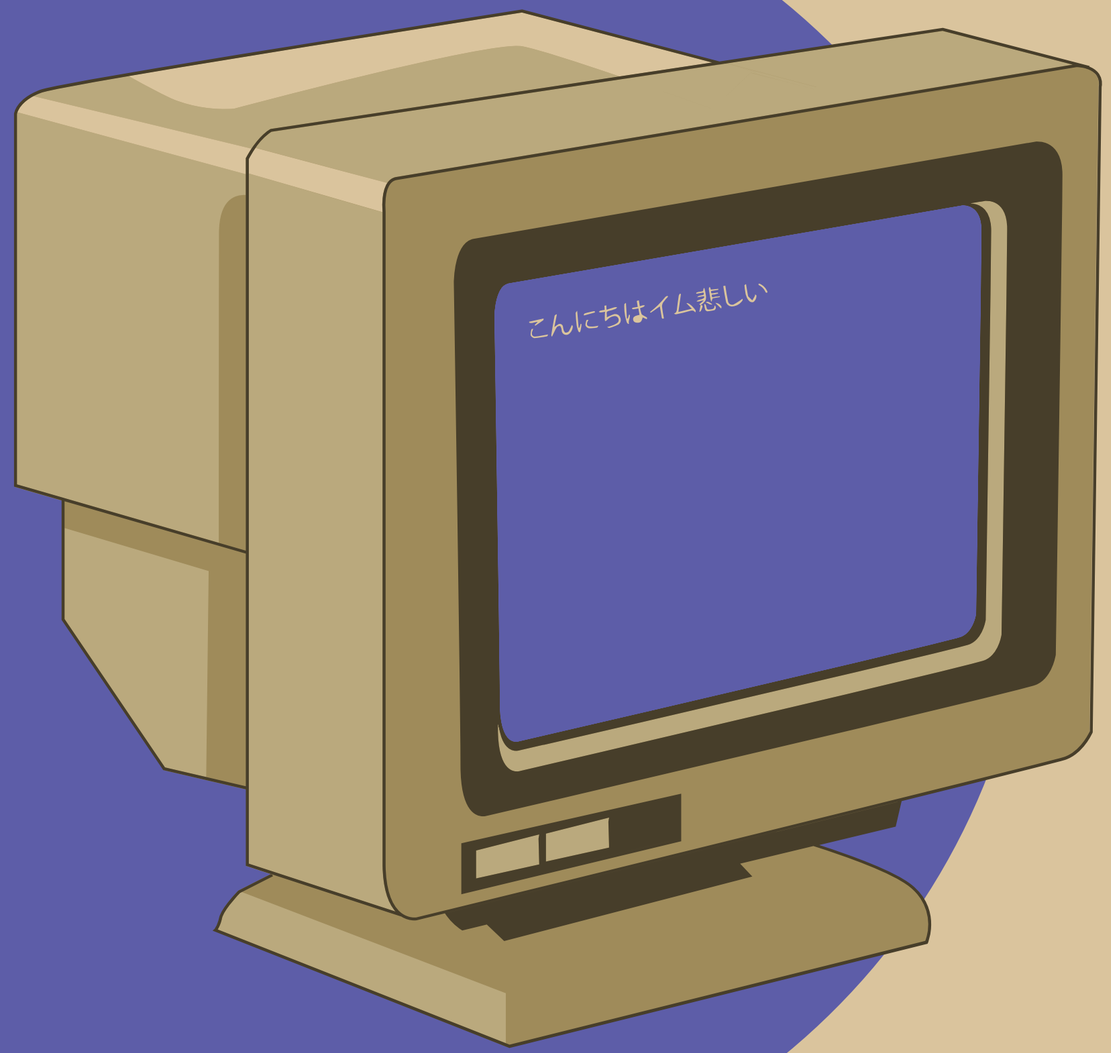
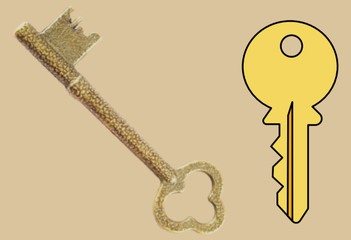


RSA OAEP ASSINATURAS DIGITAIS

GERAÇÃO DE CHAVES



Miller Rabin
n : número a ser testado
rounds : número de iterações (quanto mais iterações maior probabilidade de sucesso).
decompor(n): decompõe o número como n-1 = 2^e * k

```
def decompondo(n):
    """
    Decompõe n-1 como 2^e * m, onde n é ímpar.
    Args:
        n: número para decompor
    Returns:
        int: expoente
        int: número
    """
    e = 0
    m = n - 1
    while m % 2 == 0:
        m //= 2
        e += 1
    return e, m
```

Pseudo Miller Rabin:
decompor(n) return e, m
a = número aleatório entre 2 e n -2
x = (a ^ m) mod n
if x == 1 or x == n - 1: return True
else:
iterar (e - 1) vezes:
x = (x ^ 2) mod n
if x == n - 1: return True
return False

```python
def miller_rabin(n, rodadas=10):
    """
    Miller-Rabin primality test
    Args:
        n: numero para ser testado
        rodadas: quantidade de rodadas a serem feitas
    Returns:
        bool: True se é primo, False se não é primo
    """
    if n <= 1 or n == 4:
        return False
    if n <= 3:
        return True
    if n % 2 == 0:
        return False

    expo_k, m = decompondo(n)  #exponte K e m tal que n - 1 = 2^k * m
    for _ in range(rodadas):
        a = random.randrange(2, n - 2) # pega um numero aleatorio entre 2 e n-2
        """ x = a^m mod n """
        x = pow(a, m, n)
        if x == 1 or x == n-1: # se x for 1 ou n-1, então n é primo
            continue
        else:
            teste = False
            for _ in range(expo_k - 1): # para i de 0 até k-1
                """ x = x^2 mod n """
                x = pow(x, 2, n)
                if x == n-1:
                    teste = True
                    break
            if(teste):
                continue
            else:
                return False
    return True
```

Os primeiros 1000 números primos

A tabela a seguir lista os primeiros 1000 primos, com 20 colunas de primos consecutivos em cada uma das 50 linhas.[1]

Aspeto

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1–20	2	3	5	7	11	13	17	19	23	29	31	37	41	43	47	53	59	61	67	71
21–40	73	79	83	89	97	101	103	107	109	113	127	131	137	139	149	151	157	163	167	173
41–60	179	181	191	193	197	199	211	223	227	229	233	239	241	251	257	263	269	271	277	281
61–80	283	293	307	311	313	317	331	337	347	349	353	359	367	373	379	383	389	397	401	409
81–100	419	421	431	433	439	443	449	457	461	463	467	479	487	491	499	503	509	521	523	541
101–120	547	557	563	569	571	577	587	593	599	601	607	613	617	619	631	641	643	647	653	659

981–1000	7727	7741	7753	7757	7759	7789	7793	7817	7823	7829	7841	7853	7867	7873	7877	7879	7883	7901	7907	7919

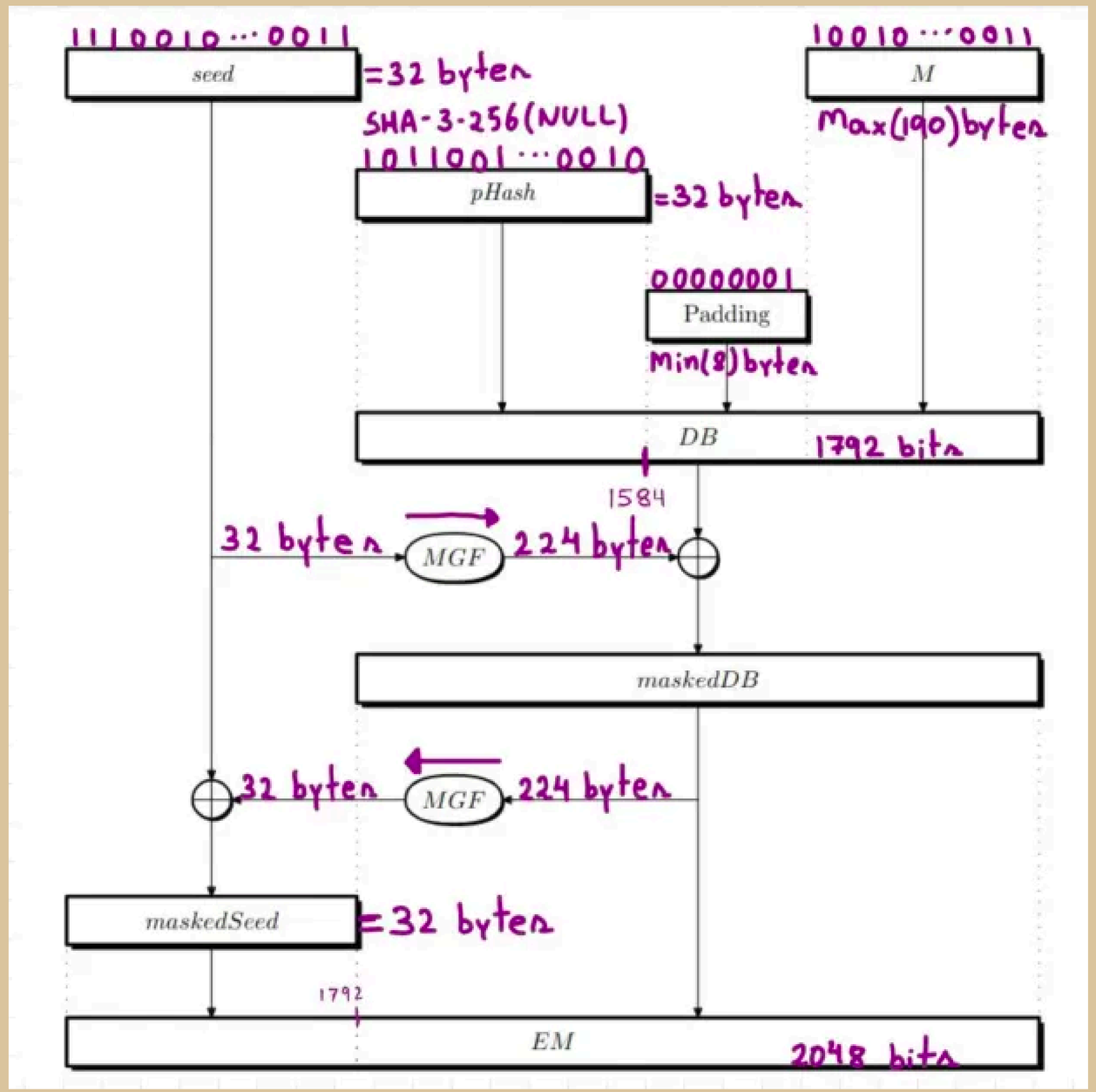
```
count = 0
for i in range(2, 7920):
    if(miller_rabin(i)):
        count += 1
print(count)
```

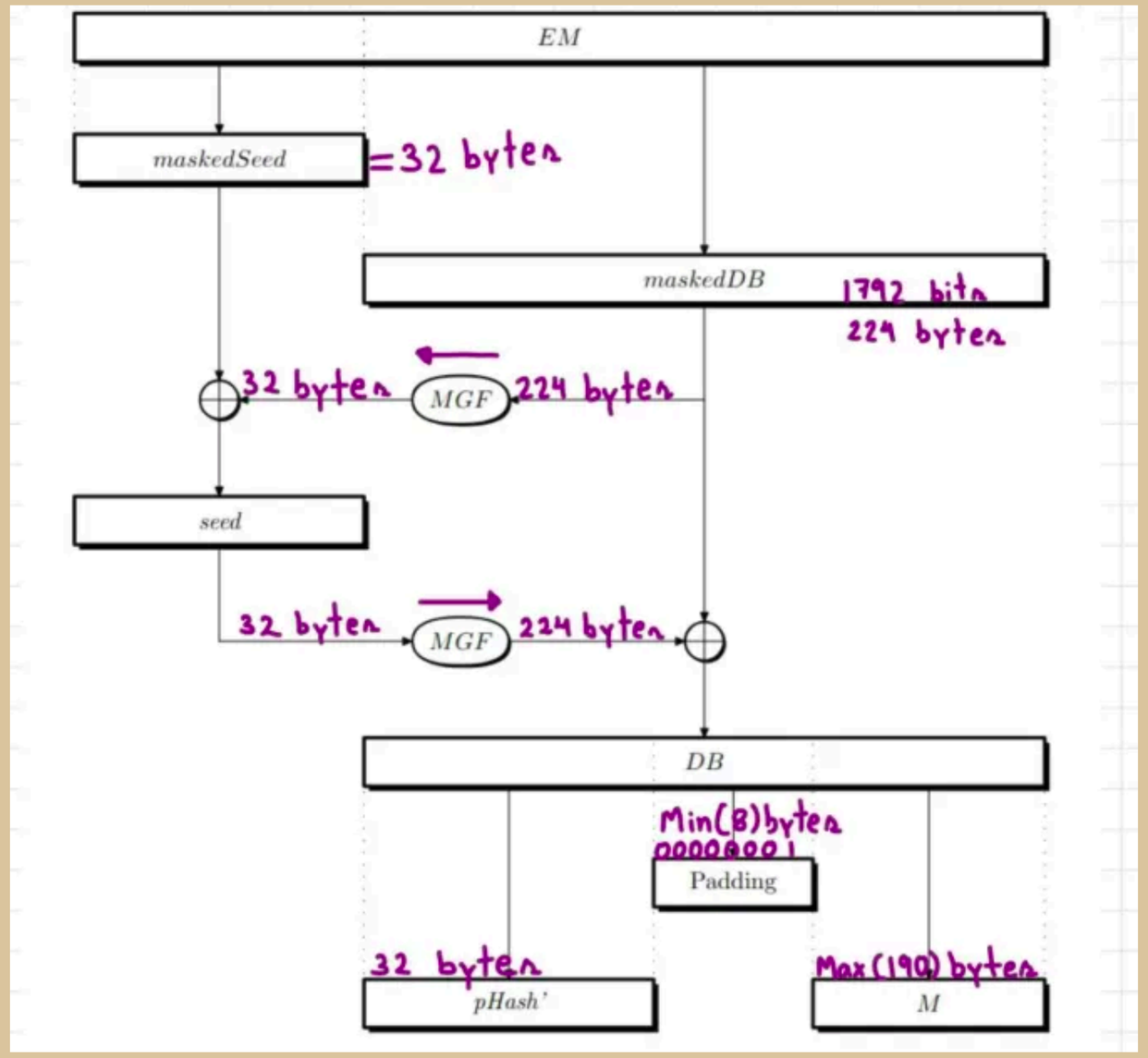
```
1000
```

```
def prime_numbers(): #descobre os números primos usando a função de Miller-R
    """
    Descobrindo os números primos
    Args: none
    Returns:
        list: lista com os números primos
    """
    n = 1 << 1024
    contador = 0
    par_de_numeros_primos = []
    while True:
        if miller_rabin(n):
            par_de_numeros_primos.append(n)
            contador += 1
        if contador == 2:
            break
        n += 1
    return par_de_numeros_primos
```

OAEP

- Camada adicional de segurança para o RSA
- No RSA para uma mesma entrada é retornado a mesma saída o que o torna vulnerável a certos tipos de ataque como o CCA.
- OAEP gera aleatorieda na saída, ou seja, para uma mesma entrada é gerado saídas diferentes.


1110010 ··· 0011
seed
=32 byten
SHA-3-256(NULL)
1011001 ··· 0010
pHash
=32 byten
10010 ··· 0011
M
Max(190)byten
00000001
Padding
Min(8)byten
DB
1792 bitn
1584
32 byten
MGF
224 byten
maskedDB
32 byten
MGF
224 byten
maskedSeed
=32 byten
1792
EM
2048 bitn


EM
maskedSeed
=32 byten
maskedDB
1792 bitn
224 byten
32 byten
MGF
224 byten
seed
32 byten
MGF
224 byten
DB
Min(8)byten
00000001
Padding
32 byten
pHash'
Max(190)byten
M

```python
def DB(pHash, mensagem): #faz a concatenação de pHash e mensagem
    """
    DB
    Obs:
        padding mínimo de 8 bits
    Args:
        pHash: pHash
        mensagem: mensagem
    Returns:
        int: DB com padding
    """
    pHash = shift_left(1536 , pHash)
    padding = shift_left(number_of_bits(mensagem), 1)
    pHash = pHash | padding
    return (pHash | mensagem)
```

```python
pHash = hashlib.sha3_256().digest()
pHash = int.from_bytes(pHash, byteorder='big')
mensagem = 15
print(bin(DB(pHash, mensagem)))
```

```
0b10100111111111111100011011111000101111110001111011010111011001100101000111000001010001110101011010100000011000011101011001100010111101011000000011111111010011011110010000111011010010
01111110101000001011011000000010100100101110000000111110000100001101001010000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000011111
```

```python
def enc_oaep(mensagem): #encriptação OAEP
    """
    Encapsulamento OAEP
    Args:
        mensagem: mensagem
    Returns:
        int: EM
    """
    seed = random.getrandbits(256).to_bytes(32, byteorder="big")                    # gera uma seed aleatória
    seed = int.from_bytes(seed, byteorder='big')                                    # converte a seed para int
    pHash = hashlib.sha3_256().digest()                                             # hash de NULL
    pHash = int.from_bytes(pHash, byteorder='big')                                  # converte o hash da seed para int
    maskedDB = DB(pHash, mensagem) ^ int.from_bytes(mgf1(seed, 1792, hashlib.sha3_256))   # faz a concatenação de pHash e mensagem
    maskedSeed = seed ^ int.from_bytes(mgf1(maskedDB, 32, hashlib.sha3_256))        # faz a máscara da seed
    maskedSeed = shift_left(1792, maskedSeed)                                       # desloca a seed para a esquerda em 1792 bits
    EM = maskedSeed | maskedDB                                                      # concatena a máscara da seed e a máscara da mensagem
    return EM
```

```python
def dec_oaep(c): #decriptaçao OAEP
    """
    Decriptando OAEP
    Args:
        c: EM
    Returns:
        int: m
    """
    bitMask = shift_left(1792, 1) - 1                                               # bitmask
    maskedDB = c & bitMask                                                          # retirando o maskedDB do texto cifrado
    maskedSeed = shift_right(1792, c)                                               # retirando o maskedSeed do texto cifrado
    seed = maskedSeed ^ int.from_bytes(mgf1(maskedDB, 32, hashlib.sha3_256), byteorder='big')   # seed = maskedSeed XOR mgf1(maskedDB)
    db = maskedDB ^ int.from_bytes(mgf1(seed, 1792, hashlib.sha3_256), byteorder='big')         # db = maskedDB XOR mgf1(seed)
    bitMask = shift_left(1536, 1) - 1                                               # bitmask
    padding_with_message = db & bitMask                                             # mensagem = db AND bitmask
    MSBit = msb(padding_with_message) - 1                                           # bit mais significativo
    MSBit = shift_left(MSBit, 1)                                                    # desloca o MSB para a esquerda
    m = m ^ MSBit                                                                   # m XOR MSBit
    return m
```

MULTIPLICATIVE INVERSE

Algoritmo de Euclides Extendido:

$B * x \cong 1 \bmod A$
Queremos descobrir x

A = 24
B = 5
Q = A // B
R = A % B
$T^1 = 0$
$T^2 = 1$
$T = T^1 - T^2 * Q$

```python
def multiplicative_inverse(a, b): # calcula o inverso multiplicativo usando o Algoritmo extendido de Euclides
    """
    Inverso multiplicativo
    Obs
        a > b
        a x t1 ≅ 1 mod b
    Args:
        a: numero
        b: numero
    Returns:
        int: inverso multiplicativo
    """
    if b > a:
        a, b = b, a
    old_a = a
    if gcd(a, b) != 1:
        return None
    q = a // b
    r = a % b
    t1 = 0
    t2 = 1
    t = t1 - t2 * q
    while b != 0:
        a, b = b, r
        if b == 0:
            t1 = t2
            if t1 < 0:
                t1 += old_a
            return t1
        q = a // b
        r = a % b
        t1, t2 = t2, t
        t = t1 - t2 * q
```

BASE 64

```python
def base64_encode(mensagem): #codificaçao usando a base64
    """
    Base64 Encode
    Args:
        mensagem: mensagem
    Returns:
        bytes: mensagem codificada
    """
    mensagem = mensagem.encode('utf-8')
    mensagem = base64.b64encode(mensagem)
    return mensagem

def base64_decode(mensagem_encriptada): #decodificaçao usando a base64
    """
    Base64 Decode
    Args:
        mensagem_encriptada: int mensagem encriptada
    Returns:
        str: mensagem decodificada
    """
    mensagem_encriptada = mensagem_encriptada.to_bytes((mensagem_encriptada.bit_length() + 7) // 8, byteorder='big')
    mensagem_encriptada = base64.b64decode(mensagem_encriptada).decode('utf-8')
    return mensagem_encriptada
```

RSA

Cifrar:
c = m ^ e mod n
Decriptar:
m = c ^ d mod n

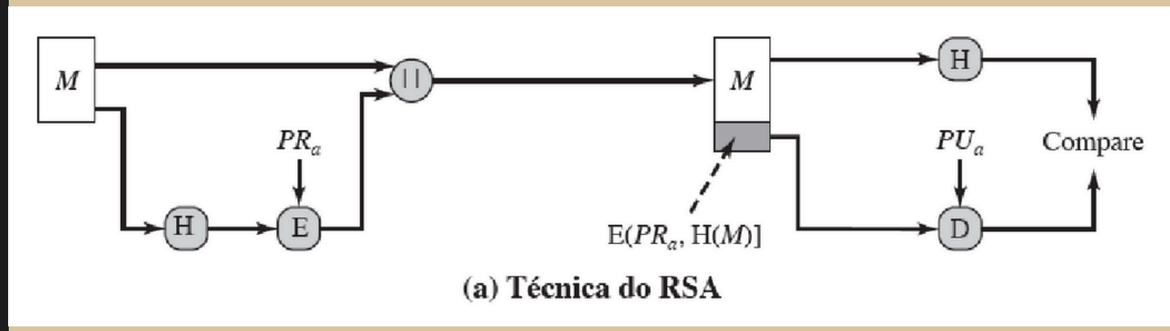
```python
def enc_rsa(n, e, m): #encriptaçao RSA
    """
    RSA Encryption
    Args:
        n: produto dos primos n
        e: chave public e
        m: mensagem
    Returns:
        int: c
        int: n
    """
    c = pow(m, e, n)
    return c, n

def dec_rsa(p, q, e, c): #decriptaçao RSA
    """
    RSA Decryption
    Args:
        p: primo p
        q: primo q
        e: chave publica e
        c: mensagem criptografada
    Returns:
        int: mensagem descriptografada m
    """
    n = p * q
    phi = (p - 1) * (q - 1)
    d = multiplicative_inverse(phi, e)
    m = pow(c, d, n)
    return m
```

ASSINATURA DIGITAL COM RSA

```python
def assinatura_com_rsa(message, key, n): #assinatura com RSA
    """
    RSA Signature
    Args:
        message: mensagem
        key: chave privada
        n: produto dos primos
    Returns:
        int: mensagem
        int: hash encriptado
    """
    message = base64_encode(message)
    message = int.from_bytes(message, byteorder='big')
    hash_message = hashlib.sha3_256(message.to_bytes(32, byteorder='big')).digest()
    hash_message = int.from_bytes(hash_message, byteorder='big')
    encrypted_hash = enc_rsa(n, key, hash_message)[0]
    return message, encrypted_hash

def verificar_assinatura_com_rsa(message, enc_hash, e, n):
    """
    RSA Signature Verification
    Args:
        message: mensagem
        enc_hash: hash encriptado
        e: chave publica e
        n: produto dos primos
    Returns:
        bool: True se a assinatura é valida, False se nao é valida
    """
    decrypted_hash = enc_rsa(n, e, enc_hash)
    hash_message = hashlib.sha3_256(message.to_bytes(32, byteorder='big')).digest()
    hash_message = int.from_bytes(hash_message, byteorder='big')
    return decrypted_hash[0] == hash_message
```



(a) Técnica do RSA

```python
# Teste de execuçao do programa

lista = prime_numbers()
p = lista[0]
q = lista[1]
n = p * q
phi = (p - 1) * (q - 1)
e = 65537
message = "Bluey Heeler"

print("Texto original:", message)
print("================================================================================")
message = base64_encode(message)
print("Message com BASE64:", message)
print("================================================================================")
message = enc_oaep(int.from_bytes(message, byteorder='big'))
print("Message com OAEP:", message)
print("================================================================================")
enc_message = enc_rsa(p * q, e, message)[0]
print("Message encriptada com RSA:", enc_message)
print("================================================================================")
dec_message = dec_rsa(p, q, e, enc_message)
print("Message descriptografada com RSA:", dec_message)
print("================================================================================")
dec_message = dec_oaep(dec_message)
print("Message descriptografada com OAEP:", dec_message)
print("================================================================================")
dec_message = base64_decode(dec_message)
print("Message descriptografada com BASE64:", dec_message)
print("================================================================================")
d = multiplicative_inverse(e, phi)          #Private key
message = "Bluey Heeler"
c = (assinatura_com_rsa(message, d, n))
print("Mensagem: ", c[0], "Assinatura: ", c[1])
print("================================================================================")
print(verificar_assinatura_com_rsa(c[0], c[1], e, n))
```

```
Texto original: Bluey Heeler
================================================================================
Message com BASE64: b'Qmx1ZXkgSGVlbGVy'
================================================================================
Message com OAEP: 19553010274892806882832785325579917646720067831508357346004911565366863156334886472634540319038896694799459230615445395705644836475195740847322647087121832086810731430531050728370045367837654918947357442318018465306246257983540969554346168424685313092542332324571199163070637200948023337514702750232694935451666310432805954280732155797546295932510714030740280737315
================================================================================
Message encriptada com RSA: 11097244115043698482266325820016857506750133430672691464382578582236163018692903972363972364922652619693721407706019442093341240481864879266374733433230174666434097110802303183363700020591509232939291662760544543089441313975413216941882232736051886699125782267327639477735956959145466306048660920059433516564491862144047748571382622421054375561276649712952203652113477284227302333751470275023269493545166631043280595428073215579754629593251071403074028073731534681727284227302333751470275023269493545166631043280595428073215579754629593251071403074028073731532350962447603812634540319038896694799459230615445395705644836475195740847322647087121832086810731434075128346211824367837654918947357442318018465306246257983540969554346168424685313092542332324571199346817272842273023337514702750232694935451666310432805954280732155797546295932510714030740280737315
================================================================================
```

```
==========================================================================
Message descriptografada com OAEP: 108235865815451426871397921985126684281
==========================================================================
Message descriptografada com BASE64: Bluey Heeler
==========================================================================
Mensagem:  108235865815451426871397921985126684281 Assinatura:  72869437866692104393864887608559093
912736529339480593131057878936512377639978300115477854921121687808970267976392688488330426889052886
253706677271966363731875637281733866418324877025988108748936891616797560395553020042119925352483864
886398114065115633302770320759089329350064966330662743762081179111598011007308180926640385473349453
==========================================================================
True
```

OBRIGADO !!!

https://github.com/BlueyHeeler/RSA_OAEP