Questão 1

Correto Atingiu 1,00 de 1,00

```
filename = input("Entre com o nome do arquivo: ")
md5_hash = hashlib.md5()
with open(filename,"rb") as f:
    # Read and update hash in chunks of 4K
    for byte_block in iter(lambda: f.read(4096),b""):
        md5_hash.update(byte_block)
    print(md5_hash.hexdigest())
```

O código Python apresentado é utilizado para calcular o valor do hash MD5 de um arquivo. Sobre as funcionalidades encontradas no código, assinale a alternativa correta:

a) O código lê o arquivo em modo texto e calcula o hash diretamente.

b) O valor MD5 é calculado em uma única operação, sem a necessidade de leitura em blocos.

c) O código permite que o usuário insira o nome do arquivo cujo hash será calculado.

d) O hash MD5 gerado é impresso em formato binário

- Todas as opções estão corretas.
- c e d estão corretos
- apenas a opção c está correta ✓
- b e d estão corretos
- a e b estão corretos

A resposta correta é:
apenas a opção c está correta

Questão 2

Correto Atingiu 1,00 de 1,00

Qual é a finalidade da função `message_to_binary` no código?

```
!pip install Pillow

from PIL import Image

# Function to convert a string message to binary
def message_to_binary(message):
    return ''.join([format(ord(char), '08b') for char in message])


# Function to convert binary data to string
def binary_to_message(binary_data):
    binary_chars = [binary_data[i:i + 8] for i in range(0, len(binary_data), 8)]
    return ''.join([chr(int(binary_char, 2)) for binary_char in binary_chars])

# Encode a message into an image
def encode_message(image_path, message, output_image_path):
    # Open image
    image = Image.open(image_path)
    image = image.convert('RGB')  # Ensure it's in RGB mode
    pixels = image.load()

    # Convert message to binary
    binary_message = message_to_binary(message) + '1111111111111110'  # Special
termination sequence
    data_index = 0

    # Modify the pixels
    for row in range(image.size[1]):
        for col in range(image.size[0]):
            if data_index < len(binary_message):
                # Get pixel RGB values
                r, g, b = pixels[col, row]

                # Modify LSB of each channel
                r = (r & 254) | int(binary_message[data_index])
                data_index += 1

                if data_index < len(binary_message):
                    g = (g & 254) | int(binary_message[data_index])
                    data_index += 1

                if data_index < len(binary_message):
                    b = (b & 254) | int(binary_message[data_index])
                    data_index += 1

                # Update pixel value
```

```
            pixels[col, row] = (r, g, b)

    # Save the image
    image.save(output_image_path)
    print(f'Message encoded and saved to {output_image_path}')

# Decode a message from an image
def decode_message(image_path):
    # Open image
    image = Image.open(image_path)
    image = image.convert('RGB')  # Ensure it's in RGB mode
    pixels = image.load()

    binary_message = ''
    for row in range(image.size[1]):
        for col in range(image.size[0]):
            r, g, b = pixels[col, row]
            binary_message += str(r & 1)
            binary_message += str(g & 1)
            binary_message += str(b & 1)

    # Split into chunks of 8 bits and convert to characters
    hidden_message = binary_to_message(binary_message)

    # Extract message before the termination sequence
    termination_index = hidden_message.find('þ')
    if termination_index != -1:
        hidden_message = hidden_message[:termination_index]

    return hidden_message

# Example usage:
# Encode a message
encode_message('natal.png', 'Feliz Natal!', 'encoded_natal.png')

# Decode the message
decoded_message = decode_message('encoded_natal.png')
print('Mensagem decodificada:', decoded_message)
```

- Converter uma mensagem em sua representação binária. ✓
- Converter uma string de caracteres em um array de inteiros.
- Extrair a mensagem oculta de uma imagem.
- Codificar uma mensagem dentro dos pixels de uma imagem.
- Modificar os pixels de uma imagem para alterar sua aparência.

A resposta correta é:
Converter uma mensagem em sua representação binária.

Questão 3

Correto Atingiu 1,00 de 1,00

O que representa a sequência de terminação `'1111111111111110'` no processo de codificação?

```
!pip install Pillow

from PIL import Image

# Function to convert a string message to binary
def message_to_binary(message):
    return ''.join([format(ord(char), '08b') for char in message])


# Function to convert binary data to string
def binary_to_message(binary_data):
    binary_chars = [binary_data[i:i + 8] for i in range(0, len(binary_data), 8)]
    return ''.join([chr(int(binary_char, 2)) for binary_char in binary_chars])

# Encode a message into an image
def encode_message(image_path, message, output_image_path):
    # Open image
    image = Image.open(image_path)
    image = image.convert('RGB')  # Ensure it's in RGB mode
    pixels = image.load()

    # Convert message to binary
    binary_message = message_to_binary(message) + '1111111111111110'  # Special
termination sequence
    data_index = 0

    # Modify the pixels
    for row in range(image.size[1]):
        for col in range(image.size[0]):
            if data_index < len(binary_message):
                # Get pixel RGB values
                r, g, b = pixels[col, row]

                # Modify LSB of each channel
                r = (r & 254) | int(binary_message[data_index])
                data_index += 1

                if data_index < len(binary_message):
                    g = (g & 254) | int(binary_message[data_index])
                    data_index += 1

                if data_index < len(binary_message):
                    b = (b & 254) | int(binary_message[data_index])
                    data_index += 1
```

```
            # Update pixel value
            pixels[col, row] = (r, g, b)

    # Save the image
    image.save(output_image_path)
    print(f'Message encoded and saved to {output_image_path}')

# Decode a message from an image
def decode_message(image_path):
    # Open image
    image = Image.open(image_path)
    image = image.convert('RGB')  # Ensure it's in RGB mode
    pixels = image.load()

    binary_message = ''
    for row in range(image.size[1]):
        for col in range(image.size[0]):
            r, g, b = pixels[col, row]
            binary_message += str(r & 1)
            binary_message += str(g & 1)
            binary_message += str(b & 1)

    # Split into chunks of 8 bits and convert to characters
    hidden_message = binary_to_message(binary_message)

    # Extract message before the termination sequence
    termination_index = hidden_message.find('þ')
    if termination_index != -1:
        hidden_message = hidden_message[:termination_index]

    return hidden_message

# Example usage:
# Encode a message
encode_message('natal.png', 'Feliz Natal!', 'encoded_natal.png')

# Decode the message
decoded_message = decode_message('encoded_natal.png')
print('Mensagem decodificada:', decoded_message)
```

- Uma sequência que melhora a qualidade da imagem.
- Um código de erro em caso de falha na codificação.
- Um delimitador que marca o final da mensagem oculta.
- Um valor que indica o início da mensagem.
- Um delimitador que marca o final da mensagem oculta. ✓

A resposta correta é:
Um delimitador que marca o final da mensagem oculta.

Questão 4

Correto Atingiu 1,00 de 1,00

Qual é a principal finalidade do método `public_key.encrypt` no código apresentado?

```
!pip install cryptography

from cryptography.hazmat.backends import default_backend
from cryptography.hazmat.primitives.asymmetric import rsa
from cryptography.hazmat.primitives import serialization
from cryptography.hazmat.primitives.asymmetric import padding
from cryptography.hazmat.primitives import hashes

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

public_key = private_key.public_key()

# Serialize the public key to PEM format
pem_public = public_key.public_bytes(
    encoding=serialization.Encoding.PEM,
    format=serialization.PublicFormat.SubjectPublicKeyInfo
)

# Serialize the private key to PEM format
pem_private = private_key.private_bytes(
    encoding=serialization.Encoding.PEM,
    format=serialization.PrivateFormat.TraditionalOpenSSL,
    encryption_algorithm=serialization.NoEncryption()
)

print("Public Key:")
print(pem_public.decode('utf-8'))

print("\nPrivate Key:")
print(pem_private.decode('utf-8'))

# Encrypt a message using the public key
message = b'Feliz Natal!'
ciphertext = public_key.encrypt(
    message,
    padding.OAEP(
        mgf=padding.MGF1(algorithm=hashes.SHA256()),
        algorithm=hashes.SHA256(),
        label=None
    )
)
```

```
print("\nCiphertext:")
print(ciphertext)

# Decrypt the message using the private key
decrypted_message = private_key.decrypt(
    ciphertext,
    padding.OAEP(
        mgf=padding.MGF1(algorithm=hashes.SHA256()),
        algorithm=hashes.SHA256(),
        label=None
    )
)

print("\nDecrypted Message:")
print(decrypted_message.decode('utf-8'))
```

- Converter a chave pública em formato PEM.
- Gerar um par de chaves públicas e privadas.
- Criptografar uma mensagem utilizando a chave pública. ✓
- Serializar a chave privada em um formato seguro.
- Descriptografar uma mensagem utilizando a chave privada.

A resposta correta é:
Criptografar uma mensagem utilizando a chave pública.

◄ APS12 - Atividade Prática Supervisionada

Seguir para...

Projeto da Disciplina (20 pontos) ►

...