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.

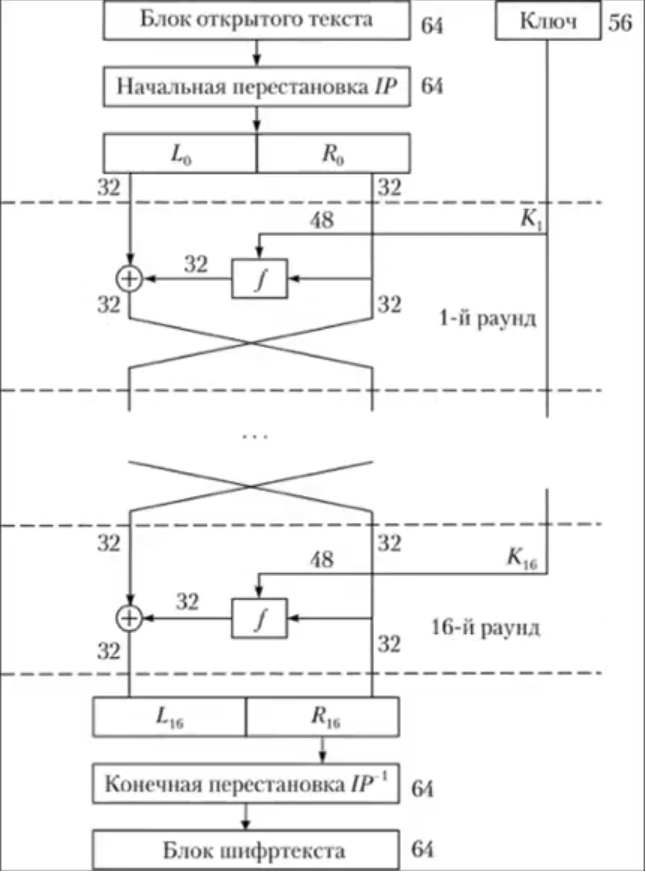
Почнемо з шифру Віженера. На початку своєї роботи він повинен побудувати квадратну таблицю із стороною, яка дорівнює кількості символів в алфавіті. Якщо використовувати тільки латинські символи, то потрібно створити квадрат, із 676 символів, потім потрібно обробити кожен символ вхідного тексту, що робиться 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:

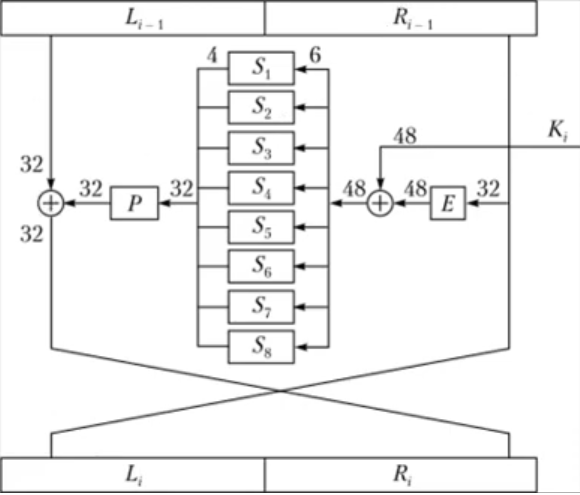
|  |  |  |
| --- | --- | --- |
| Довжина рядку | Кількість запусків програми | Час роботи програми в секундах |
| 10000 символів | 5000 разів | 0.01547 секунди |
| 100000 символів | 500 разів | 0.158 секунди |
| 1000000 символів | 50 разів | 1.751 секунди |

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 .

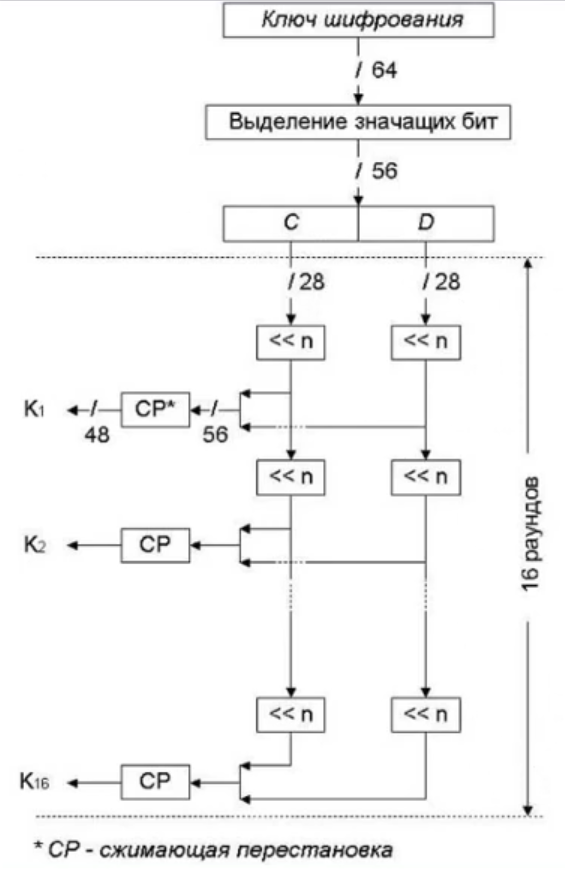


Функція отримує на вхід бітовий блок. Після чого він проходить через функцію розширення, яка повертає 48 бітовий блок. бітовий блок додається до бітового ключа по модулю 2. Після чого отримується новий блок із біт, який ділиться на підблоків по бітів у кожному. Кожен блок проходить через бокс підстановки, 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.