

Cryptopals - Relatório do desafio 2

Link: <https://cryptopals.com/sets/1/challenges/2>

1 - Introdução

O desafio 2 é chamado de “Fixed XOR” e pede para escrever uma função que receba duas strings na base hexadecimal, ambas com a mesma quantidade de caracteres. A função deve imprimir a combinação XOR dessas duas strings na tela.

No relatório do exercício 1, vimos o que é e como funcionam as strings na base hexadecimal e na base binária. Neste exercício, precisamos converter as duas strings para a base binária e fazer a combinação XOR bit a bit entre essas duas strings. Na computação existe uma área chamada Álgebra Booleana, desta área sabemos hoje em dia que XOR significa Ou-Exclusivo. É uma operação lógica entre dois valores que resulta em um valor lógico verdadeiro – 1, se e somente se, os dois operandos forem diferentes e se forem iguais resulta em falso – 0. É basicamente isso que temos que fazer no neste desafio, com isso, vamos ter uma nova string binária. Por fim, temos que converter essa string para a base hexadecimal e exibir para o usuário.

2 - Como funciona o Fixed XOR

Para entender esse desafio, podemos utilizar um exemplo. Neste vamos fazer a combinação XOR entre as palavras “alegria” e “palavra” que possuem a mesma quantidade de caracteres. Em hexadecimal essas palavras são representadas por, respectivamente, “70616c61767261” e “616c6567726961”. Passando para a base binária e efetuando a combinação XOR, temos:

```
01100001 01101100 01100101 01100111 01110010 01101001 01100001 - - > “palavra”  
^ 01110000 01100001 01101100 01100001 01110110 01110010 01100001 - - > “alegria”  
-----  
00010001 00001101 00001001 00000110 00000100 00011011 00000000 - - > resultado
```

Convertendo o resultado para a base hexadecimal temos “110d0906041b00”. Logo a operação XOR entre “70616c61767261” e “616c6567726961” resulta em “110d0906041b00”.

3 - Desenvolvimento

O programa, basicamente, pede duas strings para o usuário, converte elas para a base binária e faz a operação XOR, bit a bit entre as duas strings como descrito na parte de funcionamento desse exercício. Por fim, o programa converte a string resultante binária para a base hexadecimal e exibe o resultado na tela. O programa está dividido em algumas

partes:

- 1ª: Criação de variáveis estáticas para serem utilizadas no programa;
- 2ª: Pedir as duas strings em hexadecimal e de mesmo tamanho para usuário;
- 3ª: Verificar se as strings estão em hexadecimal. Se não estiverem, mostrar mensagem de erro e termina o programa;
- 4ª: Remover espaços das strings informadas pelo usuário, caso tenha;
- 5ª: Converter strings para a base binária;
- 6ª: Faz a operação XOR bit a bit entre as duas strings;
- 7ª: Transforma o resultado da base binária para a base hexadecimal;
- 8ª: Exibe o resultado na tela.

4 - Nomes e breve explicação das variáveis utilizadas

- **int main()**
 - **int col:** utilizada para auxiliar no acesso à informações de matrizes e para iterar matrizes de strings;
 - **int lin:** mesma funcionalidade de **int col**;
 - **int binaryResultXorOperationToDecimal:** auxiliar na conversão de **binaryResultXorOperation** para decimal a cada 4 bits;
 - **char character:** Guarda resultados de operação XOR bit a bit entre as duas strings do vetor **strings**;
 - **char strings:** Vetor com duas posições, cada com capacidade máxima de 1024 caracteres. Serve para armazenar as duas strings iniciais, uma para cada posição;
 - **char stringsInBinary:** vetor com duas posições, cada com capacidade máxima de 1024 caracteres. Serve para armazenar a conversão das strings para binário;
 - **char binaryResultXorOperation:** Armazena a string resultado da operação XOR bit a bit entre as duas strings iniciais de entrada;
 - **char stringResultXor:** Armazena uma string em texto claro como resultado da operação XOR bit a bit entre as strings[0] e strings[1];
 - **const char valuesInHexadecimalToBinary:** É o mesmo vetor utilizado no

desafio 1, tem a mesma função: proporcionar valores da base hexadecimal para a base binária;

→ **int count:** auxilia no processo de remoção de espaços em branco das `strings[0]` e `strings[1]`.

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

A única função construída nesta aplicação para utilização na `main()` foi a função **void askForStringHexadecimal(char *string, int size)** que foi utilizada no exercício 1, portanto para mais informações sobre o seu funcionamento, ela pode ser vista com mais detalhes no relatório do desafio 1.

7 - Explicação do código completo da aplicação

Nas primeiras três linhas do programa estamos importando as bibliotecas `stdio.h` (padrão de entrada e saída), `string.h` (manipulação de strings) e `math.h` (funções matemáticas). E logo na linha 5 a diretiva `#define` é usada para criar a macro `BASE16TABLE`, que é uma *string* com todos os caracteres que compõem a *base16*, a princípio tem a mesma função do desafio 1.

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

Imagem 1 - Importando bibliotecas e definindo macros

Na imagem 3, podemos ver a declaração de algumas variáveis no início da função `main()`. Na linha 20 vemos a declaração das variáveis inteiras `col`, `lin` e `binaryResultXorOperationToDecimal`. As duas primeiras são para acessar elementos da matriz `strings` do tipo `char` e a última é para armazenar resultados da conversão de valores binários para a *base10*. Na linha 21, temos a variável `caracter` e os vetores: `strings` com 2 posições para armazenar as duas strings de entrada, `stringsInBinary`, `binaryResultXorOperation` e por fim o vetor `stringResultXor`. Na linha 23, temos um vetor de strings, `valuesInHexadecimalToBinary`, que armazena valores da base hexadecimal para a base binária, como descrito no relatório do desafio 1.

```

17 int main()
18 {
19     /** Criando variaveis que vão ser utilizadas ao longo do programa **/
20     int col, lin, binaryResultXorOperationToDecimal = 0;
21     char caracter, strings[2][1024], stringsInBinary[2][1024] = {}, binaryResultXorOperation[1024] = "", stringResultXor[1024] = "";
22
23     const char valuesInHexadecimalToBinary[16][5] = {"0000", "0001", "0010", "0011",
24                                                       "0100", "0101", "0110", "0111",
25                                                       "1000", "1001", "1010", "1011",
26                                                       "1100", "1101", "1110", "1111"};

```

Imagem 2 - Declaração de variáveis

Na imagem seguinte temos uma estrutura que pede as strings para o usuário. Primeiro mostra uma mensagem informando que o usuário deve entrar com duas strings de mesmo tamanho, depois mostra uma mensagem pedindo a string 1 e outra pedindo a string 2. Após cada uma destas mensagens, rodamos a função *askForStringHexadecimal* e passamos a posição 0 na linha 30 e 1 na 32 da variável *strings*, que guarda as *strings* de entrada como citadas anteriormente. Passamos também o valor máximo de cada string que é de 1024 caracteres. Dessa forma, o programa vai ler e guardar as *strings*.

```

28     /** Pedindo strings **/
29     printf("Informe valores hexadecimais de mesmo tamanho para as strings abaixo: \n\n");
30     printf("String 1 : "); askForStringHexadecimal(strings[0], 1024);
31     printf("String 2 : "); askForStringHexadecimal(strings[1], 1024);

```

Imagem 4 – Entrada de dados

Na imagem 5 o programa verifica se as strings de entrada possuem o mesmo tamanho. Para isso, ele utiliza a função *strlen(const char *str)* que retorna a quantidade de dígitos que o parâmetro *str* possui. Assim usamos o *if* para comparar os tamanhos de *strings[0]* e *strings[1]*. Se os tamanhos forem diferentes(*!=*), então o programa mostra uma mensagem de erro e termina com *return 1*.

```

33     //Verificar se as string possuem o mesmo tamanho
34     if(strlen(strings[0]) != strlen(strings[1])) {
35         printf("[ERRO --> TAMANHOS_DIFERENTES!] Informe strings de mesmo tamanho!\n");
36         return 1;
37     }

```

Imagem 5 – Verificando se strings de entrada possuem o mesmo tamanho

Na próxima imagem, o programa verifica se as strings estão na base correta. A forma de verificação é bem semelhante a verificação descrita no desafio 1, só difere pois agora rodamos um laço para verificar a *strings[0]* e outro para a *strings[1]*. Caso encontre algum valor que não pertence à base hexadecimal em uma das posições de *strings*, o programa mostra uma mensagem de erro informando que uma das bases está incorreta, pede valores na base corretamente e termina o programa com o comando *return 1*.

```

39  /* Verificando se as strings estão na base correta
40  * Se não, o programa mostra uma mensagem de erro e termina
41  */
42  for(col = 0; col <= 1; ++col) {
43      for(lin = 0; strings[col][lin] != '\0'; ++lin) {
44          if((strchr(BASE16TABLE, strings[col][lin]) == NULL) && (strings[col][lin] != ' ')) {
45              printf("[ERRO --> BASE_INCORRETA!] Informe a(s) string(s) corretamente!\n");
46              return 1;
47          }
48      }
49  }

```

Imagem 6 – Validação das strings de entrada

O próximo passo é remover os espaços em branco das strings de entrada, caso tenham. A forma de remover os espaços em branco é bem semelhante a forma descrita no exercício 1. O programa analisa cada caractere de cada *string* e procura por espaços em branco, quando encontra, substitui ele pelo próximo caractere da sequência. Quando termina a *string[0]*, o programa faz o mesmo processo para a *string[1]*, como podemos ver na imagem abaixo:

```

51  /** Removendo espaços em branco das strings, caso tenha **/
52  for(lin = 0; lin <= 1; ++lin) {
53      int count = 0;
54      for(col = 0; strings[lin][col] != '\0'; ++col) {
55          if(strings[lin][col] != ' ') {
56              strings[lin][count] = strings[lin][col];
57              ++count;
58          }
59      }
60      strings[lin][count] = '\0';
61  }

```

Imagem 7 – Remoção de espaços em branco das strings de entrada

Prosseguindo, o programa converte as strings de entrada para a base binária. Para isso ele usa um *for* que vai percorrer cada caractere de uma das strings até que chegue no caractere '\0'. Em cada laço desse, rodamos outro *for* duas vezes para converter cada caractere de cada *string* para binário, após obter o valor do caractere em binário, o programa concatena este valor com a sua posição específica no vetor *stringBinary*, por meio da função *strcat()*. O processo acontece por meio de estruturas de decisão, de forma bem semelhante ao que foi feito no exercício 1. Podemos observar todo esse processo na imagem abaixo:

```
63 //** Converter ambas strings para binário e armazenar em um array com 2 posições: 1 para cada string **/  
64 for(col = 0; strings[0][col] != '\0'; ++col) {  
65     for(lin = 0; lin <= 1; ++lin) {  
66         if((strings[lin][col] - '0') >= 0 && (strings[lin][col] - '0') <= 9) {  
67             strcat(stringsInBinary[lin], valuesInHexadecimalToBinary[(strings[lin][col] - '0')]);  
68         } else {  
69             if(strings[lin][col] == 'a' || strings[lin][col] == 'A')  
70                 strcat(stringsInBinary[lin], valuesInHexadecimalToBinary[10]);  
71             else if(strings[lin][col] == 'b' || strings[lin][col] == 'B')  
72                 strcat(stringsInBinary[lin], valuesInHexadecimalToBinary[11]);  
73             else if(strings[lin][col] == 'c' || strings[lin][col] == 'C')  
74                 strcat(stringsInBinary[lin], valuesInHexadecimalToBinary[12]);  
75             else if(strings[lin][col] == 'd' || strings[lin][col] == 'D')  
76                 strcat(stringsInBinary[lin], valuesInHexadecimalToBinary[13]);  
77             else if(strings[lin][col] == 'e' || strings[lin][col] == 'E')  
78                 strcat(stringsInBinary[lin], valuesInHexadecimalToBinary[14]);  
79             else if(strings[lin][col] == 'f' || strings[lin][col] == 'F')  
80                 strcat(stringsInBinary[lin], valuesInHexadecimalToBinary[15]);  
81         }  
82     }  
83 }
```

Imagem 8 – Conversão das strings de entrada para binário

Na imagem 9 o programa realiza a operação *XOR bit a bit* entre as duas *strings*. Para isso utilizamos a estrutura de repetição *for*, nela definimos a variavel *col* para o valor 0 e vamos percorrer cada *caracter* de uma das *strings* até chegar no caracter '0'. A string escolhida foi a *string[0]*. Dentro desse laço, a cada loop pegamos o valor inteiro de ambas as strings com *(stringInBinary[0][col] - '0')* e *(stringInBinary[1][col] - '0')*. No desenvolvimento do exercício 1, vimos que isso funciona. Dessa forma fazemos a operação *XOR* entre esses 2 bits com o operador *^*. Após a operação transformamos o novo valor para o tipo *char*, somando '0' ao resultado. Se tirando '0', transforma em inteiro, então o processo inverso tranforma em *char*. Todo este processo esta descrito nas linha 87 da imagem abaixo. Por fim, usamos a função *strncat()* para concatenar esse dígito novo com o vetor *binaryResultXorOperation*.

```
85 >> /** Fazer operação xor entre as duas strings bit a bit */
86 >> for(col = 0; stringsInBinary[0][col] != '\0'; ++col) {
87 >> >> caractere = ((stringsInBinary[0][col] - '0') ^ (stringsInBinary[1][col] - '0')) + '0';
88 >> >> strncat(binaryResultXorOperation, &caractere, 1);
89 >> }
```

Imagem 9 – Operação XOR bit a bit

Com o resultado da operação *XOR* entre as duas *strings* de entrada em uma *string* binária, agora o programa irá converter essa *string* para a base *hexadecimal*. Para isso, ele vai usar o *for* para percorrer todos os dígitos da *string* binária, nele definimos o valor da variável *col* para 0 e a condição é que o caractere, *binaryResultXorOperation[col]*, tem que ser diferente de '0', como visto no relatório 1, por exemplo. Dentro desse *for*, o programa converte inicialmente cada 4 *bits* da *string* binária para a base decimal através do método visto no relatório do exercício 1. Este processo é feito no *for* dentro do *loop* principal, em que define-se a variável *lin* para 3, ela vai diminuir uma unidade a cada *loop*, enquanto *col* vai aumentar em uma unidade. Desse modo, o programa vai ir convertendo os valores de *binaryResultXorOperation* e armazenando na variável inteira *binaryResultXorOperationToDecimal*. Logo depois dessa conversão, o programa no laço *for* principal abaixo do secundário, utiliza a função *strncat()* para concatenar o valor hexadecimal de *binaryResultXorOperationToDecimal* com o vetor *binaryResultXorOperation*. O programa passa o valor decimal como índice para a macro *BASE16TABLE*, obtém o dígito correspondente na base16 e concatena com *stringResultXor*. Logo abaixo o programa

redefine o valor de `binaryResultXorOperationToDecimal` para 0 e continua o laço de repetição principal até chegar no último caractere. Ao fim, temos o resultado da operação XOR em hexadecimal das duas strings iniciais de entrada.

```
91  >>  /** Converter binaryResultXorOperation para base 16 */
92  >>  for(col = 0; binaryResultXorOperation[col] != '\0'; ) {
93
94      //converte cada 4 bits de binaryResultXorOperation para decimal
95      >>  for(lin = 3; lin >= 0 ; --lin) {
96          >>  >>  binaryResultXorOperationToDecimal += (pow(2, lin) * (binaryResultXorOperation[col] - '0'));
97          >>  ++col;
98      >>  }
99
100     //verifica qual valor binaryResultXorOperationToDecimal corresponde em hexadecimal
101     //na macro BASE16TABLE e concatena com stringResultXor
102     strncat(stringResultXor, &BASE16TABLE[binaryResultXorOperationToDecimal], 1);
103     >>  binaryResultXorOperationToDecimal = 0;
104     >>  }
```

Imagem 10 – Conversão de `binaryResultXorOperation` para base16

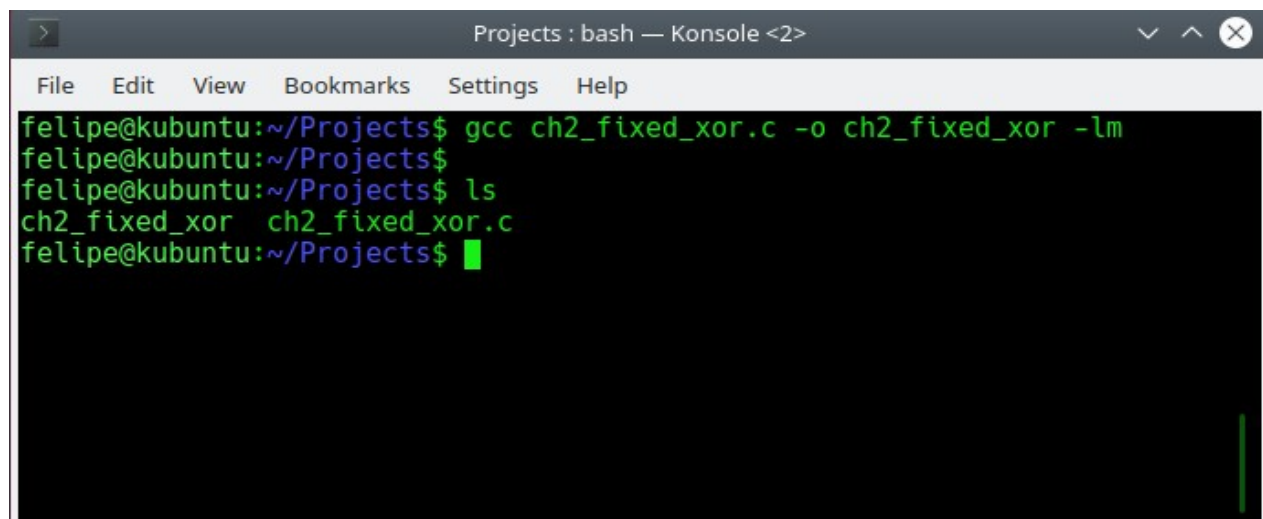
Por fim, mostramos o resultado para o usuário com o seguinte comando descrito na imagem abaixo:

```
105  >>
106  >>  printf("Fixed XOR: %s\n", stringResultXor);
107
```

Imagem 11 – Exibir resultado(`stringResultXor`)

8 – Testando o programa

O teste desse programa é bem semelhante ao que foi descrito no relatório do desafio anterior. Segue abaixo a imagem da compilação do programa.

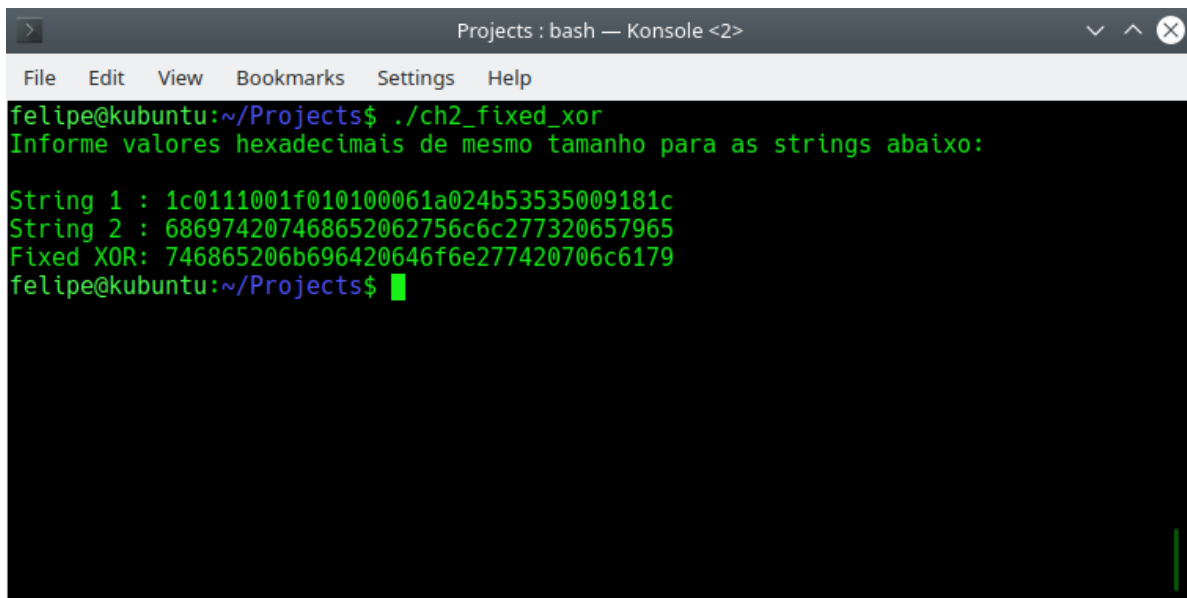
A terminal window titled "Projects : bash — Konsole <2>" with a menu bar (File, Edit, View, Bookmarks, Settings, Help). The terminal shows the following commands and output:

```
felipe@kubuntu:~/Projects$ gcc ch2_fixed_xor.c -o ch2_fixed_xor -lm
felipe@kubuntu:~/Projects$
felipe@kubuntu:~/Projects$ ls
ch2_fixed_xor  ch2_fixed_xor.c
felipe@kubuntu:~/Projects$
```

A estrutura para compilação como visto nesta imagem acima e na parte de testes do relatório do exercício é:

`gcc arquivo-de-entrada.c -o arquivo-de-saida -lm`

Para executar basta fazer `./arquivo-de-saida` . Vamos utilizar os mesmos exemplos do Cryptopals neste teste. Tudo pode ser observado na próxima imagem.



```
Projects : bash — Konsole <2>
File Edit View Bookmarks Settings Help
felipe@kubuntu:~/Projects$ ./ch2_fixed_xor
Informe valores hexadecimais de mesmo tamanho para as strings abaixo:

String 1 : 1c0111001f010100061a024b53535009181c
String 2 : 686974207468652062756c6c277320657965
Fixed XOR: 746865206b696420646f6e277420706c6179
felipe@kubuntu:~/Projects$
```