Шифри існують вже багато століть, але найбільшу популярність вони отримали тільки при появі інтернету та способів комунікації в ньому. На даний момент існує велика кількість сфер, де використовуються шифри, починаючи із захисту особистої інформації в інтернеті, закінчуючи армією та воєнними справами.

В даному випадку розглянемо лікарні. Часто у пацієнтів, які не можуть встати є потреба в спілкуванні, але родичі не можуть бути поряд не весь час. Але через те, що в лікарнях часто стоїть старе або дуже слабке обладнання на нього не можна поставити програми для спілкування в інтернеті. Саме для цього і був розроблений мій чат. Тому, що його можна запускати на будь-якому пристрої. Але з’являється проблема – зловмисники можуть потрапити до лікарні, та перехоплювати повідомлення. Саме для цього і були розроблені алгоритми шифрування.

Основна ціль цієї роботи – створити та порівняти алгоритми шифрування на прикладі чату.

Сам чат був написаний за допомогою мови програмування python 3.7.4, та за допомогою бібліотеки . Що дозволяє запускати програму на будь-яких пристроях, та потребує малу потужність від клієнта. Програма – кросплатформена, тому що код на мові програмування python можна відкомпілювати під будь-яку систему.

Для реалізації цієї задачі буде порівняно роботу табличних шифрів, блочних шифрів, та шифру, який не піддається дешифруванню.

Також в цій роботі буде створений власний шифр, який в собі поєднує ідею блочних шифрів та реалізацію найвідомішого табличного шифру – шифру Віженера.

На прикладі розробленої програми розглянемо та проаналізуємо вже відомі алгоритми шифрування даних, та порівняємо їх з одним новим алгоритмом.

Для виконання цієї задачі необхідно:

1. Охарактеризувати та розглянути вже відомі алгоритми шифрування.
2. Охарактеризувати та створити новий алгоритм шифрування.
3. Створити тести для деяких алгоритмів.
4. Порахувати асимптотику роботи алгоритмів та протестувати їх в реальних умовах.
5. Порівняти роботу алгоритмів шифрування.
6. Знайти оптимальний варіант для шифрування даних.
7. Охарактеризувати, в яких умовах краще використовувати які алгоритми.

Дане дослідження може бути застосоване при розробці програм, вхідні дані якої – рядок символів, та потрібно його зашифрувати чи дешифрувати.

У шифрі Цезаря кожна літера зсувається на кілька позицій. Наприклад в шифрі Цезаря при зсуві на 2 символи праворуч A стане C, а X набуде значення Z. Шифр Віженера – поліалфавітний шифр, який складається з послідовності кількох шифрів Цезаря із різними зсувами. Для шифровки повідомлень може використовуватись таблиця алфавітів, чи як її ще називають – квадрат (таблиця) Віженера. Розмір (довжина сторони) таблиці Віженера дорівнює кількості символів в алфавіті. Таким чином застосовуючи шифр Віженера до латинського алфавіту вийде квадрат із стороною в 26 символів, тобто буде 26 різних шифрів Цезаря. На кожному етапі шифрування буде застосований окремий шифр Цезаря, засновуючись на літері ключового та секретного слів.

Працює шифр наступним чином. Людина, яка надсилає повідомлення записує ключове слово циклічно, доки довжина його не буде дорівнювати повідомленню. Після цього iй символ в шифротексті буде дорівнювати перетину iго символу в повідомленні та ключовому слові у таблиці.

Щоб розшифрувати текст потрібно у стовбці із iм символом ключа знайти iй символ шифротесту, тож iм символом повідомлення буде перший символ у рядку, який містить iй символ шифротесту.

Перейдемо до криптоаналізу. На відміну від шифру Цезаря, шифр Віженера не можна дешифрувати тільки використовуючи частотний аналіз чи атаку по масці. Але його можна дешифрувати, дізнавшись про довжину його ключа. Індекс співпадінь(тест Касіскі) – один з методів криптоаналізу шифру Віженера. Метод заснований на обчисленні вірогідності того, що дві випадкових частини тексту співпадуть. Цю вірогідність будемо називати індексом співпадінь.

Припустимо рядок х є відкритим текстом або отриманий за допомогою звичайної перестановки. В цьому випадку індекс співпадінь зручно виразити через вірогідності з’явлення і-го символу. Позначимо їх як pі, тоді отримаємо наступну формулу:

Так як величини мають певні значення, то для відкритого тексту індекс співпадінь не залежить від його змісту, а залежить тільки від мови, на якій написаний текст. Більше того, значення вже давно були дослідженні та відомі, що дозволяє розрахувати значення індексу співпадінь відкритого тексту для різноманітних мов.

|  |  |
| --- | --- |
| **Мова** | **Індекс співпадінь** |
| російська | 0.0553 |
| англійська | 0.0644 |
| італійська | 0.0738 |
| іспанська | 0.0775 |
| німецька | 0.0762 |
| французька | 0.0778 |

Якщо – випадковий рядок, тоді вірогідність з’явлення кожного символу дорівнює

Використовуючи першу формулу, отримаємо:

Цією формулою можна керуватися для оцінки індексу співпадінь поліалфавітного шифру. Для англійської мови індекс співпадінь поліалфавітного шифру дорівнює 0.03856, для української (без літери «Ґ») – 0.03125.

Значення індексу співпадінь для відкритого тексту і для поліалфавітного шифру суттєво відрізняються. Це дозволяє, маючи індекс співпадінь, визначити, чи отриманий текст з відкритого звичайною перестановкою, чи є поліалфавітним шифром.

Ще одним важливим поняттям є взаємний індекс співпадінь.

В загальному випадку розглянемо два рядки та із довжинами та відповідно. Алфавіт, як і раніше складається із символів. Взаємним індексом співпадінь цих рядків називають вірогідність того, що взявши по одному випадковому символу із кожного рядку, вони співпадуть. Нехай – кількість -х символів алфавіту в першому та другому рядках відповідно. Тоді взаємний індекс співпадінь буде дорівнювати:

Досить важливим для методу індекс співпадінь є його часний випадок, коли обидві рядки отримані зсувом алфавіту відкритого тексту. Позначимо – вірогідності з’явлення -го символу в рядку , – зсув алфавіту рядка відносно алфавіту рядка (ліворуч). Тоді вірогідності появи -го символу алфавіту в рядку дорівнюють (якщо використовувати нумерацію алфавіту рядка ). Для взаємного індексу співпадінь отримаємо наступну формулу:

Помітимо, що так як зсув циклічний, то і взаємний індекс співпадінь для зсувів і приймає одне й те саме значення. Нижче приведені значення взаємного індексу співпадінь в залежності від зсуву для англійської мови. Значення наведені для зсувів від 0 до . Як згадувалось вище, на основі цих значень взаємний індекс співпадінь може бути вирахуваний для будь-якого зсуву.

|  |  |
| --- | --- |
| **Зсув** | **Взаємний індекс** |
| 0 | 0.0644 |
| 1 | 0.0394 |
| 2 | 0.0319 |
| 3 | 0.0345 |
| 4 | 0.0436 |
| 5 | 0.0332 |
| 6 | 0.0363 |
| 7 | 0.0389 |
| 8 | 0.0338 |
| 9 | 0.0342 |
| 10 | 0.0378 |
| 11 | 0.0440 |
| 12 | 0.0387 |
| 13 | 0.0428 |

Помітимо, що при нульовому зсуві взаємний індекс співпадінь набагато більший, ніж при ненульових зсувах, тож по відомому значенню взаємного індексу співпадінь можна зробити висновок, чи є зсув алфавітів рядків нульовим, або ні.

Розіб’ємо текст на стовбці розміру .

… … … …

Якщо довжині ключа, то кожні два елементи тексту, які відрізняються друг від друга на позицій, , зашифровані одним і тим самим алфавітом. А це означає, що кожний рядок в виписаній вище таблиці отримана з відкритого тексту перестановки. Якщо ж не ділиться на довжину ключа, то рядки є поліалфавітним шифром.

Вище було сказано, що індекс співпадінь для перестановки відкритого тексту та для поліалфавітного шифру суттєво відрізняються. Таким чином перебравши різні значення та вираховуючи для кожного з них індекс співпадінь, можна виділити ті , які ділиться на довжину ключа. Визначити довжину ключа по цим даним буде не важко.

Допустимо, ми визначили довжину ключа . Тепер знайдемо сам ключ. Знову випишемо текст в стовбці розміру .

… … … …

Розглянемо два рядки цієї таблиці. Зсунемо алфавіт одного з рядків на символів та вирахуємо взаємний індекс співпадінь отриманих рядків. Так як кожний з цих двох рядків отриманий зсувом алфавіту відкритого тексту, то максимум взаємного індексу співпадінь буде спостерігатися при нульовому кінцевому відносному зсуві.

Тому використовується наступний алгоритм: вираховується взаємний індекс співпадінь для різних , шукається значення , при якому взаємний індекс співпадінь буде максимальним. Тоді початковий відносний зсув буде дорівнювати ( – розмір алфавіту). Рахуються відносні зсуви між кожною парою рядків. Так як зсуви рядків таблиці відповідають зсувам літер ключа, то залишається перебрати можливих ключей, і вибрати серед них найбільш вдалий.

Для початку роботи шифру Вернама потрібно виписати весь алфавіт, та проставити біля кожного символу унікальний двійковий код(однакової довжини). Після чого, замінити в тексті символи на двійкові коди. Далі потрібно створити ключ. Ключем буде послідовність випадкових двійкових чисел. Їх кількість потрібна дорівнювати кількості символів в тексті.

Вийшло два рядки. В першому написані двійкові числа, якими замінили символи в тексті, а в другому рядку написано стільки ж випадкових двійкових чисел(довжина кожного числа в першому та другому рядках – однакова). Після цього потрібно застосувати операцію до кожної пари -их елементу в першому та другому рядах. Отримані числа й будуть шифротекстом.

У цього шифру є багато недоліків. По перше, для його роботи потрібен ключ, довжина якого дорівнює довжині тексту. По друге, кожен елемент в ключі потрібен бути випадковим(не псевдовипадковим). А, як відомо, будь який алгоритм, що генерує послідовність є псевдовипадковим, тому потрібно щось, дійсно, випадкове. Наприклад, радіоактивний розпад, чи створений електрогенератором білий шум. Також не можна використовувати одні й ті самі випадкові числа кілька разів поспіль. Ще, при великих розмірах тексту потрібно мати велуку кількість випадкових чисел. Наприклад для тексту із ста слів може потребуватись кількасот випадкових чисел.

Для шифру Вернама доведена його абсолютна криптостійкість. Іншими словами, якщо хтось перехопить повідомлення, то єдиний спосіб дізнатися текст – повний перебір ключа. Біля того, цей шифр – єдиний шифр для якого була доведена абсолютна криптостійкість. Але з цього випливає інша проблема. Якщо канал зв’язку прослуховується, то для кожного наступного повідомлення потрібно мати нові випадкові числа обом сторонам, але тоді вони не зможуть їх передати. Натомість, якщо канал зв’язку не прослуховується, то й в шифруванні повідомлень нема ніякого сенсу. Саме тому цей алгоритм припинили використовувати.

DES – симетричний, блочний шифр. Тому для початку потрібно сказати що таке блочні шифри.

Блочні шифри влаштовані досить просто. Для початку потрібно встановити вектор ініціалізації та ключ. За допомогою ключа та вектору ініціалізації формується вектор ініціалізації для наступного блоку, тобто виходить лавинний ефект. І при зміненні будь якого символу зміниться й усе, що йде після нього в кінцевому шифротексті.

Як вже було сказано DES – блочний шифр. Його дані шифруються блоками по 64 біти. Алгоритм працює із кожним 64-бітним блоком явного тексту, а в результаті повертає 64-бітний блок шифрограми. Більше того, як для шифруванн, так і для шифрації кожен раз використовується один й той самий алгоритм(за винятком дещо іншої генерації робочих ключів).

Довжина ключа кожен раз однакова – 56 біт(в джерельному варіанті використовується 64-бітний ключ, тому що кожний восьмий біт – біт паритету, вони можуть бути винесені в останній байт ключа). Ключем може бути довільна 64-бітна комбінація, яка може бути змінена в будь-який момент часу. Деякі з цих комбінацій вважаються слабкими ключами, оскільки можуть бути легко визначені. Безпечність алгоритму на пряму залежить від безпечності ключа.

Перейдемо до алгоритму. На вході подаються 64-бітний блок даних, які переставляються згідно з таблицею:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| 58 | 50 | 42 | 34 | 26 | 18 | 10 | 2 | 60 | 52 | 44 | 36 | 28 | 20 | 12 | 4 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 |
| 62 | 54 | 46 | 38 | 30 | 22 | 14 | 6 | 64 | 56 | 48 | 40 | 32 | 24 | 16 | 8 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 |
| 47 | 49 | 41 | 33 | 25 | 17 | 9 | 1 | 59 | 51 | 43 | 35 | 27 | 19 | 11 | 3 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 | 64 |
| 61 | 53 | 45 | 37 | 29 | 21 | 13 | 5 | 63 | 55 | 47 | 39 | 31 | 23 | 15 | 7 |

Далі йде повторюється наступні операції шістнадцять разів.

Функція розширює 32-бітовий вектор до 48-бітового вектору шляхом повторення деяких біт з згідно з таблицею:

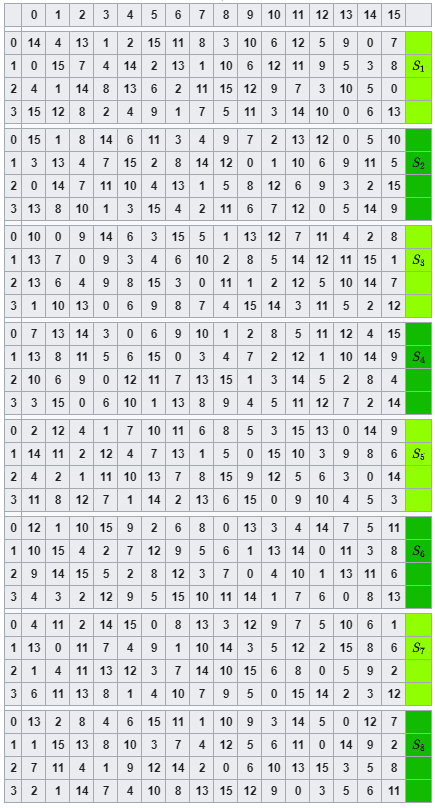
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| \_ | 1 | 2 | 3 | 4 | \_ | \_ | 5 | 6 | 7 | 8 | \_ | \_ | 9 | 10 | 11 | 12 |
| 32 | 1 | 2 | 3 | 4 | 5 | 4 | 5 | 6 | 7 | 8 | 9 | 8 | 9 | 10 | 11 | 12 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| \_ | \_ | 13 | 14 | 15 | 16 | \_ | \_ | 17 | 18 | 19 | 20 | \_ | \_ | 21 | 22 | 23 |
| 13 | 12 | 13 | 14 | 15 | 16 | 17 | 16 | 17 | 18 | 19 | 20 | 21 | 20 | 21 | 22 | 23 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 24 | \_ | \_ | 25 | 26 | 27 | 28 | \_ | \_ | 29 | 30 | 31 | 32 | \_ |
| 24 | 25 | 24 | 25 | 26 | 27 | 28 | 29 | 28 | 29 | 30 | 31 | 32 | 1 |

Перший рядок – номера вхідних біт, другий рядок – вхідні біти. Повторення номерів означає повторення біт.

Блок 48 біт -иться з раундовим ключем .



«Розширені» біти використовуються для визначення номера 0-1-2-3 таблиці (ліва колонка).

Далі виконується перестановка.

Перестановка

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| 16 | 7 | 20 | 21 | 29 | 12 | 28 | 17 | 1 | 15 | 23 | 26 | 5 | 18 | 31 | 10 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 |
| 2 | 8 | 24 | 14 | 32 | 27 | 3 | 9 | 19 | 13 | 30 | 6 | 22 | 11 | 4 | 25 |

За формулою отримуємо значення .

Після 16 раундів перестановка біт виглядає ось так:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| 40 | 8 | 48 | 16 | 56 | 24 | 64 | 32 | 39 | 7 | 47 | 15 | 55 | 23 | 63 | 31 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 |
| 38 | 6 | 46 | 14 | 54 | 22 | 62 | 30 | 37 | 5 | 45 | 13 | 53 | 21 | 61 | 29 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 33 | 34 | 35 | 36 | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 |
| 36 | 4 | 44 | 12 | 52 | 20 | 60 | 28 | 35 | 3 | 43 | 11 | 51 | 19 | 59 | 27 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 49 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 60 | 61 | 62 | 63 | 64 |
| 34 | 2 | 42 | 10 | 50 | 18 | 58 | 26 | 33 | 1 | 41 | 9 | 49 | 17 | 57 | 25 |

Вхідний біт ставить на місце номер 40, другий номер 8 і.т.д.

В DES використовується 16 циклів вихідними даними для кожного з них є біти відкритого тексту(на вході першого раунду), чи 64 біти результату роботи попереднього раунду(у всіх наступних).

Для роботи алгоритму потрібно згенерувати раундові ключі. Ключі отримують з початкового ключа (один символ ASII 8 байтів 64 біт) наступним чином. Вісім бітів, що знаходяться в позиціях 8, 16, 24, 32, 40, 48, 56, 64 додаються в ключ таким чином, щоб кожен байт містив непарне число одиниць.

Початкова перестановка біт ключа



Ця перестановка визначається двома блоками і по 28 біт кожен. Перші 3 біта є біти 57, 49, 41 розширеного ключа. А перші 3 біта є біти 63, 55, 47 розширеного ключа. Для раундів , отримують з одним або двома лівими циклічними зрушеннями згідно з таблицею:

Циклічний зсув

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| -й раунд | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| Число зсувів | 1 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 1 | 2 | 2 | 2 | 2 | 2 | 2 | 1 |

Ключ складається з 48 біт, вибраних з бітів вектору (56 біт) згідно з приведеною нижче таблицею. Перший і другий біти є біти 14, 17 вектору .

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| -й біт раундового ключа | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| Номери біт 3 векторів | 14 | 17 | 11 | 24 | 1 | 5 | 3 | 28 | 15 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| -й біт раундового ключа | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
| Номери біт 3 векторів | 6 | 21 | 10 | 23 | 19 | 12 | 4 | 26 | 8 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| -й біт раундового ключа | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 |
| Номери біт 3 векторів | 16 | 7 | 27 | 20 | 13 | 2 | 41 | 52 | 31 |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| -й біт раундового ключа | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 |
| Номери біт 3 векторів | 37 | 47 | 55 | 30 | 40 | 51 | 45 | 33 | 48 |

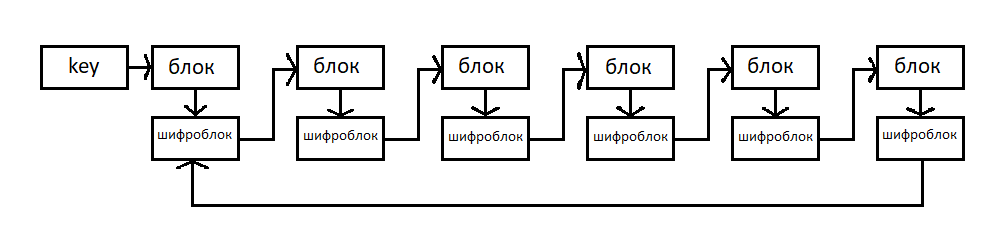
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| -й біт раундового ключа | 37 | 38 | 39 | 40 | 41 | 42 | 43 | 44 | 45 |
| Номери біт 3 векторів | 44 | 49 | 39 | 56 | 34 | 53 | 46 | 42 | 50 |

|  |  |  |  |
| --- | --- | --- | --- |
| -й біт раундового ключа | 46 | 47 | 48 |
| Номери біт 3 векторів | 36 | 29 | 32 |

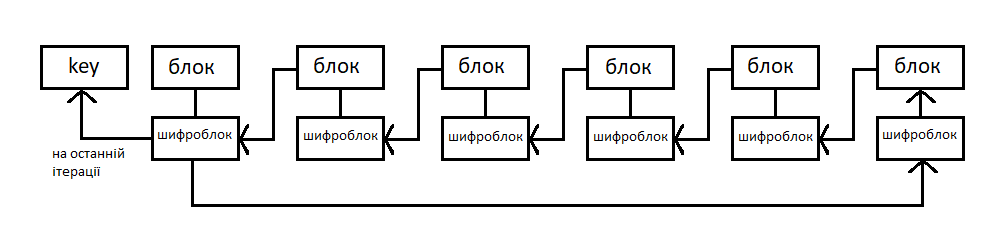
Біти 1-48 використовують для раундового -у.

В цього алгоритму є недолік. Він нестійкий до «лобової» атаки, тобто ключ можна просто підібрати, бо його розмір лише бітів.

Мій шифр поєднує в собі ідею блочних шифрів, та шифр Віженера. Саме тому він біде стійкий до атак, які застосовуються до шифру Віженера. Працює мій шифр досить просто. Для початку формується алфавіт(в мене це латинські символи з до ). Потім зіставляється таблиця Віженера, але перед цим я формую псевдовипадкову послідовність чисел від 1 до 26, і перемішую алфавіт згідно із послідовністю. Після формування таблиці, алгоритм розбиває текст на блоки довжини ключа, та зашифровує текст використовуючи алгоритм Віженера для блоку, як відкритого тексту, і вектору ініціалізації, як для ключа. Коли алгоритм доходить до останнього блоку, він присвоює вектору ініціалізації значення зашифрованого останнього блоку, і далі повторює все с початку ще разів.



Дешифрування проходить за схожим алгоритмом. Шифротекст розбивається на блоки довжини ключа. Потім, використовуючи шифр Віженера, дешифрується шифротекст(блок), використовуючи ключ, як минулий блок. А на останній ітерації цей алгоритм використовує сам ключ, як ключ для дешифрації.



Щоб розшифрувати повідомлення зловмиснику потрібно знати довжину ключа, кількість проходів, та сам ключ. Бо якщо зловмисник буде знати лише ключ, то йому знадобиться багато часу, щоб підібрати кількість проходів. А якщо зловмисник знає лише кількість проходів, йому знадобиться підбирати ключ, що не є гарною ідеєю. Також людина, яка спробує вкрасти повідомлення не зможе дізнатися довжину ключа керуючись методом Касікі, тому що для цього йому буде потрібно знати довжину ключа та кількість проходів.

Цей алгоритм був реалізований на мові програмування python 3.7.4. Користувався я цією мовою програмування тому, що я хотів показати ці алгоритми на прикладі простого консольного чату. Чат був написаний використовуючи бібліотеку socket. Я використовував UDP протокол для зв’язку сервера із клієнтом.

Я вирішив порівнювати алгоритми шифрування за кількома критеріями: часом виконання їхньої роботи, пам’яттю, яку вони потребують та кількістю ключей, що знадобиться для їхньої роботи.

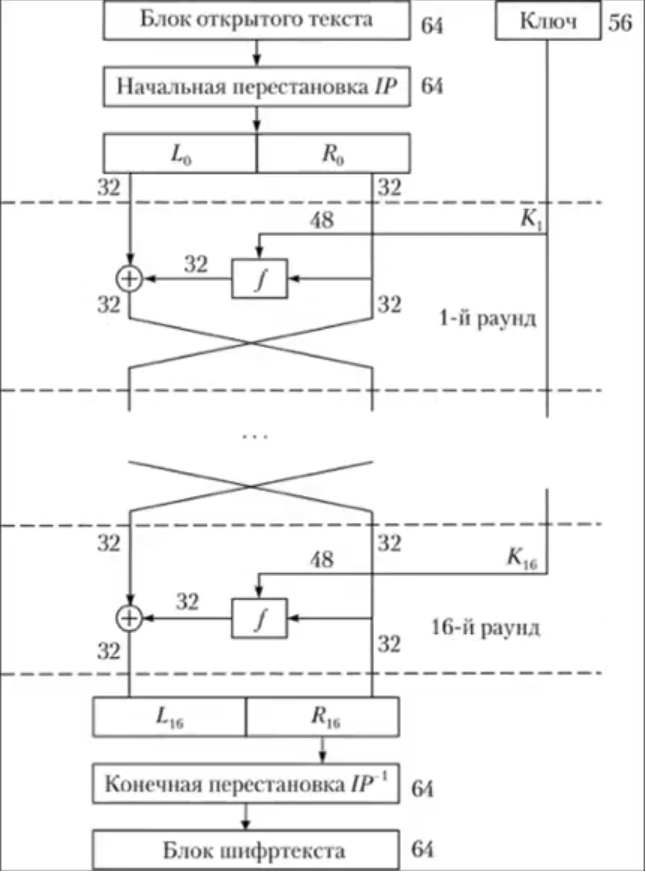
Почнемо з шифру Віженера. На початку своєї роботи він повинен побудувати квадратну таблицю із стороною, яка дорівнює кількості символів в алфавіті. Якщо використовувати тільки латинські символи, то потрібно створити квадрат, із 676 символів, потім потрібно обробити кожен символ вхідного тексту, що робиться за ітерацій. Тобто цей алгоритм працює досить швидко. Натомість в нього є великий недолік – його можна досить легко взломати, тому він програє в цьому питанні всім іншим алгоритмам в даному списку.

Для аналізу я запустив шифр Віженера для рядку довжиною символів разів. Середній показник часу роботи програми – секунди. При довжині рядку в символів, запустивши програму разів середній час роботи програми дорівнює приблизно секунди. А при довжині рядка в символів, при запуску програми разів, середній час роботи програми дорівнював секунди. При збільшенні довжини рядка(більше ніж на символів) час роботи програми буде нелінійно збільшуватися, тому що в мові програмування python 3 чим більше число, тим довше буде проходити операції із ним, тому робити тести на більших рядках немає сенсу. Із вище отриманих даних, можна скласти таку таблицю:

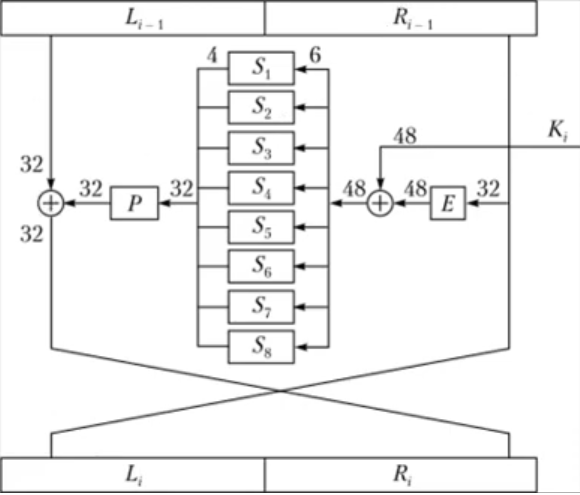
|  |  |  |
| --- | --- | --- |
| Довжина рядку | Кількість запусків програми | Час роботи програми в секундах |
| 10000 символів | 5000 разів | 0.01547 секунди |
| 100000 символів | 500 разів | 0.158 секунди |
| 1000000 символів | 50 разів | 1.751 секунди |

Перейдемо до наступного алгоритму. Для алгоритму Вєрнама, як вже було сказано, доведена абсолютна криптостійкість. Але в нього є великий недолік, а саме те, що для кожного нового повідомлення потрібно обом сторонам мати однакові випадкові числа, а так як потрібно для кожного нового повідомлення мати величезну кількість випадкових чисел, а також обмінятися їми через захищений канал зв’язку, то використовувати цей алгоритм в реальному житті майже неможливо. Тепер кілька слів про його асимптотику. Нехай – час, за який отримується випадкове число. Тоді, якщо довжина повідомлення – символів, то час витраченний на створення випадкових чисел дорівнює , для того, що зробити додавання по модулю 2, потрібно ще ітерацій, тому асимптотика роботи цього алгоритму дорівнює . Також цей алгоритм може займати багато пам’яті, тому що при великому розмірі алфавіту(хоча б символів) потрібно для кожного символу мати 2 числа довжини , де – найближча ступінь двійки більша за кількість символів в алфавіті. І для того, що зберегти шифротекст такого повідомлення знадобиться байтів(де – кількість символів в алфавіті), тому що ще потрібно зберігати чому дорівнює кожен символ алфавіту.

На черзі алгоритм шифрування DES. Якщо говорити про недоліки цього алгоритму, то найважливішим буде довжина його ключа, бо вона дорівнює бітам, тобто можна провести «лобову» атаку на цей алгоритм. Тепер поговоримо про час його роботи. По перше, в самому початку його роботи алгоритм робить перестановку бітів в кожному блоці, тобто робить мінімум ітерацій(на початку роботи із блоком, та у кінці). Після цього алгоритм 16 разів повторює наступну операцію: ділить блок на 2 рівні частини по біти, бере праву частину, та пропускає її через функцію подаючи на вхід ключ розміром 48 біт, після чого лівий блок ксориться із тим, що вийшло з функції, після чого правий блок ставиться без змін на позицію лівого, а лівий(проксорений) опиняється на позиції правого блоку. Але на останній з ітерацій ліва та праві частини не міняються містами. Після чого йде перестановка, яка схожа на ту, що була на початку роботи алгоритму. З цього можна вирахувати асимптотику, яка дорівнює , але ще потрібно сказати за який час працює функція .

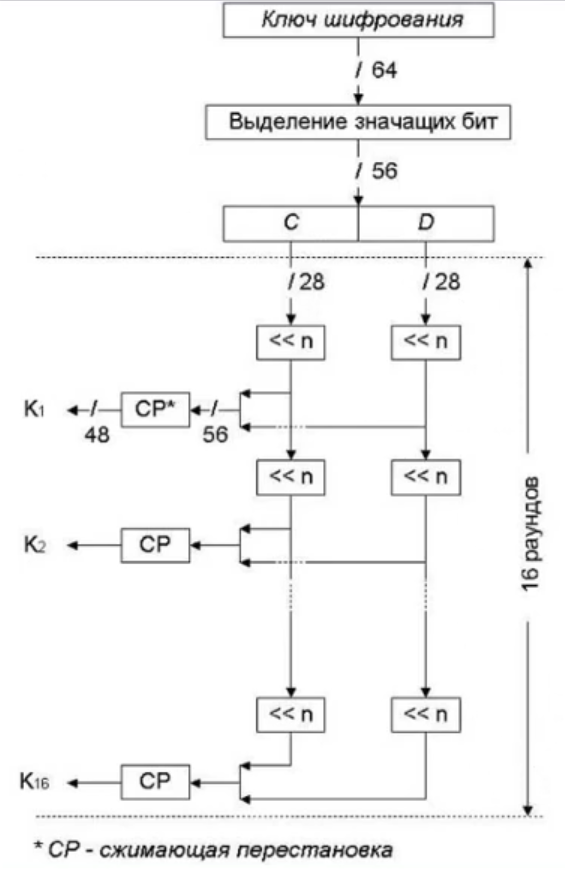


Функція отримує на вхід бітовий блок. Після чого він проходить через функцію розширення, яка повертає 48 бітовий блок. бітовий блок додається до бітового ключа по модулю 2. Після чого отримується новий блок із біт, який ділиться на підблоків по бітів у кожному. Кожен блок проходить через бокс підстановки, який приймає -ти бітовий блок, а повертає -х бітовий блок. Далі кожен -х бітовий блок додаємо, і отримуємо новий бітовий блок, який проходить через перестановку.



Можна сказати, що ця функція працює за лінійний час, бо із кожною ччастиною блока проходять маніпуляції, та в усьому блоку символи змінюють свої позиції. Тому асимптотика роботи функції буде , де – довжина блоку. Тому асимптотика роботи всієї програми становить (коли рахується асимптотика – не враховуються константи, але, звісно, алгоритм буде працювати трохи довше). Ще тут не врахований час побудови ключа, але його і не потрібно враховувати, бо працює ця побудова для кожного раунду за лінійний час.

Ключ для кожного раунду формується наступним чином. Спочатку виділяються значущі біти, тому кожний восьмий біт не враховується, та довжина ключа стає рівною не бітам, як на вході, а бітам. Далі ключ ділиться на дві рівні частини по біт в кожній. Треба зазначити, що виділення бітів – не видалення кожного восьмого, а ще й перестановка. В кожному раунді кожний з бітів у блоках зсуваються на біт ліворуч(при яких значеннях бітів на скільки потрібно зсунути елемент описано в розділі про алгоритм DES, і також там приведена таблиця). Після цього ці блоки склеюються, і входять до стискаючої функції, яка перетворює бітний блок у бітний. І так відбувається раундів, для кожного з яких генерується свій бітний ключ.



Залишилося проаналізувати роботу мого алгоритму. То на скільки важко буде розшифрувати мій алгоритм залежить від кількох факторів. По перше, від кількості проходів по блокам, по друге, від складності й довжини ключа. Час роботи алгоритму залежить тільки від кількості проходів по блоках та довжині відкритого тексту. Якщо довжина тексту дорівнює символів, а кількість проходів – , то асимптотика роботи алгоритму дорівнює , бо в сумі для кожного блоку буде застосований алгоритм Віженера разів, а як було сказано раніше, асимптотика шифру Віженера – лінійна, тож алгоритм буде працювати ітерацій, де – довжина кожного блоку, а – їх кількість. Так як то й асимптотика дорівнює .

Подивимось на результат роботи алгоритму в різних умовах.

|  |  |  |  |
| --- | --- | --- | --- |
|  | 1 прохід | 10 проходів | 100 проходів |
| 1000 символів | 0.0009973 с | 0.00498 с | 0.0428 с |
| 10000 символів | 0.0079779 с | 0.04985 с | 0.4363 с |
| 100000 символів | 0.0797855 с | 0.48044 с | 4.4514 с |
| 1000000 символів | 0.7979397 с | 4.96143 с | 48.784 с |

Як можна побачити, всі результати досить схожі друг на друга. Також можна зауважити, що результат при довжині рядка в 1000 символів при 1 проході не схожий на інші, це можна пояснити, як не точність функції в мові програмування python 3. А на великих тестах результати можуть бути непропорційно більшими, по тим самим причинам чому і в шифрі Віженеру.

Так як в моєму завданні потрібно обрати найкращій шифр для чату, то шифр Вєрнома не підходить, бо постійно потрібно буде генерувати велику кількість випадкових чисел, та обмінюватися їми, що зробити не вийде, якщо канал зв’язку прослуховується. Шифр Віженера також не підходить, тому що вже є способи по дешифрації повідомлень, які зашифровані цим шифром. В шифрі DES недолік в тому, що ключ можна просто підібрати, якщо є багато комп’ютерів із великою потужністю. А в алгоритмі, який написав я інший недолік. Як показує таблиця, коли довжина рядку досить велика, зробити багато проходів по блокам не вийде, тому що програма буде досить довго працювати, натомість із рядками порядку символів та менше, алгоритм показує себе добре. Хоча дивлячись на асимптотику шифру DES, можна побачити, що він на великих рядках буде також досить довго працювати. Але перевага мого алгоритму в тому, що можна змінювати швидкість роботи програми, зменшуючи або збільшуючи кількість проходів по блокам, тобто чим довше буде працювати програма, тим надійніше буде зашифровано повідомлення.

З вищесказаного можна зробити висновок, що алгоритми шифрування дуже важливі. Вони можуть використовуватися, як для захисту власної інформації, так і у воєнному ділі.

В кожного алгоритму шифрування є свої переваги та недоліки. Шифр Віженера працює швидше за всіх інших конкурентів в даному списку, але він піддається до дешифрування. Алгоритм Вєрнама – єдиний алгоритм для якого була доведена абсолютна криптостійкість. Але для його роботи потрібно багато випадкових чисел, тому його використовувати в написанні чату неможливо. Також його використовували на початку двадцятого століття для обміну повідомленнями. Вєрнам розробив машину, яка сама генерує випадкові числа, тому від людини, що відправляє повідомлення нічого не потребувалося, окрім цього пристрою. Алгоритм шифрування DES був стандартом протягом багатьох десятків років, та його криптостійкість залежить від важкості його ключа, натомість його ключ можна підібрати, якщо маєш багато потужності та часу. А мій алгоритм може добре зашифрувати повідомлення, але витратити на це багато часу, або може зробити це гірше, але швидше.

На рядках порядку символів краще використовувати мій алгоритм, тому що алгоритм DES буде дуже довго працювати, а шифр Віженера зашифрує повідомлення не надійно.

На рядках більшої довжини якщо обирати між шифром Віженера та моїм алгоритмом, я виберу мій алгоритм, тому що, навіть, зробивши проходи по блокам мій алгоритм зашифрує повідомлення надійніше, ніж шифр Віженера, а працювати, в цьому випадку, вони будуть приблизно один й той самий час.

На рядках меншої довжини використовувати шифр Віженера взагалі немає сенсу, бо при таких маленьких рядках якщо запустити шифр DES, він буде працювати приблизно той самий час, що і шифр Віженера(у секундах). Але якщо порівнювати який алгоритм лібше використовувати на малих рядках мій чи шифр DES, точно сказати не можна, бо вони обидва добре впораються із поставленим завданням.

Тож зробивши порівняльну характеристику роботи алгоритмів можна сказати, що на великих рядках краще всього використовувати мій алгоритм, а на малих рядках можна використовувати, як мій алгоритм, так і алгоритм шифрування DES.