Relatório de Criptografia

Resolução dos exercícios Cryptopals Set-1

Candidato: Eduardo Archanjo Cavalcante

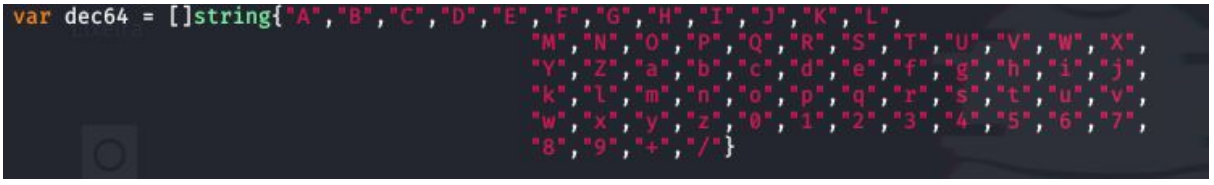
Resumo

Como parte do processo seletivo para o GRIS foi pedido a solução de três dos primeiros desafios de criptografia do site cryptopals, no qual foram escolhidos respectivamente os desafios 1 ,2 e 3.
A fim de unir com outras disciplinas, todos os desafios foram feitos na linguagem Go.

Desafio 1:

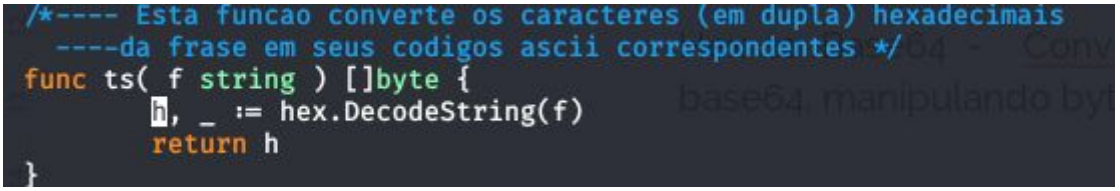
Hex to Base64 - Converta a mensagem em hexadecimal para a base64, manipulando bytes puros.

Descrição do código:



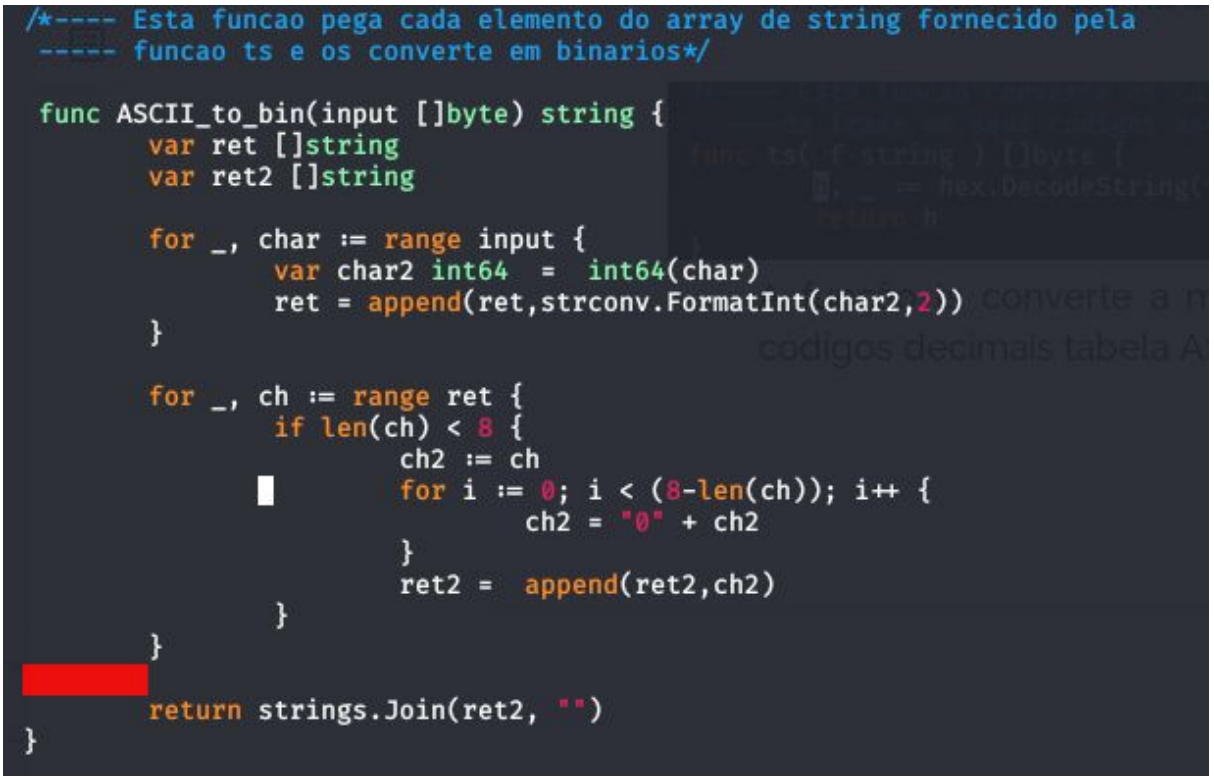
```
var dec64 = []string{"A","B","C","D","E","F","G","H","I","J","K","L",
                     "M","N","O","P","Q","R","S","T","U","V","W","X",
                     "Y","Z","a","b","c","d","e","f","g","h","i","j",
                     "k","l","m","n","o","p","q","r","s","t","u","v",
                     "w","x","y","z","0","1","2","3","4","5","6","7",
                     "8","9","+","/"}
```

- Primeiro criamos um array de strings onde cada character está em sua exata posição na base64, o que facilitará a conversão.



```
/*---- Esta funcao converte os caracteres (em dupla) hexadecimais
  ----da frase em seus codigos ascii correspondentes */
func ts( f string ) []byte {
        h, _ := hex.DecodeString(f)
        return h
}
```

- A função ts converte a mensagem em hexadecimal em seus códigos decimais tabela ASCII.



```
/*---- Esta funcao pega cada elemento do array de string fornecido pela
 ----- funcao ts e os converte em binarios*/

 func ASCII_to_bin(input []byte) string {
        var ret []string
        var ret2 []string

        for _, char := range input {
                var char2 int64  =  int64(char)
                ret = append(ret,strconv.FormatInt(char2,2))
        }

        for _, ch := range ret {
                if len(ch) < 8 {
                        ch2 := ch
                        for i := 0; i < (8-len(ch)); i++ {
                                ch2 = "0" + ch2
                        }
                        ret2 =  append(ret2,ch2)
                }
        }

        return strings.Join(ret2, "")
}
```

- A função ASCII_to_bin recebe como parâmetro a mensagem já convertida em seus bytes em ASCII e os transforma no binário correspondente dessa mensagem armazenado como string
- No primeiro loop percorremos cara byte e os convertemos em binário, já no segundo checamos cada um desses números e verificamos se possuem exatos 8 bits, caso não, adicionamos bits “0” até completar; é feito para evitar padding ao final da mensagem

```go
func bin_to_b64 (input string) string {
        //var ret string
        var array_of_bins []string

        if (len(strings.Split(input, "")) % 3) == 0 {
                input_splited := strings.Split(input, "")
                pos := 0
                p := 0
                //fmt.Println("len(input_splited)",len(input_splited))
                //fmt.Println(input_splited)

                for p < len(input_splited) {
                        if pos == len(input_splited) {break}
                        str := strings.Join(input_splited[pos:pos+6], "")
                        array_of_bins = append(array_of_bins, str)
                        pos += 6
                        p++
                }
        }

        // Convertendo binario para decimal
        var bin_int []int64
        //var str_bin string
        for _, bin := range array_of_bins{
                int_bin, _ := strconv.ParseInt(bin,2,64)
                //fmt.Println(int_bin)
                //str_bin = string(int_bin)
                                bin_int = append(bin_int, int_bin)
                }


                var list_char []string
        for _, num := range bin_int{
                list_char = append(list_char, dec64[num])
        }
                decoded_msg := strings.Join(list_char, "")

        return decoded_msg
}
```

- A função bin_to_b64 recebe o binário em string da função ASCII_to_bin e o converte diretamente na mensagem decodificada na base64
- Na primeira condição checamos se o número total de bits da mensagem é divisível por 3, assim no primeiro loop separamos a mensagem em blocos de 6 bits, para depois converter esses blocos nos decimais correspondentes na base64;
- No segundo loop fazemos essa conversão, para no último acessarmos os caracteres da lista diretamente pelo decimal da posição

```go
func main() {
        fmt.Println("\tCRYPTOPALS SET 1 - HEX TO B64:")
    ret := ASCII_to_bin(ts(hex_frase))
        ret2 := bin_to_b64(ret)
        fmt.Println("Frase em HEX: ", hex_frase)
        fmt.Println("Frase codificada em B64: ", ret2)
}
```

- Assim chamamos na main as funções para enfim converter a mensagem

Desafio 2:

Fixed XOR - Executar uma operação XOR entre dois buffers em hexadecimal de comprimentos iguais

Descrição do código:

```go
// ---- Esta funcao converte a string em hexadecimal para array de bytes ascii correspondentes
func string_to_byte(input string) []byte {
        input_to_bytes := []byte(input)
        message := make([]byte, hex.DecodedLen(len(input)))  // Cria um array de bytes prontos para receber os bytes em ascii
        hex.Decode(message, input_to_bytes)

        return message
}
```

- A função string_to_byte recebe como parâmetros um buffer em string e retorna um array de bytes em ascii correspondente;
- Utiliza a função Decode do pacotes encoding/hex que decodifica a string hexadecimal para um array de bytes

```go
func x_o_r(input1, input2 []byte) []byte {

        if len(input1) != len(input2) {
                fmt.Println("Entradas com tamanhos diferentes")
        }
        var msg []byte

        for i,_ := range input1 {
                        msg = append(msg, input1[i]^input2[i])          // "^" É o operador binario xor em go
                }
        return msg

}
```

- A função x_o_r recebe os doi buffers já convertidos em em array de bytes para aplicar a operação e retornar o buffer final
- Em Go “^” é o operador binário de xor

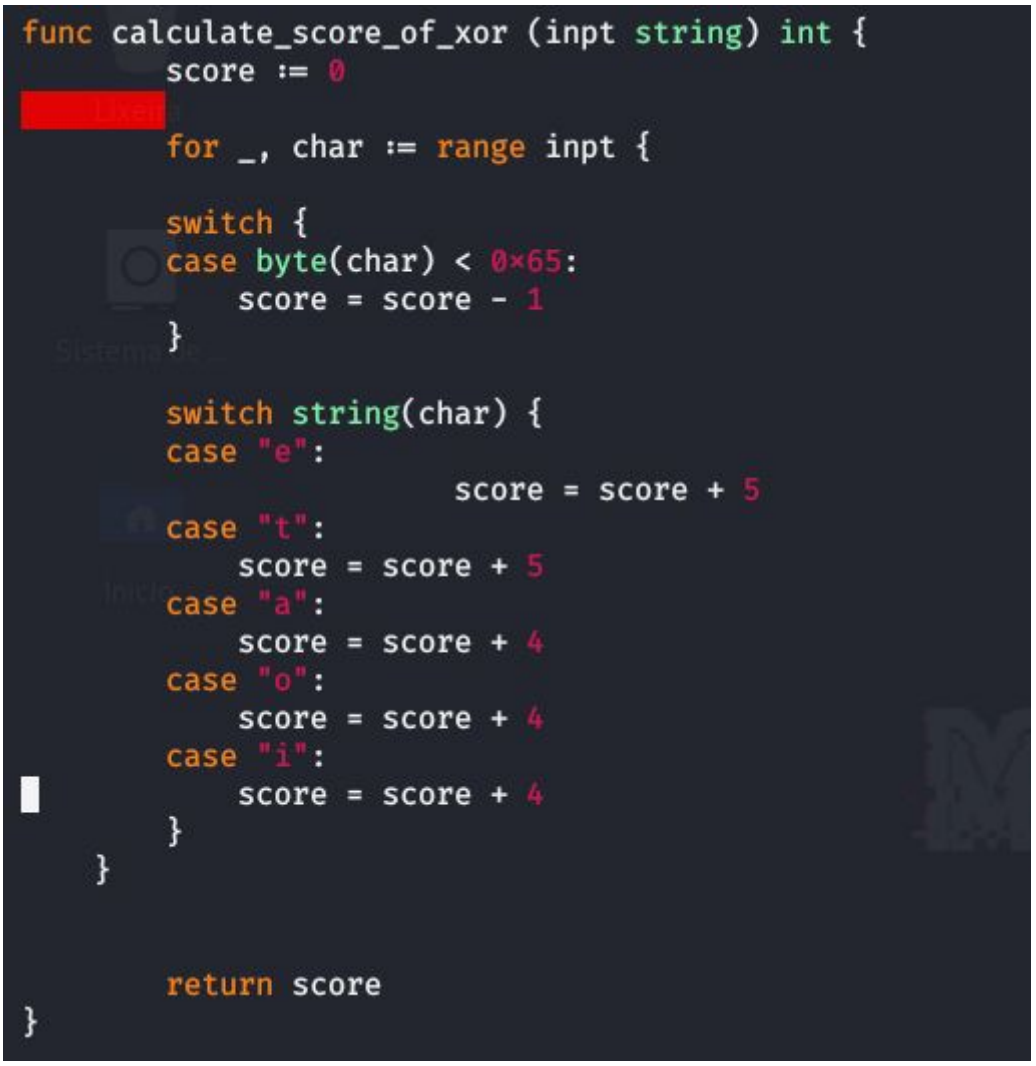
Desafio 3:

Single-byte XOR cipher - Uma mensagem em hexadecimal foi codificada com um único caractere. Deve-se encontrar o caractere e desvendar a mensagem original

A lógica deste desafio consiste em aplicar xor me todos os caracteres possíveis e através de uma contagem de pontos saber qual foi a mensagem original.
A contagem de ponto se dá ao analisar a frequência média de caracteres presentes nas palavras em inglês, vemos que as letras “e”, “t”, “a”, “o”, “i” e “n” são as mais frequentes. Ao analisar a tabela de frequência vemos que as dessas letras são muito maiores que outras letras, o que podemos usar ao nosso favor na hora de aplicar nosso algoritmo.
Então ao aplicar o xor em cada caractere temos todas as possibilidades de uma frase e em cada uma dessas possibilidades checamos as frequências as letras acima e cada ocorrência somamos a uma variável de score. A possibilidade com o maior score é a mensagem final, pois possui a maior ocorrência das letras mais frequentes.

fonte para a tabela:
http://pi.math.cornell.edu/~mec/2003-2004/cryptography/subs/frequencies.html.

Descrição do código:



```
func calculate_score_of_xor (inpt string) int {
        score := 0

        for _, char := range inpt {

        switch {
        case byte(char) < 0x65:
            score = score - 1
        }

        switch string(char) {
        case "e":
                        score = score + 5
        case "t":
            score = score + 5
        case "a":
            score = score + 4
        case "o":
            score = score + 4
        case "i":
            score = score + 4
        }
    }


        return score
}
```

- A função calcule_score_of_xor recebe como parâmetro uma das possibilidades da frase e calcula o os pontos referente as ocasiões das letras mais usadas nas frases americanas.
- Por estar com dificuldade usei a lógica da postagem que li sobre este desafio. Usa-se as cinco mais frequentes, pois suas ocorrências são bem maiores que as de outras letras e para simplificar os cálculos coloco valores mais simples e inteiros por causa das operações anteriores

```
func hex_to_string(input string) string {
    input_hex, _ := hex.DecodeString(input)

    dec_string := make([]byte, 94)
    score_for_char := make(map[int]string)
    score_array := make([]int, 94)

    for n,_ := range dec_string {                // Existem ao todo 94 characteres para se fazer xor
        array_of_xor := make([]byte, 0)

        for _, char := range input_hex { // vamos aplicar para cada character o xor e criar um array dos resultados
            array_of_xor = append(array_of_xor, byte(n + 32) ^ char)  // n+32 representa o inicio dos characteres na tabela ascii
        }

        string_possibilities := string(array_of_xor)
        score_array[n] = calculate_score_of_xor(string_possibilities)
        score_for_char[n] = string_possibilities
    }

    sortedScoreArray := make([]int, 94)
copy(sortedScoreArray, score_array)

sort.Ints(sortedScoreArray)
sort.Sort(sort.Reverse(sort.IntSlice(sortedScoreArray)))

highestScore := sortedScoreArray[0]
var decoderIndex int

for i, v := range score_array {
    if v == highestScore {
        decoderIndex = i
    }
}

//fmt.Println(score_for_char[decoderIndex])

    //fmt.Println(score_for_char)

    return score_for_char[decoderIndex]
```

- A função hex_to_string recebe como parâmetro a frase codificada e retorna decodificada
- No primeiro loop criamos uma lista para armazenar todos os XOR's de cada caractere possível, para no segundo fazermos a operação e por fim ter a possibilidade para cada caractere;
- Calculamos os pontos referente a essa possibilidade e armazenamos no dicionário de possibilidades onde temos o par chave-valor sendo caractere-possibilidade
- Por fim organizamos os pontos e pegamos a possibilidade com maior pontuação possível, que é a frase codificada no caractere correto.