PMR3412 - Redes Industriais - 2023

Aula 09 - Segurança: Conceitos Básicos - Criptografia Simétrica e Assimétrica

Prof. Dr. Newton Maruyama

12 de Outubro de 2023

PMR-EPUSP

Notas de autoria (Disclaimer) - Equipe envolvida

Os slides que serão utilizados nesse ano são baseados no curso desenvolvido para os anos 2020, 2021 e 2022. Participaram da concepção do curso e desenvolvimento do material os seguintes professores:

- Prof. Dr. André Kubagawa Sato
- Prof. Dr. Marcos de Sales Guerra Tsuzuki
- Prof. Dr. Edson Kenji Ueda
- Prof. Dr. Agesinaldo Matos Silva Junior
- Prof. Dr. André César Martins Cavalheiro

Conteúdo

Revisão

Criptografia - Princípios básicos

- Princípio de Kerckhoff: **um sistema criptográfico deve ser seguros mesmo se tudo é conhecido sobre ele, exceto a chave**.
- A criptografia é a principal ferramenta para providenciar proteção para informação. Ela fornece as seguintes proteções:

Criptografia Simétrica

Criptografia Simétrica - Introdução

- A criptografia simétrica é a fundação de toda a comunicação segura moderna. Nela, uma chave é utilizada em ambas as pontas da comunicação, tanto para criptografar como para descriptografar.
- Exemplo de código usando AES no modo ECB (**OBS: não seguro!**)

```
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
from cryptography.hazmat.backends import default_backend
import os

key = os.urandom(16)
aesCipher = Cipher(algorithms.AES(key), modes.ECB(), backend=default_backend())
aesEncryptor = aesCipher.encryptor()
aesDecryptor = aesCipher.decryptor()
```

- Note que a função `update` funciona de 16 em 16 bytes. Para testar, utilize os seguintes comandos:

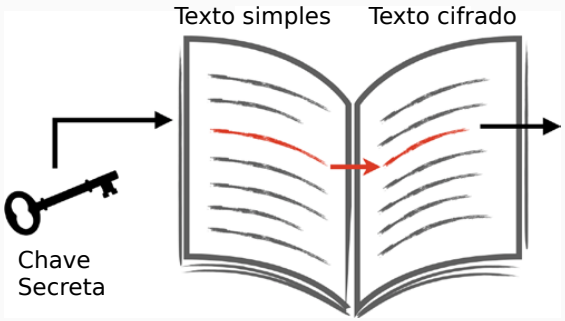
```
aesEncryptor.update(b'alice') # saida: b''
aesEncryptor.update(b'bob') # saida: b''
aesEncryptor.update(b'bob') # saida: b''
aesEncryptor.update(b'bob') # saida: b''
aesEncryptor.update(b'bob') # saida: b'\xe7\xf9\x19\xe3!\x1d\x17\x9f\x80\x9d\xf5\xa2\xbaTi\xb2'
```

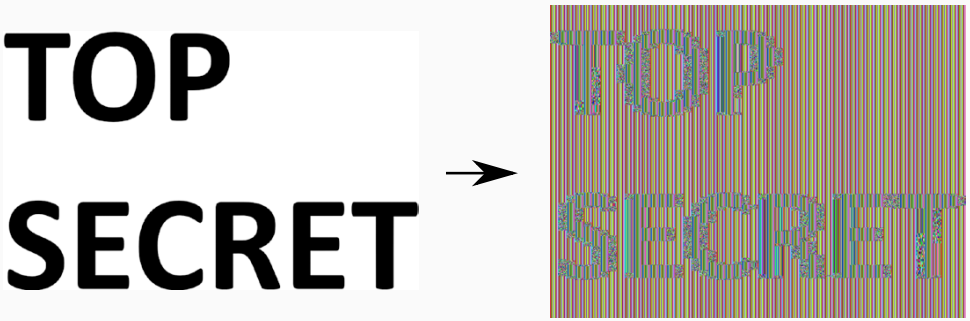
- Quando alguém ouve algo sobre criptografia, muito provavelmente "encriptação" (cifragem) é o que vêm à mente.
- Sites e serviços recorrentemente mencionam a utilização de encriptação para demonstrar que são "seguros". A utilização de encriptação é um requisito mas não necessariamente leva a sistemas seguros.
- Como vimos antes, existem pelo menos três propriedades geralmente consideradas como essenciais:
  1. **confidencialidade**: apenas as partes com a chave correta conseguem ler os dados;
  2. **integridade**: os dados não podem ser modificados sem você perceber;
  3. **autenticação**: relativo ao conhecimento da identidade da pessoa com quem você está se comunicando.
- Cada propriedade contribui para diferentes partes da segurança, e devem trabalhar em conjunto. No restante desta aula, abordaremos apenas a propriedade de confidencialidade.

- Como mencionado anteriormente, encriptação simétrica significa que a mesma chave é usada para criptografar e para descriptografar.
- Existem dois tipos de algoritmos para cifra com chave simétrica:
  - cifras de bloco: trabalha com blocos de dados. Todos os blocos devem estar preenchidos;
  - cifras de *stream*: conseguem codificar um byte por vez.
- O AES é uma das cifras mais utilizadas. Ele é utilizado no TLS (HTTPS), IPSec e em criptografia de discos.
- O AES é essencialmente uma cifra de bloco, mas pode operar de modo similar a uma cifra de stream, dependendo do modo de operação. Os principais modos de operação incluem:

Criptografia Simétrica - AES: Não utilize o modo ECB

- ECB pode ser entendido como o modo "puro" do AES, ele trata cada bloco de 16 bytes independentemente, codificando cada um da mesma maneira.
- Uma vez que possuímos a chave, todos os blocos de 16 bytes cifrados podem ser traduzidos (e o inverso também), como se estivéssemos consultando um livro.
- Sendo assim, possui as propriedades de determinismo e independência, que são úteis, mas não são suficientes para a segurança.
- Um dos problemas ocorre quando as mensagens possuem estruturas semelhantes, o que acontece com arquivos XML, HTML e emails. Como são sempre iguais, o texto cifrado de muitos blocos também se repetirão. Veja exemplo de imagem binária apresentado abaixo.





- Com as limitações do ECB em vista, definimos que, para uma cifra efetiva, temos duas necessidades:

1. Criptografar uma mesma mensagem de forma diferente toda vez:
   - Solução: Initialization Vector (IV), que é uma string aleatória de conhecimento público.
   - Serve como uma terceira entrada (IV + chave + texto simples), semelhante ao sal no hashing.
2. Eliminar padrões previsíveis entre blocos:
   - Solução: encriptação da mensagem de modo integral.
   - Para cifras de bloco, é desejável uma propriedade similar ao "avalanche" de hashes.
   - No modo ECB, essa propriedade esta limitada a um único bloco.

- No modo CBC, a alteração do texto cifrado de um bloco acarreta mudanças em todos os blocos seguintes. Isto é feito a partir de operações XOR.
- Considere:
  - $P[n]$: o bloco n de texto simples;
  - $P'[n]$: o bloco n de texto simples "embaralhado" com operação XOR, antes de ser encriptado; e
  - $C[n]$: o bloco n de texto cifrado.

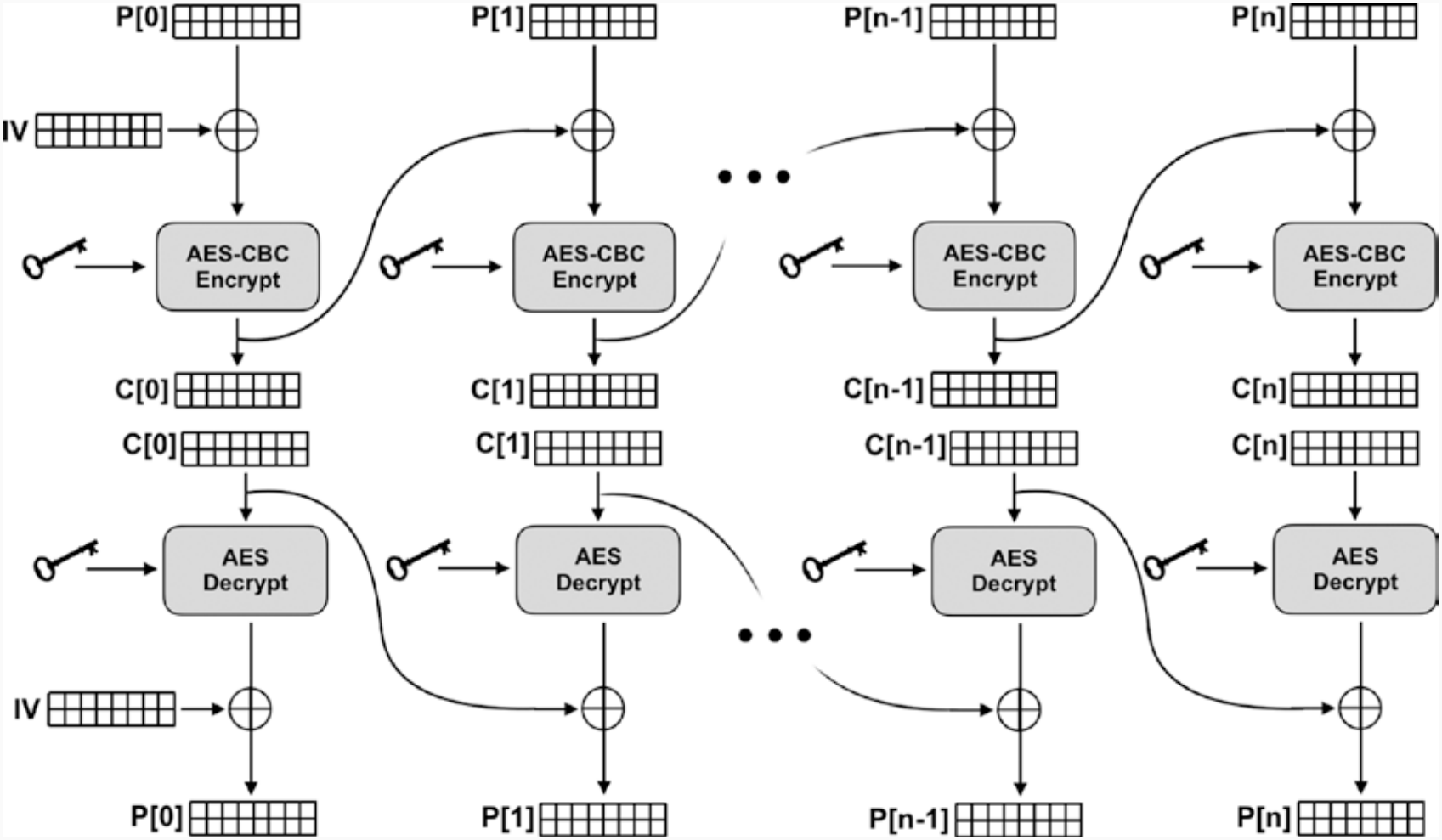
  Software abstraction
- Sendo assim, temos que, para criptografar:

$$P'[n] = P[n] \oplus C[n-1]$$

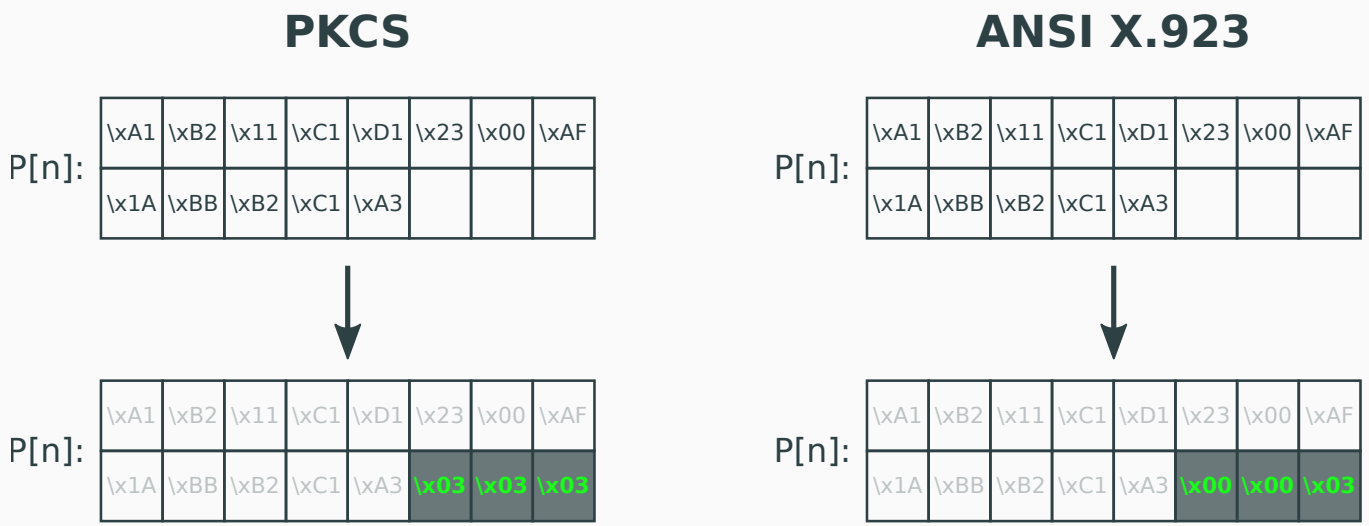
- Lembrando que a operação XOR é a próprio inversa ($(A \oplus B) \oplus B = A$), podemos descriptografar com:

$$P[n] = P'[n] \oplus C[n-1]$$

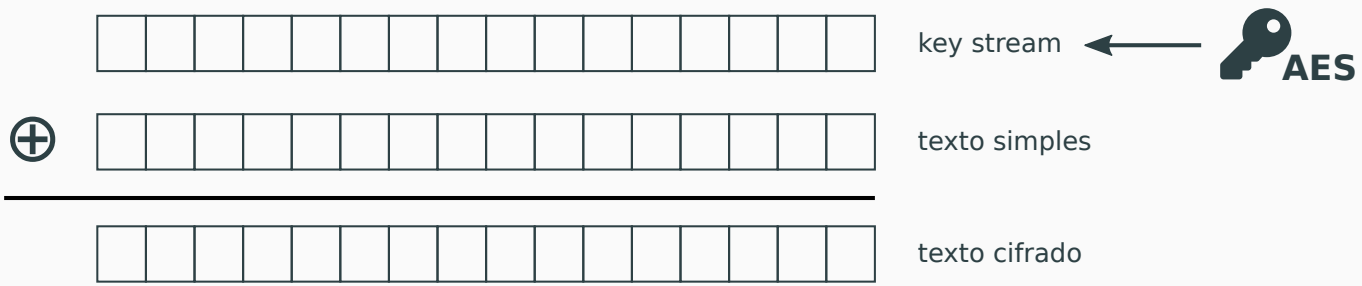
Criptografia Simétrica - AES: Visão geral do modo CBC



- Relembrando: para cifras de bloco, a encriptação é feita bloco a bloco. Sendo, assim, todos os blocos devem ser preenchidos. Este processo é chamado de *padding*.
- Por razões que não serão abordadas nesta aula, esta aparente inofensiva operação pode ser fonte de vulnerabilidade.
- O módulo `cryptography` do Python disponibiliza dois esquemas de padding: PKCS e ANSI X.923. Exemplo:



- O modo CTR é inusitado: AES não é utilizado para encriptar os dados! 😲
- Na realidade, o AES é utilizado para gerar uma *key stream* com o mesmo tamanho do texto simples. É então realizado um XOR do *key stream* com os dados para obter o texto cifrado (onetime pad, ou OTP).
- O processo de geração da *key stream* é o seguinte:
  1. Inicialize um contador de 16 bytes, encripta com o AES, gerando os primeiros 16-bytes do *key stream*.
  2. Incrementa o contador.
  3. Gera mais 16 bytes com o novo contador e junta com o *key stream*.
  4. Repete os passos 2-3 até conseguir o tamanho desejado do *key stream* (= ao do texto simples).



- Diferentemente dos outros modos abordados, o AES-CTR é uma cifra de *stream*. Sendo assim, não necessita de *padding*.
- No entanto, podemos pensar em um bloco de cada vez. Para codificar um dado bloco n, deve-se gerar a *key stream* referente ao índice do bloco e fazer o XOR com o bloco:

$$C[n] = P[n] \oplus n_k \text{ , onde } k \text{ indica "encriptado com a chave } k\text{".}$$

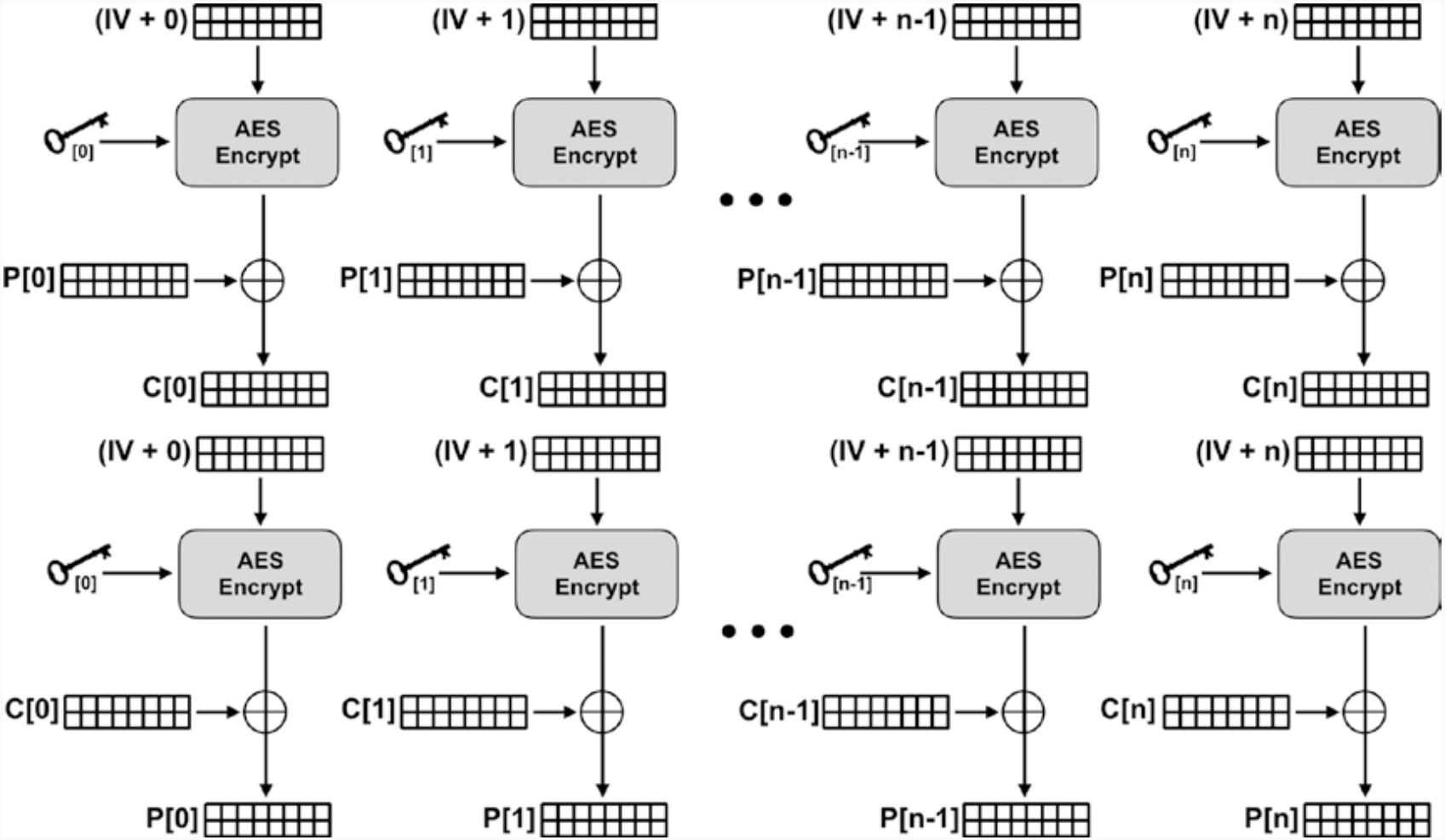
- Para evitar que o contador inicie sempre com o mesmo valor, vamos introduzir um IV (initialization vector), aqui chamado de "nounce":

$$C[n] = P[n] \oplus (IV + n_k).$$

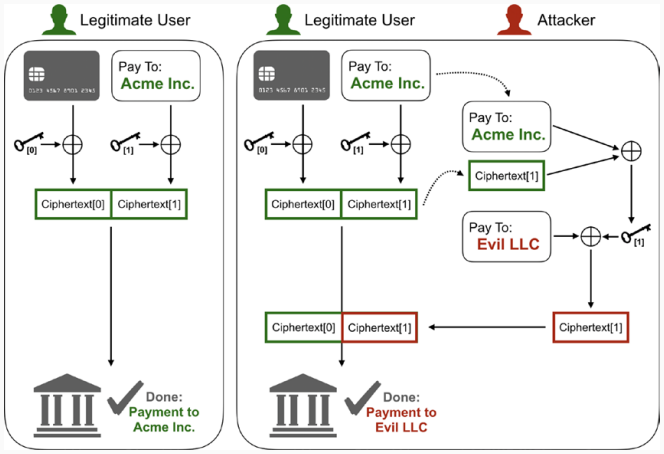
- Por fim, para descriptografar, basta fazer

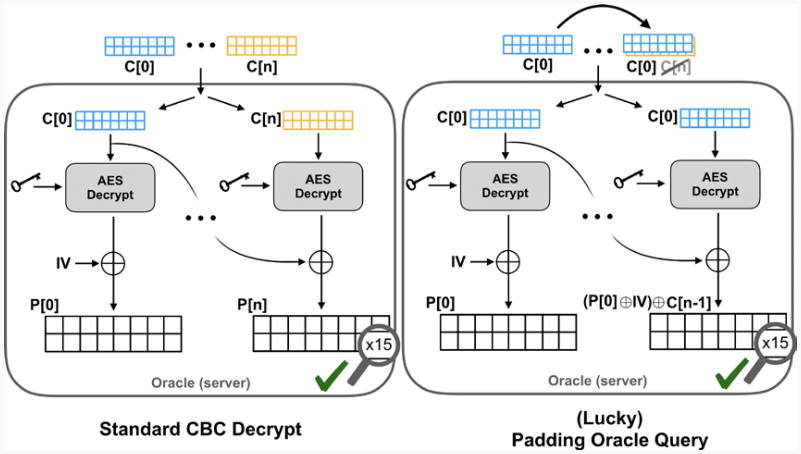
$$P[n] = C[n] \oplus (IV + n_k).$$

Criptografia Simétrica - AES: Visão geral do modo CTR



- Nunca utilizar o mesmo par chave/IV!
- No modo CBC, reutilizar o par chave/IV resultará em blocos repetidos de texto cifrado para cabeçalhos comuns (como por exemplo HTML).
- No modo CTR, é ainda pior! Se você sabe o texto simples, você pode obter a chave com $K \oplus P \oplus P = K$. Uma vez que você reutilize o par chave/IV, é possível descriptografar outras mensagens.
- Ainda que não reutilize o par chave/IP, é possível modificar uma mensagem (maleabilidade), mesmo sem conseguir decodificá-la. Lembre que o algoritmo AES garante apenas confidencialidade.
- **Lição:** A encriptação sozinha não é capaz de proteger os dados.





- Gerenciamento de chaves é umas das tarefas mais complexas de sistemas criptográficos. Isso inclui a sua geração, compartilhamento, atualização e revogação.
- Para o caso particular de geração de chaves, é necessária uma fonte com uma boa aleatoriedade. Por exemplo, o código abaixo é inseguro:

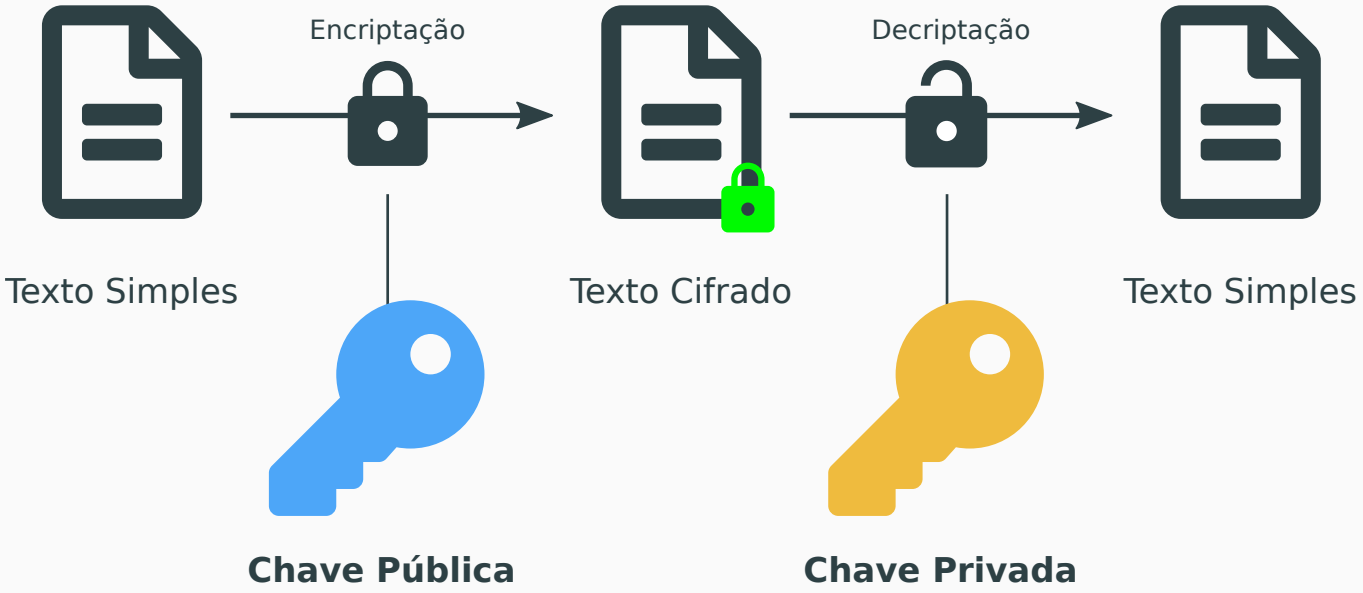
```
import random
key = random.getrandbits(16, "big")
```

- O gerador de números pseudo-aleatório do Python é baseado no horário do sistema. Se um invasor consegue adivinhar o horário que a chave foi gerada, pode prever os números. Solução: utilizar `os.urandom()`.
- Outra opção é derivar a chave de uma senha. Nesse caso, a senha deve ser bem segura! Utilize algoritmos prontos (scrypt, bcrypt ou Argon2) para derivar a chave.

Criptografia Assimétrica

Criptografia Assimétrica - Chaves Pública e Privada

- A criptografia assimétrica é um dos mais importantes avanços em segurança criptográfica. Está presente em toda segurança da Web, nas conexões Wi-Fi, sistemas seguros de email e outras comunicações.
- Na encriptação assimétrica, são utilizadas duas chaves: o que é encriptada com uma chave só pode ser decriptada com a outra chave!
- Em uma comunicação entre duas pessoas, cada uma possui um par de chaves (pública + privada), além da chave pública da outra pessoa.



- Nessa aula utilizaremos o RSA, que é um algoritmo praticamente obsoleto, mas permite uma compreensão geral de algoritmos de criptografia assimétrica.
- Considerando c o texto cifrado e m a mensagem, temos que, para encriptação:

$$c \equiv m^e (mod\ n)$$

- Na decriptação, temos:

$$m \equiv c^d (mod\ n)$$

- onde os parâmetros d, e e n se originam das chaves privada e pública.
- No RSA, a chave pública é derivada da chave privada. Para gerá-las, é necessário encontrar um par de números inteiros com grande chance de serem co-primos.

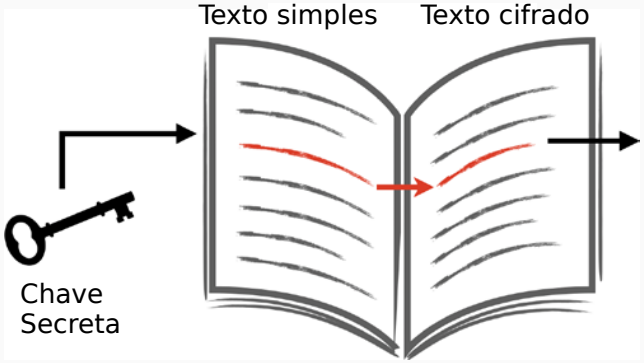
- As chaves RSA podem ser geradas com o código:

```
from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives.asymmetric import rsa
from cryptography.hazmat.primitives import serialization

# Generate a private key.
private_key = rsa.generate_private_key(public_exponent=65537, key_size=2048, backend=
        default_backend())

# Extract the public key from the private key.
public_key = private_key.public_key()
```

- E a encriptação/decodificação com RSA pode ser ingenuamente (e perigosamente) implementado com o código:

```
def simple_rsa_encrypt(m, publickey):
    numbers = publickey.public_numbers()
    return gmpy2.powmod(m, numbers.e, numbers.n) # Encryption is(m^e) % n.

def simple_rsa_decrypt(c, privatekey):
    numbers = privatekey.private_numbers()
    return gmpy2.powmod(c, numbers.d, numbers.public_numbers.n) # Decryption is(c^d) % n.
```

- A técnica de *padding* deve ser utilizada aqui para introduzir um bloco não determinístico.
- **Aviso:** não utilize RSA sem *padding*, pois é muito fácil "quebrar"! A biblioteca `cryptography` sequer permite utilizar RSA sem *padding*.
- Um dos problemas do RSA sem *padding* é o seu determinismo: a mesma entrada gera a mesma saída. Lembra da figura abaixo?



- Como o invasor possui a chave pública, é possível descobrir o texto cifrado utilizando "tentativa e erro".
- Permite ataque do tipo *Common Modulus Attack*, que se baseia em chaves públicas com o mesmo valor de n.

- Outro problema é a maleabilidade, i.e., capacidade de modificar o texto cifrado de modo útil.
- O RSA é particularmente afetado devido a sua propriedade homomórfica: o produto de dois texto cifrados é descriptografado para o produto dos dois textos simples:

$$(m_1)^e(m_2)^e(mod\ n) \equiv (m_1 m_2)^e(mod\ n)$$

- Isto pode ser explorado pelo *Chosen Cyphertext Attack*, que se baseia em persuadir a vítima a decodificar alguns texto cifrados escolhidos por você.
- **Conclusão**: utilize *padding* com elementos aleatórios, o que geralmente é empregado quando o RSA é adotado. Dois dos esquemas de padding são: PKCS #1 v1.5 e OAEP (preferível).

Criptografia Assimétrica - Qual é o diferencial?

- Em criptografia **simétrica**, qualquer pessoa com o poder para gerar uma mensagem criptografada tem a habilidade de decodificá-la.
- Em criptografia **assimétrica**, existe uma chave privada que não deve ser divulgada e uma chave pública que pode ser distribuída. O que pode ser feito com o par de chaves depende do algoritmo.
- No RSA, é possível utilizar uma chave para encriptar e outra para descriptografar (ambas as chaves pode assumir qualquer um desses papéis). Com isso, é possível realizar duas operações:

Dropbox Criptográfico
(confidencialidade)

Chave pública: encriptação
Chave privada: decriptação

Assinaturas
(identidade)

Chave privada: encriptação
Chave pública: decriptação

Note: com criptografia assimétrica, duas pessoas não precisam se encontrar antes de começar a trocar a mensagem

Referências

Referências - Aula 09

- Capítulos 3 e 4 do livro “Practical Cryptography in Python: Learning Correct Cryptography by Example” deSeth James Nielson e Christopher K. Monson.

The End!