

Cryptopals - Relatório do desafio 3

Link: <https://cryptopals.com/sets/1/challenges/3>

1 - Introdução

O exercício 3 é chamado de “*Single-byte XOR cipher*”, traduzindo para o português fica “Cifra XOR de byte único”. Nele temos que construir um programa que receba uma mensagem encriptada hexadecimal do teclado, encontre uma chave e retorne a mensagem descriptada em inglês. A chave é um caractere(valor binário) da tabela ASCII e temos que fazer XOR byte a byte com a string cifrada para obter a mensagem defifrada em texto claro e em inglês.

Acredito que exista algumas formas de resolver este exercício, como análise de frequência e força bruta, entretando, escolhi o método de força bruta para resolver ele.

2 - Como funciona a Single-byte XOR cipher

É um método de criptografia com chaves simétricas(chaves iguais para encriptar e descriptar). Ele funciona através de uma chave e uma mensagem para encriptar por meio da operação XOR bit a bit. Basicamente temos que escolher algum caractere da tabela ASCII para ser nossa chave; converter essa chave e a mensagem para a base binária e fazer a operação XOR bit a bit da esquerda para a direita, a cada 8 bits da mensagem e os 8 bits da chave; juntar todos os bits em uma nova mensagem binária; converter esta mensagem para texto de acordo com a tabela ASCII e pronto. Temos assim uma mensagem encriptada.

Para descriptografar a mensagem, basta aplicar o mesmo método com a chave. De forma geral, este método de criptografia funciona assim:

$$\text{mensagem XOR chave} = \text{mensagem encriptada}$$
$$\text{mensagem encriptada XOR chave} = \text{mensagem}$$

3 - Desenvolvimento

Basicamente o programa converte a string de entrada para a base binária e testa todos os valores de 1 ... 256 (quantidade de valores possíveis com 8 bits) bit a bit com cada 8 bits da string binária, coloca esse resultado em uma segunda string, converte essa ultima string para texto de acordo com a tabela ASCII e mostra este resultado junto com a chave binária na tela. Ao fim do programa, iremos ter 256 mensagens na tela e temos que procurar a frase que faz mais sentido, ou seja, estamos em busca da frase em inglês que dê pra ler e ver seu significado com clareza, como citado anteriormente. O programa da mesma forma que os anteriores está dividido em algumas partes:

- 1ª: Criação de variáveis estáticas para serem utilizadas no programa;
- 2ª: Pedir a string em hexadecimal do teclado;
- 3ª: Verificar se a string está na base hexadecimal. Se não estiver, mostra mensagem de erro e termina o programa;
- 4ª: Remove espaços em branco da string hexadecimal, caso tenha;
- 5ª: Converte string hexadecimal para a base binária;
- 6ª: Faz a operação XOR bit a bit com todos os caracteres(chaves) possíveis da tabela ASCII, em cada, converte o resultado para texto e exibe na tela;

4 - Nomes e breve explicação das variáveis utilizadas

- **void decimalToBinary(char *string, int number)**
 - ➔ **char temp[8]:** variável para armazenar conversão de number para base binária com até 8 bits;
 - ➔ **char digit:** a função usa essa variável pra capturar o caractere de cada resto e adicionar em *temp*;
 - ➔ **int index:** utilizada para acessar índices de *string* e *temp*;
- **int main()**
 - ➔ **int index:** utilizada para acessar índice de *strings* e na hora de percorrer cada uma delas com o *for*;
 - ➔ **int value:** auxilia na remoção de espaços em branco de *stringInput* e é uma das entre 0 e 256, em cada, no momento de *XOR bit a bit* por força bruta;
 - ➔ **int var:** mesma função de *index*, porém é utilizada quando esta última está ocupada;
 - ➔ **int valueIntDigitResult:** Serve para armazenar o valor inteiro do dígito resultado da operação XOR bit a bit;
 - ➔ **char character:** serve para armazenar o valor em *char* do valor decimal de cada byte de *stringInputInBinary*;
 - ➔ **char stringInput[1024]:** armazena a *string* inicial de entrada, fornecida através do teclado;
 - ➔ **char stringInputInBinary[1024]:** armazena conversão de *stringInput* para a base binária;

- ➔ **const char valuesInHexadecimalToBinary[16][5]:** É a mesma variável utilizada nos exercícios 1 e 2, possui a mesma função que podemos perceber pelo nome dela – valores em hexadecimal para binário. Para mais detalhes veja o relatório do desafio 1;
- ➔ **char stringResultOfXor[1024]:** armazena o texto da *string* resultado da operação XOR bit a bit entre *stringInputInBinary* e cada chave. Esse processo acontece em cada *loop*;
- ➔ **char stringValueToBinary[9]:** armazena o valor binário de *value*;
- ➔ **int byteStringResultDecimal:** armazena resultado decimal do resultado da operação XOR bit a bit de *byteString* com *stringValueToBinary*;
- ➔ **char byteString[9]:** armazena cada byte(8 bits) de *stringInputBinary*, da esquerda para a direita;

5 - Funções utilizadas no programa

As funções utilizadas foram as mesmas do desafio 1, para mais detalhes veja o relatório do 1º desafio.

6 – Funções construídas no programa

- ➔ **void decimalToBinary(char *string, int number):**

Serve de forma geral para converter o parâmetro *number* para a base binária. Ela armazena a conversão na variável endereçada por *string*. Nas linhas 11 e 12 criamos as variáveis *temp* e *digit*, do tipo *char* e o inteiro *index*.

Para converter um número na base decimal para a base binária, basta dividir o número sucessivamente por 2 até que esta divisão seja igual a zero, em seguida juntar todos os restos e inverter a ordem deles e pronto. Na função *decimalToBinary*, este processo é feito dentro do *while* (linhas 16-23), na linha 17 atribui-se o resto de *number* por 2 à variável *digit* e na linha seguinte utilizamos a função *strncat()* para concatenar este digit com o vetor *temp*.

Em seguida usamos o *if* para verificar se a *number/2 == 0*, caso verdadeiro paramos o *while* com o comando *break* e temos o *number* convertido para a base binária na *string temp*. Entretanto, caso falso dividimos *number* por 2 e o ciclo *while* continua a conversão até que a condição inicial seja satisfeita.

Na linha 27 atribui-se '0' à *digit*, em seguida com *while* o programa verifica se a quantidade de caracteres de *temp* é menor que oito, caso positivo adicionamos zeros ao final de *temp* com o comando *strncat()*, até que *temp* tenha 8 caracteres totalmente preenchidos.

Por fim, utilizamos o *for* para atribuir valores de *temp*, de traz para a frente, na variável endereçada pelo parâmetro *string*. Para isso, no *for* temos *index* = 7 com a condição *index* >= 0 e o programa diminui 1 unidade de *index* a cada *loop* com *--index*. Por fim na linha 34 atribui-se a última posição de *temp* à primeira de *string*, a penúltima de *temp* à segunda de *string* e assim sucessivamente. Dessa forma, concluímos o processo de conversão de *number* para a base binária em na variável *string*.

```
7  /* Função para converter number para base binária de 8 bits
8  * Armazena a conversão em string
9  */
10 void decimalToBinary(char *string, int number) {
11     char temp[8] = "", digit;
12     int index;
13
14     //Converte número para base binária
15     while(1) {
16
17         digit = number%2 + '0';
18         strncat(temp, &digit, 1);
19
20         if(number/2 == 0)
21             break;
22
23         number /= 2;
24     }
25
26     //Adiciona 0's ao final do número, caso o tamanho seja menor que 8
27     digit = '0';
28     while(strlen(temp) < 8) {
29         strncat(temp, &digit, 1);
30     }
31
32     //invertendo numberInBinary com ajuda de outra variavel
33     for(index = 7; index >= 0; --index) {
34         string[7 - ind] = temp[ind];
35     }
36 }
```

Imagem 1 – Função *decimalToBinary(char *string, int number)*

- **void askForStringHexadecimal(char *string, int size):** Esta função foi utilizada nos desafios anteriores e possui o mesmo objetivo: pedir *string hexadecimal* para usuário do teclado. Para mais detalhes veja o relatório do desafio 1.

7 - Explicação do código completo do programa

Nas primeiras cinco linhas temos a definição das estruturas básicas para o funcionamento do programa, como importação de bibliotecas e definição de macros com a diretiva *#define*. Essa parte pode ser vista com mais detalhes no relatório do desafio 1.

```
1  #include <stdio.h>
2  #include <string.h>
3  #include <math.h>
4
5  #define BASE16TABLE "0123456789abcdefABCDEF"
```

Imagem 2 – Importando bibliotecas e definindo macros

Na imagem 3 o programa cria variáveis para utilizar na função *main()*. Elas seguem a mesma estrutura dos exercícios anteriores, temos as variáveis inteiras: *index*, *value*, *var* e *valueIntDigitResult*; as variáveis do tipo *char*: *character*, *stringInput*, *stringInputBinary* e *valuesInHexadecimalToBinary*.

```
50      /** Declaração de variáveis para serem utilizadas no programa **/  
51      int index, value, var, valueIntDigitResult;  
52      char character, stringInput[1024], stringInputInBinary[1024] = "";  
53  
54      const char valuesInHexadecimalToBinary[16][5] = {"0000", "0001", "0010", "0011",  
55                                                         "0100", "0101", "0110", "0111",  
56                                                         "1000", "1001", "1010", "1011",  
57                                                         "1100", "1101", "1110", "1111"};
```

Imagem 3 – Declaração de variáveis

Na figura seguinte temos uma estrutura que pede a *string hexadecimal* como entrada pelo teclado, como descrito nos relatórios 1 e 2. Primeiro mostra uma mensagem pedindo para informar uma *string hexadecimal* e depois, na linha seguinte, o programa executa a função *askForStringHexadecimal(stringInput, 1024)* para capturar uma *string* da entrada padrão e armazenar na variável *stringInput*.

```
59      /** Pedir string hexadecimal do teclado e armazenar em stringInput **/  
60      printf("Informe uma string em hexadecimal abaixo: \n\n");  
61      askForStringHexadecimal(stringInput, 1024);
```

Imagem 5 – Entrada de dados

Na próxima imagem temos uma estrutura para verificar se a *string* digitada esta na base *hexadecimal* corretamente. Funciona da mesma forma como descrito no relatório do desafio 1, porém, os nomes das variáveis mudam e caso a *string* esteja na base incorreta, o programa mostra uma mensagem de erro e termina com *return 1*.

```
63      /** verificar se stringInput esta na base hexadecimal **/  
64      for(index = 0; stringInput[index] != '\0'; ++index) {  
65          if((strchr(BASE16TABLE, stringInput[index]) == NULL) && (stringInput[index] != ' ')) {  
66              printf("\n[ERRO --> BASE_INCORRETA!] Informe a string corretamente!\n");  
67              return 1;  
68          }  
69      }
```

Imagem 6 - Validação da string de entrada

O próximo passo é retirar os espaços em branco da *string*, caso possua. Este passo esta descrito na imagem abaixo e funciona de forma semelhante ao que foi descrito no relatório 1.

```

71      /** Removendo espaços em branco de stringInput, caso tenha **/
72      for(value = 0, index = 0; stringInput[index] != '\0'; ++index) {
73          if(stringInput[index] != ' ') {
74              stringInput[value] = stringInput[index];
75              ++value;
76          }
77      }
78      stringInput[value] = '\0';

```

Imagem 7 – Remoção de espaços em branco na string de entrada

Prosseguindo, o programa converte a *string* de entrada para a base binária. O processo é bem semelhante ao dos relatórios anteriores. De forma ele percorre todos os caracteres da *string hexadecimal* por meio de *estruturas de repetição*, verifica através de *estruturas condicionais* qual o valor binário de cada caractere em *valuesInHexadecimalToBinary* e concatena com *stringInputBinary*. Ao final do processo tem-se a *string* em *hexadecimal* convertida para binária em *stringInputBinary*. Tal processo pode ser observado na imagem abaixo:

```

80      /** Converter stringInput para a base binária e armazenar em stringInputInBinary **/
81      for(index = 0; stringInput[index] != '\0'; ++index){
82          if((stringInput[index] - '0') >= 0 && (stringInput[index] - '0') <= 9)
83              strcat(stringInputInBinary, valuesInHexadecimalToBinary[(stringInput[index] - '0')]);
84          else
85              if(stringInput[index] == 'a' || stringInput[index] == 'A')
86                  strcat(stringInputInBinary, valuesInHexadecimalToBinary[10]);
87              else if(stringInput[index] == 'b' || stringInput[index] == 'B')
88                  strcat(stringInputInBinary, valuesInHexadecimalToBinary[11]);
89              else if(stringInput[index] == 'c' || stringInput[index] == 'C')
90                  strcat(stringInputInBinary, valuesInHexadecimalToBinary[12]);
91              else if(stringInput[index] == 'd' || stringInput[index] == 'D')
92                  strcat(stringInputInBinary, valuesInHexadecimalToBinary[13]);
93              else if(stringInput[index] == 'e' || stringInput[index] == 'E')
94                  strcat(stringInputInBinary, valuesInHexadecimalToBinary[14]);
95              else if(stringInput[index] == 'f' || stringInput[index] == 'F')
96                  strcat(stringInputInBinary, valuesInHexadecimalToBinary[15]);
97      }

```

Imagem 8 – Conversão da string de entrada para base binária

A próxima imagem imprime a mensagem “Resultados: ” na tela. Isso indica que abaixo dessa *string* irão vir outras *strings* indicando o resultado da *decriptação XOR bit a bit* com todas as 256 possibilidades de chaves e a mensagem cifrada inicialmente.

```

98
99      printf("\n\nResultados: \n\n");
100

```

Imagem 9 – Exibir resultados

Enfim, chegamos na parte mais importante de todo o programa. Agora o programa vai fazer o *XOR bit a bit* com todas os caracteres binários da tabela ASCII como citado anteriormente.

Inicialmente, na linha 104, o programa utiliza um *for* com *value = 0* até 255 e a cada *loop* *value* aumenta em uma unidade com *++value*. Dentro deste *for*, na linha 107 declara-se os vetores *stringResultOfXor*, que irá armazenar o resultado em texto ASCII da operação *XOR bit a bit* entre a possível chave e a mensagem cifrada na base binária, tem-se também a *stringValueToBinary* para armazenar o valor binário de *value*. Na linha 108, a função *decimalToBinary(stringValueToBinary, value)* é chamada, converte *value* para a base2 e armazena o resultado em *stringValueToBinary*. Na linha 110, o programa mostra o valor binário de *value*.

Na linha 113 a aplicação usa outro *for* para fazer a operação *XOR bit a bit* entre cada caractere da *string hexadecimal* em binário e a possível chave binária. Para isso iniciamos esse *for* com *index = 0* indo até o final da *string* binária com a condição *stringInputInBinary[index] != '\0'* e a cada *loop* o valor de *index* aumenta em 8 unidades com *index += 8* (para operação *XOR* entre cada 8 *bits* de *stringInputBinary* com os 8 *bits* de *stringValueToBinary*). Dentro desse *for* criamos a variável *byteStringResultDecimal* do tipo inteiro com valor 0 e o vetor *byteString* com 8 posições iniciando com "". Logo em seguida nas linhas 120-122 tem-se outro *for*, para capturar 8 posições de *stringInputBinary* a partir do valor de *index* para frente e armazenar na variável *byteString*, assim temos o valor binário de cada caractere à cada *loop* em *byteString*. Em seguida o programa por meio de outro *for* faz a operação *XOR bit a bit* entre *byteString* e *stringValueToBinary* e nessa mesma estrutura vai convertendo esse resultado para a base decimal na variável *byteStringResultDecimal*. Após este último *for*, o programa na linha 133 converte *byteStringResultDecimal* para o seu valor correspondente na tabela ASCII através da conversão para *char* com *(char)byteStringResultDecimal* e guarda em *character*, na próxima linha utiliza *strncat(stringResultOfXor, &character, 1)* para concatenar *character* com *stringResultOfXor*.

Desse modo citado no último parágrafo, o programa faz operação *XOR byte a byte* entre os valores binários da possível chave (*value*) e a *string hexadecimal* de entrada, converte cada *byte* resultado para caractere em ASCII e armazena em uma *string* resultado. Com isso, ao final do processo, temos um texto que pode ser o possível resultado da *decriptação* de *stringInput*.

Na linha 138, exibimos na tela esse possível resultado e o loop principal continua. Agora ele vai realizar o mesmo processo com a próxima possível *chave*. Ao final de todo o processo de teste por *força-bruta* teremos 256 resultados na tela no formato "*chave – string decriptada*". Por fim, para encontrar a chave de *decriptação* e o resultado temos que procurar a *string* que esteja totalmente em inglês e que faça mais sentido.


```

101  ▼ /* Fazer combinação XOR com stringInputInBinary com todas as 256 possibilidades de chaves
102      * Mostrar cada chave seguida da mensagem descryptada gerada
103      */
104  ▼ for(value = 0; value <= 255; ++value) {
105
106      //Criando variáveis para serem utilizadas nesse loop
107      char stringResultOfXor[1024] = "", stringValueToBinary[9] = "";
108      decimalToBinary(stringValueToBinary, value);
109
110      printf("%s - ", stringValueToBinary); //Mostrando chave em valores binários
111
112      //For para fazer combinação Xor entre stringValueToBinary com stringInputInBinary
113  ▼ for(index = 0; stringInputInBinary[index] != '\0'; index += 8){
114
115      //Variáveis para serem utilizadas nesse segundo loop
116      int byteStringResultDecimal = 0;
117      char byteString[9] = "";
118
119      //Pegando 8 bits da esquerda para a direita e armazenando em byteString
120  ▼ for(var = index; var < (index + 8); ++var) {
121          strncat(byteString, &stringInputInBinary[var], 1);
122      }
123
124      //Fazer operação XOR bit a bit entre stringValueToBinary e byteString
125      //e converter resultado para decimal na variavel byteStringResultDecimal
126  ▼ for(var = 0; var < 8; ++var) {
127          valueIntDigitResult = (byteString[var] - '0') ^ (stringValueToBinary[var] - '0');
128          byteStringResultDecimal += pow(2, 7 - var) * valueIntDigitResult;
129      }
130
131      //Converter byteStringbyteStringResultDecimal para decimal
132      //e concatenar com stringResultOfXor
133      character = (char)byteStringResultDecimal;
134      strncat(stringResultOfXor, &character, 1);
135  }
136
137      //Mostrar resultado da string descryptada na tela
138      printf("%s\n\n", stringResultOfXor);
139  }

```

Imagem 10 – XOR bit a bit com todas as chaves e exibição de resultados

Por fim, através de *força-bruta* o programa consegue encontrar a *chave* binária de *decriptação* da *string* de entrada inicial.

8 - Testando o programa

O processo de testes é bem semelhante ao descrito nos relatórios dos desafios 1 e 2. Dessa forma, basta compilar com o comando:

```
gcc nome-do-programa.c -o nome-arquivo-de-saida -lm
```



```
Projects : bash — Konsole
File Edit View Bookmarks Settings Help
felipe@kubuntu:~/Projects$ gcc ch3-singleByteXorCipher-BruteForce.c -o ch3-singleByteXorCipher-BruteForce -lm
felipe@kubuntu:~/Projects$
felipe@kubuntu:~/Projects$ ls
ch3-singleByteXorCipher-BruteForce  ch3-singleByteXorCipher-BruteForce.c
felipe@kubuntu:~/Projects$
```

Executar *ls* para garantir que o arquivo *nome-arquivo-de-saida* foi gerado e rodar o programa com o comando:

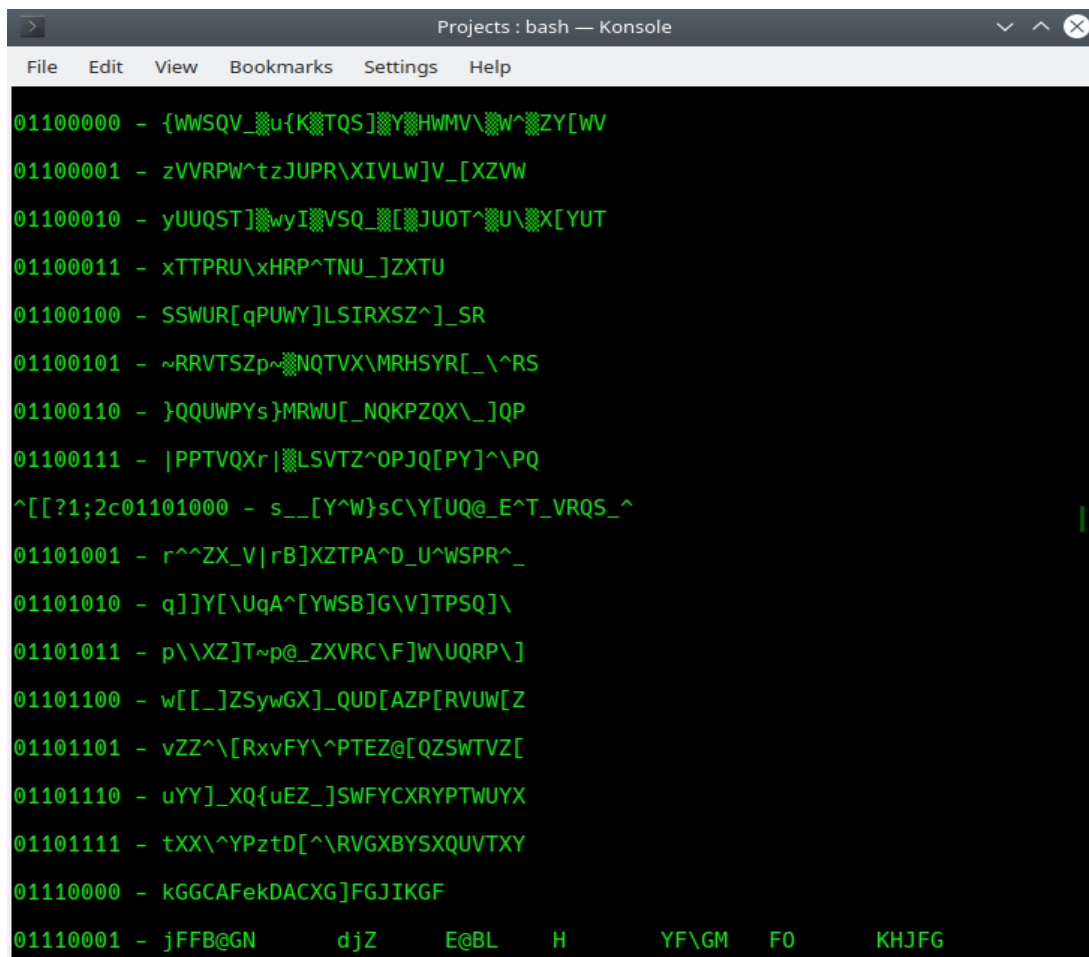
./nome-arquivo-de-saida

Neste teste vamos utilizar a *string hexadecimal* fornecida no site do exercício:

1b37373331363f78151b7f2b783431333d78397828372d363c78373e783a393b3736

```
Projects : ch3-singleByteX — Konsole
File Edit View Bookmarks Settings Help
felipe@kubuntu:~/Projects$ ./ch3-singleByteXorCipher-BruteForce
Informe uma string em hexadecimal abaixo:
1b37373331363f78151b7f2b783431333d78397828372d363c78373e783a393b3736
```

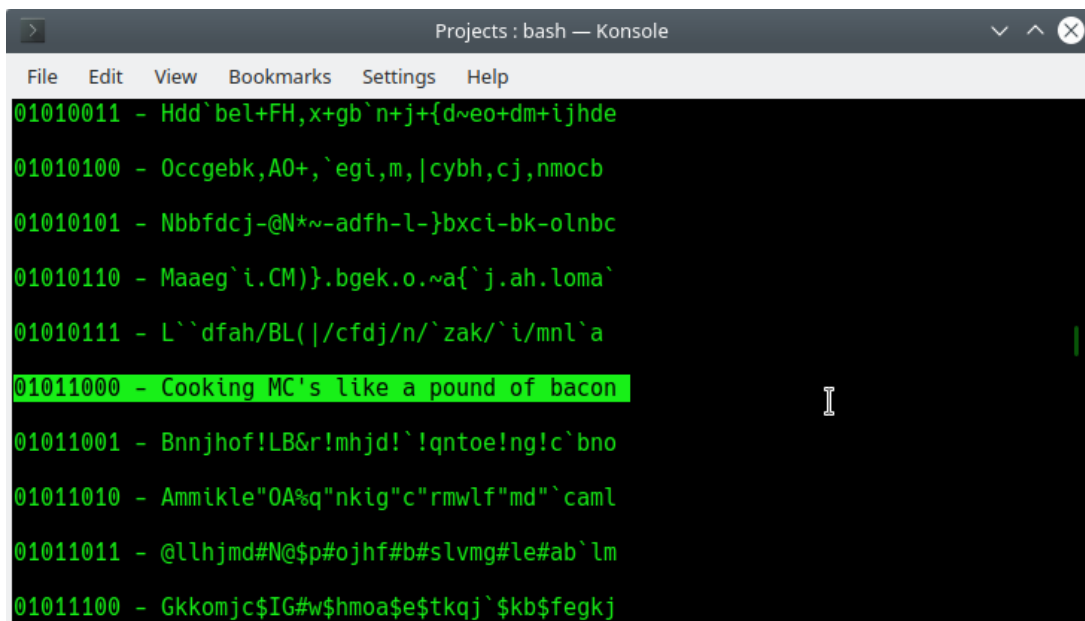
Ao clicar no <enter> vamos obter uma série de resultados como descrito na imagem abaixo.



```
Projects : bash — Konsole
File Edit View Bookmarks Settings Help

01100000 - {WWSQV_u{KTS]YHWMV\W^ZY[WV
01100001 - zVVRPW^tzJUPR\XIVLW]V_[XZVW
01100010 - yUUQST]wyI\VSQ_[J]UOT^U\X[YUT
01100011 - xTTPRU\xHRP^TNU_]ZXTU
01100100 - SSWUR[qPUWY]LSIRXSZ^_]SR
01100101 - ~RRVTSZp~NQTVX\MRHSYR[_^RS
01100110 - }QQUWPYs}MRWU[_NQKPZQX_]QP
01100111 - |PPTVQXr|LSVTZ^OPJQ[PY]^PQ
^[[?1;2c01101000 - s__[Y^W}sC\Y[UQ@_E^T_VRQS_^
01101001 - r^^ZX_V|rB]XZTPA^D_U^WSPR^_
01101010 - q]]Y[\UqA^[YWSB]G\V]TPSQ\
01101011 - p\\XZ]T~p@_ZXVRC\F]W\UQRP\
01101100 - w[_]ZSywGX_]QUD[AZP[RVUW[Z
01101101 - vZZ^\[RxvFY\^PTEZ@[QZSWTVZ[
01101110 - uYY_]XQ{uEZ_]SWFYCRYPTWUYX
01101111 - tXX\^YPztD[^RVGXBYSXQUVTXY
01110000 - kGGCAFekDACXG]FGJIKGF
01110001 - jFFB@GN      djZ      E@BL      H      YF\GM      F0      KHJFG
```

Agora basta procurar o resultado como descrito anteriormente e teremos, finalmente, a chave binária e a mensagem em texto normal em inglês decriptada. Elas podem ser observadas na imagem abaixo (*estão destacadas*):



```
Projects : bash — Konsole
File Edit View Bookmarks Settings Help

01010011 - Hdd`bel+FH,x+gb`n+j+{dneo+dm+ijhde
01010100 - Occgebk,A0+`,`egi,m,|cybh,cj,nmocb
01010101 - Nbbfdcj-@N*~-adfh-l-}bxci-bk-olnbc
01010110 - Maaeg`i.CM)).bgek.o.~a{`j.ah.loma`
01010111 - L``dfah/BL(|/cfdj/n/`zak/`i/mnl`a
01011000 - Cooking MC's like a pound of bacon
01011001 - Bnnjhof!LB&r!mhjd!`!qntoe!ng!c`bno
01011010 - Ammikle"OA%q"nkig"c"rmwlf"md"`caml
01011011 - @llhjmd#N@$p#ojhf#b#slvmg#le#ab`lm
01011100 - Gkkomjc$IG#w$hmoa$e$tkqj`$kb$fegkj
```

Portanto, concluímos pelo método de *força-bruta* que a chave de deciptação para

“1b37373331363f78151b7f2b783431333d78397828372d363c78373e783a393b3736”

é “**01011000**”

e a mensagem decifrada é

“***Cooking MC`s like a pound of bacon***”.