


	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

			PROYECTO INTERCICLO		
CARRERA: Computación			ASIGNATURA: Computación Paralela		
NRO. PRÁCTICA:	1	TÍTULO PRÁCTICA: Aceleración de Cifrado y Descifrado de Datos con PyCUDA			
OBJETIVO ALCANZADO: <ul style="list-style-type: none"> Usar PyCUDA para acelerar operaciones criptográficas que son computacionalmente intensivas. 					
ACTIVIDADES DESARROLLADAS					
1. Codificar una versión básica de AES y RSA en Python para entender los algoritmos.					
<p style="text-align: center;">Algoritmo AES</p> <pre> import os import base64 class AesEncryptCpu(): """ Obtiene la caja de sustitución (S-box) para el algoritmo AES. """ @staticmethod def get_s_box(self): s_box_string = '63 7c 77 7b f2 6b 6f c5 30 01 67 2b fe d7 ab 76' \ 'ca 82 c9 7d fa 59 47 f0 ad d4 a2 af 9c a4 72 c0' \ 'b7 fd 93 26 36 3f f7 cc 34 a5 e5 f1 71 d8 31 15' \ '04 c7 23 c3 18 96 05 9a 07 12 80 e2 eb 27 b2 75' \ '09 83 2c 1a 1b 6e 5a a0 52 3b d6 b3 29 e3 2f 84' \ '53 d1 00 ed 20 fc b1 5b 6a cb be 39 4a 4c 58 cf' \ 'd0 ef aa fb 43 4d 33 85 45 f9 02 7f 50 3c 9f a8' \ '51 a3 40 8f 92 9d 38 f5 bc b6 da 21 10 ff f3 d2' \ 'cd 0c 13 ec 5f 97 44 17 c4 a7 7e 3d 64 5d 19 73' \ '60 81 4f dc 22 2a 90 88 46 ee b8 14 de 5e 0b db' \ 'e0 32 3a 0a 49 06 24 5c c2 d3 ac 62 91 95 e4 79' \ 'e7 c8 37 6d 8d d5 4e a9 6c 56 f4 ea 65 7a ae 08' \ 'ba 78 25 2e 1c a6 b4 c6 e8 dd 74 1f 4b bd 8b 8a' \ '70 3e b5 66 48 03 f6 0e 61 35 57 b9 86 c1 1d 9e' \ 'e1 f8 98 11 69 d9 8e 94 9b 1e 87 e9 ce 55 28 df' \ </pre>					


```
'8c a1 89 0d bf e6 42 68 41 99 2d 0f b0 54 bb 16'.replace(" ", "")
s_box = bytearray.fromhex(s_box_string)
return s_box

"""
    Obtiene la S-box inversa para el algoritmo AES.
"""
@staticmethod
def get_inv_s_box(self):
    inv_s_box_string = '52 09 6a d5 30 36 a5 38 bf 40 a3 9e 81 f3 d7 fb' \
        '7c e3 39 82 9b 2f ff 87 34 8e 43 44 c4 de e9 cb' \
        '54 7b 94 32 a6 c2 23 3d ee 4c 95 0b 42 fa c3 4e' \
        '08 2e a1 66 28 d9 24 b2 76 5b a2 49 6d 8b d1 25' \
        '72 f8 f6 64 86 68 98 16 d4 a4 5c cc 5d 65 b6 92' \
        '6c 70 48 50 fd ed b9 da 5e 15 46 57 a7 8d 9d 84' \
        '90 d8 ab 00 8c bc d3 0a f7 e4 58 05 b8 b3 45 06' \
        'd0 2c 1e 8f ca 3f 0f 02 c1 af bd 03 01 13 8a 6b' \
        '3a 91 11 41 4f 67 dc ea 97 f2 cf ce f0 b4 e6 73' \
        '96 ac 74 22 e7 ad 35 85 e2 f9 37 e8 1c 75 df 6e' \
        '47 f1 1a 71 1d 29 c5 89 6f b7 62 0e aa 18 be 1b' \
        'fc 56 3e 4b c6 d2 79 20 9a db c0 fe 78 cd 5a f4' \
        '1f dd a8 33 88 07 c7 31 b1 12 10 59 27 80 ec 5f' \
        '60 51 7f a9 19 b5 4a 0d 2d e5 7a 9f 93 c9 9c ef' \
        'a0 e0 3b 4d ae 2a f5 b0 c8 eb bb 3c 83 53 99 61' \
        '17 2b 04 7e ba 77 d6 26 e1 69 14 63 55 21 0c 7d'.replace(" ",
    )

    inv_s_box = bytearray.fromhex(inv_s_box_string)
    return inv_s_box

"""
    Genera una clave segura de la longitud especificada.
"""
@staticmethod
def generate_secure_key(length):
    return os.urandom(length)

"""
    Rellena los datos con bytes de relleno según el estándar de AES.
"""
@staticmethod
def pad(data):
    length = 16 - (len(data) % 16)
    return data + bytes([length] * length)
```

	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

```

"""
    Elimina el relleno de los datos según el estándar de AES.
"""
@staticmethod
def unpad(data):
    return data[:-data[-1]]

"""
    Realiza la operación XOR entre dos bloques de datos.
"""
@staticmethod
def xor_blocks(block1, block2):
    return bytes(a ^ b for a, b in zip(block1, block2))

"""
    Realiza la sustitución de bytes en una palabra utilizando la S-box.
"""
def sub_word(self, word: [int]) -> bytes:
    s_box = self.get_s_box(self)
    substituted_word = bytes(s_box[i] for i in word)
    return substituted_word

"""
    Genera el valor Rcon utilizado en la expansión de clave de AES.
"""
def rcon(self, i: int) -> bytes:
    # From Wikipedia
    rcon_lookup = bytearray.fromhex('01020408102040801b36')
    rcon_value = bytes([rcon_lookup[i-1], 0, 0, 0])
    return rcon_value

"""
    Realiza la operación XOR a nivel de bytes entre dos secuencias de bytes.
"""
def xor_bytes(self, a: bytes, b: bytes) -> bytes:
    return bytes([x ^ y for (x, y) in zip(a, b)])

"""
    Realiza una rotación circular de una palabra de 4 bytes a la izquierda.
"""
def rot_word(self, word: [int]) -> [int]:
    return word[1:] + word[:1]

```

```
"""
    Expansión de clave para AES que genera la clave de ronda.
"""
def key_expansion(self, key: bytes, nb: int = 4) -> [[[int]]]:

    nk = len(key) // 4

    key_bit_length = len(key) * 8

    if key_bit_length == 128:
        nr = 10
    elif key_bit_length == 192:
        nr = 12
    else: # 256-bit keys
        nr = 14

    w = self.state_from_bytes(self, key)

    for i in range(nk, nb * (nr + 1)):
        temp = w[i-1]
        if i % nk == 0:
            temp = self.xor_bytes(self, self.sub_word(self, self.rot_word(self,
temp)), self.rcon(self, i // nk))
        elif nk > 6 and i % nk == 4:
            temp = self.sub_word(self, temp)
        w.append(self.xor_bytes(self, w[i - nk], temp))

    return [w[i*4:(i+1)*4] for i in range(len(w) // 4)]

"""
    Operación de clave de ronda (AddRoundKey) en AES.
"""
def add_round_key(self, state: [[int]], key_schedule: [[[int]]], round: int):
    round_key = key_schedule[round]
    for r in range(len(state)):
        state[r] = [state[r][c] ^ round_key[r][c] for c in range(len(state[0]))]

"""
    Operación SubBytes en AES que aplica la S-box a cada byte del estado.
"""
def sub_bytes(self, state: [[int]]):
    s_box = self.get_s_box(self)
```

```
for r in range(len(state)):
    state[r] = [s_box[state[r][c]] for c in range(len(state[0]))]

"""
Operación ShiftRows en AES, desplaza las filas del estado.
"""
def shift_rows(self, state: [[int]]):
    # [00, 10, 20, 30]      [00, 10, 20, 30]
    # [01, 11, 21, 31] --> [11, 21, 31, 01]
    # [02, 12, 22, 32]      [22, 32, 02, 12]
    # [03, 13, 23, 33]      [33, 03, 13, 23]
    state[0][1], state[1][1], state[2][1], state[3][1] = state[1][1], state[2][1],
state[3][1], state[0][1]
    state[0][2], state[1][2], state[2][2], state[3][2] = state[2][2], state[3][2],
state[0][2], state[1][2]
    state[0][3], state[1][3], state[2][3], state[3][3] = state[3][3], state[0][3],
state[1][3], state[2][3]

"""
Operación de multiplicación por x en el campo de Galois.
"""
def xtime(self, a: int) -> int:
    if a & 0x80:
        return ((a << 1) ^ 0x1b) & 0xff
    return a << 1

"""
Operación MixColumns en AES, mezcla las columnas del estado.
"""
def mix_column(self, col: [int]):
    c_0 = col[0]
    all_xor = col[0] ^ col[1] ^ col[2] ^ col[3]
    col[0] ^= all_xor ^ self.xtime(self, col[0] ^ col[1])
    col[1] ^= all_xor ^ self.xtime(self, col[1] ^ col[2])
    col[2] ^= all_xor ^ self.xtime(self, col[2] ^ col[3])
    col[3] ^= all_xor ^ self.xtime(self, c_0 ^ col[3])

"""
Operación MixColumns en AES, mezcla las columnas del estado.
"""
def mix_columns(self, state: [[int]]):
    for r in state:
        self.mix_column(self, r)
```

```
"""
    Convierte los datos de bytes en un estado para AES.
"""
def state_from_bytes(self, data: bytes) -> [[int]]:
    state = [data[i*4:(i+1)*4] for i in range(len(data) // 4)]
    return state

"""
    Convierte un estado de AES en datos de bytes.
"""
def bytes_from_state(self, state: [[int]]) -> bytes:
    return bytes(state[0] + state[1] + state[2] + state[3])

"""
    Realiza el cifrado AES en modo CBC (Cipher Block Chaining).
"""
def aes_encrypt_cbc(self, text, key, iv):
    padded_text = self.pad(text.encode()) # Asegúrate de que el texto esté
rellenado
    blocks = [padded_text[i:i+16] for i in range(0, len(padded_text), 16)]

    encrypted_blocks = []
    previous_block = iv
    for block in blocks:
        block_to_encrypt = self.xor_blocks(block, previous_block)
        encrypted_block = self.aes_encryption(self, block_to_encrypt, key)
        encrypted_blocks.append(encrypted_block)
        previous_block = encrypted_block

    return b''.join(encrypted_blocks)

"""
    Realiza el descifrado AES en modo CBC (Cipher Block Chaining).
"""
def aes_decrypt_cbc(self, encrypted_text, key, iv):
    blocks = [encrypted_text[i:i+16] for i in range(0, len(encrypted_text), 16)]

    decrypted_blocks = []
    previous_block = iv
    for block in blocks:
        decrypted_block = self.aes_decryption(self, block, key)
        decrypted_text_block = self.xor_blocks(decrypted_block, previous_block)
```

```
        decrypted_blocks.append(decrypted_text_block)
        previous_block = block

    decrypted_text = b''.join(decrypted_blocks)
    return self.unpad(decrypted_text).decode()

"""
    Función principal para el cifrado de datos utilizando AES.
"""
def aes_encryption(self, data: bytes, key: bytes) -> bytes:

    key_bit_length = len(key) * 8

    if key_bit_length == 128:
        nr = 10
    elif key_bit_length == 192:
        nr = 12
    else: # 256-bit keys
        nr = 14

    state = self.state_from_bytes(self, data)

    key_schedule = self.key_expansion(self, key)

    self.add_round_key(self, state, key_schedule, round=0)

    for round in range(1, nr):
        self.sub_bytes(self, state)
        self.shift_rows(self, state)
        self.mix_columns(self, state)
        self.add_round_key(self, state, key_schedule, round)

    self.sub_bytes(self, state)
    self.shift_rows(self, state)
    self.add_round_key(self, state, key_schedule, round=nr)

    cipher = self.bytes_from_state(self, state)

    return cipher

"""
    DESENCRIPTACIÓN
"""
```

```
"""
    Operación inversa de ShiftRows en AES.
"""
def inv_shift_rows(self, state: [[int]]) -> [[int]]:
    # [00, 10, 20, 30]    [00, 10, 20, 30]
    # [01, 11, 21, 31] <-- [11, 21, 31, 01]
    # [02, 12, 22, 32]    [22, 32, 02, 12]
    # [03, 13, 23, 33]    [33, 03, 13, 23]
    state[1][1], state[2][1], state[3][1], state[0][1] = state[0][1], state[1][1],
state[2][1], state[3][1]
    state[2][2], state[3][2], state[0][2], state[1][2] = state[0][2], state[1][2],
state[2][2], state[3][2]
    state[3][3], state[0][3], state[1][3], state[2][3] = state[0][3], state[1][3],
state[2][3], state[3][3]
    return

"""
    Operación inversa de SubBytes en AES.
"""
def inv_sub_bytes(self, state: [[int]]) -> [[int]]:
    inv_s_box = self.get_inv_s_box(self)
    for r in range(len(state)):
        state[r] = [inv_s_box[state[r][c]] for c in range(len(state[0]))]

"""
    Realiza la operación de multiplicación por 0x0e en el campo de Galois.
"""
def xt看imes_0e(self, b):
    # 0x0e = 14 = b1110 = ((x * 2 + x) * 2 + x) * 2
    return self.xtime(self, self.xtime(self, self.xtime(self, b) ^ b) ^ b)

"""
    Realiza la operación de multiplicación por 0x0b en el campo de Galois.
"""
def xt看imes_0b(self, b):
    # 0x0b = 11 = b1011 = ((x*2)*2+x)*2+x
    return self.xtime(self, self.xtime(self, self.xtime(self, b)) ^ b) ^ b

"""
    Realiza la operación de multiplicación por 0x0d en el campo de Galois.
"""
def xt看imes_0d(self, b):
```



```
# 0x0d = 13 = b1101 = ((x*2+x)*2)*2+x
return self.xtime(self, self.xtime(self, self.xtime(self, b) ^ b)) ^ b

"""
Realiza la operación de multiplicación por 0x09 en el campo de Galois.
"""
def xtimes_09(self, b):
    # 0x09 = 9 = b1001 = ((x*2)*2)*2+x
    return self.xtime(self, self.xtime(self, self.xtime(self, b))) ^ b

"""
Realiza la operación inversa de MixColumn en AES.
"""
def inv_mix_column(self, col: [int]):
    c_0, c_1, c_2, c_3 = col[0], col[1], col[2], col[3]
    col[0] = self.xtimes_0e(self, c_0) ^ self.xtimes_0b(self, c_1) ^
self.xtimes_0d(self, c_2) ^ self.xtimes_09(self, c_3)
    col[1] = self.xtimes_09(self, c_0) ^ self.xtimes_0e(self, c_1) ^
self.xtimes_0b(self, c_2) ^ self.xtimes_0d(self, c_3)
    col[2] = self.xtimes_0d(self, c_0) ^ self.xtimes_09(self, c_1) ^
self.xtimes_0e(self, c_2) ^ self.xtimes_0b(self, c_3)
    col[3] = self.xtimes_0b(self, c_0) ^ self.xtimes_0d(self, c_1) ^
self.xtimes_09(self, c_2) ^ self.xtimes_0e(self, c_3)

"""
Realiza la operación inversa de MixColumns en AES.
"""
def inv_mix_columns(self, state: [[int]]) -> [[int]]:
    for r in state:
        self.inv_mix_column(self, r)

"""
Función principal para el descifrado de datos utilizando AES.
"""
def aes_decryption(self, cipher: bytes, key: bytes) -> bytes:

    key_byte_length = len(key)
    key_bit_length = key_byte_length * 8
    nk = key_byte_length // 4

    if key_bit_length == 128:
        nr = 10
    elif key_bit_length == 192:
```

```
        nr = 12
    else: # 256-bit keys
        nr = 14

    state = self.state_from_bytes(self, cipher)
    key_schedule = self.key_expansion(self, key)
    self.add_round_key(self, state, key_schedule, round=nr)

    for round in range(nr-1, 0, -1):
        self.inv_shift_rows(self, state)
        self.inv_sub_bytes(self, state)
        self.add_round_key(self, state, key_schedule, round)
        self.inv_mix_columns(self, state)

    self.inv_shift_rows(self, state)
    self.inv_sub_bytes(self, state)
    self.add_round_key(self, state, key_schedule, round=0)

    plain = self.bytes_from_state(self, state)
    return plain

"""
Realiza el cifrado de texto cifrado en AES en modo CBC.
"""
def encrypt(self, plaintext, key_size, public_key, iv):

    if public_key == None or public_key == '':
        iv = os.urandom(16) # Vector de Inicialización aleatorio
        key = self.generate_secure_key(key_size)
        ciphertext = self.aes_encrypt_cbc(self, text=plaintext, iv=iv, key=key)
        encoded_ciphertext = base64.b64encode(ciphertext).decode('utf-8')
        encoded_key = base64.b64encode(key).decode('utf-8')
        encoded_iv = base64.b64encode(iv).decode('utf-8')
        return {
            'ciphertext': encoded_ciphertext,
            'key': encoded_key,
            'iv': encoded_iv
        }
    else:
        ciphertext = self.aes_encrypt_cbc(self, text=plaintext, iv=iv,
key=public_key)
        encoded_ciphertext = base64.b64encode(ciphertext)
        return {
```

```
        'ciphertext': encoded_ciphertext,
        'key': 0,
        'iv': 0
    }

    """
    Realiza el descifrado de texto cifrado en AES en modo CBC.
    """

    def decrypt(self, ciphertext, public_key, iv):
        decrypted_plaintext = self.aes_decrypt_cbc(self, encrypted_text=ciphertext,
key=public_key, iv=iv)
        return decrypted_plaintext
```

Algoritmo RSA

```
import os
import re
import base64

class RsaEncryptCpu():

    """
        Obtiene la cantidad de factores de potencia de 2 y el número restante después de
        dividir por potencias de 2.
    """

    def get_power_2_factors(self, n: int) -> (int, int):
        r = 0
        d = n
        while n > 0 and d % 2 == 0:
            d = d // 2
            r += 1
        return r, d

    """
        Realiza el test de primalidad de Miller-Rabin para verificar si un número es
        probablemente primo.
    """

    def miller_rabin_prime_test(self, n: int, k: int) -> bool:

        # Factor powers of 2 from n - 1 s.t. n - 1 = 2^r * d
        r, d = self.get_power_2_factors(self, n-1)
```

```
for i in range(k):
    a = self.get_random_bits(self, n.bit_length())
    while a not in range(2, n-2+1):
        a = self.get_random_bits(self, n.bit_length())
    x = pow(a, d, n)
    if x == 1 or x == n - 1:
        continue
    n_1_found = False
    for j in range(r-1):
        x = pow(x, 2, n)
        if x == n - 1:
            n_1_found = True
            break
    if not n_1_found:
        return False
return True
```

```
"""
```

Genera un número aleatorio con la longitud de bits especificada.

```
"""
```

```
def get_random_bits(self, bit_length: int) -> int:
    return int.from_bytes(os.urandom((bit_length + 7) // 8), 'big')
```

```
"""
```

Genera un número primo aleatorio dentro de un rango específico.

```
"""
```

```
def generate_prime_number(self, bit_length: int) -> int:
```

```
    # prime needs to be in range  $[2^{(n-1)}, 2^{n-1}]$ 
```

```
    low = pow(2, bit_length - 1)
```

```
    high = pow(2, bit_length) - 1
```

```
    while True:
```

```
        # Generate odd prime candidate in range
```

```
        candidate_prime = self.get_random_bits(self, bit_length)
```

```
        while candidate_prime not in range(low, high+1) or not candidate_prime % 2:
            candidate_prime = self.get_random_bits(self, bit_length)
```

```
        # with k rounds, miller rabin test gives false positive with probability
```

```
(1/4)^k = 1/(2^2k)
```

```
k = 64
if self.miller_rabin_prime_test(self, candidate_prime, k):
    return candidate_prime

"""
Calcula el máximo común divisor extendido de dos números.
"""
def extended_gcd(self, a, b):
    if not b:
        return 1, 0

    u, v = self.extended_gcd(self, b, a % b)
    return v, u - v * (a // b)

"""
Calcula la clave privada para RSA.
"""
def calculate_private_key(self, e: int, p: int, q: int) -> int:
    u, _ = self.extended_gcd(self, e, (p-1)*(q-1))
    return u

"""
Realiza el cifrado RSA de manera bloqueada sobre los bytes del texto plano.
"""
def rsa_encrypt_blockwise(self, plaintext_bytes, e, n):
    block_size = (n.bit_length() + 7) // 8 - 1
    ciphertext_blocks = []

    for i in range(0, len(plaintext_bytes), block_size):
        block = plaintext_bytes[i:i + block_size]
        block_int = int.from_bytes(block, "big")
        encrypted_block_int = pow(block_int, e, n)
        ciphertext_blocks.append(encrypted_block_int.to_bytes((n.bit_length() + 7)
// 8, 'big'))

    return b''.join(ciphertext_blocks)

"""
Realiza el descifrado RSA de manera bloqueada sobre el texto cifrado.
```

```
"""
def rsa_decrypt_blockwise(self, ciphertext, d, n):
    block_size = (n.bit_length() + 7) // 8
    plaintext_blocks = []

    for i in range(0, len(ciphertext), block_size):
        block = ciphertext[i:i + block_size]
        block_int = int.from_bytes(block, "big")
        decrypted_block_int = pow(block_int, d, n)
        plaintext_blocks.append(decrypted_block_int.to_bytes(block_size, 'big'))

    return b''.join(plaintext_blocks).rstrip(b'\x00')

"""
Función principal para cifrar el texto plano utilizando RSA.
"""
def encrypt(self, plaintext, rsa_key_size, public_key):

    e = 65537
    if public_key == None or public_key == '':
        prime_number_bit_length = rsa_key_size // 2
        # Generate prime numbers p and q
        p = self.generate_prime_number(self, prime_number_bit_length)
        q = self.generate_prime_number(self, prime_number_bit_length)

        # Calculate public key
        n = p * q

        # Calculate private key
        d = self.calculate_private_key(self, e, p, q)
        ciphertext = self.rsa_encrypt_blockwise(self, plaintext.encode(), e, n)

        encoded_ciphertext = base64.b64encode(ciphertext).decode('utf-8')

        return {
            'ciphertext': encoded_ciphertext,
            'd': str(d),
            'n': str(n)
        }

    else:
        n = public_key
```

```

ciphertext = base64.b64decode(plaintext)
ciphertext = self.rsa_encrypt_blockwise(self, plaintext.encode(), e, n)
encoded_ciphertext = base64.b64encode(ciphertext).decode('utf-8')
return {
    'ciphertext': encoded_ciphertext,
    'd': 0,
    'n': 0
}

"""
Función para descifrar el texto cifrado utilizando RSA.
"""
def decrypt(self, ciphertext, d, n):
    recovered_plaintext_bytes = self.rsa_decrypt_blockwise(self, ciphertext, d, n)
    recovered_plaintext = recovered_plaintext_bytes.decode('utf-8')
    recovered_plaintext = re.sub(r'\x00', '', recovered_plaintext)
    return recovered_plaintext

```

2. Diseñar kernels PyCUDA para las operaciones de cifrado y descifrado en AES y RSA.

Algoritmo AES

```

from .cuda_context import CudaContext
from pycuda.compiler import SourceModule
import numpy as np
import pycuda.driver as drv
import os
import base64

class AesEncryptGpu():

    @staticmethod
    def get_s_box(self):
        s_box_string = '63 7c 77 7b f2 6b 6f c5 30 01 67 2b fe d7 ab 76' \
            'ca 82 c9 7d fa 59 47 f0 ad d4 a2 af 9c a4 72 c0' \
            'b7 fd 93 26 36 3f f7 cc 34 a5 e5 f1 71 d8 31 15' \
            '04 c7 23 c3 18 96 05 9a 07 12 80 e2 eb 27 b2 75' \
            '09 83 2c 1a 1b 6e 5a a0 52 3b d6 b3 29 e3 2f 84' \
            '53 d1 00 ed 20 fc b1 5b 6a cb be 39 4a 4c 58 cf' \
            'd0 ef aa fb 43 4d 33 85 45 f9 02 7f 50 3c 9f a8' \
            '51 a3 40 8f 92 9d 38 f5 bc b6 da 21 10 ff f3 d2' \
            'cd 0c 13 ec 5f 97 44 17 c4 a7 7e 3d 64 5d 19 73' \
            '60 81 4f dc 22 2a 90 88 46 ee b8 14 de 5e 0b db' \

```

```
'e0 32 3a 0a 49 06 24 5c c2 d3 ac 62 91 95 e4 79' \
'e7 c8 37 6d 8d d5 4e a9 6c 56 f4 ea 65 7a ae 08' \
'ba 78 25 2e 1c a6 b4 c6 e8 dd 74 1f 4b bd 8b 8a' \
'70 3e b5 66 48 03 f6 0e 61 35 57 b9 86 c1 1d 9e' \
'e1 f8 98 11 69 d9 8e 94 9b 1e 87 e9 ce 55 28 df' \
'8c a1 89 0d bf e6 42 68 41 99 2d 0f b0 54 bb 16'.replace(" ", "")
s_box = bytearray.fromhex(s_box_string)
return s_box
```

```
@staticmethod
```

```
def get_inv_s_box(self):
```

```
    inv_s_box_string = '52 09 6a d5 30 36 a5 38 bf 40 a3 9e 81 f3 d7 fb' \
        '7c e3 39 82 9b 2f ff 87 34 8e 43 44 c4 de e9 cb' \
        '54 7b 94 32 a6 c2 23 3d ee 4c 95 0b 42 fa c3 4e' \
        '08 2e a1 66 28 d9 24 b2 76 5b a2 49 6d 8b d1 25' \
        '72 f8 f6 64 86 68 98 16 d4 a4 5c cc 5d 65 b6 92' \
        '6c 70 48 50 fd ed b9 da 5e 15 46 57 a7 8d 9d 84' \
        '90 d8 ab 00 8c bc d3 0a f7 e4 58 05 b8 b3 45 06' \
        'd0 2c 1e 8f ca 3f 0f 02 c1 af bd 03 01 13 8a 6b' \
        '3a 91 11 41 4f 67 dc ea 97 f2 cf ce f0 b4 e6 73' \
        '96 ac 74 22 e7 ad 35 85 e2 f9 37 e8 1c 75 df 6e' \
        '47 f1 1a 71 1d 29 c5 89 6f b7 62 0e aa 18 be 1b' \
        'fc 56 3e 4b c6 d2 79 20 9a db c0 fe 78 cd 5a f4' \
        '1f dd a8 33 88 07 c7 31 b1 12 10 59 27 80 ec 5f' \
        '60 51 7f a9 19 b5 4a 0d 2d e5 7a 9f 93 c9 9c ef' \
        'a0 e0 3b 4d ae 2a f5 b0 c8 eb bb 3c 83 53 99 61' \
        '17 2b 04 7e ba 77 d6 26 e1 69 14 63 55 21 0c 7d'.replace(" ",
```

```
""")
```

```
    inv_s_box = bytearray.fromhex(inv_s_box_string)
    return inv_s_box
```

```
@staticmethod
```

```
def generate_secure_key(length):
```

```
    return os.urandom(length)
```

```
@staticmethod
```

```
def pad(data):
```

```
    length = 16 - (len(data) % 16)
    return data + bytes([length] * length)
```



```
@staticmethod
def unpad(data):
    return data[:-data[-1]]

@staticmethod
def xor_blocks(block1, block2):
    return bytes(a ^ b for a, b in zip(block1, block2))

def sub_word(self, word: [int]) -> bytes:
    s_box = self.get_s_box(self)
    substituted_word = bytes(s_box[i] for i in word)
    return substituted_word

def rcon(self, i: int) -> bytes:
    # From Wikipedia
    rcon_lookup = bytearray.fromhex('01020408102040801b36')
    rcon_value = bytes([rcon_lookup[i-1], 0, 0, 0])
    return rcon_value

def xor_bytes(self, a: bytes, b: bytes) -> bytes:
    return bytes([x ^ y for (x, y) in zip(a, b)])

def rot_word(self, word: [int]) -> [int]:
    return word[1:] + word[:1]

def key_expansion(self, key: bytes, nb: int = 4) -> [[[int]]]:

    nk = len(key) // 4

    key_bit_length = len(key) * 8

    if key_bit_length == 128:
        nr = 10
    elif key_bit_length == 192:
        nr = 12
    else: # 256-bit keys
        nr = 14
```

```
w = self.state_from_bytes(self, key)

for i in range(nk, nb * (nr + 1)):
    temp = w[i-1]
    if i % nk == 0:
        temp = self.xor_bytes(self, self.sub_word(self, self.rot_word(self,
temp)), self.rcon(self, i // nk))
    elif nk > 6 and i % nk == 4:
        temp = self.sub_word(self, temp)
    w.append(self.xor_bytes(self, w[i - nk], temp))

return [w[i*4:(i+1)*4] for i in range(len(w) // 4)]

# PARALELIZABLE
def shift_rows(self, state: [[int]]):
    # [00, 10, 20, 30]      [00, 10, 20, 30]
    # [01, 11, 21, 31] --> [11, 21, 31, 01]
    # [02, 12, 22, 32]      [22, 32, 02, 12]
    # [03, 13, 23, 33]      [33, 03, 13, 23]
    state[0][1], state[1][1], state[2][1], state[3][1] = state[1][1], state[2][1],
state[3][1], state[0][1]
    state[0][2], state[1][2], state[2][2], state[3][2] = state[2][2], state[3][2],
state[0][2], state[1][2]
    state[0][3], state[1][3], state[2][3], state[3][3] = state[3][3], state[0][3],
state[1][3], state[2][3]

def xtime(self, a: int) -> int:
    if a & 0x80:
        return ((a << 1) ^ 0x1b) & 0xff
    return a << 1

def mix_column(self, col: [int]):
    c_0 = col[0]
    all_xor = col[0] ^ col[1] ^ col[2] ^ col[3]
    col[0] ^= all_xor ^ self.xtime(self, col[0] ^ col[1])
    col[1] ^= all_xor ^ self.xtime(self, col[1] ^ col[2])
    col[2] ^= all_xor ^ self.xtime(self, col[2] ^ col[3])
    col[3] ^= all_xor ^ self.xtime(self, c_0 ^ col[3])

# PARALELIZABLE
```

```
def mix_columns(self, state: [[int]]):
    for r in state:
        self.mix_column(self, r)

def state_from_bytes(self, data: bytes) -> [[int]]:
    state = [data[i*4:(i+1)*4] for i in range(len(data) // 4)]
    return state

def bytes_from_state(self, state: [[int]]) -> bytes:
    return bytes(state[0] + state[1] + state[2] + state[3])

def aes_encrypt_cbc(self, text, key, iv):
    padded_text = self.pad(text.encode()) # Asegúrate de que el texto esté
rellenado
    blocks = [padded_text[i:i+16] for i in range(0, len(padded_text), 16)]

    encrypted_blocks = []
    previous_block = iv
    for block in blocks:
        block_to_encrypt = self.xor_blocks(block, previous_block)
        encrypted_block = self.aes_encryption(self, block_to_encrypt, key)
        encrypted_blocks.append(encrypted_block)
        previous_block = encrypted_block

    return b''.join(encrypted_blocks)

def aes_decrypt_cbc(self, encrypted_text, key, iv):
    blocks = [encrypted_text[i:i+16] for i in range(0, len(encrypted_text), 16)]

    decrypted_blocks = []
    previous_block = iv
    for block in blocks:
        decrypted_block = self.aes_decryption(self, block, key)
        decrypted_text_block = self.xor_blocks(decrypted_block, previous_block)
        decrypted_blocks.append(decrypted_text_block)
        previous_block = block

    decrypted_text = b''.join(decrypted_blocks)
    return self.unpad(decrypted_text).decode()
```

```
def aes_encryption(self, data: bytes, key: bytes) -> bytes:

    if CudaContext.initialized:
        CudaContext.pop_context()

    CudaContext.get_context()

    try:
        aes_kernels = """

            __global__ void subBytesKernel(unsigned char *state, const unsigned char
*s_box) {
                int idx = blockIdx.x * blockDim.x + threadIdx.x;
                if (idx < 16) { // Asegurando que solo se procesen 16 bytes (tamaño
del estado en AES)
                    state[idx] = s_box[state[idx]];
                }
            }

            __global__ void addRoundKeyKernel(unsigned char *state, unsigned char
*key_schedule, int round) {
                int idx = blockIdx.x * blockDim.x + threadIdx.x; // Índice lineal
para cada byte del estado

                if (idx < 16) { // Aseguramos que solo procesamos los 16 bytes del
estado
                    int round_key_idx = round * 16 + idx;
                    state[idx] ^= key_schedule[round_key_idx];
                }
            }

        """

        mod = SourceModule(aes_kernels)
        subBytesKernel = mod.get_function("subBytesKernel")
        addRoundKeyKernel = mod.get_function("addRoundKeyKernel")

        key_bit_length = len(key) * 8

        if key_bit_length == 128:
            nr = 10
        elif key_bit_length == 192:
```

```
        nr = 12
    else: # 256-bit keys
        nr = 14

    state = self.state_from_bytes(self, data)
    flat_state = [byte for row in state for byte in row]
    state_np = np.array(flat_state, dtype=np.uint8)

    key_schedule = self.key_expansion(self, key)
    flattened_key_schedule = [byte for round_key in key_schedule for row in
round_key for byte in row]
    key_schedule_np = np.array(flattened_key_schedule, dtype=np.uint8)

    addRoundKeyKernel(
        drv.InOut(state_np),
        drv.In(key_schedule_np),
        np.int32(0),
        block=(16, 1, 1),
        grid=(1, 1)
    )

    s_box_np = np.array(self.get_s_box(self), dtype=np.uint8)

    for round in range(1, nr):
        subBytesKernel(
            drv.InOut(state_np),
            drv.In(s_box_np),
            block=(16, 1, 1),
            grid=(1, 1)
        )
        drv.Context.synchronize()
        self.shift_rows(self, state_np.reshape(4, 4))
        self.mix_columns(self, state_np.reshape(4, 4))
        addRoundKeyKernel(
            drv.InOut(state_np),
            drv.In(key_schedule_np),
            np.int32(round),
            block=(16, 1, 1),
            grid=(1, 1)
        )

    subBytesKernel(
        drv.InOut(state_np),
```

```

        drv.In(s_box_np),
        block=(16, 1, 1),
        grid=(1, 1)
    )
    drv.Context.synchronize()
    self.shift_rows(self, state_np.reshape(4, 4))
    addRoundKeyKernel(
        drv.InOut(state_np),
        drv.In(key_schedule_np),
        np.int32(nr),
        block=(16, 1, 1),
        grid=(1, 1)
    )

    cipher_gpu = self.bytes_from_state(self, state_np.reshape(4, 4).tolist())

    return cipher_gpu
finally:
    CudaContext.pop_context()

```

```

def inv_shift_rows(self, state: [[int]]) -> [[int]]:
    # [00, 10, 20, 30]      [00, 10, 20, 30]
    # [01, 11, 21, 31] <-- [11, 21, 31, 01]
    # [02, 12, 22, 32]      [22, 32, 02, 12]
    # [03, 13, 23, 33]      [33, 03, 13, 23]
    state[1][1], state[2][1], state[3][1], state[0][1] = state[0][1], state[1][1],
state[2][1], state[3][1]
    state[2][2], state[3][2], state[0][2], state[1][2] = state[0][2], state[1][2],
state[2][2], state[3][2]
    state[3][3], state[0][3], state[1][3], state[2][3] = state[0][3], state[1][3],
state[2][3], state[3][3]
    return

```

```

def xtimes_0e(self, b):
    # 0x0e = 14 = b1110 = ((x * 2 + x) * 2 + x) * 2
    return self.xtime(self, self.xtime(self, self.xtime(self, b) ^ b) ^ b)

```

```

def xtimes_0b(self, b):
    # 0x0b = 11 = b1011 = ((x*2)*2+x)*2+x
    return self.xtime(self, self.xtime(self, self.xtime(self, b)) ^ b) ^ b

```

```
def xtimes_0d(self, b):
    # 0x0d = 13 = b1101 = ((x*2+x)*2)*2+x
    return self.xtime(self, self.xtime(self, self.xtime(self, b) ^ b)) ^ b

def xtimes_09(self, b):
    # 0x09 = 9 = b1001 = ((x*2)*2)*2+x
    return self.xtime(self, self.xtime(self, self.xtime(self, b))) ^ b

def inv_mix_column(self, col: [int]):
    c_0, c_1, c_2, c_3 = col[0], col[1], col[2], col[3]
    col[0] = self.xtimes_0e(self, c_0) ^ self.xtimes_0b(self, c_1) ^
self.xtimes_0d(self, c_2) ^ self.xtimes_09(self, c_3)
    col[1] = self.xtimes_09(self, c_0) ^ self.xtimes_0e(self, c_1) ^
self.xtimes_0b(self, c_2) ^ self.xtimes_0d(self, c_3)
    col[2] = self.xtimes_0d(self, c_0) ^ self.xtimes_09(self, c_1) ^
self.xtimes_0e(self, c_2) ^ self.xtimes_0b(self, c_3)
    col[3] = self.xtimes_0b(self, c_0) ^ self.xtimes_0d(self, c_1) ^
self.xtimes_09(self, c_2) ^ self.xtimes_0e(self, c_3)

def inv_mix_columns(self, state: [[int]]) -> [[int]]:
    for r in state:
        self.inv_mix_column(self, r)

def aes_decryption(self, cipher: bytes, key: bytes) -> bytes:

    if CudaContext.initialized:
        CudaContext.pop_context()

    CudaContext.get_context()

    try:
        aes_kernels = """
            __global__ void invSubBytesKernel(unsigned char *state, const unsigned
char *inv_s_box) {
                int idx = blockIdx.x * blockDim.x + threadIdx.x;
                if (idx < 16) {
                    state[idx] = inv_s_box[state[idx]];
                }
            }
        """
```

```
    }
    }

    __global__ void addRoundKeyKernel(unsigned char *state, unsigned char
*key_schedule, int round) {
        int idx = blockIdx.x * blockDim.x + threadIdx.x; // Índice lineal
para cada byte del estado

        if (idx < 16) { // Aseguramos que solo procesamos los 16 bytes del
estado
            int round_key_idx = round * 16 + idx;
            state[idx] ^= key_schedule[round_key_idx];
        }
    }

    """
    mod = SourceModule(aes_kernels)
    addRoundKeyKernel = mod.get_function("addRoundKeyKernel")
    invSubBytesKernel = mod.get_function("invSubBytesKernel")

    key_byte_length = len(key)
    key_bit_length = key_byte_length * 8
    nk = key_byte_length // 4

    if key_bit_length == 128:
        nr = 10
    elif key_bit_length == 192:
        nr = 12
    else: # 256-bit keys
        nr = 14

    state = self.state_from_bytes(self, cipher)
    flat_state = [byte for row in state for byte in row]
    state_np = np.array(flat_state, dtype=np.uint8)

    key_schedule = self.key_expansion(self, key)
    flattened_key_schedule = [byte for round_key in key_schedule for row in
round_key for byte in row]
    key_schedule_np = np.array(flattened_key_schedule, dtype=np.uint8)

    addRoundKeyKernel(
        drv.InOut(state_np),
        drv.In(key_schedule_np),
```



```
        np.int32(nr),
        block=(16, 1, 1),
        grid=(1, 1)
    )

    inv_s_box_np = np.array(self.get_inv_s_box(self), dtype=np.uint8)

    for round in range(nr-1, 0, -1):
        self.inv_shift_rows(self, state_np.reshape(4, 4))
        invSubBytesKernel(
            drv.InOut(state_np),
            drv.In(inv_s_box_np),
            block=(16, 1, 1),
            grid=(1, 1)
        )
        addRoundKeyKernel(
            drv.InOut(state_np),
            drv.In(key_schedule_np),
            np.int32(round),
            block=(16, 1, 1),
            grid=(1, 1)
        )
        self.inv_mix_columns(self, state_np.reshape(4, 4))

    self.inv_shift_rows(self, state_np.reshape(4, 4))
    invSubBytesKernel(
        drv.InOut(state_np),
        drv.In(inv_s_box_np),
        block=(16, 1, 1),
        grid=(1, 1)
    )
    addRoundKeyKernel(
        drv.InOut(state_np),
        drv.In(key_schedule_np),
        np.int32(0),
        block=(16, 1, 1),
        grid=(1, 1)
    )

    plain = self.bytes_from_state(self, state_np.reshape(4, 4).tolist())
    return plain
finally:
    CudaContext.pop_context()
```

```
def encrypt(self, plaintext, key_size, public_key, iv):

    if public_key == None or public_key == '':
        iv = os.urandom(16) # Vector de Inicialización aleatorio
        key = self.generate_secure_key(key_size)
        ciphertext = self.aes_encrypt_cbc(self, text=plaintext, iv=iv, key=key)
        encoded_ciphertext = base64.b64encode(ciphertext).decode('utf-8')
        encoded_key = base64.b64encode(key).decode('utf-8')
        encoded_iv = base64.b64encode(iv).decode('utf-8')
        return {
            'ciphertext': encoded_ciphertext,
            'key': encoded_key,
            'iv': encoded_iv
        }
    else:
        ciphertext = self.aes_encrypt_cbc(self, text=plaintext, iv=iv,
key=public_key)
        encoded_ciphertext = base64.b64encode(ciphertext)
        return {
            'ciphertext': encoded_ciphertext,
            'key': 0,
            'iv': 0
        }

def decrypt(self, ciphertext, public_key, iv):
    try:
        decrypted_plaintext = self.aes_decrypt_cbc(self, encrypted_text=ciphertext,
key=public_key, iv=iv)
        if decrypted_plaintext == '':
            return 'Error al desencriptar el texto.'
        else:
            return decrypted_plaintext
    except:
        return 'Error al desencriptar el texto.'
```

Algoritmo RSA

```
from .cuda_context import CudaContext
```

```
from pycuda.compiler import SourceModule
import numpy as np
import pycuda.driver as drv
import random
import base64
import re

class RsaEncryptGpu():

    def get_power_2_factors(self, n: int) -> (int, int):
        r, d = 0, n
        while d % 2 == 0:
            d //= 2
            r += 1
        return r, d

    def miller_rabin_prime_test_cuda(n: int, k: int, d: int, r: int) -> bool:

        if CudaContext.initialized:
            CudaContext.pop_context()

        CudaContext.get_context()

        try:
            rsa_kernel = """

            __device__ int powerMod(long long a, long long b, long long m) {
                long long result = 1;
                long long x = a % m;

                for (int i = 1; i <= b; i <= 1) {
                    if ((b & i) != 0) {
                        result = (result * x) % m;
                    }
                    x = (x * x) % m;
                }
                return result;
            }

            __global__ void millerRabinKernel(long long n, int k, long long d, long long
r, bool* isPrimeArray) {
                int idx = threadIdx.x + blockIdx.x * blockDim.x;
```

```
        if (idx >= k) return;

        long long a = 2 + idx;
        long long x = powerMod(a, d, n);

        if (x == 1 || x == n - 1) {
            isPrimeArray[idx] = true;
            return;
        }

        int powerOf2 = 1;
        for (int i = 0; i < r - 1; i++) {
            x = powerMod(a, d * powerOf2, n);
            if (x == n - 1) {
                isPrimeArray[idx] = true;
                return;
            }
            powerOf2 *= 2;
        }

        isPrimeArray[idx] = false;
    }

    """"
    mod = SourceModule(rsa_kernel)
    miller_rabin_kernel = mod.get_function("millerRabinKernel")
    is_prime_array = np.zeros(k, dtype=bool)
    miller_rabin_kernel(
        np.int64(n),
        np.int64(k),
        np.int64(d),
        np.int64(r),
        drv.Out(is_prime_array),
        block=(1024, 1, 1),
        grid=(1, 1, 1)
    )
    drv.Context.synchronize()
    return np.all(is_prime_array)
finally:
    CudaContext.pop_context()

def generate_prime_number_gpu(self, bit_length: int) -> int:
```

```
low, high = 2** (bit_length - 1), 2** bit_length - 1

while True:
    candidate = random.randrange(low, high)
    r, d = self.get_power_2_factors(self, candidate - 1)
    if candidate % 2 != 0 and self.miller_rabin_prime_test_cuda(candidate, 64,
d, r):
        return candidate

def extended_gcd(self, a, b):
    if not b:
        return 1, 0
    u, v = self.extended_gcd(self, b, a % b)
    return v, u - v * (a // b)

def calculate_private_key(self, e: int, p: int, q: int) -> int:
    return self.extended_gcd(self, e, (p-1)*(q-1))[0]

def rsa_encrypt_blockwise(self, plaintext_bytes, e, n):
    block_size = (n.bit_length() + 7) // 8 - 1
    return b''.join(pow(int.from_bytes(plaintext_bytes[i:i + block_size], "big"), e,
n).to_bytes((n.bit_length() + 7) // 8, 'big') for i in range(0, len(plaintext_bytes),
block_size))

def rsa_decrypt_blockwise(self, ciphertext, d, n):
    block_size = (n.bit_length() + 7) // 8
    return b''.join(pow(int.from_bytes(ciphertext[i:i + block_size], "big"), d,
n).to_bytes(block_size, 'big') for i in range(0, len(ciphertext),
block_size)).rstrip(b'\x00')

def encrypt(self, plaintext, rsa_key_size, public_key):

    e = 65537
    if public_key == None or public_key == '':
        prime_number_bit_length = rsa_key_size // 2
        # Generate prime numbers p and q
        p = self.generate_prime_number_gpu(self, prime_number_bit_length)
        q = self.generate_prime_number_gpu(self, prime_number_bit_length)
```

```
# Calculate public key
n = p * q

# Calculate private key
d = self.calculate_private_key(self, e, p, q)
ciphertext = self.rsa_encrypt_blockwise(self, plaintext.encode(), e, n)

encoded_ciphertext = base64.b64encode(ciphertext).decode('utf-8')

return {
    'ciphertext': encoded_ciphertext,
    'd': d,
    'n': n
}

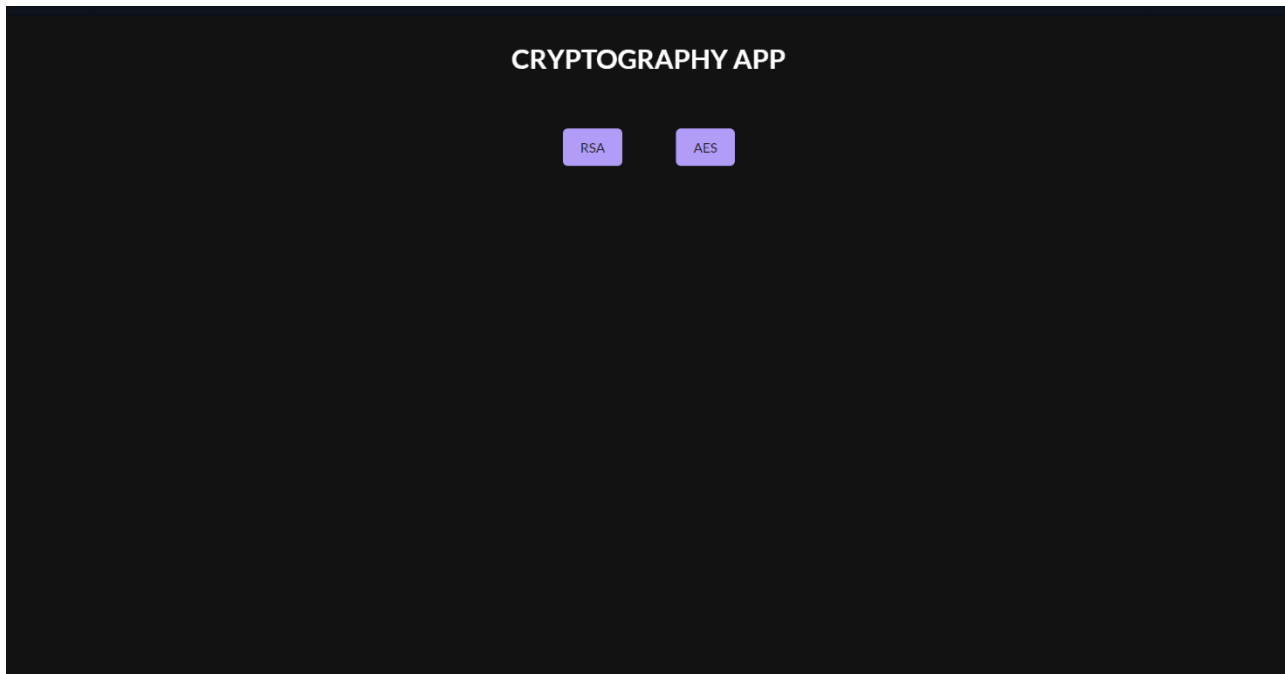
else:
    n = public_key
    ciphertext = base64.b64decode(plaintext)
    ciphertext = self.rsa_encrypt_blockwise(self, plaintext.encode(), e, n)
    encoded_ciphertext = base64.b64encode(ciphertext).decode('utf-8')
    return {
        'ciphertext': encoded_ciphertext,
        'd': 0,
        'n': 0
    }

def decrypt(self, ciphertext, d, n):
    try:
        recovered_plaintext_bytes = self.rsa_decrypt_blockwise(self, ciphertext, d,
n)
        recovered_plaintext = recovered_plaintext_bytes.decode('utf-8')
        recovered_plaintext = re.sub(r'\x00', '', recovered_plaintext)
        if recovered_plaintext == '':
            return 'Error al desencriptar el mensaje'
        else:
            return recovered_plaintext
    except:
        return 'Error al desencriptar el mensaje'
```

	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

3. Crear una aplicación web en Django que permita a los usuarios cifrar y descifrar archivos de texto. Integrar la lógica de cifrado/descifrado en la aplicación Django.

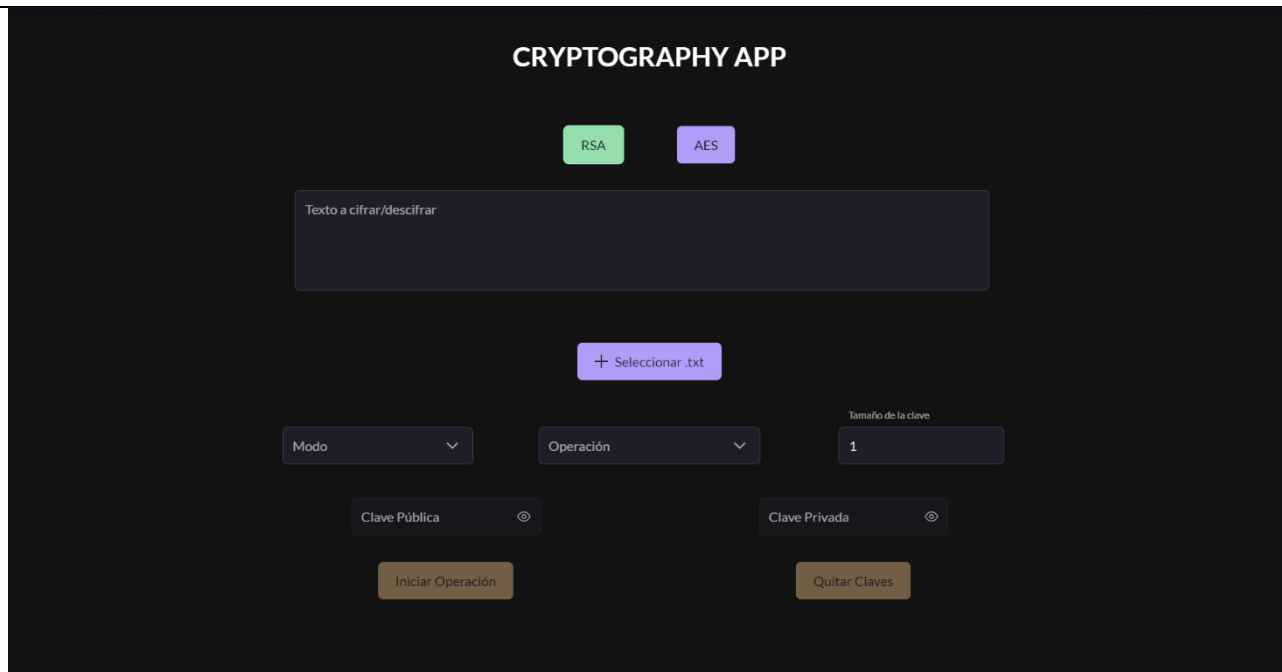
Primeramente, como podemos ver está es la página principal de la aplicación donde podemos ver el título y 2 opciones a elegir: RSA o AES. Cada una de estas opciones desplegará una misma interfaz.



Ahora, como podemos ver hemos elegido la opción de RSA y se ha desplegado una interfaz compuesta de:

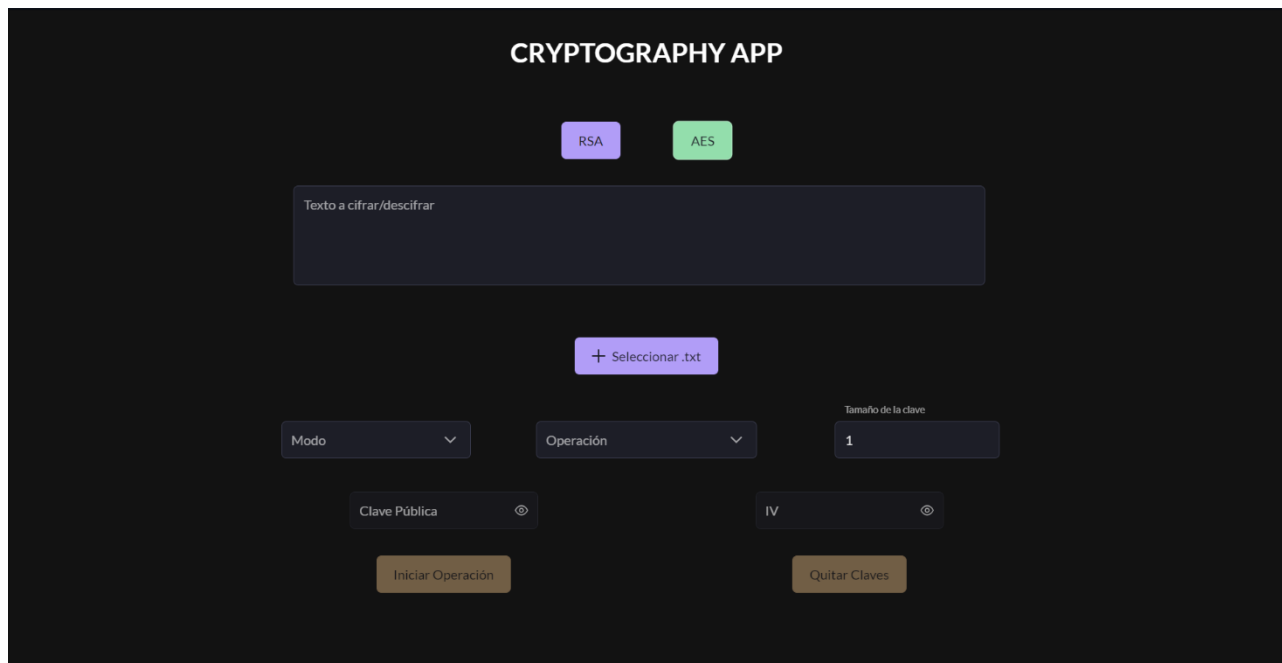
- Text Area: donde se desplegará el texto que el usuario suba o que escriba.
- Botón "Seleccionar .txt": es un botón de tipo file donde se podrá elegir únicamente entre archivos de tipo .txt para poder subirlos.
- Botón "Modo": donde se podrá elegir entre CPU o GPU para realizar la encriptación.
- Botón "Operación": donde se podrá elegir entre Encriptar o Desencriptar.
- Input "Tamaño de la clave": donde se podrá colocar el tamaño de la clave preferida.
- Input "Clave Pública": donde se colocará la clave pública elegida.
- Input "Clave Privada": donde se colocará la clave privada.
- Botón "Iniciar Operación": botón a pulsar para realizar la encriptación.
- Botón "Quitar Claves": retirará las claves elegidas con el fin de generar nuevas.

	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		



The screenshot shows the 'CRYPTOGRAPHY APP' interface. At the top, there are two buttons: 'RSA' (highlighted in green) and 'AES' (purple). Below them is a large text input field labeled 'Texto a cifrar/descifrar'. Underneath this field is a button labeled '+ Seleccionar .txt'. Further down, there are three dropdown menus: 'Modo', 'Operación', and 'Tamaño de la clave' (which has '1' selected). Below these are two input fields: 'Clave Pública' and 'Clave Privada', each with a toggle icon. At the bottom, there are two buttons: 'Iniciar Operación' and 'Quitar Claves'.

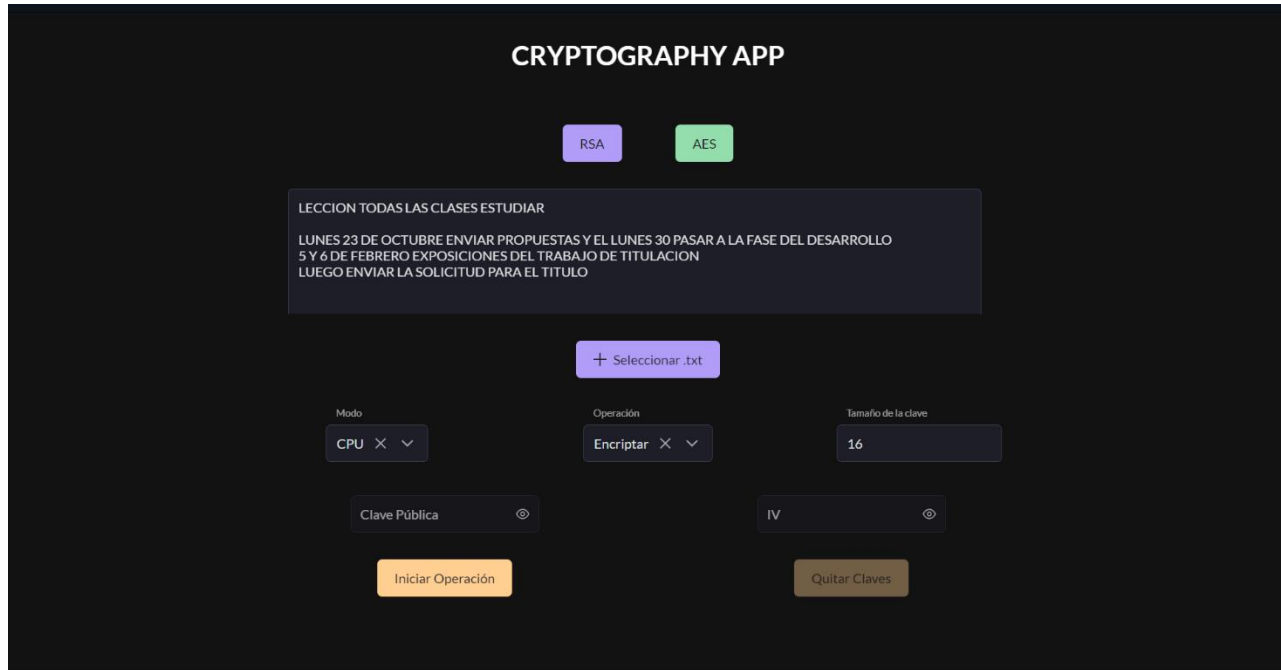
Después, vemos que hemos elegido la opción de AES y cómo podemos ver es prácticamente igual a la anterior salvo por el Input “IV” donde se colocará el “IV” elegido.



This screenshot shows the same 'CRYPTOGRAPHY APP' interface, but with 'AES' selected instead of 'RSA'. The 'AES' button is now highlighted in green. The 'Clave Privada' input field has been replaced by an 'IV' input field, also with a toggle icon. All other elements, including the text input field, the '+ Seleccionar .txt' button, the dropdown menus, and the 'Iniciar Operación' and 'Quitar Claves' buttons, remain the same as in the previous screenshot.

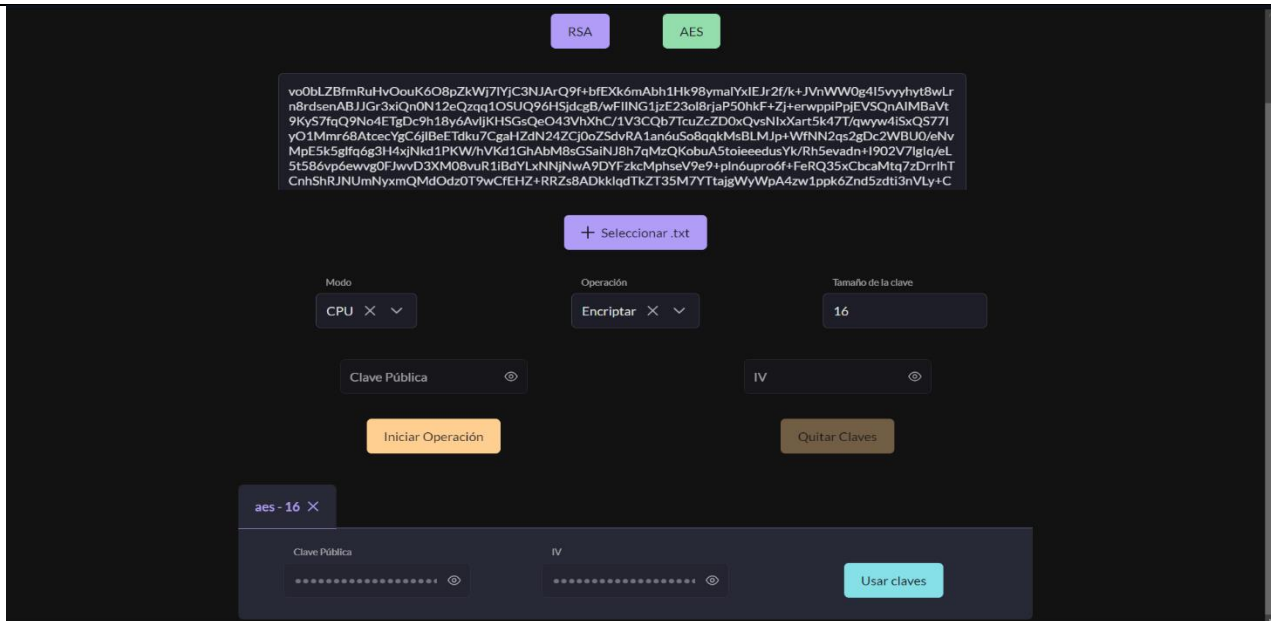
	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

Mas tarde, nos quedamos en AES y cargamos un archivo de texto. Seleccionas las opciones preferidas y damos click en “Iniciar Operación”.

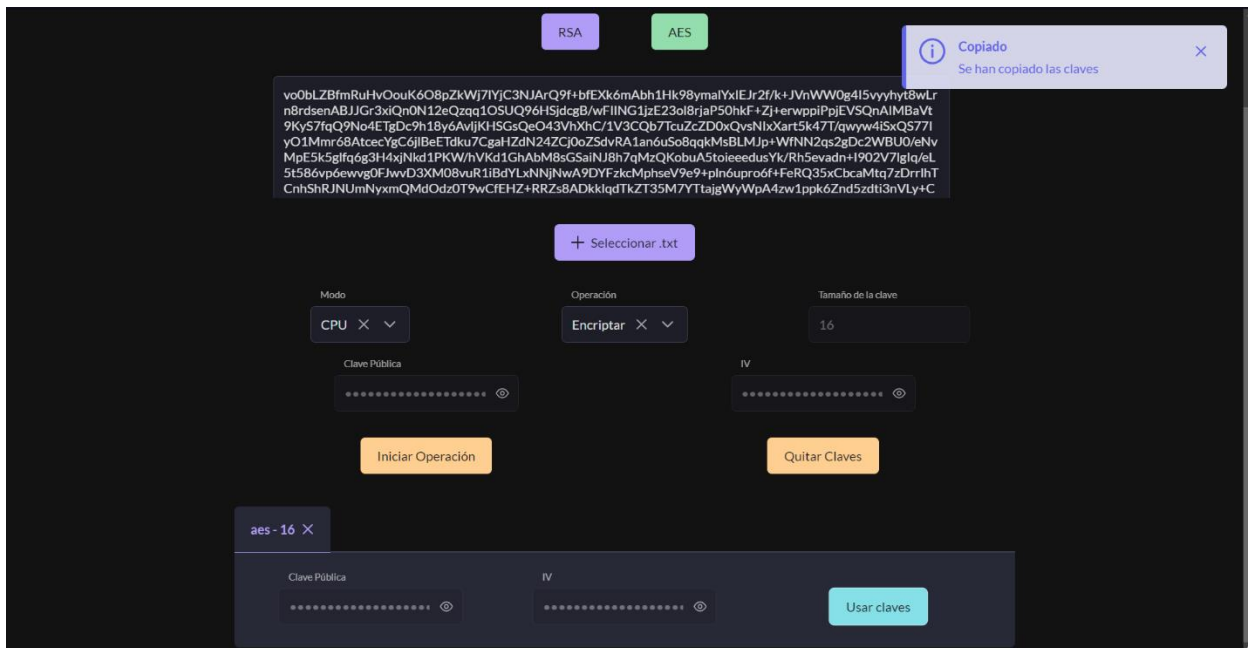


Tal como podemos ver el texto se ha encriptado y ha aparecido un nuevo elemento en la interfaz. El elemento inferior despliega las claves que se han generado y tenemos la opción de podemos verlas de manera limitada y seleccionarlal. Decidimos hacer esto para agilizar el proceso de encriptación debido a que así podemos saltarnos todo el proceso de generación de claves y pasar directo a la encriptación. Además, es más útil para un usuario seleccionar simplemente una clave generada y empezar con el proceso.

	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

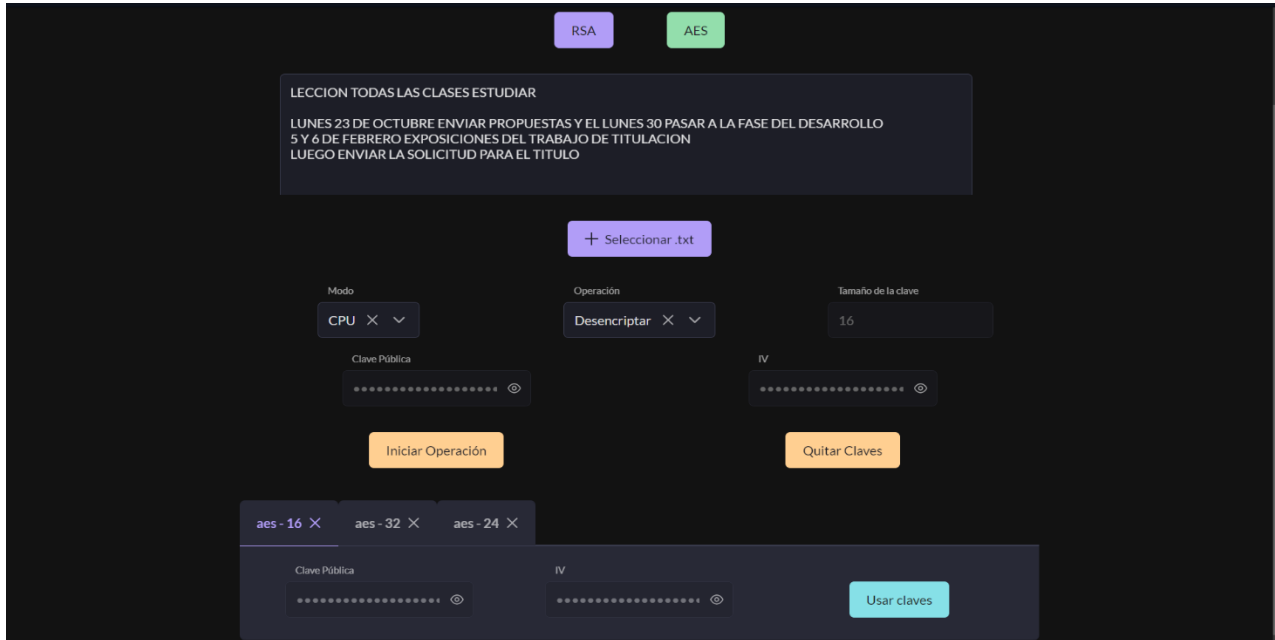


Como podemos ver en esta imagen hemos colocado las claves de las que disponemos y procedimos a encriptar el texto. Una vez se coloquen las claves se activa el Botón “Quitar Claves” con el fin de vaciar las claves y poder generar otras. Además, cuando se colocan las claves se deshabilita el Input para colocar el tamaño de clave ya que este se coloca automáticamente dependiendo de las claves seleccionadas.

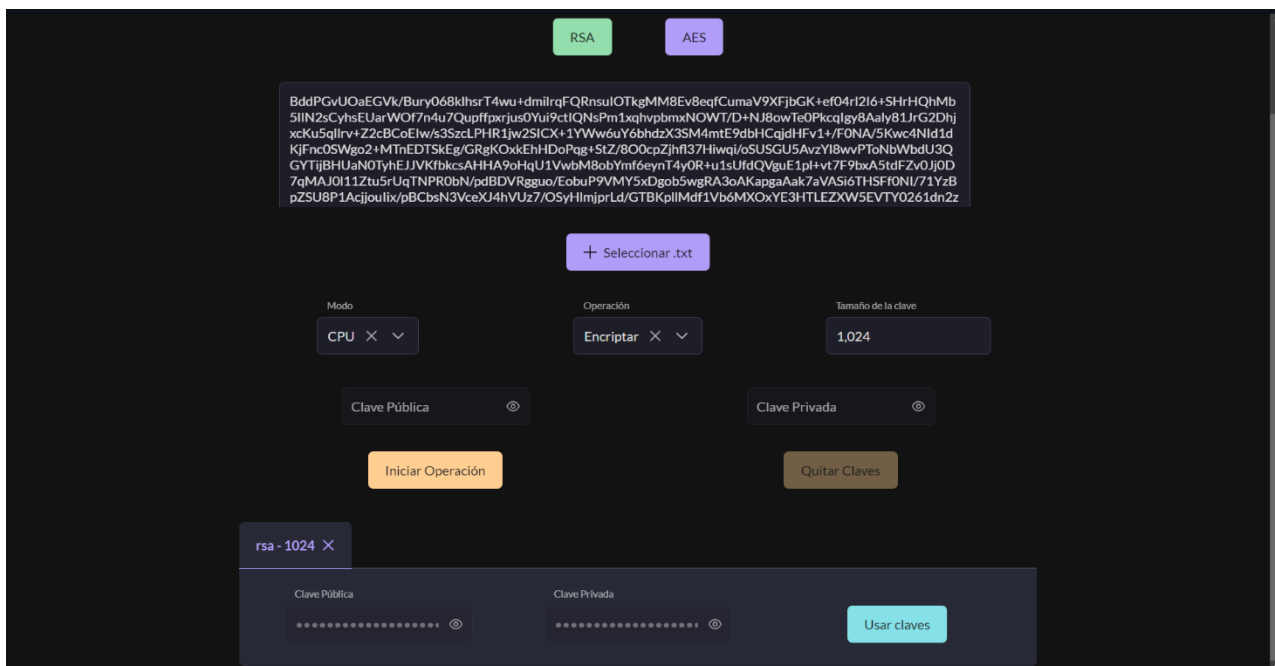


	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

Como último punto de esta pantalla podemos ver como hemos generado varias claves estándar para AES que cumplen con AES-128, AES-192 Y AES-256 los cuales son tamaños de clave de 16, 24 y 32 respectivamente.

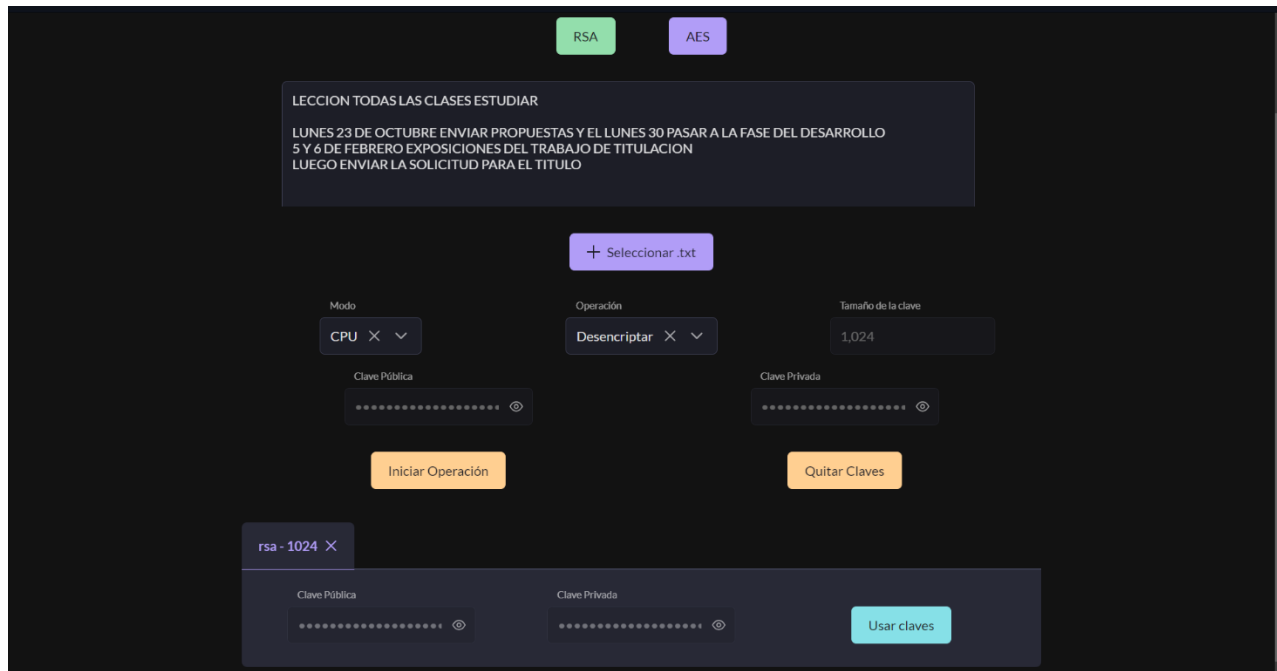


Una vez conocido el funcionamiento de la aplicación procedemos rápidamente por la pantalla de RSA. Como vemos hemos encriptado un texto con un tamaño de clave de 1024 el cual actualmente está obsoleto. El texto se ha encriptado correctamente y se ha agregado la clave a la lista.



	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

Luego, descriptamos el texto al elegir usar la clave recientemente generada.



The screenshot shows a web-based interface for RSA operations. At the top, there are two tabs: 'RSA' (active) and 'AES'. Below the tabs, a dark grey box contains the text: 'LECCION TODAS LAS CLASES ESTUDIAR', 'LUNES 23 DE OCTUBRE ENVIAR PROPUESTAS Y EL LUNES 30 PASAR A LA FASE DEL DESARROLLO', '5 Y 6 DE FEBRERO EXPOSICIONES DEL TRABAJO DE TITULACION', and 'LUEGO ENVIAR LA SOLICITUD PARA EL TITULO'. Below this, there is a '+ Seleccionar .txt' button. The main interface is divided into three sections: 'Modo' (set to 'CPU'), 'Operación' (set to 'Descriptar'), and 'Tamaño de la clave' (set to '1,024'). Below these, there are two input fields: 'Clave Pública' and 'Clave Privada', both containing masked text. Below the input fields, there are two buttons: 'Iniciar Operación' and 'Quitar Claves'. At the bottom, there is a section titled 'rsa - 1024' which contains two input fields: 'Clave Pública' and 'Clave Privada', both containing masked text, and a 'Usar claves' button.

Por último, podemos ver como hemos generado varios tamaños de clave que son estándar para RSA. Actualmente el tamaño de 1024 es obsoleto por ende los tamaños de clave recomendados van desde el 2048 en adelante.

4. Crear un Dockerfile para construir una imagen Docker de la aplicación Django.

```
# Usa una imagen base de NVIDIA CUDA
FROM nvidia/cuda:11.2.2-cudnn8-devel-ubuntu20.04

# Actualiza el sistema y añade las dependencias necesarias
RUN apt-get update && \
    apt-get install -y python3-pip python3-dev

# Limpia el caché de APT para reducir el tamaño de la imagen
RUN apt-get clean && rm -rf /var/lib/apt/lists/*

# Crea y cambia al directorio de trabajo de la aplicación
WORKDIR /app

# Copia los archivos necesarios a la imagen de Docker
COPY requirements.txt /app/
COPY . /app/

# Instala las dependencias de la aplicación
RUN pip3 install -r requirements.txt

# Expone el puerto 8000 para acceder a la aplicación
EXPOSE 8000

# Define el comando para ejecutar la aplicación
CMD ["python3", "manage.py", "runserver", "0.0.0.0:8000"]
```

5. Crear un Dockerfile para construir una imagen Docker de la aplicación Angular.

```
# Usa la imagen de Nginx
FROM nginx:alpine

# Copia los archivos estáticos construidos a la ubicación esperada por Nginx
COPY /dist/django-front /usr/share/nginx/html

# (Opcional) Si tienes una configuración personalizada de Nginx, inclúyela
COPY default.conf /etc/nginx/conf.d/default.conf

# Expone el puerto 80
EXPOSE 80

# Inicia Nginx
CMD ["nginx", "-g", "daemon off;"]
```

	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		


6. Configurar docker-compose para gestionar la aplicación

```
version: '3.8'
services:
  frontend:
    image: front-interciclo:latest
    ports:
      - "80:80" # Mapea el puerto 80 del contenedor al puerto 80 del host
    depends_on:
      - backend # Indica que el frontend depende del backend

  backend:
    image: back-interciclo:latest
    ports:
      - "8000:8000" # Ajusta este puerto según el puerto en el que tu backend esté escuchando
    deploy:
      resources:
        reservations:
          devices:
            - driver: nvidia
              capabilities: [gpu]
```

- Ejecución del docker-compose

```
jhell@jhell:~/Downloads/Proyecto_Interciclo_Frontend$ sudo docker-compose up --build
Starting proyecto_interciclo_frontend_backend_1 ... done
Starting proyecto_interciclo_frontend_frontend_1 ... done
Attaching to proyecto_interciclo_frontend_backend_1, proyecto_interciclo_frontend_frontend_1
frontend_1 | /docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
frontend_1 | /docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
backend_1 | /docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
frontend_1 | 10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
backend_1 | =====
backend_1 | == CUDA ==
backend_1 | =====
backend_1 | CUDA Version 11.2.2
backend_1 | Container image Copyright (c) 2016-2023, NVIDIA CORPORATION & AFFILIATES. All rights reserved.
backend_1 | This container image and its contents are governed by the NVIDIA Deep Learning Container License.
backend_1 | By pulling and using the container, you accept the terms and conditions of this license:
backend_1 | https://developer.nvidia.com/ngc/nvidia-deep-learning-container-license
backend_1 | A copy of this license is made available in this container at /NGC-DL-CONTAINER-LICENSE for your convenience.
frontend_1 | 10-listen-on-ipv6-by-default.sh: info: /etc/nginx/conf.d/default.conf differs from the packaged version
frontend_1 | /docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
frontend_1 | /docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
frontend_1 | /docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
frontend_1 | /docker-entrypoint.sh: Configuration complete; ready for start up
frontend_1 | 2023/11/30 22:14:16 [notice] 1#1: using the "epoll" event method
frontend_1 | 2023/11/30 22:14:16 [notice] 1#1: nginx/1.25.3
frontend_1 | 2023/11/30 22:14:16 [notice] 1#1: built by gcc 12.2.1 20220924 (Alpine 12.2.1_git20220924-r10)
frontend_1 | 2023/11/30 22:14:16 [notice] 1#1: OS: Linux 6.2.0-37-generic
frontend_1 | 2023/11/30 22:14:16 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
frontend_1 | 2023/11/30 22:14:16 [notice] 1#1: start worker processes
frontend_1 | 2023/11/30 22:14:16 [notice] 1#1: start worker process 29
frontend_1 | 2023/11/30 22:14:16 [notice] 1#1: start worker process 30
frontend_1 | 2023/11/30 22:14:16 [notice] 1#1: start worker process 31
frontend_1 | 2023/11/30 22:14:16 [notice] 1#1: start worker process 32
frontend_1 | 2023/11/30 22:14:16 [notice] 1#1: start worker process 33
frontend_1 | 2023/11/30 22:14:16 [notice] 1#1: start worker process 34
frontend_1 | 2023/11/30 22:14:16 [notice] 1#1: start worker process 35
frontend_1 | 2023/11/30 22:14:16 [notice] 1#1: start worker process 36
backend_1 | Watching for file changes with StatReloader
```

	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

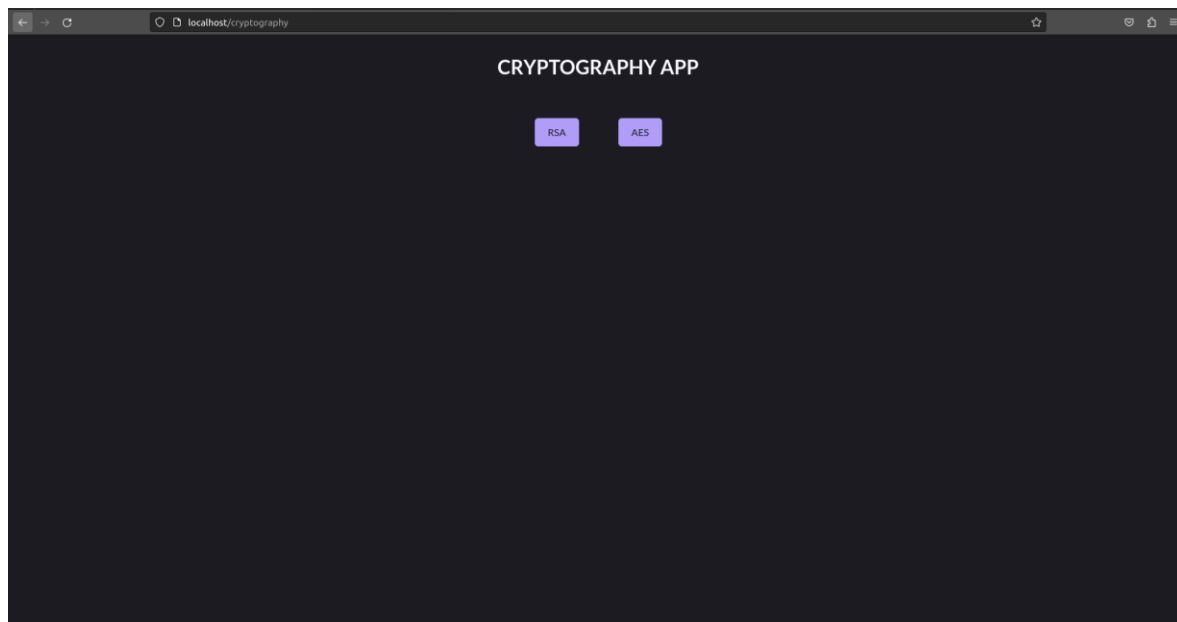
- Peticiones realizadas

```

backend_1 | [30/Nov/2023 22:18:54] "POST /cryptography/encrypt HTTP/1.1" 200 87
backend_1 | [30/Nov/2023 22:19:01] "POST /cryptography/decrypt HTTP/1.1" 200 52
frontend_1 | 172.20.0.1 - - [30/Nov/2023:22:19:10 +0000] "GET /cryptography/rsa HTTP/1.1" 200 6037 "-" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/116.0"
frontend_1 | 172.20.0.1 - - [30/Nov/2023:22:19:10 +0000] "GET /runtime.84c2c9745f66245a.js HTTP/1.1" 200 2700 "http://localhost/cryptography/rsa" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/116.0"
frontend_1 | 172.20.0.1 - - [30/Nov/2023:22:19:10 +0000] "GET /polyfills.b8f2e2f1e2ed2e18.js HTTP/1.1" 200 33827 "http://localhost/cryptography/rsa" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/116.0"
frontend_1 | 172.20.0.1 - - [30/Nov/2023:22:19:10 +0000] "GET /main.4e643f2b20455696.js HTTP/1.1" 200 358083 "http://localhost/cryptography/rsa" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/116.0"
frontend_1 | 172.20.0.1 - - [30/Nov/2023:22:19:10 +0000] "GET /styles.46c9dad873d8b487.css HTTP/1.1" 200 678752 "http://localhost/cryptography/rsa" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/116.0"
frontend_1 | 172.20.0.1 - - [30/Nov/2023:22:19:10 +0000] "GET /410.d0cb509e9e5238ea.js HTTP/1.1" 200 307016 "http://localhost/cryptography/rsa" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/116.0"
frontend_1 | 172.20.0.1 - - [30/Nov/2023:22:19:10 +0000] "GET /lato-v17-latin-ext_latin-regular.56d5c18495ed5c43.woff2 HTTP/1.1" 200 25320 "http://localhost/cryptography/rsa" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/116.0"
frontend_1 | 172.20.0.1 - - [30/Nov/2023:22:19:10 +0000] "GET /favicon.ico HTTP/1.1" 200 949 "http://localhost/cryptography/rsa" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/116.0"
frontend_1 | 172.20.0.1 - - [30/Nov/2023:22:19:10 +0000] "GET /lato-v17-latin-ext_latin-700.099b6e41b78d99dc.woff2 HTTP/1.1" 200 24712 "http://localhost/cryptography/rsa" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/116.0"
frontend_1 | 172.20.0.1 - - [30/Nov/2023:22:19:10 +0000] "GET /lato-v17-latin-ext_latin-regular.56d5c18495ed5c43.woff2 HTTP/1.1" 200 25320 "http://localhost/styles.46c9dad873d8b487.css" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/116.0"
frontend_1 | 172.20.0.1 - - [30/Nov/2023:22:19:10 +0000] "GET /lato-v17-latin-ext_latin-700.099b6e41b78d99dc.woff2 HTTP/1.1" 200 24712 "http://localhost/styles.46c9dad873d8b487.css" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/116.0"
frontend_1 | 172.20.0.1 - - [30/Nov/2023:22:19:24 +0000] "GET /assets/loader.svg HTTP/1.1" 200 2395 "http://localhost/cryptography/aes" "Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:109.0) Gecko/20100101 Firefox/116.0"
backend_1 | [30/Nov/2023 22:19:24] "POST /cryptography/encrypt HTTP/1.1" 200 117
backend_1 | [30/Nov/2023 22:19:26] "POST /cryptography/decrypt HTTP/1.1" 200 25
backend_1 | Internal Server Error: /cryptography/encrypt
backend_1 | [30/Nov/2023 22:19:42] "POST /cryptography/encrypt HTTP/1.1" 500 62
backend_1 | [30/Nov/2023 22:19:51] "POST /cryptography/encrypt HTTP/1.1" 200 843
backend_1 | [30/Nov/2023 22:19:56] "POST /cryptography/decrypt HTTP/1.1" 200 25
backend_1 | [30/Nov/2023 22:20:16] "POST /cryptography/encrypt HTTP/1.1" 200 105
backend_1 | Bad Request: /cryptography/decrypt
backend_1 | [30/Nov/2023 22:20:15] "POST /cryptography/decrypt HTTP/1.1" 400 36
backend_1 | [30/Nov/2023 22:20:21] "POST /cryptography/decrypt HTTP/1.1" 200 25
backend_1 | [30/Nov/2023 22:20:31] "POST /cryptography/encrypt HTTP/1.1" 200 64
backend_1 | [30/Nov/2023 22:20:33] "POST /cryptography/decrypt HTTP/1.1" 200 25
backend_1 | [30/Nov/2023 22:20:36] "POST /cryptography/encrypt HTTP/1.1" 200 64
backend_1 | [30/Nov/2023 22:20:37] "POST /cryptography/encrypt HTTP/1.1" 200 76
backend_1 | [30/Nov/2023 22:20:38] "POST /cryptography/encrypt HTTP/1.1" 200 96
backend_1 | [30/Nov/2023 22:20:38] "POST /cryptography/encrypt HTTP/1.1" 200 128
backend_1 | [30/Nov/2023 22:20:39] "POST /cryptography/encrypt HTTP/1.1" 200 172
backend_1 | [30/Nov/2023 22:20:40] "POST /cryptography/encrypt HTTP/1.1" 200 244
backend_1 | [30/Nov/2023 22:20:40] "POST /cryptography/encrypt HTTP/1.1" 200 352
backend_1 | [30/Nov/2023 22:20:41] "POST /cryptography/encrypt HTTP/1.1" 200 512
backend_1 | [30/Nov/2023 22:20:41] "POST /cryptography/encrypt HTTP/1.1" 200 756
backend_1 | [30/Nov/2023 22:20:48] "POST /cryptography/decrypt HTTP/1.1" 200 481
backend_1 | [30/Nov/2023 22:20:48] "POST /cryptography/decrypt HTTP/1.1" 200 321
backend_1 | [30/Nov/2023 22:20:49] "POST /cryptography/decrypt HTTP/1.1" 200 213
backend_1 | [30/Nov/2023 22:20:50] "POST /cryptography/decrypt HTTP/1.1" 200 141
backend_1 | [30/Nov/2023 22:20:51] "POST /cryptography/decrypt HTTP/1.1" 200 97
backend_1 | [30/Nov/2023 22:20:52] "POST /cryptography/decrypt HTTP/1.1" 200 65
backend_1 | [30/Nov/2023 22:20:53] "POST /cryptography/decrypt HTTP/1.1" 200 45
backend_1 | [30/Nov/2023 22:20:53] "POST /cryptography/decrypt HTTP/1.1" 200 33
backend_1 | [30/Nov/2023 22:20:54] "POST /cryptography/decrypt HTTP/1.1" 200 25

```

- Ejecución del localhost



	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

7. Desarrollar pruebas para verificar la correcta ejecución del cifrado y descifrado. Obtener tiempos de procesamiento para comparar el rendimiento de la GPU frente a la CPU.

Para corroborar la eficacia de nuestros algoritmos desarrollados a la hora de encriptar o desencriptar vamos a someterlos a pruebas bien conocidas.

Empezando por el algoritmo AES vamos a usar las pruebas que se encuentran en el apéndice c “Example Vector” del NIST-197 donde se encuentra la descripción técnica del algoritmo. En ese apéndice tenemos tres vectores de prueba para cada extensión de clave, es decir: un vector para AES-128, otro para AES-192 y otro para AES-256. El objetivo principal es que con el texto y clave dadas hay que descifrar el texto. El código será el siguiente:

```
if __name__ == "__main__":

    time_start = time.time()

    # NIST FIPS PUB 197 ADVANCED ENCRYPTION STANDARD (AES)

    # NIST AES-128 test vector 1 (Ch. C.1, p. 35)
    plaintext = bytearray.fromhex('00112233445566778899aabbccddeeff')
    key = bytearray.fromhex('000102030405060708090a0b0c0d0e0f')
    expected_ciphertext = bytearray.fromhex('69c4e0d86a7b0430d8cdb78070b4c55a')
    ciphertext = aes_encryption(plaintext, key)
    recovered_plaintext = aes_decryption(ciphertext, key)

    assert (ciphertext == expected_ciphertext)
    assert (recovered_plaintext == plaintext)

    # NIST AES-192 test vector 2 (Ch. C.2, p. 38)
    plaintext = bytearray.fromhex('00112233445566778899aabbccddeeff')
    key = bytearray.fromhex('000102030405060708090a0b0c0d0e0f1011121314151617')
    expected_ciphertext = bytearray.fromhex('dda97ca4864cdfef06eaf70a0ec0d7191')
    ciphertext = aes_encryption(plaintext, key)

    recovered_plaintext = aes_decryption(ciphertext, key)

    assert (ciphertext == expected_ciphertext)
    assert (recovered_plaintext == plaintext)

    # NIST AES-256 test vector 3 (Ch. C.3, p. 42)
    plaintext = bytearray.fromhex('00112233445566778899aabbccddeeff')
    key = bytearray.fromhex('000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f')
    expected_ciphertext = bytearray.fromhex('8ea2b7ca516745bfeafc49904b496089')
    ciphertext = aes_encryption(plaintext, key)
    recovered_plaintext = aes_decryption(ciphertext, key)

    assert (ciphertext == expected_ciphertext)
    assert (recovered_plaintext == plaintext)

    time_end = time.time()

    print("AES-128, AES-192, AES-256 test vectors passed in {} seconds".format(time_end - time_start))
```


	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

Es el mismo para ambos métodos: CPU y GPU. Estamos tomando el tiempo al inicio de toda la ejecución y al final después de haber descifrado los tres textos y pasar por las verificaciones si es que el texto cifrado es igual al cifrado esperado y si el texto plano es igual al texto plano requerido.

Modo	Tiempo
CPU	0,00207
GPU	0,07857

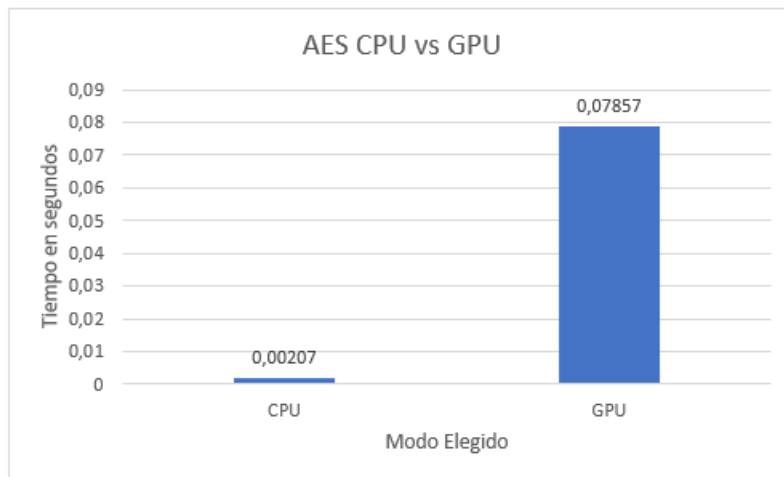


Ilustración 1 Resultados AES NIST: CPU vs GPU

En la imagen 1, se observa que la CPU tuvo un rendimiento considerablemente mejor que la GPU. Esto se debió a una distribución imprecisa del trabajo en paralelo, donde se aplicó la paralelización a funciones que trabajaban con conjuntos de datos diferentes y que no requerían invocar otras funciones o fusionar datos, ya que hacerlo habría complicado excesivamente las funciones de paralelización. Sin embargo, a pesar de esta estrategia, el rendimiento del algoritmo empeoró en las funciones que se paralelizaron, debido a los tiempos de llamada e inicio del proceso principal (kernel), junto con las operaciones de transformación a arreglos de numpy y la preparación correspondiente para que la GPU pueda procesar la información. Además, se realizaron transformaciones adicionales para poder emplear las funciones sin paralelizar.

Ahora, realizamos una prueba por nuestra cuenta donde encriptamos y desencriptamos un texto largo usando AES-CBC en su versión de AES-256. Este fue el resultado:

	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

Modo	Tiempo
GPU	10,45996
CPU	0,29198

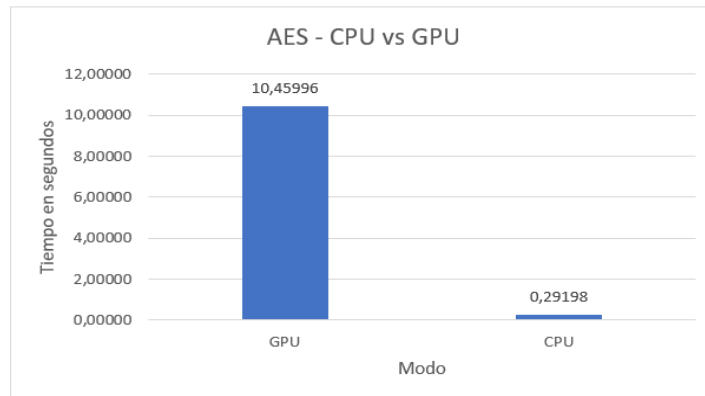


Ilustración 2 Resultados AES: CPU vs GPU

Tal como se puede observar en la imagen 2, en nuestra prueba la ejecución del algoritmo en la CPU fue nuevamente superior al algoritmo modificado para ejecutarse en la GPU.

Vamos a probar nuestra idea para el algoritmo RSA. No tenemos ejemplos específicos para este algoritmo, así que lo haremos de manera sencilla. Tomaremos un texto corto, lo encriptaremos y luego lo desencriptaremos usando diferentes tamaños de clave. Vamos a medir el tiempo que tarda en hacerlo. En esta ocasión, usaremos un texto corto porque la versión del algoritmo que usamos en la tarjeta gráfica no acepta claves más grandes que 63, ya que en el lenguaje C el número más grande que puede representarse es 2 elevado a la potencia 64. En la RSA, hoy en día, el estándar es usar claves de 2 elevado a la potencia 2048 o más. Para manejar números grandes en el lenguaje C, necesitaríamos usar librerías externas que no tenemos disponibles en PyCUDA o tendríamos que crear nuestras propias funciones para manejar la multiplicación, suma y exponenciación de números grandes. Esto último es un trabajo largo y complicado. Por eso, en la GPU solo podremos usar claves hasta 2 elevado a la potencia 63, que nos sirven para encriptar y desencriptar textos cortos. Aquí está el código que usaremos:

```
start = time.time()

rsa_key_size = 63
prime_number_bit_length = rsa_key_size // 2

# Generate prime numbers p and q
p = generate_prime_number(prime_number_bit_length)
q = generate_prime_number(prime_number_bit_length)

# Calculate public key
n = p * q
e = 65537

# Calculate private key
d = calculate_private_key(e, p, q)

# Encrypt
plaintext = ""
LECCION TODAS LAS CLASES ESTUDIAR
```

LUNES 23 DE OCTUBRE ENVIAR PROPUESTAS Y EL LUNES 30 PASAR A LA FASE DEL DESARROLLO
5 Y 6 DE FEBRERO EXPOSICIONES DEL TRABAJO DE TITULACION
LUEGO ENVIAR LA SOLICITUD PARA EL TITULO

Evaluando el rendimiento de un programa paralelo
Aceleramiento, eficiencia y escalabilidad
Este es un tema de software, encontrar recursos interactivos

Temas de seguridad
Los de la Costa por la inseguridad
Estado de las vías
Convenio con Startups, competir en Startups
UPS y Latacunga
Viernes → Logística, el resto de miembros, encargarse que todo el evento vaya bien
repartir comida, ver que todos los participantes disfruten el evento, repartir los energizantes
Premios a reducir, 3 trofeos, 9 medallas y el resto para la organización, snacks
2 proyectos. Detector de olores.
- Proponer el proyecto para el orfanato de azogues alguna idea de juego para repartir varios, además de donaciones
"""

```
ciphertext = rsa_encrypt_blockwise(plaintext.encode(), e, n)

# Decrypt
recovered_plaintext_bytes = rsa_decrypt_blockwise(ciphertext, d, n)
recovered_plaintext = recovered_plaintext_bytes.decode('utf-8')

end = time.time()
```

Primeramente, estos son los resultados cuando usamos una clave de 32, como vemos la GPU se sublevo a la CPU teniendo un tiempo de ejecución menor.

Modo	Tiempo
GPU	0,00996
CPU	0,01564

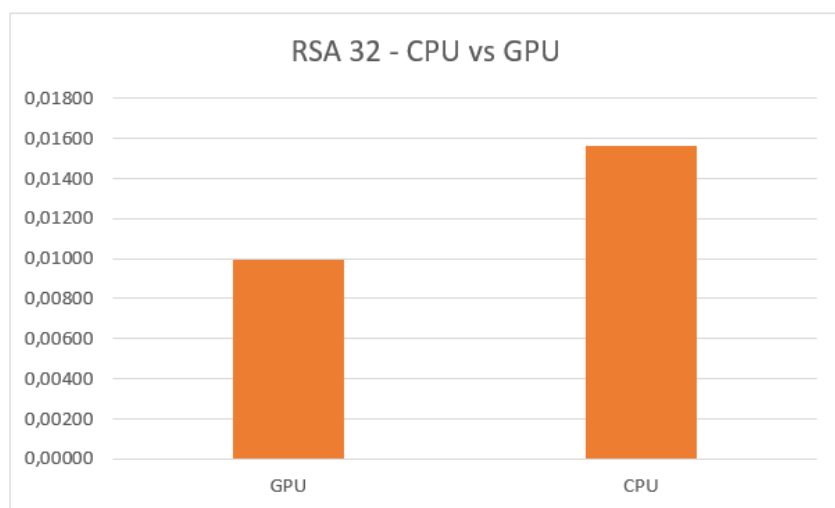


Ilustración 3 Resultados RSA 32

Lo mismo ocurre cuando usamos una clave de 63:

Modo	Tiempo
GPU	0,00900
CPU	0,01561

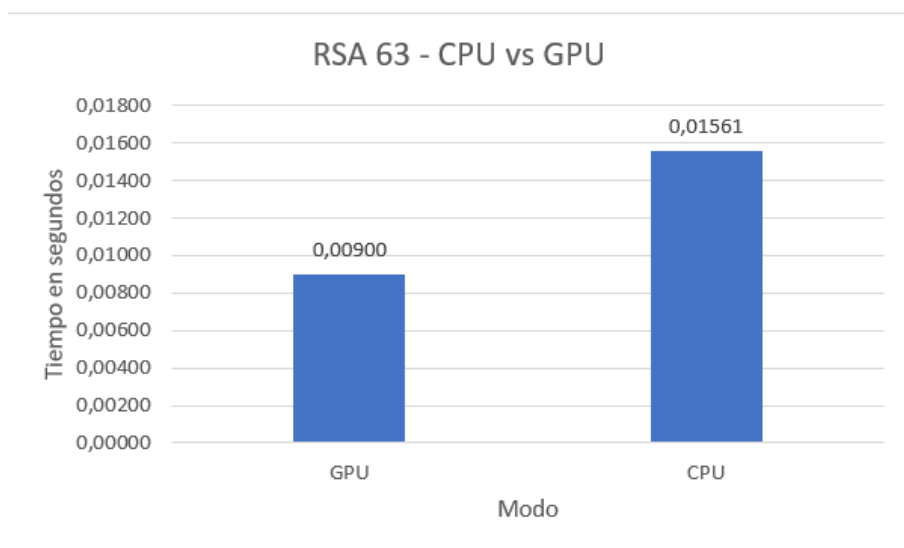


Ilustración 4 Resultados RSA 63

	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

Ahora, para realizar esta prueba modificamos el código original y agregamos un texto mucho más largo y usamos claves de mucho mayor tamaño. En este punto la GPU ya no nos puede seguir puesto que su límite de claves es de 63. Estos fueron los resultados:

Clave	Tiempo
1024	0,56222
2048	3,833908
3072	34
4096	83,11

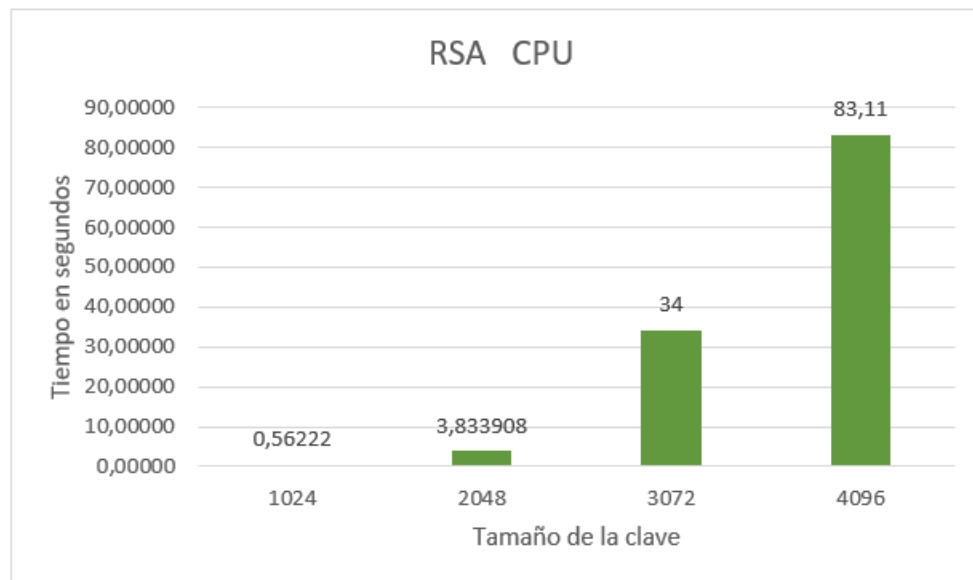


Ilustración 5 Resultados RSA Grandes Claves

Como podemos observar en la imagen 5 el tiempo de ejecución sube muy dramáticamente a medida que agregamos claves de mayor tamaño. El trabajo que quedaría por hacer sería realizar la implementación de la exponenciación, multiplicación y suma de números grandes con el fin de poder acceder a las claves de gran tamaño cuando trabajemos con la GPU.

Enlaces:

- Docker hub


<https://hub.docker.com/r/ovegero/proyecto-interciclo>

- GitHub con los Dockerfile y el docker-compose

https://github.com/OVEGERO/Proyecto_Interciclo_Docker

- GitHub del Frontend

https://github.com/OVEGERO/Proyecto_Interciclo_Frontend

	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

- GitHub del Backend

https://github.com/OVEGERO/Proyecto_Interciclo

- GitHub de la investigación

https://github.com/OVEGERO/Proyecto_Interciclo_Documentacion

RESULTADO(S) OBTENIDO(S):

- Desarrollar un entendimiento práctico de los algoritmos criptográficos y su implementación en Python.
- Comprender los fundamentos de la seguridad en la transmisión de datos y el almacenamiento seguro de información.
- Implementar y comparar el rendimiento de algoritmos de cifrado ejecutados en CPU y GPU.

CONCLUSIONES:

- El proyecto realizado nos proporcionó una comprensión profunda sobre el desarrollo e implementación en Python de los algoritmos criptográficos tanto con el algoritmo simétrico AES y el algoritmo asimétrico RSA, permitiendo realizar operaciones de cifrado y descifrado de datos.
- Se llevó a cabo una comparación exhaustiva del rendimiento entre la ejecución de algoritmos de cifrado en la CPU y la GPU, evidenciando las diferencias en velocidad y eficiencia para determinar la viabilidad de su implementación en diferentes entornos.
- La dockerización de la solución facilitó su despliegue y escalabilidad, proporcionando un entorno de ejecución consistente y la capacidad de ampliar la solución según las necesidades cambiantes del sistema.

RECOMENDACIONES:

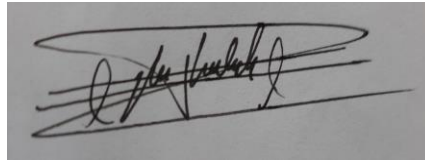
- Realizar una investigación profunda sobre los algoritmos asimétricos y simétricos con los respectivos cálculos matemáticos de los algoritmos AES y RSA.
- Revisar toda la documentación proporcionada por el docente.
- Seguir los pasos y objetivos marcados en la guía del proyecto.

	VICERRECTORADO DOCENTE	Código: GUIA-PRL-001
	CONSEJO ACADÉMICO	Aprobación: 2016/04/06
Formato: Guía de Práctica de Laboratorio / Talleres / Centros de Simulación		

Nombre de estudiante:

Jhon Llivicota

Firma de estudiante:



Nombre de estudiante:

Michael Alvarez

Firma de estudiante:

