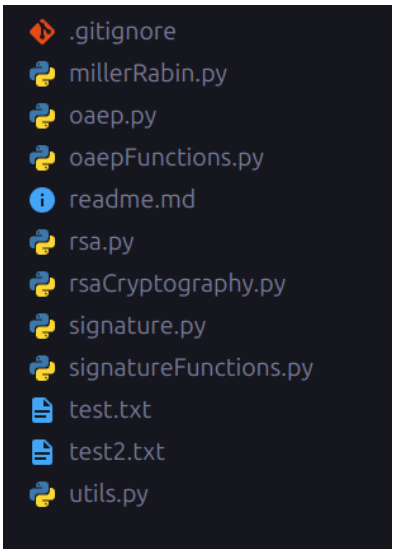
# Segurança Computacional - Trabalho 3

Gabriel Cruz Vaz Santos - 200049038

**Descrição do código - Estrutura de Arquivos**




**Descrição do código - geração de número primo e Rsa tradicional**

A função *isPrime* é responsável por realizar o Teste primalidade de Miller-Rabin

A função *randomOddValue* gera um inteiro de 1024 bits ímpar.

A função *generatePrime* faz uso das duas funções acima. Primeiro, ele gera um número de 1024 bits ímpar e continua esse processo até um deles passar no teste de Miller-Rabin.

A função *isMututallyPrime* recebe um número primo e outro gerado aleatoriamente e caso sejam primos entre si retorna verdadeiro.

A função *totientFunction* recebe dois números primos e retornará o produto deles e o totient desses dois números (p -1 ) * (q - 1).

A função *findTotientE* receberá o produto o totient da função *totientFunction* e irá gerar inteiros aleatórios até que um deles passem na função *isMutuallyPrime*.

A função *findPublicKeyAndTotient* recebe dois números primos gerados pela função *generatePrime* e chama as funções *totientFunction* e *findTotientE* e retorna um array de 3 elementos, sendo os 2 primeiros a chave pública no formato [p*q, e] e o terceiro elemento (p -1)*(q -1).

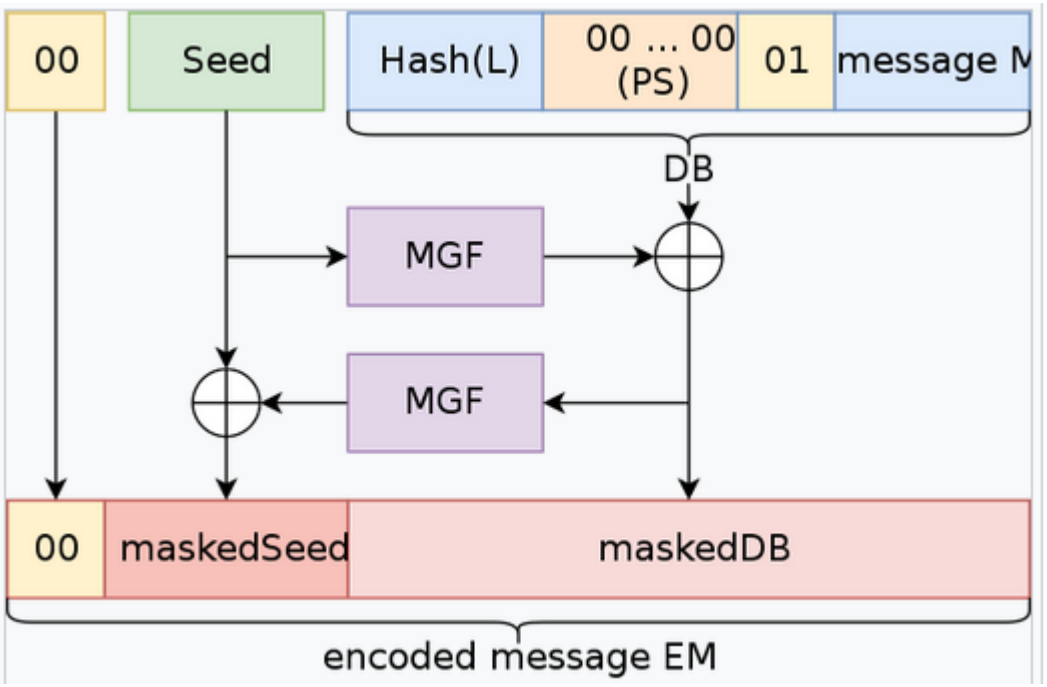
A função *findPrivateKey* recebe o (p -1)*(q-1) e “e” e calcula a chave privada “d” que é o inverso multiplicativo modular.

A função *cipherBytes* recebe o arquivo em bytes e a chave pública [p*q, e] para realizar a cifragem Rsa. Primeiro os bytes são convertidos para inteiro para posteriormente a cifragem.

A função *decryptdBytes* recebe o arquivo cifrado, o produto p*q, a chave privada “d” e o tamanho dos bytes  para realizar a decifrar usando algoritmo Rsa.

**Descrição do código - Rsa e Oaep**

Oaep é utilizado em conjunto da criptografia RSA pois a seed gerado aleatoriamente faz com que e as funções adicionadas aproximam o sistema criptográfico de um não determinístico. O esquema abaixo específica o funcionamento.




A função *calculateK* cálcula o tamanho da variável “k” com base no módulo de totient (p*q) em bits e será usado posteriormente.

A função *hashLabel* gera o hash com base no label e de tamanho “k” dado que será utilizado para gerar o dataBlock da nossa mensagem, representado no diagrama acima como Hash(L).

A função *generatePaddingString* é responsável por gerar o padding de zeros no dataBlock, representado no diagrama acima com PS, o padding possui tamanho de: $k - m\ L\ e\ n - 2 * h\ L\ e\ n - 2$ , onde mLen é o tamanho da mensagem e hLen o tamanho do hash gerado pela função *hashLabel*.

A função *mgf1* é responsável por gerar a máscara para determinada entrada. Essa máscara é aplicada utilizando da função *xor*. (ambas funções estão no arquivo Utils)

A função *oaepEncode* é responsável por chamar as funções descritas acima a fim de construir o dataBlock que será cifrado utilizando a cifra RSA, pela função *oaepCipher*, que possui o mesmo algoritmo que a função *cipherBytes,* no entanto, adaptada para os parâmetros que o *oaepEncode* retorna.

A função *oaepDecode* é responsável por fazer o caminho contrário da *oaepEncode* e por fi mgera ro dataBlock. Ainda dentro da função, o dataBlock é fragmentada e a função retorna a sessão com a mensagem

original que será decifrada pelo esquema Rsa através da função *oaepDecrypt*.

```
def oaepCipher(strFile, publicKey, label=b""):
  [n,e] = publicKey
  bytesMessage = strFile.encode()
  [encodedMessage, dataBlockMask, seedMask] = Oaep.oaepEncode(bytesMessage, n, label)
  print(f"Arquivo pós oaep: {encodedMessage}")
  convertedMessage = int.from_bytes(encodedMessage, byteorder='big')
  print(f"Arquivo original em inteiros: {convertedMessage}")
  encryptedFile = pow(convertedMessage, e, n)
  return [encryptedFile,dataBlockMask, seedMask, len(encodedMessage)]

def oaepDecrypt(cipherText, n, privateKey,dataBlockMask, seedMask,encodedMessageSize ,label=b""):
  decrypted = pow(cipherText, privateKey, n)
  print(f"Arquivo decifrado em inteiros: {decrypted}")
  message = Oaep.oaepDecode(decrypted, dataBlockMask, seedMask, encodedMessageSize ,label)
  return message.decode()
```

**Descrição do código - Signature**
Não foi possível implementar a assinatura em conjunto com a cifração, no entanto suas funções foram implementadas e testadas.

A função *hashMessage* recebe a string original e retorna o seu hash utilizando o sha-3.

A função *genSignature* recebe o hash retornado pela função acima e faz a cifração rsa utilizando-se da chave privada de quem está enviando a mensagem.

A função *getOriginalMessage* recebe a cifra do hash e, com a chave pública de quem enviou a mensagem, decifra com o algoritmo Rsa e dessa forma retorna o hash original.

```python
class Signature:
  def hashMessage(message):
    message = message.encode()
    hasher = hashlib.sha3_256()
    hasher.update(message)
    updatedHash = hasher.digest()
    return updatedHash

  def genSignature(encodedMessage, privateKey, n):
    convertedMessage = int.from_bytes(encodedMessage, byteorder='big')
    encryptedFile = pow(convertedMessage, privateKey, n)
    signature = encryptedFile
    return signature

  def getSignatureBytes(signature, lenBytes=256):
    signature = signature.to_bytes(lenBytes, byteorder='big')
    return signature


  def getOriginalMessage(encodedSignature, e, n, lenBytes):
    decrypted = pow(encodedSignature, e, n)
    signature = decrypted.to_bytes(lenBytes, byteorder='big')
    return signature
```