Redes de Computadores – Unioeste
Prof. Luiz Antonio

Aula Prática 09 - Criptografia

Documentação da API PyCryptodome: https://pycryptodome.readthedocs.io/en/latest/index.html

Princípios básicos

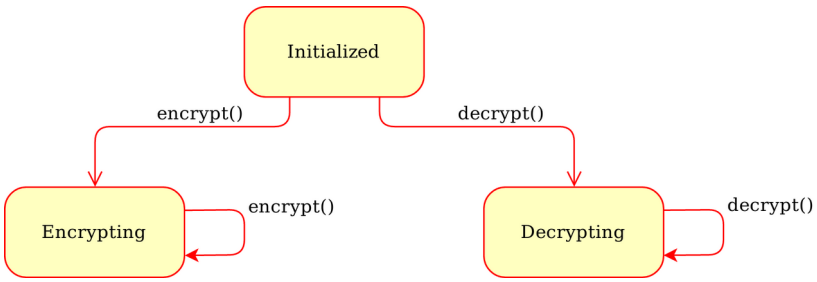

Figura 1: Diagrama básico.

O Pacote Cypto.Cipher possui algoritmos para criptografia simétrica e assimétrica, tanto para fluxo (Ex. Salsa20) quanto para blocos (Ex. AES).

1. **Criptografia Simétrica**

Cifra de fluxo Salsa20 → https://pycryptodome.readthedocs.io/en/latest/src/cipher/salsa20.html

Cifra de bloco AES → https://pycryptodome.readthedocs.io/en/latest/src/cipher/aes.html
Outras opções: DES e 3DES (*deprecated*)

Atividade 1: Uma solução de comunicação por socket com AES: ver arquivos aes_cliente e aes_servidor em anexo.

2. **Criptografia Assimétrica**

O RSA é o algoritmo padrão. Inicialmente gere o par de chaves.

https://pycryptodome.readthedocs.io/en/latest/src/examples.html#generate-public-key-and-private-key

```
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP
from Crypto.Random import get_random_bytes
import base64

# Geração de chave pública e privada
chave = RSA.generate(2048)
chave_publica = chave.publickey()
chave_privada = chave

print(chave_publica.exportKey())

# Função de criptografia com a chave pública
```

```
def criptografar_com_publica(mensagem, chave_publica):
cipher = PKCS1_OAEP.new(chave_publica)
return base64.b64encode(cipher.encrypt(mensagem.encode())).decode()

# Função de descriptografia com a chave privada
def descriptografar_com_privada(mensagem, chave_privada):
cipher = PKCS1_OAEP.new(chave_privada)
return cipher.decrypt(base64.b64decode(mensagem)).decode()

# Teste
mensagem_original = "Mensagem secreta"
criptografada = criptografar_com_publica(mensagem_original, chave_publica)
print(f"Criptografada: {criptografada}")
descriptografada = descriptografar_com_privada(criptografada, chave_privada)
print(f"Descriptografada: {descriptografada}")
```

Na criptografia **assimétrica**, normalmente usamos a **chave pública** para criptografar e a **chave privada** para descriptografar. Isso garante **confidencialidade**: somente o dono da chave privada pode ler a mensagem.

No entanto, o inverso também é possível: **criptografar com a chave privada e descriptografar com a chave pública**. Isso não garante confidencialidade, pois qualquer pessoa com a chave pública pode abrir a mensagem, mas garante **autenticidade**—ou seja, a mensagem veio do dono da chave privada e não foi alterada.

Atividade 2: modifique o código anterior para implementar este processo.

Servidor AES em Python

```
import socket
from Crypto.Cipher import AES
import base64

# Chave secreta compartilhada (deve ter 16, 24 ou 32 bytes)
chave_simetrica = b"minha_chave_secreta1234"  # 16 bytes

def descriptografar(mensagem_cifrada):
    dados = base64.b64decode(mensagem_cifrada)
    nonce, tag, ciphertext = dados[:16], dados[16:32], dados[32:]

    cipher = AES.new(chave_simetrica, AES.MODE_EAX, nonce=nonce)
    try:
        mensagem = cipher.decrypt_and_verify(ciphertext, tag)
        return mensagem.decode()
    except ValueError:
        return "Erro: Mensagem adulterada!"

# Configuração do servidor
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind(("0.0.0.0", 12345))  # Escuta na porta 12345
server.listen(1)

print("Aguardando conexão...")
conn, addr = server.accept()
print(f"Conectado a {addr}")

while True:
    dados = conn.recv(1024)
    if not dados:
        break
    mensagem_decifrada = descriptografar(dados.decode())
    print(f"Mensagem recebida: {mensagem_decifrada}")
conn.close()
```

```
server.close()
```

Cliente AES em Python

```
import socket
from Crypto.Cipher import AES
import base64

# Chave secreta compartilhada (deve ser igual à do servidor)
chave_simetrica = b"minha_chave_secreta1234"  # 16 bytes

def criptografar(mensagem):
    cipher = AES.new(chave_simetrica, AES.MODE_EAX)
    nonce = cipher.nonce
    ciphertext, tag = cipher.encrypt_and_digest(mensagem.encode())

    return base64.b64encode(nonce + tag + ciphertext).decode()

# Configuração do cliente
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect(("127.0.0.1", 12345))  # Conecta ao servidor

mensagem_original = "Olá, servidor! Aqui é o cliente!"
mensagem_cifrada = criptografar(mensagem_original)

client.send(mensagem_cifrada.encode())  # Envia a mensagem criptografada
client.close()
```