МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: 123 «Комп'ютерна інженерія»

Освітня програма: «Комп'ютерна інженерія»

Група: 2БКС-28

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

здобувача освіти денної форми навчання

БКС.28.24.000.КРБ

Ситнікова Андрія Сергійовича

м. Одеса
2024 р.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Спеціальність: **123 «Комп'ютерна інженерія»**

Освітньо-професійна програма: **«Комп'ютерна інженерія»**

Група: **2БКС-28**

ПОЯСНЮВАЛЬНА ЗАПИСКА

До кваліфікаційної роботи бакалавра на тему: **«Дослідження системи потокового шифрування даних на основі мови програмування Python»**

Проектний матеріал складається з пояснювальної записки на 93 сторінках та графічного (презентаційного) матеріалу на 20 аркушах (слайдах)

Виконавець _____ (Ситніков А.С.)

Керівник проекту _____ (Кільдішев В.Й.)

Консультанти:

з розділу охорони праці та техніки безпеки _____ (Чорновол Н.І.)

з нормоконтролю _____ (Петрашова В.І.)

старший консультант _____ (Кривченко Ю.В.)

До захисту допущений

Завідувач кафедри _____ (Іванова Л.В.)

Завідувач відділення _____ (Скорнякова О.В.)

Захист «27» 06 2024 р. Протокол ЕК № 3

Оцінка ЕК 4 (добре) 85

Секретар ЕК _____

АНОТАЦІЯ

У випускній кваліфікаційній роботі на тему “Дослідження системи потокового шифрування даних на основі мови програмування Python” було розглянуто архітектуру та фундаментальні принципи побудови систем потокового шифрування. Проведено аналіз методології та алгоритмів потокового шифрування, реалізованих на Python.

Основна мета дослідження — аналіз ефективності та надійності реалізації алгоритмів потокового шифрування на Python, а також порівняння їх з блоковими шифрами. Для об'єктивного аналізу було проведено тестування розробленого програмного забезпечення, результати якого показали ефективність та надійність потокового шифрування на основі Python. Було розроблено програмне забезпечення для потокового шифрування даних, проведено його тестування, та зроблено висновки щодо його застосовності в реальних умовах.

У даній роботі детально розглянуто алгоритми потокового шифрування, такі як RC4 та ChaCha20, з акцентом на їхню реалізацію за допомогою мови програмування Python. Було проведено порівняння цих алгоритмів з блоковими шифрами, такими як AES, з точки зору швидкості, використання пам'яті та стійкості до криптоаналітичних атак.

В рамках дослідження було створено кілька програмних модулів, що реалізують різні алгоритми потокового шифрування. Кожен модуль було протестовано на різних наборах даних для оцінки їхньої продуктивності та надійності. Також було враховано зручність використання бібліотек Python для криптографічних операцій, таких як PyCryptodome. Та створено декілька програм ля демонстрації.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

Відділення *Комп'ютерних систем* Кафедра *Комп'ютерної інженерії*

Спеціальність *123 «Комп'ютерна інженерія»*

Освітньо-професійна програма *«Комп'ютерна інженерія»*

ЗАТВЕРДЖУЮ:

Заст. дир. з НВР *Беркань І.В.*

"16" 01 2024 р.

ЗАВДАННЯ

на кваліфікаційну роботу бакалавра

здобувачеві освіти *Ситнікову Андрію Сергійовичу*

(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи *Дослідження системи потокового шифрування даних на основі мови програмування Python*

затверджена наказом по коледжу від "1" 11 2023 р. № 244-АД

2. Термін здачі студентом кваліфікаційної роботи 10.06.24

3. Вихідні дані до роботи

1 Аналіз методологій та алгоритмів потокового шифрування, реалізованих за допомогою Python 2 Алгоритми та методи потокового шифрування. 3 Ефективність та надійність реалізації алгоритмів потокового шифрування на Python. 4 Порівняння потокових шифрів з блоковими шифрами. 5 Обґрунтування вибору Python та інших інструментів для реалізації проекту.

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)

1 Аналітичний огляд методів потокового шифрування;

2 Опис використаних технологій і методів;

3 Розробка програмного забезпечення;

4 Тестування і результати;

5. Перелік графічного матеріалу (слайдів мультимедійної презентації)

1. *Порівняльний аналіз існуючих рішень.*
2. *Технології геоінформаційних систем та мобільних додатків.*
3. *Архітектура мобільного додатку.*
4. *Макет мобільного додатку*
5. *Скріншоти /відео роботи мобільного додатку*

6. Консультанти по кваліфікаційній роботі, із зазначенням розділів, що їх стосуються

Розділ	Консультант	ПІДПИС	
		Завдання видав	Завдання прийняв
Основний розділ	*Кільдішев В.І.*		
Розділ охорони праці	*Чорновол Н.І.*		
Нормоконтроль	*Петрашова В.І.*		
Старший консультант	*Кривченко Ю.В.*		

7. Дата видачі завдання _____

Керівник роботи *Кільдішев В.І.* _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

Пор. №	Назва етапів кваліфікаційної роботи	Термін виконання етапів роботи	Примітка
1.	*Огляд літератури.*	4.05.2024	
2.	*Вступ.*	8.05.2024	
3.	*Аналітичний огляд методів потокового.*	15.05.2024	
4.	*Опис використаних технологій і методів.*	16.05.2024	
5.	*Розробка програмного забезпечення.*	17.05.2024	
6.	*Тестування і результати.*	19.05.2024	
7.	*Остаточна перевірка тексту дипломної роботи на помилки, форматування документу згідно вимог.*	20.05.2024	
8.	*Підготовка до друку, перевірка всіх додатків та супровідної документації.*	31.05.2024	
9.	*Друк дипломної роботи, підготовка та зшивання копій.*	6.06.2024	
10.	*Отримання відгуків та коментарів від рецензента, аналіз зауважень.*	7.06.2024	
11.	*Висновки.*	10.06.2024	
12.	*Отримання рецензії.*	13.06.2024	

Здобувач освіти _____
(підпис)

Керівник роботи _____
(підпис)

Формат	Зона	Поз.	Позначення	Назва	Кіл.	Примітка
				Документація		
			БКС 28. 00 000. 00 КРБ	Випускна робота		
А4			БКС 28. 00 000. 00 КРБ ПЗ	Пояснювальна записка	1	
			БКС 28. 00 000. 00 КРБ МП	Мультимедійна		
				презентація		

Зм.	Арк.	№ докум.	Підпис	Дата	БКС 28. 24 000. 00 КРБ		
Розробив		Ситніков А.С.		10.05	Дослідження системи потокового шифрування даних на основі мови програмування Python	Літ.	Аркуш 4 / Аркушів 95
Перевірив		Кільдішев В.Й.		10.05		Н Д П	
				10.05		ВСП "ОТФК ОНТУ" гр.2БКС-28	
Н. Контр.		Петрашова В.І.		10.05			
Затверд.		Іванова Л.В.		10.05			

ЗМІСТ

Вступ 8
1 Основний розділ 9
1.1Огляд методів потокового шифрування 9
1.1 Розвитку потокового шифрування 9
1.1.2 Сучасні методології та алгоритми потокового шифрування 9
1.1.3 Теоретичні аспекти потокового шифрування 9
1.1.4 Класифікація потокових шифрів 9
1.1.5. Генератори псевдовипадкових чисел для потокових шифрів.. 10
1.1.6.Синхронні потокові шифри…. 11
1.1.7. Класифікація потокових шифрів…….. 11
1.2. Порівняння потокових і блочних шифрів 12
1.2.1 Принципи роботи блочних шифрів 12
1.2.2 Порівняння ефективності та надійності 12
1.2.3 Надійність 13
1.2.4 Приклади використання в реальних системах 13
1.3. Види атак… 14
1.3.1 Класифікація атак 14
1.4 Детальний розгляд шифрів RC4, операція XOR та Chacha20. 15
1.4.1Алгоритм RC4 15
1.4.2Алгоритм ChaCha20 17
1.4.3Потоковий шифр з використанням операції XOR 20
1.5 Порівняльний аналіз методів потокового шифрування 21
1.6 Простота реалізації 22
1.6.1 Атаки на основі аналізу алгоритму 23
1.7 Обґрунтування вибору Python 25
1.7.1 Аналіз Python…… 26
1.7.2 Переваги Python для реалізації потокових шифрів 26

1.7.3 Аналіз C++…… 28
1.7.4 Аналіз Java…… 31
1.8 Реалізація дослідження потокового шифрування на мові Python 34
1.8.1Вибір середовища розробки PyCharm 34
1.8.2 Порівняння з іншими IDE 34
1.8.3 Функціональні можливості PyCharm 35
1.8.4 Користувальницький інтерфейс 35
1.9 Реалізація проекту потокового шифрування 36
1.9.1 Опис алгоритму потокового шифрування 36
1.9.2 Створення проекту в PyCharm 36
1.10 Стоворення програм 38
1.10.1 Простий поточний шифр XOR 38
1.10.2 Шифр RC4 40
1.11 Демонстрація роботи програми 43
1.11.1 Простий потичнийний шифр XOR 43
1.11.2 Потичнийний шифр RC4 46
2. Розділ охрони праці та техніки безпеки 47
2.1 Вимоги до робочого місця програміста 47
2.1.1 Ергономіка 47
2.1.2 Освітлення 48
2.1.3 Мікроклімат 49
2.2 Профілактика професійних захворювань 50
2.3 Безпека роботи з комп'ютерним обладнанням 51
Висновки……. 52
Перелік використаних інформаційних джерел 53
Додаток А Програми шифрування 54
Додаток Б Слайди мультимедійної презентації 59

ВСТУП

У сучасному світі велика кількість даних, що зберігаються та передаються електронними засобами, постійно зростає. Це пов'язано з тим, що інформаційні технології широко використовуються у різних сферах діяльності: від бізнесу та фінансів до науки та освіти. Ключовим елементом цього процесу є забезпечення захисту інформації від несанкціонованого доступу та зловмисних діяльності. Одним з найбільш ефективних способів захисту даних є використання криптографії, зокрема потокового шифрування. Потокове шифрування дозволяє забезпечити високий рівень безпеки даних завдяки шифруванню інформації по байтах та бітам що робить його важливим для захисту потоків даних у реальному часі. В умовах стрімкого розвитку технологій, зокрема Інтернету речей (IoT) та мобільних комунікацій, потреба у використанні ефективних криптографічних рішень зростає, таких як потокові шифри, стає все більш актуальною.

Метою кваліфікаційної роботи є дослідження системи потокового шифрування даних на основі мови програмування Python. У рамках цього дослідження буде проведено аналіз методів потокового шифрування, розглянуто основні принципи їхньої роботи, а також проведено порівняння різних алгоритмів. Окрему увагу буде приділено реалізації та застосуванню потокових шифрів на мові Python.

Актуальність теми є необхідністю забезпечення рівня безпеки даних у сучасних інформаційних системах, а також простих та ефективних інструментах для реалізації криптографічних алгоритмів. А використання Python для дослідження та розробки потокових шифрів дозволить забезпечити зручність роботи, що робить його ідеальним вибором для даного дослідження. Таким чином, проведене дослідження не тільки для поглибленню знань у галузі криптографії, але й допоможе розробникам створювати більш безпечні та надійні системи для захисту інформації, що є надзвичайно важливим у сучасному інформаційному суспільстві.

1 ОСНОВНИЙ РОЗДІЛ

1.1 Огляд методів потокового шифрування

1.1.1 Розвитку потокового шифрування.

Початки потокового шифрування розпочинаються в середині 20-го століття, коли вчені та інженери почали шукати методи забезпечення безпеки комунікацій. Одним із перших широко відомих потокових шифрів був Вернама, винайдений у 1917 році. Згодом з'явилися більш складні алгоритми, такі як One-time pad, що став одним із найбезпечніших, але найважчих у застосуванні через вимогу унікальних ключів для кожного повідомлення. Наступні дослідження призвели до створення шифрів, таких як RC4, що широко використовується в багатьох системах, наприкад як HTTPS-протоколів. Незважаючи на свою популярність, RC4 має відомі вразливості, що спричинило розробку нових алгоритмів, таких як ChaCha, який на сьогоднішній день вважається одним із найбезпечніших потокових шифрів.

1.1.2 Сучасні методології та алгоритми потокового шифрування

Потокове шифрування є важливою частиною криптографії, яка забезпечує захист даних шляхом шифрування потоку даних по одному біту або байту за раз. Основні сучасні алгоритми потокового шифрування включають RC4, ChaCha20 та інші.Далі розглянемо деякі з них докладніше.

1.1.3 Теоретичні аспекти потокового шифрування

1.1.4 Основні принципи

Потокове шифрування є однією з двох основних категорій симетричного шифрування, поряд із блочним шифруванням. Основний принцип потокового шифрування полягає у генерації псевдовипадкової послідовності ключів (key stream), яка потім комбінується з відкритим текстом (plaintext) для створення зашифрованого тексту (ciphertext).

Потокове шифрування зазвичай реалізується за допомогою двох основних компонентів:

1 Генератор ключового потоку (Key Stream Generator): Відповідає за створення псевдовипадкової послідовності, яка повинна бути криптографічно стійкою.

2 Механізм комбінування (Combining Mechanism): Застосовує ключовий потік до відкритого тексту, зазвичай використовуючи операцію XOR.

1.1.5 Генератори псевдовипадкових чисел для потокових шифрів

Генератори псевдовипадкових чисел є критично важливими компонентами для сучасних криптографічних систем, включаючи потокові шифри. Ці генератори використовуються для створення ключових послідовностей, які потім застосовуються для шифрування даних. У цьому розділі ми розглянемо архітектуру та принципи роботи генераторів, а також наведемо приклади популярних алгоритмів, таких як RC4 та ChaCha.

Основні концепції генераторів псевдовипадкових чисел використовуються для створення послідовностей чисел, які мають властивості випадковості. Хоча вони не є абсолютно випадковими, їх досить складно передбачити, що робить їх корисними для криптографії. Зазвичай складаються з трьох основних компонентів:

1 Стан: внутрішнє значення, яке змінюється з кожною новою генерацією числа.

2 Функція зворотного зв'язку: визначає, як стан оновлюється на основі попереднього стану.

3 Вихідна функція: генерує псевдовипадкове число на основі поточного стану.

Архітектура Генератори псевдовипадкових чисел

На рисунку нижче показана типова схема роботи у потоковому шифрі. Вхідний ключ ініціалізує стан, який потім оновлюється функцією зворотного зв'язку. Вихідна функція генерує псевдовипадкові числа (keystream), які використовуються для шифрування відкритого тексту шляхом операції XOR.

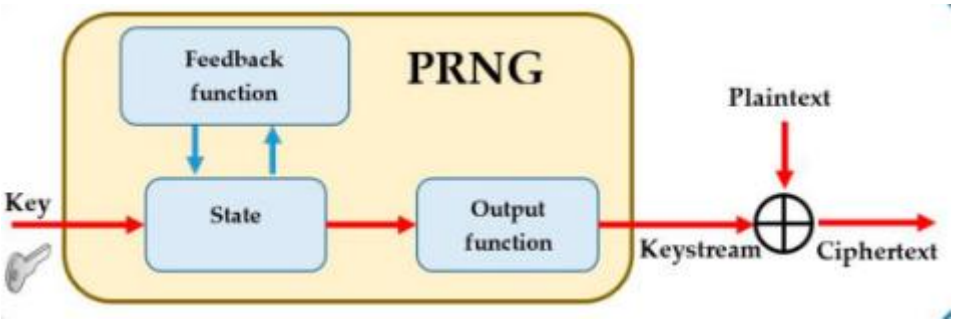


Рисунок 1.1. Генератори псевдовипадкових чисел для потокових шифрів

Ця схема є універсальною і може бути застосована для різних алгоритмів потокових шифрів, таких як RC4 та ChaCha20, забезпечуючи високу швидкість шифрування і безпеку даних.

1.1.6 Класифікація потокових шифрів

Стокові шифри можна класифікувати на дві категорії залежно від різних аспектів їх роботи та застосування.

1.1.7 Синхронні потокові шифри

У синхронних потокових шифрах ключовий потік генерується незалежно від відкритого чи зашифрованого тексту. Генерація ключового потоку залежить тільки від секретного ключа та вихідного вектора. Таким чином, якщо відомі секретний ключ і вихідний вектор, ключовий потік можна відтворити.

Переваги синхронних потокових шифрів:

• Простота у реалізації.

• Висока швидкість шифрування та дешифрування.

Недоліки:

• Якщо при передачі даних відбувається втрата або зміна одного біта, це може призвести до розсинхронізації та втрати можливості розшифрування всього наступного тексту.

Асинхронні потокові шифри або шифри зі зворотним зв'язком

Асинхронні шифри використовують попередні біти зашифрованого тексту для

створення ключового потоку. Це забезпечує певну стійкість до втрат даних, оскільки ключовий потік можна відновити після певного інтервалу.

1.2. Порівняння потокових і блочних шифрів

1.2.1 Принципи роботи блочних шифрів

Блокові шифри є другою основною категорією симетричних шифрів. Вони працюють із фіксованими блоками даних, зазвичай 64 або 128 біт, і шифрують кожен блок окремо. Принцип роботи блокових шифрів полягає у застосуванні однієї чи більше ітерацій перетворень (раундів) до кожного блоку даних, використовуючи секретний ключ.

Основні компоненти блочних шифрів

1. Розбивка на блоки

Вхідні дані розбиваються на блоки фіксованого розміру.

2. Перетворення (раунди)

Кожен блок проходить через серію раундів перетворень, включаючи заміни, перестановки, та XOR з підключами, що генеруються з основного ключа.

3. Режими шифрування

Для шифрування послідовності блоків використовуються різні режими шифрування, такі як ECB, CBC, CFB, OFB та інші.

Основні алгоритми блочних шифрів

• DES (Data Encryption Standard): Старий стандарт, який використовує 64-бітові блоки та 56-бітний ключ. Вважається застарілим через низьку криптографічну стійкість.

• AES (Advanced Encryption Standard): Сучасний стандарт, який використовує 128-бітові блоки та ключі розміром 128, 192 або 256 біт. AES дуже безпечний та широко використовується.

• Blowfish: Алгоритм з довжиною блоку 64 біта і довжиною ключа, що змінюється, від 32 до 448 біт. Використовується у багатьох сучасних додатках.

1.2.2 Порівняння ефективності та надійності

1 Ефективність

Швидкість шифрування та дешифрування:

Стокові шифри: Зазвичай швидше, оскільки працюють із окремими бітами чи байтами. Ідеально підходять для програм, де важлива висока швидкість передачі даних, таких як мережеві комунікації.

Блокові шифри: Більш повільні через обробку фіксованих блоків даних і складних перетворень. Однак, оптимізовані реалізації, такі як AES-NI, забезпечують високу продуктивність навіть для блокових шифрів.

2. Вимоги до пам'яті:

Потікові шифри: Використовують менше пам'яті, оскільки обробляють дані по одному біту або байту.

Блокові шифри: Вимагають більше пам'яті для зберігання блоків даних та проміжних результатів раундів.

1.2.3 Надійність

Стійкість до атак

Потокові шифри: Вразливі до атак, якщо використовується той самий ключовий потік більше одного разу. Атаки на відомому відкритому тексті можуть скомпрометувати ключовий потік.

Блочні шифри: Стійкіші до різних типів атак завдяки складним перетворенням і використанню режимів шифрування, які забезпечують додатковий рівень безпеки.

Забезпечення цілісності даних

Потокові шифри: Не мають вбудованих механізмів забезпечення цілісності даних, що потребує використання додаткових методів автентифікації.

Блочні шифри: Можуть використовувати такі режими, як GCM (Galois/Counter Mode), які забезпечують як конфіденційність, так і цілісність даних.

1.2.4 Приклади використання в реальних системах

Інтернет-комунікації

1. TLS (Transport Layer Security)

Потокові шифри: RC4 був одним із перших потокових шифрів, що використовуються в TLS. Однак через виявлені вразливості його замінили безпечнішими алгоритмами, такими як ChaCha20.

Блокові шифри: AES є основним блоковим шифром, який використовується в сучасних версіях TLS. Він забезпечує високий рівень безпеки та ефективності.

2. VPN (Virtual Private Network):

Потокові шифри: У деяких VPN-реалізаціях використовуються потокові шифри для забезпечення високошвидкісного шифрування.

Блокові шифри: AES є стандартом більшості VPN-рішень, забезпечуючи надійний захист даних.

Мобільні додатки та пристрої

1. Мобільні додатки:

Потокові шифри: ChaCha20 використовується в багатьох мобільних додатках завдяки високій швидкості та низькому споживанню ресурсів.

Блокові шифри: AES також широко використовуються в мобільних програмах, особливо в тих, де важлива висока безпека.

2. Вбудовані системи та IoT:

Поточні шифри: Потікові шифри, такі як Grain та Trivium, використовуються у вбудованих системах та IoT-пристроях через їх високу швидкість та низькі вимоги до пам'яті.

Блокові шифри: У деяких IoT-рішеннях використовують легкі блокові шифри, такі як PRESENT, для забезпечення безпеки даних.

1.3. Види атак

Криптографічні системи, розроблені для забезпечення конфіденційності, цілісності та справжності даних, можуть бути вразливими для різних видів атак. У цьому розділі розглядаються основні типи атак на криптографічні алгоритми та системи, їх методи та наслідки.

1.3.1 Класифікація атак

Атаки на криптографічні системи можна класифікувати за різними ознаками, включаючи методи здійснення, цілі та вразливості, які вони експлуатують.

За методами здійснення Атаки на основі тексту є атаки на основі відкритого тексту (Known-Plaintext Attacks, KPA): Нападник має доступ до деяких пар відкритий текст – шифротекст і намагається відновити ключ або алгоритм шифрування. Атаки на основі вибраного відкритого тексту: Нападник може вибрати відкритий текст та отримати відповідний шифротекст, що дозволяє здійснити аналіз алгоритму. Атак на основі вибраного шифротексту (CHOsen-Ciphertext Attacks, CCA): Нападник може вибрати шифротекст і отримати відповідний відкритий текст, який допомагає аналізувати шифр. Атаки на основі адаптивного вибраного шифротексту (ACCA): Нападник може взаємодіяти з декриптором та адаптивно вибирати шифротексти на основі отриманих результатів.

1.4 Детальний розгляд шифрів RC4, операція XOR та Chacha20.

1.4.1Алгоритм RC4

У 1987 році Рон Рівест розробив RC4 – один з найвідоміших і широко використовуваних потокових шифрів кінця XX століття. RC4 став популярним завдяки своїй простоті, швидкості і відносній безпеці на той час. Алгоритм RC4 використовувався в численних протоколах, включаючи SSL/TLS для захисту інтернет-комунікацій, а також у WEP для забезпечення безпеки бездротових мереж. RC4 працює шляхом генерації псевдовипадкового ключового потоку, який потім через XOR отримають шифр тексту.

Основні етапи алгоритму RC4 включають два основних процеси

1 KSA (Ініціалізація ключового потоку):

- Створюється масив S, що містить числа від 0 до 255.
- Масив S перемішується за допомогою ключа, щоб створити початковий стан масиву S.

Ініціалізація масиву S:

Вхід: ключ K

S[i] = i (i від 0 до 255)

Ініціалізація ключового масиву K:

i = 0, j = 0

Для i від 0 до 255:

j = (j + S[i] + K[i % |K|]) % 256

Поміняти місцями S[i] і S[j]

2 PRGA (Генерація псевдовипадкових чисел):

- Масив S використовується для генерації псевдовипадкових чисел, які будуть використовуватися для шифрування/дешифрування.

Ініціалізація індексів i та j:

$$i = 0, \quad j = 0$$

Генерація псевдовипадкових чисел:

- i = (i + 1) % 256
- j = (j + S[i]) % 256
- Поміняти місцями S[i] і S[j]
- t = (S[i] + S[j]) % 256

keystream = S[t]

3 Шифрування/дешифрування:

- Текст шифрується/дешифрується шляхом XOR кожного байта тексту з наступним байтом псевдовипадкового потоку.

$$\text{ciphertext}[n] = \text{plaintext}[n] \oplus K$$

де K — псевдовипадкове число, згенероване PRGA, $\oplus$ — операція побітового XOR.

RC4 має високу ефективність та простоту реалізації, що зробило його популярним вибором у багатьох системах. Однак, через відомі вразливості, такі як слабкість до атаки на вектор ініціалізації, його використання в сучасних системах зменшується. Докладніше на Рисунок 1.2.

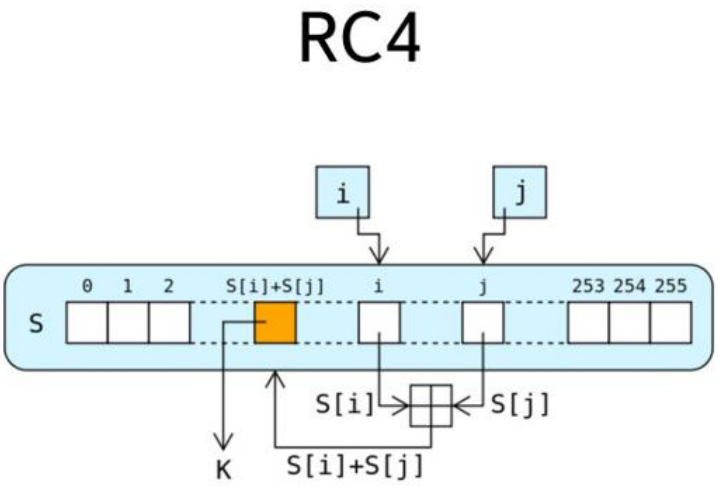


Рисунок 1.2. Схема RC4

1.4.2Алгоритм ChaCha20

Шифр ChaCha20, розроблений Деніелом Бернштейном, є сучасним потоковим шифром, що використовує 20 раундів перестановок для досягнення високої безпеки та продуктивності. Він є надійною альтернативою RC4 і використовується в багатьох сучасних протоколах, таких як TLS.

ChaCha20 використовує 256-розрядний ключ і 64-розрядний і nonce. Він генерує ключовий потік, який потім поєднується з відкритим текстом за допомогою

операції XOR для отримання шифртексту. В основі ChaCha20 лежить функція, звана "quarter round", яка застосовується до 4 32-бітових слів. Алгоритм складається з 20 раундів, де кожен раунд включає кілька таких функцій “quarter round”.

Основні етапи алгоритму ChaCha20 включають:

Основні кроки алгоритму ChaCha20

1 Ініціалізація

2 Генерація ключового потоку

3 Шифрування/дешифрування

1. Ініціалізація

Алгоритм використовує 512-бітний (64-байтовий) становий вектор, який включає:

256-бітний ключ

64-бітний nonce

64-бітний блоків

128 біт фіксованих констант

2. Генерація ключового потоку

Алгоритм ChaCha20 використовує 20 раундів перетворень для генерації ключового потоку. Кожен раунд складається з послідовності додавання, XOR та побітових зсувів.

Основні операції

1 Додавання з модулем 2^{32}:

$a=(a+b) \bmod 2^{32}$

2 Побітове XOR:

$a=a \oplus b$

3 Побітовий зсув:

$a=(a<<n)\vee(a>>(32-n))$

де <<< - це лівий циклічний зсув, а >>> - правий циклічний зсув.

Раунди

Раунди ChaCha20 складаються з чергування "квадратних" перетворень для стовпців і діагоналей.

1 Квадратно-заплутуючі перетворення для стовпців.

2 Квадратно-заплутуючі перетворення для діагоналей.

$a += b; d \oplus= a; d <<<= 16;$

$c += d; b \oplus= c; b <<<= 12;$

$a += b; d \oplus= a; d <<<= 8;$

$c += d; b \oplus= c; b <<<= 7;$

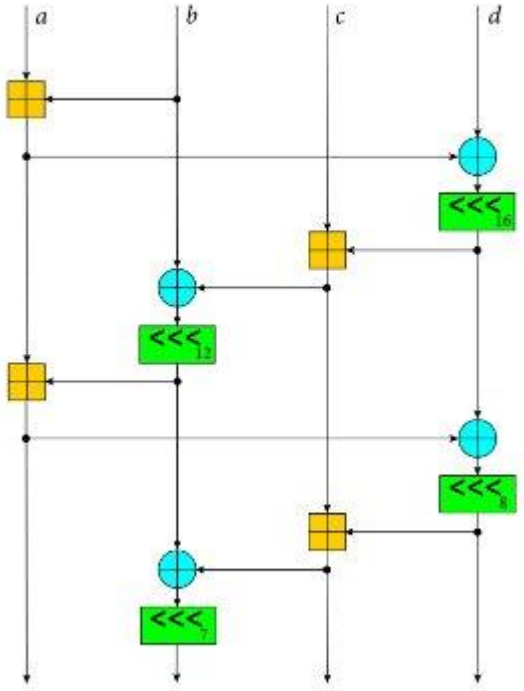


Рисунок 1.3. Діаграма функції чвертьраунду Chacha

ChaCha20 забезпечує високу швидкість та безпеку, роблячи його ідеальним для використання у реальних системах. Він також стійкий до багатьох видів атак, які є вразливими для інших потокових шифрів, як RC4.

1.4.3Потоковий шифр з використанням операції XOR

Потоковий шифр з використанням операції XOR є одним із найпростіших і найпоширеніших методів шифрування. Його основний принцип полягає в тому, що кожен біт відкритого тексту комбінується з відповідним бітом ключового потоку за допомогою побітової операції виключно "АБО" (XOR). У цьому шифрі ключовий потік генерується випадковим чином або з використанням псевдовипадкових генераторів.

Основні кроки алгоритму:

1 Генерація ключового потоку: Використання генератора псевдовипадкових чисел (PRNG) для створення ключового потоку, що має таку довжину, як і відкритий текст.

2 Шифрування: Виконує операцію XOR між кожним бітом відкритого тексту та відповідним бітом ключового потоку.

1 Дешифрування: Виконання XOR між кожним бітом зашифрованого тексту і

2 відповідним бітом ключового потоку для відновлення відкритого тексту.

Формула

Нехай P - відкритий текст, K - ключовий потік, C - зашифрований текст.

Шифрування:

$C_i=P_i \oplus K_i$

де C_i - біт зашифрованого тексту, P_i - біт відкритого тексту, K_i - біт ключового потоку.

Дешифрування:

$P_i = C_i \oplus K_i$

1.5 Порівняльний аналіз методів потокового шифрування

Порівняння різних методів потокового шифрування зазвичай здійснюється за такими критеріями, як ефективність, безпека, простота реалізації та швидкість роботи. Розглянемо основні відмінності між RC4, ChaCha20 та шифром на основі операції XOR.

Ефективність потокового шифрування залежить від швидкості генерації ключового потоку та шифрування даних. RC4 і ChaCha20 мають високу ефективність завдяки оптимізованим алгоритмам та генерації ключів. Як це можно побачити у таблиці

Таблиця 1.1. Порівняння ефективності

Алгоритм	**Швидкість генерації ключового потоку**	**Швидкість шифрування**
RC4	Средня	Висока
ChaCha20	Дуже висока	Дуже висока
XOR	Висока	Дуже висока

Як видно з таблиці , алгоритм ChaCha20 забезпечує найвищу швидкість генерації ключового потоку та шифрування даних, що робить його привабливим вибором для систем, де важлива висока продуктивність.

Безпека

Безпека потокових шифрів визначається складністю алгоритму та стійкістю до атак. RC4 має відомі вразливості, які можуть бути використані для розкриття шифрованого тексту. ChaCha20, навпаки, розроблений з урахуванням сучасних вимог до безпеки та стійкий до багатьох видів атак.

Таблиця 1.2. Порівняння безпеки

Алгоритм	**Вразливості**	**Рівень безпеки**
RC4	Відомі вразливості	Низький
ChaCha20	Немає відомих серйозних вразливостей	Високий
XOR	Вразливий при використанні слабких ключів	Середній

Як видно , ChaCha20 є найбільш безпечним серед розглянутих алгоритмів, що робить його придатним для використання в системах, де важлива висока безпека даних.

1.6 Простота реалізації

Простота реалізації є важливим чинником під час виборів алгоритму шифрування. Простий шифр на основі XOR дуже легко реалізувати, в той час як RC4 і ChaCha20 вимагають більш складної реалізації, але забезпечують більш високий рівень безпеки.

Таблиця 1.3. Порівняння реалізації

Алгоритм	**Складність реалізації**	**Вимоги до ресурсів**
RC4	Средня	Низькі
ChaCha20	Висока	Середня
XOR	Низька	Низькі

Як видно, шифр на основі XOR є найпростішим для реалізації, проте для забезпечення високої безпеки рекомендується використовувати складніші алгоритми, такі як ChaCha20.

Швидкість роботи

Швидкість алгоритмів шифрування залежить від швидкості генерації ключового потоку та шифрування даних. ChaCha20 демонструє високу швидкість

серед даних алгоритмів, що робить його оптимальним для застосування в реальних системах, де важлива висока продуктивність.

Таблиця 1.4. Порівняння швидкості роботи

Алгоритм	**Складність реалізації**
RC4	Висока
ChaCha20	Дуже висока
XOR	Дуже висока

Як видно з таб.3, алгоритм ChaCha20 забезпечує високу швидкість роботи, що робить його привабливим вибором для багатьох сучасних програм. Під час всіх аналізованих методів потокового шифрування можно побачити що вони мають свої переваги та недоліки в залежності від вимог до продуктивності, безпеки та простоти реалізації. RC4, незважаючи на свою простоту та високу швидкість, має серйозні вразливості, що обмежують його використання у сучасних системах. ChaCha20 забезпечує високу безпеку та продуктивність, що робить його оптимальним вибором для багатьох програм. Шифр на основі XOR, хоч і дуже простий у реалізації, вимагає сильних ключів для забезпечення безпеки, що може бути складним завданням.

1.6.1 Атаки на основі аналізу алгоритму

Диференціальний криптоаналіз: Аналізує різниці в шифротекстах, що виникають через певні зміни у відкритих текстах, для виявлення шаблонів.

Лінійний криптоаналіз: Використовує лінійні апроксимації для знаходження кореляцій між відкритими текстами та шифротекстами. Атаки на основі аналізу ключа: Направлені на пошук або відновлення ключа шифрування через аналіз слабкостей у генерації чи зберіганні ключів.

Атак на основі часу Використовують інформацію про час, необхідний для виконання криптографічних операцій. А атаки на основі аналізу електромагнітних

випромінювань використовують електромагнітні випромінювання, що утворюються під час виконання криптографічних операцій. Атак на основі аналізу енергії, що споживається: Використовують зміни в споживанні енергії пристроєм при виконанні криптографічних операцій.

Деталізація основних типів атак

Атаки на основі відкритого тексту (KPA). Цей тип атак здійснюється, коли нападник має доступ до пар "відкритий текст - шифротекст". Відомі тексти дозволяють аналізувати алгоритм шифрування та шукати вразливості чи шаблони, які можуть допомогти відновити ключ чи інші відкриті тексти.

Атаки на основі вибраного відкритого тексту (CPA). У таких атаках нападник може вибирати відкриті тексти та отримувати відповідні шифротексти. Це дозволяє тестувати гіпотези щодо алгоритму шифрування та уразливостей, таких як залежність шифротексту від конкретних вхідних даних.

Атаки на основі вибраного шифротексту (CCA). У цьому випадку нападник може вибирати шифротексти та отримувати відповідні відкриті тексти. Це допомагає в аналізі алгоритму дешифрування та пошуку вразливостей, які можуть дозволити відновити ключ або інші відкриті тексти.

Диференціальний криптоаналіз цей метод криптоаналізу аналізує різниці в шифротекстах, що виникають через певні зміни у відкритих текстах. Аналізуючи ці різниці, можна знайти шаблони та слабкості у методі шифрування, які допомагають повернути ключ.

Лінійний криптоаналіз цей метод використовує лінійні апроксимації для виявлення кореляцій між відкритими текстами та шифротекстами. Лінійний криптоаналіз дозволяє виявляти слабкі місця в алгоритмі та відновлювати ключ за допомогою статистичного аналізу.

Атаки на основі побічних каналів ці атаки використовують фізичні властивості пристроїв, які виконують криптографічні операції. До таких атак відносяться аналіз часу виконання, електромагнітних випромінювань та споживаної енергії. Вони

дозволяють нападникам отримувати додаткову інформацію, що допомагає у розкритті ключа чи алгоритму.

Атаки на основі грубої сили (Brute-Force Attacks) передбачають перебір усіх можливих ключів або паролів доти, доки не буде знайдено правильний. Це один із найпростіших і найнадійніших методів, але він вимагає великих обчислювальних ресурсів та часу.

Приклад грубої сили для симетричного шифрування: Використовуючи потужні обчислювальні засоби, нападник перебирає всі можливі ключі. Для кожного ключа дешифрується шифротекст та перевіряється результат. Процес триває доти, доки не буде знайдений відкритий текст, що має сенс. Наслідки успішних атак на криптографічні системи можуть бути серйозними і включати: Компрометація конфіденційності даних: Розкриття ключа або алгоритму може призвести до доступу до всіх шифрованих даних.

1.7 Обґрунтування вибору Python

Особливості Python для криптографії

Python є однією з найпопулярніших мов програмування, що широко використовується в різних областях, включаючи криптографію. Основні особливості Python, які роблять його привабливим для криптографічних проектів, включають:

Простота та зручність у використанні

Python відомий своєю простотою та читабельністю коду, що дозволяє розробникам швидко створювати та тестувати криптографічні алгоритми. Синтаксис мови легкий вивчення, що знижує поріг входження нових розробників.

Порівняння потокових шифрів на основі Python з іншими мовами програмування

Під час розробки криптографічних систем важливий вибір мови програмування. Кожна мова має свої особливості, що впливають на ефективність, безпеку та зручність реалізації криптографічних алгоритмів. У цьому розділі ми порівняємо

реалізацію потокових шифрів на основі Python з реалізацією іншими мовами програмування, таких як C++ та Java. Розглянемо аспекти продуктивності, безпеки, простоти використання, кросплатформенності та інші важливі характеристики.

1.7.1 Аналіз Python

Python є високорівневою мовою програмування, яка отримала визнання завдяки своїй простоті та зручності. Вона забезпечує широкий спектр інструментів та бібліотек, що полегшує реалізацію складних криптографічних алгоритмів. Переваги та недоліки Python для реалізації потокових шифрів розглянемо докладніше.

1.7.2 Переваги Python для реалізації потокових шифрів:

Python відомий своїм простим і зрозумілим синтаксисом, що робить його доступним широкому колу розробників. Це дозволяє швидко та легко розробляти та підтримувати криптографічні алгоритми. Існує безліч бібліотек криптографії, таких як PyCryptodome, які забезпечують широкий набір функцій для реалізації шифрування та дешифрування даних. Це скорочує час та зусилля, необхідні для реалізації складних алгоритмів.Python працює на багатьох платформах, включаючи Windows, MacOS та Linux, що робить його зручним для розробки багатоплатформних програм. Це важливо для криптографічних систем, які мають бути доступні на різних пристроях.

Велика спільнота розробників:

Велика та активна спільнота розробників Python забезпечує швидкий доступ до допомоги та ресурсів, що може бути корисним при розробці та налагодженні криптографічних алгоритмів.

Порівняння переваг Python для реалізації потокових шифрів можно побачити .

Таблиця 1.5. Порівняння переваг Python

Критерій	**Переваги**
Простота синтаксису	Легкий для читання та написання код
Багатство бібліотек	PyCryptodome, cryptography та інші
Кросплатформеність	Підтримка Windows, macOS, Linux
Спільнота	Велика та активна

Для приладу була створена простота реалізації потокових шифрів на Python

```python
# Функція для шифрування та дешифрування тексту методом XOR
def xor_cipher(text, key):
    # Перетворення тексту та ключа в масиви байтів
    text_bytes = text.encode()
    key_bytes = key.encode()

    # Шифрування/дешифрування кожного байта
    result_bytes = bytearray()
    for i in range(len(text_bytes)):
        # XOR байта тексту з байтом ключа
        result_byte = text_bytes[i] ^ key_bytes[i % len(key_bytes)]
        result_bytes.append(result_byte)

    # Перетворення результату назад у рядок
    result_text = result_bytes.decode()
    return result_text


# Вхідний текст та ключ
input_text = "Hello, World!"
key = "secret"

# Шифрування тексту
encrypted = xor_cipher(input_text, key)
print(f"Зашифрований текст: {encrypted}")

# Дешифрування тексту
decrypted = xor_cipher(encrypted, key)
print(f"Розшифрований текст: {decrypted}")
```

Рисунок 1.4. Приклад коду на мові Python

Цей код використовує функцію xor_cipher, яка приймає рядок тексту та ключ,

перетворює їх на байти, а потім виконує операцію XOR на кожному біті тексту з відповідним байтом ключа

Недоліки Python:

Швидкість виконання Python є інтерпретованою мовою, що може призводити до меншої швидкості виконання в порівнянні з компілюваними мовами, такими як C++. Це може бути критично важливим для криптографічних додатків, де потрібна висока продуктивність. Python може вимагати більше пам'яті та інших ресурсів ніж низькорівневі мови. Це може бути проблемою для вбудованих систем або пристроїв з обмеженими ресурсами.

1.7.3 Аналіз C++

C++ є компільованою мовою програмування, яка забезпечує високу продуктивність і контроль над апаратними ресурсами. Вона широко використовується для розробки систем, де важлива висока швидкість виконання та ефективність використання ресурсів. Переваги та недоліки C++ для реалізації потокових шифрів розглянемо детальніше.

Переваги C++ для реалізації потокових шифрів:

Висока швидкість виконання**:** C++ є компільованою мовою, що дозволяє досягти високої швидкості виконання за рахунок оптимізації на рівні машинного коду. Це особливо важливо для криптографічних алгоритмів, де необхідна висока продуктивність. C++ надає розробникам детальний контроль над використанням пам'яті та інших ресурсів, що дозволяє оптимізувати програми для досягнення максимальної ефективності. Існує багато бібліотек, таких як OpenSSL, які забезпечують підтримку різноманітних криптографічних алгоритмів та протоколів. Це полегшує розробку складних систем шифрування. C++ забезпечує гнучкість у виборі підходів до реалізації алгоритмів та оптимізацій, що дозволяє розробникам налаштовувати програми під конкретні вимоги.

Переваги C++ для реалізації потокових шифрів у таб.5

Таблиця 1.6. Порівняння переваг C++

Критерій	**Переваги**
Швидкість виконання	Висока
Контроль над ресурсами	Детальний контроль
Багатий набір бібліотек	OpenSSL та інші
Гнучкість	Вибір підходів до реалізації

Недоліки C++:

Складний синтаксис C++ може ускладнити розробку та підтримку програм. Це може вимагати більшого часу для вивчення та налагодження коду.

Управління пам'яттю має необхідність ручного керування пам'яттю може призвести до помилок, таких як витік пам'яті або порушення доступу, що ускладнює розробку та тестування програм.

У таб.6 Витрати C++ реалізації потокових шифрів.

Таблиця 1.7. Порівняння реалізації C++

Критерій	**Недоліки**
Складність синтаксису	Висока складність
Управління пам'яттю	Необхідність ручного управління

Приклад коду на C++

```
#include <iostream>

#include <string>

// Функція для шифрування та дешифрування тексту методом XOR

std::string xor_cipher(const std::string &text, const std::string &key) {
```

```
    std::string result;
    for (size_t i = 0; i < text.length(); ++i) {
        // XOR символу тексту з символом ключа
        result += text[i] ^ key[i % key.length()];
    }
    return result;
}
int main() {
  // Вхідний текст та ключ
    std::string input_text = "Hello, World!";
    std::string key = "secret";
    // Шифрування тексту
    std::string encrypted = xor_cipher(input_text, key);
 std::cout << "Зашифрований текст: ";
    for (char c : encrypted) {
        std::cout << std::hex << (int)c; // Виведення в шістнадцятковому форматі
    }
    std::cout << std::endl;
    // Дешифрування тексту
    std::string decrypted = xor_cipher(encrypted, key);
    std::cout << "Розшифрований текст: " << decrypted << std::endl;
    return 0;
}
```

1.7.4 Аналіз Java

Java є високорівневою мовою програмування, відомою своєю платформонезалежністю завдяки використанню віртуальної машини Java (JVM). Вона широко використовується для розробки корпоративних додатків та систем, де важлива надійність та захищеність. Переваги та недоліки Java для реалізації потокових шифрів розглянемо докладніше.

Переваги Java для реалізації потокових шифрів

Java дозволяє розробляти програми, які можуть виконуватися на будь-якій платформі, що підтримує JVM. Це забезпечує високу доступність та гнучкість при розгортанні криптографічних систем. Java використовує збирач сміття для автоматичного керування пам'яттю, що знижує ризик помилок, пов'язаних з керуванням пам'яттю, таких як витоку пам'яті. Існує безліч бібліотек для криптографії, таких як BouncyCastle, що забезпечують підтримку широкого спектру алгоритмів і протоколів. Java забезпечує високий рівень захисту за рахунок вбудованих механізмів безпеки, таких як керування доступом до пам'яті та багаторівневе шифрування.

Таблиця 1.8. Порівняння переваг Java

Критерій	**Переваги**
Кросплатформеність	Висока
Автоматичне управління пам'яттю	Знижений ризик помилок
Багатий набір бібліотек	BouncyCastle та інші

Приклад коду на java.

```
public class XorEncryption {

    // Функція для шифрування та дешифрування тексту методом XOR

    public static String xorCipher(String text, String key) {
```

```
        StringBuilder result = new StringBuilder();
        for (int i = 0; i < text.length(); i++) {
            // XOR символу тексту з символом ключа
            char encryptedChar = (char) (text.charAt(i) ^ key.charAt(i % key.length()));
            result.append(encryptedChar);
        }

 return result.toString();
    }
public static void main(String[] args) {
 // Вхідний текст та ключ
    String inputText = "Hello, World!";
 String key = "secret";
        // Шифрування тексту
        String encrypted = xorCipher(inputText, key);
        System.out.println("Зашифрований текст: " + encrypted);
        // Дешифрування тексту
        String decrypted = xorCipher(encrypted, key);
        System.out.println("Розшифрований текст: " + decrypted);
    }
}
```

Цей код використовує функцію xorCipher, яка приймає рядок тексту та ключ, а потім виконує операцію XOR на кожному символі тексту з відповідним символом

ключа.

Недоліки Java:

1. Швидкість виконання
    - Java може бути повільнішою за C++ через виконання на віртуальній машині JVM. Це може впливати на продуктивність криптографічних додатків.
2. Використання ресурсів
    - Високі вимоги до пам'яті та ресурсів через JVM можуть бути проблемою для деяких систем, особливо для вбудованих пристроїв
    - або систем з обмеженими ресурсами.

Таблиця 1.9. Порівняння недоліків Java

Критерій	**Недоліки**
Швидкість виконання	Повільніша в порівнянні з C++
Використання ресурсів	Високі вимоги до пам'яті та ресурсів

Порівняння реалізації потокових шифрів на основі Python з реалізацією іншими мовами програмування, таких як C++ і Java, показує, що кожна мова має свої переваги і недоліки. Python забезпечує простоту використання та багатий набір бібліотек, але може бути менш продуктивним через інтерпретацію коду. C++ забезпечує високу швидкість виконання та контроль над ресурсами, проте має більш складний синтаксис і потребує ручного керування пам'яттю. Java забезпечує кроссплатформенність та автоматичне управління пам'яттю, але може бути повільнішим і мати високі вимоги до ресурсів через JVM.

Вибір мови програмування для реалізації потокових шифрів залежить від конкретних вимог проекту, таких як продуктивність, зручність розробки, потреба у кросплатформенності та обмеження ресурсів. У кожному разі необхідно

враховувати переваги та недоліки кожної мови, щоб забезпечити ефективну реалізацію криптографічних алгоритмів.

1.8 Реалізація дослідження потокового шифрування на мові Python

У цьому розділі буде розглянуто процес реалізації дослідження потокового шифрування мовою Python, а також обґрунтовано вибір середовища розробки PyCharm. Буде детально описано основні функціональні можливості PyCharm, його переваги перед іншими IDE (інтегрованими середовищами розробки), а також буде показано, як використовувати середовище для розробки програмного забезпечення. Додатково будуть вставлені скріншоти для візуального представлення процесу роботи PyCharm.

1.8.1 Вибір середовища розробки PyCharm

PyCharm – це потужне інтегроване середовище розробки (IDE), спеціально розроблене для мови програмування Python. Воно пропонує широкий спектр інструментів та функцій, які значно спрощують процес розробки програмного забезпечення. Серед основних причин вибору PyCharm можна виділити: Інтеграція із системами контролю версій PyCharm має вбудовану підтримку Git, SVN та інших систем контролю версій, що дозволяє легко керувати версіями коду. Розширені можливості налагодження надає потужні налагоджувальні інструменти, які допомагають виявляти та виправляти помилки в коді. PyCharm пропонує функцію інтелектуального автодоповнення, що значно прискорює процес написання коду. Аналіз коду в реальному часі Це дозволяє виявляти потенційні помилки та проблеми ще на етапі написання коду. PyCharm підтримує численні фреймворки, такі як Django, Flask, Pyramid та інші, що робить його універсальним інструментом розробки веб-додатків. PyCharm дозволяє зручно працювати з різними базами даних безпосередньо із середовища розробки.

1.8.2 Порівняння з іншими IDE

На ринку є кілька популярних IDE для Python, таких як Visual Studio Code, Jupyter Notebook, Spyder та інші. Кожен з них має свої переваги та недоліки, проте PyCharm виділяється завдяки своїм розширеним функціональним можливостям.

Visual Studio Code: Це легке та швидке середовище розробки, яке підтримує безліч мов програмування через плагін. Однак воно вимагає додаткового налаштування і не має всіх вбудованих інструментів PyCharm.

Jupyter Notebook: Ідеально підходить для наукових досліджень та аналізу даних завдяки можливості виконувати код в окремих блоках. Однак для розробки великих програмних проектів воно менш зручне, ніж PyCharm.

Spyder: Призначений для наукових обчислень та має вбудовану підтримку бібліотек, таких як NumPy, SciPy, Matplotlib. Але, знову ж таки, для розробки великих програмних проектів PyCharm надає більше можливостей.

1.8.3Функціональні можливості PyCharm

Користувальницький інтерфейс

Інтерфейс користувача PyCharm інтуїтивно зрозумілий і добре структурований, що дозволяє швидко орієнтуватися в середовищі. Основні компоненти інтерфейсу включають:

Проектний огляд (Project view): Дозволяє переглядати структуру проекту та швидко переходити між файлами.

Редактор коду (Code editor): Основне місце для написання коду з підсвічуванням синтаксису та автодоповненням.

Консоль (Console): Для виконання команд та взаємодії з інтерпретатором Python.

Панель інструментів (Toolbar): Містить основні інструменти для запуску, зупинки, налагодження та тестування коду.

Налагодження та тестування PyCharm надає потужні інструменти для налагодження коду, що дозволяють: Ставити точки зупину (breakpoints): Можливість зупиняти виконання програми в певних місцях для аналізу стану змінних.

Пошагове виконання коду (Step-by-step execution): Дозволяє виконувати програму покроково, щоб зрозуміти логіку роботи.

Перегляд стеку викликів (Call stack): Допомагає відстежувати, як функції викликають одна одну.

Тестування коду: PyCharm підтримує різні фреймворки для тестування, такі як

unittest, pytest, doctest, що дозволяє легко створювати і запускати тести. Інтеграція з базами даних PyCharm підтримує інтеграцію з різноманітними базами даних, такими як MySQL, PostgreSQL, SQLite та іншими. Це дозволяє розробникам легко переглядати таблиці, стовпці та індекси. Безпосередньо з PyCharm можна виконувати SQL-запити і переглядати результати.Можливість редагувати дані в таблицях безпосередньо з середовища розробки.

Плагіни та розширення це є однією з головних переваг PyCharm є підтримка плагінів, що дозволяє розширювати функціональні можливості IDE. Існує безліч плагінів для різних завдань, таких як інтеграція із системами управління проектами, розширення можливостей налагодження, додавання підтримки нових мов програмування та багато іншого.

1.9 Реалізація проекту потокового шифрування

1.9.1Опис алгоритму потокового шифрування

У цьому розділі описується процес розробки програми для демонстрації двох шифрів: RC4 і власного шифру. Програми реалізовані мовою Python і дозволяє продемонструвати роботу кожного із шифрів на прикладі. Для реалізації потокового шифрування в Python ми використовуємо бібліотеку cryptography, що забезпечує безпечні та ефективні алгоритми шифрування.

1.9.2 Створення проекту в PyCharm

Відкриваємо PyCharm та вибераємо варіант створення нового проекту. Додаємо проекту назву, виберіть місцезнаходження та налаштуйте інтерпретатор Python. PyCharm дозволяє вибрати та налаштувати різні інтерпретатори Python, включаючи віртуальні середовища (virtual environments). Встановлення необхідних бібліотек для роботи з шифруванням нам знадобиться бібліотека криптографії. Її можна встановити через вбудований інструмент керування пакетами. Створіть необхідні директорії та файли для зберігання коду, тестів та ресурсів.

Но для початку створюється алгоритм це важливий крок програми у процесі розробки. По перше це концептуалізація. Алгоритм допомагає уявити загальну структуру та логіку програми, перш ніж переходити до написання коду. Ще оокументація алгоритм служить документацією, яка пояснює, як програма має

працювати, що є корисним для розуміння та подальшої підтримки програми. Створення алгоритму перед програмуванням також допомагає уникнути помилок у логіці програми та забезпечує, що всі вимоги до програми були враховані. Це особливо важливо для роботи, де потрібно продемонструвати глибоке розуміння предмету та здатність застосувати на практиці.

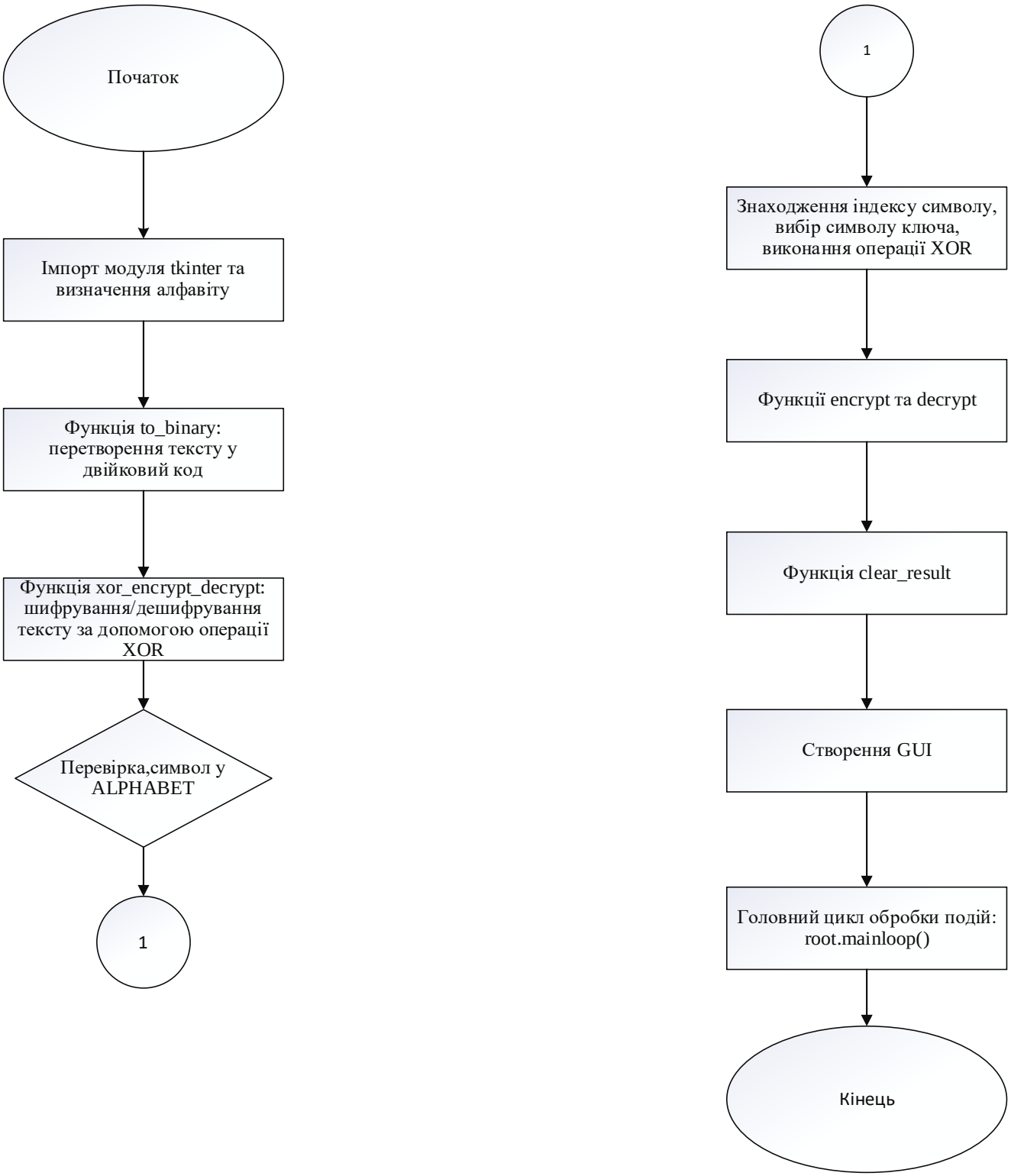


Рисунок 1.5. Алгоритм програми простого поточного шифра XOR

1.10 Створення програм

Після створення алгоритму. Переходимо до створення програмного коду.

1.10.1 Простий поточний шифр XOR

Спочатку буде створений власний шифр. Після створення алгоритму ми переходимо до програмної частини. Спочатку імпортуємо модуль tkinter, який дозволяє створювати графічний інтерфейс користувача.

```
import tkinter as tk
```

Потім створюємо алфавіт, який використовується для шифрування/дешифрування. Він включає великі та малі літери англійського та українського алфавітів, а також цифри.

```
ALPHABET =
'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyzАБВГҐДЕЄЖ
ЗИІЇЙКЛМНОПРСТУФХЦЧШЩЬЮЯабвгґдеєжзиіїйклмнопрстуфхцчшщьюя012
3456789'
```

Далі ми створюємо функцію перетворення тексту в двійковий код. Функція to_binary перетворює кожен символ вхідного рядка на його двійкове представлення.

```
def to_binary(text):
    # ...
def xor_encrypt_decrypt(text, key):
    # ...
```

Наступною є функція XOR шифрування/дешифрування. За допомогою функції xor_encrypt_decrypt виконується шифрування або дешифрування вхідного тексту за допомогою ключа, використовуючи операцію XOR. Далі йде створення основного вікна програми. Створюється основне вікно програми з назвою “XOR Шифрування та Дешифрування”, де розміщуються всі віджети для введення даних, кнопки дій та відображення результатів.

```
root = tk.Tk()
root.title("XOR Шифрування та Дешифрування")
```

```
# ...
root.mainloop()
```

Головний цикл обробки подій GUI запускається за допомогою root.mainloop(). У цій частині буде продемонстровано роботу програми, яка реалізує алгоритм потокового шифрування на основі мови програмування Python. Буде показано, як програма шифрує та дешифрує дані, використовуючи власний алгоритми , і продемонстровано скріншотами з поясненнями.

У цій програмі було продемонстровано шифрування алфавіту з латиниці і кирилиці, а також цифр від 0 до 9. Шифр здійснювався за допомогою операції XOR. На рисунку можемо побачити інтерфейс програми, де видно вхідний рядок, в який користувач вводить текст для шифрування, та рядок для введення ключа — випадковий ключ. Вихідний текст може бути представлений у символах або в бінарному коді, що дозволяє перевірити роботу алгоритму на різних рівнях представлення даних. Програма дозволяє з легкістю експериментувати з різними ключами та спостерігати, як змінюється шифрований текст. Такий підхід демонструє основні принципи роботи шифрів на основі операції XOR та допомагає зрозуміти їхню функціональність у криптографічних системах. Шифрування та розшифрування за допомогою XOR є простим і ефективним способом забезпечення конфіденційності переданих даних. Основна перевага такого методу полягає у його швидкості та легкості реалізації. Однак важливо зазначити, що безпека XOR шифру багато в чому залежить від довжини та якості ключа. Використання коротких або передбачуваних ключів може призвести до швидкого зламу шифру. Програма, створена для демонстрації цього алгоритму, також включає функціонал для відображення двійкового коду зашифрованого тексту. Це корисно для розуміння того, як саме операція XOR змінює бітове представлення символів. Користувач може спостерігати, як різні ключі впливають на кінцевий результат, що сприяє глибшому розумінню роботи криптографічних алгоритмів. Крім того, інтерфейс програми створений з урахуванням зручності користувача. Він дозволяє легко вводити дані та ключі, а також одразу бачити результати

шифрування та дешифрування. Це робить програму не тільки навчальним інструментом, але й практичним засобом для експериментів з криптографією.

Варто підкреслити, що хоча операція XOR є основою для багатьох складніших криптографічних алгоритмів, сама по собі вона не забезпечує достатнього рівня безпеки для захисту важливої інформації. У реальних системах шифрування використовуються складніші методи та підходи, які включають комбінування кількох криптографічних операцій, забезпечення випадковості ключів та додаткові механізми захисту.

Загалом, цей проект демонструє важливі аспекти криптографії та допомагає зрозуміти базові принципи шифрування за допомогою простих алгоритмів. Він служить відправною точкою для подальшого вивчення більш складних та надійних криптографічних методів, необхідних для забезпечення безпеки інформації в сучасному цифровому світі.

1.10.2 Шифр RC4

Для початку створюється алгортим.

Процес створення алгоритму RC4 починається з імпорту необхідних бібліотек, які забезпечують основні функції для шифрування та дешифрування. Після цього відбувається ініціалізація ключа за допомогою алгоритму KSA (Key Scheduling Algorithm). KSA відповідає за створення первинної перестановки байтів на основі заданого ключа. Далі генерується псевдовипадкова послідовність за допомогою алгоритму PRGA (Pseudo-Random Generation Algorithm), яка використовується для створення потоку ключів. Цей потік ключів потім комбінується з вихідним текстом за допомогою операції XOR для отримання зашифрованого тексту або розшифрування шифротексту. На діаграмі представлено етапи створення та використання алгоритму RC4.

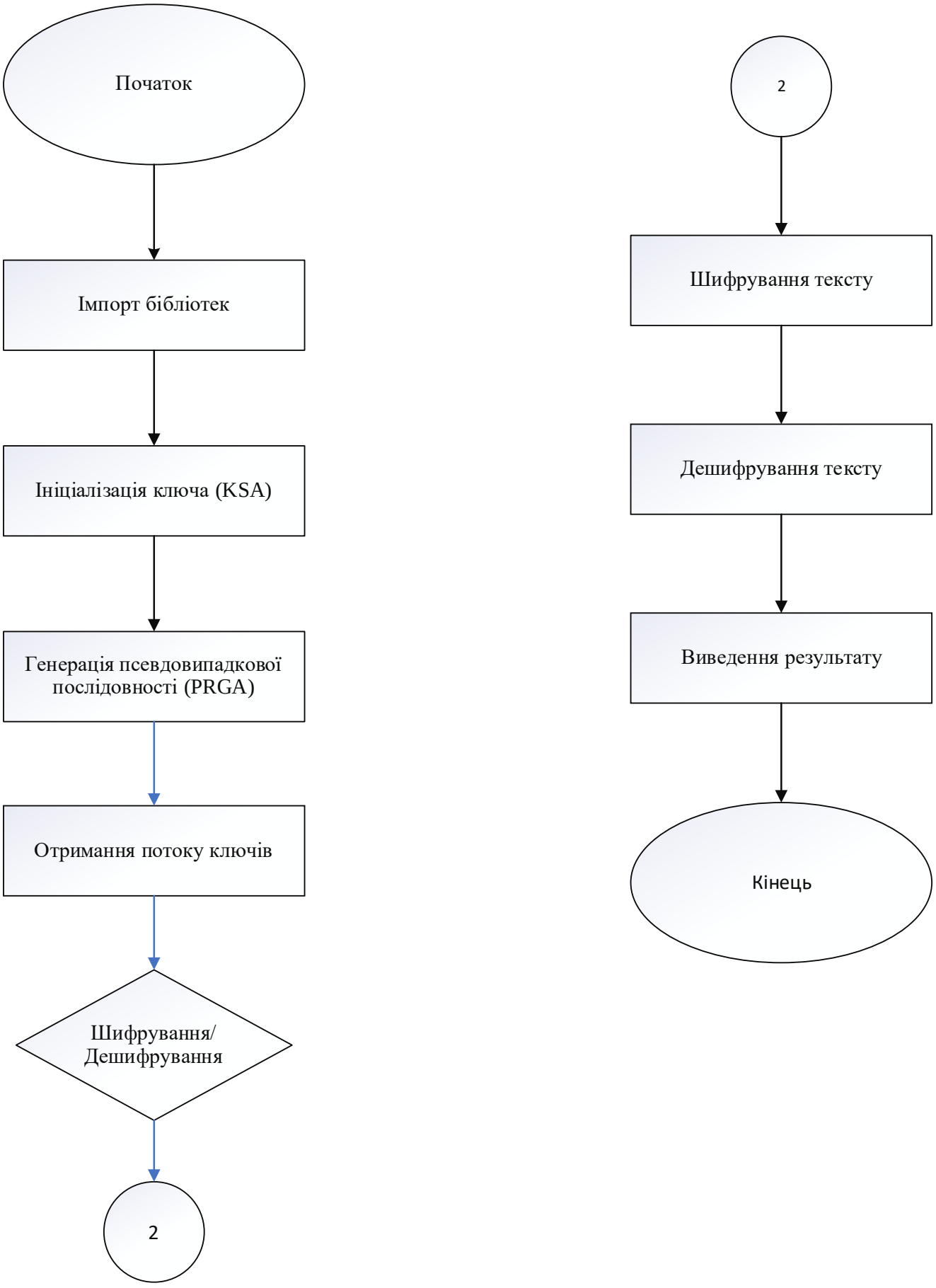


Рисунок 1.6. Алгоритм програми RC4

Спочатку ми імпортуємо модуль tkinter для створення графічного інтерфейсу користувача та messagebox для відображення діалогових вікон з повідомленнями.

```
import tkinter as tk
from tkinter import messagebox
```

Далі Функція KSA (Key Scheduling Algorithm) ініціалізує масив S, використовуючи секретний ключ.

```
def PRGA(S):
    # ...
```

Функція PRGA (Pseudo-Random Generation Algorithm) генерує псевдовипадкову послідовність для шифрування.

```
def get_keystream(key):
    # ...
```

Функція get_keystream використовує KSA та PRGA для створення потоку ключів.

```
def encrypt_logic(key, text):
    # ...
```

Функція encrypt_logic виконує шифрування тексту, використовуючи потік ключів.

```
def decrypt_logic(key, text):
    # ...
def encrypt():
    # ...
```

Функції encrypt та decrypt інтерактивно зв’язуються з користувачем через GUI.

```
root = tk.Tk()
root.title("RC4 Шифрування та Дешифрування")
# ...
```

Створюється основне вікно програми з назвою “RC4 Шифрування та Дешифрування”.

```
text_entry = tk.Entry(root, width=40)
key_entry = tk.Entry(root, width=40)
```

```
result_label = tk.Label(root, text="")

# ...
```

Віджети для введення тексту та ключа, а також мітка для відображення результату розміщуються у вікні.

```
encrypt_button = tk.Button(root, text="Зашифрувати", command=encrypt)

decrypt_button = tk.Button(root, text="Розшифрувати", command=decrypt)

root.mainloop()
```

Кнопки для шифрування та дешифрування дозволяють користувачу взаємодіяти з програмою, а головний цикл програми запускається для обробки подій.

1.11 Демонстрація роботи програми

У цьому розділі буде продемонстровано роботу програми, яка реалізує алгоритм потокового шифрування на основі мови програмування Python.

1.11.1 Простий поточний шиф XOR

У цій програмі було продемонстровано програму, яка шифрує алфавіт з латиниці і кирилиці, а також цифри від 0 до 9. Сам шифр шифрував через XOR**.** На рисунку можемо побачити сам інтерфейс програми, де можна побачити вхідний рядок, куди пишемо шифрований текст, а в рядку ключа — випадковий ключ. За допомогою яких ми можемо отримати шифрований текст у символах або у бінарному коді.

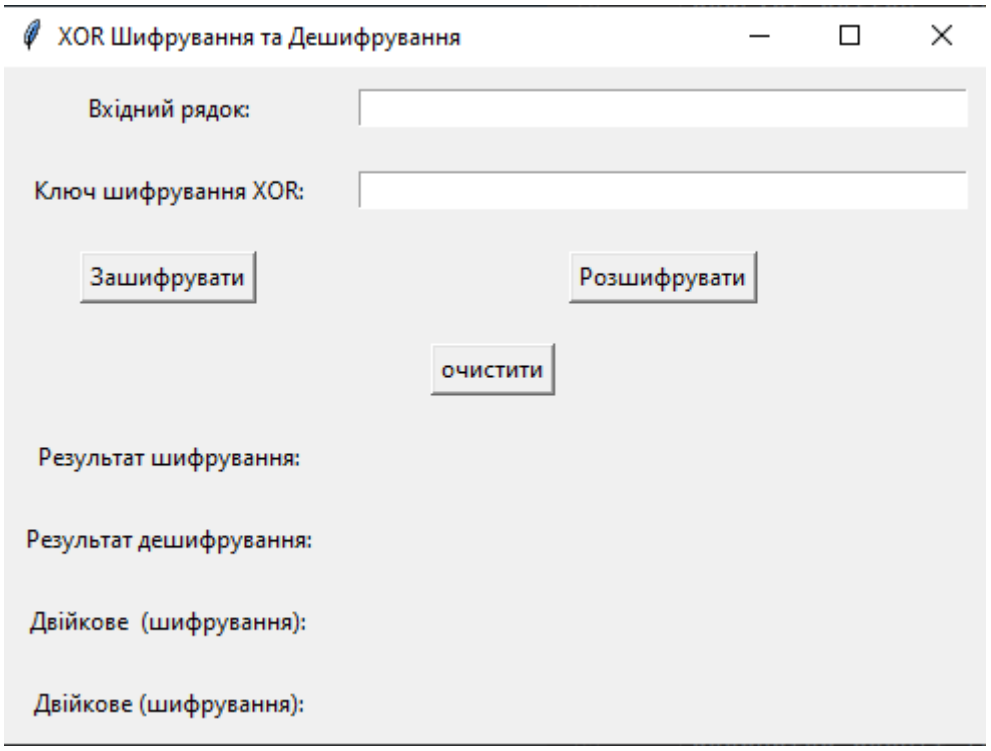


Рисунок 1.7. Демонстрація программи

На було протестовано програму на працездатність. Як можна побачити, для прикладу використовуємо слово для прикладу “HELP” і випадковий ключ. Ось ми можемо побачити, як після натискання кнопки шифрування, ми отримуємо шифрований текст та бінарний шифр.

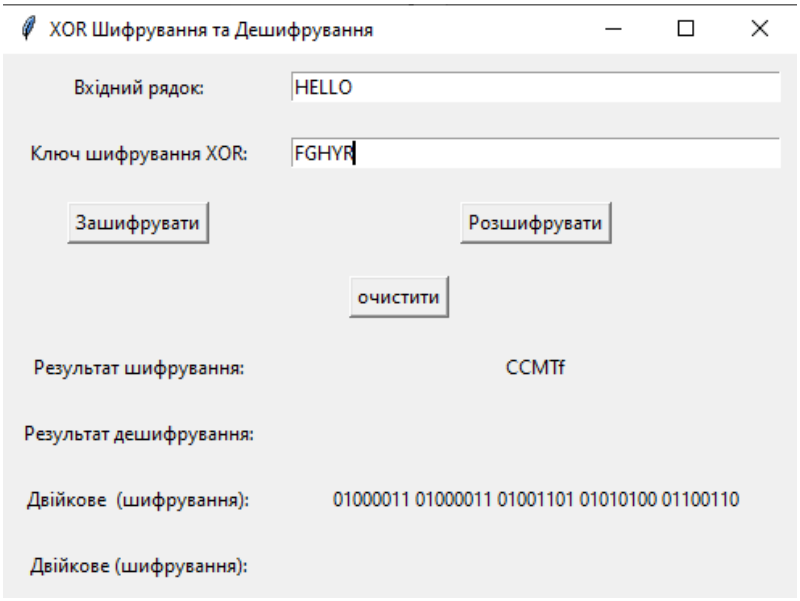


Рисунок 1.8. Демонстрація шифрування программи

На рис. 3 було продемонстровано, як після натискання на кнопку “Розшифрувати”, був показаний бінарний код розшифрованого тексту.

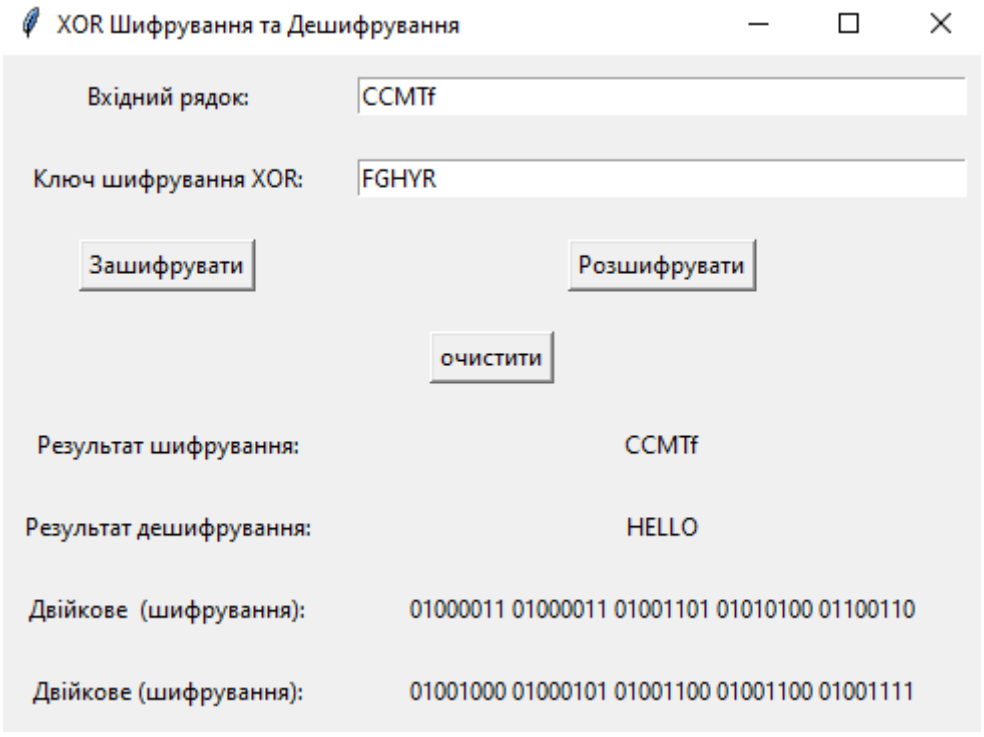


Рисунок 1.9. Демонстрація дешифрування программи

На рис. 4 можна побачити, що після натискання кнопки “очистити” можна побачити, як програма перейшла в початковий стан.

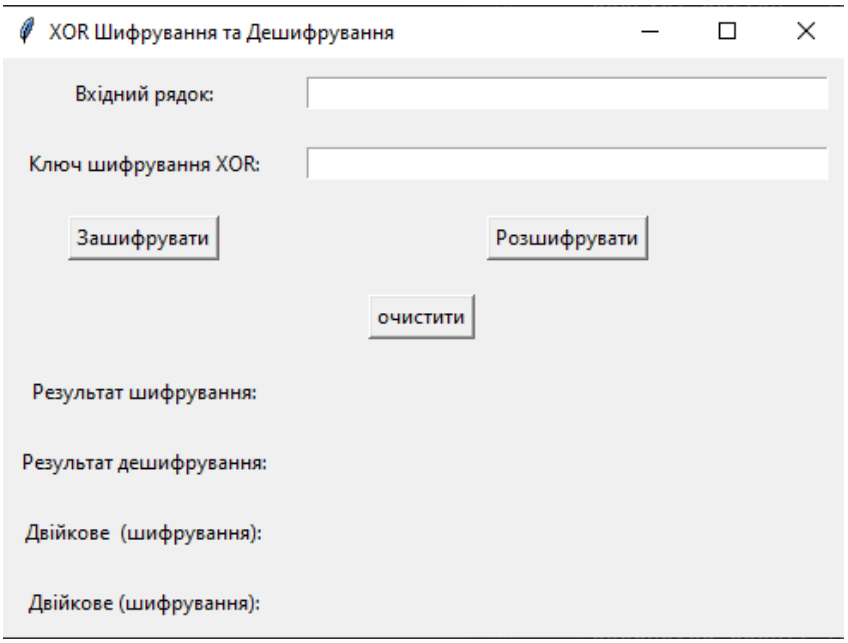


Рисунок 1.10. Демонстрація очистки программи

Ця програма використовує простий поточний шифр XOR, демонструє ефективний спосіб шифрування тексту, що підтримує як латинський, так і кириличний алфавіти, а також цифри. Використання операції XOR забезпечує простий механізм шифрування, який, незважаючи на свою простоту, є досить надійним за умови використання випадкових ключів. Однією з головних переваг використання XOR є його симетричність, тобто той самий ключ, що використовувався для шифрування, може бути використаний для дешифрування. Це робить процес шифрування/дешифрування дуже простим та ефективним з точки зору обчислювальних витрат. Програма має інтуїтивно зрозумілий інтерфейс, що дозволяє легко вводити текст і ключ, шифрувати його та переглядати результати у різних форматах. Відображення бінарного коду шифрованого тексту додає додатковий рівень прозорості та дозволяє користувачам краще зрозуміти процес шифрування на побітовому рівні. Протестований приклад зі словом "HELP" демонструє, як текст перетворюється у зашифрований вигляд за допомогою випадкового ключа, і як цей текст може бути повернений у вихідний стан при використанні того ж самого ключа. Це підтверджує правильність та надійність алгоритму. Функція очистки інтерфейсу також є корисною для користувачів, оскільки дозволяє швидко підготувати програму до нової сесії шифрування або дешифрування. Це зручно для багаторазового використання програми без необхідності її перезапуску. Таким чином, проста реалізація шифру XOR демонструє ефективність цього методу для базового шифрування текстових даних. Використання випадкових ключів значно підвищує рівень безпеки, роблячи шифр достатньо надійним для повсякденного використання. Програма може бути корисною як для навчальних цілей, так і для простих практичних застосувань, де необхідно забезпечити базовий рівень захисту інформації.

1.11.2 Поточний шифр RC4

У цій програмі було продемонстровано алгоритм RC4, написаний на мові Python. На першому рисунку можна побачити простий інтерфейс: текстове поле

під назвою “Вхідний текст”, де ми вводимо текст, який потрібно зашифрувати. У третьому полі, підписаному як “Ключ шифрування”, ми вводимо випадковий ключ.

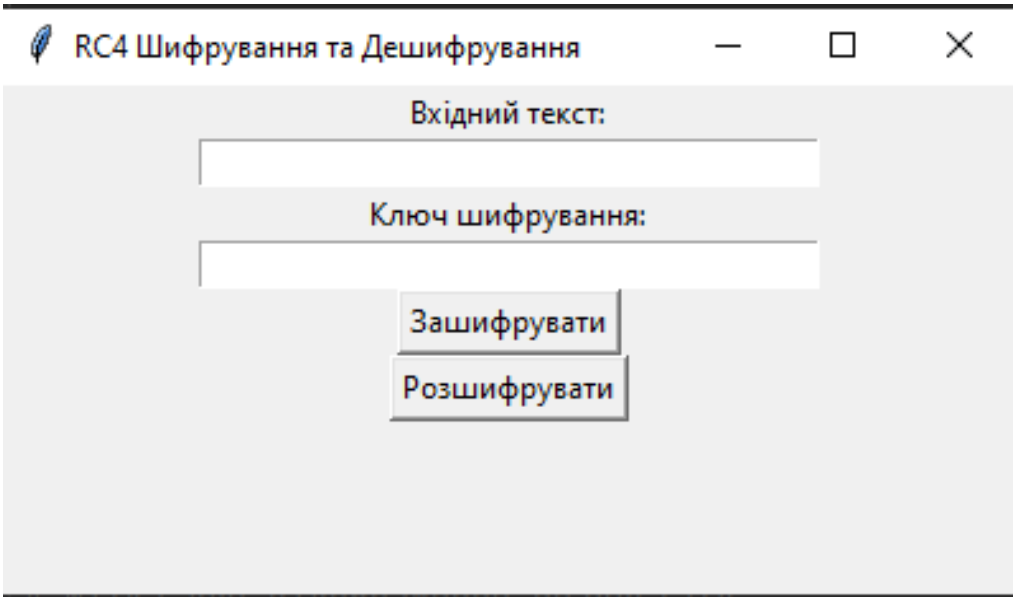


Рисунок 1.11. Демонстрація RC4 программи

На рисунку було протестовано програму. Для прикладу використовуємо слово для прикладу “HELLO” і випадковий ключ а саме q1j. Ось ми можемо побачити, як після натискання кнопки шифрування, ми отримуємо шифрований текст та бінарний шифр.

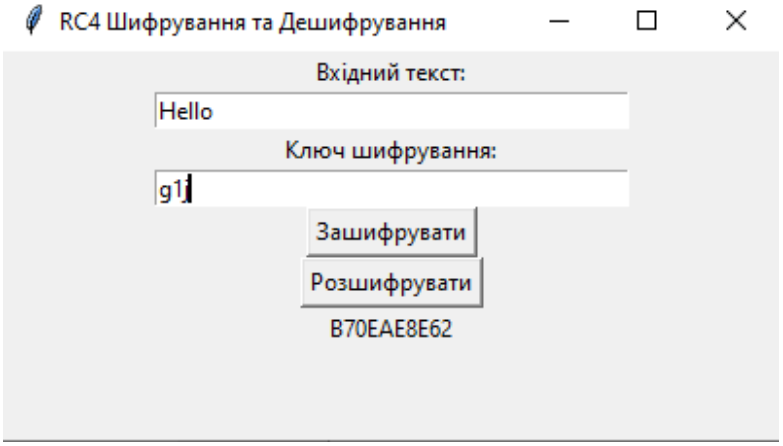


Рисунок 1.12. Демонстрація шифрування та жешифрування программи

Як можно побачити ми отримуємо шифротекст в 16 річній системі. Але головне в цих програмах продемонструвати, як легко та цікаво можна програмувати потокові шифри текстів на мові Python, а також показати, як працює

шифрування тексту. RC4 є одним з найбільш відомих потокових шифрів і широко використовується в різних системах безпеки. Незважаючи на свою простоту, RC4 забезпечує досить високий рівень захисту інформації за умов правильного використання ключів. Алгоритм RC4 заснований на простих перестановках і замінах, що робить його ефективним з точки зору швидкості обробки даних.

Інтерфейс програми дозволяє користувачу зручно вводити текст для шифрування та ключ, а також одразу бачити результати. Це робить програму не тільки навчальним інструментом, але й практичним засобом для експериментів з шифруванням. Використання випадкових ключів дає можливість побачити, як змінюється шифротекст при зміні ключа, що підкреслює важливість вибору надійних ключів для забезпечення безпеки. Програма також включає функціонал для відображення зашифрованого тексту у двійковому форматі, що дозволяє користувачам краще зрозуміти процес шифрування на рівні бітів. Це важливо для розуміння того, як працює RC4 і які зміни вносить ключ у бітову структуру тексту. Основною перевагою RC4 є його швидкість та ефективність, особливо при обробці великих обсягів даних. Це робить його ідеальним для використання в різних мережевих протоколах та системах зв’язку. Однак важливо пам’ятати, що RC4 має деякі вразливості, особливо якщо ключі використовуються повторно або мають недостатню довжину. Тому в реальних системах безпеки важливо дотримуватися рекомендацій щодо безпечного використання алгоритму. Цей проект демонструє важливі аспекти роботи RC4 і надає користувачам можливість побачити алгоритм у дії. Він служить відправною точкою для подальшого вивчення більш складних та надійних криптографічних методів, які використовуються для забезпечення безпеки інформації у сучасному цифровому світі.

2 РОЗДІЛ ОХОРОНИ ПРАЦІ ТА ТЕХНІКИ БЕЗПЕКИ

Однією з найважливіших аспектів життя людини є робота. У кожній професії існує ризик отримання травм, тому важливо дотримуватися встановлених норм контролю та безпеки. Охорона праці в галузі інформаційних технологій представляє собою цілісну систему заходів для працівників, що включає та охоплює аспекти, такі як медичні та правові. Ця система гарантує дотримання прав працівників, які забезпечені Конституцією та законодавством України, і спрямована на забезпечення безпеки, здоров'я та працездатності програмістів у процесі їхньої роботи. Важливим є проведення детального аналізу їх робочого місця та умов праці для ідентифікації шкідливих та небезпечних факторів, таких як стресові ситуації, які можуть виникнути під час програмування за комп'ютером. Наступним кроком є оцінка впливу цих факторів та розробка ефективних заходів та методів захисту для зниження ризику та негативного впливу .

У дипломному проекті включає дослідження системи потокового шифрування даних на основі мови програмування Python. Тому у цьому розділі дипломного проекту буде розглядається питання охорони праці саме програміста.

2.1 Вимоги до робочого місця програміста

Робоче місце програміста має відповідати низці вимог, спрямованих на забезпечення комфортних та безпечних умов праці. Ці вимоги включають освітлення, ергономіку та мікроклімат.

2.1.1 Ергономіка

Ергономічні вимоги до робочого місця програміста спрямовані на створення умов, що сприяють зниженню фізичного навантаження та втомлюваності. Основні ергономічні вимоги включають:

- **Робочий стіл**: Стіл має мати достатні розміри для розміщення всього необхідного обладнання, документів та особистих речей. Рекомендована

висота столу становить 680-760 мм. Поверхня столу має бути матовою, щоб уникнути відблисків.

- **Робочий стілець**: Стілець має бути регульованим по висоті і забезпечувати підтримку поперекового відділу хребта. Висота сидіння має бути 400-500 мм. Рекомендується використання стільців з підлокітниками і можливістю регулювання кута нахилу спинки.
- **Монітор**: Екран монітора має розташовуватися на відстані 50-70 см від очей користувача. Верхня межа екрану має знаходитися на рівні очей або трохи нижче, щоб мінімізувати навантаження на шию та очі. Кут нахилу екрану має складати 10-20 градусів.
- **Клавіатура та миша**: Клавіатура має розташовуватися на рівні ліктів, щоб руки знаходилися в природному положенні під час набору тексту. Миша має розташовуватися поруч з клавіатурою на одній висоті. Рекомендується використання ергономічних моделей клавіатури та миші для зниження навантаження на зап'ястя.

2.1.2 Освітлення

Освітлення на робочому місці програміста має бути організовано таким чином, щоб мінімізувати навантаження на зір та забезпечити комфортні умови для роботи. Згідно з **«ДБН В 2.5-28-2018 » Природне та штучне освітлення»** вимоги до освітлення включають:

- **Природне освітлення**: Робочі місця мають розташовуватися таким чином, щоб денне світло поступало збоку, бажано зліва. Це дозволяє уникнути прямого попадання світла на екран монітора і зменшує відблиски. Коефіцієнт природної освітленості (КПО) має складати не менше 1,5%.
- **Штучне освітлення**: Для забезпечення комфортних умов роботи за недостатнього природного освітлення використовуються системи штучного освітлення. Освітленість на робочому місці має складати не менше 300-500 люкс. Рекомендується використання розсіяного освітлення для мінімізації

відблисків і тіней. Лампи денного світла і світлодіодні світильники є кращими для забезпечення рівномірного освітлення.

2.1.3 Мікроклімат

Параметри мікроклімату на робочому місці програміста відіграють важливу роль в забезпеченні комфортних умов праці та збереженні здоров'я працівників. Згідно з **ДСН 3.3.6.042-99 « Санітарні норми мікроклімату виробничих приміщень»** параметри мікроклімату включають:

- **Температура повітря**: Температура повітря в приміщенні має бути в межах 22-24 °C в холодний період року і 23-25 °C в теплий період. Різкі коливання температури та протяги неприпустимі.
- **Відносна вологість повітря**: Відносна вологість повітря має складати 40-60%. Для підтримання оптимального рівня вологості рекомендується використання зволожувачів повітря та регулярне провітрювання приміщення.
- **Провітрювання та кондиціонування**: Приміщення має бути обладнане системою вентиляції або кондиціонування для забезпечення регулярного оновлення повітря та підтримання комфортного мікроклімату. Рекомендується провітрювання приміщення не рідше ніж кожні 2 години.

2.2 Профілактика професійних захворювань

Робота програміста пов'язана з ризиком розвитку різних професійних захворювань, таких як синдром зап'ястного каналу, проблеми із зором та захворювання опорно-рухового апарату. Основні заходи профілактики включають:

- **Використання ергономічних меблів та обладнання**: Застосування ергономічних меблів та обладнання допомагає знизити навантаження на організм та запобігти розвитку професійних захворювань. Рекомендується використання регульованих по висоті столів та стільців, ергономічних клавіатур та мишей.

- **Регулярна фізична активність**: Для підтримання здоров'я та профілактики захворювань опорно-рухового апарату рекомендується регулярна фізична активність, включаючи гімнастику, розтяжку та вправи для рук і ніг. Фізичні вправи допомагають поліпшити кровообіг, зміцнити м'язи та зняти напруження.
- **Правильна постава**: Слідкувати за правильною поставою під час роботи за комп'ютером дуже важливо для профілактики захворювань хребта та опорно-рухового апарату. Спина має бути прямою, плечі розслаблені, а ноги стояти на підлозі або на спеціальній підставці.
- **Профілактика зорових навантажень**: Для зниження зорових навантажень рекомендується дотримуватися правил роботи за комп'ютером, включаючи регулярні перерви для відпочинку очей, використання спеціальних окулярів для роботи за комп'ютером та налаштування монітора для оптимальної яскравості та контрастності..

2.3 Безпека роботи з комп'ютерним обладнанням

Для забезпечення безпеки роботи з комп'ютерним обладнанням необхідно дотримуватися низки заходів обережності, спрямованих на запобігання травмам та аварійним ситуаціям. Основні заходи безпеки включають:

- **Електробезпека**: Усі електричні пристрої мають бути правильно заземлені для запобігання ураження електричним струмом. Кабелі мають бути акуратно укладені, щоб уникнути ризику спіткнутися та пошкодити обладнання.
- **Захист від перепадів напруги**: Для захисту комп'ютерного обладнання від перепадів напруги рекомендується використання мережевих фільтрів та джерел безперебійного живлення (ДБЖ). Це дозволяє запобігти поломці обладнання та втраті даних у разі раптового вимкнення електроенергії.
- **Регулярне технічне обслуговування**: Комп'ютери та периферійні пристрої мають проходити регулярне технічне обслуговування та перевірку на

наявність несправностей. Регулярне обслуговування допомагає виявити та усунути потенційні проблеми до їх виникнення.

- **Використання ліцензійного програмного забезпечення**: Для забезпечення безпеки даних та запобігання вірусним атакам рекомендується використання ліцензійного програмного забезпечення та регулярне оновлення антивірусних програм.
- **Організація робочого простору**: Робоче місце має бути організоване таким чином, щоб мінімізувати ризики травм та створити комфортні умови для роботи. Усі предмети мають знаходитися на своїх місцях, а проходи та робоча поверхня бути вільними від сторонніх предметів.

І під час аналізу робочого місця програміста було виявлено, що організація безпечного та комфортного робочого місця програміста є важливою складовою праці. Дотримання норм і правил, встановлених законодавством України, допомагає створити умови які сприятливі для збереження здоров'я та підвищення продуктивності. Включення ергономічних рішень, правильна організація робочого часу, профілактика професійних захворювань та дотримання заходів безпеки під час роботи з комп'ютерним обладнанням є ключовими аспектами охорони праці для програмістів. Дотримання цих рекомендацій дозволяє мінімізувати ризики та створити сприятливі умови для ефективної та безпечної роботи.

ВИСНОВОК

У ході виконання кваліфікаційної роботи було проведено глибоке дослідження потокових шифрів, зокрема RC4, операції XOR та ChaCha20, на мові програмування Python. Робота включала аналіз теоретичних основ потокових шифрів, їхньої структури та принципів роботи. Було розглянуто особливості реалізації шифрів на Python, що дозволило оцінити переваги та недоліки кожного з них у контексті сучасних вимог до криптографічного захисту інформації. RC4, як один з найвідоміших потокових шифрів, був детально проаналізований з точки зору його криптостійкості та можливостей оптимізації. Операція XOR, яка лежить в основі багатьох потокових шифрів, була розглянута як фундаментальний елемент для забезпечення симетричного шифрування. ChaCha20, як сучасний потоковий шифр, продемонстрував високу ефективність та безпеку, що робить його придатним для використання в різноманітних криптографічних системах. Практична частина роботи включала розробку програмного забезпечення на Python, яке демонструє процес шифрування та дешифрування тексту з використанням алгоритмів. Програми були реалізовані з використанням графічного інтерфейсу користувача, що забезпечило зручність та інтуїтивність у використанні. Загалом, дослідження підтвердило, що мова програмування Python є ефективним інструментом для розробки криптографічного програмного забезпечення, здатного забезпечити надійний захист інформації. Результати роботи можуть бути використані для подальшого вдосконалення потокових шифрів та розробки нових криптографічних рішень. Водночас, робота вказує на необхідність постійного оновлення знань у галузі криптографії, адже сфера інформаційної безпеки є динамічною та вимагає від фахівців гнучкості та готовності до швидкого реагування на нові виклики.

ПЕРЕЛІК ВИКОРИСТАНИХ ІНФОРМАЦІЙНИХ ДЖЕРЕЛ

1. Бондаренко В.І., Іваненко О.М. Основи криптографії та захисту інформації. - Київ: Наукова думка, 2015. – 420 с.
2. Ковальчук М.С. Програмування на Python: навчальний посібник. – Львів: СПОЛОМ, 2018. – 298 с.
3. Петров В.М. Інформаційна безпека: технології та засоби захисту. - Харків: ХНУРЕ, 2016. – 375 с.
4. Шевченко О.О. Математичні основи криптографії. – Київ: Видавництво КНУ, 2019. – 289 с.
5. Василенко П.П., Горбенко В.П. Комп'ютерна безпека та криптографія: навчальний посібник. - Київ: Видавничий дім "Києво-Могилянська академія", 2021. – 256 с.
6. Сидоренко А.В. Кібербезпека: захист інформації та програмних систем. – Одеса: Центр учбової літератури, 2020. – 310 с.
7. Бондар В.О., Поляков І.М. Захист інформації в комп'ютерних системах. – Львів: Новий Світ-2000, 2017. – 340 с.
8. Мельник О.І., Сидоренко В.М. Криптографічні методи захисту інформації. - Київ: Академвидав, 2016. – 320 с.
9. Левченко І.П., Тарасенко Ю.М. Системи та методи захисту інформації: навчальний посібник. - Харків: ХНУРЕ, 2017. – 295 с.
10. Дмитренко В.В. Кібербезпека: основи та практика. - Львів: Видавництво ЛНУ, 2020. – 360 с.
11. Кравченко О.С. Програмні засоби захисту інформації. - Одеса: Видавництво ОНУ, 2018. – 275 с.
12. Чорний О.В., Петрова Н.С. Алгоритми та протоколи криптографії. - Дніпро: ДНУ, 2019. – 310 с.
13. Гаврилюк А.В. Основи захисту інформації: теорія та практика. - Тернопіль: Видавництво ТНТУ, 2021. – 280 с.
14. Михайлов О.П. Інформаційна безпека в комп'ютерних мережах. - Вінниця: ВНТУ, 2017. – 300 с.
15. Ткаченко В.Л. Методи та засоби криптографічного захисту. - Житомир: ЖДУ, 2018. – 290 с.

ДОДАТОК А

Програма шифрування XOR

```
# Імпортуємо модуль tkinter для створення GUI
import tkinter as tk
# алфавіт, що використовується для шифрування/дешифрування
ALPHABET =
'ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyzАБВГҐДЕЄЖЗИІЇЙКЛМНОПР
СТУФХЦЧШЩЬЮЯабвгґдеєжзиіїйклмнопрстуфхцчшщьюя0123456789'
# Функція для перетворення рядка на двійковий код
def to_binary(text):
    return ' '.join(format(ord(x), 'b').zfill(8) for x in text)
# Функція для шифрування/дешифрування тексту
def xor_encrypt_decrypt(text, key):
    # Індекс для відстеження позиції у ключі
    key_index = 0
    encrypted_decrypted_text = ''  # Рядок для зберігання результату
    for char in text:  # Перебір кожного символу в тексті
        if char in ALPHABET:  # Перевірка  символ в алфавіті
            char_index = ALPHABET.index(char)  # Отримання індексу символу
            key_char = key[key_index % len(key)]  # Отримання символу ключа
            key_char_index = ALPHABET.index(key_char)  # Отримання індексу символу ключа
            # Застосування операції XOR та додавання результату у рядка
            encrypted_decrypted_text += ALPHABET[(char_index ^ key_char_index) % len(ALPHABET)]
            key_index += 1  # Перехід до наступного символу ключа
        else:
            encrypted_decrypted_text += char # Додавання символу без змін
    return encrypted_decrypted_text  # Повернення результату
# Повернення результату
def encrypt():
    text = text_entry.get()  # Отримання тексту від користувача
    key = key_entry.get()  # Отримання ключа від користувача
    result = xor_encrypt_decrypt(text, key)  # Шифрування тексту
    encrypted_text.set(result) # Отримання зашифрованого тексту
    binary_encrypted.set(to_binary(result))  # Отримання двійкового код
# Функція для дешифрування тексту
def decrypt():
    text = text_entry.get()  # Отримання шифр текст від користувача
    key = key_entry.get()  # Отримання ключа від користувача
    result = xor_encrypt_decrypt(text, key)  # Дешифрування тексту
    decrypted_text.set(result)  # Отримання дешифрованого тексту
    binary_decrypted.set(to_binary(result))  # Отримання двійкового код
# Функція для видалення результатів
def clear_result():
    encrypted_text.set('')
    decrypted_text.set('')
    binary_encrypted.set('')
    binary_decrypted.set('')
# Створення основного вікна програми
root = tk.Tk()
root.title("XOR Шифрування та Дешифрування")

# Створення та розміщення віджетів для введення тексту
text_label = tk.Label(root, text="Вхідний рядок:")
text_label.grid(row=0, column=0, padx=10, pady=10)
```

```
text_entry = tk.Entry(root, width=50)
text_entry.grid(row=0, column=1, padx=10, pady=10)
# Створення та розміщення віджетів для введення ключа
key_label = tk.Label(root, text="Ключ шифрування XOR:")
key_label.grid(row=1, column=0, padx=10, pady=10)
key_entry = tk.Entry(root, width=50)
key_entry.grid(row=1, column=1, padx=10, pady=10)
# Створення та розміщення кнопок для шифрування та дешифрування
encrypt_button = tk.Button(root, text="Зашифрувати", command=encrypt)
encrypt_button.grid(row=2, column=0, padx=10, pady=10)
decrypt_button = tk.Button(root, text="Розшифрувати", command=decrypt)
decrypt_button.grid(row=2, column=1, padx=10, pady=10)
# Створення та розміщення кнопки для видалення результатів
clear_button = tk.Button(root, text="Очистить результат", command=clear_result)
clear_button.grid(row=3, columnspan=2, padx=10, pady=10)
# Змінні для зберігання та відображення зашифрованого та дешифрованого тексту
encrypted_text = tk.StringVar()
encrypted_label = tk.Label(root, text="Результат шифрування:")
encrypted_label.grid(row=4, column=0, padx=10, pady=10)
encrypted_result_label = tk.Label(root, textvariable=encrypted_text)
encrypted_result_label.grid(row=4, column=1, padx=10, pady=10)
decrypted_text = tk.StringVar()
decrypted_label = tk.Label(root, text="Результат дешифрування:")
decrypted_label.grid(row=5, column=0, padx=10, pady=10)
decrypted_result_label = tk.Label(root, textvariable=decrypted_text)
decrypted_result_label.grid(row=5, column=1, padx=10, pady=10)
# Змінні для зберігання та відображення двійкового код
binary_encrypted = tk.StringVar()
binary_encrypted_label = tk.Label(root, text="Двійкове  (шифрування):")
binary_encrypted_label.grid(row=6, column=0, padx=10, pady=10)
binary_encrypted_result_label = tk.Label(root, textvariable=binary_encrypted)
binary_encrypted_result_label.grid(row=6, column=1, padx=10, pady=10)
binary_decrypted = tk.StringVar()
binary_decrypted_label = tk.Label(root, text="Двійкове (шифрування):")
binary_decrypted_label.grid(row=7, column=0, padx=10, pady=10)
binary_decrypted_result_label = tk.Label(root, textvariable=binary_decrypted)
binary_decrypted_result_label.grid(row=7, column=1, padx=10, pady=10)
# Запуск головного циклу обробки подій GUI
root.mainloop()
```

Програма RC4

```
Import tkinter as tk
from tkinter import messagebox

# Функція ініціалізації ключа
def KSA(key):
    key_length = len(key)
    S = list(range(256))
    j = 0
    for i in range(256):
        j = (j + S[i] + key[i % key_length]) % 256
        S[i], S[j] = S[j], S[i]
    return S

# Генератор псевдовипадкової послідовності
def PRGA(S):
    i = 0
    j = 0
    while True:
        i = (i + 1) % 256
        j = (j + S[i]) % 256
        S[i], S[j] = S[j], S[i]
        K = S[(S[i] + S[j]) % 256]
        yield K

# Функція для отримання потоку ключів
def get_keystream(key):
    S = KSA(key)
    return PRGA(S)

# Логіка шифрування
def encrypt_logic(key, text):
    key = [ord(c) for c in key]
    keystream = get_keystream(key)
    res = []
    for c in text:
        val = ("%02X" % (ord(c) ^ next(keystream)))  # XOR і перетворення в hex
        res.append(val)
    return ''.join(res)

# Логіка дешифрування
def decrypt_logic(key, text):
    key = [ord(c) for c in key]
    keystream = get_keystream(key)
    res = []
    for i in range(0, len(text), 2):
        val = text[i:i+2]
        val = chr(int(val, 16) ^ next(keystream))
        res.append(val)
    return ''.join(res)

# Функція шифрування
def encrypt():
    if not key_entry.get() or not text_entry.get():
        messagebox.showerror("Помилка", "Введіть текст та ключ")
        return
    encrypted_text = encrypt_logic(key_entry.get(), text_entry.get())
    result_label.config(text=encrypted_text)

# Функція дешифрування
```

```
def decrypt():
    if not key_entry.get() or not text_entry.get():
        messagebox.showerror("Помилка", "Введіть текст та ключ")
        return
    decrypted_text = decrypt_logic(key_entry.get(), text_entry.get())
    result_label.config(text=decrypted_text)

# Створення кореневого вікна
root = tk.Tk()
root.title("RC4 Шифрування та Дешифрування")
root.geometry("400x200")  # Збільшення розміру вікна

# Мітка для поля вводу тексту
text_label = tk.Label(root, text="Вхідний текст:")
text_label.pack()
# Поле вводу тексту
text_entry = tk.Entry(root, width=40)
text_entry.pack()

# Мітка для поля вводу ключа
key_label = tk.Label(root, text="Ключ шифрування:")
key_label.pack()
# Поле вводу ключа
key_entry = tk.Entry(root, width=40)
key_entry.pack()

# Кнопка шифрування
encrypt_button = tk.Button(root, text="Зашифрувати", command=encrypt)
encrypt_button.pack()

# Кнопка дешифрування
decrypt_button = tk.Button(root, text="Розшифрувати", command=decrypt)
decrypt_button.pack()

# Мітка для відображення результату
result_label = tk.Label(root, text="")
result_label.pack()

# Запуск головного циклу програми
root.mainloop()
```

ДОДАТОК Б

Слайди мультимедійної презентації

Дослідження системи потокового шифрування даних на основі мови програмування Python

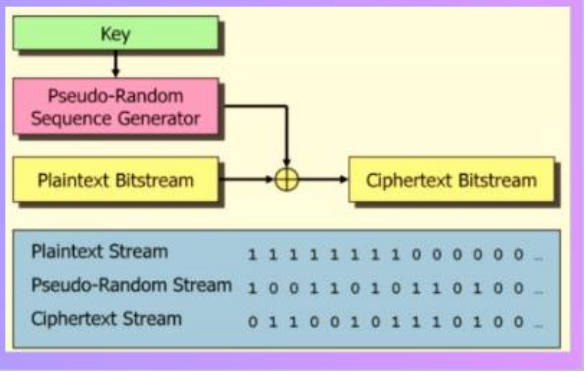
Розробник: ***Ситнікова А.С. ОТФК ОНТУ***

Керівник роботи: к.т.н. ***Кільдішев В.Й. ОТФК ОНТУ***

Що таке потоковий шифр?

Шифр - це метод перетворення інформації в так, щоб вона була незрозуміла для сторонніх осіб.

Потоковий шифр - це тип шифру, який шифрує дані по одному біту або символу за раз. Потокові шифри використовують генератор псевдовипадкових чисел для створення ключового потоку, який потім комбінується з відкритим текстом для отримання шифротексту.



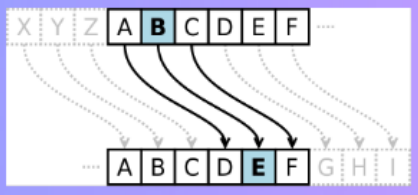
Огляд методів потокового шифрування

- Розвиток: Еволюція від ранніх шифрів до сучасних потокових шифрів.
- Сучасні методології: Алгоритми, такі як RC4, ChaCha20.
- Теоретичні аспекти: Основи безпеки та принципи.

Перші методи шифрування

1 Шифр Цезаря

Шифр Цезаря, один з найстаріших відомих шифрів, використовувався Юлієм Цезарем для захисту військових повідомлень. Метод полягає у зсуві кожної літери в повідомленні на фіксовану кількість позицій у алфавіті.



2 Поліалфавітні шифри

Шифри, такі як Віженера, використовували кілька алфавітів для шифрування, що підвищувало їхню складність і стійкість до зломів.

Перші потокві шифри

Шифр Вернама, також відомий як одноразовий блокнот, є системою симетричного шифрування, яку винайшли у 1917 році.

Основні принципи шифру Вернама:

Випадковість ключа: Ключ повинен бути абсолютно випадковим і використовуватися тільки один раз.

Довжина ключа: Довжина ключа має бути рівною довжині повідомлення, що шифрується.

Секретність ключа: Ключ має залишатися секретним і відомим тільки відправнику та одержувачу.

Шифрування та дешифрування в шифрі Вернама відбувається за допомогою операції XOR між кожним бітом повідомлення та відповідним бітом ключа.

Приклад Повідомлення: 01010100 (відповідає символу 'T' у ASCII)

Ключ: 11001100 (повинен бути випадковим та такої ж довжини, як повідомлення)

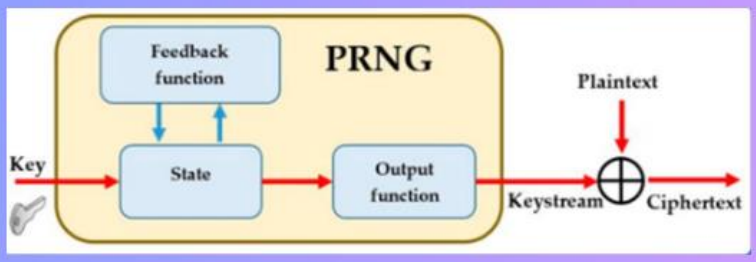
шифрування

```
Повідомлення:  01010100
Ключ:          11001100
XOR:           --------
Шифротекст:    10011000
```

дешифрування

```
Шифротекст:    10011000
Ключ:          11001100
XOR:           --------
Повідомлення:  01010100
```

Генератори псевдовипадкових чисел



Ця схема є універсальною і може бути застосована для різних алгоритмів потокових шифрів, таких як RC4 та ChaCha20, забезпечуючи високу швидкість шифрування і безпеку даних.

Класифікація потокових шифрів

•Синхронні потокові шифри: Потік ключів не залежить від відкритого тексту та шифротексту.

1 Генератор ключового потоку (Key Stream Generator) генерує потік ключів (Key Stream).
2 Вхідний відкритий текст (Plaintext) проходить через операцію XOR з потоком ключів.
3 На виході отримуємо шифротекст (Ciphertext).

•Асинхронні потокові шифри: Потік ключів залежить від попередніх шифротекстів (механізм зворотного зв'язку).

1 Генератор ключового потоку (Key Stream Generator) генерує потік ключів на основі попередніх шифротекстів.
2 Вхідний відкритий текст (Plaintext) проходить через операцію XOR з потоком ключів.
3 На виході отримуємо шифротекст (Ciphertext).
4 Зворотний зв'язок (Feedback) від шифротексту впливає на генерацію наступного ключового потоку.

Порівняння потокових та блочних шифрів

- Блочні шифри: Шифрують дані блоками фіксованого розміру (наприклад, AES).
- Потокові шифри: Шифрують безперервний потік даних.

Приклади	AES, DES, 3DES, Blowfish	RC4, ChaCha20, Salsa20
Швидкість шифрування	Зазвичай повільніші через обробку великих блоків даних.	Зазвичай швидші, особливо для потокових даних.
Складність реалізації	Вища складність, потребує реалізації різних режимів шифрування.	Простіша реалізація, особливо для простих алгоритмів.
Безпека	Висока безпека, особливо з сучасними алгоритмами.	Може бути менш безпечними, особливо для простих алгоритмів (якщо використовувати слабкі ключі).
Режими шифрування	Підтримують різні режими шифрування (ECB, CBC, CFB, OFB).	Не потребують режимів шифрування.
Вимоги до пам'яті	Зазвичай вимагають більше пам'яті для обробки блоків.	Вимагають менше пам'яті, оскільки обробляють дані по одному біту/байту.
Стійкість до помилок	Помилка в одному блоці може вплинути на наступні блоки (залежно від режиму).	Помилка впливає лише на той біт/байт, де вона виникла.
Використання	Добре підходять для шифрування файлів та баз даних.	Добре підходять для шифрування потоків даних, таких як відео та аудіо.

Порівняльний аналіз методів потокового шифрування

Ефективність

Алгоритм	Швидкість генерації ключового потоку	Швидкість шифрування
RC4	Висока	Висока
ChaCha20	Дуже висока	Дуже висока
XOR	Дуже висока	Дуже висока

Безпека

Алгоритм	Вразливості	Рівень безпеки
RC4	Відомі вразливості	Низький
ChaCha20	Немає відомих серйозних вразливостей	Високий
XOR	Вразливий при використанні слабких ключів	Середній

Простота реалізації

Алгоритм	Складність реалізації	Вимоги до ресурсів
RC4	Середня	Низькі
ChaCha20	Висока	Середні
XOR	Низька	Низькі

Використання в реальних умовах

- Інтернет-комунікації: Шифрування даних під час передачі, VPN, безпечні повідомлення.

Детальний аналіз алгоритму RC4

основна формула алгоритму шифрування RC4:

- Ініціалізація:

S[i]=i

T[i]=K[imodklen]

де (S) - це масив стану, (K) - ключ, (klen) - довжина ключа.

- Початкова перестановка (S):

j=0

для i=0 до 255:

j=(j+S[i]+T[i])mod256

обмін S[i] та S[j]

- Генерація псевдовипадкового потоку:

i=j=0

поки вірно:

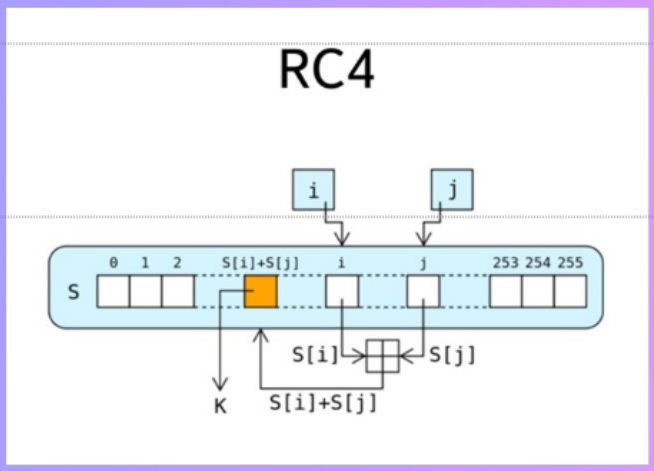
i=(i+1)mod256

j=(j+S[i])mod256

обмін S[i] та S[j]

t=(S[i]+S[j])mod256

K=S[t]



Детальний аналіз алгоритму Chacha20

1 Ініціалізація:

state=constants||key||counter||nonce

де `state` - це внутрішній стан, що складається з констант, ключа, лічильника та nonce (випадкове число).

- Перемішування (20 раундів):

2 Для кожного раунду:

QuarterRound(a,b,c,d)

де `QuarterRound` - це функція, що перемішує чотири елементи стану.

3 Додавання вихідного стану:

Вихідний стан=початковий стан+перемішаний стан

4 Виведення потоку ключів: Потік ключів генерується з вихідного стану, і кожен блок шифрується за допомогою операції XOR з відповідним блоком відкритого тексту.

Потоковий шифр з використанням операції XOR

Основні кроки алгоритму:

1 Генерація ключового потоку: Використання генератора псевдовипадкових чисел (PRNG) для створення ключового потоку, що має таку довжину, як і відкритий текст.

2 Шифрування: Виконує операцію XOR між кожним бітом відкритого тексту та відповідним бітом ключового потоку.

- Дешифрування: Виконання XOR між кожним бітом зашифрованого тексту і
- відповідним бітом ключового потоку для відновлення відкритого тексту.

Формула

Нехай P - відкритий текст, K - ключовий потік, C - зашифрований текст.

Шифрування:

$C_i = P_i \oplus K_i$

де C_i - біт зашифрованого тексту, P_i - біт відкритого тексту, K_i - біт ключового потоку.

Дешифрування:

$P_i = C_i \oplus K_i$

Обґрунтування вибору Python

Python є високорівневою мовою програмування, яка отримала визнання завдяки своїй простоті та зручності. Вона забезпечує широкий спектр інструментів та бібліотек, що полегшує реалізацію складних криптографічних алгоритмів. Переваги та недоліки Python для реалізації потокових шифрів розглянемо докладніше.

Python у порівнянні з іншими мовами

- Python: Читабельність, широкі бібліотеки, легкість використання.

Критерій	Переваги
Простота синтаксису	Легкий для читання та написання код
Багатство бібліотек	PyCryptodome, cryptography та інші
Кросплатформеність	Підтримка Windows, macOS, Linux
Спільнота	Велика та активна

- C++: Продуктивність, контроль над системними ресурсами.

Критерій	Недоліки
Складність синтаксису	Висока складність
Управління пам'яттю	Необхідність ручного управління

- Java: Портативність, надійні стандартні бібліотеки.

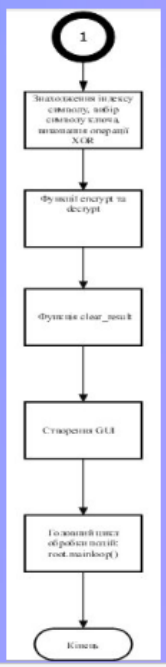
Критерій	Переваги
Кросплатформеність	Висока
Автоматичне управління пам'яттю	Знижений ризик помилок
Багатий набір бібліотек	BouncyCastle та інші

Реалізація проекту потокового шифрування

PyCharm – це потужне інтегроване середовище розробки (IDE), спеціально розроблене для мови програмування Python.

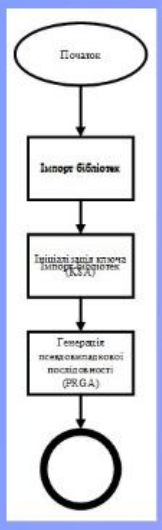
```python
import tkinter as tk
from tkinter import messagebox

# Функція ініціалізації ключа
def KSA(key):
    key_length = len(key)
    S = list(range(256))
    j = 0
    for i in range(256):
        j = (j + S[i] + key[i % key_length]) % 256
        S[i], S[j] = S[j], S[i]
    return S

# Генератор псевдовипадкової послідовності
def PRGA(S):
    i = 0
    j = 0
    while True:
        i = (i + 1) % 256
        j = (j + S[i]) % 256
        S[i], S[j] = S[j], S[i]
        K = S[(S[i] + S[j]) % 256]
        yield K

# Функція для отримання потоку ключів
def get_keystream(key):
    S = KSA(key)
```

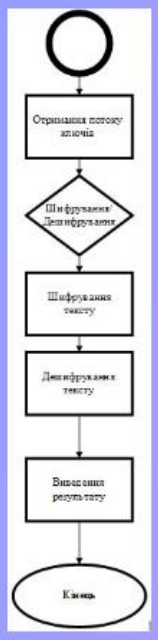
Алгоритм програми XOR



У цьому алгоритмі показує як в програмі на мові Python використовує бібліотеку tkinter для створення графічного інтерфейсу користувача (GUI), який дозволяє шифрувати та дешифрувати текст за допомогою методу XOR. Користувач може ввести текст та ключ у відповідні поля, а програма виконає шифрування або дешифрування, показуючи результати у вікні. Також є можливість перетворити текст у двійковий код та очистити всі поля.

Алгоритм програми RC4

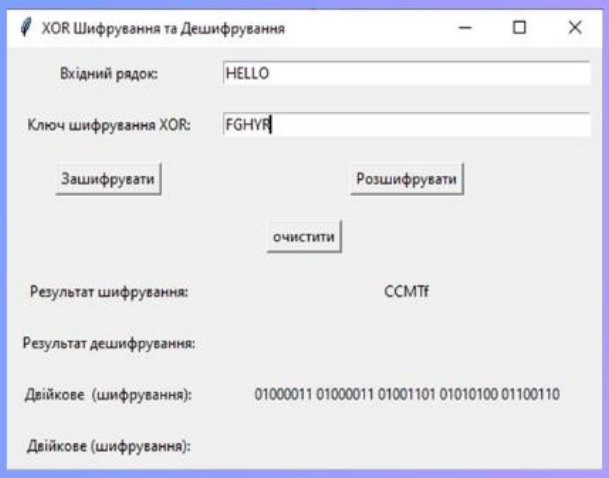


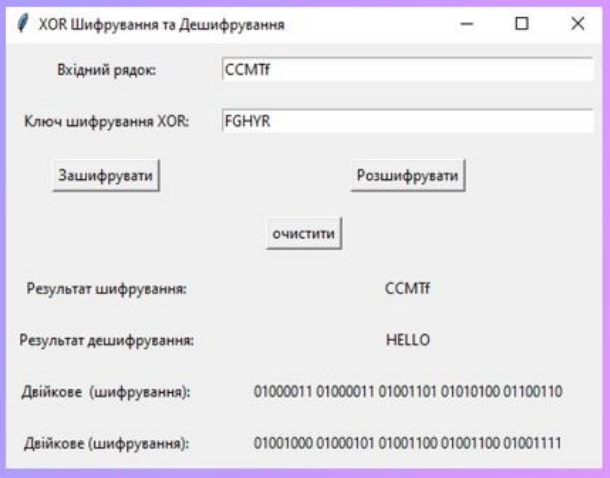


У цьому алгоритмі показує як в програмі на мові на Python використовує бібліотеку tkinter для створення простого графічного інтерфейсу користувача (GUI), який дозволяє шифрувати та дешифрувати текст за допомогою алгоритму RC4.

Демонстрація програм

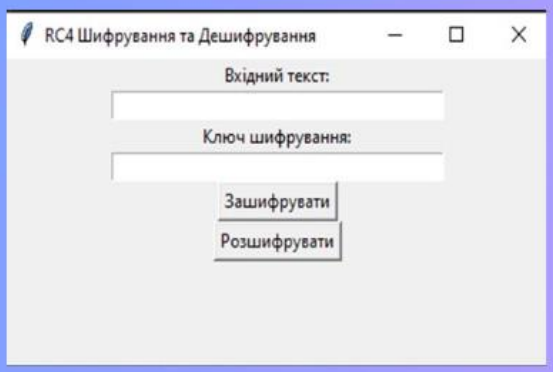
Простий потоковий шифр XOR: Реалізація та результати.

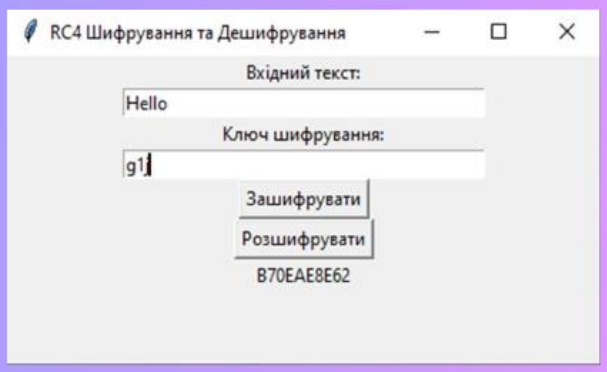




Демонстрація програм

Потоковий шифр RC4: Реалізація та результати.





Кінець

Дякую за увагу.

UNICHECK
by Turnitin

Ім'я користувача:
Катерина Григоріївна Краснокутська

Дата перевірки:
11.06.2024 17:27:56 EEST

Дата звіту:
11.06.2024 17:36:37 EEST

ID перевірки:
1016348393

Тип перевірки:
Doc vs Internet + Library

ID користувача:
100011688

Назва документа: 2БКС-28_Андрій_Ситніков

Кількість сторінок: 44 Кількість слів: 6287 Кількість символів: 46726 Розмір файлу: 253.66 KB ID файлу: 1016151515

1.5%
Схожість

Найбільша схожість: 0.38% з Інтернет-джерелом (https://research.nccgroup.com/2023/08/22/dancing-offbit-the-story-of...

1.5% Джерела з Інтернету | 154 ..... Сторінка 46

Не знайдено джерел з Бібліотеки

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

Замінені символи | 8

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

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

ВІДГУК

керівника про кваліфікаційну роботу бакалавра

Ситнікова Андрія Сергійовича
(прізвище, ім'я та по батькові здобувача/здобувачки освіти)

Освітньо-професійна програма *«Комп'ютерна інженерія»*

Спеціальність *123 «Комп'ютерна інженерія»*

Тема кваліфікаційної роботи

«Дослідження системи потокового шифрування даних на основі мови програмування Python»

ХАРАКТЕРИСТИКА КВАЛІФІКАЦІЙНОЇ РОБОТИ

а) обсяг і якість виконання роботи (розрахунково-пояснювальної записки)

Пояснювальна записка виконана якісно, у достатньому обсязі, відповідно до індивідуального завдання та теми дипломного проекту, розділи пояснювальної записки відповідають етапам рішення завдання, поставленого у дипломному проекті

Презентація виконана якісно, у достатньому обсязі. Презентація наочно демонструє результати роботи.

б) самостійність роботи над кваліфікаційною роботою

Студент самостійно обрав напрям та тематику кваліфікаційній роботи, присвяченої дослідженню систем потокового шифрування даних. Та в рамках застосування засобів захисту запропоновано використання мови програмування Python для реалізації алгоритмів шифрування. Особлива увага приділялася розробці та оптимізації алгоритмів на основі XOR.

в) теоретична підготовка бакалавра

відповідає вимогам, що надаються до бакалавра зі спеціальності «Комп'ютерна інженерія»

г) вміння розв'язувати виробничі та конструкторські питання _____

У кваліфікаційній роботі досліджується криптографічного шифрування даних даних.

Розглянуто роль поточних шифрів та обміну даних.

Оцінка розрахункової частини добре

Оцінка графічної (презентаційної) частини добре

Загальна оцінка добре

Прізвище, ім'я, по батькові керівника роботи Кільдішев Віталій Йосипович

Місце роботи і посада керівника роботи к.т.н., доцент кафедри кібербезпеки та технічного захисту інформації ДУІТЗ

«10» червня 2024 р. _____ _____

(підпис) (прізвище та ініціали керівника)

ВСП «ОДЕСЬКИЙ ТЕХНІЧНИЙ ФАХОВИЙ КОЛЕДЖ ОНТУ»

РЕЦЕНЗІЯ

на кваліфікаційну роботу бакалавра
відділення комп'ютерних систем

Ситнікова Андрія Сергійовича

(прізвище, ім'я та по батькові)

Напрямку підготовки *123 «Комп'ютерна інженерія»*

Керівник кваліфікаційної роботи

Кільдішев Віталій Йосипович

(прізвище, ім'я та по батькові)

Тема кваліфікаційної роботи

«Дослідження системи потокового шифрування даних на основі мови програмування Python»

Обсяг пояснювальної записки 67 сторінок

Обсяг графічної (презентаційної) частини проекту 20 аркушів (слайдів)

ХАРАКТЕРИСТИКА КВАЛІФІКАЦІЙНОЇ РОБОТИ

а) заключення про ступінь відповідності виконаної роботи завданню

Робота відповідає технічному завданню до дипломного проекту. Виконана у відповідності з вимогами.

б) характеристика виконання кожного розділу роботи

При виконанні дипломного проекту студент продемонстрував уміння використовувати останні досягнення науки та техніки, уміння працювати з літературою. Так, студент грамотно дослідив та проаналізував методів та засобів потокового шифрування.

в) оцінка якості виконання графічної (презентаційної) частини роботи і пояснювальної записки

Графічна частина відповідає вимогам, виконана якісно та відображає основні елементи проектування системи. Розглянуто роль і значення захисту інформації у сфері кібербезпеки, з акцентом на важливість шифрування даних. Схематично представлено взаємодію алгоритмів шифрування з потенційними кіберзагрозами та механізмами захисту даних. Розроблено основні методики запобігання несанкціонованому доступу до інформації за допомогою потокового шифрування на основі Python.

г) перелік позитивних якостей роботи

Тема дипломного проекту є актуальною, виконана у достатньому обсязі, відповідно до поставленого завдання

д) основні недоліки роботи

1. Пояснювальна записка недостатньо структурована, етапи розробки недостатньо обґрунтовані.
2. Блок-схеми алгоритмів не відображують суть роботи потокових криптосистем
3. Наявні помилки оформлення пояснювальної записки

Оцінка розрахункової частини Добре

Оцінка графічної (презентаційної) частини Добре

Загальна оцінка Добре

Прізвище, ім'я та по батькові рецензента к.т.н. Селіванова Алла Віталіївна

Місце роботи і посада рецензента Одеський національний технологічний університет, декан факультету комп'ютерної інженерії, програмування та кіберзахисту

«26» 06 2024 р.

(підпис)

Селіванова А.В.

(прізвище та ініціали рецензента)

UNICHECK
by Turnitin

Ім'я користувача:
Катерина Григоріївна Краснокутська

ID перевірки:
1016348393

Дата перевірки:
11.06.2024 17:27:56 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
11.06.2024 17:36:37 EEST

ID користувача:
100011688

Назва документа: **2БКС-28_Андрій_Ситніков**

Кількість сторінок: **44** Кількість слів: **6287** Кількість символів: **46726** Розмір файлу: **253.66 KB** ID файлу: **1016151515**

1.5%
Схожість

Найбільша схожість: **0.38%** з Інтернет-джерелом (https://research.nccgroup.com/2023/08/22/dancing-offbit-the-story-of...

1.5% Джерела з Інтернету	154	Сторінка 46

Не знайдено джерел з Бібліотеки

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	8

ДОЗВІЛ
НА РОЗМІЩЕННЯ
ВИПУСКНОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ
В ЕЛЕКТРОННОМУ РЕПОЗИТАРІЇ ВСП «ОТФК ОНТУ»

Ми, що нижче підписалися,

Ситніков Андрій Сергійович
здобувач освіти гр. 2БКС-28, та

Кільдішев Віталій Йосипович,
керівник випускної кваліфікаційної роботи,

не заперечуємо щодо розміщення електронного варіанту пояснювальної записки до випускної кваліфікаційної роботи бакалавра на тему:

«Дослідження системи потокового шифрування даних на основі мови програмування Python» (автор роботи – Ситніков А.С., керівник роботи – Кільдішев В.Й.)

виконаного у ВСП «Одеський технічний фаховий коледж Одеського національного технологічного університету» в 2024 році, у повному обсязі в електронному репозитарії ВСП «ОТФК ОНТУ» для вільного доступу через мережу Інтернет.

Несемо відповідальність за ідентичність електронного та друкованого варіантів випускної кваліфікаційної роботи і даємо згоду на обробку персональних даних.

Виконавець _____ / Ситніков А.С./

Керівник _____ / Кільдішев В.Й./

«13» червня 2024 р.