжини (кількості бітів) ключа. Множину можливих ключів називають простором ключів. При шифруванні треба знати шифр (алгоритм) і секретний ключ (тип перетворювання).

Основною характеристикою шифру є криптостійкість, яка визначає його стійкість до розкриття методами криптоаналізу. Зазвичай ця характеристика визначається інтервалом часу, необхідним для розкриття шифру.

До шифрів, які використовуються для криптографічного захисту інформації, пред’являється ряд вимог [2, 3]:

а) достатня криптостійкість (надійність закриття даних);

б) простота процедур зашифрування і розшифрування;

в) незначна надмірність інформації за рахунок шифрування;

г) нечутливість до невеликих помилок шифрування та ін.

В тій чи іншій мірі цим вимогам відповідають [3, 4]:

а) шифри перестановок;

б) шифри заміни;

в) шифри гамування;

г) шифри, засновані на аналітичних перетвореннях шифрованих даних.

Шифрування перестановкою полягає в тому, що символи шифруючого тексту переставляються за певним правилом в межах деякого блоку цього тексту.

При достатній довжині блоку, і межах якого здійснюється перестановка, і складному неповторюваному порядку перестановки можна досягти прийнятної для простих практичних додатків стійкості шифру.

Шифрування заміною (підстановкою) полягає в тому, що символи шифруючого тексту замінюються символами того ж або іншого алфавіту відповідно до заздалегідь зумовленої схемою заміни.

Шифрування гамуванням полягає в тому, що символи шифруючого тексту складаються з символами деякої випадкової послідовності, що іменується гамою шифру. Стійкість шифрування визначається в основному довжиною (періодом) повторюваної частини гами шифру. Оскільки за допомогою ЕОМ можна генерувати практично нескінченну гаму шифру, то даний спосіб є одним з основних для шифрування інформації в автоматизованих системах.

Шифрування аналітичним перетворенням полягає в тому, що шифруючий текст перетворюється за певним аналітичному правилом (формулою).

Криптографія являє собою сукупність методів перетворення даних, спрямованих на те, щоб зробити ці дані марними для противника. Такі перетворення дозволяють вирішити дві головні проблеми захисту даних: проблеми конфіденційності (шляхом позбавлення противника можливості отримати інформацію з каналу зв'язку) і проблему цілісності (шляхом позбавлення противника можливості змінити повідомлення так, щоб змінився його сенс, або ввести помилкову інформацію в канал зв'язку). Проблеми конфіденційності та цілісності інформації тісно пов'язані між собою, тому методи вирішення однієї з них часто застосовані для вирішення іншої.

Узагальнена схема криптографічної системи, що забезпечує шифрування переданої інформації, показана на рис. 1.1.

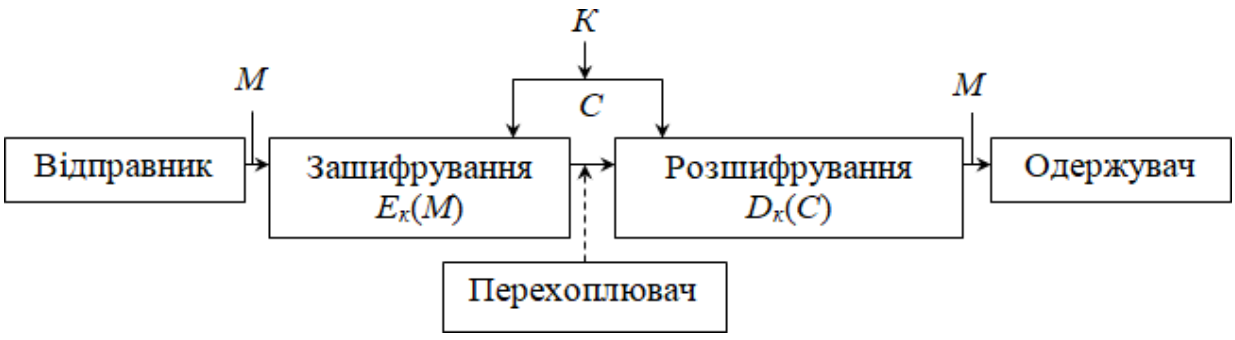

Рисунок 1.1 – Узагальнена схема криптосистеми шифрування

Відправник генерує відкритий текст вихідного повідомлення *M*, яке

повинно бути передане законному одержувачу по незахищеному каналу. За каналом стежить перехоплювач з метою перехопити і розкрити передане повідомлення. Для того щоб перехоплювач не зміг дізнатися зміст повідомлення *M*, відправник шифрує його з допомогою зворотнього перетворення і отримує шифртекст (або криптограму), який відправляє одержувачу. Функція шифрування *E* діє на відкритий текст, створюючи шифртекст $E_к(M) = C$. Позначимо шифртекст як *C*. Це теж двійкові дані, іноді того ж розміру, що й *M*, іноді більше. Законний одержувач, прийнявши шифртекст *C*, розшифровує його за допомогою зворотного перетворення і отримує вихідне повідомлення у вигляді відкритого тексту *M*.

Процес відновлення відкритого тексту по шифртексту є розшифруванням і виконується за допомогою функції розшифрування *D*: $D_к(C) = M$.

1.2 Класифікація алгоритмів шифрування

Криптографічні системи за типом алгоритму шифрування поділяються на дві великі групи: симетричні і асиметричні. Якщо для зашифрування і розшифрування інформації використовується один ключ, то такі алгоритми називається симетричними або криптосистеми з секретним ключем (рис. 1.2).

Рисунок 1.2 – Структурна схема криптосистеми з секретним ключем

В асиметричних криптосистемах використовується два різних ключі – один використовується для зашифровування, який називається відкритим (K_1), інший – для розшифрування, який називається секретним або особистим (K_2) (рис. 1.3). Розшифрування даних за допомогою відкритого ключа не є можливим. Ключ розшифрування не можна здобути з ключа зашифрування. Генератор ключів

розташовується на боці отримувача, щоби не пересилати секретний ключ K_2 незахищеним каналом зв'язку.

Рисунок 1.3 – Структурна схема криптосистеми з відкритим ключем

До елементарних криптографічних перетворень (шифрів) належать (рис. 1.4):

Рисунок 1.4 – Узагальнені перетворення в криптосистемі

– перетворення типу "підстановка" (substitution) символів повідомлення M, згідно з ключем K або гами Γ;

– перетворення типу "перестановка" (permutation) символів повідомлення згідно з ключем $K(\Gamma)$;

– зсув бітів, символів, слів згідно з ключем $K(\Gamma)$ тощо;

– гамування, тобто складання символів інформації M (повідомлення) з

ключем *К* чи гамою шифрування *Г*, за відповідним модулем;

– афінні перетворення бітів, символів, слів згідно з ключем *К*(*Г*) тощо;

– аналітичні перетворення згідно з правилами, формулами, залежностями згідно з ключем *К*(*Г*) тощо.

За типом обробки вхідної інформаційної послідовності криптографічні системи поділяються на потокові, у яких перетворюються всі повідомлення одразу, і блокові, у яких повідомлення обробляються у вигляді блоків певної довжини.

1.3 Стандарт шифрування DES

Алгоритм шифрування даних DES (Data Encryption Standard) – типовий блочний шифр, призначений для захисту від несанкціонованого доступу до важливої, але несекретної інформації в державних та комерційних організаціях США. Суть алгоритму DES зводиться до наступного. Передана до каналу зв’язку послідовність розбивається на блоки довжиною в 64 біти. Кожний з цих блоків шифрується незалежно від інших за допомогою 64-бітового ключа, в якому значущими є лише 56 біт (інші 8 біт – перевірні біти для контролю на парність).

Алгоритм DES використовує операції заміни, перестановок символів, розширення 32-бітових полублоків, операція циклічного зсуву та додавання за модулем 2. Узагальнена схема процесу зашифрування в алгоритмі DES має вигляд рис. 1.5. Процес зашифрування полягає в початковій перестановці бітів 64-бітового блоку, шістнадцяти циклах шифрування і, нарешті, зворотної перестановки бітів [5].

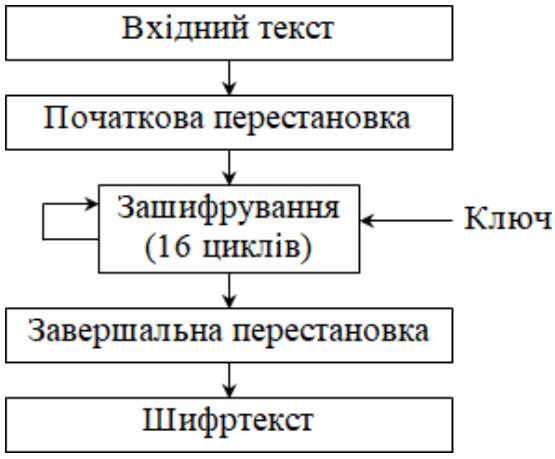

Рисунок 1.5 – Узагальнена схема шифрування в алгоритмі DES

Алгоритм DES (рис. 1.6) побудовано на базі мережі Фейстеля [1, 5].

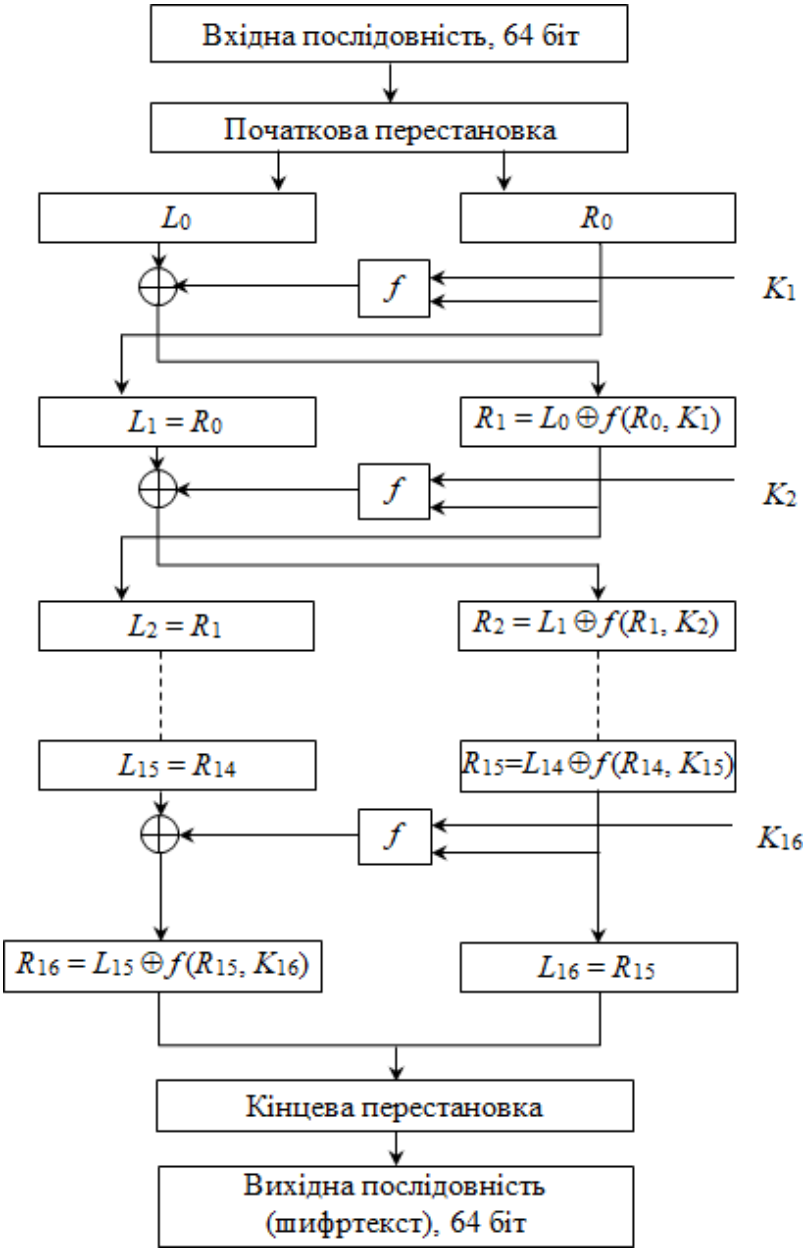

Рисунок 1.6 – Схема алгоритму DES

Процес шифрування здійснюється в такий спосіб. Вхідна послідовність $M = m_1, m_2, \ldots, m_{64}$ перетворюється блоком початкових перестановок на послідовність $IP(M) = m_{58}, m_{50}, \ldots, m_7$. Здобута після першої перестановки 64-бітова послідовність M розбивається навпіл на два 32-розрядних блоки L_0 та R_0. Потім виконується ітеративний процес шифрування, котрий складається з 16-ти кроків. Нехай M – результат i-тої ітерації [2]:

$$M_i = L_i R_i,$$

де $L_i = m_1, m_2, \ldots, m_{32}$; $R_i = m_{33}, m_{34}, \ldots, m_{64}$. У цьому разі результат ітерації описується формулами [2]:

$$L_i = R_{i-1}, \quad i = 1, \ldots, 16;$$
$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i), \; i = 1, \ldots, 16.$$

Функція f називається функцією шифрування. Її аргументами є послідовність здобута на попередньому кроці шифрування, та 48-бітовий ключ шифрування K, що формується за певним правилом з 64-бітового ключа шифрування K.

Після останнього раунду правий і лівий півблоки об'єднуються (i_B), після чого слід завершальна зворотна перестановка $IP^{-1}(i_B)$, блок $C = IP^{-1}(i_B)$ є не що інше, як DES (M, K). Усі перетворювання, виконувані в межах алгоритму DES, ілюструє схема подана на рис. 1.6.

Логіка обчислення значень f (R_{i-1}, K_i) представлена у вигляді структурної схеми, наведеної на рис. 1.7 [5].

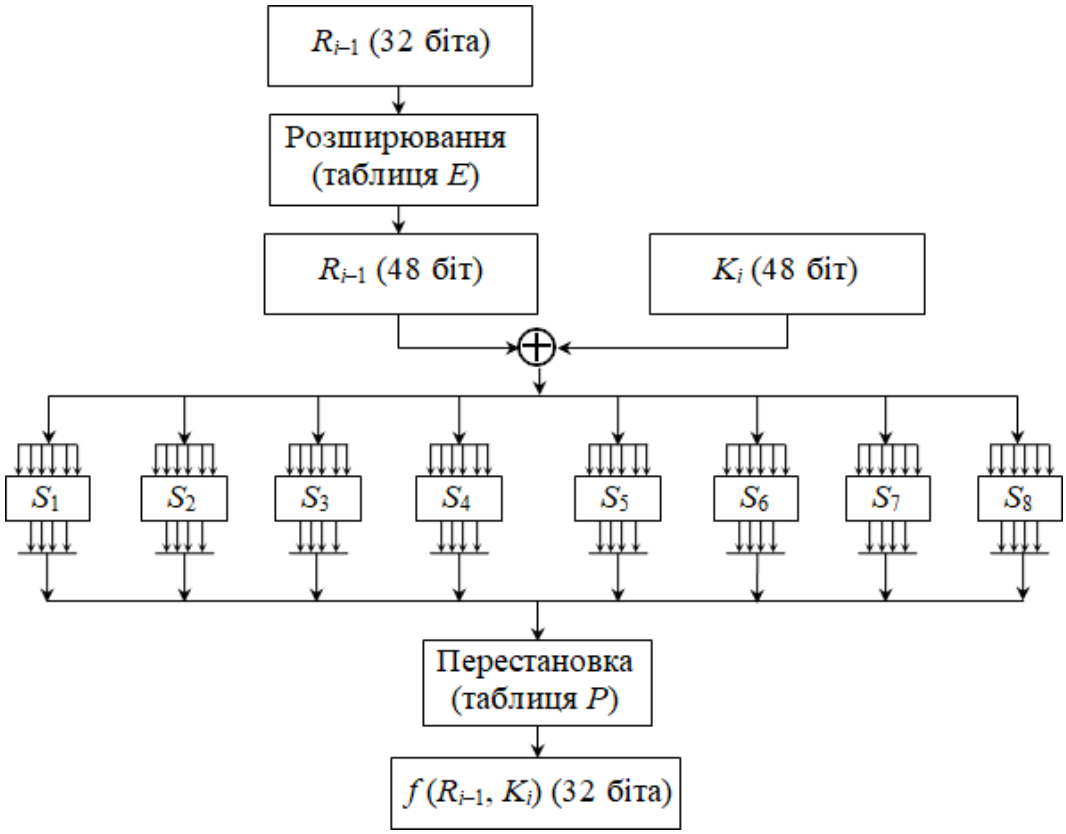

Рисунок 1.7 – Блок-схема обчислення функції f (R_{i-1}, K_i)

Обчислення починається з розширювання 32-бітового правого півблока R_{i-1}

до 48 бітів (за рахунок повторень бітів) відповідно до перестановки розширювання таблиця E (рис. 1.6). Отримані 48 біт R_{i-1} потім порозрядно додається за модулем 2 з 48 бітами раундового ключа K_i. Потім, результат розбивається на вісім 6-бітових векторів, що надходять на входи S-блоків (S-box substitution) S_1, ..., S_8. Кожен S-блок видає 4-бітовий вектор. Вісім 4-бітових векторів, що надходять з виходів S-блоків, утворюють 32-бітовий полублок, який піддається перестановці (таблиця P). Таким чином, операції, починаючи з перестановки розширювання E і закінчуючи табличній перестановці P утворюють раундову функцію шифрування $f(R_{i-1}, K_i)$.

На кожному кроці ітерації для функційного перетворювання використовується нове значення раундового ключа K_i. Схема алгоритму обчислювання ключів K_i наведена на рис. 1.8.

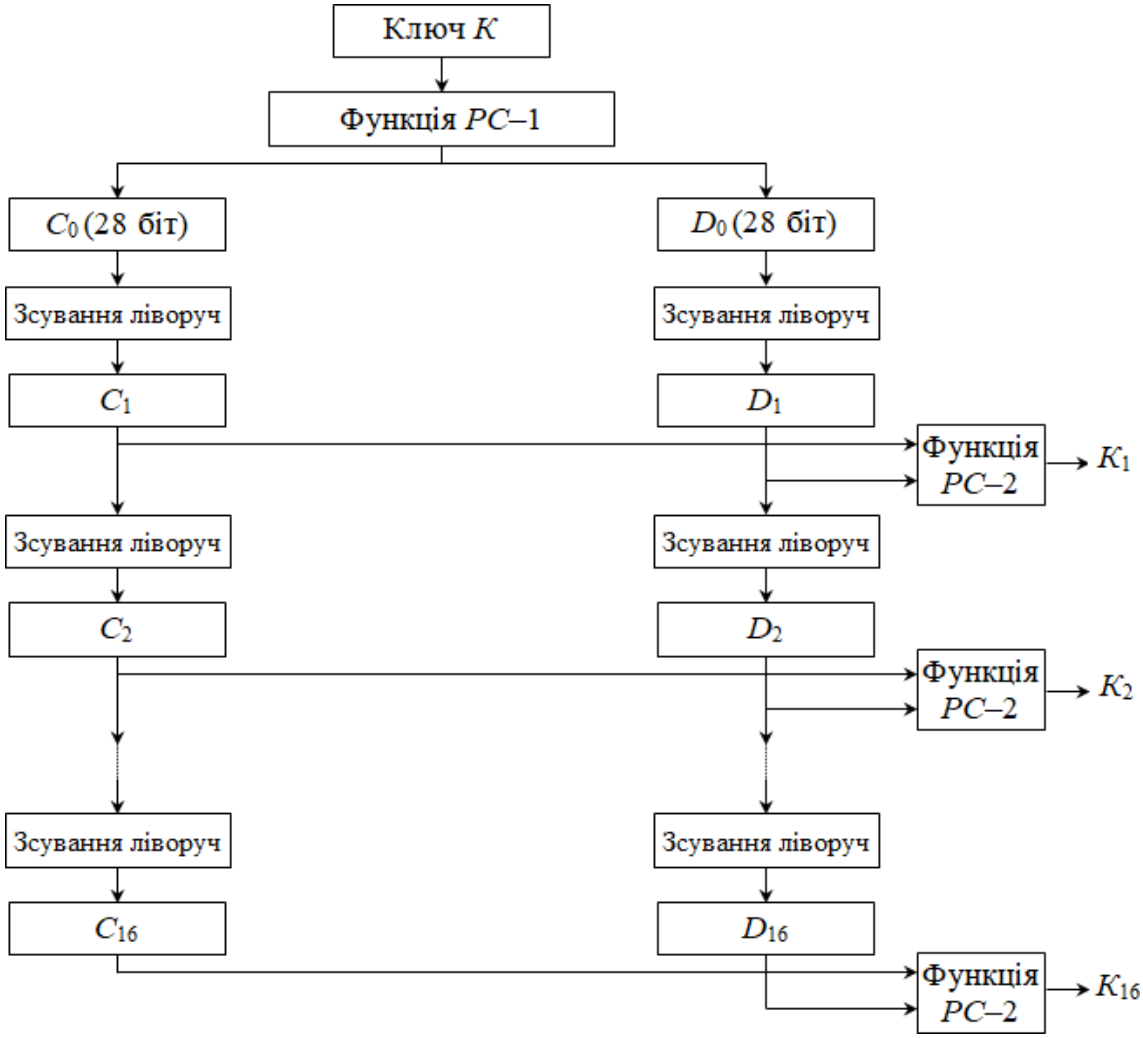

Рисунок 1.8 – Схема алгоритму обчислювання ключів K_i

Раундовий ключ створюється за таким алгоритмом:

а) із загального ключа шифрування K (64 біт) вилучається кожен восьмий біт у позиціях: 8, 16, 24, 32, 40, 48, 56, 64 – біти парності (ці біти не впливають на шифрування). Довжина ключа таким чином зменшується до 56 бітів;

б) біти ключа (56 біт) розділяються на два блоки C_0 і D_0 (по 28 біт) відповідно до стандартної таблиці вибір-перестановка *PC*–1 (Permuted Choice–1),.

в) на кожному i-му раунді C_i і D_i циклічно зсуваються вліво на 1 або 2 позиції, залежно від номера раунду. Операції зсування виконуються незалежно для послідовностей C_i і D_i;

г) після циклічного зсуву послідовностей C_i і D_i об'єднуються, щобстворити блок у 56 бітів, та за допомогою функції *PC*–2 (Permuted Choice–2) (перестановка із стисненням) змінює 56 бітів на 48 бітів раундового підключа K_i.

Перевагами алгоритму DES вважаються:

– висока швидкодія як в апаратній, так і в програмній реалізації;

– можливість використання одних і тих самих апаратних або програмних блоків як для зашифрування, так і для розшифрування інформації.

Основними недоліками алгоритму DES на сьогодні вважають:

– невелику довжину ключа, усього 56 бітів. При сучасному рівні розвитку комп'ютерних засобів така довжина ключа не може забезпечувати потрібний рівень захисту для деяких типів інформації;

– наявність слабких ключів, викликана тим, що для генерування ключової послідовності виконується два незалежних регістри зсуву. Приклади слабких ключів показано у табл. 1.1, де значення $[0]^{28}$ – вектор, що складається з 28 нульових бітів; значення $[1]^{28}$ – вектор, що складається з 28 одиничних бітів.

Слабкими ключами називається ключі K такі, що

$$M = \mathrm{DES}(\mathrm{DES}(M, K), K).$$

Таблиця 1.1 – Слабкі ключі для алгоритму DES

Слабкі ключі, hex	C_0	D_0
1F1F 1F1F 0E0E 0E0E	$[0]^{28}$	$[1]^{28}$
0101 0101 0101 0101	$[0]^{28}$	$[0]^{28}$
E0E0 E0E0 F1F1 F1F1	$[1]^{28}$	$[0]^{28}$
FEFE FEFE FEFE FEFE	$[1]^{28}$	$[1]^{28}$

При цьому результатом генерування будуть ключові послідовності, однакові з вихідним ключем, в усіх 16 раундах.

1.4 Режимі блокового шифрування

Для симетричних блокових алгоритмів визначені наступні основні режимі шифрування (modes of operation) [7]:

а) “електронна кодова книга” (Electronic Code Book – ECB). У цьому режимі виконання кожний блок відкритих даних зашифровується незалежно від інших блоків, із застосуванням того самого ключа шифрування. Типові застосування – безпечна передача поодиноких значень (наприклад криптографічного ключа);

б) “зчеплення блоків зашифрованих даних” (Cipher Block Chaining – CBC). У цьому режимі виконання вхід криптографічного алгоритму є результатом застосування операції XOR до наступного блока незашифрованих і попереднього блока зашифрованих даних. Типові застосування – загальна блокоорієнтована передача, автентифікація;

в) “зворотний зв’язок за зашифрованими даними” (Cipher Feedback – CFB). У цьому режимі виконання за кожного виклику алгоритму обробляється i бітів вхідного значення. Попередній зашифрований блок використовується як вхід в алгоритм; до i бітів виходу алгоритму й наступного незашифрованого блока з i бітів застосовується операція XOR, результатом якої є наступний зашифрований блок з l бітів. Типові застосування – потокоорієнтована передача, автентифікація;

г) “зворотний зв’язок за виходом” (Output Feedback – OFB). Цей режим виконання аналогічний режиму виконання CFB, за винятком того, що на вхід алгоритму при зашифруванні наступного блока подається результат зашифрування попереднього блока; тільки після цього виконується операція XOR із черговими бітами незашифрованих даних. Типові застосування – потокоорієнтована передача по зашумленому каналу (наприклад супутниковий зв’язок).

д) режим лічильника (Counter Mode – CTR). Цей режим виконання схожий на режим OFB, але замість використання випадкових унікальних значень вектора ініціалізації для генерації значень ключового потоку, цей режим використовує лічильник, значення якого додається до кожного блока відкритого тексту, який необхідно зашифрувати. Унікальне значення лічильника гарантує, що кожний блок об’єднується з унікальним значенням ключового потоку.

е) розповсюдженого зчеплення блоків зашифрованих даних (Propagating Cipher Block Chaining – PCBC). Режим PCBC подібний до режиму CBC, за виключенням того, що як попередній блок вхідних даних, так і попередній блок зашифрованих даних зазнають операції XOR із поточним блоком вхідних даних

перед зашифруванням або після розшифрування.

1.5 Порівняльна характеристика криптографічних систем

Вибір криптографічного алгоритму й режиму його використання залежить від особливостей переданої інформації (її цінності, обсягу, способу вистави, необхідної швидкості передачі і т.д.), а так само можливостей власників по захисту своєї інформації. Усе це істотно впливає на вибір криптографічного алгоритму й організацію захисту даних. Аналіз літературних джерел [1 – 5] показує, що кожний з найпоширеніших типів симетричних і асиметричні алгоритмів шифрування має свої переваги й недоліки. Тому при виборі того або іншого алгоритму шифрування або їх комбінації, необхідно враховувати в якій ситуації, який з алгоритмів працює краще. При цьому при виборі того або іншого алгоритму шифрування можуть ураховуватися такі показники як: довжина ключа, кількість раундів, розмір блоку, продуктивність, сумісність.

У результати проведеного аналізу найпоширеніші симетричні алгоритми шифрування наведено в табл. 1.2. Відповідно до табл. 1.2 можна зробити висновок про те, що найбільш стійкими є такі алгоритми як: AES, IDEA, ДСТУ 7624:2014. Так само дані алгоритми дозволяють використовувати ключі різної довжини, зокрема найпоширенішою довжиною є 128 біт, найменшої 56 має DES, а найбільшої 512 біт – ДСТУ 7624:2014. При цьому, чому довше ключ, тем надійніше захист і тем повільніше буде працювати алгоритм. А це означає, що в тому випадку якщо симетричний алгоритм буде використовуватися разом з асиметричним, те на перше місце виходить саме продуктивність, і довжина ключа не буде настільки критичною.

Таблиця 1.2 – Порівняльна характеристика алгоритмів шифрування

Симетричні алгоритми	Довжина ключа, біт	Розмір блоку, біт	Кількість раундів	Автор
DES	56	64	16	США
Triple DES	112	64	48	США
IDEA	128	64	8	Швейцарія J. Massey, X. Lai
AES	128, 192, 256	128	10, 12, 14	США
ДСТУ 7624:2014	128, 256, 512	128, 256, 512	10, 14, 18	Україна

2 ПРИНЦИПИ ПОБУДОВИ КРИПТОСИСТЕМ

2.1 Основні вимоги до криптографічних систем

Процес криптографічного захисту даних може здійснюватися як програмно, так і апаратно. Апаратна реалізація відрізняється істотно більшою вартістю, проте їй притаманні і переваги: висока продуктивність, простота, захищеність і т.д. Програмна реалізація більш практична, допускає відому гнучкість у використанні.

Для сучасних криптографічних систем захисту інформації сформульовані наступні загальноприйняті вимоги:

– зашифроване повідомлення повинно піддаватися читанню тільки при наявності ключа;

– число операцій, необхідних для визначення використаного ключа шифрування за фрагментом шифрованого повідомлення і відповідного йому відкритого тексту, має бути не менше загального числа можливих ключів;

– число операцій, необхідних для розшифровки інформації шляхом перебору різноманітних ключів повинно мати строгу нижню оцінку і виходити за межі можливостей сучасних комп'ютерів (з урахуванням можливості використання мережевих обчислень);

– знання алгоритму шифрування не повинно впливати на надійність захисту;

– незначна зміна ключа повинна приводити до істотної зміни виду зашифрованого повідомлення навіть при шифруванні одного і того ж вихідного тексту;

– незначна зміна вихідного тексту повинно приводити до істотної зміни виду зашифрованого повідомлення навіть при використанні одного і того ж ключа;

– структурні елементи алгоритму шифрування повинні бути незмінними;

– додаткові біти, що вводяться в повідомлення в процесі шифрування, повинен бути повністю та надійно сховані в зашифрованому тексті;

– довжина шифрованого тексту повинна бути рівною довжині вихідного тексту;

– не повинно бути простих і легко встановлюваних залежностей між ключами, послідовно використовуються в процесі шифрування;

– будь-який ключ із безлічі можливих повинен забезпечувати надійний

захист інформації;

– алгоритм повинен допускати як програмну, так і апаратну реалізацію, при цьому зміна довжини ключа не повинна призводити до якісного погіршення алгоритму шифрування.

2.2 Архітектура побудови блокових шифрів

Деякі з найраніших шифрсистем насправді спочатку розроблялися як генератори псевдовипадкових чисел, а не як генератори шифрувальної послідовності. Хоча ці завдання тісно один з одним пов'язані, але в більшості випадків до генерації псевдовипадкових чисел на основі даної техніки звертаються у зв'язку з проблемами статистичного тестування.

Лінійний конгруентний генератор − це генератор псевдовипадкової послідовності виду:

$$X_n = (aX_{n-1} + b) \bmod m,$$

де X_n − n-е число в послідовності, а X_{n-1} − попереднє число послідовності.

Параметри a, b і m − константи: a − коефіцієнт, b − приріст, а m − модуль. Ключем є початкове значення X_0.

Такий генератор має період, що не перевищує m. Якщо параметри a, b і m підібрані належним чином, то генератор буде генератором максимального періоду (іноді кажуть «максимальної довжини») з періодом $m - 1$. Для цього, наприклад, параметр b повинен бути взаємно простий з m.

У табл. 2.1 дається список констант для лінійних конгруентних генераторів [5]. Всі вони породжують генератори максимального періоду і, що ще більш важливо, проходять спектральні тести на випадковість для розмірностей 2, 3, 4, 5 і 6. Їх підбір зроблений на основі принципу максимального множення, що не викликає переповнення певної довжини слова. Головні переваги лінійних конгруентних генераторів в тому, що вони швидко працюють і вимагають мало операцій на біт послідовності. Найбільший недолік − передбачуваність таких послідовностей. Перевагою лінійних конгруентних генераторів є їх простота й висока швидкість отримання псевдовипадкових значень. Вони ефективні і в більшості використовуваних емпіричних тестах. Д. Кнут [5] розглянув варіацію генератора, коли модуль m є ступенем двійки, а вихідними бітами є тільки старші біти чисел послідовності.

Таблиця 2.1 – Константи для лінійних конгруентних генераторів

Переповнення при:	*a*	*b*	*m*
2_{20}	106	1281	6075
2_{21}	211	1663	7875
2_{22}	421	1663	7875
2_{23}	430	2531	11979
	936	1399	6655
	1366	1283	6075
2_{24}	171	11213	53125
	859	2531	11979
	419	6173	29282
	967	3041	14406
2_{25}	141	28411	134456
	625	6571	31104
	1541	2957	14000
	1741	2731	12960
	1291	4621	21870
	205	29573	139968
2_{26}	421	17117	81000
	1255	6173	29282
	281	28411	134456
2_{27}	1093	18257	86436
	421	54773	259200
	1021	24631	116640
	1021	25673	121500
2_{28}	1277	24749	117128
	741	66037	312500
	2041	25673	121500
2_{29}	2311	25367	120050
	1807	45289	214326
	1597	51749	244944
	1861	49297	233280
	2661	36979	175000
2_{30}	3877	29573	139968
	3613	45289	214326
	1366	150889	714025
2_{31}	8121	28411	134456
	4561	51349	243000
	7141	54773	259200
2_{32}	9301	49297	233280
	4096	150889	714025
2_{33}	2416	374441	1771875
2_{34}	17221	107839	510300
2_{35}	84589	45989	217728

Для криптографічних цілей необхідно, щоб числа, що генеруються були непередбачувані; якщо модуль m відомий, то параметри a і b нескладно підібрати по двом сусіднім числах такій послідовності.

Перевагою лінійних конгруентних генераторів є їх простота й висока швидкість отримання псевдовипадкових значень. Лінійні конгруентні генератори застосовуються під час розв’язання задач моделювання та математичної статистики, однак з криптографічною метою їх можна рекомендувати для використання у реальних шифрсистемах.

3 РОЗРОБКА АЛГОРИТМУ КРИПТОГРАФІЧНОЇ СИСТЕМИ

3.1 Постанова задачі

Завдання в роботі поставлена таким чином, щоб розробити алгоритм шифрування в розрахунку на потенційного зловмисника, який не є професійним криптоаналітиком. Всі криптосистеми високої стійкості використовують процедури заміни, перестановок і функціональних перетворень. У роботі був проведений аналіз симетричних алгоритмів шифрування, які мають нелінійну структуру і за рахунок цього досягається висока стійкість, але ускладнюється програмна реалізація і збільшується потреба в обчислювальних ресурсах. На основі проведеного аналізу алгоритмів шифрування зроблено висновок про доцільність використання лінійного і блочного алгоритму шифрування для спрощення програмної реалізації. Для позбавлення від основного недоліку блокових алгоритмів, а саме шифрування однакових блоків однаково в алгоритм внесена процедура режиму зворотний зв’язок по шифру.

Генерування випадкових послідовностей із заданим законом ймовірності і перевірка їх адекватності − одні з найважливіших проблем сучасної криптології. Генератори випадкових послідовностей використовуються в існуючих криптосистемах для генерації ключової інформації і заданою ряду параметрів криптосистем. Наукова та практична значущість цієї проблеми настільки велика, що їй присвячені окремі монографії в області криптології, організовуються розділи в наукових журналах «Journal of Cryptology», «Cryptologia» і спеціальні засідання на міжнародних наукових конференціях «Eurocrypt», «Asiacrypt», «Crypto» та ін.

Перед розробкою алгоритму програми спочатку встановимо основні вимоги, котрим повинні задовольняти криптостійкі генератори псевдовипадкових послідовностей або гами. Період гами повинен бути достатньо великим для шифрування повідомлень різної довжини. Гамма повинна бути важко передбачуваною. Це означає, що якщо відомі тип генератора і фрагмент гами, то неможливо передбачити наступний за цим фрагмент біт гами або попередній фрагмент біт гами. Генерування гами не повинно бути пов’язане з великими технічними і організаційними труднощами.

Найважливіша характеристика генератора псевдовипадкових чисел − це інформаційна довжина його періоду, після якого числа будуть або просто

повторюватися, або їх можна буде передбачити. Ця довжина практично визначає можливе число ключів криптосистеми. Чим ця довжина більше, тим складніше підібрати ключ.

Друга із зазначених вище вимог пов'язана з наступною проблемою: на підставі чого можна зробити висновок, що гама конкретного генератора дійсно є непередбачуваною? Поки в світі немає універсальних та практично перевірених критеріїв для перевірки цієї властивості. Інтуїтивно випадковість сприймається як непередбачуваність.

Щоб гама вважалася випадковою і непередбачуваною як мінімум необхідно, щоб її період був дуже великим, а різні комбінації біт певної довжини рівномірно розподілялися по всій її довжині. Цю вимогу статистично можна тлумачити і як складність закону генерації псевдовипадкової послідовності чисел. Якщо за досить довгий відрізок цієї послідовності не можна ні статистично, ні аналітично визначити цей закон генерації, то в принципі цим можна задовольнитися.

І, нарешті, третя вимога повинна гарантувати можливість практичної реалізації генераторів псевдовипадкових послідовностей з урахуванням необхідної швидкодії та зручності практичного використання.

Характеристики генераторів випадкових чисел. Послідовності випадкових чисел, що формуються тим чи іншим генератором псевдовипадкових чисел (ГПЧ), повинні задовольняти ряд вимог. По-перше, числа повинні вибиратися з певної множини (найчастіше це дійсні числа в інтервалі від 0 до 1 або цілі від 0 до N). По-друге, послідовність повинна підкорятися певному розподілу на заданій множині (найчастіше розподіл рівномірний). Необов'язковим є вимога відтворюваності послідовності. Якщо ГПЧ дозволяє відтворити заново послідовність, яка колись вже була сформована, налагодження програм з використанням такого ГПЧ значно спрощується. Крім того, вимога відтворюваності часто висувається при використанні ГПЧ в криптографії. Оскільки псевдовипадкові числа не є дійсно випадковими, якість ГПЧ дуже часто оцінюється за «випадковістю» одержуваних чисел. У цю оцінку можуть входити різні показники, наприклад, довжина циклу (кількість ітерацій, після якого ГПЧ зациклюється), взаємозалежності між сусідніми числами (можуть виявлятися за допомогою різних методів теорії ймовірностей і математичної статистики) і т.п.

Для того, щоб максимально ускладнити можливість статистичного криптоаналізу метод шифрування запропонованого алгоритму повинен містити операції, які забезпечують «перемішування» і «розсіяння» символів, а також операцій перетворення ключа.

Розсіяння – це властивість шифру, при якому один символ (біт) початкового тексту впливає на шифрування декількох символів (біт) шифртексту, оптимально – на всі символи в межах одного блоку. Якщо дана умова виконується, то при шифруванні двох блоків даних з мінімальними відмінностями між ними повинні виходити зовсім несхожі один на одного блоки шифртексту. Така сама картина повинна мати місце і для залежності шифртексту від ключа – один символ (біт) ключа повинен впливати на шифрування декількох символів (біт) шифртексту [1].

Перемішування – це властивість шифру ховати залежності між символами початкового тексту і шифртексту [1]. Якщо шифр досить добре "перемішує" біти початкового тексту, то відповідний шифртекст не містить ніяких статистичних і, тим більше, функціональних закономірностей для стороннього спостерігача, який володіє обмеженими обчислювальними ресурсами.

Якщо шифр задовольняє переліченим властивостям, то це означає, що при зміні в початковому блоці хоч би одного символу, всі символи зашифрованого блоку, з вірогідністю 1/2, отримають нові значення.

Запропонуємо для створення надійних шифрів будувати їх з простих шифрів. Під простим перетворенням, яке шифрує, будемо розуміти таке перетворення, яке реалізується логічною схемою або програмно декількома обчислювальними командами. Таким чином, можна виділити наступні групи простих криптографічних перетворень:

а) перестановка – полягає в перестановках структурних елементів шифруючого блоку даних – біт, символів, цифр;

б) заміна – полягає в заміні одних значень на інші по індексній таблиці. Заміні піддаються групи елементів шифруючого блоку – біт або символів;

в) функціональні перетворення – полягають у виконанні зрушень логічних та арифметичних операцій над елементами даних.

Для того, щоб побудувати надійний шифр з елементарних перетворень, які шифрують, їх необхідно каскадувати. У такого роду криптографічних системах використовують, як правило, не один ключ K, а декілька ключів K_i, $i = 1, 2, \ldots, n$, по числу кроків шифрування. Вся сукупність проміжних ключів генерується з одного, головного, ключа K до шифру. Додатково слід сказати, що біти кожного з сформованих ключів повинні бути рівноімовірні і незалежні.

Не має сенсу комбінувати дві однотипні операції підряд. Максимальний ефект досягається при чергуванні процедур різного типу. Це пояснюється тим, що, наприклад, дві перестановки можна представити як одну, результуючу

перестановку, а кількість операцій на цю процедуру буде удвічі більша.

Для того, щоб шифр був не тільки стійким, але і зручним в реалізації, повинні бути виконані наступні умови:

а) операції зашифровування і розшифровування повинні бути близькими настільки, щоб могли бути виконані одним і тим самим апаратним або програмним модулем;

б) бажано мати ключ розміром в 8 Байт (64 розряди), з метою підвищення швидкодії алгоритму і мінімізації появи помилок при його використанні. Але виходячи зі слів Шенона [1] про нескінченну величину ключа, зрозуміло, що ключ повинен мати як можна більшу розрядність, тому в процесі шифрування ключ повинен перетворюватися, що приведе до його фактичного збільшення;

в) реалізація шифру (код програми і постійних даних) повинна бути досить компактною для того, щоб "вміститися" на мікроконтролерах з відносно невисоким об’ємом пристрою, що запам’ятовує. Зі всього видно, що операція перестановки зворотна за визначенням. Прийнявши до уваги вище сказане, розробимо криптографічний алгоритм шифрування.

3.2 Описання алгоритму та реалізація програми

Текст програми написаний на мові Python 3 з використанням фреймворка Anaconda Navigator 2.4.0 (рис. 3.1).

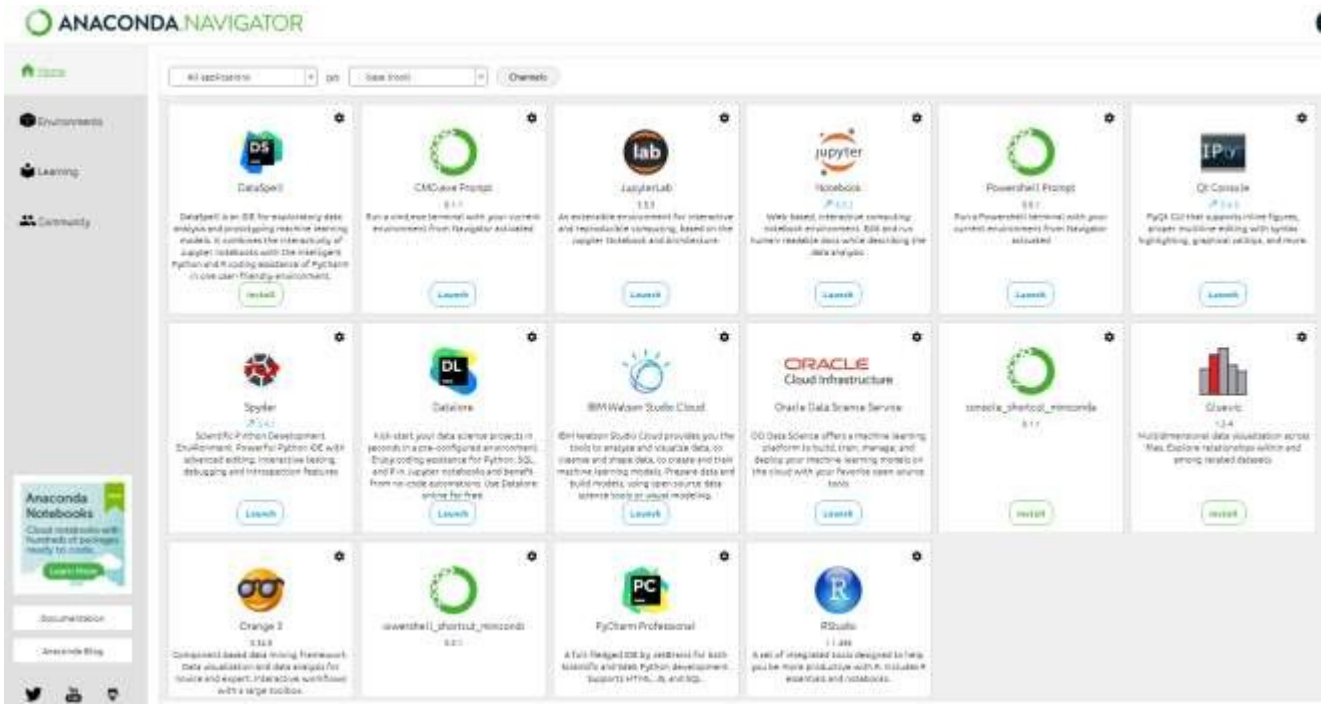

Рисунок 3.1 – Фреймворк Anaconda Navigator 2.4.0

Фреймворк Anaconda Navigator програмне середовище, яке спрощує та прискорює створення програмного забезпечення, у структурі якої включено відкрита крос-платформна Spyder IDE (integrated development environment – інтегроване середовище розробки). Spyder IDE інтегрується з рядом відомих пакетів у стеку Python, включаючи NumPy, SciPy, а також інше програмне забезпечення з відкритим кодом (рис. 3.2).

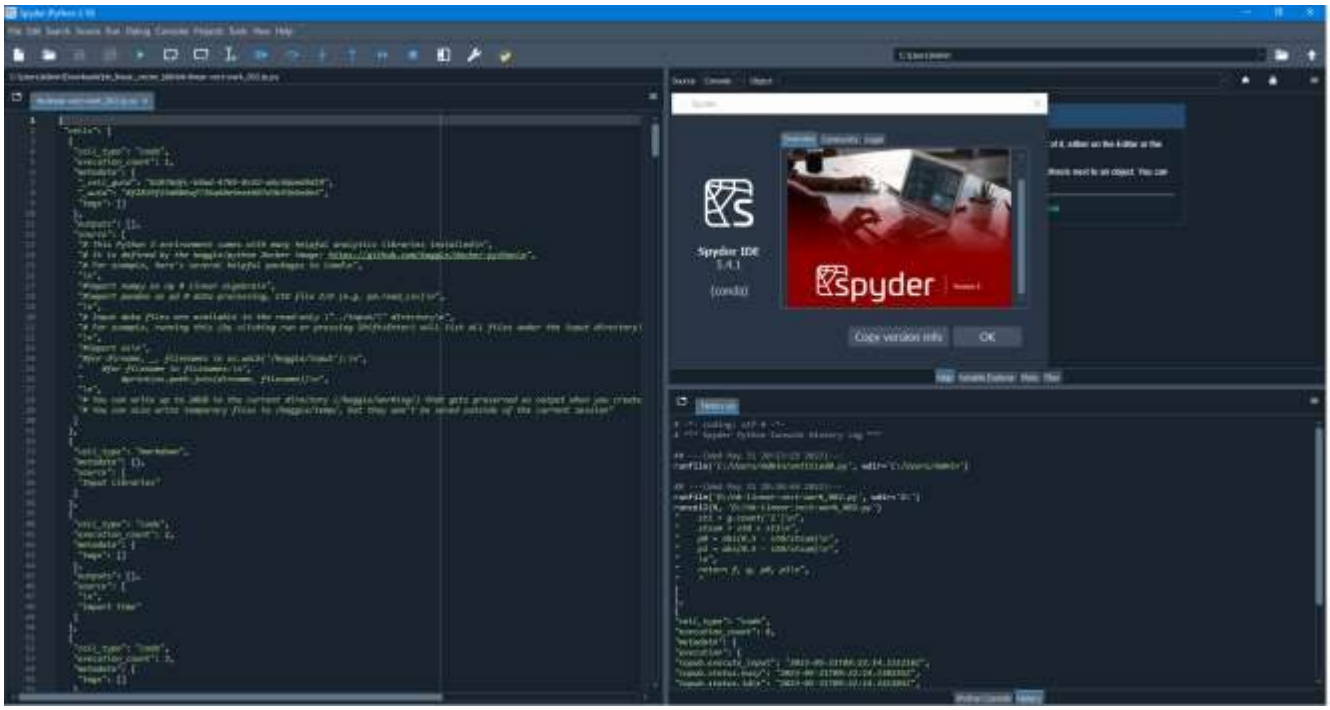

Рисунок 3.2 – Інтерфейс крос-платформи Spyder IDE

Тестування та перевірка програми шифрування виконано за допомогою Python Jupyter Notebook та Jupyterlab (рис. 3.3), це програми з відкритим вихідним кодом, в яких можна відразу побачити результат виконання коду.

Рисунок 3.3 – Програми тестування Python Jupyter Notebook, Jupyterlab

Головна відмінність від традиційних інструментів розробки – можливість розбити код на частини та виконувати їх окремо. Наприклад, можна написати одну функцію і одразу перевірити, як вона працює, не запускаючи інші фрагментів коду. Також можна змінювати порядок виконання коду.

Програма починається з введення необхідних початкових даних, які повинен ввести користувач (рис. 3.4). Спочатку програмі необхідно вказати, який тип операції буде проводитися – зашифровування або розшифрування. Далі потрібно ввести текст для зашифровування або розшифрування. Далі пропонується ввести ключ, довжина якого становить 8 символів. Символи можуть бути як літерного регістра, так і цифрового. Результат виконання модуля ok_exe() показано на рис. 3.5.

```
def ok_exe():
    A = True
    while A == True:
        a = input("Модуль зашифрування та розшифрування. Почати роботу: натиснить 1, завершити роботу:  0 ")
        if a == '0':
            print("Дякуємо за співпрацю!")
            A = False
            break
        elif a == '1':
             #switching of regime
            AR = regime()
            if AR == True:
                tx0 = int(time.time())
                print("Режим Зашифрування")
                ms_base = input_txt()
                print('ms_base', ms_base)
                print(len(ms_base))
                ms_sm = mixing(rt, ms_base)
                print('ms_sm',ms_sm, len(ms_sm))
                ap = input_pass()
                print(ap)
                aps = conv_pass(ap)
                print(aps)
                A_def = a_defn()
                print(A_def)
                A_one, A_two = switch_n(A_base)
                A_three = ['A_three',A_one[1] + A_two[1] + 1, A_one[2] + A_two[2]+1, A_one[3] + A_two[3]+1]
                print(A_three[1:])
                spa = a1_2(A_one, A_two, aps)
                print(spa)
                th,X,p0,p1 = conv_test_1(A_def, A_three, spa)
                ##print(th)
                print("Знайдене число X та окремі відхилення від 0.5 ",X,p0,p1)
                txt_mix, bin_mix, x0,x1 = conv_mod_2(ms_sm, th, 16)
                txtd_mix = list_to_str_CHR(txt_mix)
                print(txt_mix, txtd_mix)
                print(bin_mix)
                print(x0,x1)
                print("Ітогове зашифроване повідомлення",str_to_list_ORD(txtd_mix))
            else:
                print("Режим Розшифрування")
                ms_rep_str = decription()
                print("Первинний текст", ms_rep_str)
        else:
            pass
```

Рисунок 3.4 – Текст програми модуля ok_exe()

```
A = ok_exe()
#print(A)
```

```
Модуль зашифрування та розшифрування. Почати роботу: натиснить 1, завершити роботу:  0   1
Виберіть режим. Зашифрування: 0, Розшифрування: 1  0
Режим Зашифрування
Введить текст  Програма тестування
ms_base [1055, 1088, 1086, 1075, 1088, 1072, 1084, 1072, 8226, 1090, 1077, 1089, 1090, 1091, 1074, 1072, 1085, 1085, 1103, 822
6, 8226, 8226, 8226, 8226, 8226, 8226, 8226, 8226, 8226, 8226, 8226, 8226]32
ms_sm [1072, 1090, 1090, 1084, 1075, 1055, 1091, 1077, 1072, 1088, 1088, 1074, 1089, 8226, 1072, 1086, 8226, 8226, 8226, 8226,
8226, 1085, 8226, 8226, 8226, 8226, 1085, 8226, 8226, 8226, 8226, 1103] 32
Будь ласка, введить пароль з 8 символів  12345678
12345678
[49, 50, 51, 52, 53, 54, 55, 56]
Введить максимальну величину відхилення після підсумкового зашифрування(звичайно від 0.001 до 0.01) 0
0.0
```

Рисунок 3.5 – Результат виконання модуля ok_exe()

Структурна схема криптографічного перетворення даних, на якій базується алгоритм роботи програми зображена на рис. 3.6.

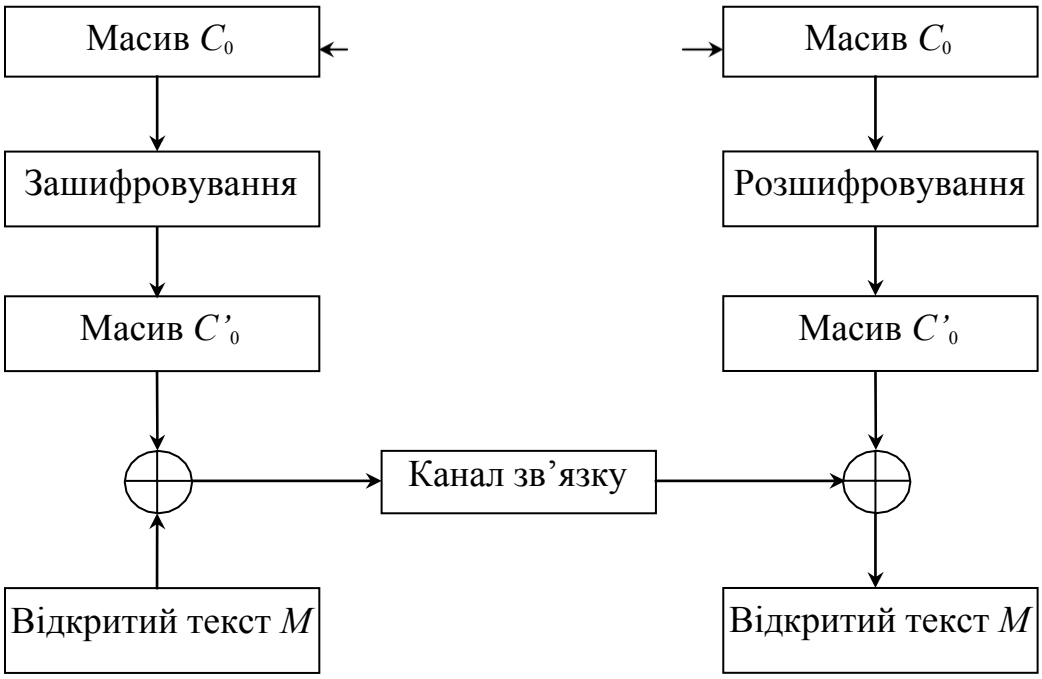

Рисунок 3.6 – Загальна структурна схема криптографічного перетворення даних

Розглянемо більш детально використання нелінійних перетворень в програмі.

а) Початкова перестановка. Як видно з програм (рис. 3.4), перед початком зашифрування додатково реалізовано початкову перестановку (ms_sm) вхідного тексту модуль functions of mixing показано на рис. 3.7.

б) Формування C_0 (рис. 3.8). Формування початкової вставки C_0 виконується за допомогою двох генераторів псевдовипадкових чисел і ключа K_i. Дана операція ускладнює завдання пошуку ключа повним перебором. Також відбувається розширення початкової вставки C_0 з 8 байт до 16-ти. Параметри a, b і

m лінійних генераторів представлені у табл. 2.1 (listing of points value a, b, m показано на рис. 3.9). Текст програми модуля switching of numbers показано на рис. 3.10. Також на етапі перетворення C_0 виконується додавання по модулю 2 випадкової величини (рис. 3.11).

```
#functions of mixing
def perem(a,b,m):
    ft = []
    for i in range (m):
        h = (a *i + b) % m
        ft.append(h)
    return ft
rt = perem(13,15,16)
print(rt)

def mixing(rt,ms_base):
    ms_sm = []
    k = 0
    w = len(ms_base)/16
    while k < w:
        for i in rt:
            ms_sm.append(ms_base[i + k*16])
        k += 1
    return ms_sm

[15, 12, 9, 6, 3, 0, 13, 10, 7, 4, 1, 14, 11, 8, 5, 2]
```

Рисунок 3.7 – Текст програми модуля functions of mixing

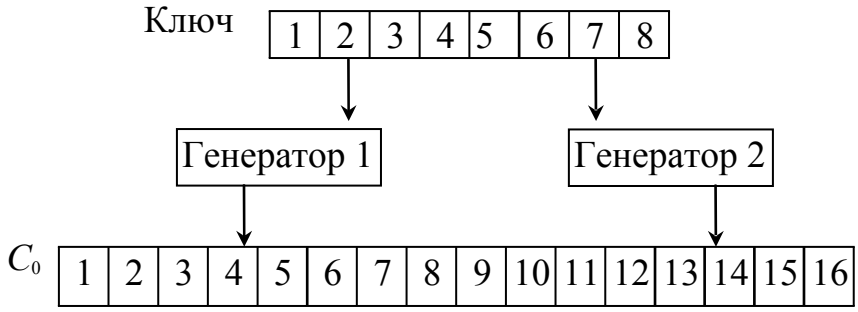

Рисунок 3.8 – Формування початкової вставки C_0

```
#listing of points (value max, a, b, m)
A_base = [['2**20',106,1281,6075],
          ['2**21',211,1663,7875],
          ['2**22',421,1663,7875],
          ['2**23',430,2531,11979],['2**23',936,1399,6655],['2**23',1366,1283,6075],
          ['2**24',171,11213,53125],['2**24',859,2531,11979],['2**24',419,6173,29282],['2**24',967,3041,14406],
          ['2**25',141,28411,134456],['2**25',625,6571,31104],['2**25',1541,2957,14000],['2**25',1741,2731,12960]
          ['2**26',421,17117,81000],['2**26',1255,6173,29282],['2**26',281,28411,134456],
          ['2**27',1093,18257,86436],['2**27',421,54773,259200],['2**27',1021,24631,116640],['2**27',1021,24673,121500],
          ['2**28',1277,24749,117128],['2**28',741,66037,312500],['2**28',2041,25673,121500],['2**29',2311,25367,120050],
          ['2**29',1807,45289,214326],['2**29',1597,51749,244944],['2**29',1861,49297,233280],['2**29',2661,36979,175000],
          ['2**30',3877,29573,139968],['2**30',3613,45289,214326],['2**30',1366,150889,714025],
          ['2**31',8121,28411,134456],['2**31',4561,51349,243000],['2**31',7141,54703,259200],
          ['2**32',9301,49297,233280],['2**32',4096,150889,714025],
          ['2**33',2416,374441,1771875],
          ['2**34',17221,107839,510300],
          ['2**35',84589,45989,217728]]
```

Рисунок 3.9 – Текст модуля listing of points

```
#switching of numbers
def switch_n(A_base):
    k = 0
    A = True
    while A == True:
        print("Для першого генератора виберить номер зі списку")
        for i in range(len(A_base)):
            print(i, A_base[i][:])
        k  = int(input())
        if k < 0 or k > len(A_base):
            print("Спробуйте ще раз")
        else:
            A = False
    A_one = A_base[k][:]
    k2 = 0
    while A == False:
        print("Для другого генератора виберить інший номер зі списку")
        for i in range(len(A_base)):
            print(i, A_base[i][:])
        k2 = int(input())
        if k2 < 0 or k2 > len(A_base):
            print("Спробуйте ще раз")
        elif k2 == k:
            print("Однакові значення. Спробуйте ще раз")
        else:
            A == False
            break
    A_two = A_base[k2][:]
     return A_one, A_two
```

Рисунок 3.10 – Текст модуля switching of numbers

```
# calculation A_three
#A_three = ['A_three',A_one[1] + A_two[1] + 1, A_one[2] + A_two[2]+1, A_one[3] + A_two[3]+1]
#print(A_three[1:])
```

```
#calculation A_three_b
A_three_b = ['A_three_b',A_one_b[1] + A_two_b[1] + 1, A_one_b[2] + A_two_b[2]+1, A_one_b[3] + A_two_b[3]+1]
print('A_three_b',A_three_b)
```

Рисунок 3.11 – Текст модуля calculation A_three

Вибір параметрів генератора у програмі показано на рис. 3.12

```
Для першого генератора виберить номер зі списку
0 ['2**20', 106, 1281, 6075]
1 ['2**21', 211, 1663, 7875]
2 ['2**22', 421, 1663, 7875]
3 ['2**23', 430, 2531, 11979]
4 ['2**23', 936, 1399, 6655]

Для другого генератора виберить інший номер зі списку
0 ['2**20', 106, 1281, 6075]
1 ['2**21', 211, 1663, 7875]
2 ['2**22', 421, 1663, 7875]
3 ['2**23', 430, 2531, 11979]
4 ['2**23', 936, 1399, 6655]
```

Рисунок 3.12 – Приклад роботи програми для вибору типу генератора

в) Перетворення правої половини блоку відбувається за допомогою додавання по модулю 2 восьми перших байт з восьми останніми байтами. Результат операції записується в праву частину C_0 з 8-го по 15 байт (рис 3.13, 3.14).

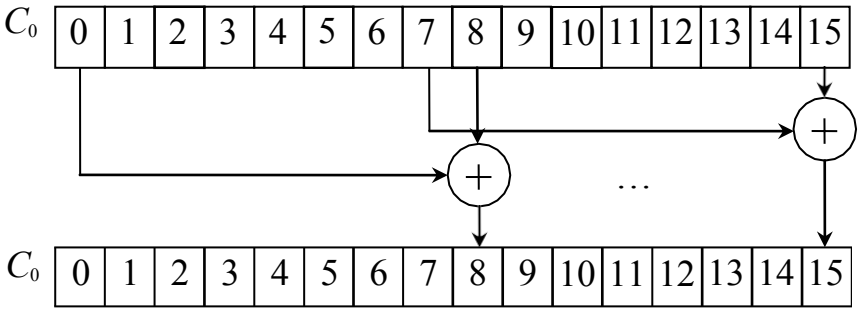

Рисунок 3.13 – Перетворення правої половини блоку C_0

```
#special filling (0:7 - A_two, 8:15 - A_one)
def a1_2(A_one, A_two, aps ):
    c = []
    d = []
    for i in range(8):
        c.append(calc(A_two, aps[i]))
        d.append(calc(A_one, aps[i]))
    c = c+d

    return c
```

Рисунок 3.14 – Текст модуля special filling

г) Перетворення лівої половини блоку C_0 відбувається аналогічно як і у правій частині, тільки додавання восьми останніх біт виконується в зворотному порядку, а результат записується в 0-й біт по 7-тий C_0 (рис 3.15).

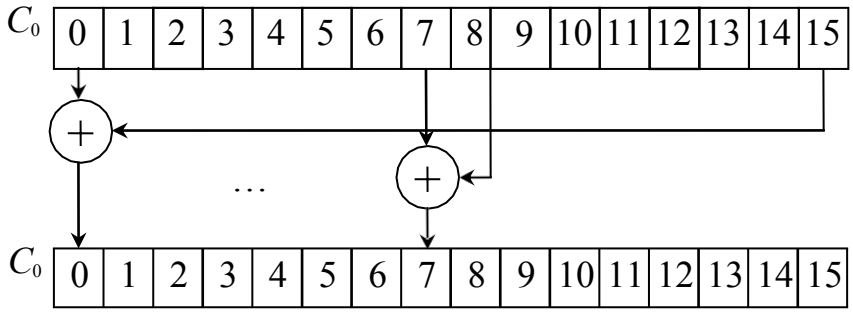

Рисунок 3.15 – Перетворення лівої половини блоку C_0

Таким чином, в результаті виконання нелінійних перетворень отримуємо

зашифрований текст (рис 3.16).

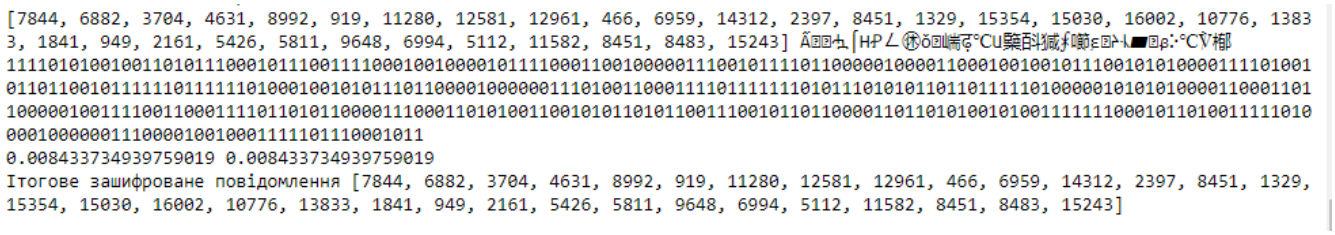

Рисунок 3.16 – Результат зашифрування

Результат розшифрування показано детально на рис. 3.17.

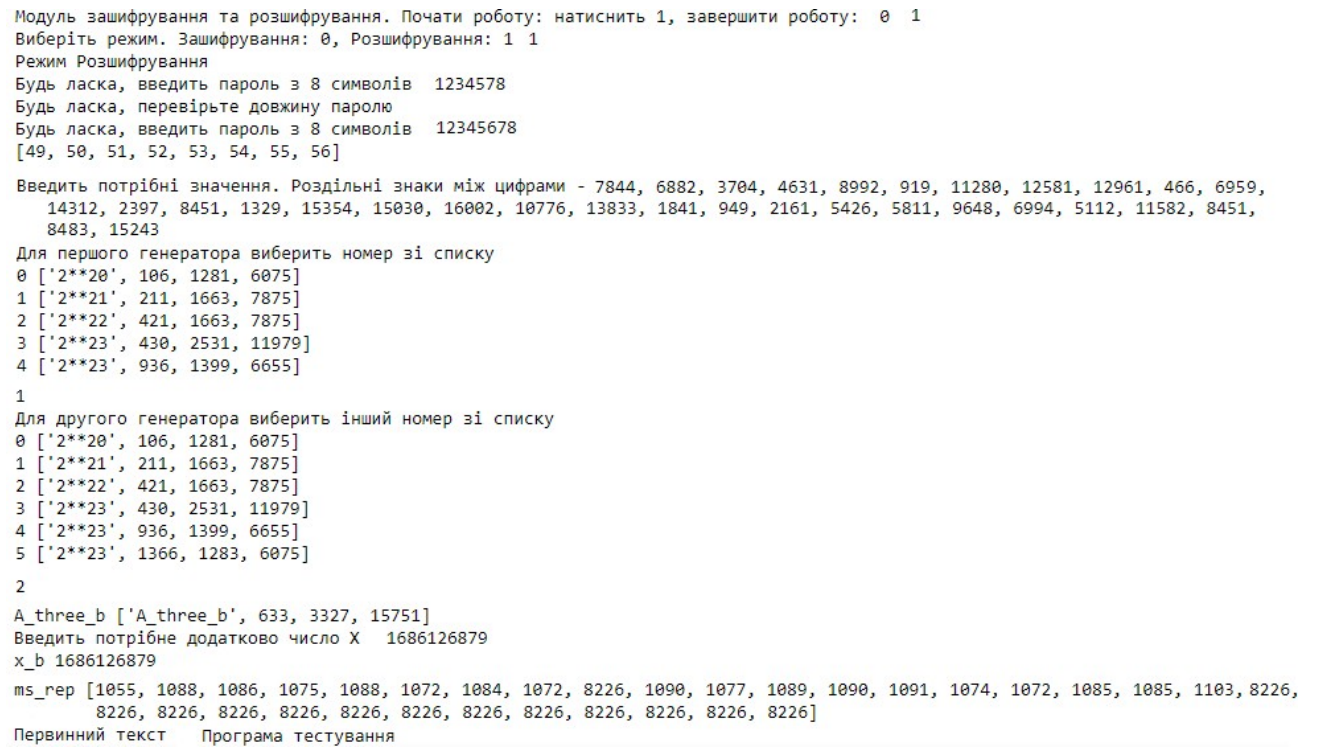

Рисунок 3.17 – Результат розшифрування

3.3 Статистичний тест

Випадкові числа використовуються для побудови гами в поточних криптосистемах, ключів для сеансів (сеансових) та інших ключів у блочних криптосистемах, початкових значень, для генерації параметрів в асиметричних криптосистемах, випадкових значень параметрів для систем електронного підпису, протоколах автентифікації тощо. Загальним підходом до генерування ключів є застосування – генераторів випадкових послідовностей (ГВП) та/або детермінованих генераторів випадкових послідовностей (ДГВП). До ГВП і ДГВП

висуваються складні вимоги щодо генерування символів послідовності випадково, рівноймовірно, незалежно та однорідно. Наприклад, для реалізації потокового симетричного шифру до криптографічно стійкого ГВП (гами шифру) висуваються три основних вимоги:

а) період гами має бути досить великим для шифрування повідомлень різної довжини;

б) гама має бути практично непередбачуваною, що означає неможливість передбачити наступний біт гами, навіть якщо відомі тип генератора і попередній відрізок гами;

в) генерування гами не повинне викликати великих технічних складностей.

Основними вимогами, що висуваються до ДГВП, є непередбачуваність, просторова і часова складність, необоротність, а також період повторення.

Базовими стандартами, що стандартизують алгоритми генерування послідовностей випадкових чисел, є:

– міжнародний стандарт ISO/IEC 18031 “Information technology – Random number generation”, який визначає алгоритми генерування псевдовипадкових і випадкових чисел, а також визначає статистичні тести перевірки генераторів;

– міжнародний стандарт ISO/IEC 18032 “Information technology – Prime number generation”, який визначає методи генерування простих чисел і методи тестування чисел на простоту;

– національний стандарт ДСТУ ISO/IEC 19790:2015 “Інформаційні технології. Методи захисту. Вимоги безпеки до криптографічних модулів” (ISO/IEC 19790:2012, IDT).

Додаткові вимоги до алгоритмів та реалізацій методів і засобів генерування і тестування послідовностей випадкових чисел визначаються наступними стандартами та рекомендаціями: FIPS 140-3, ANSI X 9.17, ANSI X 9.31, NIST SP 800-22, AIS-20, AIS-31 та ін.

Наприклад, рекомендація NIST SP 800-22 “A statistical test suite for random and pseudorandom number generators for cryptographic applications” [12] – застосовується як засіб комплексного контролю. NIST SP 800-22 містить необхідний набір статистичних тестів, сукупність яких обґрунтована, пропонує критерії прийняття рішення відносно не тільки окремої послідовності, але й відносно всього ГВП. До складу методики NIST STS (statistical test suite) входять статистичні тести (табл. 3.1), метою яких є визначення міри випадковості двійкових послідовностей, породжених або апаратними або програмними генераторами випадкових чисел. Ці тести ґрунтуються на різних статистичних властивостях, властивих лише

випадковим послідовностям. Додатковим фактором вибору цієї методики є позитивний досвід її використання при дослідженні статистичних властивостей алгоритмів блокового і поточного шифрування.

Таблиця 3.1 – Статистичні тести NIST STS

№	Статистичний тест	Дефект, що виявляється тестом
1	Частотний (монобітовий тест)	Надто багато нулів або одиниць у послідовності
2	Частотний тест (всередині блоку)	Локалізовані відхилення частоти появи одиниць у блоці від ідеального значення 1/2
3	Перевірка накопичених сум	Велика кількість одиниць або нулів на початку або наприкінці двійкової послідовності
4	Перевірка серій	Надто швидка або надто повільна зміна знака в ході генерації послідовності
5	Перевірка максимальної довжини серії у блоці	Відхилення від теоретичного закону розподілу максимальних довжин серій одиниць
6	Перевірка шаблонів, що перекриваються	Велика кількість *m*-бітових серій із одиниць у послідовності

У програмі реалізовано перевірку отриманої випадкової послідовності на частотний тест (монобітовий тест). Текст програми показано на рис. 3.18 та результат тесту на рис. 3.19.

```
#Value X = tx0 + k
def conv_test_1(A_def, A_three, spa):
    k = 0
    tx0 = int(time.time())
    while k < 10000:
        f_three = fill_abm(A_three, tx0 + k)
        f, st0, st1 = test_1(spa, f_three, 16)
        stsum = st0 + st1
        p0 = abs(0.5 - st0/stsum)
        p1 = abs(0.5 - st1/stsum)
        if p0 > A_def or p1 > A_def:
            k += 1
        else:
            break
    X = tx0 + k
    return f, X, p0, p1
```

Рисунок 3.18 – Текст частотний тест

```
[6542, 6963, 7384, 7805, 351, 772, 1193, 1614, 4127, 4338, 4549, 4760, 4971, 5182, 5393, 5604]
Знайдене число X та окремі відхилення від 0.5  1686126879
0.008433734939759019
```

Рисунок 3.18 – Результат тесту

ВИСНОВКИ ТА РЕКОМЕНДАЦІЇ

У роботі був проведений аналіз симетричних алгоритмів шифрування, які мають складну структуру і за рахунок цього досягається висока стійкість, але ускладнюється програмна реалізація і збільшується потреба в обчислювальних ресурсах. На основі проведеного аналізу алгоритмів шифрування зроблено висновок про доцільність використання нелінійного і блочного алгоритму шифрування для спрощення програмної реалізації. Також було написано текст програми, що реалізує даний алгоритм та виконано його тестування.

В результаті проведеного аналізу в роботі дозволили встановити, що:

1. Одним з найбільш перспективних напрямків розвитку криптографічних засобів для захисту конфіденційності переданих повідомлень, є створення швидкодіючих симетричних алгоритмів шифрування, що працюють у реальному масштабі часу;

2. Використання нелінійних операцій, а також генераторів випадкових послідовностей ускладнює завдання криптоаналітику пошуку ключа повним перебором та виконання дешифрування вхідного повідомлення;

3. Як правило, розробка нових симетричних алгоритмів шифрування розраховуються на те, що криптоаналітик розташовує необмеженими тимчасовими й обчислювальними ресурсами. У той же час мають місце ситуації, при яких строк життя конфіденційної інформації відносно невеликий і “абсолютна” стійкість шифрування просто не затребувана в силу того, що зловмисник просто немає часу на здійснення тривалого криптоаналізу. У цьому випадку швидкість шифрування може бути збільшена, за рахунок скорочення обчислювальних ресурсів, необхідних для реалізації алгоритмів шифрування.

Перераховані висновки роблять завдання побудови простих і ефективних блокових шифрів особливо актуальної. Працюючи в реальному масштабі часу, вони дозволять виключити оперативне втручання зловмисників у процеси керування й обробки критичної інформації. Запропонований та реалізований алгоритм шифрування відповідає всім сучасним вимогам для побудови симетричних криптосистем та може використовуватися в інформаційних системах для захисту конфіденційної інформації у реальному часі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Горбенко І. Д. Прикладна криптологія. Теорія. Практика: монографія / І. Д. Горбенко, Ю. І. Горбенко. – Харків: Видавництво «Форт», 2012. – 880 с.

2. Корченко О. Г. Прикладна криптологія: системи шифрування: підручник / О. Г. Корченко, В. П. Сіденко, Ю. О. Дрейс. – К.: ДУТ, 2014. – 448 с.

3. Schneier B. Applied Cryptography: Protocols, Algorithms and Source Code in C: 20th Anniversary Edition. Wiley, 2015. – 784 p.

4. Остапов С. Е. Технології захисту інформації: навчальний посібник / С. Е. Остапов, С. П. Євсеєв, О. Г. Король. – Х.: Вид. ХНЕУ, 2013. – 476 с.

5. DES Animation. [Електронний ресурс]. – Режим доступу: https://www.youtube.com/watch?v=Vcld7CMAnNs

6. DES Visualization. [Електронний ресурс]. – Режим доступу: https://www.youtube.com/watch?v=XjZmirLc4k4

7. Stavroulakis P., Stamp M. Handbook of Information and Communication Security. Berlin: Springer-Verlag, 2010. – 863 p.

8. Shannon Claude. Communication Theory of Secrecy Systems / Bell System Technical Journal, 1949, – 59 p.

9. Остапов С. Е. Технології захисту інформації: навчальний посібник / С. Е. Остапов, С. П. Євсеєв, О. Г. Король. – Х.: Вид. ХНЕУ, 2013. – 476 с.

10. Кузнецов Г. В. Математичні основи криптографії / Г. В. Кузнецов, В. В. Фомічов, С. О. Сушко. – Дніпропетровськ: НГУ, 2004. – 389 с.

11. Горбенко Ю. І. Інфраструктури відкритих ключів. Електронний цифровий підпис. Теорія та практика: монографія / Ю. І. Горбенко, І. Д. Горбенко. – Харків: Видавництво «Форт», 2010. – 608 с.

12. NIST SP 800-22. [Електронний ресурс]. – Режим доступу: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-22r1a.pdf

Додаток А

ПЕРЕЛІК КОПІЙ ДЕМОНСТРАЦІЙНОГО МАТЕРІАЛУ

Перелік основних слайдів презентації:

Слайд 1 – Узагальнена схема криптосистеми шифрування (рис. 1.1, с. 12);

Слайд 2 – Структурна схема криптосистеми з секретним ключем (рис. 1.2, с. 13);

Слайд 3 – Узагальнені перетворення в криптосистемі (рис. 1.4, с. 14) ;

Слайд 4 – Схема алгоритму DES (рис. 1.6, с. 6);

Слайд 5 – Порівняльна характеристика алгоритмів шифрування (табл. 1.9, с. 21);

Слайд 6 – Фреймворк Anaconda Navigator (рис. 3.1, с. 29);

Слайд 7 – Інтерфейс крос-платформи Spyder IDE (рис. 3.2, с. 30);

Слайд 8 – Програми тестування Python Jupyter Notebook, Jupyterlab (рис. 3.3, с. 30);

Слайд 9 – Текст програми модуля ok_exe() (рис. 3.4, с. 31);

Слайд 10 – Результат виконання модуля ok_exe() (рис. 3.5, с. 32);

Слайд 11 – Використання нелінійних перетворень (рис. 3.8, с. 32; рис. 3.13, рис. 3.15, с. 35);

Слайд 12 – Текст модуля switching of numbers (рис. 3.10, с. 34);

Слайд 13 – Текст модуля listing of points (рис. 3.9, с. 33);

Слайд 14 – Результат зашифрування (рис. 3.16, с. 36);

Слайд 15 – Результат розшифрування (рис. 3.17, с. 36).

Додаток Б

ТЕКСТ ПРОГРАМИ

In [1]:
```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

#import numpy as np # linear algebra
#import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

#import os
#for dirname, _, filenames in os.walk('/kaggle/input'):
    #for filename in filenames:
        #print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version us
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

Input libraries

In [2]:
```
import time
```

In [3]:
```
#functions
```

In [4]:
```
#calculation of i-value in vectors
def calc(AE, p):
    c = (AE[1]* p + AE[2])%AE[3]
    return c
```

In [14]:
```
#Getting a password from a user, strictly 8 characters long
def input_pass():
    a_pass = ""
    A = True
    while A == True:
        a_pass = input("Будь ласка, введить пароль з 8 символів")
        if len(a_pass) != 8:
            print("Будь ласка, перевірьте довжину пароля")
            A = True
        else:
            A = False
    return a_pass
```

In [15]:
```
# Convert string to sequence from ASCII code to list
def conv_pass(a_pass):
    a_ps = []
    for i in a_pass:
        k = ord(i)
        a_ps.append(k)
    return a_ps
```

In [16]:
```
#input txt with filling missing symbols
#using '*'
def input_txt():
    A = True
    while A == True:
        d_txt = '*'
        c_txt = ''
        a_txt = input("Введить текст")
        a_txt = a_txt.replace(' ', '*')
        b = len(a_txt)
        k = b % 16

        if k!= 0:
            a_txt = a_txt + '*' * (16 - k)


        c_txt = conv_pass(a_txt)

        A = False
    return c_txt
```

In [19]:
```
#select a regime
def regime():
    A = True
    A_reg = True
    while A == True:
        a_reg = input("Виберіть режим. Зашифрування: 0, Розшифрування: 1")
        if a_reg == '0':
            A_reg = A
            A = False
        elif a_reg == '1':
            A_reg = False
            A = False
        else:
            pass

    return A_reg
```

In [20]:
```
#AR = regime()
#print(AR)
```

In [21]:
```
#Input a value of deflection
def a_defn():
    A = True

    while A ==  True :
        A_def = float(input("Введить максимальну величину відхилення після підсумкового зашифрування(звичайно від 0.001 до 0.01)"
        if A_def < 0 or A_def > 0.01:
            pass
        else:
            A = False
    return A_def
```

In [23]:
```
#listing of points (value max, a, b, m)
A_base = [['2**20',106,1281,6075],
          ['2**21',211,1663,7875],
          ['2**22',421,1663,7875],
          ['2**23',430,2531,11979],['2**23',936,1399,6655],['2**23',1366,1283,6075],
          ['2**24',171,11213,53125],['2**24',859,2531,11979],['2**24',419,6173,29282],['2**24',967,3041,14406],
          ['2**25',141,28411,134456],['2**25',625,6571,31104],['2**25',1541,2957,14000],['2**25',1741,2731,12960],['2**25',1291,
          ['2**26',421,17117,81000],['2**26',1255,6173,29282],['2**26',281,28411,134456],
          ['2**27',1093,18257,86436],['2**27',421,54773,259200],['2**27',1021,24631,116640],['2**27',1021,24673,121500],
          ['2**28',1277,24749,117128],['2**28',741,66037,312500],['2**28',2041,25673,121500],['2**29',2311,25367,120050],
          ['2**29',1807,45289,214326],['2**29',1597,51749,244944],['2**29',1861,49297,233280],['2**29',2661,36979,175000],
          ['2**30',3877,29573,139968],['2**30',3613,45289,214326],['2**30',1366,150889,714025],
          ['2**31',8121,28411,134456],['2**31',4561,51349,243000],['2**31',7141,54703,259200],
          ['2**32',9301,49297,233280],['2**32',4096,150889,714025],
          ['2**33',2416,374441,1771875],
          ['2**34',17221,107839,510300],
          ['2**35',84589,45989,217728]]
```

In [25]:
```
#switching of numbers
def switch_n(A_base):
    k = 0
    A = True
    while A == True:
        print("Для першого генератора виберить номер зі списку")
        for i in range(len(A_base)):
            print(i, A_base[i][:])
        k  = int(input())
        if k < 0 or k > len(A_base):
            print("Спробуйте ще раз")
        else:
            A = False
    A_one = A_base[k][:]

    k2 = 0

    while A == False:
        print("Для другого генератора виберить інший номер зі списку")
        for i in range(len(A_base)):
            print(i, A_base[i][:])
        k2 = int(input())
        if k2 < 0 or k2 > len(A_base):
            print("Спробуйте ще раз")
        elif k2 == k:
            print("Однакові значення. Спробуйте ще раз")
        else:
            A == False
            break

    A_two = A_base[k2][:]


    return A_one, A_two
```

In [36]:
```
#get txt_mix and transforming
def get_txt_mix():
    A = True
    while A == True:
        txt_b = []
        a = ''
        txt_m = input ("Введить потрібні значення. Роздільні знаки між цифрами - кома")
        t_m = txt_m.replace(' ', '')
        t_m += ','
        for i in t_m:
            if i != ',':
                a += i
            else:
                txt_b.append(int(a))
                a = ''
        t = len(txt_b)
        if t % 16 == 0:
            A = False
        else:
            print("Перевірьте ще раз текст!")
    return txt_b
```

In [37]:
```
#txt_mix = get_txt_mix()
#print(type(txt_mix),txt_mix)
```

In [38]:
```
def get_X():
    A = True
    while A == True:
        x = int(input("Введить потрібне додатково число X"))
        if x < 1:
            print("Перевірьте число!")
        else:
            A = False
            break
    return x
```

In [42]:
```
def ok_exe():
    A = True
    while A == True:
        a = input("Модуль зашифрування та розшифрування. Почати роботу: натиснить 1, завершити роботу:  0 ")
        if a == '0':
            print("Дякуємо за співпрацю!")
            A = False
            break
        elif a == '1':
             #switching of regime
            AR = regime()
            if AR == True:
                tx0 = int(time.time())

                print("Режим Зашифрування")

                ms_base = input_txt()
                print('ms_base', ms_base)
                print(len(ms_base))
                ms_sm = mixing(rt, ms_base)
                print('ms_sm',ms_sm, len(ms_sm))

                ap = input_pass()
                print(ap)
                aps = conv_pass(ap)
                print(aps)

                A_def = a_defn()
                print(A_def)

                A_one, A_two = switch_n(A_base)

                A_three = ['A_three',A_one[1] + A_two[1] + 1, A_one[2] + A_two[2]+1, A_one[3] + A_two[3]+1]
                print(A_three[1:])

                spa = a1_2(A_one, A_two, aps)
                print(spa)

                th,X,p0,p1 = conv_test_1(A_def, A_three, spa)
                ##print(th)
                print("Знайдене число X та окремі відхилення від 0.5 ",X,p0,p1)

                txt_mix, bin_mix, x0,x1 = conv_mod_2(ms_sm, th, 16)
                txtd_mix = list_to_str_CHR(txt_mix)
                print(txt_mix, txtd_mix)
                print(bin_mix)
                print(x0,x1)
                print("Ітогове зашифроване повідомлення",str_to_list_ORD(txtd_mix))
```

```
        else:
            print("Режим Розшифрування")

            ms_rep_str = decription()
            print("Первинний текст", ms_rep_str)
    else:
        pass


return A
```