

# 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 se 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

---

Revision #3

Created Tue, Nov 13, 2018 3:05 PM by Andrew

Updated Thu, Mar 7, 2019 10:08 PM by Andrew