Problem 16: Gdzie są moje klucze?

Punkty: 35

Autor: Marcus Garza, Fort Worth, Teksas, Stany Zjednoczone

Wprowadzenie do problemu

W kryptografii istnieje koncepcja szyfrowania, którą w pewnych warunkach uznaje się za idealną (nie do złamania); jest to tzw. szyfr z kluczem jednorazowym (OTP). Szyfr ten zakłada, że każdy klucz jest tak długi jak sama wiadomość i jest używany tylko raz.

Do słabości szyfru z kluczem jednorazowym należą prawdopodobieństwo, że osoba próbująca złamać szyfr zna zaszyfrowaną informację („leksykon") oraz rozmiar klucza. Ale nawet jeśli osoba próbująca złamać szyfr zna leksykon, to i tak atak brute force - sprawdzający każdą możliwą kombinację - jest praktycznie niewykonalny.

Załóżmy, że próbujemy zaszyfrować wiadomość z kluczem o długości 2^{512} bitów. 2^{512} to OGROMNA liczba:

13 407 807 929 942 597 099 574 024 998 205 846 127 479 365 820 592 393 377 723 561 443 721 764 030 073 546 976 801 874 298 166 903 427 690 031 858 186 486 050 853 753 882 811 946 569 946 433 649 006 084 096

Jeśli osoba łamiąca szyfr chce wypróbować na tej wiadomości atak brute force i byłaby w stanie testować jeden klucz co nanosekundę (0,000000001 sekundy), to sprawdzenie wszystkich kluczy zajęłoby jej 100 trylionów trylionów trylionów trylionów trylionów trylionów trylionów trylionów trylionów (to 9 „trylionów") lat. To wielkość o kilka rzędów większa niż wiek wszechświata i choć ostatecznie ta metoda pozwoliłaby uzyskać poprawną wiadomość, to również jej efektem byłoby uzyskanie wszystkich innych potencjalnie poprawnych wiadomości, co uniemożliwiłoby ustalenie, która odpowiedź jest *właściwa*.

Opis problemu

Wasz program ma zaimplementować szyfr z kluczem jednorazowym. Otrzymacie do niego 128-znakowy łańcuch w układzie szesnastkowym, reprezentujący szyfrowaną wiadomość. Dodatkowo otrzymacie drugi 128-znakowy łańcuch w układzie szesnastkowym, który będzie stanowił klucz do szyfru. Wiadomość niezaszyfrowana składa się z 64 znaków ASCII. Aby zamienić wiadomość zaszyfrowaną na niezaszyfrowaną, należy użyć procesu przedstawionego w poniższej tabeli.

1. Weź dwa kolejne znaki szesnastkowe z wiadomości zaszyfrowanej (np. „4F")
2. Zamień wartość szesnastkową na jej 8-bitowy odpowiednik w układzie dwójkowym (4F = 01001111)
3. Weź dwa kolejne znaki szesnastkowe z klucza (np. „0C")

4. Zamień wartość szesnastkową na jej 8-bitowy odpowiednik w układzie dwójkowym (0C = 00001100)
5. Użyj funkcji XOR (albo; alternatywa rozłączna) dla tych dwóch wartości, aby utworzyć nową liczbę dwójkową
6. Zamień tę nową liczbę dwójkową na wartość dziesiętną ASCII i wyświetl ten znak
7. Powtórzy proces dla reszty wiadomości

Szesnastkowe 4F = Dziesiętne 79 = Dwójkowe 01001111

Szesnastkowe 0C = Dziesiętne 12 = Dwójkowe 00001100

4F	0	1	0	0	1	1	1	1
0C	0	0	0	0	1	1	0	0
XOR	0	1	0	0	0	0	1	1

Dwójkowe 01000011 = Dziesiętne 67 = „C” w układzie ASCII

Jeśli nie znacie funkcji XOR, jest to funkcja boolowska (logiczna) sprawdzający, czy dwie różne wartości logiczne są różne. Jeśli tak, wynikiem jest „true” (1). Jeśli są takie same, wynikiem jest „false” (0).

XOR	0	1
0	0	1
1	1	0

Większość języków programowania pozwala użyć funkcji XOR do dwóch liczb, aby utworzyć nową liczbę; te operacje „bitowe” służą do wykonania porównania XOR na każdym bicie dwójkowej reprezentacji tych liczb, podobnie jak wykonaliśmy w powyższym przykładzie: 79 XOR 12 = 67. W językach Java, C, C++ i Python służy do tego kareta (^), a w języku VB.NET operator `Xor`.

Przykładowe dane wejściowe

Pierwszy wiersz danych wejściowych waszego programu, **otrzymanego przez standardowe wejście**, będzie zawierać dodatnią liczbę całkowitą oznaczającą liczbę przypadków testowych. Każdy przypadek testowy będzie zawierać następujące wiersze:

- Wiersz zawierający dodatnią liczbę całkowitą, **X**, reprezentującą liczbę używanych kluczy
- Wiersz zawierający 128-znakowy łańcuch w układzie szesnastkowym, reprezentujący zaszyfrowaną wiadomość
- **X** wierszy, z których każdy zawiera 128-znakowy łańcuch w układzie szesnastkowym, reprezentujący klucz

```
1
2
4F6F0E14089E040E286156061893404F658D1F6510D5744098DB1DF8904D5F0DF23710
D30230F4F985D4FAAE50F4984AF40B4C98F70E98F94998F043DF16D89F
0C006A7128CF716B5B15766F6BB3263A0BAC3F227FBA1060F4AE7E93B0393069934E31
F3515F988FE0F48EC63F87FD6A847923FA9B6BF58A68B8D063FF36F8BF
1B07676728EE686F410F226360E7602704FE3F1178B05433F9B678D8F3242F65974564
B67A44D49BF0A0DACF7090F12C926E3EFD997AB8922CE1DE639E5799DE
```

(Należy pamiętać, że powyżej znajdują się tylko trzy wiersze tekstu w układzie szesnastkowym; są zawinięte, bo nie mieszczą się na stronie)

Przykładowe dane wyjściowe

W każdym przypadku testowym wasz program powinien wyświetlić 64-znakową odszyfrowaną wiadomość uzyskaną przez odszyfrowanie zaszyfrowanej wiadomości za pomocą przekazanego klucza - jedną dla każdego wiersza. Wiadomość odszyfrowana musi być umieszczona w nawiasach kwadratowych. Może zawierać na końcu łańcucha dodatkowe spacje, ale muszą mieścić się wewnątrz nawiasów.

```
[Code Quest is fun! Good luck today! Solve those problems!       ]
[This plaintext has the same ciphertext but a different key. AAAA]
```