Algoritmos de criptografia e hash

1. Conceito de Criptografia Simétrica

A criptografia de chave simétrica é um método de proteção de dados no qual a mesma chave é usada tanto para criptografar quanto para descriptografar uma informação. Isso significa que quem envia e quem recebe a mensagem precisam compartilhar previamente uma chave secreta — e mantê-la em sigilo absoluto, pois quem tiver acesso a essa chave poderá ler o conteúdo das mensagens.

Em termos simples:

Criptografia: transforma o texto original em um texto ilegível usando a chave secreta.

Descriptografia: aplica o processo inverso, usando a mesma chave para recuperar o texto original.

A criptografia simétrica é amplamente utilizada, especialmente porque é muito mais rápida que a criptografia assimétrica. Por isso, ela é comum em sistemas onde o desempenho é essencial, como:

Criptografia de discos e arquivos (BitLocker, VeraCrypt);

Conexões seguras (TLS/SSL, após o handshake inicial);

Protocolos de rede (Wi-Fi WPA2/WPA3).

2. O Algoritmo AES (Advanced Encryption Standard)

O AES, sigla para Advanced Encryption Standard, é atualmente o padrão mundial de criptografia simétrica. Ele foi desenvolvido para substituir o antigo DES (Data Encryption Standard), que se tornou inseguro devido ao aumento do poder computacional.

Em 2001, o NIST (National Institute of Standards and Technology) dos Estados Unidos escolheu o algoritmo Rijndael, criado pelos criptógrafos belgas Vincent Rijmen e Joan Daemen, como o novo padrão de criptografia simétrica — o AES.

3. Características do AES

Tamanhos de chave: 128, 192 ou 256 bits (quanto maior a chave, maior a segurança — mas também o tempo de processamento).

Tamanho do bloco: sempre 128 bits (16 bytes). Isso significa que o AES trabalha com blocos fixos de 128 bits por vez, aplicando uma sequência de operações matemáticas em cada bloco.

Número de rodadas: depende do tamanho da chave:

AES-128 → 10 rodadas

AES-192 → 12 rodadas

AES-256 → 14 rodadas

Cada rodada aplica substituições, permutações, misturas e operações XOR com partes da chave, de forma a embaralhar completamente os dados e tornar impossível descobrir o texto original sem a chave correta.

4. Estrutura do AES (visão simplificada)

Cada bloco de 128 bits é tratado como uma matriz de 4x4 bytes. O AES aplica várias transformações a essa matriz durante suas rodadas:

SubBytes: substitui cada byte por outro, segundo uma tabela chamada S-Box — é o passo que adiciona não linearidade ao algoritmo.

ShiftRows: desloca as linhas da matriz para a esquerda, misturando os dados.

MixColumns: combina os bytes de cada coluna usando operações matemáticas no corpo finito GF(2^8), aumentando a difusão dos bits.

AddRoundKey: aplica uma operação XOR entre o bloco e uma subchave derivada da chave principal.

Essas etapas garantem que pequenas alterações no texto original ou na chave gerem grandes diferenças no resultado final —

um princípio essencial da criptografia segura.

5. Modos de Operação

Como o AES trabalha com blocos fixos de 128 bits, para criptografar mensagens maiores é necessário usar modos de operação, que definem como os blocos se encadeiam. Os mais conhecidos são:

ECB (Electronic Codebook): criptografa cada bloco separadamente — simples, mas inseguro, pois blocos idênticos produzem saídas idênticas.

CBC (Cipher Block Chaining): cada bloco é combinado com o anterior antes da criptografia, usando um vetor de inicialização (IV). É mais seguro.

CFB e OFB: transformam o AES em um fluxo de bits cifrados, úteis para transmissões contínuas.

GCM (Galois/Counter Mode): modo moderno que combina criptografia e autenticação — amplamente usado em conexões HTTPS, VPNs e Wi-Fi.

6. Vantagens do AES

Alta segurança: até hoje, não há ataques práticos que quebrem o AES quando bem implementado.

Eficiência: pode ser executado rapidamente em hardware e software.

Flexibilidade: suporta diferentes tamanhos de chave e modos de operação.

Padrão global: adotado por governos, empresas e sistemas de comunicação em todo o mundo.

7. Exemplo de código em python

baixando dependências

Criação de exceções

In [ ]:
```python
"""Exceções customizadas para operações criptográficas"""


class CryptoBaseException(Exception):
    """Exceção base para erros de criptografia"""

    pass


class KeyGenerationError(CryptoBaseException):
    """Erro ao gerar chaves"""

    pass


class KeyLoadError(CryptoBaseException):
    """Erro ao carregar chaves"""

    pass


class EncryptionError(CryptoBaseException):
    """Erro ao cifrar dados"""

    pass


class DecryptionError(CryptoBaseException):
    """Erro ao decifrar dados"""

    pass
```

Algoritmo Dummy - cifra de César

In [ ]:
```python
import random

class CryptoDummy:
    def __init__(self) -> None:
        """
```

```
        Classe para fazer a criptografia de forma burra

        params:
            - textoCifrado: Armazena o texto após a cifragem
            - textoDecifrado: Armazena o texto decifrado
        """
        self.textoCifrado = None
        self.textoDecifrado = None

    @staticmethod
    def create_key(file_key: str):
        """
        Função para gerar a chave para o alogirmos dummy e salvar em um arquivo

        params:
            - file_key: Caminho do arquivo onde está a chave de decifragem
        """
        key = random.randint(1, 100)
        try:
            with open(file_key, "w") as f:
                f.write(str(key))
        except OSError as e:
            raise KeyGenerationError(
                f"Erro ao salvar chave no arquivo: {e}"
            ) from e

        except Exception as e:
            raise KeyGenerationError(
                f"Erro inesperado ao gerar chave{e}"
            ) from e

    @staticmethod
    def read_key(file_key: str) -> int:
        """
        Função para gerar a chave para o alogirmos dummy e salvar em um arquivo

        params:
            - file_key: Caminho do arquivo onde está a chave de decifragem
        """
        try:
            with open(file_key, "r") as f:
                return int(f.read())
        except FileNotFoundError as e:
            raise KeyLoadError(
                f"Arquivo de chave não encontrado: {file_key}"
            ) from e
        except (ValueError, TypeError) as e:
            raise KeyLoadError(
                f"Arquivo de chave inválido ou corrompido: {e}"
            ) from e
        except Exception as e:
            raise KeyLoadError(f"Erro inesperado ao carregar chave: {e}") from e

    def cipher(self, texto: str, file_key: str):
        """
        Gerar a cifra a partir do texto e da chave

        params:
            - texto (str): Mensagem que vai ser decifrada
            - file_key (str): Caminho do arquivo onde está a chave de decifragem
        """
        try:
            texto = texto.encode("utf-8")
            key = self.read_key(file_key)
            if key == 0:
                print("Não existe uma chave para criptografar")
                raise ValueError
            self.textoCifrado = bytes([(b + key) % 256 for b in texto])
        except KeyLoadError:
            raise
        except (ValueError, TypeError) as e:
            raise EncryptionError(f"Erro ao cifrar texto: {e}") from e
        except UnicodeEncodeError as e:
            raise EncryptionError(f"Erro ao codificar texto em UTF-8: {e}") from e
        except Exception as e:
            raise EncryptionError(f"Erro inesperado durante cifragem: {e}") from e

    def decipher(self, cifrado: bytes, file_key: str):
        """
        Decifrando o texto cifrado

        params:
            - cifrado (bytes): Texto cifrado pelo algoritimo AES
```

```
            - file_key (str): Caminho do arquivo onde está a chave de decifragem
        """
        try:
            key = self.read_key(file_key)
            if key == 0:
                print("Não existe uma chave para descifrar")
                return 0
            self.textoDecifrado = bytes([(b - key) % 256 for b in cifrado])
        except KeyLoadError:
            raise
        except (ValueError, TypeError) as e:
            raise DecryptionError(
                f"Erro ao decifrar texto (chave incorreta ou texto corrompido): {e}"
            ) from e
        except Exception as e:
            raise DecryptionError(f"Erro inesperado durante decifragem: {e}") from e

    def get_texto_cifrado(self) -> bytes:
        """Retorna o texto cifrado"""
        return self.textoCifrado

    def get_texto_decifrado(self) -> bytes:
        """Retorna o texto decifrado"""
        return self.textoDecifrado
```

Algoritmo AES de chave simétrica

In [ ]:

```
"""Algoritmo de criptografia com chave simétrica"""
import os
from cryptography.hazmat.primitives.ciphers.aead import AESGCM
from cryptography.exceptions import InvalidTag


class CryptoAes:
    def __init__(self):
        """
        Classe para fazer a criptografia com o algoritmo AES de chave simétrica

        params:
            - textoCifrado: Armazena o texto após a cifragem
            - textoDecifrado: Armazena o texto decifrado
        """
        self.textoCifrado = None
        self.textoDecifrado = None

    @staticmethod
    def create_key(file_key: str):
        """
        Função para gerar a chave para o alogirmos AES e salvar em um arquivo

        params:
            - file_key: Caminho do arquivo onde está a chave de decifragem
        """
        key = AESGCM.generate_key(bit_length=256)  # 256 bits
        try:
            with open(file_key, "wb") as f:
                f.write(key)
        except OSError as e:
            raise KeyGenerationError(
                f"Erro ao salvar chave no arquivo: {e}"
            ) from e

        except Exception as e:
            raise KeyGenerationError(
                f"Erro inesperado ao gerar chave{e}"
            ) from e

    @staticmethod
    def read_key(file_key: str):
        """
        Função para ler a chave usada para decifrar

        params:
            - file_key: Caminho do arquivo onde está a chave de decifragem
        """
        try:
            with open(file_key, "rb") as f:
                return f.read()
        except FileNotFoundError as e:
            raise KeyLoadError(
                f"Arquivo de chave não encontrado: {file_key}"
            ) from e
```

```
        except (ValueError, TypeError) as e:
            raise KeyLoadError(
                f"Arquivo de chave inválido ou corrompido: {e}"
            ) from e
        except Exception as e:
            raise KeyLoadError(f"Erro inesperado ao carregar chave: {e}") from e

    def cipher(self, texto: str, file_key: str):
        """
        Gerar a cifra a partir do texto e da chave

        params:
            - texto (str): Mensagem que vai ser decifrada
            - file_key (str): Caminho do arquivo onde está a chave de decifragem
        """
        try:
            # Convertento texto em bytes
            texto = texto.encode("utf-8")

            key = self.read_key(file_key)
            aesgcm = AESGCM(key)
            if not key:
                print("Erro ao ler a cave")
                raise ValueError

            # Nonce
            # Usado para gerar valores diferentes para para cada cifragem
            nonce = os.urandom(12)  # 96 bits

            # Cifrar
            textoCifrado = aesgcm.encrypt(nonce, texto, None)
            self.textoCifrado = nonce + textoCifrado
        except KeyLoadError:
            raise
        except (ValueError, TypeError) as e:
            raise EncryptionError(f"Erro ao cifrar texto: {e}") from e
        except UnicodeEncodeError as e:
            raise EncryptionError(f"Erro ao codificar texto em UTF-8: {e}") from e
        except Exception as e:
            raise EncryptionError(f"Erro inesperado durante cifragem: {e}") from e

    def decipher(self, cifrado: bytes, file_key: str):
        """
        Decifrando o texto cifrado

        params:
            - cifrado (bytes): Texto cifrado pelo algoritimo AES
            - file_key (str): Caminho do arquivo onde está a chave de decifragem
        """
        try:
            key = self.read_key(file_key)
            if not key:
                print("Não existe uma chave para decifrar")
                raise ValueError

            aesgcm = AESGCM(key)
            nonce = cifrado[:12]
            textoCifrado = cifrado[12:]
            self.textoDecifrado = aesgcm.decrypt(nonce, textoCifrado, None)

        except KeyLoadError:
            raise
        except (ValueError, TypeError) as e:
            raise DecryptionError(
                f"Erro ao decifrar texto (chave incorreta ou texto corrompido): {e}"
            ) from e
        except Exception as e:
            raise DecryptionError(f"Erro inesperado durante decifragem: {e}") from e

    def get_texto_cifrado(self) -> bytes:
        """
        Retorna o texto cifrado em formato string
        """
        return self.textoCifrado

    def get_texto_decifrado(self) -> bytes:
        """Retorna o texto decifrado em string UTF-8"""
        return self.textoDecifrado
```

1. Conceito de Criptografia Assimétrica

A criptografia de chave assimétrica, também chamada de criptografia de chave pública, é um tipo de criptografia que utiliza duas

chaves diferentes e matematicamente relacionadas:

Chave pública: pode ser compartilhada livremente com qualquer pessoa;

Chave privada: deve ser mantida em segredo absoluto.

Essas chaves são geradas em par, e o funcionamento baseia-se em uma relação matemática unidirecional — ou seja, o que é criptografado com uma das chaves só pode ser descriptografado com a outra.

Isso permite dois usos principais:

Confidencialidade: se Alice criptografa uma mensagem com a chave pública de Bob, apenas Bob (com sua chave privada) pode ler.

Autenticidade / Assinatura digital: se Alice assina um documento com sua chave privada, qualquer pessoa pode verificar a autenticidade usando sua chave pública.

Essa abordagem elimina o problema da troca de chaves secretas (como ocorre na criptografia simétrica), mas o custo é o desempenho — os algoritmos assimétricos são muito mais lentos, por isso geralmente são usados para proteger chaves simétricas, e não dados extensos diretamente.

2. O Algoritmo RSA

O RSA (iniciais de seus criadores: Rivest, Shamir e Adleman) é o algoritmo de chave pública mais famoso e amplamente utilizado no mundo. Ele foi proposto em 1977 e baseia sua segurança na dificuldade de fatorar números primos grandes — um problema matemático considerado computacionalmente inviável para chaves grandes o suficiente.

O RSA é usado em:

Assinaturas digitais e certificados (HTTPS, SSL/TLS);

Troca segura de chaves em conexões criptografadas;

Tokens de autenticação e carteiras digitais.

3. Conceitos do RSA

Algoritmo RSA é um algoritmo de criptografia criado por Rivest, Shamir e Adleman. A segurança desse algoritmo se baseia na dificuldade de fatoração de números primos grandes, pois para um atacante saber a chave privada D, ele teria que saber quais foram os primos P e Q que foram utilizados para achar o N, para assim calcular o Phi(N). Como o RSA utiliza números de 2048 bits, significa então 2^{2048} possibilidades, ou seja, 10^{617} sendo extremamente maior do que a quantidade de átomos no universo observável, que estima-se ser na ordem de grandeza de 10^{81}. Fazer a multiplicação de dois números primos de 2048 bits é facil (computacionalmente falando), porém para achar os números primos originários de um N de 4096 bits é preciso fatorar, algo que é computacionalmente inviável, levando milhares de anos para achar os dois números primos que foram multiplicados para dar o N. Sendo assim um algoritmo seguro para comunicação na internet.

As aplicações deste algoritmo vão desde o Login que você faz no Facebook, até as chaves que são utilizadas por um servidor SSH.

Exemplo de execução do algoritmo

1º Passo:

Para gerar a chva pública precisa primeiramente gerar dois múmeros que vamos chamar de P e Q. P e Q precisam ser primos e muito grandes. Contudo, para esse exemplo vou usar número s pequenos para facilitar os cálculos.

P = 17 e Q = 41.

2º Passo:

Agora calcularemos o N, sendo a multiplicação de P e Q.

N = P * Q = 17 * 41 = 697

N = 697

3º Passo:

Utilizaremos agora a função totiente de Euller, também chamada de phi, no N. Como P e Q são primos, phi é P -1 * Q-1.

Phi(N) = Phi(P) * Phi(Q)

Phi(N) = (P-1) * (Q-1)

Phi(N) = (17–1) * (41–1) = 640

Phi(N) = 640.

4º Passo:

Agora teremos que achar um outro número Aleatório E, que tem que satisfazer as condições: ser maior que 1 e menor que Phi(N), e também ser primo entre Phi(N).

1 < E < Phi(N) => 1 < E < 640

mdc(640, E) == 1

E = 13

1 < 13 < 640 e mdc(640, 13) == 1 (Condições atendidas)

5º Passo:

A chave pública é composta pelo N e o E, 697 e 13 respectivamente. Agora criptografar o nosso texto que será enviado. Transformaremos os caracteres em ASCII, podendo assim fazer operações aritméticas com os valores das letras.

Mensagem = 'oi'

ASCII('o') == 111

ASCII('i') == 105

6º Passo:

Para criptografar devemos seguir o seguinte algoritmo: Para cada caracter, elevaremos seu valor em ASCII por E e faremos a operação modular com N.

Valor cifrado de 'o' == 111 ^ 13 = 388328016259855395064461231

388328016259855395064461231 mod 697 = 110

Valor cifrado de 'o' == 110

Valor cifrado de 'i' == 105 ^ 13 = 188564914232323560791015625

188564914232323560791015625 mod 697 = 318

Valor cifrado de 'i' == 318

Mensagem cifrada = [110, 318] é enviada para o servidor.

7º Passo:

Agora devemos achar a chave privada que será usada para descriptografar a Mensagem cifrada enviada pelo cliente. A chave privada é chamada de D e é encontrada seguindo o algoritmo:

D * E mod Phi(N) == 1

D * 13 mod 640 == 1

D = 197

8º Passo:

Usaremos o D para descriptografar a Mensagem cifrada, para cada número da mensagem, iremos elevalo por D e fazer a operação modular por N.

Mensagem cifrada = [110, 318]

110 ^ 197 mod (N) == 111

317 ^ 197 mod (N) == 105

111 == ASCII('o')

105 == ASCII('i')

4. Exemplo de código em python

In [ ]:
```
from cryptography.hazmat.primitives.asymmetric import rsa, padding
```

```python
from cryptography.hazmat.primitives import serialization, hashes


class CryptoRSA:
    def __init__(self):
        """
        Classe para fazer a criptografia com o algoritmo AES de chave simétrica

        params:
            - textoCifrado: Armazena o texto após a cifragem
            - textoDecifrado: Armazena o texto descifrado
        """
        self.textoCifrado = None
        self.textoDecifrado = None

    @staticmethod
    def crete_key(file_private_key: str, file_public_key: str):
        """
        Função para gerar as chaves privada e pública e salvar em um arquivo

        params:
            - file_private_key: Caminho do arquivo onde está a chave privada
            - file_public_key: Caminho do arquivo onde está a chave pública

        raises:
            - KeyGenerationError: Se houver erro ao gerar ou salvar as chaves
        """
        try:
            # Gerando a chave privada
            private_key = rsa.generate_private_key(public_exponent=65537, key_size=2048)

            with open(file_private_key, "wb") as f:
                f.write(
                    private_key.private_bytes(
                        encoding=serialization.Encoding.PEM,
                        format=serialization.PrivateFormat.PKCS8,
                        encryption_algorithm=serialization.NoEncryption(),
                    )
                )
        except OSError as e:
            raise KeyGenerationError(
                f"Erro ao salvar chave particular no arquivo: {e}"
            ) from e

        except Exception as e:
            raise KeyGenerationError(
                f"Erro inesperado ao gerar chave particular: {e}"
            ) from e

        try:
            public_key = private_key.public_key()
            with open(file_public_key, "wb") as f:
                f.write(
                    public_key.public_bytes(
                        encoding=serialization.Encoding.PEM,
                        format=serialization.PublicFormat.SubjectPublicKeyInfo,
                    )
                )
        except OSError as e:
            raise KeyGenerationError(
                f"Erro ao salvar chave particular no arquivo: {e}"
            ) from e

        except Exception as e:
            raise KeyGenerationError(
                f"Erro inesperado ao gerar chave particular: {e}"
            ) from e

    @staticmethod
    def read_private_key(file_private_key: str):
        """
        Função para ler a chave privada

        params:
            - file_private_key: Caminho do arquivo onde está a chave privada

        raises:
            - KeyLoadError: Se houver erro ao carregar a chave
        """
        try:
            with open(file_private_key, "rb") as f:
                priv_pem = f.read()
            return serialization.load_pem_private_key(priv_pem, password=None)
```

```
        except FileNotFoundError as e:
            raise KeyLoadError(
                f"Arquivo de chave privada não encontrado: {file_private_key}"
            ) from e
        except (ValueError, TypeError) as e:
            raise KeyLoadError(
                f"Arquivo de chave privada inválido ou corrompido: {e}"
            ) from e
        except Exception as e:
            raise KeyLoadError(f"Erro inesperado ao carregar chave privada: {e}") from e

    @staticmethod
    def read_public_key(file_public_key: str):
        """
        Função para ler a chave pública

        params:
            - file_public_key: Caminho do arquivo onde está a chave pública

        raises:
            - KeyLoadError: Se houver erro ao carregar a chave
        """
        try:
            with open(file_public_key, "rb") as f:
                pub_pem = f.read()
                return serialization.load_pem_public_key(pub_pem)

        except FileNotFoundError as e:
            raise KeyLoadError(
                f"Arquivo de chave pública não encontrado: {file_public_key}"
            ) from e
        except ValueError as e:
            raise KeyLoadError(
                f"Arquivo de chave pública inválido ou corrompido: {e}"
            ) from e
        except Exception as e:
            raise KeyLoadError(f"Erro inesperado ao carregar chave pública: {e}") from e

    def cipher(self, texto: str, file_public_key: str):
        """
        Gerar a cifra a partir do texto e da chave

        params:
            - texto (str): Mensagem que vai ser decifrada
            - file_public_key (str): Caminho do arquivo onde está a chave pública

        raises:
            - EncryptionError: Se houver erro ao cifrar
        """
        try:
            texto = texto.encode("utf-8")
            key = self.read_public_key(file_public_key)
            self.textoCifrado = key.encrypt(
                texto,
                padding.OAEP(
                    mgf=padding.MGF1(algorithm=hashes.SHA256()),
                    algorithm=hashes.SHA256(),
                    label=None,
                ),
            )

        except KeyLoadError:
            raise
        except (ValueError, TypeError) as e:
            raise EncryptionError(f"Erro ao cifrar texto: {e}") from e
        except UnicodeEncodeError as e:
            raise EncryptionError(f"Erro ao codificar texto em UTF-8: {e}") from e
        except Exception as e:
            raise EncryptionError(f"Erro inesperado durante cifragem: {e}") from e

    def decipher(self, cifrado: bytes, file_private_key: str):
        """
        Decifrando o texto cifrado

        params:
            - cifrado (bytes): Texto cifrado pelo algoritimo RSA
            - file_private_key (str): Caminho do arquivo onde está a chave privada

        raises:
            - DecryptionError: Se houver erro ao decifrar
        """
        if cifrado is None:
```

```
            raise DecryptionError("Nenhum texto cifrado fornecido")

        try:
            key = self.read_private_key(file_private_key)
            self.textoDecifrado = key.decrypt(
                cifrado,
                padding.OAEP(
                    mgf=padding.MGF1(algorithm=hashes.SHA256()),
                    algorithm=hashes.SHA256(),
                    label=None,
                ),
            )
        except KeyLoadError:
            raise
        except (ValueError, TypeError) as e:
            raise DecryptionError(
                f"Erro ao decifrar texto (chave incorreta ou texto corrompido): {e}"
            ) from e
        except Exception as e:
            raise DecryptionError(f"Erro inesperado durante decifragem: {e}") from e

    def get_texto_cifrado(self) -> bytes:
        """Retorna o texto cifrado"""
        return self.textoCifrado

    def get_texto_decifrado(self) -> bytes:
        """Retorna o texto decifrado"""
        return self.textoDecifrado
```

Definindo testes dos algoritmos de criptografia

In [ ]:

```
import base64

class CryptoTest:
    def __init__(self, texto: str):
        """
        Classe para realizar testes dos algoritmos de criptografia

        paramans:
            - texto (str): Texto para criptografar.
        """
        self.texto = texto

        print("------------------------------------------------------------")
        print(">>> Imprimindo mensagem original...")
        print("Mensagem (String):")
        print(self.texto)

        print("Mensagem (Hexadecimal):")
        print(self.texto.encode("UTF-8").hex())
        print(" ")

    def testar_dummy(self, file_key: str):
        try:
            dummy = CryptoDummy()
            print(">>>> Cifrando com o algoritmo Dummy...\n")
            dummy.create_key(file_key)
            dummy.cipher(self.texto, file_key)

            cifrado = base64.b64encode(dummy.get_texto_cifrado())
            print("Mensagem cifrada (Hexadecimal):")
            print(cifrado.hex())
            print("Mensagem cifrado (String):")
            print(cifrado.decode("utf-8"))
            print(" ")

            print(">>>> Decifrando com algoritmo Dummy...\n")
            dummy.decipher(dummy.get_texto_cifrado(), file_key)
            decifrado = dummy.get_texto_decifrado()
            print("Mensagem Decifrada (Hexadecimal):")
            print(decifrado.hex())
            print("Mensagem Decifrada (String):")
            print(decifrado.decode("utf-8"))
            print(" ")
        except KeyGenerationError as e:
            print(f"Erro ao gerar chaves: {e}")
        except KeyLoadError as e:
            print(f"Erro ao carregar chaves: {e}")
        except EncryptionError as e:
            print(f"Erro ao cifrar: {e}")
        except DecryptionError as e:
            print(f"Erro ao decifrar: {e}")
```

```
        except Exception as e:
            print(f"Erro inesperado: {e}")

    def testar_aes(self, file_key: str):
        try:
            aes = CryptoAes()
            print(">>>> Cifrando com o algoritmo AES...\n")
            aes.create_key(file_key)
            aes.cipher(self.texto, file_key)

            cifrado = base64.b64encode(aes.get_texto_cifrado())
            print("Mensagem cifrada (Hexadecimal):")
            print(cifrado.hex())
            print("Mensagem cifrado (String):")
            print(cifrado.decode("utf-8"))
            print(" ")

            print(">>>> Decifrando com algoritmo AES...\n")
            aes.decipher(aes.get_texto_cifrado(), file_key)
            decifrado = aes.get_texto_decifrado()
            print("Mensagem Decifrada (Hexadecimal):")
            print(decifrado.hex())
            print("Mensagem Decifrada (String):")
            print(decifrado.decode("utf-8"))
            print(" ")
        except KeyGenerationError as e:
            print(f"Erro ao gerar chaves: {e}")
        except KeyLoadError as e:
            print(f"Erro ao carregar chaves: {e}")
        except EncryptionError as e:
            print(f"Erro ao cifrar: {e}")
        except DecryptionError as e:
            print(f"Erro ao decifrar: {e}")
        except Exception as e:
            print(f"Erro inesperado: {e}")

    def testar_rsa(self, file_private_key: str, file_public_key: str):
        try:
            rsa = CryptoRSA()
            print(">>>> Cifrando com o algoritmo RSA...\n")
            rsa.crete_key(file_private_key, file_public_key)
            rsa.cipher(self.texto, file_public_key)

            cifrado = base64.b64encode(rsa.get_texto_cifrado())
            print("Mensagem cifrada (Hexadecimal):")
            print(cifrado.hex())
            print("Mensagem cifrado (String):")
            print(cifrado.decode("utf-8"))
            print(" ")

            print(">>>> Decifrando com algoritmo RSA...\n")
            rsa.decipher(rsa.get_texto_cifrado(), file_private_key)
            decifrado = rsa.get_texto_decifrado()
            print("Mensagem Decifrada (Hexadecimal):")
            print(decifrado.hex())
            print("Mensagem Decifrada (String):")
            print(decifrado.decode("utf-8"))
            print(" ")
        except KeyGenerationError as e:
            print(f"Erro ao gerar chaves: {e}")
        except KeyLoadError as e:
            print(f"Erro ao carregar chaves: {e}")
        except EncryptionError as e:
            print(f"Erro ao cifrar: {e}")
        except DecryptionError as e:
            print(f"Erro ao decifrar: {e}")
        except Exception as e:
            print(f"Erro inesperado: {e}")
```

Resultados da criptografia

In [ ]:

```
import os


def main():
    try:
        # Limpando o terminal sempre que rodar o código

        os.system("clear")
        # Classe para teste dos algoritmos de criptografia
        test = CryptoTest("Testando vários algoritmos de criptografia")
```

```
        print("-----------------------------------------------------------")
        print("Testando algoritmos de criptografia\n")

        # Testando Dummy
        chave_dummy = "dummy.txt"
        test.testar_dummy(chave_dummy)
        # Testando AES
        chave_aes = "aes.txt"
        test.testar_aes(chave_aes)
        # Testando RSA
        chave_publica_rsa = "rsa_publica.txt"
        chave_privada_rsa = "rsa_privada.txt"
        test.testar_rsa(chave_privada_rsa, chave_publica_rsa)
        print("\n   Todos os testes concluídos com sucesso!")

    except KeyboardInterrupt:
        print("\n\nExecução interrompida pelo usuário")
    except Exception as e:
        print(f"\nErro fatal durante execução: {e}")
        import traceback
        traceback.print_exc()


if __name__ == "__main__":
    main()
```

```
------------------------------------------------------------
>>> Imprimindo mensagem original...
Mensagem (String):
Testando vários algoritmos de criptografia
Mensagem (Hexadecimal):
54657374616e646f2076c3a172696f7320616c676f7269746d6f732064652063726970746f677261666961

------------------------------------------------------------
Testando algoritmos de criptografia

>>>> Cifrando com o algoritmo Dummy...

Mensagem cifrada (Hexadecimal):
6248324c6a486d47664963346a747535696f4748697a683568482b48696f474d6859654c4f4878394f48754b6759694d68332b4b65583642
65513d3d
Mensagem cifrado (String):
bH2LjHmGfIc4jtu5ioGHizh5hH+HioGMhYeLOHx9OHuKgYiMh3+KeX6BeQ==

>>>> Decifrando com algoritmo Dummy...

Mensagem Decifrada (Hexadecimal):
54657374616e646f2076c3a172696f7320616c676f7269746d6f732064652063726970746f677261666961
Mensagem Decifrada (String):
Testando vários algoritmos de criptografia

>>>> Cifrando com o algoritmo AES...

Mensagem cifrada (Hexadecimal):
6b75754d48416745744c345a5a533046516670456c76376b333530712f617a526373416a733471676233437a446a4e634432666b76535530
5633396657676f45375565427231766f625a6a464565746b6d4846646f464e492f6545704862553d
Mensagem cifrado (String):
kuuMHAgEtL4ZZS0FQfpElv7k350q/azRcsAjs4qgb3CzDjNcD2fkvSU0V39fWgoE7UeBr1vobZjFEetkmHFdoFNI/eEpHbU=

>>>> Decifrando com algoritmo AES...

Mensagem Decifrada (Hexadecimal):
54657374616e646f2076c3a172696f7320616c676f7269746d6f732064652063726970746f677261666961
Mensagem Decifrada (String):
Testando vários algoritmos de criptografia

>>>> Cifrando com o algoritmo RSA...

Mensagem cifrada (Hexadecimal):
6f783033506959466e764436716c584e476f332f4b6d7937482f3052364e4c6f78786e784d6559524c4269663865624d6759343972456a63
464e776e496a354b6238515152645a754e4d73764f7064596e64644f4f4e6839674e496d592b6c6f326c36384b31363143532b7970586855
6176796a5543322f594650323275685445557631 4c48444f447865434c4831447235554e30526a666d776a796f2f4b7137304a7a5575717a
4859484353336e6d35362f6d4272336d343573654f5442453073387069457254594a786a6c6d5165562b73502f4a433871593455624f7a33
41324e774244653879455159 4b65464130372f7230446f51556e6962422b6e545a45676f2b373838666f4c4e384c49726c4e734a6b6c6e4e
446d522b2f62753756484e35576e6e415853732b782f2f646868534f316961706d4a4364535a573954352f7941783869756958436e4f6d48
343433646b413d3d
Mensagem cifrado (String):
ox03PiYFnvD6qlXNGo3/Kmy7H/0R6NLoxxnxMeYRLBif8ebMgY49rEjcFNwnIj5Kb8QQRdZuNMsvOpdYnddOONh9gNImY+lo2l68K161CS+ypXhU
avyjUC2/YFP22uhTEWV1LHDODxeCLH1Dr5UN0Rjfmwjyo/Kq70JzUuqzHYHCS3nm56/mBr3m45seOTBE0s8piErTYJxjlmQeV+sP/JC8qY4UbOz3
A2NwBDe8yEQYKeFA07/r0DoQUnibB+nTZEgo+788foLN8LIrlNsJklnNDmR+/bu7VHN5WnnAXSs+x//dhhSO1iapmJCdSZW9T5/yAx8iuiXCnOmH
443dkA==

>>>> Decifrando com algoritmo RSA...

Mensagem Decifrada (Hexadecimal):
54657374616e646f2076c3a172696f7320616c676f7269746d6f732064652063726970746f677261666961
Mensagem Decifrada (String):
Testando vários algoritmos de criptografia


  Todos os testes concluídos com sucesso!
```

Funções Hash

Um algoritmo hash (ou função hash criptográfica) transforma uma entrada de tamanho arbitrário (mensagem, arquivo, senha) em uma saída de tamanho fixo (o digest). Ex.: sha256("olá") = <valor de 256 bits em hex>.

Propriedades desejáveis de um hash criptográfico:

- Determinístico: mesma entrada → mesmo hash.
- Rápido de calcular.
- Resistente à colisão: é difícil achar duas entradas diferentes que produzam o mesmo hash.
- Resistente à pré-imagem: dado um hash H, é difícil encontrar uma mensagem M tal que hash(M) = H.

- Resistente à segunda pré-imagem: dado M1, é difícil achar M2 ≠ M1 tal que hash(M1)=hash(M2).
- Efeito avalanche: pequena mudança na entrada muda muito o hash.

Para que serve um hash?

Integridade: detectar alterações em arquivos (checksums, assinaturas digitais).

Armazenamento de senhas: hash + salt (não salvar senha em texto claro). Para senhas, use funções projetadas para senhas (bcrypt, Argon2, PBKDF2 com parâmetros fortes).

Assinaturas digitais e estruturas de dados (Merkle trees).

Indexação e deduplicação (não-criptográficas às vezes).

Exemplo de código em Python

In [6]:
```python
import os
import hashlib
import hmac
from typing import Tuple

class PasswordHasher:
    """
    Gera e verifica hashes de senhas usando PBKDF2-HMAC-SHA256.
    Formato armazenado: <iterations>$<salt_hex>$<hash_hex>
    """

    def __init__(self, iterations: int = 200_000, salt_size: int = 16):
        """
        iterations: número de iterações PBKDF2 (aumente com o tempo)
        salt_size: tamanho do salt em bytes (16 é razoável)
        """
        if iterations < 1:
            raise ValueError("iterations deve ser >= 1")
        self.iterations = iterations
        self.salt_size = salt_size

    def _derive(self, password: bytes, salt: bytes, iterations: int) -> bytes:
        """Deriva o key (hash) usando PBKDF2-HMAC-SHA256."""
        return hashlib.pbkdf2_hmac('sha256', password, salt, iterations)

    def hash_password(self, password: str) -> str:
        """
        Gera um hash para a senha fornecida.
        Retorna string no formato: iterations$salt_hex$hash_hex
        """
        if not isinstance(password, str):
            raise TypeError("password deve ser uma string")
        salt = os.urandom(self.salt_size)
        pwd_bytes = password.encode('utf-8')
        derived = self._derive(pwd_bytes, salt, self.iterations)
        return f"{self.iterations}${salt.hex()}${derived.hex()}"

    def verify_password(self, password: str, stored: str) -> bool:
        """
        Verifica se 'password' corresponde ao hash armazenado 'stored'.
        'stored' deve ter o formato gerado por hash_password.
        """
        try:
            parts = stored.split('$')
            if len(parts) != 3:
                return False
            iterations_str, salt_hex, hash_hex = parts
            iterations = int(iterations_str)
            salt = bytes.fromhex(salt_hex)
            expected = bytes.fromhex(hash_hex)
        except (ValueError, TypeError):
            return False

        pwd_bytes = password.encode('utf-8')
        derived = self._derive(pwd_bytes, salt, iterations)
        # comparação segura para evitar timing attacks
        return hmac.compare_digest(derived, expected)

    @staticmethod
    def parse_stored(stored: str) -> Tuple[int, bytes, bytes]:
        """
```

```
        (opcional) Retorna (iterations, salt_bytes, hash_bytes) de uma string armazenada.
        """
        parts = stored.split('$')
        if len(parts) != 3:
            raise ValueError("Formato inválido. Esperado iterations$salt_hex$hash_hex")
        iterations = int(parts[0])
        salt = bytes.fromhex(parts[1])
        hash_bytes = bytes.fromhex(parts[2])
        return iterations, salt, hash_bytes


if __name__ == "__main__":
    ph = PasswordHasher()
    senha = "minhaSenhaSecreta123!"
    print("-------------------------------------------------------")
    print(">>>> Aplicando algoritmo Hash... ")
    print("Senha original:")
    print(senha)
    armazenado = ph.hash_password(senha)
    print("\nHash da senha armazenado:")
    print(armazenado)


    print("\n-------------------------------------------------------")
    print(">>>> Usando o hash para verificar se a senha está correta... ")
    print("\nTentativa de senha (certa):")
    nova_senha = senha
    print(f"Senha: {nova_senha}")
    print(f"Hash da tentativa com: {nova_senha}")
    print(ph.hash_password(nova_senha))
    print("Verifica:", ph.verify_password(nova_senha, armazenado))
    print(" ")
    print("Tentativa de senha (errada):")
    nova_senha = "outraSenha"
    print(f"Senha: {nova_senha}")
    print(f"Hash da tentativa com: {nova_senha}")
    print(ph.hash_password(nova_senha))
    print("Verifica:", ph.verify_password(nova_senha, armazenado))
```

```
-------------------------------------------------------
>>>> Aplicando algoritmo Hash...
Senha original:
minhaSenhaSecreta123!

Hash da senha armazenado:
200000$2d4738959e6f8eaf7d5c1993c19cbb08$d0c26f06bd91fe6550fa054597600426affd12c6499ec1e3e39d920a60b018f6

-------------------------------------------------------
>>>> Usando o hash para verificar se a senha está correta...

Tentativa de senha (certa):
Senha: minhaSenhaSecreta123!
Hash da tentativa com: minhaSenhaSecreta123!
200000$1b32f75aa445d79c8227e1e173859167$afa343cfa94a486ccaa9f761ce1d8a2be3d51820266d60a4ee22bbefcfbabee4
Verifica: True

Tentativa de senha (errada):
Senha: outraSenha
Hash da tentativa com: outraSenha
200000$e8425f3d22fe20d1401466de35151f14$6f23f671f1808fdb639a0fa68c7afa124ef7eb24ee21f6e63e9a8cf6f61753a8
Verifica: False
```

Link do github

Link para testar esse código de python: https://github.com/Guilhermeslsand/test_cipher