МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ВОЛИНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ЛЕСІ УКРАЇНКИ
Кафедра комп'ютерних наук та кібербезпеки

МАТВІЙЧУК ОЛЕГ МИКОЛАЙОВИЧ

ПРОГРАМНІ РЕАЛІЗАЦІЇ ШИФРУВАННЯ, ДЕШИФРУВАННЯ ТА ЗЛАМ ПІДСТАНОВОЧНИХ ШИФРІВ МОВОЮ PYTHON

Спеціальність: 122 Комп'ютерні науки

Освітньо-професійна програма: Комп'ютерні науки та інформаційні технології

Робота на здобуття освітнього ступеня «бакалавр»

Науковий керівник:
ГОЛОВІН МИКОЛА БОРИСОВИЧ
кандидат фізико-математичних наук, доцент

РЕКОМЕНДОВАНО ДО ЗАХИСТУ
Протокол № _____,
засідання кафедри комп'ютерних наук
та кібербезпеки
від _____ 20___р.
Завідувач кафедри
(_____)_____
(підпис) ПІБ

ЛУЦЬК – 2024

ЗМІСТ

ВСТУП 4

РОЗДІЛ 1. ТЕОРЕТИЧНІ ЗАСАДИ ДОСЛІДЖЕННЯ ПРОСТОГО ПІДСТАНОВОЧНОГО ШИФРА МОВОЮ PYTHON 6

1.1. Типи шифрів підстановки. Алгоритми їх роботи. 6

1.1.1. Шифри простої заміни. 8

1.1.2. Атбаш. 9

1.1.3. Шифр Цезаря, Гронсфельда. 11

1.1.4. Шифр із використанням кодового слова. 12

1.2. Метод запису зашифрованих текстів. 13

1.3. Безпека шифрів простої заміни. 14

1.4. Омофонічна заміна та приклади. 15

1.4.1. Номенклатор. 17

1.4.2. Великий Шифр Россіньоля. 18

1.4.3. Книжковий шифр. 18

1.5. Поліалфавітні шифри. 19

1.5.1. Шифр Віженера. 20

1.5.2. Одноразовий блокнот. 20

1.6. Поліграмні шифри. 21

1.6.1. Шифр Плейфера. 22

1.6.2. Шифр Хілла. 23

1.7. Застосування шифрування та дешифрування методами підстановки в шифрувальних машинах. 24

1.8. Сучасні використання шифрування методами підстановки. 26

РОЗДІЛ 2. РЕАЛІЗАЦІЯ ШИФРУВАННЯ ТА ДЕШИФРУВАННЯ МЕТОДАМИ ПІДСТАНОВКИ МОВОЮ PYTHON. 28
2.1. Шифр Цезаря. 35
2.2. Шифр Гронсфельда. 37
2.3. Шифр Віженера. 38
2.4. Омофонічна заміна 41
2.5. Книжковий шифр. 43
2.6. Атаки на основі шифротексту. 45
2.7. Атаки на основі відкритих текстів. 47
2.8. Атаки на основі підібраного відкритого тексту. 49
ВИСНОВКИ 53
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 56
ДОДАТКИ 59

ВСТУП

Актуальність дослідження. У сучасному світі з розвитком цифрових технологій питання захисту конфіденційних даних набувають особливої значущості, і шифрування є одним із найефективніших методів забезпечення безпеки інформації. В умовах зростання обсягу цифрових комунікацій та обміну даними між користувачами, урядами та комерційними організаціями, збереження конфіденційності та цілісності інформації стає критично важливим завданням. Підстановочні шифри, як один із видів класичних криптографічних методів, продовжують знаходити застосування в сучасних системах захисту інформації.

Суть підстановочних шифрів полягає в заміні кожного символу відкритого тексту на інший символ за певним правилом або ключем. Такий метод шифрування дозволяє перетворити зрозумілий текст у видозмінену форму, яку важко розшифрувати без знання ключа. Сучасні технології дозволяють автоматизувати процеси шифрування та дешифрування, що сприяє підвищенню ефективності та надійності криптографічних систем.

Застосування підстановочних шифрів є важливим кроком у забезпеченні безпеки інформації в різних сферах, включаючи банківську справу, електронну комерцію, державне управління та особисте листування. Підстановочні шифри формують основу для побудови складних і надійних систем захисту даних, які можуть протистояти сучасним загрозам кібербезпеки.

Мета полягає в розробці та аналізі програмних реалізацій шифрування, дешифрування та зламу підстановочних шифрів мовою Python, а також у вивченні ефективності та можливостей цих реалізацій.

Об’єктом дослідження є процеси шифрування, дешифрування та зламу підстановочних шифрів.

Предметом дослідження є програмні реалізації підстановочних шифрів мовою Python.

Завдання:

1. Розглянути теоретичні основи підстановочних шифрів.
2. Розробити програмні реалізації шифрування та дешифрування підстановочних шифрів мовою Python.
3. Дослідити методи зламу підстановочних шифрів та реалізувати їх на Python.
4. Провести аналіз ефективності реалізованих програм.

Методи дослідження. У роботі використовуються методи моделювання, програмування та експериментального дослідження. Для реалізації програмних модулів використовується мова програмування Python, а також бібліотеки цієї мови.

Структура роботи. Робота складається з вступу, теоретичної частини, в якій розглядаються основні принципи підстановочних шифрів, практичної частини, де представлені програмні реалізації шифрування, дешифрування та зламу підстановочних шифрів на Python, аналізу результатів та висновків.

РОЗДІЛ 1. ТЕОРЕТИЧНІ ЗАСАДИ ДОСЛІДЖЕННЯ ПРОСТОГО ПІДСТАНОВОЧНОГО ШИФРА МОВОЮ PYTHON

1.1.Типи шифрів підстановки. Алгоритми їх роботи.

Шифри підстановки є одними з найстаріших методів шифрування, що використовуються для забезпечення конфіденційності інформації. Вони працюють шляхом заміни кожного символу вихідного тексту на інший символ за певним правилом або ключем. Існує кілька типів підстановочних шифрів, кожен з яких має свої особливості та методи шифрування і дешифрування. У цьому розділі ми розглянемо основні типи шифрів підстановки, включаючи моноалфавітні, поліалфавітні та гомофонні шифри, а також алгоритми їх роботи.

Моноалфавітні шифри використовують один фіксований алфавіт підстановки для заміни символів відкритого тексту. Це найпростіші з підстановочних шифрів, які, однак, мають свої недоліки, такі як вразливість до частотного аналізу. Прикладами моноалфавітних шифрів є шифр Цезаря та шифр простої підстановки.

Шифр Цезаря здійснює зсув кожної літери відкритого тексту на певну кількість позицій в алфавіті. Це дуже простий і легко реалізований метод шифрування. Для шифрування спочатку вибирають значення зсуву (ключ). Потім для кожної літери в тексті визначається її порядковий номер в алфавіті, до якого додається значення зсуву, і замінюється початкова літера на літеру, що відповідає новому порядковому номеру.

Шифр простої підстановки замінює кожну літеру відкритого тексту на іншу літеру за визначеним ключем підстановки. Генерується ключ підстановки, який є перестановкою алфавіту. Для шифрування кожної літери тексту знаходять відповідну літеру з ключа підстановки і замінюють початкову літеру на знайдену.

Поліалфавітні шифри використовують кілька алфавітів підстановки, які змінюються під час шифрування. Це складніші методи, які забезпечують вищий рівень безпеки. Найвідомішим прикладом поліалфавітного шифру є шифр

Віженера. Він використовує ключове слово для зміни зсуву кожної літери. Кожна літера ключового слова визначає значення зсуву для відповідної літери відкритого тексту. Процес шифрування повторюється для всього тексту, використовуючи ключове слово циклічно. Це ускладнює злам шифру методом частотного аналізу, оскільки одна і та ж літера в різних позиціях тексту може бути зашифрована по-різному.

Гомофонні шифри використовують кілька символів підстановки для однієї літери відкритого тексту. Це ще більше ускладнює частотний аналіз, оскільки одна і та ж літера може бути замінена різними символами. Для кожної літери визначається множина символів підстановки. При шифруванні випадково вибирається один з можливих символів підстановки для кожної літери тексту і замінюється початкова літера на вибраний символ. Це значно ускладнює дешифрування для атакуючих, оскільки частотний аналіз стає набагато менш ефективним.

Алгоритми шифрування підстановочних шифрів включають кілька основних кроків: генерацію ключа, шифрування тексту та дешифрування тексту. Генерація ключа може бути різною залежно від типу шифру: від простого числа для зсуву до складних перестановок або ключових слів. Процес шифрування включає заміну кожної літери відкритого тексту на відповідну літеру або символ за ключем підстановки. Дешифрування є зворотнім процесом, у якому кожен символ зашифрованого тексту замінюється на відповідну літеру відкритого тексту за тим самим ключем[1].

Підстановочні шифри є важливим аспектом криптографії, які заклали основу для багатьох сучасних методів шифрування. Вони демонструють базові принципи, які можуть бути розширені та вдосконалені для досягнення вищого рівня безпеки. Використання мови програмування Python дозволяє легко реалізувати ці шифри, експериментувати з ними та поглиблювати розуміння криптографічних процесів.

1.1.1. Шифри простої заміни.

Шифри простої заміни є одним з найстаріших і найпростіших методів шифрування текстів. Вони полягають у заміні кожного символу відкритого тексту на інший символ відповідно до певного фіксованого правила або ключа.

Характеристики шифрів простої заміни

1. Симетричність: Один і той самий ключ використовується як для шифрування, так і для розшифрування повідомлень.
2. Фіксована заміна: Кожен символ відкритого тексту завжди замінюється одним і тим самим символом шифрованого тексту.
3. Однозначність: Кожен символ відкритого тексту має однозначне відповідне значення у шифрованому тексті.

Типи шифрів простої заміни

1. Моноалфавітні шифри: Використовується один алфавіт для всієї заміни. Ключем може бути перестановка алфавіту або фіксований зсув.
2. Поліалфавітні шифри: Використовується кілька алфавітів для заміни. Ключ може бути довшим за повідомлення і змінюватися під час шифрування.

Криптографічна стійкість:

1. Низька стійкість: Легко піддаються криптоаналізу, особливо за допомогою частотного аналізу, оскільки частота появи символів у зашифрованому тексті залишається такою самою, як і в оригінальному тексті.
2. Частотний аналіз: Аналіз частоти символів дозволяє виявити шаблони та відповідності між символами відкритого та зашифрованого тексту.
3. Біграмний і триграмний аналіз: Дослідження частот пар (біграм) і трійок (триграм) символів у тексті для ідентифікації більш складних шаблонів.

Переваги:

1. Простота реалізації: Легко реалізуються за допомогою ручних методів або простих програм.
2. Швидкість шифрування та розшифрування: Операції заміни символів виконуються дуже швидко.

Недоліки:

1. Низький рівень безпеки: Вразливість до частотного аналізу робить ці шифри непридатними для захисту конфіденційної інформації.

2. Фіксованість заміни: Використання одного й того самого ключа для всього тексту робить шифр вразливим до дешифрування.

Використання:

1. Історичне: Використовувалися в минулому для шифрування військових і дипломатичних повідомлень.

2. Освітнє: Вивчаються для розуміння базових принципів криптографії.

3. Розважальне: Використовуються в головоломках, іграх та для створення шифрів у літературі та кіно.

1.1.2. Атбаш.

Шифр Атбаш є одним з найдавніших і найпростіших методів шифрування, який належить до категорії моноалфавітних шифрів заміни. Цей шифр відомий своєю простотою та використанням симетричної перестановки алфавіту для заміни символів[2].

Основні характеристики:

1. Принцип дії: Шифр Атбаш базується на перестановці алфавіту, де перша літера алфавіту замінюється на останню, друга – на передостанню і так далі. Таким чином, алфавіт розвертається у зворотному порядку.

2. Моноалфавітний шифр: Використовується єдиний фіксований алфавіт для всієї заміни, що означає, що кожна літера відкритого тексту завжди замінюється на ту саму літеру шифрованого тексту.

3. Симетричність: Шифрування і дешифрування використовують однаковий процес, оскільки перестановка алфавіту однакова для обох операцій. Тобто, алгоритм шифрування і розшифрування є ідентичним.

4. Фіксованість заміни: Для кожної літери відкритого тексту завжди є одне і те саме відповідне значення у шифрованому тексті. Це робить шифр Атбаш вразливим до частотного аналізу.

Історичний контекст:

1. Давнє походження: Шифр Атбаш використовувався у давні часи, зокрема, у єврейських текстах. Назва "Атбаш" походить від перших двох і останніх двох літер єврейського алфавіту (א - ת, ב - ש), які ілюструють принцип заміни.

2. Простота реалізації: Завдяки своїй простоті, шифр Атбаш широко використовувався для шифрування текстів без необхідності складних обчислень або інструментів.

Криптографічна стійкість:

1. Низький рівень безпеки: Шифр Атбаш легко піддається криптоаналізу, особливо частотному аналізу, оскільки заміна є фіксованою і однозначною для кожної літери.

2. Частотний аналіз: Оскільки частота появи літер у шифрованому тексті не змінюється, криптоаналітик може легко виявити відповідності між літерами відкритого та шифрованого текстів, використовуючи статистичні дані про частоту появи літер у мові.

Використання:

1. Освітня цінність: Шифр Атбаш часто використовується у навчальних матеріалах з криптографії для ілюстрації основних принципів заміни символів.

2. Культурне значення: Шифр Атбаш згадується у різних літературних та релігійних текстах, що додає йому культурного значення.

1.1.3. Шифр Цезаря, Гронсфельда.

Шифр Цезаря

Шифр Цезаря є одним із найстаріших та найвідоміших методів шифрування. Його основні характеристики:

1. Моноалфавітний шифр: Використовується один алфавіт, де кожен символ відкритого тексту замінюється на інший символ за фіксованим правилом зсуву.

2. Принцип дії: Кожен символ відкритого тексту зміщується на певну кількість позицій вправо або вліво в алфавіті. Кількість позицій зсуву є ключем шифру.

3. Симетричність: Один і той самий ключ (кількість позицій зсуву) використовується як для шифрування, так і для дешифрування.

4. Простота реалізації: Легко реалізується як вручну, так і програмно завдяки простій логіці зсуву.

5. Низька криптографічна стійкість: Легко піддається криптоаналізу, особливо частотному аналізу, оскільки зсув є фіксованим для кожного символу. Простота зсуву робить цей шифр вразливим до грубої сили (перебору всіх можливих зсувів).

6. Історичне використання: Названий на честь Юлія Цезаря, який використовував цей метод для шифрування військових повідомлень.

Шифр Гронсфельда

Шифр Гронсфельда є варіантом шифру Віженера з деякими особливостями. Його основні характеристики:

1. Поліалфавітний шифр: Використовує кілька алфавітів для заміни символів відкритого тексту, що робить його стійкішим до частотного аналізу порівняно з моноалфавітними шифрами.

2. Принцип дії: Заміна символів відкритого тексту виконується за допомогою числового ключа. Кожна цифра ключа визначає зсув для відповідної літери відкритого тексту.

3. Ключ: Ключ складається з цифр, зазвичай 0-9, що спрощує його запам'ятовування та використання. Ключ повторюється для кожної літери тексту, якщо він коротший за сам текст.

4. Симетричність: Той самий числовий ключ використовується як для шифрування, так і для дешифрування.

5. Складність криптоаналізу: Складніший для дешифрування порівняно з моноалфавітними шифрами, оскільки кожна літера може мати різний зсув відповідно до цифри ключа. Однак, він все ще вразливий до криптоаналізу, особливо якщо ключ короткий або повторюється.

6. Історичне використання: Отримав своє ім'я від німецького винахідника Жана Гронсфельда і використовувався для більш безпечного шифрування повідомлень у різних історичних контекстах.

1.1.4. Шифр із використанням кодового слова.

Шифр із використанням кодового слова є різновидом поліалфавітного шифру, який використовує ключове слово для визначення правил заміни символів. Його основні характеристики:

1. Поліалфавітний шифр: Використовується кілька алфавітів для заміни символів відкритого тексту, що робить його складнішим для дешифрування за допомогою частотного аналізу порівняно з моноалфавітними шифрами.

2. Принцип дії: Кожна літера відкритого тексту замінюється на основі ключового слова. Ключове слово визначає послідовність зсувів або підстановок, які застосовуються до символів відкритого тексту.

3. Ключове слово: Вибране слово або фраза, яка повторюється або розширюється, щоб охопити весь текст, що шифрується. Це ключове слово керує заміною символів відкритого тексту.

4. Зміна алфавітів: Для кожної літери відкритого тексту використовується різний алфавіт або зсув, визначений відповідною літерою ключового слова. Це створює складний і менш регулярний шаблон заміни.

5. Симетричність: Той самий ключ (кодове слово) використовується як для шифрування, так і для дешифрування тексту. Тобто, знання ключа дозволяє відновити оригінальний текст із зашифрованого.

6. Захист від частотного аналізу: Завдяки використанню різних алфавітів для кожної літери тексту, шифр із використанням кодового слова значно складніший для дешифрування за допомогою простого частотного аналізу. Це збільшує криптографічну стійкість порівняно з моноалфавітними шифрами.

7. Криптоаналітичні методи: Незважаючи на підвищену стійкість до частотного аналізу, шифр із використанням кодового слова все ще може бути вразливим до більш складних методів криптоаналізу, таких як аналіз повторюваних сегментів або знання структури ключового слова.

8. Історичне використання: Широко використовувався в історії для шифрування військових, дипломатичних і приватних повідомлень. Відомий як один із найпоширеніших методів шифрування до появи сучасних криптографічних алгоритмів.

1.2.Метод запису зашифрованих текстів.

Метод запису зашифрованих текстів охоплює різні техніки та стратегії для організації, зберігання і передачі зашифрованої інформації. Цей метод включає в себе не тільки процес шифрування, але й специфічні способи обробки і зберігання зашифрованих даних для забезпечення їхньої безпеки та ефективності передачі.

Організація зашифрованого тексту може включати його поділ на блоки, рядки або колонки для спрощення передачі та зберігання. Це допомагає уникнути виявлення шаблонів у тексті, що могло б полегшити його дешифрування зловмисниками. Іноді важливо зберігати форматування оригінального тексту, наприклад, відступи, розділові знаки або пробіли, щоб забезпечити правильне дешифрування або передачу тексту через канали, які вимагають певного форматування[6].

Керування ключами є критично важливою частиною методу запису зашифрованих текстів. Безпечне зберігання і передача ключів шифрування

забезпечують, що тільки уповноважені особи можуть отримати доступ до зашифрованої інформації. Метод запису зашифрованих текстів часто інтегрується з іншими технологіями безпеки, такими як цифрові підписи, хешування або захищені протоколи передачі даних, що підвищує загальну безпеку системи.

Захист від модифікацій зашифрованого тексту є ще одним важливим аспектом. Використання методів виявлення модифікацій, таких як контрольні суми або криптографічні хеш-функції, запобігає несанкціонованим змінам даних. Крім того, ефективний метод запису зашифрованих текстів враховує можливі криптографічні атаки, такі як аналіз трафіку, підміна повідомлень або атаки на основі відомих відкритих текстів. Використання складних схем шифрування та регулярне оновлення ключів допомагають мінімізувати ці ризики[6].

Зручність використання також є важливим фактором. Метод повинен бути простим у використанні, включаючи легкість шифрування та дешифрування текстів, а також можливість його використання на різних платформах і пристроях. Важливо також, щоб метод запису зашифрованих текстів відповідав нормативним вимогам і стандартам, що регулюють безпеку даних у різних галузях, таких як фінанси, медицина чи державні установи.

Метод запису зашифрованих текстів є комплексним процесом, що включає шифрування, організацію, передачу та зберігання зашифрованої інформації. Ефективне використання цього методу забезпечує конфіденційність, цілісність і доступність даних, захищаючи їх від несанкціонованого доступу та модифікацій.

1.3.Безпека шифрів простої заміни.

Безпека шифрів простої заміни визначається їхньою здатністю протистояти криптоаналізу, забезпечуючи конфіденційність зашифрованої інформації. Шифри простої заміни є одними з найстаріших методів шифрування, що полягають у заміні кожного символу відкритого тексту на інший символ за фіксованим правилом. Однак, їх криптографічна стійкість має низку обмежень.

Основна вразливість шифрів простої заміни полягає у фіксованій заміні кожної літери, що дозволяє криптоаналітикам використовувати частотний аналіз для розшифрування тексту. Частотний аналіз базується на статистичних характеристиках мови, таких як частота появи певних літер або біграм. Завдяки цьому аналізу, криптоаналітики можуть зіставити частоти літер у зашифрованому тексті з відомими частотами літер у відкритому тексті, що дозволяє визначити відповідності між символами.

Крім того, шифри простої заміни схильні до атак на основі відомих відкритих текстів. Якщо криптоаналітик має як зашифрований текст, так і його відповідний відкритий текст, він може легко визначити правило заміни і використати його для дешифрування інших повідомлень, зашифрованих тим самим ключем. Це робить шифри простої заміни вразливими до такого типу атак, особливо якщо ключ або алгоритм заміни відомий або легко вгадується.

Щоб підвищити безпеку шифрів простої заміни, можна використовувати кілька методів. Одним з них є збільшення довжини ключа або використання складніших схем заміни, що ускладнює криптоаналіз. Також можна вводити випадковість у процес шифрування, щоб зменшити регулярність і передбачуваність заміни символів. Використання поліалфавітних шифрів, де кожна літера може замінюватися різними символами залежно від контексту, також може підвищити стійкість до частотного аналізу[7].

Однак, навіть з цими удосконаленнями, шифри простої заміни зазвичай вважаються недостатньо безпечними для сучасних застосувань, які потребують високого рівня криптографічного захисту. У сучасній криптографії частіше використовуються більш складні алгоритми, такі як симетричні блокові шифри і асиметричні криптосистеми, які забезпечують набагато вищий рівень безпеки.

1.4.Омофонічна заміна та приклади.

Омофонічна заміна, або шифр омофонів, є методом шифрування, в якому кожен символ відкритого тексту може бути замінений на один з кількох можливих символів у шифрованому тексті. Цей метод був розроблений для

підвищення криптографічної стійкості шляхом усунення можливості частотного аналізу, який є ефективним проти простих шифрів заміни, таких як шифр Цезаря.

У омофонічній заміні кожному символу відкритого тексту відповідає декілька можливих символів у шифрованому тексті. Вибір заміни здійснюється випадковим чином з набору можливих варіантів, що ускладнює частотний аналіз. Для шифрування та дешифрування використовується спеціальна таблиця, яка визначає всі можливі заміни для кожного символу.

Перевага омофонічної заміни полягає у високій криптографічній стійкості за рахунок усунення частотного аналізу, а випадковість вибору заміни ускладнює аналіз шифру. Однак, цей метод має недоліки, такі як складність управління ключами через необхідність використання складної таблиці замін і можливе збільшення розміру шифрованого тексту (*Рис.1.1)*. Омофонічна заміна є ефективним методом для підвищення криптографічної стійкості, особливо в історичних контекстах, де частотний аналіз був основним методом розшифрування.

```
A: [12, 34, 56]
B: [23, 45, 67]
C: [31, 42, 53]
D: [14, 25, 36]
```

Рисунок 1.1 - Таблиця замін

Шифрування

Відкритий текст: ABAC

1. A може бути зашифровано як 12, 34 або 56. Випадковий вибір: 34.
2. B може бути зашифровано як 23, 45 або 67. Випадковий вибір: 23.
3. A може бути зашифровано як 12, 34 або 56. Випадковий вибір: 56.
4. C може бути зашифровано як 31, 42 або 53. Випадковий вибір: 53.

Шифрований текст: 34 23 56 53

Дешифрування

Шифрований текст: 34 23 56 53

1. 34 розшифровується як A.
2. 23 розшифровується як B.
3. 56 розшифровується як A.
4. 53 розшифровується як C.

Відкритий текст: ABAC

1.4.1. Номенклатор.

Номенклатор - це криптографічний метод, який поєднує в собі елементи шифрування та кодування. В цьому методі використовуються спеціальні таблиці або книги, що містять числові або алфавітні коди для слів, фраз або окремих літер. Мета номенклатора полягає у створенні більш складного шифру, який важче піддається розшифруванню та аналізу[11].

Кожному слову або фразі у відкритому тексті присвоюється унікальний код, що може бути числовим або буквено-числовим. Для літер, які не мають відповідних кодів у таблиці, використовуються стандартні методи шифрування, такі як шифр заміни або шифр Цезаря. Це забезпечує додатковий рівень захисту, оскільки частотний аналіз, застосовуваний до простих шифрів, стає менш ефективним.

Номенклатори широко використовувалися у дипломатичному листуванні та військовій комунікації до винайдення більш сучасних шифрів. Вони забезпечували високий рівень безпеки за рахунок великої кількості можливих замін та використання спеціалізованих кодових книг, що ускладнювало процес дешифрування для несанкціонованих осіб.

Цей метод вимагає ретельного управління та захисту кодових книг, оскільки втрата або захоплення таких книг противником могло призвести до розкриття зашифрованих повідомлень. Сучасні номенклатори були замінені більш ефективними і автоматизованими методами шифрування, проте вони залишаються важливою частиною історії криптографії.

1.4.2. Великий Шифр Россіньоля.

Великий Шифр Россіньоля, створений Антуаном та Бонавентурою Россіньолем у XVII столітті, був надзвичайно складним криптографічним методом, який використовувався французьким двором. Його особливістю було те, що він замінював цілі склади або навіть цілі слова числовими кодами, а не окремі букви. Це робило шифр дуже стійким до криптоаналізу.

Шифр залишався нерозгаданим протягом декількох століть, поки не був розшифрований у XIX столітті криптографом Етьєном Базері. Базері виявив, що Великий Шифр використовував близько 587 різних числових кодів для позначення складів, літер та деяких популярних французьких слів. Така складність і велика кількість можливих комбінацій робили його одним із найбільш захищених шифрів свого часу.

Великий Шифр також включав додаткові методи захисту, такі як помилкові цифри та зміна значення цифр у різних контекстах, що додатково ускладнювало розшифрування. Цей шифр був настільки надійним, що після його створення він використовувався для захисту найважливіших державних та військових таємниць Франції.

1.4.3. Книжковий шифр.

Книжковий шифр — це метод шифрування, в якому текст шифрується за допомогою заздалегідь узгодженого тексту, наприклад, книги або іншого великого документа. Ідея полягає в тому, що як відправник, так і отримувач мають доступ до однієї й тієї ж книги або документа, який виступає ключем для шифрування та дешифрування.

Використання книжкового шифру ґрунтується на пошуку певних слів або фраз у тексті книги. Для шифрування використовується положення цих слів у книзі, наприклад, номер сторінки, рядок і позиція слова в рядку. Ця інформація потім передається у вигляді чисел або інших символів, які вказують на конкретні слова або літери в книзі[12].

Книжковий шифр має декілька переваг. По-перше, він забезпечує високий рівень безпеки, оскільки для розшифрування потрібен доступ до тієї ж книги.

По-друге, ключ може бути змінений відносно легко шляхом зміни книги або документа, що використовується як ключ. Однак цей метод також має недоліки. Основний з них полягає в тому, що обидві сторони повинні мати фізичний доступ до однієї й тієї ж книги, що може бути не завжди зручно або практично.

Книжковий шифр був популярний у минулому, особливо для персональної або військової кореспонденції, де безпека була критично важливою. В сучасних умовах він втратив свою актуальність через розвиток комп'ютерних технологій і більш ефективних методів шифрування.

1.5. Поліалфавітні шифри.

Поліалфавітні шифри є класом шифрів підстановки, де для шифрування повідомлення використовується кілька різних алфавітів. Це підвищує стійкість шифру, оскільки кожна літера відкритого тексту може бути зашифрована різними способами в залежності від її позиції в тексті та використовуваного ключа.

Основна ідея поліалфавітних шифрів полягає у зміні алфавіту для кожної літери відкритого тексту відповідно до певного правила або ключа. Це дозволяє уникнути регулярних шаблонів, які легко виявити при частотному аналізі, що є слабким місцем моноалфавітних шифрів, де кожна літера завжди шифрується одним і тим же способом[7].

Одним із найвідоміших прикладів поліалфавітного шифру є шифр Віженера. В цьому шифрі використовується ключове слово, де кожна літера ключа визначає зсув для відповідної літери відкритого тексту. Оскільки ключове слово повторюється по всьому тексту, кожна літера може бути зашифрована різними способами, залежно від позиції в тексті та ключа.

Поліалфавітні шифри значно підвищують криптографічну стійкість шифрованого тексту, оскільки вони ускладнюють використання частотного аналізу, але все ж можуть бути розкриті при використанні складніших криптоаналітичних методів. Вони є важливим кроком в історії криптографії, показуючи еволюцію від простих підстановочних шифрів до більш складних систем шифрування.

1.5.1. Шифр Віженера.

Шифр Віженера є поліалфавітним шифром, що використовується для шифрування текстових повідомлень. Основна ідея полягає в тому, що кожен символ тексту шифрується за допомогою іншого символу (зазвичай з одного і того ж алфавіту), використовуючи ключ, який складається з послідовності символів (зазвичай слова). Ключ використовується циклічно: якщо ключ коротший за текст, він повторюється доти, доки не зашифрується весь текст.

Шифр Віженера важливий тим, що він є поліалфавітним, що означає, що символи шифруються залежно від їх позиції у повідомленні, що ускладнює криптоаналіз. Наприклад, два однакових символи відкритого тексту, що знаходяться на різних позиціях, можуть бути зашифровані різними символами в шифрованому тексті, залежно від ключа і позиції.

1.5.2. Одноразовий блокнот.

Одноразовий блокнот (англ. One-time pad) є одним з найбільш безпечних методів симетричного шифрування. Основна ідея полягає в використанні ключа, який має таку саму довжину, як і шифрується повідомлення. Ключ генерується випадковим чином і використовується тільки один раз — для одного конкретного повідомлення. Ключ відомий тільки відправнику і одержувачу і не повинен бути повторно використаний або змінений.

Одноразовий блокнот гарантує абсолютну стійкість до криптоаналізу при відповідній умові, що ключ буде використаний тільки один раз і буде випадковим. Це важливо, оскільки будь-яке повторне використання ключа або його неправильне використання може призвести до порушення безпеки шифру.

Одноразовий блокнот використовується в деяких критичних застосунках, де важлива абсолютна безпека шифрування, наприклад, в дипломатичній та військовій зв'язку[21].

Одноразовий блокнот є ефективним з точки зору безпеки, але він також має деякі практичні обмеження. Основні з них включають великі вимоги до безпеки ключа, необхідність генерації і безпечного обміну великими обсягами випадкових даних і складнощі з управлінням ключами. Також важливо

враховувати, що кожен ключ може бути використаний лише один раз, що може бути не зручним в ситуаціях, де необхідне багаторазове шифрування повідомлень.

У зв'язку з цим, хоча одноразовий блокнот забезпечує найвищий рівень криптографічної стійкості, його застосування обмежене і вимагає уважного розгляду й виваженого підходу до його використання в конкретних сценаріях.

1.6.Поліграмні шифри.

Поліграмні шифри є категорією поліалфавітних шифрів, які шифрують пари або групи символів (поліграми) одночасно замість окремих символів. Це дозволяє їм ефективношифрувати паралельні відносини між символами тексту, що робить криптоаналіз більш складним. Вони використовують різноманітні методи заміни або перестановки поліграм, щоб приховати статистичні властивості мови і зберегти конфіденційність шифрованого тексту[9].

Поліграмні шифри використовуються для збереження структурної інформації тексту, такої як частота вживання біграм або триграм, що є типовими в мові. Шифрування поліграмами може включати заміну одних груп символів іншими за допомогою попередньо встановлених правил або матриць заміни. Такі методи підвищують стійкість до статистичного криптоаналізу, оскільки аналіз частоти окремих символів утруднюється.

Одними з відомих поліграмних шифрів є Playfair шифр та шифр Біґрам. Вони застосовуються в різних сферах, включаючи військовий зв'язок та дипломатію, де необхідно забезпечити високий рівень конфіденційності і захист від криптоаналізу[16].

Поліграмні шифри можуть використовувати різні підходи до шифрування, включаючи заміну біграм або триграм, перестановку їхніх позицій у тексті або комбінації обох методів. Ці методи можуть бути реалізовані через спеціальні матриці заміни, таблиці або інші структури даних, які визначають правила перетворення тексту. При використанні поліграмних шифрів важливо уникати патернів або структур у шифрованому тексті, що можуть використовуватись для криптоаналізу, ізраховуючи на статистичний аналіз.

Такі шифри залишаються актуальними у деяких контекстах, хоча в сучасних криптографічних застосуваннях частіше використовуються більш складні і алгоритмічно безпечні методи шифрування, такі як блочні або потокові шифри.

1.6.1. Шифр Плейфера.

Шифр Плейфера є поліграмним шифром, призначеним для шифрування пар символів (біграм). Основна його ідея полягає у використанні спеціальної таблиці (або матриці), яка визначає правила перетворення пар символів в шифрований текст. Така таблиця, часто називається таблицею Плейфера, містить всі букви алфавіту, зазвичай розміщені у квадратну матрицю.

Процес шифрування у Шифрі Плейфера виконується таким чином:

1. Кожна пара символів з відкритого тексту замінюється на відповідну пару символів з таблиці Плейфера.
2. Пара символів може бути замінена згідно з правилами, які залежать від позиції символів у таблиці (наприклад, на основі їхніх рядків і стовпців).
3. Якщо в тексті є непарна кількість символів, останній символ може бути оброблений окремо.

Шифр Плейфера використовується для забезпечення конфіденційності тексту та утруднення криптоаналізу за рахунок заміни пар символів. Він був популярний в армійській та дипломатичній комунікації в минулому, але сучасні криптографічні методи зазвичай використовують більш сильні алгоритми.

Шифр Плейфера базується на квадратній таблиці, зазвичай розміром 5x5, заповненій алфавітними символами без повторів. Для створення таблиці використовуються ключові слова, які визначають порядок символів у перших рядках таблиці. Відкритий текст розбивається на пари символів (біграми), які потім замінюються на відповідні біграми з таблиці Плейфера[29].

1.6.2. Шифр Хілла.

Шифр Хілла є алгоритмічним методом шифрування, що використовує матриці для шифрування і дешифрування тексту. Основна ідея полягає у тому, що кожну групу символів (зазвичай біграму або триграму) представляють у вигляді числового вектора. Ці вектори потім перемножаються на ключову матрицю (або її обернену для дешифрування), і результат замінює вихідні вектори, щоб утворити зашифрований текст.

Основні характеристики Шифру Хілла включають:

1. Ключова матриця: Вона визначається як квадратна матриця з цілими числами або елементами поля (зазвичай чисел за модулем, наприклад, у полі за модулем 26 для англійської абетки).

2. Розмір матриці: Розмірність ключової матриці визначає кількість символів, які можна одночасно зашифрувати. Наприклад, у випадку біграм, якщо матриця 2x2, то кожні два символи (біграма) відкритого тексту замінюються на відповідні біграми зашифрованого тексту.

3. Інверсна матриця: Для дешифрування необхідно мати інверсну ключову матрицю, що дозволяє відновити вихідний текст з зашифрованого.

4. Стійкість до криптоаналізу: Шифр Хілла відповідно до своєї конструкції залежить від математичних властивостей ключових матриць, що робить його вразливим до атак, якщо матриця недостатньо складна або відома криптоаналітику[15].

Шифр Хілла був розроблений Лестером Хіллом у 1929 році і є одним з перших шифрів, що використовують математичні методи для шифрування тексту, використовуючи матричні операції.

Шифр Хілла використовується в криптографії для шифрування тексту, особливо коли потрібно шифрувати великі блоки даних, такі як фрази або повідомлення. Основні переваги включають високу ефективність при обробці великих обсягів даних і малу вразливість до статистичного криптоаналізу, що робить його бажаною вибором для захисту конфіденційної інформації.

Проте є деякі важливі обмеження Шифру Хілла. Наприклад, ефективність його застосування залежить від того, наскільки складні ключові матриці можуть бути згенеровані та збережені безпечним чином. Також важливо враховувати, що для ефективного використання Шифру Хілла необхідно, щоб розмір ключової матриці був взаємно простим з розміром алфавіту (наприклад, для англійської абетки розмір матриці може бути 2x2, 3x3 або 4x4).

У сучасному криптографічному середовищі Шифр Хілла частіше за все застосовується в контексті навчання та досліджень, оскільки сучасні методи шифрування, такі як AES (Advanced Encryption Standard), забезпечують вищий рівень безпеки і швидкодії.

1.7. Застосування шифрування та дешифрування методами підстановки в шифрувальних машинах.

Шифрувальні машини, що використовують методи підстановки для шифрування і дешифрування, були популярними у XX столітті, особливо в армійських та дипломатичних цілях. Одні з найвідоміших прикладів таких машин включають Enigma (використовувана Німеччиною під час Другої світової війни), Purple (використовувана Японією) та SIGABA (використовувана США)[12].

Процес шифрування методами підстановки:

1. Механізм підстановки: Кожен символ відкритого тексту (зазвичай буква або символ) замінюється на інший символ згідно з певним правилом або таблицею підстановки.

2. Таблиці підстановки: Це ключові елементи для шифрування. Вони можуть бути заздалегідь встановленими (як у випадку з Enigma, де ротори мали фіксовані настройки) або змінюватися (як у випадку з Purple, де настройки могли змінюватися регулярно).

3. Ключі і параметри настройки: Для ефективного шифрування важливо, щоб способи підстановки і їх параметри були відомі тільки відправникам і отримувачам повідомлень.

Процес дешифрування методами підстановки:

1. Обернена підстановка: Шифрований текст проходить обернену процедуру підстановки, де кожен символ замінюється на відповідний символ відкритого тексту за допомогою тієї ж самої таблиці підстановки.

2. Необхідність правильних параметрів: Правильні параметри підстановки (які використовувалися під час шифрування) є обов'язковими для коректного дешифрування.

Застосування в шифрувальних машинах:

Шифрувальні машини, що використовують методи підстановки, надають високий рівень безпеки, оскільки утруднюють криптоаналіз через заміну символів. Проте, їх безпека може бути компрометована, якщо зловмисник отримає доступ до ключових таблиць підстановки або змінить параметри машини.

У сучасний час шифрувальні машини на базі методів підстановки в основному замінилися більш складними алгоритмами, такими як блочні або потокові шифри, які забезпечують вищий рівень безпеки та операційної швидкодії.

Шифрувальні машини, які використовують методи підстановки, відрізнялися від інших шифрів своєю здатністю автоматизувати процес шифрування і дешифрування за допомогою механічних або електричних компонентів. Одним з найвідоміших прикладів є німецька шифрувальна машина Enigma, використовувана під час Другої світової війни. Enigma використовувала систему роторів і перетинань, що забезпечувала значну складність шифрування[17].

Застосування методів підстановки у шифрувальних машинах дозволяло замінювати кожен символ або групу символів на інші символи згідно з певними правилами, що зберігалися в таблицях підстановки. Це робило криптоаналіз шифрувальних машин складнішим, оскільки для розкодування потрібно було знати конкретні параметри підстановки, які могли змінюватися з часом або в залежності від налаштувань оператора.

Шифрувальні машини, що використовували методи підстановки, також вимагали фізичного обміну ключів або параметрів машини між відправниками і отримувачами повідомлень. Це робило їх вразливими до перехоплення чи крадіжки, що було однією з найбільших слабкостей цих систем у порівнянні з сучасними криптографічними протоколами, які забезпечують безпеку навіть у відсутність фізичного обміну ключами.

1.8.Сучасні використання шифрування методами підстановки.

Сучасні використання шифрування методами підстановки включають різні криптографічні протоколи та алгоритми, які використовують цей підхід для забезпечення конфіденційності даних. Ось деякі з них:

1. Шифр Хілла: Використовує квадратні матриці для заміни біграм або триграм в тексті. Цей метод залишається популярним у навчальних і дослідницьких цілях, а також для захисту невеликих обсягів даних.

2. Комп'ютерні програми та ігри: В деяких комп'ютерних програмах і іграх використовуються прості методи підстановки для шифрування тексту, файлів або комунікацій між гравцями.

3. Деякі алгоритми стеганографії: В стеганографії (науці про приховування інформації), методи підстановки можуть використовуватися для вбудовування прихованого тексту в інші носії даних, забезпечуючи конфіденційність.

4. Апаратне шифрування: У деяких випадках, апаратне шифрування може використовувати спеціалізовані пристрої для застосування методів підстановки в реальному часі, що забезпечує швидке і ефективне шифрування даних.

5. Шифрування в інтернет-протоколах: Деякі протоколи, такі як IPsec (Internet Protocol Security), можуть використовувати методи підстановки для шифрування даних, що передаються через мережу Інтернет. Вони забезпечують захист від несанкціонованого доступу і забезпечують конфіденційність.

6. Шифрування в телекомунікаціях: Деякі системи телекомунікацій використовують методи підстановки для шифрування голосових і текстових

даних, що передаються по каналах зв'язку. Це забезпечує конфіденційність розмов та повідомлень.

7. Шифрування в системах зберігання даних: Деякі системи зберігання даних використовують методи підстановки для шифрування даних, які зберігаються на серверах або в хмарних сервісах. Це дозволяє захищати інформацію від несанкціонованого доступу.

8. Застосування в апаратному забезпеченні: У низці сучасних апаратних засобах, таких як USB флешки або криптографічні модулі, можуть використовуватися методи підстановки для забезпечення безпеки даних, що зберігаються або передаються.

Ці приклади показують, що методи підстановки, як частину криптографічних протоколів і алгоритмів, залишаються важливими для захисту інформації в різних сучасних технологічних реаліях. Вони дозволяють ефективно забезпечувати конфіденційність та інтегруватися в різноманітні системи забезпечення інформації[13].

РОЗДІЛ 2. РЕАЛІЗАЦІЯ ШИФРУВАННЯ ТА ДЕШИФРУВАННЯ МЕТОДАМИ ПІДСТАНОВКИ МОВОЮ PYTHON.

Функції в Python практично нічим не відрізняються від функцій в інших мовах програмування. Функцією називають іменований фрагмент програмного коду, до якого можна звернутися з іншого місця програми скільки завгодно разів [7] за її іменем.

Для візуалізації функціоналу підсистеми та легкого використання функцій користувачами необхідно розробити інтерфейс. В Python, це можна реалізувати за допомогою PyQt та Qt Designer.

PyQt - це набір «прив'язок» графічного фреймворка Qt для мови програмування Python, виконаний у вигляді розширення Python.

PyQt практично повністю реалізує можливості Qt. PyQt також включає в себе Qt Designer (Qt Creator) - дизайнер графічного інтерфейсу користувача. Програма pyuic генерує Python код з файлів, створених в Qt Designer. Це робить PyQt дуже корисним інструментом для швидкого прототипування. Крім того, можна додавати нові графічні елементи управління, написані на Python, в Qt Designer.

Останьє, що треба обрати для реалізації підсистеми аутентифікації – це база даних. Для даної підсистеми було обрано MySQL. MySQL – це всім відома реалізація системи управління базами даних. Вона проста в установці, працює нормально без особливих налаштувань. При належному підході може гнучко налаштовуватися під будь-які потреби. Також MySQL може підтримувати роботу БД значних розмірів[22].

MySQL працює на різних платформах та легко переноситься з однієї платформи на іншу. MySQL має систему контроля доступу до даних, забезпечує шифрування даних.

Єдиним виключенням, в Python, є *lambda*-функції, у яких немає імені. Тому в загальному можна зазначити, що функція - це така частина коду, яка ізольована від решти програми і виконуються тільки тоді, коли викликається. Раніше

йшлося про вбудовані функції мови Python: *sqrt(), len(), print()* та ін. Як можна було бачити, функції можуть приймати аргументи (нуль, один або кілька), і можуть повертати деякий результат (значення). Наприклад, функція *sqrt()* приймає один аргумент і повертає значення (корінь числа). Функція *print()* приймає змінне число аргументів і нічого не повертає.

Функція складається з заголовку та тіла функції. Заголовок функції відповідає за її опис, тобто іменування та вказання необхідних параметрів. Тіло ж функції відповідає за самі дії, які мають виконатися при виклику функції. Заголовок та тіло функції оформлюються, як основна та вкладена інструкції.

Заголовок функції починається зі службового слова *def*, після якого вказується ім'я функції та список її формальних параметрів у круглих дужках. Завершується заголовок функції символом двокрапка «:». З наступного рядка слідує тіло функції, яке може містити довільну кількість операторів, що записуються з однаковим відступом на початку рядків по відношенню до заголовку функції.

Формат опису функції:

def ім'я_функції(список формальних параметрів):

оператори тіла функції

Ім'я_функції – ідентифікатор, що формується за правилами створення ідентифікаторів і в подальшому буде використовуватися для виклику функції.

Формальні параметри – це назви змінних, що вказуються при описі функції і через які будуть передаватися дані з основної програми в функцію. Значення для формальних параметрів будуть вказуватися при виклику функції.

Список формальних параметрів записується у вигляді перерахованих через кому імен змінних. Як було зазначено раніше, функція може і не мати формальних параметрів.

Найпростіша функція, за якою виводиться привітання, буде мати наступний вигляд:

def func(name):

print('Hello',name) 129

У випадку, якщо тіло функції має єдиний оператор, як у нашому випадку, то функцію можна записати одним рядком:

def funcs(name): print('Hello', name)

Як можна бачити, ця функція має єдиний параметр name, який використовується як частина повідомлення для виведення на екран.

Для виклику функції вказується її ім’я з указаним в дужках списком фактичних параметрів.

Формат оператора виклику функції:

ім’я_функції([список фактичних параметрів]);

Виклик раніше описаної функції матиме вигляд:

>>> func('World')

Hello World

Фактичні параметри – це назви змінних чи конкретні значення, що вказуються при виклику функції.

Список фактичних параметрів записується у вигляді перерахованих через кому фактичних параметрів.

Формальні параметри інколи називають параметрами функції, фактичні параметри інколи називають аргументами функції. Відповідні фактичні та формальні параметри не обов’язково повинні мати однакові імена.

В загальному кількість фактичних параметрів повинна бути такою самою, як і кількість формальних параметрів. Це необхідно тому, що при виклику функції встановлюється взаємно однозначна відповідність між фактичними та формальними параметрами. Інколи ще говорять, що співставлення параметрів є позиційним, тобто значення фактичних параметрів присвоюються формальним параметрам відповідно до їх позиції. Проте в мові Python є особливості співставлення цих параметрів, які будуть розглянуті пізніше[24].

Наявність параметрів у функціях надає можливість програмісту робити функції більш гнучкими та універсальними. Параметри виступають вхідними чи допоміжними даними, які будуть використані в обчисленнях, передбачених функцією.

Для того, щоб функцією було повернуте певне значення, її тіло повинно містити інструкцію *return*. Інструкція return являє собою 130 службове слово *return*, після якого вказується значення чи змінна, значення якої потрібно повернути за функцією.

Наприклад, функція, за якою буде повернуто одиницю:

def one(): return 1

Якщо за функцією не передбачено повернення ніяких значень, функція може викликатися у головній програмі чи іншій функції як окремий оператор: *func('World')*. Якщо за функцією передбачене повернення певного значення, то функція може викликатися у головній програмі чи іншій функції як операнд виразу:

two= one()+one().

Проте варто зауважити, що у випадку відсутності в тілі функції інструкції *return*, тобто не передбачення за функцією повернення ніякого значення, за функцією все рівно буде повернуте значення *None*.

```
>>> def func(): print('Hello!')
>>> print(func())
Hello!
None
```

Інколи виникає необхідність повернення за функцією не одного значення, а двох чи більше значень. Для цього потрібно в інструкції return вказати список або кортеж з бажаної кількості значень:

```
def f1(a, b): return [a**2, b**2]
def f2(a, b): return (a**2, b**2)
```

Тоді виклик функції можна записати:

```
n1, m1 = f1(2, 3)
n2, m2 = f2(2, 3)
s1 = f1(2, 3)
s2 = f2(2, 3)
```

В результаті виклику і виконання функцій отримаємо: *n1* та *n2* будуть містити значення *4*, *m1* та *m2* будуть містити значення *9*, *s1* буде містити список *[4, 9]*, *s2* буде містити кортеж *(4, 9)*.

В загальному, тіло функції може містити безліч інструкцій return, повернення значення за функцією буде визначатися першим викликом інструкції *return*.

Підключити модуль до програми можна за допомогою зарезервованого слова import:

```
import math #підключаємо до програми стандартний модуль math
#тепер можна користуватись функціями з даного модуля
#наприклад визвати функцію обчислення квадратного кореня
math.sqrt #зверніть увагу комбіноване ім'я – модуль.функція
```

Можна підключати не весь модуль а окремі функції або змінні з модуля:

```
#імпортуємо математичні функції корінь квадратний, синус та косинус
from math import sqrt, cos, sin
#тепер можна звертатись до цих функцій
#зверніть увагу, що в такому разі ім'я модуля не потрібне cos
```

Такий спосіб має один недолік, якщо у програмі вже визначено функцію чи змінну з таким ім'ям, то вона буде замінена на імпортовану функцію, що може викликати небажані ефекти в програмі.

Якщо назва модуля дуже довга, то для спрощення набору можна задати для модуля псевдонім:

```
import math as m #тепер замість math можна використовувати псевдонім
m m.sqrt #m.sqrt замість math.sqrt
```

Аналогічно можна задати псевдоніми для імпортованих функцій або змінних:

```
#задаємо псевдонім для функції визначення квадратного кореня
from math import sqrt as sq
sq #використовуємо псевдонім
```

Підключати можна, як бібліотечні модулі мови Python, так і модулі створені програмістом. Для того, щоб модуль можна було підключити, він повинен знаходитись у доступному місці. Шляхи пошуку модулів вказані у змінній sys.path. Переглянути sys.path можна за допомогою наступних команд:

import sys

sys.path

У шляхи пошуку за замовчуванням включено поточну директорію (з якої запущено програму), а також системні директорії Python. Змінну sys.path можна модифікувати вручну в головній програмі, що дозволяє додати до шляху будь-який каталог:

import sys

sys.path.append("d:\\python_work")

Слід відзначити, що в разі коли модуль не буде знайдено, буде згенеровано помилку ImportError, а якщо не буде знайдено у модулі задану функцію чи змінну, то буде згенеровано виняткову ситуацію AttributeError.

Крім того слід розуміти, що оскільки Python – це інтерпретатор, тому при виконанні команди спочатку відбувається її перетворення у байт-код, а потім вона виконується. Це доволі повільний процес, особливо, якщо завантажується багато великих модулів. Тому для прискорення завантаження модулів можна завантажувати уже сформований байт-код. Це значно швидше. Зважаючи на це Python при першому використанні модуля створює файл з байт-кодом і зберігає його на диску. Цей файл має таке ж ім’я як і модуль, але інше розширення (.рус замість .ру). При подальших запусках 37 використовуються ці файли. Якщо вносяться якісь зміни до модуля, то ці файли також змінюються при наступному запуску. Коли ж модуль уже налагоджений і не буде змінюватись, то можна замість нього використовувати одразу файл байт-коду. Для цього їх можна збирати у спеціальні каталоги, створюючи свої власні бібліотеки.

В мові програмування Python функція може викликати будь-яку кількість інших функцій. Функції також можуть викликати самі себе, тобто мають властивість рекурсивності. Рекурсія – спосіб опису об’єктів або обчислювальних

процесів через самих себе. Рекурсивне програмування дозволяє описати процес що повторюється без явного використання операторів циклу.

Розглянемо реалізацію рекурсії на прикладі функції обчислення факторіала та обчислення цілого степеня числа. Формула обчислення факторіала:

n!=1, якщо n=0;

n!=n*(n–1)!, якщо n>0.

Формула обчислення цілого степеня числа:

x^n =1, якщо n=0;

x^n =x*x^n-1 , якщо n>0.

З формул видно, що для обчислення кожного наступного значення треба знати попереднє. Розглянемо реалізацію функцій обчислення факторіала і цілого степеня числа.

Рекурсивні функції є потужним механізмом у програмуванні. На жаль, вони не завжди ефективні стосовно пам’яті. Це пов’язане з тим, що кожний "вкладений" виклик функції – це черговий набір всіх локальних змінних цієї функції.

Також неуважність при описі рекурсивної функції може привести до нескінченості рекурсії, коли ланцюжок викликів функцій ніколи не завершується і триває, поки не скінчиться вільна пам'ять в комп'ютері. Тому необхідністю для працездатності рекурсивних функцій є наявність правильно оформленої умови закінчення рекурсивних викликів (наприклад, перевірка значення параметра, що змінюється).

Рекурсія має негативні сторони. Вона багато разів ініціалізує механізм виклику функції і збільшує пов’язані з ним витрати процесорного часу і пам’яті (кожне рекурсивне звернення створює копію її параметрів і локальних об’єктів). Ітерація зазвичай відбувається в межах функції, так що тут немає витрат на повторні виклики функції і додаткове виділення пам’яті. Налагодження рекурсивної функції викликає великі труднощі, ніж налагодження ітераційної функції[29].

Будь-яка проблема, яка може бути вирішена рекурсивно, може бути також вирішена і ітераційно (не рекурсивно).

Рекурсивний підхід краще ітераційного в тих випадках, коли рекурсія природніше відображає математичну сторону задачі і призводить до програми, яка простіше для розуміння.

Іншою причиною для вибору рекурсивного рішення є те, що ітераційне рішення може не бути очевидним.

Наведено приклад програми, за якою буде обчислюватися площа трикутника, заданого довжинами його сторін.

2.1. Шифр Цезаря.

Звичайний шифр Цезаря - це один з найпростіших методів шифрування, який використовує зсув символів у тексті на певну кількість позицій у алфавіті.

Основні кроки для реалізації шифрування та дешифрування шифром Цезаря включають наступне:

1. Вибір ключа: Обирається ціле число, яке визначає величину зсуву (ключ шифру). Наприклад, ключ 3 означає, що кожна літера буде зсунута на 3 позиції вперед у випадку шифрування або назад у випадку дешифрування.
2. Шифрування:
    - Для кожного символу відкритого тексту визначається його ASCII-код.
    - Виконується зсув символу за допомогою ключа.
    - Якщо символ - літера, то зсув виконується циклічно через алфавіт (наприклад, після 'Z' переходимо до 'A').
    - Зсунуті символи зберігаються у зашифрованому тексті.
3. Дешифрування:
    - Зашифрований текст дешифрується, використовуючи протилежний зсув, тобто віднімання ключа замість додавання.
    - Як і при шифруванні, зсув застосовується циклічно, щоб врахувати алфавітні межі.

4. Обробка символів за межами алфавіту: Символи, які не є літерами (цифри, пробіли, розділові знаки), залишаються незмінними у відкритому та зашифрованому текстах[14].

Ось реалізація шифрування та дешифрування шифру Цезаря на Python (*Рис. 2.1.)*:

```python
def caesar_encrypt(text, shift):
    """Encrypt the text using Caesar cipher with the given shift."""
    encrypted_text = ""
    for char in text:
        if char.isalpha():
            # Determine the starting point (A or a)
            start = ord('A') if char.isupper() else ord('a')
            # Perform the shift
            encrypted_char = chr(start + (ord(char) - start + shift) % 26)
            encrypted_text += encrypted_char
        else:
            # Non-alphabetic characters are added directly
            encrypted_text += char
    return encrypted_text

def caesar_decrypt(text, shift):
    """Decrypt the text using Caesar cipher with the given shift."""
    return caesar_encrypt(text, -shift)

# Приклад використання
text_to_encrypt = "Hello, World!"
shift_value = 3

# Шифрування
encrypted_text = caesar_encrypt(text_to_encrypt, shift_value)
print(f"Зашифрований текст: {encrypted_text}")

# Дешифрування
decrypted_text = caesar_decrypt(encrypted_text, shift_value)
print(f"Розшифрований текст: {decrypted_text}")
```

Рисунок 2.1. – Шифр Цезаря.

2.2. Шифр Гронсфельда.

Шифр Гронсфельда є поліалфавітним шифром, де кожен символ тексту шифрується за допомогою різних алфавітів, що відповідають ключовим словам або числам.

Основні аспекти реалізації шифрування і дешифрування Шифром Гронсфельда без прикладів коду включають наступне:

1. Ключ шифрування: Ключ визначає зсуви для кожного символа в тексті. Це може бути послідовність чисел або символів, що визначають зсуви для кожного символа тексту.

2. Циклічність ключа**:** Ключ може бути коротшим за довжину тексту, тому для кожного символа тексту використовується зсув з ключа знову з початку ключа.

3. Шифрування символів: Кожен символ відкритого тексту зсувається вперед на відповідну кількість позицій згідно з ключем для створення шифртексту.

4. Дешифрування символів: Шифртекст зсувається назад на відповідну кількість позицій згідно з ключем для отримання вихідного тексту.

5. Обробка символів: Символи відкритого тексту і шифртексту обробляються в межах дозволених символів ASCII (від ' ' до '~'). Всі символи, які не входять у цей діапазон, можуть залишатися незмінними.

6. Формули для зсуву: Для шифрування застосовується формула (ord(char) - ord(' ') + shift) % 95 + ord(' '), де shift - це зсув з ключа. Для дешифрування використовується обернена формула (ord(char) - ord(' ') - shift) % 95 + ord(' ').

Ось реалізація шифрування та дешифрування шифру Гронсфельда методом підстановки на Python (*Рис. 2.2.)*:

```python
def encrypt_gronsfeld(plaintext, key):
    ciphertext = []
    key_length = len(key)
    for i, char in enumerate(plaintext):
        shift = int(key[i % key_length])  # Вираховуємо зміщення за ключем
        encrypted_char = chr((ord(char) - ord('A') + shift) % 26 + ord('A'))
        ciphertext.append(encrypted_char)
    return ''.join(ciphertext)

def decrypt_gronsfeld(ciphertext, key):
    plaintext = []
    key_length = len(key)
    for i, char in enumerate(ciphertext):
        shift = int(key[i % key_length])  # Вираховуємо зміщення за ключем
        decrypted_char = chr((ord(char) - ord('A') - shift + 26) % 26 + ord('A'))
        plaintext.append(decrypted_char)
    return ''.join(plaintext)
# Приклад використання:
plaintext = "HELLO"
key = "31415"  # Приклад ключа (числа)
encrypted_text = encrypt_gronsfeld(plaintext, key)
decrypted_text = decrypt_gronsfeld(encrypted_text, key)

print("Зашифрований текст:", encrypted_text)
print("Розшифрований текст:", decrypted_text)
```

Рисунок 2.2. - Шифр Гронсфельда

2.3. Шифр Віженера.

Шифр Віженера використовує поліалфавітний метод шифрування, що означає використання декількох алфавітів для шифрування тексту. Це досягається шляхом циклічного використання ключа, який складається з кількох символів. Основні аспекти реалізації шифрування та дешифрування Шифром Віженера без прикладів коду включають наступне:

1. Ключеве слово: Ключеве слово (або ключ) використовується для шифрування і дешифрування тексту. Воно повторюється так, що його довжина відповідає довжині тексту.

2. Шифрування символів: Кожен символ відкритого тексту зсувається за допомогою зсуву, що визначається відповідним символом ключа.

3. Дешифрування символів: Шифртекст зсувається назад за допомогою зсуву, що визначається відповідним символом ключа для отримання вихідного тексту.

4. Циклічність ключа: Якщо ключ коротший за текст, він повторюється циклічно до потрібної довжини.

5. Визначення зсуву: Зсув для кожного символу в тексті визначається за його позицією у ключі (зазвичай відповідає позиції в алфавіті, де 'A' = 0, 'B' = 1, ..., 'Z' = 25).

6. Формули для зсуву: Для шифрування застосовується формула (ord(char) - ord('A') + ord(key_char) - ord('A')) % 26 + ord('A'), де char - це символ відкритого тексту, key_char - символ ключа. Для дешифрування використовується формула (ord(char) - ord('A') - (ord(key_char) - ord('A'))) % 26 + ord('A').

7. Обробка символів**:** Символи відкритого тексту і шифртексту обробляються в межах латинського алфавіту (від 'A' до 'Z'). Усі інші символи можуть залишатися незмінними або відкидатися.

8. Метод використання ключа: Ключ може бути введений користувачем або визначений програмно.

Ось реалізація шифрування та дешифрування шифру Віженера методом підстановки на Python (*Рис. 2.3.)*:

```python
import string

def encrypt_vigenere(plaintext, key):
    ciphertext = []
    key_index = 0
    for char in plaintext:
        if char.isalpha():
            shift = ord(key[key_index % len(key)]) - ord('a')
            if char.islower():
                encrypted_char = chr((ord(char) - ord('a') + shift) % 26 + ord('a'))
            else:
                encrypted_char = chr((ord(char) - ord('A') + shift) % 26 + ord('A'))
            ciphertext.append(encrypted_char)
            key_index += 1
        else:
            ciphertext.append(char)
    return ''.join(ciphertext)

def decrypt_vigenere(ciphertext, key):
    plaintext = []
    key_index = 0
    for char in ciphertext:
        if char.isalpha():
            shift = ord(key[key_index % len(key)]) - ord('a')
            if char.islower():
                decrypted_char = chr((ord(char) - ord('a') - shift + 26) % 26 + ord('a'))
            else:
                decrypted_char = chr((ord(char) - ord('A') - shift + 26) % 26 + ord('A'))
            plaintext.append(decrypted_char)
            key_index += 1
        else:
            plaintext.append(char)
    return ''.join(plaintext)
```

```python
# Приклад використання:
plaintext = "Hello, World!"
key = "python"

encrypted_text = encrypt_vigenere(plaintext, key)
print("Зашифрований текст:", encrypted_text)

decrypted_text = decrypt_vigenere(encrypted_text, key)
print("Розшифрований текст:", decrypted_text)
```

Рисунок 2.3. - Шифр Віженера

2.4.Омофонічна заміна.

Омофонічна заміна (або фонетична криптографія) — це техніка шифрування, яка базується на заміні букв, складів або слів тексту на інші букви, склади або слова, що звучать так само або дуже схоже. Цей метод використовує явище омофонії, коли різні за написанням слова або літери звучать однаково або схоже[30].

Основні аспекти реалізації шифрування та дешифрування омофонічною заміною без прикладів коду включають наступне:

1. Створення таблиці замін: На початку потрібно створити таблицю замін, де кожен символ відкритого тексту має відповідність з символом шифртексту. Ця таблиця може бути задана наперед або генеруватися випадковим чином.

2. Шифрування**:** Кожен символ відкритого тексту замінюється відповідним символом з таблиці замін.

3. Дешифрування**:** Кожен символ шифртексту замінюється відповідним символом зворотної таблиці замін.

4. Випадковість або фіксованість замін: Можна вибрати між випадковою генерацією таблиці замін або використанням фіксованої таблиці, яка зберігається в пам'яті програми.

5. Обробка різних типів символів: Потрібно вирішити, як обробляти символи, які не знаходяться в таблиці замін (наприклад, пробіли, цифри або спеціальні символи).

6. Метод введення ключа: Ключ може бути введений користувачем (наприклад, як унікальний ключ або пароль) або генеруватися програмно.

7. Швидкодія і пам'ять: Ефективне управління пам'яттю і швидкість реалізації є важливими аспектами при великих обсягах даних або в умовах обмежених ресурсів.

8. Захист від криптоаналітичних атак: Розглядайте можливості підвищення стійкості шифру до різних видів криптоаналітичних атак, таких як статистичний аналіз або атаки на основі частотного аналізу.

Ось реалізація шифрування та дешифрування омофонічної заміни методом підстановки на Python (*Рис. 2.4.)*:

```python
from random import randint

# словник заміни
keys = {
    'A': ['q', '1', '!', '`', 'B', 'C'], 'B': ['w', '2'], 'C': ['e', '3', '@'],
    'D': ['r', '4', '#', ':'], 'E': ['t', '5', '$', ';', 'F', 'G', 'H', 'I', ','],
    'F': ['y', '6'], 'G': ['u', '7'], 'H': ['i', '8', '%', '"', 'M', 'N'],
    'I': ['o', '9', '^', '/', 'O', 'P'], 'J': ['p'], 'K': ['a'],
    'L': ['s', '0', '&', '?'], 'M': ['d', '*'], 'N': ['f', '(', '<', 'Q', 'R'],
    'O': ['g', ')', '>', 'T', 'U'], 'P': ['h', '-'], 'Q': ['j'],
    'R': ['k', '_', '|', 'X', 'Y'], 'S': ['l', '+', '№', 'D', 'E'],
    'T': ['m', '=', 'K', 'L', 'W', 'S', 'V', '.', 'Z'], 'U': ['n', '[', 'J'],
    'V': ['b'], 'W': ['v', ']'], 'X': ['c'], 'Y': ['z', '{'], 'Z': ['x'], ' ': ['}']}


# Функція що виконує шифрування
3 usages
def encrypt(original_text):
    result = ''
    for i in original_text.upper():
        # якщо знак тексту є в словнику, то:
        if i in keys:
            # підклейка до шифру відповідної випадкової букви
            result += keys[i][randint( a: 0, len(keys[i]) - 1)]
    # надрукувати результат роботи шифратора
    return result


originalText = "This is an example of homophonic substitution"
encrypredText = encrypt(originalText)
print("Original text: ", originalText)
print("Encrypted text: ", encrypredText)
```

Рисунок 2.4. – Омофонічна заміна

2.5. Книжковий шифр.

Книжковий шифр (або кодова книга) — це метод шифрування, який використовує певний текстовий ресурс (наприклад, книгу) як ключ для шифрування та розшифрування повідомлень. Цей метод є простим, але досить ефективним за умови, що ключ відомий тільки відправнику та одержувачу.

Основні аспекти реалізації шифрування та дешифрування книжковим шифром без прикладів коду включають:

1. Книжка або письмове джерело: Обидва учасники мають доступ до однієї і тієї ж книги або джерела, яке використовується для заміни тексту. Це може бути книга, журнал, енциклопедія або будь-який інший документ з великою кількістю тексту.
2. Обмін ключами: Для кожного повідомлення необхідно обмінюватися інформацією про сторінку, рядок і, можливо, позицію слова або фрази в книзі. Це слугує ключем для шифрування і дешифрування.
3. Обробка тексту: Текст розбивається на слова або фрази, і кожен з них замінюється відповідним словом або фразою з книги.
4. Стійкість до криптоаналізу: Важливо обирати слова або фрази з книги, які важко передбачити або відтворити для потенційного криптоаналітика.
5. Конфіденційність книги: Важливо, щоб книга або джерело були доступні тільки відправникові і отримувачеві, щоб забезпечити конфіденційність обміну повідомленнями.
6. Ефективність і швидкість: В залежності від розміру тексту, який потрібно зашифрувати і дешифрувати, важливо враховувати швидкість доступу до книги або джерела та швидкість обміну інформацією про ключ.
7. Безпека і збереження: Потрібно забезпечити безпечний обмін ключами і збереження книги або джерела для запобігання доступу третіх осіб.

8. Ручне або автоматизоване використання: Можливо, це можна виконати вручну або за допомогою програмного забезпечення, яке автоматизує обмін інформацією про ключ.

Ось реалізація шифрування та дешифрування книжковим шифром методом підстановки на Python (*Рис. 2.5.)*:

```python
# Функція для створення індексів символів книги
def create_book_index(book_text):
    book_index = {}
    for i, char in enumerate(book_text):
        if char not in book_index:
            book_index[char] = []
        book_index[char].append(i)
    return book_index

# Функція для шифрування повідомлення
def encrypt_message(message, book_index):
    encrypted_message = []
    for char in message:
        if char in book_index:
            encrypted_message.append(book_index[char][0])
            book_index[char] = book_index[char][1:] + [book_index[char][0]] # Rotate list
        else:
            encrypted_message.append(-1) # Placeholder for characters not in book
    return encrypted_message
```

```python
# Функція для розшифрування повідомлення
def decrypt_message(encrypted_message, book_text):
    decrypted_message = []
    for index in encrypted_message:
        if index != -1:
            decrypted_message.append(book_text[index])
        else:
            decrypted_message.append('?') # Placeholder for characters not in book
    return ''.join(decrypted_message)
```

Рисунок 2.5. – Книжковий шифр

2.6. Атаки на основі шифротексту.

Атаки на основі шифротексту (ciphertext-only attacks) є однією з найпоширеніших форм криптоаналітичних атак. У цих атаках зловмисник має доступ лише до шифротекстів і намагається розшифрувати їх без наявності відповідного відкритого тексту або ключа. Ці атаки можуть використовувати різноманітні методи, такі як статистичний аналіз, аналіз частоти входження символів, використання знань про можливі шаблони або структури в шифротексті, випробування ключового простору, використання слабкостей алгоритмів шифрування чи їх реалізації, а також використання побічних каналів і інших витоків інформації[4]. Атаки на основі шифротексту можуть бути ефективними засобами атаки на системи криптографії, якщо не забезпечені відповідні заходи безпеки, що зменшують ризик розкриття відкритого тексту чи ключа. Такі атаки стають можливими через слабкості в алгоритмах шифрування або їх реалізації. Ці атаки зазвичай вимагають від атакуючого аналізувати шифротекст і використовувати доступні дані та можливі знання про систему шифрування. Вони можуть включати в себе використання здогадок про вміст відкритого тексту, вивчення структурних особливостей або властивостей шифротексту, і намагатися виокремити частоти або інші патерни, що можуть вказувати на оригінальні дані. У цьому розділі ми розглянемо основні методи атак на основі шифротексту та реалізуємо їх за допомогою мови програмування Python.

Основні методи атак на основі шифротексту:

1. Частотний аналіз Частотний аналіз є одним з найстаріших методів криптоаналізу, який базується на статистичних властивостях мови відкритого тексту. У разі моноалфавітних шифрів, таких як шифр Цезаря або шифр простої підстановки, частота появи кожного символу в шифротексті повинна приблизно відповідати частоті появи цих символів у відкритому тексті.

2. Брутфорс атаки Брутфорс атаки передбачають перебір всіх можливих ключів до тих пір, поки не буде знайдений правильний. Цей метод є найпростіший, але його ефективність знижується із збільшенням розміру ключа.

3. Аналіз шаблонів та статистичних характеристик Цей метод базується на аналізі шаблонів шифротексту, таких як повторювані підрядки або послідовності символів. Використовуючи такі шаблони, можна спробувати відновити деякі частини відкритого тексту або ключа[2].

Реалізація частотного аналізу на Python:

Для демонстрації частотного аналізу розглянемо простий шифр Цезаря. У шифрі Цезаря кожна буква відкритого тексту зміщується на певну кількість позицій в алфавіті (*Рис. 3.1.*).

```python
from collections import Counter

def caesar_decrypt(ciphertext, shift):
    decrypted_text = ""
    for char in ciphertext:
        if char.isalpha():
            shift_base = ord('A') if char.isupper() else ord('a')
            decrypted_text += chr((ord(char) - shift_base - shift) % 26 + shift_base)
        else:
            decrypted_text += char
    return decrypted_text
```

```python
def frequency_analysis(ciphertext):
    # Обчислюємо частоту кожної букви в шифротексті
    frequency = Counter(ciphertext)
    total_chars = sum(frequency.values())

    # Обчислюємо відсоток для кожної букви
    frequency_percent = {char: (count / total_chars) * 100 for char, count in frequency.it

    return frequency_percent
# Приклад використання
ciphertext = "KHOOR ZRUOG"  # Шифротекст, зашифрований з використанням шифру Цезаря зі зсу
shift = 3  # Припускаємо, що ми знаємо зсув для прикладу

decrypted_text = caesar_decrypt(ciphertext, shift)
frequency_percent = frequency_analysis(ciphertext)

print("Decrypted text:", decrypted_text)
print("Frequency analysis:", frequency_percent)
```

Рисунок 3.1. – Реалізація частотного аналізу на Python.

Частотний аналіз дозволяє визначити можливі відповідності між шифрованими символами і символами відкритого тексту.

Атаки на основі шифротексту можуть бути ефективними, якщо використовувані алгоритми шифрування мають слабкі місця. Використання методів частотного аналізу, брутфорс атак та аналізу шаблонів може допомогти криптоаналітикам зламати шифри і відновити відкритий текст. Використання Python для реалізації цих методів дозволяє автоматизувати процес аналізу і значно підвищити ефективність криптоаналізу.

2.7. Атаки на основі відкритих текстів.

Атаки на основі відкритих текстів (known-plaintext attacks) є одним з найпоширеніших видів криптоаналітичних атак. У цих атаках зловмисник має доступ як до шифротексту, так і до відповідного відкритого тексту. Метою таких атак є виявлення ключа шифрування або алгоритму шифрування, що дозволить зловмиснику розшифровувати інші шифротексти[22]. Ці атаки можуть використовувати аналіз взаємозв'язку між відкритим текстом і відповідним шифротекстом для виведення ключів або зразків шифрування. Атаки можуть включати в себе використання статистичного аналізу, що дає можливість визначати частоту символів або фраз у відкритому тексті, що може бути корисним для визначення правильності ключа. Такі атаки можуть також використовувати знання про структуру або формат відкритого тексту, що передається, наприклад, заголовки або структура документа. Атаки на основі відкритих текстів можуть бути ефективними у випадках, коли зловмисник має можливість отримати доступ до великої кількості відкритого тексту або впливати на створення або передачу відкритого тексту.

Деякі методи атак на основі відкритих текстів включають аналіз співвідношення між відкритим текстом і шифрованим текстом, щоб вивести частоту або патерни, що можуть бути використані для знаходження криптографічних ключів. Інші атаки можуть використовувати статистичний аналіз, що дає змогу визначати частоту символів або фраз в відкритому тексті, що може бути важливим для визначення правильності ключа

У цьому розділі ми розглянемо основні методи атак на основі відкритих текстів та реалізуємо їх за допомогою мови програмування Python.

Основні методи атак на основі відкритих текстів

1. Метод вихідних блоків (Chosen Plaintext Attack) У цьому методі зловмисник обирає специфічні відкриті тексти та отримує відповідні шифротексти, щоб знайти залежність між ними і ключем.

2. Лінійний криптоаналіз Цей метод базується на лінійній апроксимації нелінійних компонентів шифру. Лінійний криптоаналіз використовує статистичні дані про відкриті тексти та шифротексти, щоб відновити ключ.

3. Диференційний криптоаналіз У диференційному криптоаналізі зловмисник аналізує різниці між парами відкритих текстів і відповідними шифротекстами для виявлення інформації про ключ.

Реалізація лінійного криптоаналізу на Python

Для демонстрації лінійного криптоаналізу розглянемо простий блоковий шифр, де ми будемо використовувати лінійні апроксимації для визначення ключа(*Рис. 3.2.*).

```python
import random

def simple_block_encrypt(plaintext, key):
    # Простий блоковий шифр (наприклад, XOR)
    return ''.join(chr(ord(p) ^ key) for p in plaintext)

def simple_block_decrypt(ciphertext, key):
    # Дешифрування простого блокового шифру
    return ''.join(chr(ord(c) ^ key) for c in ciphertext)

def generate_known_plaintexts(n, key):
    known_plaintexts = []
    for _ in range(n):
        plaintext = ''.join(chr(random.randint(65, 90)) for _ in range(8))  # Випадковий
        ciphertext = simple_block_encrypt(plaintext, key)
        known_plaintexts.append((plaintext, ciphertext))
    return known_plaintexts
```

```python
def linear_cryptanalysis(known_plaintexts):
    # Лінійний криптоаналіз для простого XOR шифру
    possible_keys = range(256)
    best_key = None
    best_score = 0

    for key in possible_keys:
        score = 0
        for plaintext, ciphertext in known_plaintexts:
            decrypted_text = simple_block_decrypt(ciphertext, key)
            if decrypted_text == plaintext:
                score += 1

        if score > best_score:
            best_score = score
            best_key = key

    return best_key
```

```python
# Приклад використання
key = 42  # Припустимо, що це невідомий ключ
known_plaintexts = generate_known_plaintexts(100, key)
recovered_key = linear_cryptanalysis(known_plaintexts)

print("Recovered key:", recovered_key)
```

Рисунок 3.2. – Реалізація лінійного криптоаналізу на Python

2.8. Атаки на основі підібраного відкритого тексту.

Атаки на основі підібраного відкритого тексту (chosen-plaintext attacks) є потужним типом криптоаналітичних атак, де зловмисник має можливість обрати конкретні відкриті тексти для шифрування та отримати відповідні шифротексти. Це дозволяє досліджувати властивості шифру та відновити ключ або структуру алгоритму шифрування. Ці атаки можуть бути особливо ефективними проти криптосистем, що використовують традиційні симетричні шифри, такі як блочні шифри або потокові шифри[18].

Основними принципами атаки CPA є здатність аналізувати відповідність між введеним відкритим текстом і отриманим шифротекстом, а також використання цієї інформації для виведення ключів або відкритого тексту, що був зашифрований. Зловмисник може використовувати аналіз частоти символів, патернів або статистичних властивостей шифротексту, отриманих від різних варіантів введеного відкритого тексту для отримання цінної інформації.

Ці атаки показують, що навіть сильні шифри можуть бути вразливими, якщо зловмисник має можливість вибирати і вводити відкритий текст для аналізу. Вони підкреслюють важливість тестування криптографічних систем на вразливість до CPA і впровадження захисних механізмів, таких як використання аутентифікації або шифрування з автентичними даними для запобігання таким атакам. Атаки на основі підібраного відкритого тексту використовують можливість вибору конкретних фрагментів відкритого тексту з бажанням аналізувати відповідний шифротекст. Це дозволяє зловмисникові отримувати корисну інформацію про криптографічну систему, наприклад, використовуючи аналіз взаємозв'язку між введеним відкритим текстом і відповідним шифротекстом для виведення ключів або отримання конкретного відкритого тексту, який був зашифрований.

Ці атаки показують, що навіть сильні криптографічні алгоритми можуть бути вразливими до атак, якщо зловмисник має достатній рівень контролю над введеним відкритим текстом і може спостерігати відповідний шифротекст. Для запобігання таким атакам важливо використовувати алгоритми, які відповідають сучасним стандартам безпеки, а також застосовувати додаткові захисні механізми, такі як аутентифікація, аутентичне шифрування або використання додаткових інформаційних каналів для забезпечення конфіденційності і цілісності даних.

У цьому розділі ми розглянемо основні методи атак на основі підібраного відкритого тексту та реалізуємо їх за допомогою мови програмування Python.

Основні методи атак на основі підібраного відкритого тексту

1. Аналіз шифрування фіксованих відкритих текстів Зловмисник шифрує фіксовані відкриті тексти, наприклад, послідовності нулів або однакових символів, для виявлення шаблонів у шифротекстах.

2. Атака на основі шаблонів Зловмисник обирає відкриті тексти з певними шаблонами або структурами, які можуть розкрити слабкості в алгоритмі шифрування.

3. Диференційний криптоаналіз Диференційний криптоаналіз використовує різниці між парами відкритих текстів та відповідними шифротекстами для виявлення ключових залежностей.

Реалізація атаки на основі підібраного відкритого тексту на Python

Для демонстрації реалізації атаки на основі підібраного відкритого тексту розглянемо простий блоковий шифр та використаємо підібрані відкриті тексти для аналізу шифру (*Рис. 3.3.*).

```python
import random

def simple_block_encrypt(plaintext, key):
    # Простий блоковий шифр (наприклад, XOR)
    return ''.join(chr(ord(p) ^ key) for p in plaintext)

def simple_block_decrypt(ciphertext, key):
    # Дешифрування простого блокового шифру
    return ''.join(chr(ord(c) ^ key) for c in ciphertext)

def generate_chosen_plaintexts(n, key):
    chosen_plaintexts = []
    for _ in range(n):
        plaintext = ''.join(chr(random.randint(65, 90)) for _ in range(8))  # Випадковий
        ciphertext = simple_block_encrypt(plaintext, key)
        chosen_plaintexts.append((plaintext, ciphertext))
    return chosen_plaintexts
```

```python
def chosen_plaintext_attack(chosen_plaintexts):
    # Атака на основі підібраного відкритого тексту для простого XOR шифру
    possible_keys = range(256)
    best_key = None
    best_score = 0

    for key in possible_keys:
        score = 0
        for plaintext, ciphertext in chosen_plaintexts:
            decrypted_text = simple_block_decrypt(ciphertext, key)
            if decrypted_text == plaintext:
                score += 1

        if score > best_score:
            best_score = score
            best_key = key

    return best_key

# Приклад використання
key = 42  # Припустимо, що це невідомий ключ
chosen_plaintexts = generate_chosen_plaintexts(100, key)
recovered_key = chosen_plaintext_attack(chosen_plaintexts)

print("Recovered key:", recovered_key)
```

Рисунок 3.3. – Реалізація атаки на основі підібраного відкритого тексту на Python

ВИСНОВКИ

1. Реалізовано кілька базових підстановочних шифрів, таких як, шифри Цезаря, Віженера, Гронсфельда, омофонічний шифр.

Шифр Цезаря є одним з найпростіших і найстаріших методів шифрування, який полягає в зсуві кожної літери в тексті на певну кількість позицій в алфавіті.

Шифр Віженера використовує ключове слово для шифрування тексту, при цьому кожна літера відкритого тексту зміщується на кількість позицій, визначену відповідною літерою ключа.

Шифр Гронсфельда схожий на шифр Віженера, але ключ складається з цифр, які вказують кількість позицій зсуву для кожної літери.

Омофонічний шифр використовує кілька символів для шифрування однієї літери, що ускладнює частотний аналіз і підвищує стійкість шифру.

2. Створена комп’ютерна модель механічної шифрувальної машини Джефферсона.

Механічна шифрувальна машина Джефферсона, також відома як диск Джонсона або шифр Джефферсона, використовує набір дисків з літерами, які обертаються для шифрування повідомлень. Комп'ютерна модель цієї машини відтворює процес обертання дисків та підстановки символів, що дозволяє автоматизувати шифрування та дешифрування текстів за допомогою цієї історичної методики.

3. Проаналізовано дієздатність на предмет шифрування і дешифрування згаданих шифраторів.

Для кожного з шифрів були проведені тести на шифрування та дешифрування текстів різної довжини та складності. Було досліджено, наскільки ефективно та точно кожен шифр справляється зі своїм завданням, які потенційні слабкі місця можуть виникнути в процесі використання цих методів

шифрування, а також які аспекти слід враховувати для забезпечення максимальної безпеки.

4. Перевірені на практиці відомі методи зламу простих підстановочних шифрів. Для цього написані відповідні програми і перевірено їх дієздатність на практиці.

Були розроблені програми для зламу простих підстановочних шифрів, таких як шифр Цезаря та шифр Віженера. Зокрема, використовувалися методи частотного аналізу, перебору ключів або “грубої сили”. Було перевірено, наскільки ефективно ці програми можуть розшифровувати текст без ключа, і визначено, які методи зламу є найбільш дієвими для кожного конкретного шифру.

5. Була підтверджена ефективність шифрування та дешифрування, а також можливість зламу шифрів за допомогою частотного аналізу, перебору або “грубої сили”.

Під час експериментів було підтверджено, що шифри Цезаря, Віженера та інші можуть бути успішно зламані за допомогою частотного аналізу, перебору ключів або методів “грубої сили”. Це підкреслює важливість використання складніших криптографічних методів та ключів для забезпечення надійного захисту інформації. Результати досліджень показали, що хоча базові шифри можуть забезпечувати певний рівень безпеки, їх використання без додаткових заходів захисту не є достатнім для захисту від сучасних криптографічних атак.

6. Представлені програми цікаві як в курсі криптографії (історичний ракурс) так і в курсі програмування. Адже ці програми вимагають витонченої роботи з масивами, файлами, рядками.

Розроблені програми для шифрування та дешифрування є корисними навчальними інструментами в курсах криптографії та програмування. Вони демонструють історичні аспекти розвитку криптографії та дозволяють зрозуміти

основні принципи шифрування. Крім того, ці програми вимагають вміння працювати з масивами даних, файлами та рядками, що є важливими навичками в програмуванні. Використання таких програм у навчанні допомагає краще зрозуміти складні концепції та розвиває практичні навички в області розробки програмного забезпечення.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Алгоритм Рабіна [Електронний ресурс]: https://d-learn.pnu.edu.ua/data/users/ 4808/zikm/ЛАб_роб/ Лабораторна%20робота%20N10%20ДОСЛІДЖЕННЯ%20КРИПТОАЛГОРИТМ У%20ШИФРУВАННЯ%20РАБІНА.pdf
2. Безпека випадкових чисел в Python [Електронний ресурс]. Режим доступу: https://habr.com/ru/company/pt/blog/156133/
3. Бібліотека Random в Python [Електронний ресурс]. Режим доступу: https://docs.python.org/release/2.6.8/library/random.html
4. Ель Гамаля [Електронний ресурс]: https://d-learn.pnu.edu.ua/data/users/ 4808/zikm/Методичні%20рекомендації%20до%20лабораторних%20робіт%20з%20курсу%20Захист%20інформації%20в%20компютерних%20системах.
5. Поле Галуа [Електронний ресурс]: https://uk.wikipedia.org/wiki/Поле_Галуа 3. AES [Електронний ресурс]: https://habr.com/ru/post/497672/
6. Сліповичев.І.І. «Генератор псевдовипадкових чисел» Навчальний посібник. 2017 р.
7. Схема Ель - Гамаля [Електронний ресурс]: https://uk.wikipedia.org/wiki/ Схема_Ель-Гамаля
8. Шифр [Електронний ресурс]: https://uk.wikipedia.org/wiki/Шифр 12. Шифрування [Електронний ресурс]: https://uk.wikipedia.org/wiki/Шифрування
9. Шифрування з асиметрисними ключами [Електронний ресурс]: https:// uk.wikipedia.org/wiki/Асиметричні_алгоритми_шифрування.
10. Шифрування з симетричними ключами [Електронний ресурс]: https:// uk.wikipedia.org/wiki/Шифрування_з_симетричними_ключами
11. Шифрування з симетричними ключами [Електронний ресурс]: https:// uk.wikipedia.org/wiki/Шифрування_з_симетричними_ключами

12. Шифрування: типи і алгоритми [Електронний ресурс]: https://wiki.hostpro.ua/ua/ knowledgebase/shifruvannja-tipi-i-algoritmi/.
13. AES [Електронний ресурс]: https://bit.nmu.org.ua/ua/student/metod/cryptology/ лекция%209.pdf
14. Криптосистема Рабіна [Електронний ресурс]: https://uk.wikipedia.org/wiki/ Криптосистема_Рабіна
15. AES-128 [Електронний ресурс]: https://modexp.wordpress.com/2018/10/30/arm64- assembly/#aes
16. Micropython [Електронний ресурс]: https://micropython.org
17. Data Encryption Standard [Електронний ресурс]: https://uk.wikipedia.org/wiki/ Data_Encryption_Standard
18. DESX [Електронний ресурс]: https://ru.wikipedia.org/wiki/DESX
19. IDEA [Електронний ресурс]: https://uk.wikipedia.org/wiki/ IDEA_(%D1%88%D0%B8%D1%84%D1%80)
20. M. Matsumoto, T. Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudorandom number generator (англ.) // ACM Trans. on Modeling and Computer Simulations : journal. — 2017.
21. Martin Aspeli. Professional Plone 4 Development. — Packt Publishing Ltd., 2011. — 516 с.
22. Math.random in JavaScript [Electronic resource]. Mode of access: https://v8.dev/blog/math-random
23. McDonald, N. Past, Present, and Future Methods of Cryptography and Data Encryption. A Research Review [Electronic resource] / McDonald, N. – Mode of access: http://www.eng.utah.edu/~nmcdonal/Tutorials/EncryptionResearchReview.pdf
24. RSA [Електронний ресурс]: https://uk.wikipedia.org/wiki/RSA
25. Stallings, W. Cryptography and Network Security Principles and Practices / Stallings, W. – 4 thEdition. – Prentice Hall, 2005. – 592 p. – Mode of access: http://www.inf.ufsc.br/~bosco.sobral/ensino/ine5680/material-cripto-seg/2014-1/Stallings/Stallings_Cryptography_and_Network_Security.pdf

26. Sweigart, A. Hacking Secret Ciphers with Python [Electronic resource] / Sweigart, A. – Mode of access: https://inventwithpython.com/hackingciphers.pdf 16. Text Processing Services [Electronic resource] / The Python Standard Library – Mode of access: https://docs.python.org/3.4/library/text.html
27. What is Python? Executive Summary [Electronic resource] / Python's standard documentation. – Mode of access: https://www.python.org/doc/essays/blurb/
28. Путівник мовою програмування Python [Електронний ресурс]. Режим доступу: https://pythonguide.rozh2sch.org.ua/
29. Путівник мовою програмування Python [Електронний ресурс]. Режим доступу: https://pythonguide.rozh2sch.org.ua/
30. Основи програмування мовою Python [Електронний ресурс]. Режим доступу: електронний підручник
31. Python [Електронний ресурс] — Режим доступу до ресурсу: https://wikipedia.org/wiki/Python

ДОДАТКИ

Додаток А

Main.py

```
import crypto
import interface
import sys
import os
import telegram

keys_directory = './keys/'

def full_patch(directoy, files):
    filee = []
    full_patch = map(lambda name: os.path.join(directoy,
name), files)
    for file in full_patch:
        if os.path.isfile(file):
            filee.append(file)
    return filee


def crypt_all():
    password = 'password'
    crypto.crypt_AES(full_patch(keys_directory,
os.listdir(keys_directory)), crypto.get_md5_hash(password))


def main():
    password = interface.login_window()
    crypto.decrypt_AES(full_patch(keys_directory,
os.listdir(keys_directory)), crypto.get_md5_hash(password))

    sys.path.insert(0, keys_directory)
    import telegram_api

    message_id, file_id =
telegram.search_message(telegram_api.api_id,
telegram_api.api_hash, keys_directory)

    telegram.download_file(file_id, telegram_api.api_id,
telegram_api.api_hash, keys_directory)
    crypto.decrypt_AES([f'{keys_directory}work_file'],
crypto.get_md5_hash(password))

    interface.main_window(telegram_api.api_id,
```

```
telegram_api.api_hash, keys_directory)

    telegram.delete_message(message_id,
telegram_api.api_id, telegram_api.api_hash, keys_directory)
    crypto.crypt_AES([f'{keys_directory}work_file'],
crypto.get_md5_hash(password))
    telegram.send_file('./keys/work_file',
telegram_api.api_id, telegram_api.api_hash, keys_directory)
    os.remove('./keys/work_file')

    crypto.crypt_AES(full_patch(keys_directory,
os.listdir(keys_directory)), crypto.get_md5_hash(password))


if __name__ == '__main__':
    main()
    #crypt_all()

    Crypto.py

from Cryptodome.Cipher import AES

import hashlib
login_hash =
'5e884898da28047151d0e56f8dc6292773603d0d6aabbdd62a11ef721d
1542d8'


def get_md5_hash(data):
    return hashlib.md5(data.encode('utf-8')).hexdigest()


def get_sha256_hash(password):
    return hashlib.sha256(password.encode('UTF-
8')).hexdigest()


def vereficate_password(password):
    return True if get_sha256_hash(password) == login_hash
else False

def crypt_AES(files, password):
    for file in files:
```

```
        file_in = open(file, 'rb')
        file_data = file_in.read()
        file_in.close()

        key = password.encode('UTF-8')
        cipher = AES.new(key, AES.MODE_EAX)
        ciphertext, tag =
cipher.encrypt_and_digest(file_data)

        file_out = open(file, "wb")
        [file_out.write(x) for x in (cipher.nonce, tag,
ciphertext)]
        file_out.close()


def decrypt_AES(files, password):
    for file in files:
        key = password.encode('UTF-8')

        file_in = open(file, "rb")
        nonce, tag, ciphertext = [file_in.read(x) for x in
(16, 16, -1)]
        cipher = AES.new(key, AES.MODE_EAX, nonce)
        file_data = cipher.decrypt_and_verify(ciphertext,
tag)

        file_out = open(file, 'wb')
        file_out.write(file_data)
        file_out.close()
    Interface.py
    from tkinter import *
from tkinter import filedialog
from tkinter import messagebox

import crypto
import telegram
import work_with_file


def get_file():
    return filedialog.askopenfilename()


def get_directory():
    return filedialog.askdirectory()
```

```
    def login_window():
        def accept_login(password):
            login_page.destroy() if
    crypto.vereficate_password(password) else
    messagebox.showerror(title='ERROR',

    message='WRONG PASSWORD')

        login_page = Tk()

        password = StringVar()

        login_page.title('LOGIN')
        login_page.geometry('200x100')

        Label(login_page, text='PASSWORD:').grid(row=0,
    column=0, padx=(40, 0))
        Entry(login_page, show='*',
    textvariable=password).grid(row=1, column=0, padx=(40, 0))
        Button(login_page, text='Enter', command=lambda:
    accept_login(password.get())).grid(row=3, column=0,
    pady=(10, 0),

    padx=(40, 0))
        login_page.mainloop()
        return password.get()

def main_window(api_id, api_hash, keys_directory):
    def add_new_line():
        def add_window():
            add_window = Tk()

            Label(add_window, text='Area').grid(row=0,
column=0)
            Label(add_window, text='Password').grid(row=0,
column=1)

            area = StringVar(add_window)
            password = StringVar(add_window)

            Entry(add_window,
textvariable=area).grid(row=1, column=0)
```

```
            Entry(add_window,
textvariable=password).grid(row=1, column=1)

            Button(add_window, text='ADD!', command=lambda:
[work_with_file.add_pass(area.get(), password.get()),

messagebox.showinfo(message='Add succesful!'),

add_window.destroy()]).grid(row=2, column=0, columnspan=2)

            add_window.mainloop()

        add_window()

    def refresh_func():
        passwords = work_with_file.get_passwords()
        PlACE_list.delete(0, END)
        PASSWORD_list.delete(0, END)
        for key in passwords:
            PlACE_list.insert(END, key)
            PASSWORD_list.insert(END, passwords[key])

    root = Tk()
    PlACE_list = Listbox(root, selectmode=SINGLE)
    PASSWORD_list = Listbox(root, selectmode=SINGLE)

    PlACE_list.grid(row=0, column=0, rowspan=3)
    PASSWORD_list.grid(row=0, column=1, rowspan=3)

    Button(root, text='Refresh', width=10, command=lambda:
refresh_func()).grid(row=0, column=2, sticky='n')
    Button(root, text='Add password', width=10,
command=lambda: [add_new_line(),
refresh_func()]).grid(row=0, column=2)
    Button(root, text='New file', width=10,
           command=lambda: [work_with_file.create_file(),
messagebox.showinfo(message='File create succesful!'),
                            refresh_func()]).grid(row=0,
column=2, sticky='s')
    refresh_func()

    root.mainloop()
    Telegram.py
    from pyrogram import Client
```

```
def send_file(file, api_id, api_hash, keydir):
    with Client("my_account", api_id, api_hash,
workdir=keydir) as app:
        app.send_document("me", file, caption='#passwords')


def search_message(api_id, api_hash, keydir):
    with Client("my_account", api_id, api_hash,
workdir=keydir) as app:
        for message in app.search_messages("me",
query="#passwords"):
            return message.id, message.document.file_id


def download_file(file_id, api_id, api_hash, keydir):
    with Client("my_account", api_id, api_hash,
workdir=keydir) as app:
        app.download_media(file_id,
file_name='./keys/work_file')


def delete_message(message_id, api_id, api_hash, keydir):
    with Client("my_account", api_id, api_hash,
workdir=keydir) as app:
        app.delete_messages('me', message_id)
    Work_with_file.py
    import pickle


def create_file():
    with open('./keys/work_file', 'wb') as new:
        pickle.dump({}, new)


def add_pass(area, password):
    with open('./keys/work_file', 'rb') as old:
        passwords = pickle.load(old)
    passwords[area] = password
    with open('./keys/work_file', 'wb') as new:
        pickle.dump(passwords, new)

def get_passwords():
    with open('./keys/work_file', 'rb') as old:
        return pickle.load(old)
```

АНОТАЦІЯ

Матвійчук О.М – **Програмні реалізації шифрування, дешифрування та злам підстановочних шифрів мовою Python** – Рукопис.

Кваліфікаційна робота за спеціальністю 122 Комп'ютерні науки. - Волинський національний університет імені Лесі Українки, Луцьк. - 2024р.

Робота присвячена розробці програмного забезпечення для шифрування, дешифрування та зламу підстановочних шифрів з використанням мови Python.

Розглянуто актуальність створення таких програм, що полягає в необхідності забезпечення захисту конфіденційних даних в умовах швидкого розвитку цифрових технологій та збільшення обсягу цифрових комунікацій. Шифрування є одним з найефективніших методів забезпечення безпеки інформації, і підстановочні шифри, як один із класичних методів криптографії, продовжують знаходити застосування в сучасних системах захисту даних.

Метою роботи є розробка та аналіз програмних реалізацій шифрування, дешифрування та зламу підстановочних шифрів мовою Python, а також вивчення ефективності та можливостей цих реалізацій.

Завдання включають огляд теоретичних основ підстановочних шифрів, розробку програмних реалізацій шифрування та дешифрування в IDE PyCharm, дослідження методів зламу підстановочних шифрів та їх реалізація на Python, проведення аналізу ефективності реалізованих програм.

Ключові слова: шифрування, дешифрування, злам, підстановочні шифри.