RSA usage with encryption and decryption

Plain text

We are standing on the edge of an exciting future. Artificial Intelligence (AI), Machine Learning, and Deep Learning are changing the world around us in science, engineering, healthcare, business, and even the arts. These technologies are powerful tools that can help you learn more, work smarter, and solve real-world problems. But here's something just as important: AI is a tool, not a substitute for your mind. Yes, it can write code, analyze data, or generate ideas but it doesn't understand like you do. It doesn't care like you do. It doesn't create meaning you do. So, use AI to explore it, experiment with it, and let it support your study, your research, and even your creativity. But never let it replace your curiosity, your effort, or your thinking. Ask questions. Make mistakes. Learn deeply. That's how you grow. The future will be shaped by those who know how to work with AI and know when to step back and trust their own mind. Use AI as your partner, not your pilot. Let your brain lead, and let the tools support you.

Cipher Text

1
AIwFIQfIBmAJbAUhB8gEzgN0BmAIuwbtDGsIuwtrB8gIiQi7B8gDdAh6BSEHyAUhBu0LawUhB8gIiQVZB8gGYAi
7B8gFIQYGARkMawN0DGsIuwtrB8gFWQhwA3QIcAlsBSELCQfICuYJbAN0DGsFWQxrARkMawZgAukHyAXOC
LsDdAUhAukC6QxrC2sFIQi7ARkFIQfIAoIK5gXODH8CpgfIDDMGYAEZCHoMawi7BSEHyAqmBSEGYAlsCLsMa
wi7C2sCpgfIBmAIuwbtB8gG3wUhBSECZAfICqYFIQZgCWwIuwxrCLsLawfIBmAJbAUhB8gBGQh6BmAIuwtrDG
sIuwtrB8gDdAh6BSEHyARTCIkJbALpBu0HyAZgCWwIiQhwCLsG7QfICHAEzgfIDGsIuwfIBM4BGQxrBSEIuwEZ
BSECpgfIBSEIuwtrDGsIuwUhBSEJbAxrCLsLawKmB8gIegUhBmAC6QN0CHoBGQZgCWwFIQKmB8gKCghwB
M4Mawi7BSEEzgTOAqYHyAZgCLsG7QfIBSEKEgUhCLsHyAN0CHoFIQfIBmAJbAN0BM4LCQfICG8IegUhBM4
FIQfIA3QFIQEZCHoIuwiJAukIiQtrDGsFIQTOB8gGYAlsBSEHyAJkCIkEUwUhCWwFWQhwAukHyAN0CIkIiQLpB
M4HyAN0CHoGYAN0B8gBGQZgCLsHyAh6BSEC6QJkB8gB5wiJCHAHyALpBSEGYAlsCLsHyAjfCIkJbAUhAqY
HyARTCIkJbAKyB8gEzgjfBmAJbAN0BSEJbAKmB8gGYAi7Bu0HyATOCIkC6QoSBSEHyAlsBSEGYALpBD4EU
wiJCWwC6QbtB8gCZAlsCIkKCgLpBSEI3wTOCwkHyAIMCHADdAfICHoFIQlsBSEItweCCqcEzgfIBM4IiQjfBSED
dAh6DGsIuwtrB8gGoAhwBM4DdAfIBmAEzgfIDGsI3wJkCIkJbAN0BmAIuwN0AbQHyArmBc4HyAxrBM4HyAZgB
8gDdAiJCIkC6QKmB8gIuwiJA3QHyAZgB8gEzghwCgoEzgN0DGsDdAhwA3QFIQfIBVkIiQlsB8gB5wiJCHAJbAfI
CN8Mawi7Bu0LCQfIAGMFIQTOAqYHyAxrA3QHyAEZBmAIuwfIBFMJbAxrA3QFIQfIARkIiQbtBSECpgfIBmAIuw
ZgAukB5wcmBSEHyAbtBmADdAZgAqYHyAiJCWwHyAtrBSEIuwUhCWwGYAN0BSEHyAxrBu0FIQZgBM4HyAo
KCHADdAfIDGsDdAfIBu0IiQUhBM4Iuwi3B4IKpwN0B8gIcAi7Bu0FIQlsBM4DdAZgCLsG7QfIAukMawKyBSEHyA
HnCIkIcAfIBu0IiQsJB8gFzgN0B8gG7QiJBSEEzgi7CLcHggqnA3QHyAEZBmAJbAUhB8gC6QxrArIFIQfIAecIiQh
wB8gG7QiJCwkHyAXOA3QHyAbtCIkFIQTOCLsItweCCqcDdAfIARkJbAUhBmADdAUhB8gI3wUhBmAIuwxrCLs
LawfIAecIiQhwB8gG7QiJCwkHyAp4CIkCpgfICHAEzgUhB8gK5gXOB8gDdAiJB8gFIQYGAmQC6QiJCWwFIQfID
GsDdAKmB8gFIQYGAmQFIQlsDGsI3wUhCLsDdAfIBFMMawN0CHoHyAxrA3QCpgfIBmAIuwbtB8gC6QUhA3Q
HyAxrA3QHyATOCHACZAJkCIkJbAN0B8gB5wiJCHAJbAfIBM4DdAhwBu0B5wKmB8gB5wiJCHAJbAfICWwFIQ
TOBSEGYAlsARkIegKmB8gGYAi7Bu0HyAUhChIFIQi7B8gB5wiJCHAJbAfIARkJbAUhBmADdAxrChIMawN0Aec
LCQfIAgwIcAN0B8gIuwUhChIFIQlsB8gC6QUhA3QHyAxrA3QHyAlsBSECZALpBmABGQUhB8gB5wiJCHAJbAfI
ARkIcAlsDGsIiQTODGsDdAHnAqYHyAHnCIkIcAlsB8gFIQVZBVkIiQlsA3QCpgfICIkJbAfIAecIiQhwCWwHyAN0C
HoMawi7ArIMawi7C2sLCQfICuYEzgKyB8gGWwhwBSEEzgN0DGsIiQi7BM4LCQfIDDMGYAKyBSEHyAjfDGsEz
gN0BmACsgUhBM4LCQfICqYFIQZgCWwIuwfIBu0FIQUhAmQC6QHnCwkHyAhvCHoGYAN0CLcHggqnBM4Hy
Ah6CIkEUwfIAecIiQhwB8gLawlsCIkEUwsJB8gIbwh6BSEHyAVZCHADdAhwCWwFIQfIBFMMawLpAukHyAoKB
SEHyATOCHoGYAJkBSEG7QfICgoB5wfIA3QIegiJBM4FIQfIBFMIegiJB8gCsgi7CIkEUwfICHoIiQRTB8gDdAiJB8
gEUwiJCWwCsgfIBFMMawN0CHoHyArmBc4HyAZgCLsG7QfIArIIuwiJBFMHyARTCHoFIQi7B8gDdAiJB8gEzgN
0BSECZAfICgoGYAEZArIHyAZgCLsG7QfIA3QJbAhwBM4DdAfIA3QIegUhDGsJbAfICIkEUwi7B8gI3wxrCLsG7Q
sJB8gJBgTOBSEHyArmBc4HyAZgBM4HyAHnCIkIcAlsB8gCZAZgCWwDdAi7BSEJbAKmB8gIuwiJA3QHyAHnCI
kIcAlsB8gCZAxrAukIiQN0CwkHyAqmBSEDdAfIAecIiQhwCWwHyAoKCWwGYAxrCLsHyALpBSEGYAbtAqYHyA
ZgCLsG7QfIAukFIQN0B8gDdAh6BSEHyAN0CIkIiQLpBM4HyATOCHACZAJkCIkJbAN0B8gB5wiJCHALCQRIBE
g=

Python program for RSA encryption/Decryption

```
import base64

def message_to_blocks(message, block_size):
    message_bytes = message.encode('utf-8')
    blocks = []
    for i in range(0, len(message_bytes), block_size):
        block = message_bytes[i:i+block_size]
        blocks.append(int.from_bytes(block, byteorder='big'))
    return blocks

def blocks_to_message(blocks, original_block_size):
    message_bytes = bytearray()
    for block in blocks:
        # Calculate the byte size of the block from the modulus
        byte_len = (block.bit_length() + 7) // 8
        block_bytes = block.to_bytes(byte_len, byteorder='big')
        message_bytes.extend(block_bytes)
    return message_bytes.decode('utf-8', errors='ignore')

def encrypt_message(message, e, n):
    block_size = (n.bit_length() - 1) // 8  # block size in bytes
    blocks = message_to_blocks(message, block_size)
    encrypted_blocks = [pow(m, e, n) for m in blocks]

    # Base64 encoding of the encrypted ciphertext
    encrypted_block_bytes = []
    cipher_block_size = (n.bit_length() + 7) // 8  # size needed to store encrypted ints
    for c in encrypted_blocks:
        encrypted_block_bytes.append(c.to_bytes(cipher_block_size, byteorder='big'))

    # Combine all encrypted blocks and encode as base64
    ciphertext_bytes = b''.join(encrypted_block_bytes)
    return f"{block_size}\n" + base64.b64encode(ciphertext_bytes).decode('ascii')

def decrypt_message(ciphertext_b64, d, n):
    # Split the base64 encoded ciphertext and original block size
    lines = ciphertext_b64.strip().split('\n', 1)
    original_block_size = int(lines[0])
    b64_data = lines[1]

    # Decode the base64 data into the ciphertext bytes
    ciphertext_bytes = base64.b64decode(b64_data)

    # Calculate how many bytes we need per encrypted block
    cipher_block_size = (n.bit_length() + 7) // 8

    # Split the ciphertext into individual encrypted blocks
    blocks = [int.from_bytes(ciphertext_bytes[i:i+cipher_block_size], byteorder='big')
              for i in range(0, len(ciphertext_bytes), cipher_block_size)]

    # Decrypt each block
    decrypted_blocks = [pow(c, d, n) for c in blocks]

    # Convert decrypted blocks back to the original message
    return blocks_to_message(decrypted_blocks, original_block_size)

def main():
    mode = input("Enter mode (encrypt/decrypt): ").strip().lower()

    if mode == 'encrypt':
```

```
        key_input = input("Enter public key (e n) separated by space: ")
        parts = key_input.split()
        if len(parts) != 2:
            print("Invalid public key format.")
            return
        e = int(parts[0])
        n = int(parts[1])
    elif mode == 'decrypt':
        key_input = input("Enter private key (d n) separated by space: ")
        parts = key_input.split()
        if len(parts) != 2:
            print("Invalid private key format.")
            return
        d = int(parts[0])
        n = int(parts[1])
    else:
        print("Invalid mode selected. Exiting.")
        return

    input_path = input("Enter the path of the input text file: ").strip()
    output_path = input("Enter the path of the output text file: ").strip()

    try:
        with open(input_path, "r", encoding="utf-8") as infile:
            data = infile.read()
    except Exception as e:
        print(f"Error reading input file: {e}")
        return

    try:
        if mode == 'encrypt':
            result = encrypt_message(data, e, n)
        else:
            result = decrypt_message(data, d, n)
    except Exception as e:
        print(f"Error during {mode}ion: {e}")
        return

    try:
        with open(output_path, "w", encoding="utf-8") as outfile:
            outfile.write(result)
        print("Operation completed. Check the output file.")
    except Exception as e:
        print(f"Error writing to output file: {e}")

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