Criptografia: Parte 1

O que criptografia quer proteger?

- Proteger informação em transito (e.g HTTPS)
- Proteger informação assíncrona (e.g email)
- Proteger informação offline (e.g encriptografia de informação no disco)

Princípios

- **Confidencialidade :** restringir quem tem conhecimento da informação (e.g cifras simétricas, assimétricas, acordos de chaves).
- **Autenticidade (e integridade):** entre o emissor A e um receptor B temos a certeza que os dados vieram de A (sem alteração).
- **Não repúdio:** assinaturas digitais. Quando envia uma mensagem autenticada, não posso mais negar o envio da mensagem.

Confidencialidade

Um princípio fundamental da criptografia são as funções **oneway.**

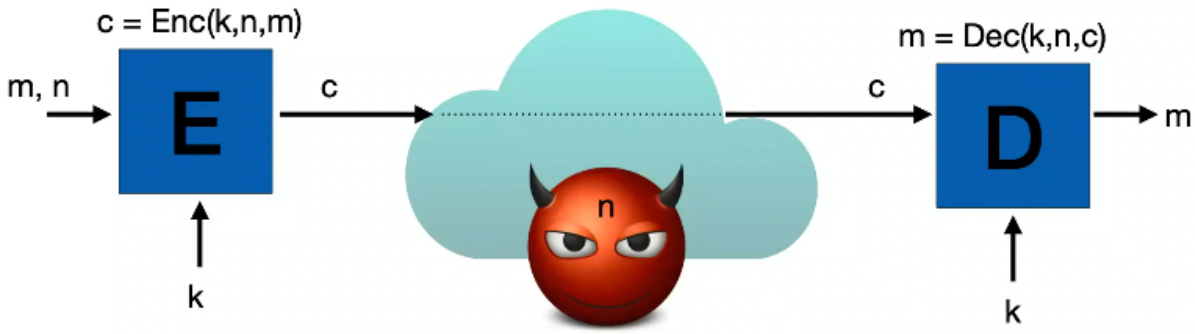
Temos os problemas **P** que são aqueles que conseguimos achar as soluções em tempo polinomial e há os problemas do tipo **NP,** que são aqueles que conseguimos **verificar** uma solução em tempo polinomial. O ponto é que criptografia usa estes problemas **NP** para garantir segurança.

O facto é que criptografias precisam ser descritas, formalmente bom base em modelos matemáticos.

E a construção deve ser justificada formalmente.

Cifras simétricas

- **Cifra:** uma primitiva que dá `confidencialidade`. Ou seja, restringimos o acesso ao conhecimento.
- **Simétrica:** usamos a mesma chave dos dois lados da comunicação.



Temos então:

- **k:** chave que deve ser igual para os dois lados (128 bits)
- **m:** a mensagem a ser transmitida
- **n:** nonce (not more than once), é um parâmetro público adicional que pode ser modificado.
- **c:** output

Do outros lado da comunicação temos um algoritmos que pode inverter a mensagem. Temos de então fornencer o **nonce** e a **chave.**

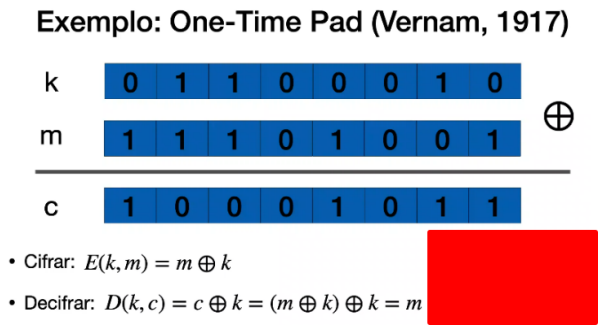
> **Porquê utilizamos chaves de 128bits.** Atualmente estimasse que o nosso poder computacional gira em torno de 2^80 (80 bits). Então não conseguimos encontrar por brute force chaves de 128 bits. (https://crypto.stackexchange.com/questions/13299/is-80-bits-of-key-size-considered-safe-against-brute-force-attacks)

Casos de uso

- **Chave é usada apenas uma vez (one-time-key** e toda vez que envíamos uma mensagem, ela é gerada novamente. Um caso destes é o **email.** Por isso, neste caso o nonce não é relevante: pode ser descartado.
- **Chave é usada muitas vezes (many-time-key).** Ela é usada muitas vezes, por exemplo, em comunicações HTTPS/TLS. Neste caso o nonce não pode se repetir.

One-time Pad

Fazemos um **xor** entre a mensagem e a key.



> Foi provado que o One-Time Pad não revela nenhuma informação sobre o conteúdo da mensagem (considerando que não temos um man in the middle.

O problema é:

- A chave só pode ser utilizada uma vez.
- A chave tem de ser do tamanho do texto lido, portanto, se estivermos a encriptografar ficheiros grandes (e.g 1GB, 1TB), a chave será enorme.

Portanto, gerar chaves diferentes para textos grandes muitas vezes não é factível.

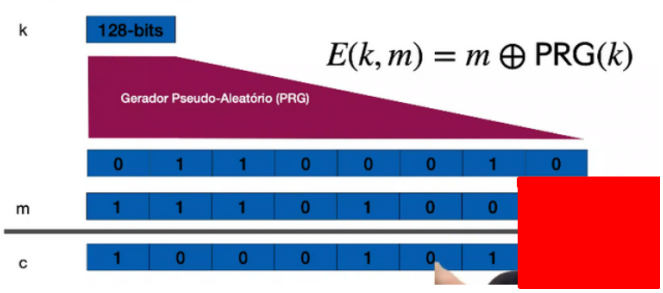
Cifras sequenciais

Resolver o problema de keys muito grandes

Precisamos consertar o problema

de chaves do tamanho do text para o caso do `one-time pad`.

Podemos ter aqui uma chave **k** com 128 bits. Então usamos um **gerador pseudo-aleatório (PRG)** para gerar uma string aleatória que possa cifrar a nossa mensagem.



Resolver o problema de keys repetidas

Se eu usar a mesma **key** o output do **PRG(k)** será igual para todas as mensagens. Isto é inseguro porque:

$$c_1 = m_1 \oplus PRG(k)$$
$$c_2 = m_2 \oplus PRG(k)$$
$$c_1 \oplus c_2 = m_1 \oplus m_2$$

Os `PRGs` modernos nos permitem usar `nounce públicos`.

Cifras de blocos (PRG)

`Não são cifras! Apenas permitem contruí-las`

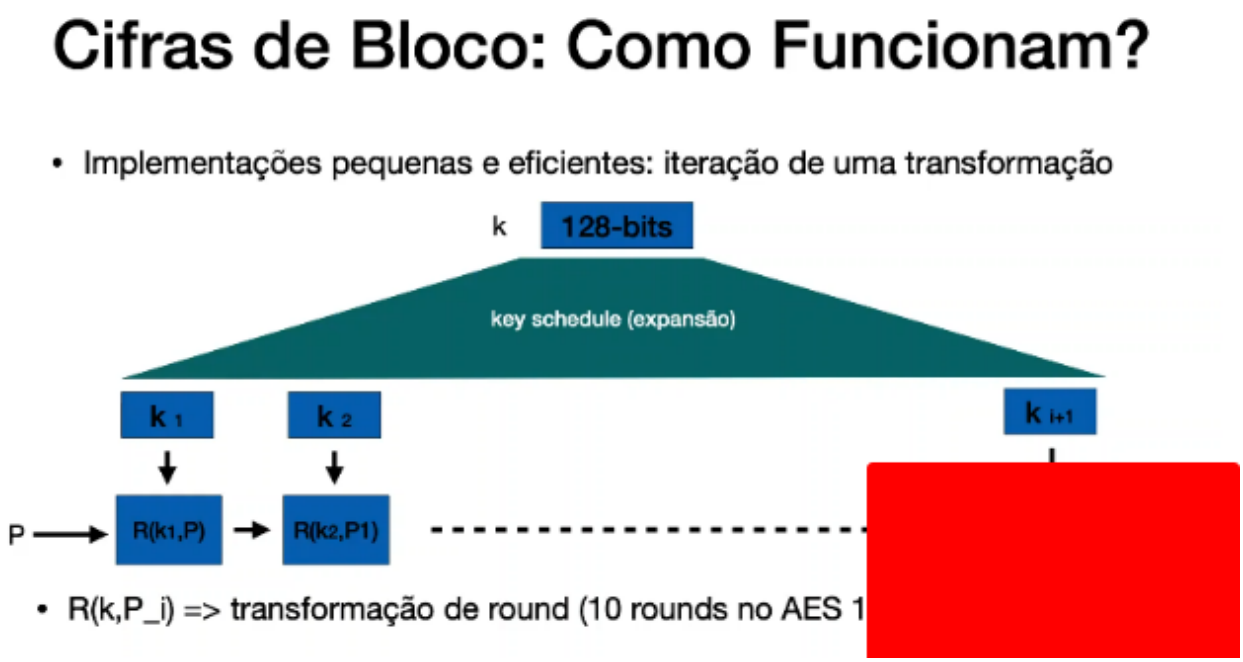
Esta é função mais popular de PRG.

Ninguem nunca conseguiu provar, mas acredita-se que:

- Para k aleatório e secreto
- E(k,B) permanece aleatório, onde B é um bloco de 64 bits
- Mesmo escolhendo B

Porque se nos derem dois outputs de E (um encriptografando a nossa mensagem e outro não), não conseguimos dizer qual deles refere-se a nossa mensagem.

Cifras de blocos: AES



AES é muito utilizado.

Perguntas

1) O que é o one-time-pad? Qual o problema dele?

▼ Resposta

O one-time-pad é simplismente gerarmos uma chave qualquer com o mesmo tamanho do texto a ser enviado. O problema é que a chave é apenas usada uma vez e para textos grandes, criar uma chave aleatória é um processo muito lento.

2) Qual a solução que se achou para o one-time-pad?

▼ Resposta

Pode-se usar o PRG (pseudo random generator) onde, dado uma key, consegue-se gerar a sequência de bytes para realizar o xor. O problema é que se não for dado um nonce o valor é sempre o mesmo.

3) Por que utilizamos nonce?

▼ Resposta

O xor possui a seguinte propriedade

m1 xor k = c1 e m2 xor k = c2, então c2 xor c1 == m1 xor m2.

Se não usarmos um nonce a confidencialidade por ser comprometida.

4) O que torna as cifras de bloco AES atualmente tão populares?

▼ Resposta

A implementação é suportada a nível de hardware.

5) Por que é inseguro usar a mesma cifra de bloco para cifrar um plain-text inteiro? Explique a situação com um exemplo.

▼ Resposta

Um exemplo é o Eletronic Code Book (ECB). Ao cifrarmos, por exemplo, uma imagem com a mesma cifra de bloco, temos que as mesmas cores irão ter os mesmos padrões.
Com isto no fim é possível identificar e interpretar uma imagem cifrada desta forma (slide 19).

6) Cite duas cifras de bloco consideradas seguras.

▼ Resposta

Cipher block chain e Counter Mode.