helion.pl
KSIĘGARNIA INTERNETOWA · WYDAWNICTWO

IDŹ DO
PRZYKŁADOWY ROZDZIAŁ
SPIS TREŚCI

KATALOG KSIĄŻEK
KATALOG ONLINE
ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK
DODAJ DO KOSZYKA

CENNIK I INFORMACJE
ZAMÓW INFORMACJE O NOWOŚCIACH
ZAMÓW CENNIK

CZYTELNIA
FRAGMENTY KSIĄŻEK ONLINE

Hakerskie łamigłówki



Autor: Ivan Sklyarov
Tłumaczenie: Radosław Meryk
ISBN: 83-246-0422-7
Tytuł oryginału: Puzzles for Hackers
Format: B5, stron: 272

Sprawdź i rozwiń swoje umiejętności

- Kryptoanaliza
- Zabezpieczanie aplikacji
- Włamania do systemu Windows

Termin „haker" nadal jest powszechnie kojarzony z komputerowym przestępcą, kradnącym dane i wyrządzającym ogromne szkody. Jednak coraz częściej znajomość hakerskich sztuczek przydaje się również informatykom stojącym po właściwej stronie barykady. Programiści, specjaliści od zabezpieczeń i administratorzy systemów wykorzystują tę wiedzę do zapobiegania atakom. Poznanie taktyki przeciwnika jest w tym wypadku najlepszym rozwiązaniem.

„Hakerskie łamigłówki" to zbiór zagadek i problemów związanych z szeroko pojętym bezpieczeństwem sieci, aplikacji i danych. Ich rozwiązywanie dostarcza nie tylko rozrywki, ale przede wszystkim uczy myśleć innymi kategoriami. Poznasz niekonwencjonalne sposoby radzenia sobie ze znanymi problemami i przekonasz się, że znajomość technik ataków hakerskich pozwala również zabezpieczać się przed nimi. W pierwszej części książki znajdują się zagadki, podzielone na kategorie, w drugiej – rozwiązania.

- Kryptoanaliza i łamanie haseł
- śledzenie pakietów i przechwytywanie danych
- Tworzenie bezpiecznych aplikacji
- Zabezpieczanie skryptów CGI i PHP
- Pisanie eksploitów
- Inżynieria wsteczna

Spójrz na problemy bezpieczeństwa z innej perspektywy

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

Przedmowa **7**

Część I Łamigłówki **9**

Rozdział 1.1. Łamigłówki z dziedziny kryptoanalizy **11**
1.1.1. Cool Crypto 11
1.1.2. Gil Bates i Cool Crypto 12
1.1.3. Korespondencja gwiazd 13
1.1.4. Algorytm Rot13 14
1.1.5. Eliminacja „poszukiwaczek złota” 14
1.1.6. Ataki siłowe i lamerzy 15
1.1.7. Admin Monkey 15
1.1.8. Dwa kolejne generatory haseł 16
1.1.9. Słynne zdanie 16
1.1.10. Tajna wiadomość 17
1.1.11. Kandydat do pracy 17
1.1.12. Inny kandydat do pracy 17
1.1.13. Gil pisze do Minusa 18
1.1.14. Minus odpowiada Gilowi 18
1.1.15. Sejf na łapę królika 18
1.1.16. Hey Hacker! 19
1.1.17. Dowcipny kryptolog 19

Rozdział 1.2. Łamigłówki sieciowe **21**
1.2.1. Odwzorowanie warstw modelu OSI 21
1.2.2. Skuteczny sniffing 22
1.2.3. Sniffing haseł 24
1.2.4. Jaki był wynik śledzenia pakietów? 25
1.2.5. W jaki sposób oszukać PGP? 26
1.2.6. Ostrzeżenia programu tcpdump 27

Rozdział 1.3. Łamigłówki związane z systemem Windows **43**
1.3.1. Nieopierzony nauczyciel informatyki walczy z wirusami 43
1.3.2. Pliki, które nigdy nie giną 44
1.3.3. Przypadek brakującego miejsca na dysku 44
1.3.4. Tajemniczy błąd systemowy 44
1.3.5. Cracking „gołymi rękami” 45

Rozdział 1.4. Łamigłówki związane z kodowaniem 47
1.4.1. Hakerskie kryptorytmy 47
1.4.2. Optymalizacja programu do obliczania ciągu Fibonacciego 48
1.4.3. Tępe ostrza 48
1.4.4. Program, który wyświetla swój kod 50
1.4.5. Program dwujęzyczny 50
1.4.6. Praktyczne kodowanie 51
1.4.7. Odwrotny cracking 52
1.4.8. Wygłupy z dyrektywą #define 52
1.4.9. Proste równanie 53
1.4.10. Pozbądź się instrukcji IF 53
1.4.11. Układ logiczny 53
1.4.12. Gwiazda logiczna 54
1.4.13. Optymalizacja w języku C 55
1.4.14. Optymalizacja dla miłośników asemblera 55

Rozdział 1.5. Bezpieczne programowanie 57
1.5.1. Łamigłówki dla „skrypciarzy” 57
1.5.2. Hasło do osobistych tajemnic 61
1.5.3. Błędy w skryptach CGI 61
1.5.4. Błędy w skryptach PHP 63
1.5.5. Szpieg w pliku CORE 64
1.5.6. Dr Jekyll i Mr Hyde 65
1.5.7. Zalecenia „eksperta” 65
1.5.8. Tajemniczy ciąg znaków: wersja 1 66
1.5.9. Tajemniczy ciąg znaków: wersja 2 66
1.5.10. Tajemniczy ciąg znaków: wersja 3 66
1.5.11. Doskonały eksploit: wersja 1 67
1.5.12. Doskonały eksploit: wersja 2 67
1.5.13. Doskonały eksploit: wersja 3 68
1.5.14. Błędy na deser 69
1.5.15. Znieważenie konta root 70
1.5.16. Kto jest kim? 70

Rozdział 1.6. Łamigłówki związane z technikami inżynierii wstecznej 71
1.6.1. Pięć razy „Cool Hacker!” 72
1.6.2. Dzień dobry, lamerze! 72
1.6.3. Kocham Windows! 73
1.6.4. Modyfikacja bitów 73
1.6.5. Niechże odpowie „OK!” 74
1.6.6. He, he, he... 74
1.6.7. Złam mnie! 75
1.6.8. Back in the USSR 76
1.6.9. Figury 76
1.6.10. Znajdź licznik 77
1.6.11. CD Crack 78
1.6.12. St. Petersburg 79
1.6.13. Woda 79

Rozdział 1.7. Różne łamigłówki 81
1.7.1. Obrazki bez obrazków 81
1.7.2. Sfabrykowany zrzut 83
1.7.3. Czyje to logo? 84
1.7.4. Skąd pochodzi klawiatura? 86
1.7.5. Kryptorytm 86

1.7.6. Przypomnienie 86
1.7.7. Co to za książki? 87
1.7.8. Trudne pytania 87

Część II Rozwiązania 89

Rozdział 2.1. Łamigłówki z dziedziny kryptoanalizy 91
2.1.1. Cool Crypto 91
2.1.2. Gil Bates i Cool Crypto 93
2.1.3. Korespondencja gwiazd 94
2.1.4. Algorytm Rot13 98
2.1.5. Eliminacja „poszukiwaczek złota” 99
2.1.6. Ataki siłowe i lamerzy 99
2.1.7. Admin Monkey 100
2.1.8. Dwa kolejne generatory haseł 100
2.1.9. Słynne zdanie 101
2.1.10. Tajna wiadomość 101
2.1.11. Kandydat do pracy 102
2.1.12. Inny kandydat do pracy 102
2.1.13. Gil pisze do Minusa 103
2.1.14. Minus odpowiada Gilowi 104
2.1.15. Sejf na łapę królika 105
2.1.16. Hey Hacker! 105
2.1.17. Dowcipny kryptolog 106

Rozdział 2.2. Łamigłówki sieciowe 109
2.2.1. Odwzorowanie warstw modelu OSI 109
2.2.2. Skuteczny sniffing 110
2.2.3. Sniffing haseł 113
2.2.4. Jaki był wynik śledzenia pakietów? 113
2.2.5. W jaki sposób oszukać PGP? 115
2.2.6. Ostrzeżenia programu tcpdump 117

Rozdział 2.3. Łamigłówki związane z systemem Windows 123
2.3.1. Nieopierzony nauczyciel informatyki walczy z wirusami 123
2.3.2. Pliki, które nigdy nie giną 126
2.3.3. Przypadek brakującego miejsca na dysku 127
2.3.4. Tajemniczy błąd systemowy 129
2.3.5. Cracking „gołymi rękami” 130

Rozdział 2.4. Łamigłówki związane z kodowaniem 133
2.4.1. Kryptorytm ze słowem HACKER 133
2.4.2. Optymalizacja programu do obliczania ciągu Fibonacciego 135
2.4.3. Tępe ostrza 135
2.4.4. Program, który wyświetla swój kod 135
2.4.5. Program dwujęzyczny 137
2.4.6. Praktyczne kodowanie 138
2.4.7. Odwrotny cracking 139
2.4.8. Wygłupy z dyrektywą #define 143
2.4.9. Proste równanie 143
2.4.10. Pozbądź się instrukcji IF 144
2.4.11. Układ logiczny 144
2.4.12. Gwiazda logiczna 150
2.4.13. Optymalizacja w języku C 152
2.4.14. Optymalizacja dla miłośników asemblera 152

Rozdział 2.5. Bezpieczne programowanie 155
2.5.1. Łamigłówki dla „skrypciarzy” 155
2.5.2. Hasło do osobistych tajemnic 156
2.5.3. Błędy w skryptach CGI 160
2.5.4. Błędy w skryptach PHP 161
2.5.5. Szpieg w pliku CORE 164
2.5.6. Dr Jekyll i Mr Hyde 165
2.5.7. Zalecenia „eksperta” 167
2.5.8. Tajemniczy ciąg znaków: wersja 1 167
2.5.9. Tajemniczy ciąg znaków: wersja 2 171
2.5.10. Tajemniczy ciąg znaków: wersja 3 175
2.5.11. Doskonały eksploit: wersja 1 177
2.5.12. Doskonały eksploit: wersja 2 193
2.5.13. Doskonały eksploit: wersja 3 195
2.5.14. Błędy na deser 197
2.5.15. Znieważenie konta root 200
2.5.16. Kto jest kim? 202

Rozdział 2.6. Łamigłówki związane z technikami inżynierii wstecznej 205
2.6.1. Pięć razy „Cool Hacker!” 205
2.6.2. Dzień dobry, lamerze! 209
2.6.3. Kocham Windows! 212
2.6.4. Modyfikacja bitów 215
2.6.5. Niechże odpowie „OK!” 216
2.6.6. He, he, he… 219
2.6.7. Złam mnie! 225
2.6.8. Back in the USSR 235
2.6.9. Figury 240
2.6.10. Znajdź licznik 245
2.6.11. CD Crack 249
2.6.12. St. Petersburg 252
2.6.13. Woda 256

Rozdział 2.7. Różne łamigłówki 261
2.7.1. Obrazki bez obrazków 261
2.7.2. Sfabrykowany zrzut 263
2.7.3. Czyje to logo? 263
2.7.4. Skąd pochodzi klawiatura? 264
2.7.5. Kryptorytm 264
2.7.6. Przypomnienie 264
2.7.7. Co to za książki? 265
2.7.8. Trudne pytania 265

Dodatki 267

Dodatek A Zalecana lektura 269

Skorowidz 271

Rozdział 2.1.

Łamigłówki z dziedziny kryptoanalizy

2.1.1. Cool Crypto

Smithnik zauważył, że zaszyfrowany tekst ma tyle samo znaków co tekst niezaszyfrowany. Było to dla niego oczywistym sygnałem, że ma do czynienia z algorytmem odwracalnym. W istocie „algorytm zastrzeżony" był jedynie banalnym szyfrowaniem XOR.



XOR to notacja dla operacji różnicy symetrycznej. Tablicę prawdy dla tej operacji zaprezentowano w tabeli 2.1.1.

Tabela 2.1.1. *Tablica prawdy operacji XOR*

A	B	A XOR B
0	0	0
0	1	1
1	0	1
1	1	0

Na przykład kod ASCII (*American Standard Code for Information Interchange*) litery *A* to 41H, co odpowiada liczbie dwójkowej 100001, natomiast kod ASCII cyfry 1 to 31H, czyli dwójkowo 110001. Wykonanie operacji XOR dla tych dwóch kodów tworzy liczbę 1110000, co odpowiada kodowi ASCII litery *p*.

```
A=1000001
XOR
1=0110001
---------
p=1110000
```

Na tym polega zasada szyfrowania XOR.

Smithnik bez trudu rozpoznał, że w programie zastosowano algorytm kodowania XOR, ponieważ znał jedną ważną właściwość tej operacji:

```
X XOR Klucz = Y
Y XOR Klucz = X
X XOR Y = Klucz
```

W związku z tym wystarczy znać zaszyfrowany tekst i przynajmniej część tekstu niezaszyfrowanego, aby uzyskać klucz szyfrowania. Dlatego właśnie algorytm jest odwracalny.

Klucz zastosowany w programie Cool Crypto można uzyskać za pomocą prostego programu w języku C. Jego kod źródłowy zamieszczono na listingu 2.1.1.

Listing 2.1.1. *Kod źródłowy programu xorer.exe*

```
#include <stdio.h>
int main (int argc, char *argv[])
{
    FILE *in, *out;
    char *key;
    int byte;
    if (argc !=4) {
        printf ("Sposób użycia: xorer <klucz> <plik_we> <plik_wy>\n");
        return 1;
    }
    key = argv[1];
    if ((in = fopen(argv[2], "rb")) != NULL) {
        if ((out = fopen(argv[3], "wb")) != NULL) {
            while ((byte = getc(in)) != EOF)
            {
            if (!*key) key = argv[1];
                byte^= *(key++);
            putc(byte,out);
       }
    fclose(out); }
    fclose(in); }
    return 0;
}
```

Program wykonuje operację XOR na znakach w pliku określonym przez argument wiersza polecenia *<plik_we>* oraz z wykorzystaniem klucza określonego argumentem *<klucz>*. Wynik jest zapisywany w pliku określonym przez argument *<plik_wy>*. Wprowadzenie ciągu `creature_creature_creature` w miejsce argumentu *<klucz>* oraz umieszczenie w pliku wejściowym ciągu `]VTYJQC]aGC]_PDJ[{RJ[EEMLA` daje wynik w pliku wyjściowym `>$18>$18>$18>$18>$18>$18>$`. Aby uzyskać ten sam wynik, można również użyć niezaszyfrowanego tekstu jako klucza, a klucz umieścić w pliku wejściowym. Własność ta wynika z odwracalności operacji XOR, którą objaśniliśmy wcześniej.

A zatem klucz zastosowany w programie Cool Crypto składa się z zaledwie czterech znaków ASCII: >$18. Program sekwencyjnie stosuje klucz dla każdego wiersza wejściowego i w ten sposób go szyfruje:

```
Wejście = creature_creature_creature
XOR
Klucz   = >$18>$18>$18>$18>$18>$18>$
-----------------------------------
Wyjście = ]VTYJQC]aGC]_PDJ[{RJ[EEMLA
```

Zaszyfrowanie słowa *Smith* z wykorzystaniem klucza >$18 daje wynik mlXLV. Można to sprawdzić za pomocą tego samego programu *xorer.exe* (listing 2.1.1).

Jeśli ktoś uważa, że przesadziłem, pisząc, że program, w którym zastosowano proste kodowanie XOR, kosztuje 1000 USD, myli się. Na rynku były (i nadal są) programy, których cena kształtuje się w tych granicach, pomimo tego, że zastosowano w nich prymitywne „zastrzeżone algorytmy szyfrowania".

2.1.2. Gil Bates i Cool Crypto

W odróżnieniu od poprzedniej łamigłówki (patrz punkt 1.1.1), teraz Smithnik nie dysponuje fragmentem niezaszyfrowanego tekstu. To jednak nie stanowi wielkiego problemu. Wystarczy wypróbować dowolny tekst lub frazę, które najprawdopodobniej znajdują się w tekście, na przykład słowa *with*, *and*, *this is* czy też *I am*. Zadanie Smithnika ułatwia również znajomość nazwy pliku tekstowego: często pliki tekstowe rozpoczynają się od nazwy pliku. Spróbuj wykonać operację XOR na zawartości pliku *The conscience of a Hacker.txt*, wykorzystując klucz w postaci nazwy pliku. Do tego celu można wykorzystać program *xorer.exe*, który znajduje się pod adresem: *ftp://ftp.helion.pl/przyklady/haklam.zip*, w folderze *\Czesc2\Rozdział1\1.1*.

W wierszu polecenia wpisz następującą instrukcję:

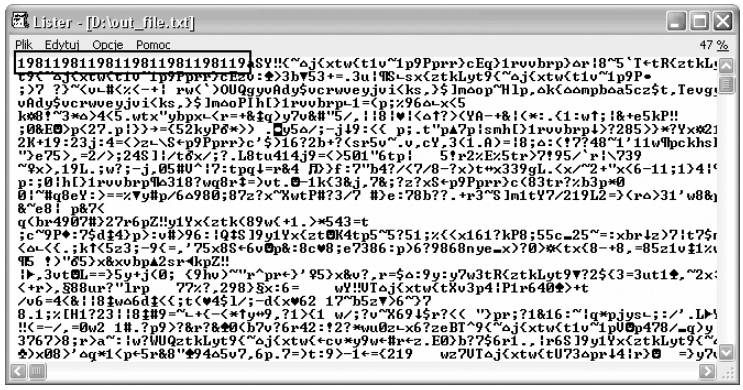
```
>xorer.exe "The conscience of a Hacker" "The conscience of a Hacker.txt"
out_file.txt
```

W książce instrukcję podzielono na dwie części, ponieważ nie mieści się w jednym wierszu, ale w rzeczywistości należy ją wprowadzić w całości.

Nazwę pliku należy ująć w cudzysłów, tak aby program nie interpretował jej jako kilku argumentów rozdzielonych spacjami. Zwróćmy uwagę, że pierwszy parametr ujęty w cudzysłów to klucz (nazwa pliku bez rozszerzenia *.txt*), natomiast drugi to nazwa pliku wraz z rozszerzeniem *.txt*.

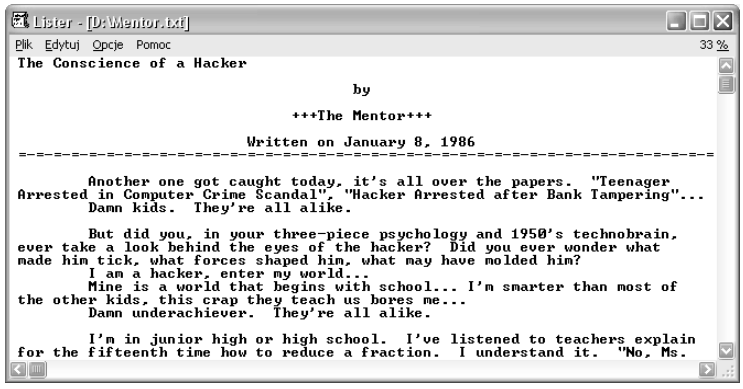
Wynik — powtarzającą się wartość *1981* — można zobaczyć na początku pliku *out_file.txt* (rysunek 2.1.2 a).

Rysunek 2.1.2 a. *Zawartość pliku out_file.txt*



Istnieje prawdopodobieństwo, że jest to tajny klucz Gila, ale równie dobrze może to być zwykły zbieg okoliczności. Można to łatwo sprawdzić. Wystarczy zastosować ciąg *1981* jako maskę XOR dla zaszyfrowanego tekstu. W wyniku tej operacji uzyskamy tekst przed zaszyfrowaniem. Jak można zauważyć na rysunku 2.1.2 b, okazało się, że jest to popularny manifest autorstwa hakera o pseudonimie The Mentor, opublikowany w czasopiśmie *Phrack* (*http://www.phrack.org/phrack/7/P07-03*).

Rysunek 2.1.2 b. *„Manifest hakerów" The Mentora*



Zastanawia mnie, po co Gil przechowywał zaszyfrowany „Manifest hakerów" na swoim serwerze? Prawdopodobnie chciał sobie z nich zażartować.

2.1.3. Korespondencja gwiazd

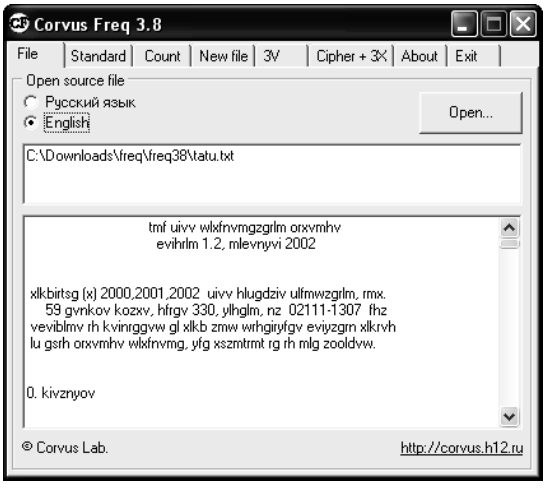
Jeśli ktoś czytał opowiadanie Arthura Conana Doyle'a *Tańczące sylwetki*, z pewnością pamięta, że każda figura z patyków odpowiadała określonej literze. Kod wykorzystywany przez Pritney i Lustina w ich korespondencji bazował na tej samej zasadzie, tyle że zamiast figur z patyków do reprezentacji liter wykorzystywali oni inne litery. Wiadomości zakodowane w taki sposób odszyfrowuje się z wykorzystaniem metod **analizy częstości** znanych od ponad tysiąca lat. Ich twórcą był znany arabski naukowiec z IX stulecia Abu Yousuf Yaqub Ibn Ishaq al-Kindi lub po prostu al-Kindi. Oto jego opis metody szyfrowania:

Jest sposób odczytania zaszyfrowanego komunikatu napisanego w znanym języku. Aby to zrobić, trzeba wziąć około strony jawnego tekstu w tym języku, policzyć w nim wszystkie litery i określić częstość występowania poszczególnych liter. Literę, która występuje najczęściej, nazwiemy „pierwszą", następną w kolejności „drugą" itd., aż do uporządkowania wszystkich liter w alfabecie. Następnie należy wziąć zakodowany tekst i policzyć w nim wszystkie znaki. Podobnie jak w przypadku jawnego tekstu, należy określić literę „pierwszą" (występującą najczęściej), „drugą", „trzecią" itd. „Pierwsza" litera zakodowanego tekstu odpowiada „pierwszej" literze tekstu jawnego, „druga" litera zakodowanego tekstu odpowiada „drugiej" literze tekstu jawnego itd.

Przybliżone częstości występowania liter dla praktycznie wszystkich języków stworzono już bardzo dawno temu (patrz tabela 2.1.3 a i b). W związku z tym, aby odszyfrować list, wystarczy wyznaczyć częstość liter w zakodowanym liście i zastąpić w nim każdy znak literą alfabetu o tej samej częstości. To wszystko!

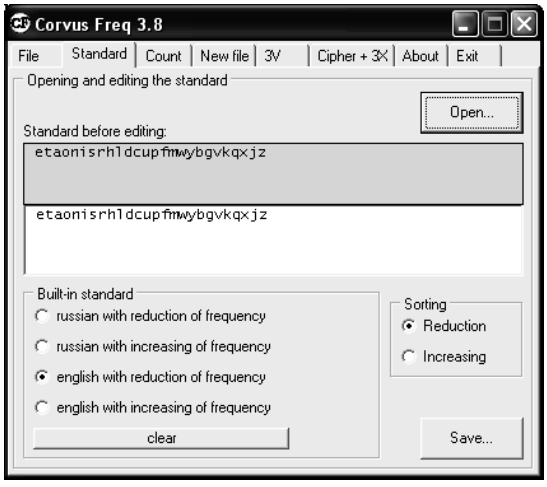
✓ Względną częstość występowania litery określa się poprzez podzielenie liczby jej wystąpień w tekście przez liczbę wszystkich znaków w tekście.

Z wyjątkiem bardzo krótkich wiadomości proces wyznaczania częstości liter jest dość żmudny, a zatem najlepiej powierzyć to zadanie specjalizowanemu narzędziu, które można znaleźć w internecie lub napisać samodzielnie. Osobiście polecam wykorzystanie do tego celu doskonałego programu pod nazwą *Freq* (od *frequency* — częstotliwość). Można go pobrać pod adresem *http://corvus.h12.ru*. Najpierw należy otworzyć zaszyfrowany list w programie (rysunek 2.1.3 a), następnie kliknąć przełącznik *English with reduction of frequency* na zakładce *Standard*. W oknie edycyjnym wyświetli się lista liter angielskiego alfabetu posortowana malejąco według częstości występowania (rysunek 2.1.3 b).



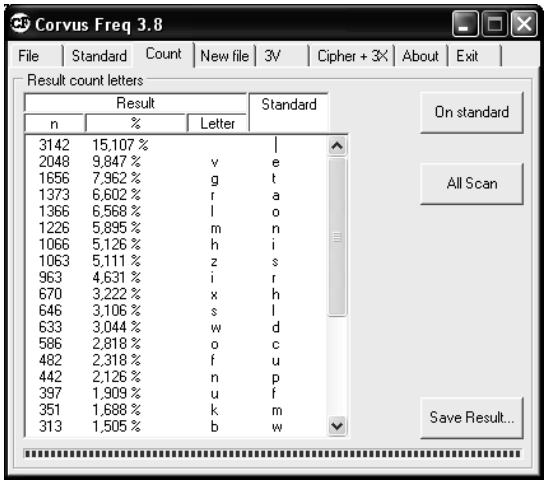
Rysunek 2.1.3 a. *Zaszyfrowany list po otwarciu w programie Freq*

Rysunek 2.1.3 b. *Względna częstość występowania liter alfabetu angielskiego*



Następnie należy przejść do zakładki *Count* i kliknąć przycisk *On standard*. Wyświetlą się względne częstości liter w zaszyfrowanym liście w porównaniu z częstościami standardowymi (rysunek 2.1.3 c).

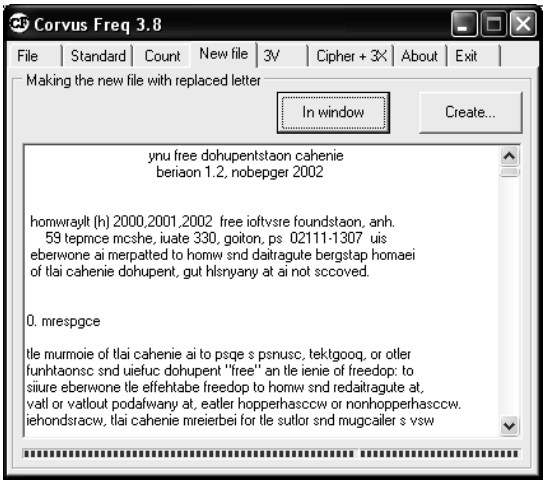
Rysunek 2.1.3 c. *Względne częstości liter w zaszyfrowanym tekście odwzorowane na częstości standardowe*



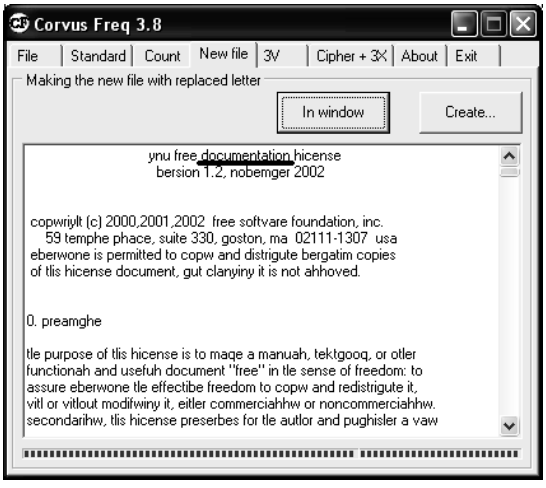
Następnie należy przejść do zakładki *New file* i kliknąć przycisk *In window*. Wyświetli się częściowo odszyfrowany list (rysunek 2.1.3 d).

Jak można zauważyć, wiele słów, zwłaszcza krótkich (np. *free*), już można rozpoznać. Pomimo to trudno zrozumieć taki tekst, dlatego trzeba go poprawić. Bez wielkiego trudu można wywnioskować, że słowo *dohupentstaon* to po prostu *documentation*, dlatego przestawimy miejsca liter *h* i *c* na zakładce *Standard*, po czym ponownie przeprowadzimy obliczenia na zakładce *Count*. W podobny sposób należy zamieniać miejscami kolejne litery do czasu, kiedy słowo *dohupentstaon* przyjmie postać *documentation* (rysunek 2.1.3 e).

Rysunek 2.1.3 d. *Częściowo odszyfrowany list*



Rysunek 2.1.3 e. *Poprawiony odszyfrowany tekst*



Jak można zauważyć, tekst jest teraz o wiele bardziej czytelny. Niektóre zwroty, a nawet całe zdania, zostały całkowicie odszyfrowane.

Można kontynuować zastępowanie miejscami poszczególnych liter tak, by tekst stał się jeszcze bardziej czytelny, jednak już na tym etapie można powiedzieć, że gwiazdy przesyłały do siebie licencję GNU darmowej dokumentacji. Jej kompletny tekst dostępny jest pod adresem *http://www.gnu.org/copyleft/fdl.html*.

Od razu widać, dlaczego dla Smithnika była to bułka z masłem.

Do zakodowania licencji użyto następujących podstawień:

```
a - z   g - t   m - n   s - h   y - b
b - y   h - s   n - m   t - g   z - a
c - x   i - r   o - l   u - f
d - w   j - q   p - k   v - e
e - v   k - p   q - j   w - d
f - u   l - o   r - i   x - c
```

Tabela 2.1.3 a. *Względna częstość liter alfabetu rosyjskiego*

Litera	Względna częstość	Litera	Względna częstość	Litera	Względna częstość
а	0,062	л	0,035	ц	0,004
б	0,014	м	0,026	ч	0,012
в	0,038	н	0,053	ш	0,006
г	0,013	о	0,090	щ	0,003
д	0,025	п	0,023	ъ	0,014
е	0,072	р	0,040	ы	0,016
ж	0,007	с	0,045	ь	0,014
з	0,016	т	0,053	э	0,003
и	0,062	у	0,021	ю	0,006
й	0,010	ф	0,002	я	0,018
к	0,028	х	0,009	Spacja	0,174

Tabela 2.1.3 b. *Względna częstość liter alfabetu angielskiego*

Litera	Względna częstość	Litera	Względna częstość	Litera	Względna częstość
a	0,0804	b	0,0154	c	0,0306
d	0,0399	e	0,1251	f	0,0230
g	0,0196	h	0,0549	i	0,0726
j	0,0016	k	0,0067	l	0,0414
m	0,0253	n	0,0709	o	0,0760
p	0,00200	q	0,0011	r	0,0612
s	0,0654	t	0,0925	u	0,0271
v	0,0099	w	0,0192	x	0,0019
y	0,00173	z	0,0009	Spacja	0,1500

2.1.4. Algorytm Rot13

Przesunięcie o 13 pozycji wybrano nieprzypadkowo i nie z powodów estetycznych. Ze względu na to, że w alfabecie angielskim jest 26 liter, zastosowanie algorytmu `rot13` do zaszyfrowanego tekstu (tzn. wykonanie funkcji `rot13(rot13(`*`ciąg`*`))`) powoduje odszyfrowanie tekstu. Nie można tego zrobić przy przesunięciu liter o żadną inną liczbę (np. `rot12` lub `rot5`). Oczywiście dotyczy to wyłącznie alfabetów składających się z 26 liter (lub wielokrotności 26), tak jak alfabet angielski. Tak więc rosyjskiego komunikatu zaszyfrowanego z wykorzystaniem algorytmu `rot13` nie można odszyfrować za pomocą tego algorytmu, ponieważ alfabet rosyjski składa się z 33 liter.

2.1.5. Eliminacja „poszukiwaczek złota”

Gil podyktował kandydatce na sekretarkę następujący tekst:

```
The sex life of the woodchuck is a provocative question for most vertebrate zoology
majors.[1]
```

Oto dziesięć par klawiszy, które pozamieniał miejscami:

```
t - u
h - k
o - s
c - m
z - l
d - e
v - i
r - n
a - f
g - p
```

Jest to kolejny problem z zastępowaniem liter, który można rozwiązać metodą analizy względnych częstości opisaną w rozwiązaniu problemu z punktu 1.1.3. Jednak metoda ta nie najlepiej nadaje się do krótkich próbek tekstu, takich jak zaprezentowane zdanie. Lepiej spróbować odgadnąć znaczenie krótkich słów i na podstawie liter wchodzących w ich skład odgadywać dłuższe słowa. Na przykład istnieje duże prawdopodobieństwo, że jednym z trzyliterowych słów w zdaniu będzie *the*, *for*, *you* itp. Litery z tych słów można wykorzystać do odgadnięcia liter w słowach dłuższych i kontynuować proces aż do odszyfrowania całego zdania.

2.1.6. Ataki siłowe i lamerzy

Problem, który był przedmiotem kłótni lamerów, należy do zagadnień kombinatoryki. Maksymalny czas potrzebny do wypróbowania wszystkich możliwych kombinacji można wyliczyć z następującego wzoru:

$$t = \frac{1}{S}\sum_{i=1}^{L} N^i$$

W zaprezentowanym wzorze S oznacza liczbę prób wykonywanych w ciągu sekundy, L to maksymalna długość hasła, a N — liczba znaków w zbiorze.

[1] Życie seksualne świszczy to prowokacyjne pytanie dla najważniejszych zoologów zajmujących się kręgowcami.

Ponieważ w angielskim alfabecie jest 26 liter, maksymalny czas potrzebny do sprawdzenia wszystkich możliwych kombinacji hasła nie dłuższego niż pięć wielkich liter zgodnie ze wzorem można wyliczyć w następujący sposób:

$t = (26^1 + 26^2 + 26^3 + 26^4 + 26^5)/50\,000 = 247{,}1326$ sekundy, czyli około 4,12 minuty.

Z kolei jeśli w haśle nie dłuższym niż cztery znaki mogą występować zarówno wielkie, jak i małe litery, a także cyfry i symbole umieszczone na klawiszach z cyframi, liczność zbioru dozwolonych znaków wynosi 72. Maksymalny czas potrzebny do sprawdzenia wszystkich możliwych kombinacji można w tym przypadku obliczyć z następującego wzoru:

$t = (72^1 + 72^2 + 72^3 + 72^4)/50\,000 = 545{,}0472$ sekundy, czyli około 9,08 minuty.

Tak więc pierwszy lamer miał rację.

Gdy znana jest dokładna liczba znaków w haśle, na przykład pięć w pierwszym przypadku i cztery w drugim, procedura obliczenia maksymalnego czasu potrzebnego do sprawdzenia wszystkich możliwych kombinacji haseł jest jeszcze prostsza:

$t = 26^5/50\,000 = 237{,}62752$ sekundy, czyli w przybliżeniu 3,96 minuty dla pierwszego przypadku.

$t = 72^4/50\,000 = 537{,}47712$ sekundy, czyli w przybliżeniu 8,96 minuty dla drugiego przypadku.

Jak widać, i tym razem pierwszy lamer miał rację.

2.1.7. Admin Monkey

W wyniku analizy znaków hasła w postaci kodów ASCII można zauważyć, że wartości kodów na pozycjach parzystych są parzyste, a na nieparzystych — nieparzyste. Oznacza to, że znalezienie wszystkich możliwych kombinacji haseł zajmie tylko połowę czasu w porównaniu z sytuacją, w której nie byłoby takiej prawidłowości.

2.1.8. Dwa kolejne generatory haseł

Jeśli ekspert w dziedzinie zabezpieczeń zasługuje na swój tytuł, powinien zauważyć, że drugi program generuje hasła ze złym rozkładem znaków. Przeanalizujmy na przykład następujące hasło:

```
fL2ffh*fL
```

Czterokrotnie wystąpił znak *f* i dwukrotnie znak *L*. Na tej podstawie można wyciągnąć wniosek, że w programie zastosowano nie najlepszy algorytm generowania liczb losowych. W związku z tym ekspert powinien odradzić administratorowi korzystanie z drugiego generatora haseł.

2.1.9. Słynne zdanie

Po odszyfrowaniu zdanie brzmi następująco:

```
software is like sex: it's better when it's free
```[2]

Jest to sentencja wypowiedziana przez Linusa Torvaldsa.

Do zakodowania frazy wykorzystano prostą metodę: każdą parę sąsiadujących ze sobą liter zamieniono miejscami. Wskazówką może być na przykład dwukropek występujący za spacją zamiast przed nią (patrz punkt 1.1.9), co nie powinno mieć miejsca w poprawnie skonstruowanym zdaniu. Na listingu 2.1.9 zamieszczono kod źródłowy programu w języku C służący do odszyfrowania komunikatów zakodowanych tą metodą. Program może także odszyfrowywać zdania przekazane jako wartość zmiennej `str`.

Listing 2.1.9. *Kod źródłowy do odszyfrowania słynnego zdania wypowiedzianego przez Linusa Torvaldsa*

```
#include <stdio.h>
int main()
{
    char str[]="ostfaweri sileks xe :tis'b teet rhwnei 't srfee";
    int i;
    for (i=0; str[i] !='\0'; i++) {
        printf("%c", str[++i]);
        printf("%c", str[i-1]);
    }
    printf("\n");
    return 0;
}
```

2.1.10. Tajna wiadomość

W zaszyfrowanym ciągu znaków można zauważyć równomiernie rozmieszczoną grupę *1*. W wyniku podzielenia ciągu na części, począwszy od tej grupy znaków, otrzymamy trzy ciągi równej długości. Co więcej, każdy z podciągów przypomina hasło zaszyfrowane za pomocą algorytmu MD5, podobne do haseł zapisanych w pliku */etc/shadow* w wielu systemach uniksowych. Prefiks *1* jest charakterystyczny dla

[2] Oprogramowanie jest jak seks: lepiej jeśli jest darmowy — *przyp. tłum.*

haseł zaszyfrowanych tą metodą. Sprawdźmy tę myśl. Wpiszemy trzy podciągi do pliku tekstowego (np. *cipher.txt*), poprzedzając każdy z nich dwukropkiem:

```
:$1$rXzFJ1wx$kigQ3k69K8V5QvGUoupCu0
:$1$xzr83KXR$nOGL2E5/iWSNIKBidzRPI1
:$1$QeBLTtWa$5KrmfyoV5h3rB3j6RBpod0
```

Następnie plik tekstowy przetworzymy za pomocą programu *John the Ripper*. W systemach uniksowych można to zrobić za pomocą następującego polecenia:

```
#./john cipher.txt
```

Oto uzyskane wyniki:

```
Secret
fiction
password
```

Wszystkie powyższe słowa należą do standardowego słownika programu John the Ripper (*password.lst*).

Należy wziąć pod uwagę fakt, że program John the Ripper nie odszyfrowuje haseł w takiej kolejności, w jakiej występują w pliku tekstowym. Kolejność odszyfrowania zależy od tego, które ze słów zostanie odnalezione w słowniku jako pierwsze.



John the Ripper to program do łamania haseł firmy Solar Designer. Za pomocą programu można łamać hasła standardowe i zaszyfrowane podwójnie za pomocą algorytmów DES, MD5 i Blowfish. Możliwości programu zostały opisane w jego pliku pomocy. Informacje na jego temat można również znaleźć pod adresem *http://www.openwall.com/john*. W witrynie można także pobrać program John the Ripper dla systemów Unix, Win32 i DOS.

2.1.11. Kandydat do pracy

Dyrektor oczywiście nie zatrudni kandydata, ponieważ raczej nie będzie skłonny czekać, aż ten „haker" napisze swój program, a następnie aż program zwróci wyniki. Liczbę siedmioznakowych haseł, które można utworzyć z liter alfabetu angielskiego, takich, które zawierają co najmniej jedną wielką literę *X*, można określić bez korzystania z programów stosujących metodę siłową. Wystarczy wykonać następujące proste obliczenie arytmetyczne:

$$26^7 - 25^7 = 8\ 031\ 810\ 176 - 6\ 103\ 515\ 625 = 1\ 928\ 294\ 551$$

2.1.12. Inny kandydat do pracy

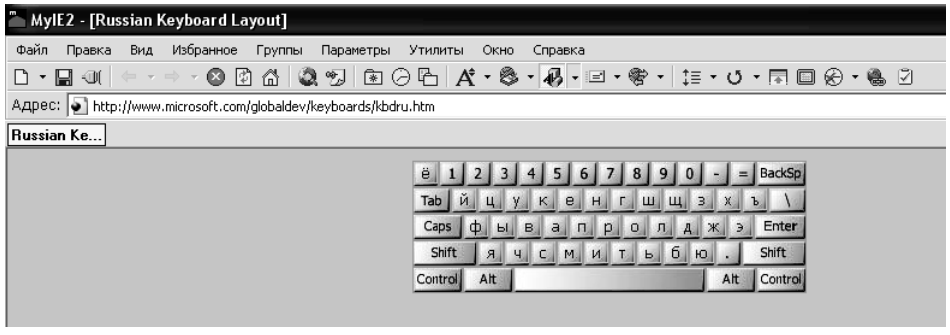
Dyrektor oczywiście nie zatrudni tego kandydata z tych samych powodów, dla których nie zatrudnił pierwszego. Liczbę możliwych haseł można określić za pomocą prostego obliczenia arytmetycznego. Gdyby wszystkie znaki w haśle miały być różne, problem

można by rozwiązać, obliczając liczbę permutacji ze wzoru *n!*, czyli w przypadku haseł składających się z dziesięciu znaków 10! = 3 628 800. W haśle podanym przez dyrektora trzykrotnie występuje litera *A* i dwukrotnie litera *h*. Wykorzystanie litery *A* w miejscu innej litery *A* nie zmieni hasła. To samo dotyczy litery *h*. W związku z tym liczbę haseł, które można utworzyć, wykorzystując znaki z hasła podanego przez dyrektora, można obliczyć z następującego wzoru:

$$\frac{10!}{3!\cdot 2!} = 302\ 400$$

2.1.13. Gil pisze do Minusa

Zdanie jest nic nieznaczącym zbiorem liter rosyjskiego alfabetu. Najprawdopodobniej Gil tak zdenerwował się na myśl o Minusie Thornvildzie i jego działalności na szkodę programowego imperium Gila, że nie zauważył, iż korzysta z rosyjskiej klawiatury. W odróżnieniu od Gila, myślenie o swoich ideologicznych (jeśli chodzi o oprogramowanie) oponentach nie pozbawiło Minusa zdolności myślenia, a zatem natychmiast spostrzegł, w jaki sposób można odszyfrować wiadomość od Gila: po prostu powiązał rosyjskie litery wiadomości z literami angielskiego alfabetu przyporządkowanymi do tych samych klawiszy na klawiaturze. Był to zatem kolejny problem z zastępowaniem liter. Tym razem nie trzeba jednak wykonywać żadnych analiz względnej częstości liter. Wystarczy jedynie znać układ rosyjskiej klawiatury (rysunek 2.1.13). Informacje na ten temat można pobrać pod adresem *http://www.microsoft.com/globaldev/reference.keyboards.aspx*. Oprócz rosyjskiego układu klawiatury w tej witrynie firmy Microsoft można znaleźć układy klawiatur praktycznie dla wszystkich języków.



Rysunek 2.1.13. *Układ rosyjskiej klawiatury pobrany z witryny Microsoft*

Wystarczy zastąpić rosyjskie litery z ich angielskimi odpowiednikami, aby uzyskać następujący komunikat:

```
We reject the claim as we consider it to be groundless.
```
[3]

Jakie żądanie Gil miał na myśli, to już nie nasza sprawa.

[3] Żądanie odrzucamy jako bezpodstawne. — *przyp. tłum.*

2.1.14. Minus odpowiada Gilowi

Oto odpowiedź Minusa na list Gila:

```
Pack my box with five dozen liquor jugs[4].
```

Pomimo tego, co mówią o umiejętnościach komputerowych Gila, nie jest on zupełnie zielony w tej materii i coś niecoś wie. Zna się też trochę na kryptologii. Odkrył, że w liście Minusa zamieniono miejscami cyfry w kodach szesnastkowych liczb (w notacji dwójkowej dokonano cyklicznego przesunięcia o cztery pozycje). Na przykład kod szesnastkowy pierwszego znaku w liście Minusa to *05h*. Przestawienie cyfr miejscami tworzy kod szesnastkowy *50h*, co odpowiada kodowi litery *P*. Kod następnego symbolu to *16h*, co po odwróceniu daje *61h* i odpowiada literze *a*, itd. W rezultacie, by odczytać wiadomość od Minusa, Gil musiał jedynie poprzestawiać miejscami cyfry w szesnastkowych kodach znaków wiadomości, a następnie zajrzeć do tablicy ASCII dla strony kodowej ANSI 1251. Na listingu 2.1.14 zamieszczono kod źródłowy programu w asemblerze, który wykonuje taką konwersję. Kod źródłowy oraz skompilowane pliki można także znaleźć pod adresem: *ftp://ftp.helion.pl/przyklady/haklam.zip*, w katalogu *\Czesc2\Rozdzial1\1.14*.

Listing 2.1.14. *Cykliczne przesunięcie kodu dwójkowego o cztery pozycje*

```
CSEG segment
assume CS:CSEG, DS:CSEG, ES:CSEG, SS:CSEG
org 100h
Start:
    jmp Go
    Fraza db 'Pack my box with five dozen liguor jugs.','$'
Go:
    lea bx, Fraza
    mov cx,40
Hi:
    mov ah,[bx]
    push ex
    mov cl,04
    ror ah,cl
    mov [bx],ah
    pop cx
    inc bx

    loop Hi
    mov ah,09
    lea dx,Fraza
    int 21h
    int 20h
CSEG ends
end Start
```

[4] Zapakuj mi pięć tuzinów flaszek alkoholu — *przyp. tłum.*

2.1.15. Sejf na łapę królika

Liczba kombinacji dla pierwszego sejfu wynosi 10^5, czyli 100 000. Dla drugiego sejfu liczba ta wynosi 5^{10}, czyli 9 765 625. Liczba siedmioznakowych haseł dla trzeciego sejfu wynosi 7^7, czyli 823 543. W związku z tym drugi sejf jest najbezpieczniejszy z trzech i to w nim Gil powinien przechowywać łapę królika, w czasie gdy on sam pływa w basenie.

2.1.16. Hey Hacker!

Zapiszmy kody szesnastkowe niezaszyfrowanej i zaszyfrowanej wersji frazy „Hey Hacker!":

```
 H   e   y       H   a   c   k   e   r   !
48h 65h 79h 20h 48h 61h 63h 6Bh 65h 72h 21h

 z   g   H       K   @   Q   k   @   @   #
7Ah 67h 48h 12h 4Bh 40h 51h 6Bh 40h 40h 23h
```

Spróbujmy teraz uzyskać maskę XOR, tzn. wykonajmy operację XOR na kodach znaków fraz w postaci odszyfrowanej i zaszyfrowanej. Oto wynik tej operacji:

```
32h 02h 31h 32h 03h 21h 32h 00h 25h 32h 02h
```

Natychmiast rzuca się w oczy następujący schemat: maska XOR co trzeciego znaku jest taka sama (*32h*), natomiast maska znaku następującego za każdym z tych znaków ma wartość z zakresu 0 – 3.

W wyniku szeregu eksperymentów polegających na zastosowaniu uzyskanej maski dla zaszyfrowanej frazy (zalecam, aby Czytelnicy wykonali je samodzielnie) uzyskamy następującą frazę:

```
This is rubric X-Puzzle!
```[5]

Logikę algorytmu szyfrowania łatwo zrozumieć na podstawie kodu źródłowego programu szyfrującego napisanego w języku C, który pokazano na listingu 2.1.16. Program sekwencyjnie wykonuje operacje `xor 50`, `or 3` oraz `and 200` dla kolejnych znaków frazy:

Listing 2.1.16. *Algorytm szyfrowania*

```
#include <stdio.h>
int main() {
    char str[]="This is rubric X-Puzzle!";
    int i=0;
    while (str[i]!='\0') {
        printf("%c", str[i++]^50);
```

[5] To jest rubryka X-Puzzle! — *przyp. tłum.*

```
        if (str[i]=='\0') break;
        printf("%c", str[i++]|3);
        if (str[i]=='\0') break;
        printf("%c", str[i++]&200) ;
        if (str[i]=='\0') break;
    }
        printf("\n") ;
    return 0;
}
```

2.1.17. Dowcipny kryptolog

Własności algorytmów użytych do szyfrowania hasła są następujące:

- *Base64*

  Opis tego algorytmu szyfrowania można znaleźć w dokumencie RFC2045. Jedną z charakterystycznych własności jest jeden lub dwa znaki wypełniające (znaki równości) umieszczone na końcu zaszyfrowanego tekstu.

- *DES*

  Długość haseł zaszyfrowanych za pomocą algorytmu DES w systemach uniksowych wynosi 13. Pierwsze 2 znaki to dane inicjalizacyjne (ang. *salt*).

- *MD5*

  Hasło zaszyfrowane algorytmem MD5 w systemach uniksowych zawsze zaczyna się od prefiksu *1*. Ogólny jego format to *1salt$hash*.

- *XOR*

  Operacja XOR jest odwracalna (zobacz rozwiązanie problemu opisanego w punkcie 1.1.1).

Spójrzmy teraz na hasło zaszyfrowane przez dowcipnego kryptologa. Na końcu ciągu znaków są znaki równości, a zatem zgodnie z przedstawionymi wcześniej właściwościami jako ostatni zastosowano algorytm Base64. Spróbujmy odszyfrować hasło. Można to zrobić za pomocą dowolnego programu odszyfrowującego Base64 dostępnego w internecie albo za pomocą prostego programu w Perlu, którego kod źródłowy zamieszczono na listingu 2.1.17.

Listing 2.1.17. *Prosty skrypt w Perlu do odszyfrowywania tekstów zaszyfrowanych za pomocą algorytmu Base64*

```
#!/usr/bin/perl
use MIME::Base64;
$code="JDEkYmxhYmxhJHFUZEhUL0h6UVBKZC9yN3ZrcOFscjE=";
print decode_base64($code);
```

Wynik operacji odszyfrowania jest następujący:

```
$1$blabla$qTdHT/HzQPJd/r7vksAlr1
```

Na podstawie prefiksu *1* można stwierdzić, że przedostatnim algorytmem użytym do zaszyfrowania hasła był MD5. Uzyskany ciąg należy przekazać do programu John the Ripper, który odszyfruje tekst za pomocą metody słownikowej. Przedtem jednak zaszyfrowane hasło należy poprzedzić dwukropkiem i zapisać w pliku tekstowym:

```
:$1$blabla$qTdHT/HzQPJd/r7vksAlr1
```

Program John the Ripper niemal natychmiast zwróci następujący ciąg znaków:

```
passwdpasswd
```

Jak pamiętamy, dowcipny kryptolog zastosował jeszcze maskę XOR oraz szyfrowanie DES. Jest mało prawdopodobne, aby powyższy ciąg znaków był wynikiem szyfrowania DES, a zatem spróbujmy zastosować do niego maskę XOR:

```
\x08\x18\x3C\x3E\x44\x32\x03\x52\x27\x47\x01\x06\x4D
```

Oto wynik tej operacji:

```
xyOM3Vs3T4vbM
```

Ten ciąg bardziej przypomina hasło zaszyfrowane za pomocą algorytmu DES (ponieważ ma rozmiar 13 znaków). Ponownie przekażemy je do programu John the Ripper w celu przetworzenia z wykorzystaniem metody słownikowej. Podobnie jak w przypadku algorytmu MD5, hasło należy zapisać w pliku tekstowym i poprzedzić dwukropkiem. Oto wynik działania programu John the Ripper:

```
Natasha
```

To jest właśnie oryginalne hasło zaszyfrowane przez dowcipnego kryptologa.