СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Навчально-науковий інститут (факультет) інформаційних технологій та електроніки

Кафедра ____інформаційних технологій та програмування_____
(повна назва кафедри)

ПОЯСНЮВАЛЬНА ЗАПИСКА
до кваліфікаційної випускної роботи

освітній ступінь ____бакалавр_____
(бакалавр, магістр)
спеціальність 126 «Інформаційні системи та технології»____
(шифр і назва спеціальності)

на тему _« Розробка методів і алгоритмів шифрування в сучасних інформаційних системах»_____

Виконав: студент групи ІСТ-21д _____ (підпис) В. С. Лось (ініціали і прізвище)

Керівник _____ (підпис) В.О. Лифар (ініціали і прізвище)

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

Рецензент _____ Д.В.Ратов _____

Київ – 2025

СХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВОЛОДИМИРА ДАЛЯ

Навчально-науковий інститут (факультет) інформаційних технологій та електроніки
Кафедра інформаційних технологій та програмування
(повна назва кафедри)
Освітній ступінь бакалавр
(бакалавр, магістр)
спеціальність 126 «Інформаційні системи та технології»
(шифр і назва спеціальності)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Захожай О.І.
"___" _____ 2025 року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ ВИПУСКНУ РОБОТУ СТУДЕНТУ
Лось Віктор Сергійович
(прізвище, ім'я, по батькові)

1. Тема роботи: Розробка методів і алгоритмів шифрування в сучасних інформаційних системах
керівник роботи Лифар Володимир Олексійович, д.т.н., доц..
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджений наказом університету від "_27_" _травня_2025 року №67/15.15-С
2. Строк подання студентом роботи 19.06.2025р.
3. Вихідні дані до роботи: Об'єктом даної роботи є процес дослідження історії криптографії і розробка програми
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): Дослідити найвідоміші історичні алгоритми шифрування. Дослідити поняття шифрування і найважливіші питання щодо цього. Класифікувати шифрування, дослідити, як працює на прикладі банкінгу Розробити програму на Python з використанням деяких алгоритмів. Висновки. Перелік використаних джерел.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслеників) мал 1, мал 2, мал 3, мал 4. мал 5, мал 6, мал 7.

6. Консультанти розділів роботи

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

7. Дата видачі завдання 01.04.2025р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання кваліфікаційної випускної роботи	Строк виконання етапів	Примітка
1	Одержання завдання на виконання роботи	01.04.25	виконано
2	Укладання і погодження з керівником плану і етапів виконання роботи	06.04.25	виконано
3	Узагальнення даних літературних джерел, укладання розділу «Аналіз предметної галузі»	16.04.25	виконано
4	Аналіз шляхів виконання завдання. погодження з керівником теоретичної частини роботи.	26.04.25	виконано
5	Укладання та тестування програмного продукту	26.05.25	виконано
6	Укладання, оформлення та погодження пояснювальної записки з керівником	18.06.25	виконано
7	Надання готової пояснювальної записки на кафедру	19.06.25	виконано
8	Підготовка доповіді і презентації	20.06.25	виконано

Студент _____ В. С. Лось
підпис (ініціали і прізвище)

Керівник роботи _____ В.О. Лифар
підпис (ініціали і прізвище)

РЕФЕРАТ

Робота містить: 59 сторінок основного тексту, 16 рисунків, 3 таблиці, 6 використаних джерел.

Метою кваліфікаційної випускної роботи є дослідження та огляд історичних алгоритмів шифрування, огляд сучасних альтернатив та розробка програми шифрування за застосуванням декількох із них. Огляд історичних алгоритмів шифрування проведений, теми сучасних алгоритмів та їх класифікація, застосування. Проведений незапланований розбір алгоритму DES. Створено програму шифрування-дешифрування. Проведено тестування пройшло успішно. Програмне забезпечення відповідає вимогам технічного завдання, програма більше демонстративна, і наврядче має практичне застосування бо створена з демонстрацією результатами хешування ключа.

Ключові слова: Encryption, шифрування, DES, AES, криптографія, історія.

ЗМІСТ

Вступ

Чому криптографія така важлива сьогодні?

Криптографія **–** це не просто складні формули й алгоритми, а справжній захист для наших даних. Щодня ми спілкуємося в соцмережах, переказуємо гроші через онлайн-банкінг, користуємося мобільними додатками – і всюди працює шифрування. Без нього хакери могли б легко перехоплювати паролі, читати повідомлення чи навіть красти гроші з рахунків.

Де ми стикаємося з криптографією?

Кібербезпека

Щороку кількість хакерських атак зростає: ламають акаунти, крадуть дані, шантажують користувачів.

Шифрування допомагає захистити інформацію, щоб вона залишалася недоступною для сторонніх.

Приватність листування та особистих даних:

Щодня ми відправляємо сотні повідомлень, навіть не думаючи про їх безпеку.

Сучасні месенджери (WhatsApp, Signal, Telegram) використовують наскрізне шифрування, щоб ніхто, крім вас і отримувача, не міг їх прочитати.

Оплата в інтернеті та банківські операції

Коли ви вводите номер картки на сайті або розраховуєтесь телефоном, дані автоматично шифруються.

Без криптографії шахраї могли б легко перехоплювати платіжну інформацію та викрадати гроші.

Криптовалюти та блокчейн

Біткоїн, Ethereum та інші цифрові валюти існують завдяки криптографії.

Шифрування гарантує, що ніхто не може підробити транзакції чи змінити записи в блокчейні.

Інтернет речей (IoT) та штучний інтелект

"Розумні" пристрої – колонки, холодильники, камери спостереження – теж потребують захисту.

Якщо їхні дані не шифрувати, хакери можуть отримати доступ і навіть керувати цими пристроями.

Державна безпека

Будь-яка країна захищає свої секретні дані за допомогою криптографії.

В армії, спецслужбах та урядових установах інформація шифрується, щоб її не змогли перехопити.

Технології розвиваються, і разом із ними зростає роль криптографії. У майбутньому з’являться квантові комп’ютери, які зможуть зламувати сучасні шифри за лічені хвилини. Тому вчені вже розробляють нові алгоритми, які будуть стійкими до таких атак.

Криптографія – це невидимий щит, який захищає нас щодня, навіть якщо ми цього не помічаємо.

Чому важливо розуміти, як працюють алгоритми шифрування?

Криптографія – це не магія, а набір правил і методів, які вирішують конкретні завдання. Якщо не розбиратися в них, можна вибрати неправильний метод шифрування й поставити під загрозу безпеку даних.

Передача повідомлень

Наприклад, у месенджерах використовується наскрізне шифрування (end-to-end encryption).

Якщо алгоритм буде слабким, хакери зможуть розшифрувати повідомлення.

WhatsApp використовує Signal Protocol, а ось звичайні SMS взагалі не шифруються.

Захист паролів

Паролі не просто шифрують, а хешують – тобто перетворюють у такий вигляд, який не можна повернути назад.

Для цього використовують bcrypt, Argon2 або SHA-256, а ось MD5 – застарілий, бо до нього є готові бази розшифрованих значень.

Симетричне чи асиметричне шифрування?

Симетричні алгоритми (AES, Blowfish) працюють швидше, але потребують безпечного зберігання ключа.

Асиметричні алгоритми (RSA, ECC) дозволяють шифрувати без передачі секретного ключа, але вони повільніші.

Наприклад, у HTTPS для безпеки спочатку використовується RSA, а потім для швидкості підключається AES.

Квантові комп’ютери – загроза майбутнього

Алгоритми RSA і ECC у майбутньому можуть стати небезпечними, бо квантові комп’ютери здатні їх зламати.

Уже зараз розробляють постквантові алгоритми, які будуть захищені від таких атак.

Розділ 1. Теоретичні основи та історія криптографії. Історія розвитку криптографії. Стародавні методи шифрування (шифр Цезаря, скитала).

Як у давнину ховали секретні повідомлення?

Ще задовго до комп'ютерів і складних алгоритмів люди розуміли: якщо не хочеш, щоб ворог дізнався твої секрети, треба їх якось зашифрувати. Тоді не було складної математики чи потужних комп'ютерів, тому доводилося викручуватися простими, але хитрими методами.

Шифр Цезаря – найпростіший, але дієвий

Римський імператор Юлій Цезар не любив, коли його листи перехоплювали вороги. Тому придумав спосіб їх шифрувати: просто зсував літери в алфавіті на кілька позицій уперед.

Як це працює?

Беремо текст і кожну букву змінюємо, зсуваючи її на кілька позицій вперед.

Наприклад, якщо зсув = 3, то «А» стає «Г», «Б» – «Д», «В» – «Е» і так далі.

Щоб розшифрувати, потрібно просто зробити зворотний зсув.

Приклад:

Оригінальний текст: «КІТ»

Після шифрування (зсув 3): «НЛХ»

Чим зручно? Дуже просто і швидко! У чому проблема? Якщо ворог здогадається, що це шифр Цезаря, то він просто спробує всі можливі варіанти зсуву (їх усього 33 для українського алфавіту). Тож це не найнадійніший варіант.

Сьогодні шифр Цезаря більше розважає людей у квестах і головоломках, ніж реально захищає інформацію.

Скитала – військова хитрість спартанців

Спартанці теж не хотіли, щоб вороги знали їхні плани. Тому вони використовували скиталу – дуже цікавий і, на той час, надійний метод шифрування.

Як це працювало?

Беремо дерев’яну палку (циліндр).

Намотуємо на неї довгу стрічку з пергаменту.

Пишемо повідомлення по горизонталі.

Знімаємо стрічку – і тепер букви виглядають як набір випадкових символів.Прочитати текст можна тільки, якщо намотати стрічку на таку саму палку.

Приклад:

Намотуємо пергамент на палку.

Пишемо: **«АТАКА ВНОЧІ».**

Знімаємо – отримуємо хаос із букв.

Якщо ворог спробує прочитати – нічого не зрозуміє.

Чим зручно? Якщо у ворога немає такої самої палки, він не зможе розшифрувати повідомлення.

То що з цього працює сьогодні?

Зараз і шифр Цезаря, і скитала виглядають як дитячі іграшки – їх легко зламати за лічені секунди. Але в давнину вони справді працювали і рятували життя! Саме з таких простих методів почалася історія криптографії, яка зараз перетворилася на складну науку, що захищає наші паролі, банківські картки та переписки.

Шифр Атбаш — це перевернути алфавіт задом наперед.

Як це працює?

Пишемо абетку в два ряди: один звичайний, а другий перевернутий:

А Б В Г Д ...
Я Ю Ї І Є ...

Тепер «А» стає «Я», «Б» – «Ю», «В» – «Ї» і так далі.

Наприклад, слово «КІТ» буде «РНГ».

Чим це зручно? Дуже просто!

Чому це не надійно? Якщо хтось здогадається, що це шифр Атбаш, він швидко розшифрує повідомлення.

До речі, цей метод згадується навіть у Біблії – там є слова, закодовані саме таким чином.

У сучасній літературі також згадується такий вид шифрів, як моно-алфавітний шифр. Навіть є приклад де цей шифр використовувала шотландська королева Марія Стюарт, щоб шифрувати свої листи.

У звичайному моно-алфавітному шифрі заміни (як у шифрі Цезаря), кожна літера відповідає одній заміні:

A → M, B → Q, і т.д.

У гомофонічному шифрі:

Найчастіші літери (наприклад, E в англійській) мають кілька можливих шифрованих символів.

Рідкісні літери — лише одну або дві заміни.

Це маскує реальну частоту появи символів, ускладнюючи криптоаналіз.

Приклад:

Літера	**Можливі шифри**
E	10, 23, 51
T	17, 34
A	01
H	11
X	99

Текст: THE HAT

Шифрується як: 17 11 23 11 01 17

*(або з іншим набором — **рандомізація важлива**)*

Перевага:

Цей тип шифрування:

Зменшує ефективність частотного аналізу — головного методу розшифрування при простих замінах.

Підвищує криптостійкість, хоч і залишається вразливим до глибшого статистичного аналізу, особливо при довгих текстах.

Історичне використання:

Марія Стюарт та інші змовники XVI ст.

Єзуїтські листи**,** королівські двори**,** дипломати**.**

У XVIII–XIX ст. такі шифри були основою шифрувальної практики (разом із шифром Полібія, військовими шифрами тощо).

Квадрат Полібія – букви в цифри

Греки, зокрема історик Полібій, придумали метод, який використовував не літери, а цифри. Це був квадрат Полібія.

Як це працює?

Створюємо таблицю 5×5, в яку записуємо букви.

Кожній букві надається свій код — номер рядка і стовпця.

Наприклад, якщо буква «К» знаходиться в рядку 3, стовпці 2, то її код — 32.

Ось як виглядає таблиця:

	1	2	3	4	5
1	А	Б	В	Г	Ґ
2	Д	Е	Є	Ж	З
3	І	Ї	Й	К	Л
4	М	Н	О	П	Р
5	С	Т	У	Ф	Х

Шифрувати слово «КІТ», це буде так:

К → 34

І → 31

Т → 52

Тобто «КІТ» перетворюється на 343152.

Чим це зручно? Якщо ворог не знає таблицю, йому буде важко розшифрувати повідомлення.

Чому це не ідеально? Якщо ворог здогадається, що це шифр Полібія, він розшифрує все дуже швидко.

Цей метод використовували навіть у XIX столітті для телеграфних повідомлень!

Китайські вузли – код через мотузку

А ось у Китаї придумали ще один дуже незвичайний спосіб шифрування — китайські вузли. Це була свого роду мотузка з кодованими зав'язками.

Як це працює?

Люди зав'язували вузли на мотузці, і кожен вузол означав певне слово чи число.

Тільки той, хто знав, як інтерпретувати ці вузли, міг прочитати повідомлення.

Чим це зручно? Якщо ворог не знає значення вузлів, то нічого не розшифрує. Чому це не дуже зручно? Цей метод не підходить для довгих повідомлень, і передавати складну інформацію через мотузки було не дуже зручно.

До речі, інки також використовували схожу систему – кіпу, де на мотузках зав'язували вузли для обліку запасів та передачі повідомлень.

Середньовіччя та епоха Відродження (шифри Віженера, дипломатичні коди).

А Б В Г Ґ Д Е Є Ж З И І Ї Й К Л М Н О П Р С Т У Ф Х Ц Ч Ш Щ Ь Ю Я

А А Б В Г Ґ Д Е Є Ж З И І Ї Й К Л М Н О П Р С Т У Ф Х Ц Ч Ш Щ Ь Ю Я

Б Б В Г Ґ Д Е Є Ж З И І Ї Й К Л М Н О П Р С Т У Ф Х Ц Ч Ш Щ Ь Ю Я А

В В Г Ґ Д Е Є Ж З И І Ї Й К Л М Н О П Р С Т У Ф Х Ц Ч Ш Щ Ь Ю Я А Б

Г Г Ґ Д Е Є Ж З И І Ї Й К Л М Н О П Р С Т У Ф Х Ц Ч Ш Щ Ь Ю Я А Б В

Ґ Ґ Д Е Є Ж З И І Ї Й К Л М Н О П Р С Т У Ф Х Ц Ч Ш Щ Ь Ю Я А Б В Г

Д Д Е Є Ж З И І Ї Й К Л М Н О П Р С Т У Ф Х Ц Ч Ш Щ Ь Ю Я А Б В Г Ґ

Е Е Є Ж З И І Ї Й К Л М Н О П Р С Т У Ф Х Ц Ч Ш Щ Ь Ю Я А Б В Г Ґ Д

Є Є Ж З И І Ї Й К Л М Н О П Р С Т У Ф Х Ц Ч Ш Щ Ь Ю Я А Б В Г Ґ Д Е

ЖЖЗ И І Ї Й К Л М Н О П Р С Т У Ф Х Ц Ч Ш Щ Ь Ю Я А Б В Г Ґ Д Е Є

З З И І Ї Й К Л М Н О П Р С Т У Ф Х Ц Ч Ш Щ Ь Ю Я А Б В Г Ґ Д Е Є Ж

Епоха Відродження: як криптографія стала справжнім мистецтвом

Уявіть собі: Європа середини 1500-х років, час, коли наука і мистецтво буквально вибухають. Ренесанс — це коли вчені і художники створюють щось нове, коли все змінюється і рухається вперед. Але, між тим, цей час був і дуже небезпечним. Секрети — це те, що могло визначити долю країн і королів. Уявіть, як важливо було не просто відправити лист, а впевнитись, що ніхто не дізнається того, що ви не хочете, аби знали. І ось тут з’являється справжня магія шифрування.

Італійський вчений Джованні Баттіста Беллазо в 1553 році придумав перший метод шифрування. Але лише коли французький дипломат Блез де Віженер у 1586 році удосконалив цей метод, шифрування стало не просто корисним, а справжнім мистецтвом. І це був справжній прорив! Віженер зрозумів, що можна зробити шифр складнішим за допомогою ключового слова, яке змінює текст у такий спосіб, що навіть найбільші експерти не зможуть його розшифрувати без цього слова. І це було неймовірно важливо для того часу!

Як це працює? Давайте на прикладі

Припустимо, ви хочете зашифрувати слово "РОЗА". Для цього берете ключове слово "ДЖЕ". І щоб його довжина була такою самою, як у вашого слова, просто повторюєте: "ДЖЕД".

Тепер шифруємо по буквах:

1. "Р" + "Д" → "Х"
2. "О" + "Ж" → "С"
3. "З" + "Е" → "М"

4. "А" + "Д" → "Д"

Результат: "ХСМД".

Якщо хтось знайде таке зашифроване повідомлення, він навіть не здогадається, що це таке, якщо не буде знати вашого ключа. Це справжня таємниця!

Код для дипломатів: як замінювали слова цифрами

Але шифрування — це не єдиний спосіб зберігати таємниці. Справжні дипломати того часу також використовували кодові книги. Це були спеціальні таблиці, в яких замість слів писали цифри або символи. Уявіть, ви отримуєте лист, і замість того, щоб прочитати в ньому фразу "король пішов на війну", ви бачите лише числа. І щоб зрозуміти, що означає повідомлення, треба мати відповідну таблицю.

Наприклад, у Франції часів кардинала Рішельє була така система кодів:

105 означало "король",

207 — "ворог",

318 — "військо".

І коли дипломат отримував повідомлення 105 207 318, він знав, що це означає: "Король — ворог війська". Звучить просто, але це працювало як справжній код, який міг розібрати тільки той, хто знає цю таблицю. І навіть якщо вороги захоплювали цей лист, без цієї таблиці вони не могли зрозуміти, про що йде мова.

Епоха механізації криптографії: шифрувальні машини "Енігма" та "Лоренц"

У першій половині XX століття світ змінився радикально. Людство зіткнулося з масштабними війнами, новими технологіями та небаченими досі обсягами інформації, яку потрібно було зберігати в таємниці. Простих

шифрів, які працювали на основі паперу, олівця й трохи логіки, вже не вистачало. На зміну їм прийшли складні механізми — шифрувальні машини, які стали справжньою революцією в криптографії.

Енігма — легендарна машина з ротором у серці

Найвідомішою з усіх шифрувальних машин стала "Енігма" — німецький пристрій, який спочатку створили для цивільного користування, а згодом адаптували для потреб армії. На вигляд це була ніби звичайна друкарська машинка, але всередині — справжній інженерний шедевр.

Основним принципом її роботи був роторний механізм: набір дисків, які оберталися при кожному натисканні клавіші. Коли користувач вводив літеру, сигнал проходив через ці ротори, змінюючи свою форму, а потім відбивався через рефлектор — спеціальний елемент, що «віддзеркалював» сигнал назад. Таким чином, кожна літера шифрувалася унікально, і те саме слово могло виглядати зовсім по-різному при кожному новому шифруванні.

Комбінацій для налаштування "Енігми" було настільки багато (понад 150 квінтильйонів), що німці вважали її незламною. І справді, для людини з олівцем і аркушем це було б неможливо.

Проте злам вдалося. Перший прорив зробили польські математики, а потім роботу продовжили британські криптоаналітики, серед яких був Алан Тюрінг. Вони створили спеціальні пристрої для автоматичного підбору налаштувань "Енігми", і таким чином змогли читати німецькі шифровки. Це стало одним з найважливіших досягнень криптоаналізу XX століття та, за оцінками істориків, суттєво наблизило завершення війни.

Лоренц — складніший, невидимий, але теж зламаний

Для особливо важливих і стратегічних повідомлень вищого командування Третього Рейху використовувалася ще одна машина — "Лоренц SZ-40/42".

Вона працювала зовсім не так, як "Енігма". Її мета — шифрування телеграфного (двійкового) коду, а не окремих букв.

"Лоренц" мав 12 коліс, кожне з яких відповідало за створення псевдовипадкових двійкових послідовностей. Ці колеса обертались за складною схемою, тож навіть якщо знати початковий текст і мати зашифрований результат, зрозуміти, що відбулося "всередині", було майже неможливо.

Ця система вважалася на порядок складнішою за "Енігму", і довгий час союзники навіть не знали про її існування. Але у 1941 році через людську помилку в німецькому штабі (повторна відправка того самого повідомлення з незначними змінами) британцям вдалося зрозуміти, як працює шифр.

Для дешифрування таких повідомлень було створено перший електронний програмований комп’ютер "Колосс". Саме так — криптоаналіз "Лоренца" став поштовхом до розвитку комп’ютерної техніки. Цей етап не просто змінив хід війни, а й започаткував цифрову еру криптографії.

Підсумки: коли шифрування стало справою машин

"Енігма" та "Лоренц" — це два символи періоду, коли криптографія вийшла за межі аркуша з ручкою й увійшла у світ механіки, електроніки й комп’ютерів. Ці машини шифрували мільйони повідомлень, але зламали їх не лише технології — а люди, які стояли за ними, їхнє мислення, наполегливість і глибоке розуміння принципів шифрування.

Цей історичний досвід показав: якою б складною не була система, вона не є непереможною. А ще — відкрив шлях до сучасної криптографії, в якій машини вже не просто допомагають, а самі шифрують, зберігають і навіть навчаються на даних.

Сучасна криптографія: перехід до комп’ютерних алгоритмів (DES, RSA).

Після того як відгриміла Друга світова, стало зрозуміло: механічні шифрувальні машини типу Енігми — це вже вчорашній день. Людство поступово перейшло до епохи комп’ютерів, і разом із цим змінилася й сама криптографія. Якщо раніше для шифрування крутили шестерні та перемикали важелі, то тепер усім почала керувати математика і програмний код.

Одним із перших великих кроків у цьому напрямі став алгоритм DES (Data Encryption Standard). Його створили у 1970-х, коли вже були перші електронні машини. DES працював із бітами — тобто з «нулями» й «одиничками» — і міг надійно захищати інформацію, наприклад, на банківських картах. Він "перемішував" біти даних за заданим ключем (комбінацією), і це було досить складно зламати — на той час.

Але минули роки, комп’ютери стали сильнішими, і те, що колись вважалось «незламним», почало зламуватись. Потужні машини змогли просто перебирати всі можливі ключі DES — як комбінації на замку. Це називається brute force, і для DES він став реальною загрозою.

І от на сцену виходить справжня крипто-революція — алгоритм RSA, який з’явився у 1977 році. Це була не просто нова схема — це був повний переворот у підході. В RSA ти не тримаєш ключ в таємниці, навпаки — ти публікуєш відкритий ключ, а шифруєш і розшифровуєш уже за чіткою математичною формулою.

Як це працює? Уявимо, що кожен має дві частини ключа:

одну відкриту (її можна давати всім), іншу — секретну (її не знає ніхто, крім тебе).

І от будь-хто може надіслати тобі зашифроване повідомлення — але тільки ти зможеш його розшифрувати. Це стало реально завдяки простій, але

дуже хитрій ідеї: великі прості числа важко розкласти на множники. І на цьому вся безпека й тримається.

RSA відкрив двері до світу безпечного інтернету — електронна пошта, онлайн-платежі, авторизація на сайтах — усе це стало можливим саме завдяки такій криптографії.

Поняття та значення криптографії • Визначення криптографії та її роль у захисті інформації.

Уявимо собі ситуацію: ми хочемо передати важливе повідомлення другові, але так, щоб ніхто інший його не прочитав. Ми можемо написати листа, але що буде, якщо хтось перехопить його дорогою? Саме тут на допомогу приходить криптографія.

Криптографія — це наука про шифрування. Її головна мета — перетворити зрозуміле повідомлення на щось незрозуміле для сторонніх, а потім повернути його у вихідний вигляд для того, хто має ключ. У широкому сенсі криптографія — це мистецтво приховувати зміст інформації, роблячи її доступною лише для тих, кому вона дійсно призначена.

На перший погляд може здатися, що це щось складне або суто технічне. Але насправді криптографія вже давно стала частиною нашого повсякденного життя. Кожного разу, коли ви:

входите в свій акаунт у соцмережі;

відправляєте повідомлення у месенджері;

оплачуєте щось картою онлайн;

вводите пароль на сайті —

ви користуєтесь плодами роботи криптографів. І навіть не помічаєте цього.

У сучасному світі, де інформація передається за секунди з одного кінця планети на інший, криптографія — це не просто додаткова функція, а

необхідна умова безпеки. Без неї будь-який пароль, будь-який секрет, будь-яка фінансова операція могла б легко опинитись у руках зловмисників.

Завдяки криптографії ми можемо:

захистити конфіденційність — щоб тільки адресат бачив, що ви передаєте;

забезпечити цілісність — щоб повідомлення не підмінили по дорозі;

перевірити автентичність — щоб переконатися, що інформація дійсно надійшла від тієї людини, яка її надіслала; у деяких випадках — навіть заперечення авторства, коли потрібно забезпечити анонімність.

Криптографія об'єднує математику, логіку, комп’ютерні науки та творчість. Це справжня гра розумів: одні шукають способи захистити інформацію, інші — намагаються знайти лазівки. І від того, наскільки сильний шифр, часто залежить доля не тільки приватної особи, а й цілої країни. Історично вона починалась з простих шифрів на кшталт шифру Цезаря, а зараз перетворилась на складні алгоритми, які захищають мільйони транзакцій щосекунди. Але головна ідея залишилась незмінною: надійно сховати інформацію від чужих очей.

Сучасні виклики та завдання криптографії

Сьогоднішній світ буквально крутиться навколо інформації. Ми щодня щось пишемо, надсилаємо, зберігаємо, оплачуючи покупки онлайн, ведучи переписки, передаючи фото, документи, коди — і все це, звісно, хочеться тримати подалі від чужих очей. Саме для цього існує криптографія. Але вона, як і все у світі технологій, має свої виклики.

1. Комп’ютери стають надто розумними

Ще не так давно шифри, які були створені у 70–80-х роках, вважалися наднадійними. Але сучасні комп’ютери, навіть не надто дорогі, вже можуть

розшифрувати деякі з них за лічені хвилини. Тобто шифр, який колись тримав державні таємниці, сьогодні може не витримати атаки звичайного ноутбука.

2. Штучний інтелект — і помічник, і загроза

ШІ може аналізувати величезні обсяги даних, шукати вразливості, підбирати ключі значно швидше, ніж людина. І якщо його використовують добрі хлопці — чудово. Але що, якщо до нього доберуться хакери? Це відкриває нову еру криптовійн.

3. Квантова загроза

І тут на горизонті з’являється квантовий комп’ютер. Це не просто “швидший комп’ютер” — це зовсім інший рівень обчислень. Шифри, які тримають захист банків, месенджерів і навіть військових каналів зв’язку, можуть зламуватись квантовими машинами в рази швидше. Усі наші цифрові замки — під загрозою. Саме тому зараз активно шукають “постквантові” алгоритми, які зможуть вистояти.

4. Забагато даних — занадто мало часу

Уявіть, скільки всього щодня створюється: сторіс, фото, файли, листи, онлайн-замовлення. І все це треба захищати. Але просто зробити “надійний” шифр уже не досить — він ще має бути швидким, зручним і працювати на різних пристроях. Інакше ніхто ним не буде користуватись.

5. Люди — теж частина системи

Навіть найкрутіший захист не спрацює, якщо користувач використовує пароль “123456” або натискає на фішингове посилання. Тому одне з головних завдань криптографії зараз — бути простою й зрозумілою. Вона має “працювати у фоновому режимі”, не вимагаючи від звичайної людини знань математики чи програмування.

Класифікація криптографічних алгоритмів

Уявімо криптографію як інструментальну скриньку — в ній є різні інструменти для різних задач. Щоб не переплутати гайковий ключ із викруткою, потрібно знати, який алгоритм для чого підходить. У криптографії всі методи шифрування можна умовно поділити за двома основними ознаками: симетричність (за типом ключа) і структура шифрування (як саме обробляється інформація — блоками чи потоком).

Симетричні та асиметричні алгоритми

Симетричні алгоритми працюють за принципом: один ключ — на все. Тобто і для шифрування, і для розшифрування використовується один і той самий ключ. Уявіть, що у вас є замок і один-єдиний ключ до нього. Ви можете передати замок іншій людині, але їй теж доведеться мати цей самий ключ. А отже — потрібно передати його безпечно, що іноді буває непросто.

Приклад: AES, DES.

Переваги:

- *швидка робота;*
- *простота реалізації;*
- *ефективність для великих обсягів даних.*

Недолік — потрібно якось передати ключ, не допустивши його перехоплення.

Асиметричні алгоритми вже складніші — тут використовуються дві частини ключа: один — відкритий (публічний), інший — закритий (приватний). Публічний ключ можна давати всім, а приватний тримати у секреті. Якщо хтось шифрує дані вашим відкритим ключем — розшифрувати їх зможете лише ви, використовуючи свій приватний.

Приклад: RSA, ECC.

Переваги:

зручно для обміну даними через відкриті канали;

можна створити цифровий підпис;

не треба передавати секретний ключ.

Недолік — повільніше за симетричні алгоритми, особливо при великому обсязі даних.

Блокові та потокові шифри

Блокові шифри обробляють дані шматками (або блоками) фіксованого розміру. Наприклад, текст розбивається на частини по 128 бітів, і кожен блок шифрується окремо. Ці шифри надійні та часто використовуються у стандартних протоколах (наприклад, HTTPS).

Приклад: AES, Blowfish.

Переваги:

висока надійність;

зручно для зберігання інформації.

Недолік — обробка може бути трохи повільнішою на малих обсягах даних.

Потокові шифри, навпаки, працюють із даними по одному біту або байту. Вони зручно накладаються “на льоту”, тому їх часто використовують для шифрування потокового відео, аудіо або швидких месенджерів.

Приклад: RC4.

Переваги:

швидкість;

гнучкість для потокових даних.

Недолік — трохи складніше гарантувати ідеальну безпеку, якщо ключ повторно використовується.

Таким чином, у реальному житті криптографи часто комбінують різні алгоритми. Наприклад, симетричний шифр — для швидкої роботи з даними, а асиметричний — для безпечної передачі ключа. Така система називається гібридною — і вона лежить в основі більшості сучасних протоколів захисту, якими ми користуємося щодня.

Розділ 2. Аналіз популярних алгоритмів шифрування

2.1. Симетричні алгоритми

- ***DES (Data Encryption Standard)***

Структура та принцип роботи

DES базується на концепції симетричного блочного шифрування — тобто для зашифрування та розшифрування інформації використовується один і той самий ключ. Алгоритм працює з фіксованими блоками даних розміром 64 біти. Кожен блок послідовно обробляється у 16 раундах перетворень, у яких використовуються підключі, що генеруються з основного 56-бітного ключа.

Основні операції, які застосовуються на кожному раунді:

Перестановки бітів (тобто зміна їх порядку),

Підстановка (S-box),

XOR (додавання по модулю 2),

Розділення та об’єднання даних.

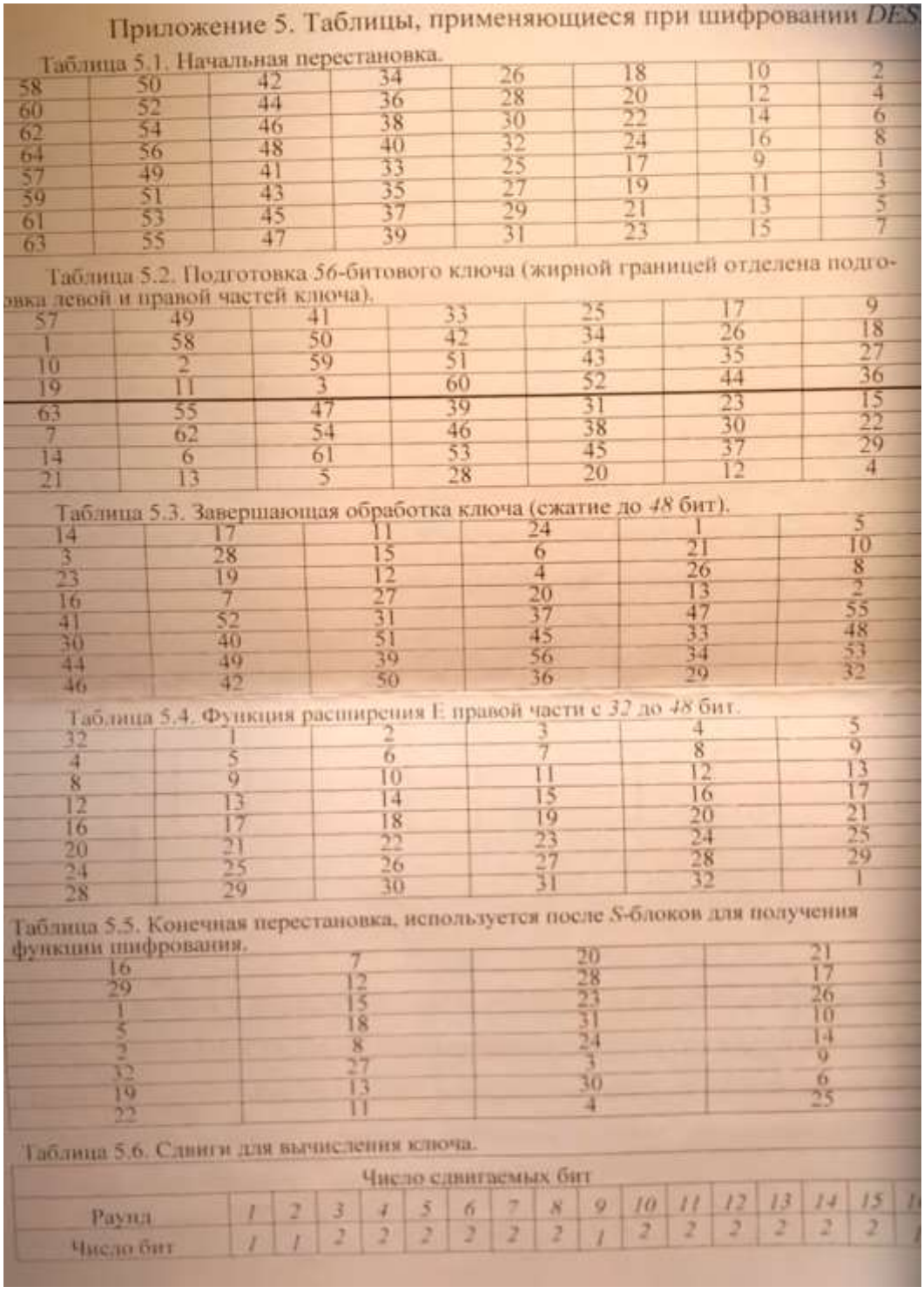
Ця багатоетапна структура забезпечує заплутаність і дифузію — дві ключові властивості безпечного шифрування, визначені Клодом Шенноном.

DES

Таблица кодировок символов кириллицы.

Unicode	win-1251	КОИ8	CP866	ISO8859-5		Unicode	win-1251	КОИ8	CP866	ISO8859-5	
0x0401	0xA8	0xB3	0xf0	0xA1	Ё	0x0430	0xE0	0xC1	0xa0	0xD0	а
0x0410	0xC0	0xE1	0x80	0xB0	А	0x0431	0xE1	0xC2	0xa1	0xD1	б
0x0411	0xC1	0xE2	0x81	0xB1	Б	0x0432	0xE2	0xD7	0xa2	0xD2	в
0x0412	0xC2	0xF7	0x82	0xB2	В	0x0433	0xE3	0xC7	0xa3	0xD3	г
0x0413	0xC3	0xE7	0x83	0xB3	Г	0x0434	0xE4	0xC4	0xa4	0xD4	д
0x0414	0xC4	0xE4	0x84	0xB4	Д	0x0435	0xE5	0xC5	0xa5	0xD5	е
0x0415	0xC5	0xE5	0x85	0xB5	Е	0x0436	0xE6	0xD6	0xa6	0xD6	ж
0x0416	0xC6	0xF6	0x86	0xB6	Ж	0x0437	0xE7	0xDA	0xa7	0xD7	з
0x0417	0xC7	0xFA	0x87	0xB7	З	0x0438	0xE8	0xC9	0xa8	0xD8	и
0x0418	0xC8	0xE9	0x88	0xB8	И	0x0439	0xE9	0xCA	0xa9	0xD9	й
0x0419	0xC9	0xEA	0x89	0xB9	Й	0x043A	0xEA	0xCB	0xaa	0xDA	к
0x041A	0xCA	0xEB	0x8a	0xBA	К	0x043B	0xEB	0xCC	0xab	0xDB	л
0x041B	0xCB	0xEC	0x8b	0xBB	Л	0x043C	0xEC	0xCD	0xac	0xDC	м
0x041C	0xCC	0xED	0x8c	0xBC	М	0x043D	0xED	0xCE	0xad	0xDD	н
0x041D	0xCD	0xEE	0x8d	0xBD	Н	0x043E	0xEE	0xCF	0xae	0xDE	о
0x041E	0xCE	0xEF	0x8e	0xBE	О	0x043F	0xEF	0xD0	0xaf	0xDF	п
0x041F	0xCF	0xF0	0x8f	0xBF	П	0x0440	0xF0	0xD2	0xe0	0xE0	р
0x0420	0xD0	0xF2	0x90	0xC0	Р	0x0441	0xF1	0xD3	0xe1	0xE1	с
0x0421	0xD1	0xF3	0x91	0xC1	С	0x0442	0xF2	0xD4	0xe2	0xE2	т
0x0422	0xD2	0xF4	0x92	0xC2	Т	0x0443	0xF3	0xD5	0xe3	0xE3	у
0x0423	0xD3	0xF5	0x93	0xC3	У	0x0444	0xF4	0xC6	0xe4	0xE4	ф
0x0424	0xD4	0xE6	0x94	0xC4	Ф	0x0445	0xF5	0xC8	0xe5	0xE5	х
0x0425	0xD5	0xE8	0x95	0xC5	Х	0x0446	0xF6	0xC3	0xe6	0xE6	ц
0x0426	0xD6	0xE3	0x96	0xC6	Ц	0x0447	0xF7	0xDE	0xe7	0xE7	ч
0x0427	0xD7	0xFE	0x97	0xC7	Ч	0x0448	0xF8	0xDB	0xe8	0xE8	ш
0x0428	0xD8	0xFB	0x98	0xC8	Ш	0x0449	0xF9	0xDD	0xe9	0xE9	щ
0x0429	0xD9	0xFD	0x99	0xC9	Щ	0x044A	0xFA	0xDF	0xea	0xEA	ъ
0x042A	0xDA	0xFF	0x9a	0xCA	Ъ	0x044B	0xFB	0xD9	0xeb	0xEB	ы
0x042B	0xDB	0xF9	0x9b	0xCB	Ы	0x044C	0xFC	0xD8	0xec	0xEC	ь
0x042C	0xDC	0xF8	0x9c	0xCC	Ь	0x044D	0xFD	0xDC	0xed	0xED	э
0x042D	0xDD	0xFC	0x9d	0xCD	Э	0x044E	0xFE	0xC0	0xee	0xEE	ю
0x042E	0xDE	0xE0	0x9e	0xCE	Ю	0x044F	0xFF	0xD1	0xef	0xEF	я
0x042F	0xDF	0xF1	0x9f	0xCF	Я	0x0451	0xB8	0xA3	0xf1	0xF1	ё

Рис. 1



Приложение 5. Таблицы, применяющиеся при шифровании *DES*

Таблица 5.1. Начальная перестановка.

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

Таблица 5.2. Подготовка *56*-битового ключа (жирной границей отделена подготовка левой и правой частей ключа).

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
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

Таблица 5.3. Завершающая обработка ключа (сжатие до *48* бит).

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

Таблица 5.4. Функция расширения E правой части с *32* до *48* бит.

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

Таблица 5.5. Конечная перестановка, используется после *S*-блоков для получения функции шифрования.

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

Таблица 5.6. Сдвиги для вычисления ключа.

	Число сдвигаемых бит															
Раунд	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*1*
Число бит	*1*	*1*	*2*	*2*	*2*	*2*	*2*	*2*	*1*	*2*	*2*	*2*	*2*	*2*	*2*	*1*

Рис. 2

Таблица 5.7. Функции преобразования S_1, S_2, ..., S_8.

Номер строки	Номера столбцов															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
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	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4	
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	
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	15	8	
3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11	

Рис. 3

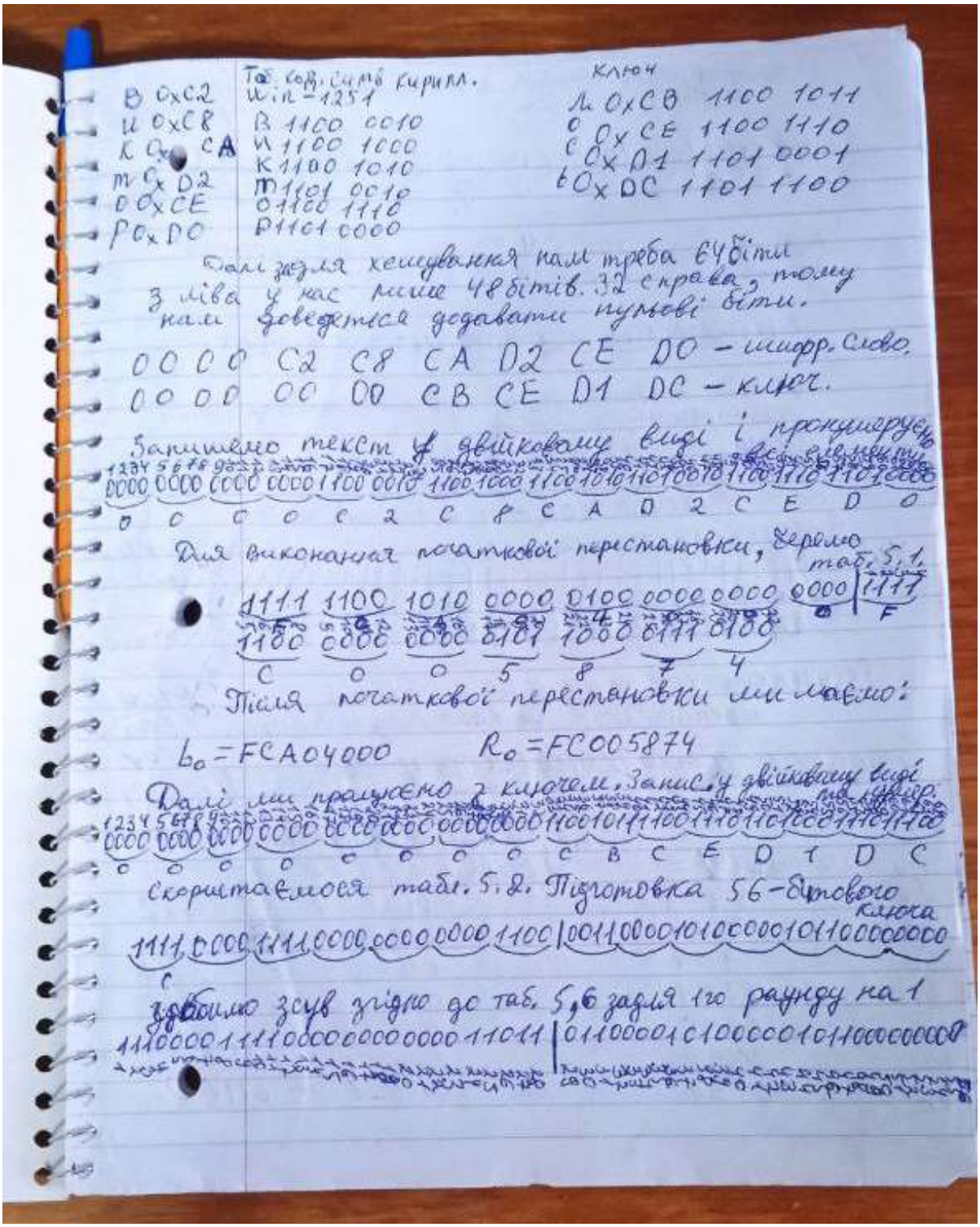

Таб. сов. симв кирилл.
Win-1251
В 0xC2 В 1100 0010
И 0xC8 И 1100 1000
К 0xCA К 1100 1010
Т 0xD2 Т 1101 0010
Е 0xCE Е 1100 1110
Р 0xDO Р 1101 0000
Ключ
Л 0xCB 1100 1011
О 0xCE 1100 1110
С 0xD1 1101 0001
Ь 0xDC 1101 1100
Далі задля хешування нам треба 64 біти
З ліва у нас лише 48 бітів. 32 справа, тому
нам доведеться додавати нульові біти.
0000 C2 C8 CA D2 CE DO – шифр. слово.
0000 00 00 CB CE D1 DC – ключ.
Запишемо текст у двійковому виді і пронумеруємо
Для виконання початкової перестановки, беремо
таб. 5.1.
Після початкової перестановки ми маємо:
L_0 = FCA04000 R_0 = FC005874
Далі ми працюємо з ключем. Запис. у двійковому виді
Скористаємося таб. 5.2. Підготовка 56-бітового
ключа
Зробимо зсув зліва до таб. 5,6 задля 1го раунду на 1

Рис. 4

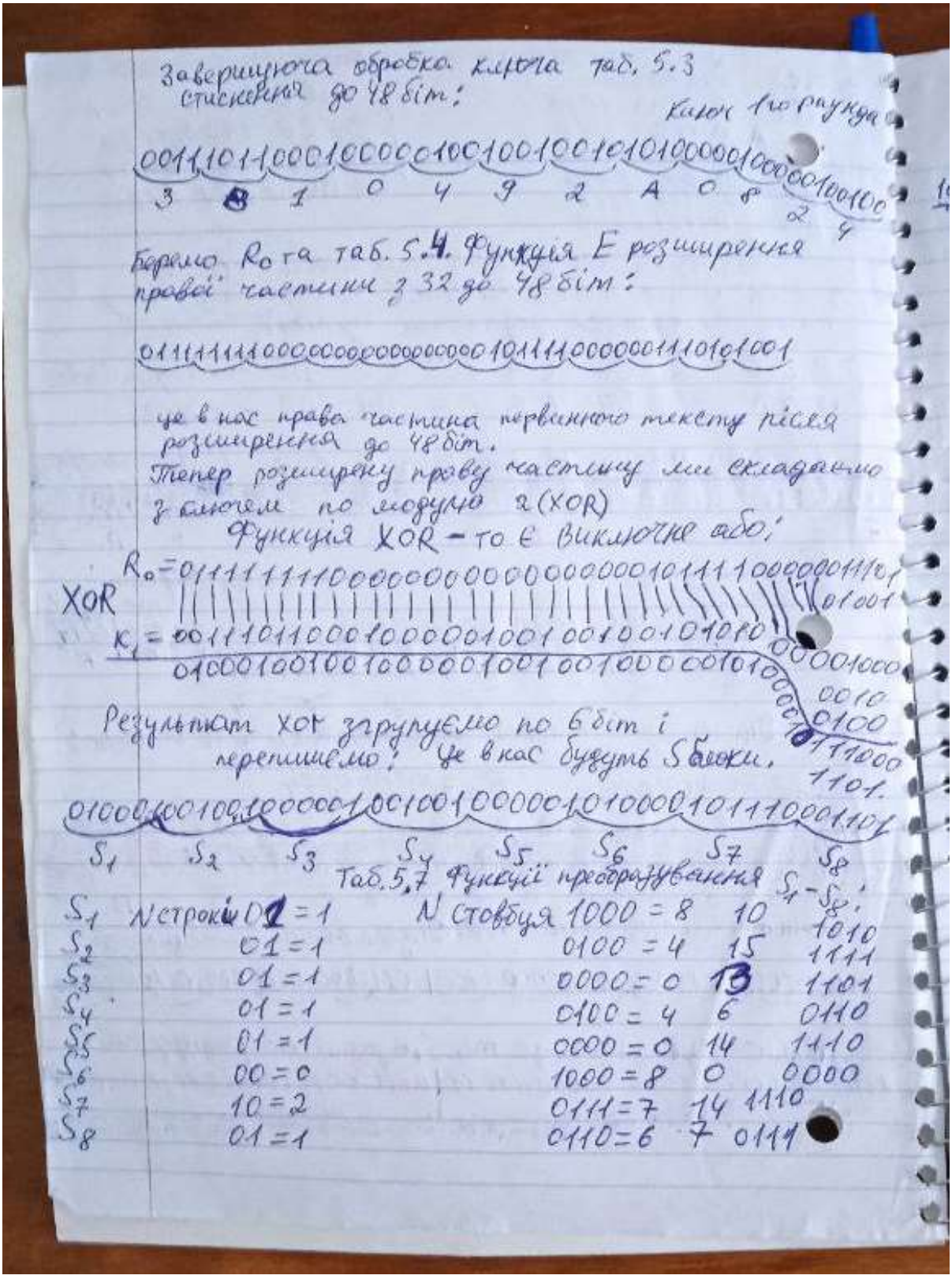

Завершуюча обробка ключа таб. 5.3
стискання до 48 біт:
Ключ 1 ітерації
00111011000010000010011001001010100000100000100100
3 B 1 0 4 9 2 A 0 8 2 4
Беремо R_0 та таб. 5.4. Функція E розширення
правої частини з 32 до 48 біт:
011111111000000000000000010111100000001110101001
це в нас права частина первинного тексту після
розширення до 48 біт.
Тепер розширену праву частину ми складаємо
з ключем по модулю 2 (XOR)
Функція XOR - то є виключне або;
R_0=011111111000000000000000010111100000001110101001
XOR
K_1=001110110000100000100100100101010000010000
010001001001000001001001000001010000
Результат XOR згрупуємо по 6 біт і
перепишемо! Це в нас будуть S блоки.
010001001001000001001001000001010000101110001101
S_1 S_2 S_3 S_4 S_5 S_6 S_7 S_8
Таб. 5.7 Функції преобразування S_1-S_8;
S_1 N строки 01 = 1 N стовбця 1000 = 8 10 1010
S_2 01 = 1 0100 = 4 15 1111
S_3 01 = 1 0000 = 0 13 1101
S_4 01 = 1 0100 = 4 6 0110
S_5 01 = 1 0000 = 0 14 1110
S_6 00 = 0 1000 = 8 0 0000
S_7 10 = 2 0111 = 7 14 1110
S_8 01 = 1 0110 = 6 7 0111

Рис. 5

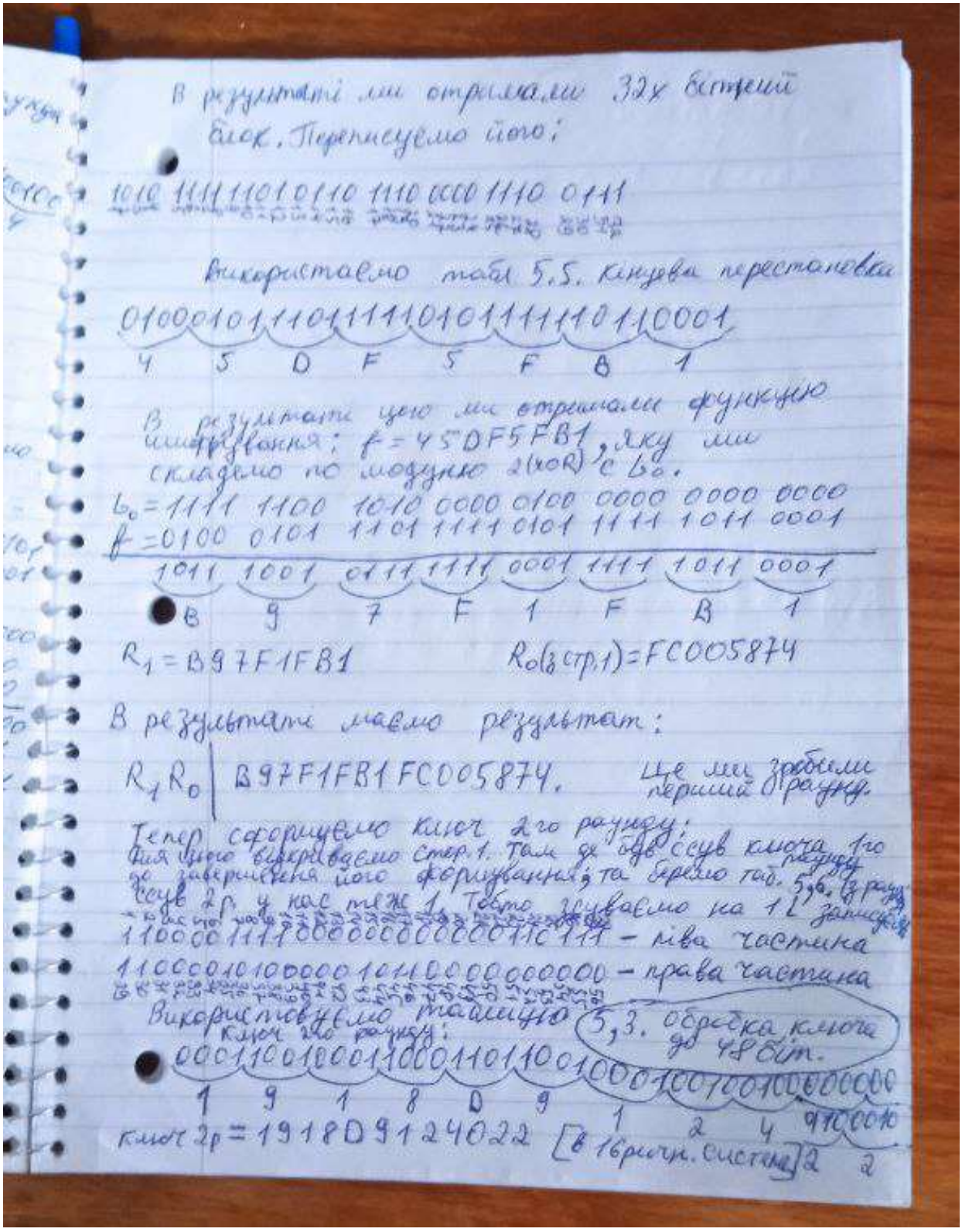
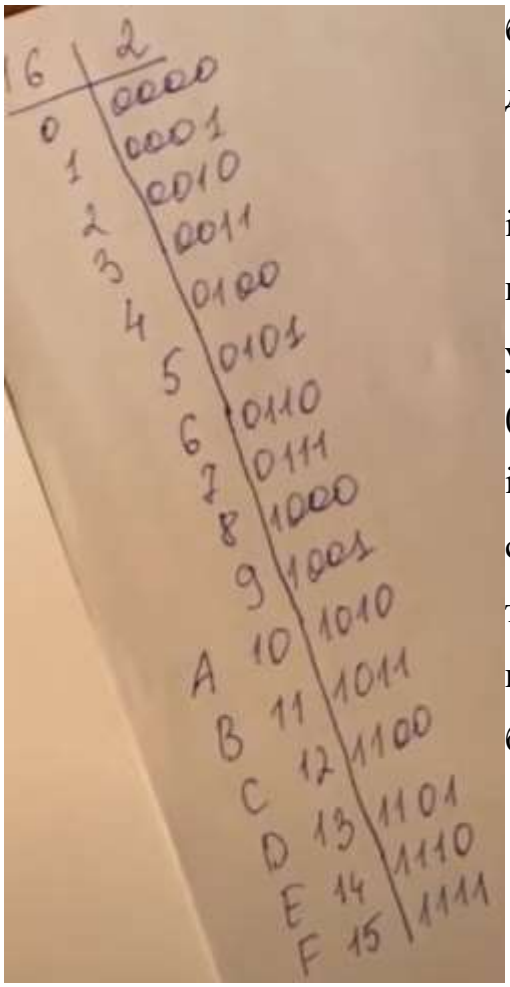

В результаті ми отримали 32х бітовий блок. Перепишемо його:
1010 1111 1101 0110 1110 0001 1110 0111
Використаємо табл 5.5. кінцева перестановка
01000101 11011111 01011111 10110001
4 5 D F 5 F B 1
В результаті цього ми отримали функцію шифрування: f = 45DF5FB1, яку ми складемо по модулю 2(xoR) с b0.
b0 = 1111 1100 1010 0000 0100 0000 0000 0000
f = 0100 0101 1101 1111 0101 1111 1011 0001
1011 1001 0111 1111 0001 1111 1011 0001
B 9 7 F 1 F B 1
R1 = B97F1FB1 R0(з стр.1) = FC005874
В результаті маємо результат:
R1R0 | B97F1FB1FC005874. Це ми зробили перший раунд.
Тепер сформуємо ключ 2го раунду:
Для цього використаємо стор.1. Там де обрахунок ключа 1го раунду де завершення його формування та беремо таб. 5,6. (2 раунд
Зсув 2р у нас теж 1. Тобто зсуваємо на 1 і записуємо
11000011 11100000 00000000 11000111 – ліва частина
11000010 10000010 11000000 00000000 – права частина
Використовуємо таблицю 5,3. обрізка ключа до 48 біт.
Ключ 2го раунду:
000110 010000 110000 110110 010000 000100 100100 000000
1 9 1 8 D 9 1 2 4 0 2 2
ключ 2р = 1918D91240 22 [в 16річн. системі]

Рис. 6



Ця технологія була актуальна доки не була легко зламана компьютером за двадцять чотири години у двотисячних.

На сьогодні DES вважається застарілим і небезпечним для будь-якого практичного використання. Він був офіційно замінений у 2001 році новим стандартом — AES (Advanced Encryption Standard). DES має історичну цінність, однак у реальних системах більше не застосовується. Тож треба визнати, що алгоритм 3x DES до недавнього часу ще використовувася та був замінений на більш сучасныший AES.

Рис. 7

Поетапний опис реалізації:

Покроково алгоритм шифрування DES

Шифруватимемо ім'я "Виктор". Ключем в нас буде "Лось". Я вибрав російськомовні літери, бо не зміг знайти необхідні таблиці рідною мовою.

1.Спочатку ми відкриваємо таблицю символи кирилиці, рис.1. З доступних кодіровок на цей раз оберемо win-1251. Тепер вписуємо кожну букву з цієї таблиці напроти букв з моїм шифрословом та ключем.

2. Далі перекладаємо у двійковий код дві останні цифри кодованих літер за допомогою рис.7, який був виписний заздалегідь для зручності.

3. Для хешування потрібно 64 біти і в нашому шифрослові і стільки ж у ключі. Виктор = 48, Лось - 32. Це означає, що ми будемо додавати додаткові "нульові" біти спочатку та записуємо їх.
4. Нижче знову переписуємо ширфотекст у двійковому вигляді та нумеруємо кожну цифру.
5. Далі відкриваємо рис.2 та звертаємо увагу на таблицю 5.1 - "Начальная перестановка".

Звіряємо всі цифри таблиці з порядковими номерами нулів та одиниць і просто виписуємо їх до кінця таблиці.

6. Переводимо по 4 біти у шістнадцятирічну систему числення. Розділяємо на 2 половини.

Після первинної перестановки виписуємо Lo та Ro.

7. Далі працюємо з ключем. Переписуємо згори. Нумеруємо та для наочності помічаємо що є що на 16 бітах.
8. Підготовка 56 бітного ключа. За допомогою таблиці 5.2 (рис. 2) здійснюємо підготовку 56 бітного ключа. Переписуємо таблицю таким ж чином, як робили з першою таблицею.
9. Далі треба зробити зсув, для цього дивимось таблицю 5.6 (рис.2) "сдвиги для вычисления ключа". В першому раунді здійснюємо зсув на 1 біт, як у таблиці. Ділимо на дві частини та зсуваємо на 1 біт кожну частину. Нумеруємо кожну цифру.
10. Тепер скористуємось таблицею 5.3 (рис.2) і здійснюємо завершуючу обробку ключа. Дивимось і таким же чином виписуємо по порядкових номерах всі нулі та одиниці, орієтуючись на цифри у таблиці. Знову по 4 біти переводимо з двійкового коду у шістнадцятирічну систему числення, та виписуємо нижче.
11. Тепер скористуємося значенням Ro, тобто правою частиною нулів та одиниць вище. Переписуємо праву частину одразу після початкової

перестановки та нумеруємо. Далі користуємося таблицею 5.4. (рис.2) і переводимо даний ключ з 32 біт до 48 біт. Таблицю використовуємо як минулого разу.

12. Тепер ми маємо 48 бітний код і хоруємо її з нашим 48 бітним ключем першого раунду вище. Хор - це логічне або.
13. Результат XOR згрупуємо по 6 біт і перепишемо. Кожну групу пронумеруємо S1,S2,S3... тощо.
14. Виписуємо всі набори значень по 6 біт (S) зліва. З кожного шестибітного S згори виписуємо крайні цифри - це буде номер строки. Виписуємо 4 цифри що залишились - це буде номер стовпця. Далі беремо (рис.7) та переводимо двійковий код у шістнадцятирічну систему. Тепер маємо значення для таблиці 5.7 (рис.3). Функціями перетворення є S1,S2,... S8. Виписуємо отримані цифри справа та знову переводимо їх у двійкову систему числення. Виписуємо по порядку та нумеруємо їх. Бітів повинно бути 32.
15. Тепер знову дивимось рис.2 та звертаємо увагу на таблицю 5.5. Робимо кінцеву перестановку після S-блоків та отримуємо функцію шифрування.
16. Переписуємо Lo та складаємо її по модулю XOR. Отримане ми виділяємо по 4біти та переводимо в 16ти річну систему зачислення згідно рис.7.

Переписуємо спочатку те, що вийшло і дописуємо версію Ro в шістнадцятирічній системі числення з початку. Це і буде результат першого раунду шифрування.

17. Ключ другого раунду: Для отримання ключа 2го раунду треба звернутися до рис. 4 та знайти зсув згідно таблиці 5.6, повернутися до цієї таблиці та зробити зсув, згідно даної таблиці для другого

раунду. Далі формуємо ключ другого раунду використовуючи таблицю 5.3(рис 2), результатом буде ключ другого раунду.

AES (розширений стандарт шифрування): принцип роботи, сфера застосування.

AES — це сучасний і дуже надійний спосіб захищати цифрову інформацію. Якщо спростити, він допомагає перетворити зрозумілий текст на набір незрозумілих символів, які можуть бути розшифровані лише тим, хто має правильний ключ. Він використовується всюди: коли ти входиш до свого банку через телефон, коли спілкуєшся в месенджері, коли Wi-Fi захищений паролем або коли твій комп’ютер шифрує диск, щоб ніхто не зміг дістати твої дані без дозволу. Як працює AES: коротко, але зрозуміло Уяви, що твій текст ділять на невеликі однакові шматочки — блоки по 16 байт (це 128 біт, або приблизно 16 символів). І кожен із цих шматочків обробляється окремо. AES — це симетричне шифрування, тобто для зашифровування й розшифровування використовується один і той самий ключ. Тож якщо зловмисник його не знає — для нього все, що залишиться від повідомлення, — це хаос із байтів. А всередині алгоритму все працює так: Кожен блок проходить через низку перетворень — у кілька "раундів", і чим довший ключ, тим більше раундів. Наприклад, для 128-бітного ключа — 10 раундів, а для 256-бітного — 14. У кожному раунді виконуються такі дії: Заміна байтів (SubBytes): Уяви, що кожну букву замінюють на іншу за спеціальною таблицею. Це ускладнює будь-яку закономірність. Зміщення рядків (ShiftRows): У таблиці, де розміщені байти, рядки просто зсуваються. Це ще більше перемішує дані. Міксування стовпців (MixColumns): Тепер ще й стовпці вмішуються за спеціальними математичними правилами. У результаті навіть одна зміна у тексті повністю змінить увесь зашифрований блок. Додавання ключа (AddRoundKey): І на завершення — до всього цього

додається частина ключа. Без нього шифр не має сенсу. Де використовується AES у реальному житті? Практично всюди, де потрібен захист інформації: У веб-сайтах із HTTPS (навіть ця сторінка!) У месенджерах на кшталт WhatsApp або Signal У банківських додатках У Wi-Fi мережах У зашифрованих файлах, дисках, архівах Чому він такий популярний? Він дуже швидкий, навіть на простих пристроях. Його важко зламати — якщо правильно використовувати ключ. Він відкритий і перевірений багатьма експертами. Навіть через понад 20 років використання AES досі залишається золотим стандартом. Це не просто алгоритм — це основа цифрової безпеки.

RSA

Уявимо, що ми хочемо надіслати комусь важливу інформацію — скажімо, номер банківської картки — людині, з якою ніколи не зустрічались і не домовлялись про якийсь спільний секрет. Як передати ці дані без страху, що хтось перехопить їх дорогою?

Саме для таких ситуацій і був створений алгоритм RSA — один із найвідоміших методів шифрування, який дозволяє двом людям обмінюватися зашифрованими повідомленнями, навіть якщо вони бачать одне одного вперше. Що таке RSA?

RSA — це асиметричний криптографічний алгоритм. Це означає, що замість одного секретного ключа тут використовуються два різних:

Відкритий ключ — для шифрування (ним може користуватися будь-хто);

Закритий ключ — для розшифрування (його має тільки власник і тримає в таємниці).

Такий підхід дозволяє, наприклад, розмістити відкритий ключ у відкритому доступі (на сайті, у підписі листа тощо), і при цьому не хвилюватися: повідомлення, зашифроване цим ключем, може розшифрувати лише той, у кого є відповідний закритий ключ.

Назва RSA — це перші літери прізвищ трьох математиків, які створили цей алгоритм у 1977 році: **Рівест**, **Шамір** і **Адлеман**.

Як це працює — простою мовою

1. Створення ключів

Спочатку вибираються два великі прості числа (назвемо їх p і q).

Обчислюється їх добуток: $n = p \times q$. Це частина майбутнього ключа.

Далі розраховується математична характеристика: $\varphi(n) = (p-1) \times (q-1)$.

Потім вибирається число e — воно повинно не мати спільних дільників із $\varphi(n)$. Часто беруть 65537 — воно досить швидке й безпечне.

Обчислюється d — число, яке «зворотне» до e (з математичної точки зору).

У результаті:

Відкритий ключ — це пара (e, n). Його можна передати будь-кому.

Закритий ключ — це пара (d, n). Його зберігає тільки власник.

2. Шифрування

Уявімо, що хтось хоче надіслати нам повідомлення. Текст (наприклад, "Привіт") перетворюється у набір чисел (наприклад, через Unicode).

Кожне з цих чисел шифрується з використанням вашого відкритого ключа. Виходить зашифроване повідомлення — набір незрозумілих чисел, які виглядають як випадкові.

3. Розшифрування

Ви, як власник закритого ключа, берете це зашифроване повідомлення і обробляєте його у зворотному напрямку. І отримуєте назад оригінальний текст — "Привіт", без спотворень.

Навіть якщо зловмисник перехопить наш відкритий ключ, він не зможе розшифрувати повідомлення без закритого ключа. А щоб його підібрати, доведеться розкласти надзвичайно велике число (n) на два простих множники — це складна задача, яка на сучасних комп’ютерах займає астрономічно багато часу.

- Де ми зустрічаємо RSA?

Насправді — майже всюди:

Браузер і HTTPS — коли ми відкриваємо сайт і бчимо замочок у рядку адреси, там працює RSA для початкової безпечної домовленості.

Цифрові підписи — підтвердження, що документ справжній, а не підробка.

Електронна пошта (PGP, GPG) — для шифрування особистих листів.

Банківські системи — для підтвердження транзакцій і перевірки користувача.

Плюси і мінуси RSA

Переваги:

Безпечний навіть у відкритому середовищі.

Підходить для цифрових підписів.

Можна використовувати без обміну секретами.

Недоліки:

Працює повільніше за симетричні алгоритми (AES, ChaCha20).

- Не підходить для великих обсягів даних (шифрує лише короткі блоки).
- Вразливий до майбутніх квантових атак (тому розробляються альтернативи).

RSA — це класика криптографії. Його принципи прості, але глибокі. Саме він дав змогу побудувати сучасний Інтернет із його системами захисту, цифровими підписами та безпечними транзакціями. І хоча на горизонті вже з’являються нові алгоритми, RSA залишається надійною основою цифрової безпеки.

ECC (Elliptic Curve Cryptography): переваги та проблеми.

2.3. Гібридні підходи Приклад:

Тут ми розглянемо, як працює шифрування даних, коли ми заходимо на сайт з інтернет банкінгу**.**

Уявімо: Ми відкриваємо сайт банку — він починається з https://. Це значить, що з’єднання буде **зашифрованим**, і ніхто в мережі не зможе підглянути, що ми робимо (вводимо логін, пароль, читаємо виписку, переказуємо гроші тощо).

А тепер — що відбувається за лаштунками:

Крок 1: Асиметричне шифрування — для обміну ключами

Коли ми вперше заходимо на сайт банку, наш браузер отримує публічний ключ сайту (той самий "відкритий ключ", який не шкода показати всім). Він використовується для зашифровування нашого повідомлення — наприклад, симетричного ключа, який ми з сайтом будемо далі використовувати. Тільки сам сайт (сервер) має закритий ключ, яким він може

розшифрувати це повідомлення. Публічний ключ — як поштова скринька, куди всі можуть кидати листи, але витягти їх може тільки власник ключа.

Крок 2: Симетричне шифрування — для швидкої роботи

Після того як браузер і сайт "домовилися" про секретний симетричний ключ (через асиметричний канал), вони починають обмінюватись даними вже через більш швидке симетричне шифрування. Наприклад, використовується AES або ChaCha20.

Чому так? Бо симетричні шифри:

працюють набагато швидше;

краще підходять для обробки великих обсягів даних;

зменшують навантаження на сервери. Дані типу "переказати 500 грн", "показати виписку за місяць", "вийти з кабінету" — шифруються симетричним шифром.

Крок 3: Кінець сесії — ключі більше не потрібні

Коли ми виходимо з сайту, симетричний ключ видаляється з пам’яті браузера. При наступному заході — усе починається знову. Це потрібно для безпеки: навіть якщо хтось зламає твоє з’єднання пізніше, він не отримає попередні ключі.

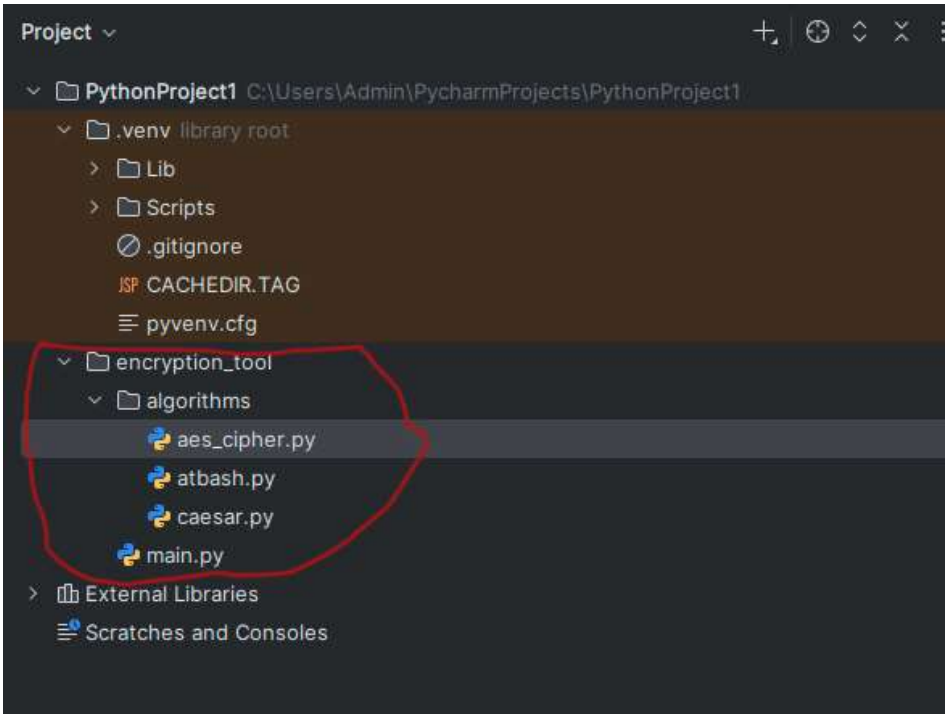
Розділ 3. Розробка програми на Python

Step 1: Постановка завдання:

Перше, що треба розробити, це ідея програми. Ідея полягатиме в тому, щоб створити програму на основі кількох з вищеперелічених алгоритмів. Тож спробуємо:

Перший алгоритм, який я додам до своєї програми це простий алгоритм Цезаря, наступним буде знаменитий Атбаш, яким ще колись шифрували Біблію. Ще я планую додати туди AES з хеш шифруванням ключа через хеш функцію Sha-256. Він захешує мій ключ до розміру 256 бітів. Пайчарм з кріптографічнми бібліотеками мені в цьому допоможе.

Зробимо таку структуру файлів. Тобто створимо папку encryption_tool та додамо в неї спочатку наш основний файл коду main.py, а також папку з нашими майбутніми алгоритмами algorithms.



Я думаю є можливість почати з шифру Цезаря для caesar.py:

Для кожного типу кодування, звісно треба зробити і можливість розкодування для перевірки.

```
# Повні українські алфавіти: великі та малі літери
ukr_upper = 'АБВГҐДЕЄЖЗИІЇЙКЛМНОПРСТУФХЦЧШЩЬЮЯ'  # 33 великі літери
ukr_lower = 'абвгґдеєжзиіїйклмнопрстуфхцчшщьюя'  # 33 малі літери

# Функція шифрування тексту шифром Цезаря
def encrypt(text, shift):
    result = ''  # Порожній рядок, куди будемо додавати зашифровані символи

    for char in text:  # Перебираємо кожен символ у вхідному тексті
        if char in ukr_upper:  # Якщо символ — велика українська літера
            index = ukr_upper.index(char)  # Знаходимо її позицію в алфавіті (0 до 32)
            new_index = (index + shift) % 33  # Додаємо зсув і обмежуємо в межах 33 літер (циклічно)
            result += ukr_upper[new_index]  # Додаємо нову літеру за зміщеним індексом
        elif char in ukr_lower:  # Якщо символ — мала українська літера
            index = ukr_lower.index(char)  # Знаходимо позицію в алфавіті
            new_index = (index + shift) % 33  # Зсув по колу
            result += ukr_lower[new_index]  # Додаємо зашифровану малу літеру
        else:
            result += char  # Якщо символ не літера (пробіл, цифра, пунктуація), залишаємо без змін

    return result  # Повертаємо повністю зашифрований текст

# Функція розшифрування — це шифрування з від’ємним зсувом
def decrypt(text, shift):
    return encrypt(text, -shift)  # Зсув у зворотному напрямку
```

atbash.py

```
# Шифр Атбаш для українського алфавіту

def atbash_encrypt(text: str) -> str:
    # Визначаємо великий український алфавіт
    alphabet_upper =
'АБВГҐДЕЄЖЗИІЇЙКЛМНОПРСТУФХЦЧШЩЬЮЯ'
    # Створюємо відповідний малий алфавіт, переводячи великий у

нижній регістр
    alphabet_lower = alphabet_upper.lower()
```

```
    # Рядок для збереження результату шифрування
    result = ''

    # Проходимо по кожному символу вхідного тексту
    for ch in text:
        # Якщо символ великий і є в алфавіті
        if ch in alphabet_upper:
            # Знаходимо позицію символу в алфавіті
            index = alphabet_upper.index(ch)
            # Додаємо до результату "дзеркальний" символ з кінця
алфавіту
            result += alphabet_upper[-index - 1]
        # Якщо символ малий і є в алфавіті
        elif ch in alphabet_lower:
            # Знаходимо позицію символу в алфавіті
            index = alphabet_lower.index(ch)
            # Додаємо до результату "дзеркальний" малий символ
            result += alphabet_lower[-index - 1]
        else:
            # Якщо символ не належить алфавіту (наприклад, пробіл,
цифра, пунктуація),
            # додаємо його без змін
            result += ch

    # Повертаємо зашифрований (або розшифрований) текст
    return result
```

```
# Визначаємо, що функція розшифрування — це та сама функція шифрування,
# бо Атбаш — саморозшифровний шифр (застосувавши її двічі, отримаємо початковий текст)
atbash_decrypt = atbash_encrypt  # Атбаш саморозшифровний
```

AES

```
from Crypto.Cipher import AES  # Імпортуємо AES для шифрування
import base64  # Імпортуємо base64 для кодування байтів у текст
from hashlib import sha256  # Імпортуємо sha256 для хешування ключа

def prepare_key(user_key: str, length: int = 32) -> bytes:
    """
    Перетворює рядковий ключ у байтовий ключ фіксованої довжини через SHA-256.

    Args:
        user_key (str): Вхідний ключ у вигляді рядка.
        length (int): Довжина ключа (16, 24 або 32 байти).

    Returns:
        bytes: Ключ потрібної довжини.
```

```
    Raises:


  ValueError: Якщо довжина ключа не підтримується AES.

    """
    if length not in (16, 24, 32):  # Перевіряємо, чи довжина ключа коректна
        raise ValueError("Довжина ключа має бути 16, 24 або 32 байти")
    hashed = sha256(user_key.encode('utf-8')).digest()  # Хешуємо ключ у байти через SHA-256
    return hashed[:length]  # Обрізаємо до потрібної довжини

def encrypt(plain_text: str, key_str: str, key_length: int = 32) -> str:
    """
    Шифрує текст за допомогою AES у режимі CFB.

    Args:
        plain_text (str): Текст для шифрування.
        key_str (str): Ключ у вигляді рядка.
        key_length (int): Довжина ключа (16, 24 або 32 байти).

    Returns:
        str: Зашифрований текст у форматі "IV:шифротекст" (base64).

    Raises:
        ValueError: Якщо ключ або текст некоректні.
    """
```

```
    try:
        key = prepare_key(key_str, key_length)  # Готуємо ключ потрібної довжини
        print(f"Підготовлений ключ (hex): {key.hex()}")  # Виводимо ключ

у hex для демонстрації
        cipher = AES.new(key, AES.MODE_CFB)  # Створюємо об’єкт AES у режимі CFB
        ct_bytes = cipher.encrypt(plain_text.encode('utf-8'))  # Шифруємо текст у байти
        iv = base64.b64encode(cipher.iv).decode('utf-8')  # Кодуємо IV у base64
        ct = base64.b64encode(ct_bytes).decode('utf-8')  # Кодуємо шифротекст у base64
        return f"{iv}:{ct}"  # Повертаємо IV і шифротекст, розділені двокрапкою
    except (ValueError, UnicodeEncodeError) as e:  # Ловимо помилки шифрування
        raise ValueError(f"Помилка шифрування: {str(e)}")  # Кидаємо виняток із деталями

def decrypt(enc_text: str, key_str: str, key_length: int = 32) -> str:
    """
    Розшифровує текст, зашифрований AES у режимі CFB.

    Args:
        enc_text (str): Зашифрований текст у форматі "IV:шифротекст" (base64).
```

```
        key_str (str): Ключ у вигляді рядка.
        key_length (int): Довжина ключа (16, 24 або 32 байти).

    Returns:
        str: Розшифрований текст.




 Raises:
        ValueError: Якщо формат зашифрованого тексту або ключ некоректні.
    """
    try:
        key = prepare_key(key_str, key_length)  # Готуємо ключ потрібної довжини
        print(f"Підготовлений ключ (hex): {key.hex()}")  # Виводимо ключ у hex для демонстрації
        iv, ct = enc_text.split(':')  # Розділяємо вхідний текст на IV і шифротекст
        iv = base64.b64decode(iv)  # Декодуємо IV із base64 у байти
        ct = base64.b64decode(ct)  # Декодуємо шифротекст із base64 у байти
        cipher = AES.new(key, AES.MODE_CFB, iv=iv)  # Створюємо об’єкт AES із IV
        pt = cipher.decrypt(ct)  # Розшифровуємо шифротекст
        return pt.decode('utf-8')  # Повертаємо розшифрований текст як рядок
```

```
    except (ValueError, base64.binascii.Error, UnicodeDecodeError) as e:
# Ловимо помилки
        raise ValueError(f"Помилка розшифрування: {str(e)}")  #
Кидаємо виняток із деталями

if __name__ == "__main__":



 try:

   text = "Привіт"  # Текст для шифрування
        key = "3254312"  # Ключ у вигляді рядка
        encrypted = encrypt(text, key)  # Шифруємо текст
        print("Зашифровано:", encrypted)  # Виводимо зашифрований
текст
        decrypted = decrypt(encrypted, key)  # Розшифровуємо текст
        print("Розшифровано:", decrypted)  # Виводимо розшифрований
текст
    except Exception as e:  # Ловимо будь-які помилки
        print(f"Помилка: {str(e)}")  # Виводимо повідомлення про
помилку
```

Main.py

Тут ми хотіли зробити консольне меню з можливістю шифрування та розшифрування кожним з цих алгоритмів, а також додати послідовне шифрування та розширування Цезарем, Атабаш і Аес, та всіма трьома алгоритмми послідовно, обирати потрібну опцію будемо цифрами:

```
# Імпорт модулів для шифрування: caesar (шифр Цезаря), aes_cipher (шифр AES), atbash (шифр Атбаш)
from algorithms import caesar, aes_cipher, atbash




# Основна функція програми

def main():
    # Виведення меню вибору режиму шифрування/розшифрування
    print("Виберіть режим:")
    print("1 - Тільки Цезар")  # Режим із використанням лише шифру Цезаря
    print("2 - Тільки AES")  # Режим із використанням лише шифру AES
    print("3 - Цезар + AES (послідовно)")  # Комбінація шифру Цезаря та AES
    print("4 - Тільки Атбаш")  # Режим із використанням лише шифру Атбаш
    print("5 - Цезар + Атбаш + AES (послідовно)")  # Комбінація трьох шифрів
    mode = input("Ваш вибір: ").strip()  # Зчитування вибору режиму та видалення пробілів

    # Виведення меню вибору дії
    print("Виберіть дію:")
    print("1 - Шифрувати")  # Шифрування тексту
```

```
    print("2 - Розшифрувати")  # Розшифрування тексту
    action = input("Ваш вибір (1/2): ").strip()  # Зчитування вибору дії

    # Перевірка коректності введеної дії
    if action not in ['1', '2']:
        print("Помилка: потрібно ввести 1 або 2.")  # Повідомлення про помилку, якщо дія некоректна return




 Визначення, чи виконується шифрування (True) чи розшифрування (False)
    is_encrypt = action == '1'

    # Зчитування вхідного тексту
    text = input("Введіть текст: ")

    # Режим 5: Комбіноване шифрування/розшифрування (Цезар → Атбаш → AES)
    if mode == '5':
        # Комбінований режим: Цезар + Атбаш + AES
        try:
            # Запит зсуву для шифру Цезаря
            shift = int(input("Введіть зсув для Цезаря: "))
        except ValueError:
            # Обробка помилки, якщо введено не число
            print("Зсув повинен бути числом.")
            return
```

```
        # Запит ключа для шифру AES
        aes_key = input("Введіть пароль для AES: ")

        if is_encrypt:
            # Шифрування: послідовно застосовуються Цезар, Атбаш, AES
            print("Початковий текст:", text)  # Виведення початкового тексту
            step1 = caesar.encrypt(text, shift)  # Застосування шифру Цезаря
            print("Після Цезаря:", step1)  # Виведення результату після



Цезаря
            step2 = atbash.atbash_encrypt(step1)  # Застосування шифру Атбаш
            print("Після Атбаш:", step2)  # Виведення результату після Атбаш
            result = aes_cipher.encrypt(step2, aes_key)  # Застосування шифру AES
            print("Після AES:", result)  # Виведення результату після AES
        else:
            # Розшифрування: послідовно застосовуються зворотні операції (AES → Атбаш → Цезар)
            try:
                print("Зашифрований текст:", text)  # Виведення вхідного зашифрованого тексту
                step1 = aes_cipher.decrypt(text, aes_key)  # Розшифрування
```

```
AES
            print("Після розшифрування AES:", step1)  # Виведення
результату після AES
            step2 = atbash.atbash_decrypt(step1)  # Розшифрування
Атбаш
            print("Після розшифрування Атбаш:", step2)  # Виведення
результату після Атбаш
            result = caesar.decrypt(step2, shift)  # Розшифрування Цезаря
            print("Після розшифрування Цезаря:", result)  # Виведення
остаточного результату
        except Exception as e:
            # Обробка помилок під час розшифрування (наприклад,
некоректний ключ AES)



            print("Помилка розшифрування:", e)

        return

    # Режим 3: Комбінація шифру Цезаря та AES
    elif mode == '3':
        # Цезар + AES
        try:
            # Запит зсуву для шифру Цезаря
            shift = int(input("Введіть зсув для Цезаря: "))
        except ValueError:
            print("Зсув повинен бути числом.")
            return
```

```
        # Запит ключа для шифру AES
        aes_key = input("Введіть пароль для AES: ")

        if is_encrypt:
            # Шифрування: Цезар → AES
            step1 = caesar.encrypt(text, shift)  # Застосування шифру
Цезаря
            result = aes_cipher.encrypt(step1, aes_key)  # Застосування
шифру AES
        else:
            # Розшифрування: AES → Цезар
            try:
                step1 = aes_cipher.decrypt(text, aes_key)  # Розшифрування
AES
                result = caesar.decrypt(step1, shift)  # Розшифрування Цезаря




      except Exception as e:
        print("Помилка розшифрування:", e)
                return
# Режим 1: Тільки шифр Цезаря
    elif mode == '1':
        # Тільки Цезар
        try:
            # Запит зсуву для шифру Цезаря
            shift = int(input("Введіть зсув для Цезаря: "))
        except ValueError:
```

```
            print("Зсув повинен бути числом.")
            return
        if is_encrypt:
            result = caesar.encrypt(text, shift)  # Шифрування Цезарем
        else:
            result = caesar.decrypt(text, shift)  # Розшифрування Цезарем

    # Режим 4: Тільки шифр Атбаш
    elif mode == '4':
        # Тільки Атбаш
        if is_encrypt:
            result = atbash.atbash_encrypt(text)  # Шифрування Атбаш
        else:
          result = atbash.atbash_decrypt(text)  # Розшифрування Атбаш




# Режим 2: Тільки шифр AES
    elif mode == '2':
        # Тільки AES
        aes_key = input("Введіть ключ для AES: ")  # Запит ключа для
AES
        if is_encrypt:
            result = aes_cipher.encrypt(text, aes_key)  # Шифрування AES
        else:
            try:
```

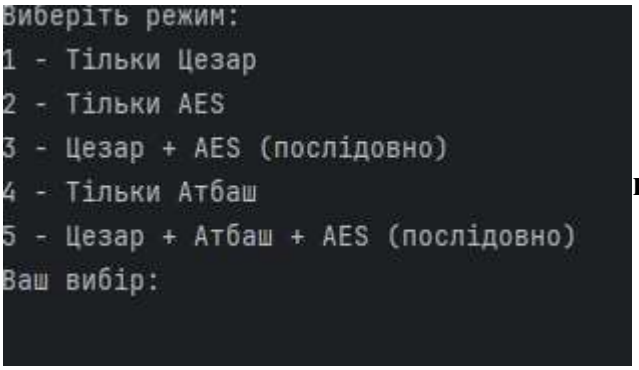
```
            result = aes_cipher.decrypt(text, aes_key)  # Розшифрування
AES
        except Exception as e:
            print("Помилка при розшифруванні AES:", e)
            return

    # Обробка некоректного режиму
    else:
        print("Невірний режим.")
        return

    # Виведення остаточного результату
    print("Результат:", result)

# Запуск програми, якщо скрипт виконується напряму
if __name__ == "__main__":
    main()
```

Перевірка програми:



Консольне меню після запуску програми.

Цезаря:

```
Ваш вибір: 1
Виберіть дію:
1 - Шифрувати
2 - Розшифрувати
Ваш вибір (1/2): 1
Введіть текст: АБВГД
Введіть зсув для Цезаря: 2
Результат: ВГДЕЖ

Process finished with exit code 0
```

```
5 - Цезар + Атбаш + AES (послідовно)
Ваш вибір: 1
Виберіть дію:
1 - Шифрувати
2 - Розшифрувати
Ваш вибір (1/2): 2
Введіть текст: ВГДЕЖ
Введіть зсув для Цезаря: 2
Результат: АБВГД

Process finished with exit code 0
```

Атабаш:

```
4 - Тільки Атбаш
5 - Цезар + Атбаш + AES (послідовно)
Ваш вибір: 4
Виберіть дію:
1 - Шифрувати
2 - Розшифрувати
Ваш вибір (1/2): 1
Введіть текст: АБВ
Результат: ЯЮЬ

Process finished with exit code 0
```

```
4 - Тільки Атбаш
5 - Цезар + Атбаш + AES (послідовно)
Ваш вибір: 4
Виберіть дію:
1 - Шифрувати
2 - Розшифрувати
Ваш вибір (1/2): 2
Введіть текст: ЯЮЬ
Результат: АБВ

Process finished with exit code 0
```

AES:

```
Ваш вибір: 2
Виберіть дію:
1 - Шифрувати
2 - Розшифрувати
Ваш вибір (1/2): 1
Введіть текст: Гваделупа больше не спит в ящике
Введіть ключ для AES: 242229
Підготовлений ключ (hex): 42d886989d08eb0cebe53064144aa59a1021a101a5f4884be61400ddd00a7de5
Результат: sQfTA14WXivtN0i1IqIYDQ==:9NEnQVETAMxAtd7lvhb0OXwS5fS3p2e6I1npYYlj9XAqVRR2+5Wk62XHa0X2nPLBVCEjV+5anI0GoNY=

Process finished with exit code 0
```

Програма пропонує ввести текст, та ввести число, яке шляхом хешування переводить пару цифр, які були вибрані випадковим чином в ключ з набору 16річних символів місткістю 256 бітів.

Розшифрування:

Вводимо результат та ключ:

Цезарь+Атабаш+AES:

ВИСНОВКИ

Ми провели аналіз історичних алгоритмі шифрування. Ми пройшлись по популярним шифрам темних віків, середньовіччя та епохи відродження. Розібрали знакові у історії великі машини шифрування часів нацистської Німеччини. Такі як Енігма і Колос. Провели невеличкий аналіз типів шифрування, та реалізували алгоритм шифрування DES. Зробили програмну реалізацію деяких алгоритмів в одні програмі.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. "The Code Book: by Simon Singh»

2. "Secret History: The Story of Cryptology" by Craig P. Bauer

3. Cryptanalysis — *Helen Fouché Gaines*

4. Historical Cryptology: A Primer — *Benedek Láng*

5. Cryptology Classical and Modern By*Richard E. Klima*, *Richard Klima*, *Neil P. Sigmon*, *Neil Sigmon*

6. Cryptography: A Comparative Analysis for Modern Techniques Faiqa Maqsood1 Dept. Computer Science & Information Technology Superior University Lahore, Pakistan Muhammad Ahmed 2 Dept. Computer Science & Information Technology Superior University Lahore, Pakistan Muhammad Mumtaz Ali3 Dept. Computer Science & Information Technology Superior University Lahore, Pakistan Munam Ali Shah4 Dept. Computer Science COMSATS Institute of Information Technology Islamabad, Pakistan