Criptografia

Aprenda as técnicas de criptografia e como abordá-las em CTFs. Desde a criptografia clássica das cifras até a criptografia moderna das chaves públicas.

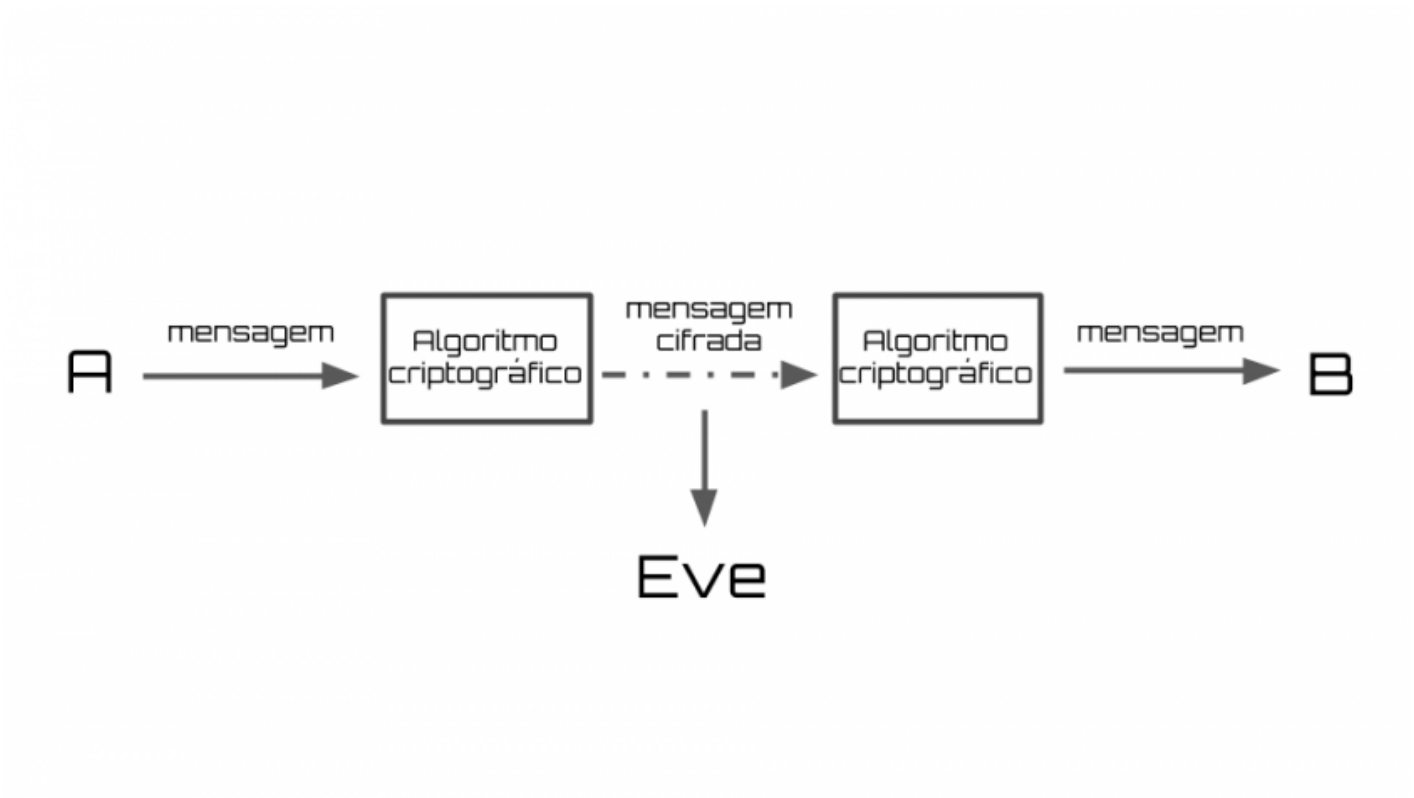
- O que é criptografia?
- Cifra de César
- Cifras de Substituição Simples
- Cifra de Vigenère
- One-Time Pad
- Cifras de Transposição

O que é criptografia?

Criptografia vem do grego *kryptós* e *graphein*, que significam "secreta" e "escrita", respectivamente. Até a era moderna, ela era sinônimo de *encriptação*, que é a conversão de uma mensagem legível para algo aparentemente sem sentido, e é esse o conceito usado em CTFs.

Para entender melhor essa ideia, digamos que *Alice* quer mandar uma mensagem para *Bob* sem que uma terceira pessoa, digamos *Eve*, descubra seu counteúdo.

Para isso, *Alice* usa um certo algoritmo para tornar a mensagem ilegível de forma que só *Bob* saberá reverter a mensagem encriptada.



Assim, quando *Eve* interceptar a mensagem por meio do canal inseguro, se ela não possui o algoritmo criptográfico usado por *Alice* e *Bob*, ela não será capaz de entender a mensagem.

Ao longo da história, várias técnicas de ocultar mensagens foram desenvolvidas. Antes da criptografia pré-computacional, a **criptografia clássica** era formada por um conjunto de métodos de *substituição* e *transpoisção* de caracteres. E com o advento da computação, a **criptografia moderna** se tornou amplamente embasada em teorias matemáticas e práticas de ciência da

computação.

Para esse guia, começaremos com os métodos da criptografia clássica.

Cifra de César

A Cifra de César é um dos métodos mais simples e comuns de encriptação. Mesmo não sendo muito comum em CTFs, ainda é um conhecimento básico de criptografia.

> “ Esse método tem esse nome pois era usado por Júlio César em suas correspondências

Nessa cifra, cada letra da mensagem é substituida por uma letra do alfabeto deslocado por um número fixo.

Por exemplo, se queremos encriptar a mensagem `hack the planet`, podemos deslocar cada letra do alfabeto **3 vezes para direita** (ou **right 3**). Assim, a substituição teria esse formato:


```
texto original: hack the planet


texto cifrado:  kdfn wkh sodqhw
```

Dessa forma, o texto gerado se torna incompreensível de forma que só quem sabe o algoritmo usado poderá recuperá-lo.

ROT13

Um dos tipos mais comuns de Cifra de César é o **ROT13**. Nele, o alfabeto é deslocado 13 vezes. Como o alfabeto tradicional possui 26 letras, o ROT13 possui a propriedade de que o mesmo algoritmo usado para encripitar a mensagem é usado para decriptar.

Detectando

Mensagens encriptadas pela cifra de césar normalmente produzirão um amontoado de caracteres sem significado, como `kdfn wkh sodqhw`, e suas letras terão uma distribuição de frequência similar à língua usada (provavelmente inglês), mas com as letras trocadas. Esse conceito será abordado com mais profundidade em Cifras de Substituição.

Devido a facilidade de quebrar essa cifra, pode ser conveniente tentar solucioná-la sem nem ao menos uma análise de frequência.

Solucionando

Como num alfabeto usual são usados apenas 26 caracteres, a Cifra de César possui apenas 25 tipos de rotações possíveis (pois a rotação 26 é a própria mensagem). Assim, um **testa tudo**, onde você faz todos os tipos de rotações possíveis, é a opção mais simples.

Existem ferramentas online muito eficientes para quebrar uma Cifra de César, como o site dcode, porém não é muito difícil codificar um *testa tudo* para isso.

Codificando um testa tudo

Primeiro, codificaremos uma função `rot()` que aplica a rotação em um caractere, de acordo com o deslocamento determinado (o `shift`):

```
def rot(char, shift):
  return chr((ord(char) - ord('A') + shift)%26 + ord('A'))
```

Assim, podemos usar essa função para criar um `caesar_brute_force()` que recebe um texto cifrado

e imprime todas as rotações possíveis.

```
def caesar_brute_force(cipher_text):
    cipher_text = cipher_text.upper()
    for i in range(26):
        line = ''
        for c in cipher_text:
            line += rot(c, i) if c.isalpha() else c
        print(f'rot{i}:\t{line}')
```

Exercícios

OverTheWire: Krypton 1

OverTheWire: Krypton 2

WeChall: Caesar

Cifras de Substituição Simples

Agora que você está familiarizado com a Cifra de César, vamos apresentar uma generalização desse conceito: as **cifras de substituição simples**.

Em uma cifra de substituição simples, cada letra é substituida individualmente de acordo com um **alfabeto de substituição**. Esse alfabeto pode ser uma rotação fixa do alfabeto normal (como a cifra de César) ou algum embaralhamento mais complexo.

Alguns exemplos notáveis de cifra de substituição simples são:

Cifra de Atbash

> “ Seu nome tem origem da primeira, última, segunda e penúltima letra Hebraica (Aleph-Taw-Bet-Shin)

Nessa cifra, cada letra é mapeada para o alfabeto invertido, ou seja, a primeira vira a última, a segunda vira a penúltima e assim por diante.

```
original: 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
cifra:    Z Y X W V U T S R Q P O N M L K J I H G F E D C B A
```

Assim, se usarmos essa cifra em `may the force be with you`, obteremos:

```
original: M A Y  T H E  F O R C E  B E  W I T H  Y O U
cifrado:  N Z B  G S V  U L I X V  Y V  D R G S  B L F
```

A Cifra de Atbash pode ser interpretada como um caso particular da Cifra de Affine, uma cifra que

usa aritimética modular para encriptar.

Cifra da Palavra-Chave

A Cifra da Palavra-Chave ou *keyword cipher* consiste em escolher uma **chave** e usá-la para decidir como as letras serão susbtituidas.

As palavras repetidas dessa chave serão removidas e a própria chave será o começo do alfabeto a ser mapeado. O resto das letras continuarão em ordem alfabética, tirando as letras já usadas.

Por exemplo, escolhendo a chave `Marvin`, o novo alfabeto terá esse formato

```
original: 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
cifra:    M A R V I N B C D E F G H J K L O P Q S T U W X Y Z
```

Assim, ao encriptar a mensagem `Arthur Dent`, obteremos:

```
original: A R T H U R  D E N T
cifrado:  M P S C T P  V I J S
```

Detectando

Como mencionado na seção de Cifra de César, uma mensagem encriptada por uma cifra de substituição simples terá uma distribuição de frequência das letras semelhante ao da língua usada, mas com as letras trocadas.

Essa distribuição de frequência de um texto pode ser identificada através de uma **análise de frequência**.

Nas línguas naturais, algumas letras aparecem mais frequentemente que outras, como uma espécie de digital do idioma. Por exemplo, a letra mais comum na lígua inglesa é o **"e"**, em português é o **"a"**.

Essa análise de frequência pode ser feita simplesmente contando as letras do texto. Existem

ferramentas online para isso como o site dcode ou pode ser feito rapidamente com um biblioteca em Python, onde `text` é o texto a ser analisado:

```
from collections import Counter
Counter(text.upper()).most_common()
```

Solucionando

O ponto fraco de cifras de substituição simples é que elas são muito suscetíveis à **análises de frequência**.

Assim, se você tiver um texto de tamanho razoável, por volta de 50 caracteres, é possível analisar a frequência com que as letras aparecem e deduzir qual foi o alfabeto de substituição usado.

O site guaballa é um excelente decodificador de cifras de substituição simples.

Referências

Cifra de Atbash: Jeremiah's Game

Cifra da Palavra-Chave: GeeksforGeeks

Learn Cryptography

Exercícios

OverTheWire: Krypton 3

Cifra de Vigenère

Devido à vulnerabilidade das cifras de substituição simples, foi necessário a criação de uma cifra que conseguisse se proteger disso. A **Cifra de Vigenère** veio com esse propóstio e é basicamente uma extensão da fórmula da Cifra de César. Ela gera uma distribuição praticamente uniforme em uma análise de frequência e foi considerada **inquebrável** por 3 séculos.

> “ Ela tem esse nome em homenagem a Blaise de Vigenère

Essa Cifra consiste basicamente em pegar uma **palavra-chave** e aplicar a cifra de César várias vezes, de acordo com os caracteres da palavra-chave.

Por exemplo, se nós queremos encriptar a mensagem `the cake is a lie` usando a palavra-chave `portal`, primeiro cada caractere da palavra-chave terá um número de rotações equivalente (de acordo com sua posição no alfabeto):

tr	**P**	**O**	**R**	**T**	**A**	**L**
rota	16	15	18	20	1	12

Assim, para cada letra da mensagem será rotacionada de acordo com a sequência de rotações acima:

```
mensagem:          T H E  C A K E  I S  A  L I E
chave:             P O R  T A L P  O R  T  A L P
mensagem cifrada:  I V V  V A V T  W J  T  L T T
```

Essa cifra, diferentemente das cifras de substituição simples, é uma **Cifra de Substituição Polialfabética**.

Detectando

Um texto encriptado por essa cifra pode ser detectado através de uma **análise de frequência**.

A Cifra de Vigenère costuma gerar textos com uma distribuição de frequência das letras próximo ao uniforme. Se um texto cifrado que não é esperado esse tipo de distribuição obter esse resultado, provavelmente é Cifra de Vigenère, ou alguma outra Cifra Polialfabética.

Solucinando

Mesmo gerando uma distribuição uniforme em análises de frequência, essa cifra tem uma vulnerabilidade: a palavra-chave é usada várias vezes em um texto grande.

Dessa forma, se a chave tiver tamanho 5, por exemplo, e ajustarmos o texto em linhas de comprimento 5, cada coluna terá a mesma rotação. Assim, podemos chutar tamanhos da palavra-chave e usar a mesma análise de cifra de substituição simples para cada coluna.

Uma ferramenta online muito útill para quebrar a Cifra de Vigenère é o site dcode.

Referências

GeeksforGeeks

Exercícios

OverTheWire: Krypton4

OverTheWire: Krypton5

picoCTF-2018: blaise's cipher

One-Time Pad

Por muitos anos, o problema de esconder os padrões da língua ainda persistia, porém no final do século XIX surgiu aquele que seria o método mais forte de criptografia: o **one-time pad**.

Funcionamento

Primeiro, precisamos gerar uma sequência **aleatória** de bits do **mesmo tamanho** da mensagem, essa será o *one-time pad*. Essa chave deverá ser passada por um meio seguro para o destinatário.

Para esse exemplo, usaremos codificação em base64 para os caracteres.

```
mensagem:     H O P E
one-time pad: y T 2 5
```

Depois, vamos criar o texto cifrado a partir da mensagem e do one-time pad. Para isso, codificaremos a mensagem e o one-time pad em binário e realizamos a operação de **ou exclusivo** bit a bit, ou **XOR**.

> A **operação XOR** de dois bits retorna 1, se eles forem diferentes, e 0, se forem iguais.

```
mensagem:      H O P E ->  000111 001110 001111 000100
one-time pad:  y T 2 5 ->  110010 010011 110110 111001
                           |
                      XOR  |
                           V
texto cifrado: 1 d 5 9 ->  110101 011101 111001 111101
```

Já para recuperar a mensagem, usamos exatamente a mesma operação, realizando um XOR bit a bit com o texto cifrado e o one-time pad.

Após **um uso**, o one-time pad deverá ser destruído.

A criptografia perfeita

No final da década de 1940, Claude Shannon provou que se cada chave for usada **uma única vez** e ela for gerada **aleatoriamente**, então o método de one-time pad é **perfeitamente seguro**.

Isso pode ser visualizado pelo seguinte exemplo: digamos que temos uma mensagem de 24 bits, logo temos 2^24 possíveis valores para a chave. A partir disso, temos dois problemas:

Poderíamos tentar **verificar todas as chaves**. Para uma mensagem de 24 bits, ainda é uma alternativa viável, mas de expandirmos para 54, mesmo se checando 1 milhão de valores por segundo, ainda levaríamos mais de 570 anos para checar todas as possibilidades.

Outro problema é que cada possível chave gera uma possível mensagem com igual probabilidade das demais, assim se checarmos todas as chaves, veríamos todas as combinações possíveis de mensagens de 24 bits.

chave	possível mensagem
A A A A	1 d 5 9
A A A B	1 d 5 8
...	...
+ J z 5	L U K E
...	...
y T 2 5	H O P E
...	...

Dessa forma, **não há como distinguir a mensagem real de todas as outras possibilidades**.

Entetanto, mesmo a técnica de One-time pad sendo simples e teoricamente inquebrável, na

prática ela possui alguns pontos negativos, que costumam trazer suas vulnerabilidades:

- A chave precisa ser **usada uma única vez**. Se repetida, ela pode ser facilmente quebrada. Essa era a principal vulnerabilidade de uma cifra semelhante, a Cifra de Vernam.
- Ainda é preciso de um **canal seguro** para distribuir as chaves.
- Conseguir **bits realmente aleatórios** é bem difícil. O exército dos Estados Unidos conseguiu decifrar, em 1944, as mensagens alemãs cifradas por one-time pad porque as chaves não eram completamente aleatórias.
- A chave precisa ser **tão longa quanto a mensagem**. Isso implica numa grande dificuldade de gerar e armazenar chaves para mensagens longas.

Esses dois últimos pontos, especialmente, acabam tornando o one-time pad teórico impraticável.

Linear Feedback Shift Register

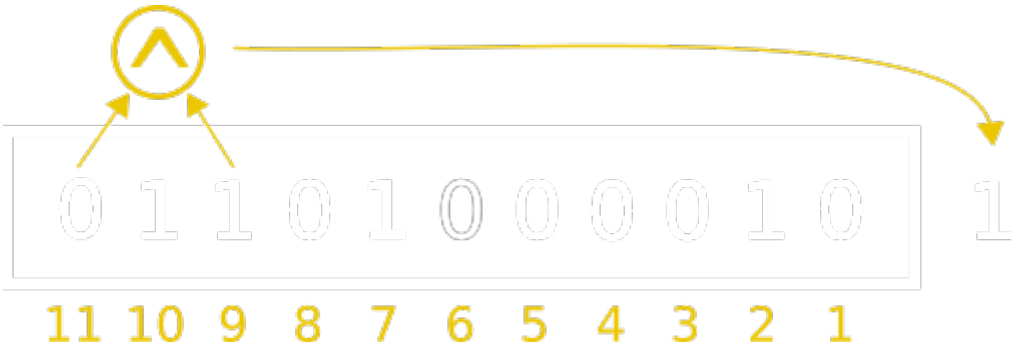
Como alternativa às longas sequências de bits realmente aleatórias, foi criado um algoritmo determinístico capaz de produzir valores **pseudo-aleatórios**: o **linear feedback shift register** ou **LFSR**.

Valores pseudo-aleatórios são sequências de bits que parecem ser aleatórias. Elas não são aleatórias de fato, pois são criadas por algoritmos determinísticos, mas, para efeitos práticos, tem as mesmas propriedades de sequências aleatórias.

Esses valores pseudo-aleatórios são criados da seguinte maneira:

Primeiro, precisamos de uma sequência de bits inicial, chamada de **seed**. Essa *seed* dará o tamanho do **registrador**, que é um elemento que armazena todos os bits necessários para gerar o próximo.

Depois, geraremos uma nova sequência através da operação *XOR* de dois bits, colocando os bits resultantes à direita.



Por exemplo, a imagem acima representa o registrador com a *seed*. Nela, os bits 11 e 9 foram escolhidos para gerar o próximo. Essas posições são chamadas de **tap**, ela é uma numeração que começa do 1, indo da direita para esquerda.

Para descrever essas posições do algoritmo, usamos a notação `[N, k] LFSR`, que representa um algoritmo de LFSR com um registrador de `N` bits com *taps* em `N` e `k`. Na imagem acima, temos um `[11, 9] LFSR`.

Abaixo, está uma pequena simulação de um `[5, 4] LFSR` com a *seed* `00001`.

```
[5, 4] LFSR

            novo bit
               V
seed ->  00001 0
         00010 0
         00100 0
         01000 1
         10001 1
         00011 0
         00110 0
         01100 1
         11001 0
         10010 1
         00101 0
           ^
      registrador
```

Assim, precisamos apenas criar uma lista de pequenas *seeds* e distribuí-las por um canal seguro, usando elas para criar chaves do one-time pad, por meio do LFSR.

Exercícios

Krypton 6

Referências

Khan Academy

Computer Science - Sedgewick & Wayne

Mensagens alemãs não aleatórias

Cifras de Transposição

Cifras de Transposição

Content about Cifras de Transposição.

Exercícios

WeChall: Transposition I