Criptografia: explicações e exemplos (Simétrica, Assimétrica e Hash)

Este documento traz explicações resumidas sobre criptografia com chaves simétricas, assimétricas e funções hash, além de exemplos de código em Python e Java para cada caso. Use os exemplos como base para estudo e implementação.

1) Criptografia Simétrica (AES)

Descrição: Criptografia simétrica usa a mesma chave para cifrar e decifrar dados. Um algoritmo comum é AES (Advanced Encryption Standard). É rápido e adequado para grandes volumes de dados, mas o desafio é a troca segura da chave.

Código (Python):

```
# Exemplo Python (AES-GCM) usando a biblioteca 'cryptography'
from cryptography.hazmat.primitives.ciphers.aead import AESGCM
import os

# Gerar chave (256 bits)
key = AESGCM.generate_key(bit_length=256)
aesgcm = AESGCM(key)
nonce = os.urandom(12)  # 96-bit nonce
data = b"Mensagem secreta"
aad = b"header"  # dados adicionais autenticados (opcional)

ciphertext = aesgcm.encrypt(nonce, data, aad)
# Para descriptografar:
plaintext = aesgcm.decrypt(nonce, ciphertext, aad)
print(plaintext)
```

Código (Java):

```
// Exemplo Java (AES-GCM) usando javax.crypto
import javax.crypto.Cipher;
import javax.crypto.KeyGenerator;
import javax.crypto.SecretKey;
import javax.crypto.spec.GCMParameterSpec;
import javax.crypto.spec.SecretKeySpec;
import java.security.SecureRandom;

KeyGenerator keyGen = KeyGenerator.getInstance("AES");
keyGen.init(256);
SecretKey key = keyGen.generateKey();

byte[] iv = new byte[12];
new SecureRandom().nextBytes(iv);
GCMParameterSpec spec = new GCMParameterSpec(128, iv);

Cipher cipher = Cipher.getInstance("AES/GCM/NoPadding");
// cifrar
cipher.init(Cipher.ENCRYPT_MODE, key, spec);
byte[] ciphertext = cipher.doFinal("Mensagem secreta".getBytes());
// decifrar
cipher.init(Cipher.DECRYPT_MODE, key, spec);
byte[] plain = cipher.doFinal(ciphertext);
```

2) Criptografia Assimétrica (RSA)

Descrição: Criptografia assimétrica usa um par de chaves — pública e privada. Dados cifrados com a chave pública só podem ser decifrados com a chave privada. É útil para troca segura de chaves, assinaturas digitais e pequenos blocos de dados (não eficiente para grandes volumes).

Código (Python):

```
# Exemplo Python (RSA) usando 'cryptography' para encriptar/assinar
from cryptography.hazmat.primitives.asymmetric import rsa, padding
from cryptography.hazmat.primitives import hashes, serialization

# Gerar par de chaves
private_key = rsa.generate_private_key(public_exponent=65537, key_size=2048)
public_key = private_key.public_key()

message = b"Mensagem secreta"
ciphertext = public_key.encrypt(
    message,
    padding.OAEP(mgf=padding.MGF1(algorithm=hashes.SHA256()), algorithm=hashes.SHA256(), label=None)
)
# Decrypt
plaintext = private_key.decrypt(
    ciphertext,
    padding.OAEP(mgf=padding.MGF1(algorithm=hashes.SHA256()), algorithm=hashes.SHA256(), label=None)
)
print(plaintext)
```

Código (Java):

```
// Exemplo Java (RSA) usando java.security / javax.crypto
import java.security.KeyPairGenerator;
import java.security.KeyPair;
import javax.crypto.Cipher;

KeyPairGenerator kpg = KeyPairGenerator.getInstance("RSA");
kpg.initialize(2048);
KeyPair kp = kpg.generateKeyPair();

Cipher cipher = Cipher.getInstance("RSA/ECB/OAEPWithSHA-256AndMGF1Padding");
cipher.init(Cipher.ENCRYPT_MODE, kp.getPublic());
byte[] ciphertext = cipher.doFinal("Mensagem secreta".getBytes());
cipher.init(Cipher.DECRYPT_MODE, kp.getPrivate());
byte[] plain = cipher.doFinal(ciphertext);
```

3) Função Hash (SHA-256)

Descrição: Funções hash produzem um resumo fixo (digest) a partir de dados arbitrários. São determinísticas e, para boas funções, é impraticável gerar colisões ou reverter o digest. Usadas para verificar integridade e armazenar senhas (com salt e derivação).

Código (Python):

```
# Exemplo Python (SHA-256)
import hashlib

data = b"texto a ser hasheado"
digest = hashlib.sha256(data).hexdigest()
print(digest)
```

Código (Java):

```
// Exemplo Java (SHA-256)
import java.security.MessageDigest;

MessageDigest md = MessageDigest.getInstance("SHA-256");
byte[] digest = md.digest("texto a ser hasheado".getBytes("UTF-8"));
StringBuilder sb = new StringBuilder();
for (byte b : digest) {
    sb.append(String.format("%02x", b));
}
System.out.println(sb.toString());
```

Boas práticas

- Nunca armazene chaves em texto simples.
- Use bibliotecas bem testadas (p.ex. 'cryptography' em Python, provedor de segurança em Java).
- Para senhas, use salt + scrypt/argon2/PBKDF2.
- Combine criptografia assimétrica (para troca de chave) com simétrica (para dados).