F20CN -Computer Network Security

COURSEWORK-2

Group-81

Abdul Rauf Phansupkar & Prudhvi Varma

H00373954 & H00373952

TASK 1- Alternative Method of Public-Key Encryption

Code:

```
import random
import math
import json


def is_prime(n):
    """
    Check if a number is a prime number.

    Args:
        n (int): The number to check.

    Returns:
        bool: True if the number is prime, False otherwise.
    """
#Reference: https://www.pythonpool.com/check-if-number-is-prime-in-python/
    if n <= 1:
        return False
    for i in range(2, int(math.sqrt(n)) + 1):
        if n % i == 0:
            return False
    return True


def find_prime_above(min_value):
    """
    Tries to find the next prime number >= min_value.

    Args:
        min_value (int): The minimum value to start searching for a prime number.

    Returns:
        int: The smallest prime number greater than or equal to min_value.
    """
    candidate = max(min_value, 2)
    while not is_prime(candidate):
```

```
        candidate += 1
    return candidate


def generate_keys():
    """
    Generate and save public and private keys with length 2-64.
    List `e` is sums of previous elements plus random (1-100).
    Prime `q` > 2 * last `e`, `w` coprime with `q`.
    Public key: `(w * ei) % q` for each `ei` in `e`.
    Saves to 'public_key.txt' and 'private_key.txt'.
    """
    while True:
        n = input("Choose e(n) (between 2 and 64): ").strip()
        if n.isdigit() and 2 <= int(n) <= 64:
            n = int(n)
            break
        print("Invalid input. Please enter a number between 2 and 64.")

    e = []
    sum_prev = 0
    for _ in range(n):
        ei = sum_prev + random.randint(1, 100)  # limited random generation between 1-100 due to computational reasons
        e.append(ei)
        sum_prev = sum(e)

    q = find_prime_above(e[-1] * 2)         # q prime number greater than twice the last element in the e list
    w = random.randint(2, q - 1)
    while math.gcd(w, q) != 1:              # w number between 2 and q-1, where q and w are coprime
        w = random.randint(2, q - 1)

    h = [(w * ei) % q for ei in e]          # computing the public key

    public_key = h
    private_key = (e, q, w)

    with open("public_key.txt", "w") as f:
```

```
        f.write('\n'.join(map(str, public_key)))  # Saving keys to files

    with open("private_key.txt", "w") as f:
        f.write(str(q) + '\n')
        f.write(str(w) + '\n')
        f.write('\n'.join(map(str, e)))

    print(f"Keys generated with length {n} and saved to files 'public_key.txt' and 'private_key.txt'.")

def load_public_key(file_name):
    with open(file_name, "r") as f:
        return [int(line.strip()) for line in f if line.strip()]

def load_private_key(file_name):
    with open(file_name, "r") as f:
        lines = f.read().splitlines()
    q = int(lines[0])
    w = int(lines[1])
    e = [int(x) for x in lines[2:]]
    return (e, q, w)

def chunk_message(message_bits, chunk_size):
    for i in range(0, len(message_bits), chunk_size):
        yield message_bits[i:i + chunk_size]                    # Trying to yield chunks of the specified size from message bits.



def pad_chunk(chunk, chunk_size):
     return chunk + [0] * (chunk_size - len(chunk))                 # Trying to pad the chunk with zeros to match the chunk size


def encrypt_chunk(chunk, public_key):
    return sum(public_key[j] * chunk[j] for j in range(len(chunk)))      # Trying to encrypt the chunk using the public key

def encrypt(message_bits, public_key):
    chunk_size = len(public_key)                                # Trying to set chunk size to the length of the public key
```

```
    ciphertext_chunks = []

    for chunk in chunk_message(message_bits, chunk_size):
        if len(chunk) < chunk_size:                    # Trying to pad the chunk if it's smaller than the chunk size
            chunk = pad_chunk(chunk, chunk_size)
        ciphertext_chunks.append(encrypt_chunk(chunk, public_key))        # Trying to encrypt the chunk using the public key

    return ciphertext_chunks

def decrypt_chunk(c_prime, e):
    chunk_bits = []                # Trying to convert the chunk back to bits
    for ei in reversed(e):
        if c_prime >= ei:
            chunk_bits.insert(0, 1)
            c_prime -= ei
        else:
            chunk_bits.insert(0, 0)
    return chunk_bits

def decrypt(ciphertext_chunks, private_key):
    e, q, w = private_key
    w_inv = pow(w, -1, q)  # Trying to find the modular inverse of w
    message_bits = []

    for c in ciphertext_chunks:
        c_prime = (c * w_inv) % q  # Trying to decrypt the chunk
        message_bits.extend(decrypt_chunk(c_prime, e))  # Trying to convert the decrypted chunk back to bits

    return message_bits

def text_to_bits(text):
    return [int(bit) for byte in bytearray(text, 'utf-8') for bit in format(byte, '08b')]
                                                    #converting text to bits and vice versa
def bits_to_text(bits):
    byte_chunks = [''.join(map(str, bits[i:i + 8])) for i in range(0, len(bits), 8)]
    bytes_array = bytearray([int(b, 2) for b in byte_chunks])
```

```
    return bytes(bytes_array).decode('utf-8', errors='ignore')

def main():
    action = input("Choose action (generate/encrypt/decrypt): ").strip().lower()

    if action == "generate":
        generate_keys()

    elif action == "encrypt":
        input_file = input("Enter the plaintext file name: ").strip()
        public_key_file = input("Enter the public key file name: ").strip()

        try:
            public_key = load_public_key(public_key_file)
        except FileNotFoundError:
            print(f"Error: Public key file '{public_key_file}' not found.")
            return
        except IOError as e:
            print(f"Error: Unable to read public key file '{public_key_file}'. Reason: {e}")
            return

        try:
            with open(input_file, 'r') as f:
                plaintext = f.read()                    # Reading plaintext and convert to binary
        except FileNotFoundError:
            print(f"Error: Plaintext file '{input_file}' not found.")
            return
        except IOError as e:
            print(f"Error: Unable to read plaintext file '{input_file}'. Reason: {e}")
            return

        message_bits = text_to_bits(plaintext)

        ciphertext_chunks = encrypt(message_bits, public_key)

        output_file = input("Enter the output file name for ciphertext: ").strip()
```

```
    try:
        with open(output_file, 'w') as f:                    # Save ciphertext chunks to file
            json.dump(ciphertext_chunks, f)
        print(f"Encryption complete. Ciphertext saved to {output_file}")
    except IOError as e:
        print(f"Error: Unable to write ciphertext to file '{output_file}'. Reason: {e}")

elif action == "decrypt":
    input_file = input("Enter the ciphertext file name: ").strip()
    private_key_file = input("Enter the private key file name: ").strip()

    try:
        private_key = load_private_key(private_key_file)
    except FileNotFoundError:
        print(f"Error: Private key file '{private_key_file}' not found.")
        return
    except IOError as e:
        print(f"Error: Unable to read private key file '{private_key_file}'.")
        return

    try:
        with open(input_file, 'r') as f:
            ciphertext_chunks = json.load(f)  # Read ciphertext chunks from file
    except FileNotFoundError:
        print(f"Error: Ciphertext file '{input_file}' not found.")
        return
    except IOError as e:
        print(f"Error: Unable to read ciphertext file '{input_file}'.")
        return

    decrypted_bits = decrypt(ciphertext_chunks, private_key)
    decrypted_text = bits_to_text(decrypted_bits)

    output_file = input("Enter the output file name for decrypted text: ").strip()
    try:
        with open(output_file, 'w') as f:  # Save decrypted text to file
```

```
            f.write(decrypted_text)
        print(f"Decryption complete. Decrypted text saved to {output_file}")
    except IOError as e:
        print(f"Error: Unable to write decrypted text to file '{output_file}'.")

else:
    print("Invalid action. Please choose 'generate', 'encrypt', or 'decrypt'.")

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

TASK 1- Alternative Method of Public-Key Encryption

I started writing this program by firstly dividing the program into 3 key parts: key generation, encryption, and decryptionI began with the key generation process, where I created a super-increasing sequence (e) and computed the necessary values for the public and private keys. Next, I moved on to the encryption process, converting plaintext into binary bits and encrypting it using the public key. Lastly, I developed the decryption process, which involves computing the modular inverse and reconstructing the original message from the ciphertext. Throughout the development, I included helper functions for tasks like prime number checking and text-to-bits conversion. I chose to divide the message into blocks equal to the length of the public key for encryption, ensuring each chunk matches the key's size.

PseudoCode & Approach:

The process of developing the program for task1 was divided into 4 parts, Generation, Encryption, Decryption and the Main Program logic, any additional helper functions which have been used in the code are described in comments

1.Keygeneration:

Input: Key Length (Parameters given between 2 to 64) 64 limit given due to computation reasons
Output: Public Key and Private Key
Steps:

1. Initialize E=[] and Sum_prev=0 //store the previous sums
2. Generate list e:
   For I to in range n:
   Generate random number ei = ei>Sum_prev
   Append (ei) to E and update Sum_prev
3. Generate q = q>2*en (last number of list E)
4. Generate random w such that gcd(w,q)=1
5. Generate public key h: hi=(w*ei) %q for each ei in e
6. Return h=[Public Key] and (e,q,w)=[Private Key] and store them as files in same directory

2.Encryption

Input: Message, Public Key h
Output: Encrypted text C
Steps:

1. Convert message to bits
2. Message_bits are converted chunks of size h, (padding is also applied to round out bits)
3. For each chunk:
   C = sum(h[i] * chunk[i] for i in range(len(chunk)))
   Append c to ciphertext list
4. Return complete ciphertext

3.Decryption

Input: Encrypted text C, Private Key (e,q,w)
Output: Decrypted message bits
Steps:

1. Compute winverse = modular_inverse(w,q)
2. For each I in ciphertext:
   Compute c'=(c * w_inv) %q
   For each ei in reversed e:
   If c' >= ei: set bit to 1 and ei is subtracted from c'
   Else set bit to 0
3. Convert bits to text and return plaintext.

4.Main Method

Input: User action choice (generate, encryption and decryption)
Steps:

1. If generate: call keygeneration() logic
2. If encryption: call encryption() logic

3. If decryption: call decryption() logic

Testing Results:

Using Key length- 5

```
PS C:\Users\prudh\desktop> python Task1-Encryption.py
Choose action (generate/encrypt/decrypt): generate
Choose e(n) (between 2 and 64): 5
Keys generated with length 5 and saved to files 'public_key.txt' and 'private_key.txt'.
```

```
PS C:\Users\prudh\desktop> python Task1-Encryption.py
Choose action (generate/encrypt/decrypt): generate
Choose e(n) (between 2 and 64): 7
Keys generated with length 7 and saved to files 'public_key.txt' and 'private_key.txt'.
```

Generating key pairs of Key Length 5 and Key Length 7

```
C:\Users\prudh>python3 Task1-Encryption.py
Choose action (generate/encrypt/decrypt): encrypt
Enter the plaintext file name: message.txt
Enter the public key file name: public_key.txt
5
Enter the output file name for ciphertext: ciphertext.txt
Encryption complete. Ciphertext saved to ciphertext.txt
```

Encryption of plaintext with public key of length 5

message.txt ciphertext.txt

File Edit View

[60, 313, 500, 201, 0, 292, 208, 201, 199, 452, 201, 187, 153, 464, 361, 172, 12, 299, 313, 187, 12, 464, 220, 151, 347, 12, 153, 299, 0, 373, 208, 160, 311, 325, 500, 187, 201, 373, 361, 347, 139, 160, 292, 500, 187, 12, 208, 220, 151, 452, 304, 201, 0, 464, 208, 359, 139, 160, 153, 340, 187, 213, 208, 172, 151, 452, 373, 187, 201, 373, 361, 347, 139, 160, 153, 340, 304, 352, 172, 0, 151, 464, 500, 340, 48, 373, 160, 151, 139, 160, 373, 340, 153, 325, 220, 292, 311, 160, 201, 500, 139, 12, 208, 153, 151, 452, 325, 347, 187, 373, 373, 199, 151, 299, 340, 48, 0, 373, 373, 201, 139, 160, 165, 347, 139, 373, 373, 187, 151, 452, 325, 500, 187, 213, 220, 359, 311, 325, 153, 340, 165, 220, 313, 0, 12, 172, 325, 187, 60, 165, 220, 220, 311, 299, 500, 187, 199, 12, 208, 139, 311, 464, 373, 340, 304, 12, 208, 160, 139, 160, 340, 187, 48, 304, 361, 347, 151, 292, 153, 187, 199, 373, 373, 160, 311, 153, 153, 187, 201, 373, 361, 292, 311, 304, 153, 347, 187, 373, 220, 160, 311, 313, 340, 187, 60, 304, 313, 151, 139, 160, 340, 340, 48, 373, 361, 160, 151, 464, 201, 340, 48, 165, 361, 151, 151, 292, 153, 347, 12, 165, 220, 361, 311, 311, 325, 201, 0, 165, 220, 151, 311, 313, 500, 500, 199, 304, 325, 0, 151, 325, 500, 500, 139, 12, 208, 172, 311, 325, 313, 500, 139, 325, 220, 311, 151, 452, 500, 347, 12, 165, 220, 151, 139, 160, 165, 347, 139, 373, 373, 347, 199, 172, 201, 500, 187, 304, 325, 151, 139, 160, 452, 340, 60, 325, 220, 153, 311, 299, 500, 187, 199, 12, 208, 311, 311, 464, 213, 187, 201, 464, 361, 160, 199, 313, 292, 340, 352, 373, 313, 0, 311, 160, 292, 340, 199, 213, 160, 0, 151, 299, 500, 187, 12, 12, 208, 201, 151, 452, 201, 347, 60, 165, 373, 172, 311, 325, 361, 201, 0, 373, 373, 340, 151, 452, 325, 347, 199, 213, 208, 299, 311, 313, 340, 201, 304, 12, 48, 361, 199, 313, 153, 500, 201, 213, 220, 199, 151, 304, 153, 187, 352, 304, 361, 347, 151, 299, 500, 187, 201, 352, 361, 299, 151, 153, 153, 347, 199, 213, 220, 292, 311, 0, 153, 347, 12, 373, 373, 359, 311, 313, 325, 201, 0, 373, 208, 172, 311, 311, 201, 201, 0, 304, 208, 151, 151, 299, 500, 187, 304, 325, 220, 201, 199, 151, 153, 187, 352, 304, 313, 0, 151, 313, 292, 340, 199, 213, 220, 292, 151, 299, 340, 48, 0, 165, 220, 153, 199, 160, 325, 201, 0, 165, 373, 160, 311, 165, 153, 347, 187, 464, 208, 201, 151, 452, 153, 340, 213, 373, 208, 172, 311, 325, 201, 201, 0, 464, 208, 12, 151, 292, 153, 347, 0, 304, 361, 359, 151, 311, 201, 500, 187, 304, 325, 151, 139, 160, 292, 187, 60, 373, 208, 153, 311, 299, 500, 187, 199, 12, 208, 325, 311, 464, 213, 201, 0, 464, 208, 201, 151, 299, 313, 340, 340, 12, 208, 160, 311, 325, 201, 48, 0, 165, 220, 139, 151, 299, 165, 347, 12, 220, 313, 0, 12, 311, 500, 340, 304, 304, 373, 172, 199, 311, 213, 187, 60, 373, 361, 208, 359, 139, 153, 187, 201, 304, 325, 0, 151, 452, 325, 500, 187, 325, 220, 199, 199, 313, 292, 340, 48, 373, 160, 373, 139, 160, 201, 500, 151, 325, 220, 199, 139, 160, 325, 340, 213, 165, 220, 151, 311, 153, 340, 48, 0, 325, 208, 160, 311, 299, 340, 347, 201, 12, 208, 160, 151, 311, 213, 187, 201, 373, 373, 199, 199, 151, 153, 187, 187, 373, 373, 311, 199, 160, 500, 500, 60, 373, 361, 139, 139, 160, 500, 500, 151, 325, 220, 201, 139, 160, 213, 187, 201, 373, 220, 299, 299, 165, 153, 139, 139, 352, 172, 0, 199, 172, 201, 500, 151, 213, 220, 299, 199, 464, 292, 340, 304, 325, 325, 0, 359, 299, 500, 500, 60, 304, 313, 0, 199, 160, 325, 187, 352, 325, 373, 201, 151, 452, 325, 500, 187, 12, 208, 201, 151, 452, 361, 500, 60, 373, 208, 160, 199, 172, 340, 347, 201, 220, 160, 0, 359, 299, 500, 500, 60, 12, 208, 208, 151, 299, 500, 48, 0, 165, 373, 299, 311, 313, 201, 340, 139, 304, 361, 160, 199, 313, 201, 201, 0, 373, 220, 172, 311, 313, 201, 500, 187, 464, 208, 359, 311, 325, 201, 500, 187, 220, 160, 0, 151, 299, 201, 187, 292, 464, 220, 361, 199, 153, 153, 347, 187, 464, 208, 201, 151, 299, 213, 187, 60, 325, 373, 172, 151, 452, 325, 201, 165, 12, 208, 160, 311, 325, 201, 48, 0, 304, 361, 299, 311, 452, 153, 340, 201, 373, 313, 0, 151, 325, 500, 340, 187, 464, 220, 361, 151, 452, 201, 48, 304]

Ciphertext of the encrypted file with a key length 5

```
C:\Users\prudh>python3 Task1-Encryption.py
Choose action (generate/encrypt/decrypt): decrypt
Enter the ciphertext file name: ciphertext.txt
Enter the private key file name: private_key.txt
Enter the output file name for decrypted text: decrypted.txt
Decryption complete. Decrypted text saved to decrypted.txt
```

Decryption of ciphertext with private key of length 5

decrypted.txt

File Edit View

To PrudhviAbdul: Planning is key to achieving any goal, whether personal or professional. Breaking down a large goal into smaller, manageable tasks allows for incremental progress, keeping motivation high and reducing overwhelm. Staying organized with tools like planners or digital apps can streamline the process, helping you track and adapt. Consistency is essential; even small, daily actions compound over time. By reviewing your progress regularly, you can celebrate milestones, adjust strategies, and remain focused.

Decrypted Text

TASK 2- Firewall Rules Application

Code:

```
import ipaddress
import os

rules = []

# Rule addition method which delegates to the appropriate method based on the direction
def add_rule(ruleno, direction, addr):
  if '-' in addr:
    iprange_insert(ruleno, direction, addr)
  else:
    if direction is None:
      # Add for both incoming and outgoing directions with the same rule number when no direction is specified
      bidirectional_insert(ruleno, addr)
    else:
      # Add for the specified direction only
      single_insert(ruleno, direction, addr)

# In case of Ip range input, the ranges are split and added as individual ip address rules
def iprange_insert(ruleno, direction, addr):
  iprange_start, iprange_end = addr.split('-')
  iprange_start = ipaddress.IPv4Address(iprange_start)
  iprange_end = ipaddress.IPv4Address(iprange_end)

  # Iterate through the range and add each IP address, reversed is used to ensure that starting ip gets highest priority
  for ip in reversed(range(int(iprange_start), int(iprange_end) + 1)):
    ip_str = str(ipaddress.IPv4Address(ip))  #Individual Ip address are converted to string to be passed to the insert method
    if direction is None:
      bidirectional_insert(ruleno, ip_str)
    else:
      single_insert(ruleno, direction, ip_str)

 # Helper function to find matching rules, can filter based on a combination of rule number, direction, and address
def find_rules(ruleno=None, direction=None, addr=None):
  return [
```

```
        rule for rule in rules  #Check for a rule matching in the rules list
        if (ruleno is None or rule['rule'] == ruleno) and
          (direction is None or rule['direction'] == direction) and
          (addr is None or rule['addr'] == addr)
    ]

# Adding rules for both directions
def bidirectional_insert(ruleno, addr):
    single_insert(ruleno, '-in', addr)
    single_insert(ruleno, '-out', addr)

# Add a single rule for outgoing or incoming direction
def single_insert(ruleno, direction, addr):
    new_rule = {'rule': ruleno, 'direction': direction, 'addr': addr}

    if find_rules(ruleno, direction, addr):  # Check if exactly same rule exists preventing duplicates
        print(f"Error: Rule {ruleno} with direction {direction} and address {addr} already exists.")
        return

    if find_rules(ruleno, direction):  # Check if rule with same number and direction exists then adjust priorities
        adjustpriorities(ruleno)

    rules.append(new_rule)
    rules.sort(key=lambda x: (x['rule'], x['direction']))  # Sort rules based on rule number first and then direction

def adjustpriorities(ruleno):
    for rule in rules:
        if rule['rule'] >= ruleno:
            rule['rule'] += 1  # Rule number is incremented for matching existing rule numbers already in the list

def remove_rule(ruleno, direction=None):
    global rules   # Accessing the global rules list
    rulefound = False  # Flag to check if the rule is found

    # Step 1: Find matching rules based on the input criteria
    if direction is None:
```

```
        rules_for_removal = find_rules(ruleno=ruleno)  # Remove all rules with the specified rule number
    else:
        rules_for_removal = find_rules(ruleno=ruleno, direction=direction)  # Remove only the rule with the specified direction

    if rules_for_removal: #once rules for removal are found they must be removd from the main list
        rules = [rule for rule in rules if rule not in rules_for_removal] #Removing the matching rules from the main list
        rulefound = True

    if not rulefound:
        print(f"Error: Rule {ruleno} not found. Please enter a valid command.")

# Listing firewall rules based on the criteria, each filter method redirects to dedicated helper method
def list_rules(ruleno=None, direction=None, addr=None):
    filtered_rules = rules
    if ruleno:
        filtered_rules = filter_byruleno(filtered_rules, ruleno)
    if direction:
        filtered_rules = filter_bydirection(filtered_rules, direction)
    if addr:
        filtered_rules = filter_byaddr(filtered_rules, addr)
    for rule in filtered_rules:
        print(f"Rule {rule['rule']} | Direction: {rule['direction']} | Address: {rule['addr']}")

# List of rules given based on given ruleno
def filter_byruleno(rules_list, ruleno):
    return [rule for rule in rules_list if rule['rule'] == ruleno]

# List of rules given based on given direction
def filter_bydirection(rules_list, direction):
    return [rule for rule in rules_list if rule['direction'] == direction]

# List of rules given based on IP address or address range
def filter_byaddr(rules_list, addr):
    if '-' in addr:
        # If the address is a range, filter based on the range
        iprange_start, iprange_end = addr.split('-')
```

```
        iprange_start = ipaddress.IPv4Address(iprange_start)
        iprange_end = ipaddress.IPv4Address(iprange_end)
        filtered_rules = [rule for rule in rules_list if iprange_start <= ipaddress.IPv4Address(rule['addr']) <= iprange_end]
    else:
        # If the address is a single IP, filter based on the exact match
        filtered_rules = [rule for rule in rules_list if rule['addr'] == addr]
    return filtered_rules

# Save rules to a file (called by the save command during runtime)
def savefile(filename):
    try:
        with open(filename, 'w') as file:
            for rule in rules:
                file.write(f"{rule['rule']} {rule['direction']} {rule['addr']}\n")
        print(f"Rules successfully saved to {filename}.")
    except Exception as e:
        print(f"Error: Unable to save firewall rules to file. {e}")

# Load rules from a file (used by the loadfile_runtime method)
def loadfile(filename):
    global rules
    if os.path.exists(filename):
        try:
            with open(filename, 'r') as file:
                rules.clear()  # For simplicity, we are considering to clear the existing rules before loading new ones
                for line in file:
                    rule_sections = line.strip().split()
                    if len(rule_sections) == 3:
                        ruleno = int(rule_sections[0])
                        direction = rule_sections[1]
                        addr = rule_sections[2]
                        rules.append({'rule': ruleno, 'direction': direction, 'addr': addr})
                rules.sort(key=lambda x: (x['rule'], x['direction']))
            print(f"Rules successfully loaded from {filename}.")
        except Exception as e:
            print(f"Error: Unable to load firewall rules from file. {e}")
```

```
    else:
        print(f"Error: File {filename} cannot be found. Please enter the filename again or type 'exit' to cancel.")
        return False
    return True

# Load rules from file during runtime (called by the load command)
def loadfile_runtime():
    while True:
        filename = input("Enter the filename to load rules from: ").strip()
        if filename.lower() == 'cancel':
            return
        if loadfile(filename):
            break

# Input handler for add command, requires at least 2 parameters add command and ipaddress is mandatory
def add_handler(command):
    if len(command) < 2:
        print("Error: Insufficient parameters for add command.")
        return

    ruleno = 1
    direction = None
    addr = None

    #Extracting parameters from the input command assigning them to ruleno, direction and addr
    for param in command[1:]:
        if param.isdigit():
            ruleno = int(param)
        elif param in ['-in', '-out']:
            direction = param
        elif validate_ip(param):
            addr = param

    if not addr:
        print("Error: Invalid or missing IP address.")
        return
```

```
    add_rule(ruleno, direction, addr)

# Input handler for remove command, requires at least 2 parameters, rule number is mandatory
def remove_handler(command):
    if len(command) < 2:
        print("Error: Rule number is required for remove command.")
        return

    #Only rule number and direction parameters are needed for removing rules
    ruleno = int(command[1])
    direction = command[2] if len(command) > 2 and command[2] in ['-in', '-out'] else None
    remove_rule(ruleno, direction)

# Input handler for list command, does not require any parameters but can work with specific parameters as well
def list_handler(command):
    ruleno = None
    direction = None
    addr = None

    for param in command[1:]:
        if param.isdigit():
            ruleno = int(param)
        elif param in ['-in', '-out']:
            direction = param
        elif validate_ip(param):
            addr = param

    list_rules(ruleno, direction, addr)

# Method to validate IP address or IP range using the IPaddress module in python to check if the input is valid Ipv4 address
def validate_ip(addr):
    try:
        if '-' in addr:
            iprange_start, iprange_end = addr.split('-')
            ipaddress.IPv4Address(iprange_start)
```

```
            ipaddress.IPv4Address(iprange_end)
        else:
            ipaddress.IPv4Address(addr)
        return True
    except ValueError:
        return False


def main():
    print("Available commands:\n"
          "  add [ruleno] [-in|-out] IP_address - Add a rule\n"
          "  remove ruleno [-in|-out] - Remove a rule\n"
          "  list [ruleno] [-in|-out] [IP_address|-IP_range] - List rules\n"
          "  load - Load rules from a file\n"
          "  save - Save rules to a file\n"
          "  exit - Exit the program")
    while True:
        command = input("Enter command: ").strip().split()
        if not command:
            continue

        action = command[0]  # Extract the action from the command as it is the first word of the input
        # Actions are handled by the respective handler methods
        if action == 'add':
            add_handler(command)

        elif action == 'remove':
            remove_handler(command)

        elif action == 'list':
            list_handler(command)

        elif action == 'load':
            loadfile_runtime()

        elif action == 'save':
            filename = input("Enter the filename to save rules to: ").strip()
```

```
            savefile(filename)

        elif action == 'exit':
            break

        else:
            print("Error: Unknown command.")

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

I structured this program into three key parts: rule addition, rule removal, and rule listing, with additional functionality for saving and loading rules. For rule addition, I implemented logic to handle single IPs, ranges, and bidirectional rules, ensuring proper prioritization and conflict resolution. Rule removal supports specific rules by number and direction, with errors flagged for non-existent entries. Rule listing allows flexible filtering by rule number, direction, or IP range using helper methods. Input validation via Python's ipaddress module ensures only valid IPv4 addresses or ranges are accepted. IPv4 module is used to validate ip addresses as discussed in class.

PseudoCode & Approach:

The process of developing the program for task 2 was divided into 3 main parts and the main program logic, with two additional capabilities (load and save) which we have added to the program:

1)Rule-addition: //ipaddr is a mandatory input
Input: ruleno, direction, ipaddr //ipaddr can be individual or range
If direction is None //direction is not mentioned

- Bidirectional_insert(ruleno, ipaddr)

Ellf direction is given [-in | -out]

- Single_insert(ruleno, direction, ipaddr)

Check for existing rule on same priority level
Adjustpriorties(ruleno)
Rule added to list

2)Rule-removal: //ruleno is mandatory
Input: ruleno, direction
If direction is None

- Remove all rules with given ruleno

Else

- Remove rule with exact matching ruleno & direction
-

3)Rule-list: //all parameters are optional
Input: ruleno, direction , ipaddr
Filter rules based on inputs
If rule no is provided, filter based on ruleno
If direction is specified, filter by direction [-in | -out]
If ipaddr is specified

- If address range is provided, print all addresses
- Else: exact ipaddr is matched

4)Main Program Logic:
Print All available commands
While True:
If action is “add”

- Parse the parameters (ruleno, direction, ipaddr)
- Call Rule-addition()

If action is “remove”

- Parse the parameters (ruleno, direction)
- Call Rule-removal()

If action is “list”

- Parse the parameters (ruleno, direction, ipaddr)
- Call Rule-list()

If action is “save” //additional functionality to save rules within a session as text file

- Create textfile with saved rules

If action is “load” //additional functionality to load rules from previous sessions

- Import textfile by asking filename

If action is “exit”
BREAK

Testing Results:

```
C:\Users\prudh>python3 firewall.py
Available commands:
  add [rule_num] [-in|-out] [IP_address] - Add a rule
  remove [rule_num] [-in|-out] - Remove a rule
  list [rule_num] [-in|-out] [IP_address] - List rules
  load - Load rules from a file
  save - Save rules to a file
  exit - Exit the program
```

- **Adding rules:**

```
Enter command: add 1 -in 10.0.0.1
Enter command: add 2 -out 10.0.0.2
Enter command: list
Rule 1 | Direction: -in | Address: 10.0.0.1
Rule 2 | Direction: -out | Address: 10.0.0.2
```

```
Enter command: add 10.0.0.2-10.0.0.4
Enter command: list
Rule 1 | Direction: -in | Address: 10.0.0.2
Rule 1 | Direction: -out | Address: 10.0.0.2
Rule 2 | Direction: -in | Address: 10.0.0.3
Rule 2 | Direction: -out | Address: 10.0.0.3
Rule 3 | Direction: -in | Address: 10.0.0.4
Rule 3 | Direction: -out | Address: 10.0.0.4
```

```
Enter command: add -in 10.0.0.3
Enter command: list
Rule 1 | Direction: -in | Address: 10.0.0.3
Rule 2 | Direction: -in | Address: 10.0.0.1
Rule 3 | Direction: -out | Address: 10.0.0.2
```

If no rule number is provided, then it is assumed to be rule 1.

```
Enter command: add 2 -in 10.0.0.4
Enter command: list
Rule 1 | Direction: -in | Address: 10.0.0.3
Rule 2 | Direction: -in | Address: 10.0.0.4
Rule 3 | Direction: -in | Address: 10.0.0.1
Rule 4 | Direction: -out | Address: 10.0.0.2
```

if a rule of the specified number already exists, the new rule is inserted above the existing rule with a higher priority (and all following rules get their numbers incremented by 1)

```
Enter command: add 2 10.0.0.2
Enter command: list
Rule 1 | Direction: -in | Address: 10.0.0.1
Rule 1 | Direction: -out | Address: 10.0.0.1
Rule 2 | Direction: -in | Address: 10.0.0.2
Rule 2 | Direction: -out | Address: 10.0.0.2
```

if no direction is provided, then the new rule applies to both directions.

```
Enter command: add 10.0.0.256
Error: Invalid or missing IP address.
Enter command: add 10.0.0.0.0
Error: Invalid or missing IP address.
```

For the command to be valid it must contain an address or address range. For this program we consider addresses in the range 10.0.0.0-10.0.0.255

- **Removing rules:**

```
Enter command: list
Rule 1 | Direction: -in | Address: 10.0.0.2
Rule 1 | Direction: -out | Address: 10.0.0.2
Rule 2 | Direction: -in | Address: 10.0.0.1
Rule 2 | Direction: -out | Address: 10.0.0.1
Enter command: remove 1
Enter command: list
Rule 2 | Direction: -in | Address: 10.0.0.1
Rule 2 | Direction: -out | Address: 10.0.0.1
Enter command: remove 4
Error: Rule 4 not found. Please enter a valid command.
```

removes an existing rule from the list of firewall rules, if such a rule exists, else an error should be returned. A rule number must be specified. If no direction is specified, then the whole rule is removed.

```
Enter command: list
Rule 2 | Direction: -in | Address: 10.0.0.1
Rule 2 | Direction: -out | Address: 10.0.0.1
Enter command: remove 2 -in
Enter command: list
Rule 2 | Direction: -out | Address: 10.0.0.1
```

if an existing rule was specified to apply to both incoming and outgoing traffic, this command may be used to remove one of the directions from the existing rule

- **Listing rules:**

```
Enter command: list
Rule 1 | Direction: -in | Address: 10.0.0.3
Rule 1 | Direction: -out | Address: 10.0.0.3
Rule 2 | Direction: -in | Address: 10.0.0.2
Rule 2 | Direction: -out | Address: 10.0.0.2
Rule 3 | Direction: -out | Address: 10.0.0.1
Enter command: list 2
Rule 2 | Direction: -in | Address: 10.0.0.2
Rule 2 | Direction: -out | Address: 10.0.0.2
Enter command: list -in
Rule 1 | Direction: -in | Address: 10.0.0.3
Rule 2 | Direction: -in | Address: 10.0.0.2
Enter command: list -out
Rule 1 | Direction: -out | Address: 10.0.0.3
Rule 2 | Direction: -out | Address: 10.0.0.2
Rule 3 | Direction: -out | Address: 10.0.0.1
```

This lists the existing firewall rules. If no rule is specified, then all rules matching the subsequent parameters are displayed. -in|-out can be used to list only incoming or outgoing packets; if not specified then both will be listed.

```
Enter command: list 10.0.0.2-10.0.0.3
Rule 1 | Direction: -in | Address: 10.0.0.2
Rule 1 | Direction: -out | Address: 10.0.0.2
Rule 2 | Direction: -in | Address: 10.0.0.3
Rule 2 | Direction: -out | Address: 10.0.0.3
```

addr will list the rules matching the provided address or address range;

```
Enter command: save
Enter the filename to save rules to: rules.txt
Rules successfully saved to rules.txt.
Enter command: load
Enter the filename to load rules from: rules.txt
Rules successfully loaded from rules.txt.
```

User can load/save rules

Appendix:

Task1-

PlainText:

To PrudhviAbdul: Planning is key to achieving any goal, whether personal or professional. Breaking down a large goal into smaller, manageable tasks allows for incremental progress, keeping motivation high and reducing overwhelm. Staying organized with tools like planners or digital apps can streamline the process, helping you track and adapt. Consistency is essential; even small, daily actions compound over time. By reviewing your progress regularly, you can celebrate milestones, adjust strategies, and remain focused.

CipherText (Key Length-5)

[60, 313, 500, 201, 0, 292, 208, 201, 199, 452, 201, 187, 153, 464, 361, 172, 12, 299, 313, 187, 12, 464, 220, 151, 347, 12, 153, 299, 0, 373, 208, 160, 311, 325, 500, 187, 201, 373, 361, 347, 139, 160, 292, 500, 187, 12, 208, 220, 151, 452, 304, 201, 0, 464, 208, 359, 139, 160, 153, 340, 187, 213, 208, 172, 151, 452, 373, 187, 201, 373, 361, 347, 139, 160, 153, 340, 304, 352, 172, 0, 151, 464, 500, 340, 48, 373, 160, 151, 139, 160, 373, 340, 153, 325, 220, 292, 311, 160, 201, 500, 139, 12, 208, 153, 151, 452, 325, 347, 187, 373, 373, 199, 151, 299, 340, 48, 0, 373, 373, 201, 139, 160, 165, 347, 139, 373, 373, 187, 151, 452, 325, 500, 187, 213, 220, 359, 311, 325, 153, 340, 165, 220, 313, 0, 12, 172, 325, 187, 60, 165, 220, 220, 311, 299, 500, 187, 199, 12, 208, 139, 311, 464, 373, 340, 304, 12, 208, 160, 139, 160, 340, 187, 48, 304, 361, 347, 151, 292, 153, 187, 199, 373, 373, 160, 311, 153, 153, 187, 201, 373, 361, 292, 311, 304, 153, 347, 187, 373, 220, 160, 311, 313, 340, 187, 60, 304, 313, 151, 139, 160, 340, 340, 48, 373, 361, 160, 151, 464, 201, 340, 48, 165, 361, 151, 151, 292, 153, 347, 12, 165, 220, 361, 311, 311, 325, 201, 0, 165, 220, 151, 311, 313, 500, 500, 199, 304, 325, 0, 151, 325, 500, 500, 139, 12, 208, 172, 311, 325, 313, 500, 139, 325, 220, 311, 151, 452, 500, 347, 12, 165, 220, 151, 139, 160, 165, 347, 139, 373, 373, 347, 199, 172, 201, 500, 187, 304, 325, 151, 139, 160, 452, 340, 60, 325, 220, 153, 311, 299, 500, 187, 199, 12, 208, 311, 311, 464, 213, 187, 201, 464, 361, 160, 199, 313, 292, 340, 352, 373, 313, 0, 311, 160, 292, 340, 199, 213, 160, 0, 151, 299, 500, 187, 12, 12, 208, 201, 151, 452, 201, 347, 60, 165, 373, 172, 311, 325, 361, 201, 0, 373, 373, 340, 151, 452, 325, 347, 199, 213, 208, 299, 311, 313, 340, 201, 304, 12, 48, 361, 199, 313, 153, 500, 201, 213, 220, 199, 151, 304, 153, 187, 352, 304, 361, 347, 151, 299, 500, 187, 201, 352, 361, 299, 151, 153, 153, 347, 199, 213, 220, 292, 311, 0, 153, 347, 12, 373, 373, 359, 311, 313, 325, 201, 0, 373, 208, 172, 311, 311, 201, 201, 0, 304, 208, 151, 151, 299, 500, 187, 304, 325, 220, 201, 199, 151, 153, 187, 352, 304, 313, 0, 151, 313, 292, 340, 199, 213, 220, 292, 151, 299, 340, 48, 0, 165, 220, 153, 199, 160, 325, 201, 0, 165, 373, 160, 311, 165, 153, 347, 187, 464, 208, 201, 151, 452, 153, 340, 213, 373, 208, 172, 311, 325, 201, 201, 0, 464, 208, 12, 151, 292, 153, 347, 0, 304, 361, 359, 151, 311, 201, 500, 187, 304, 325, 151, 139, 160, 292, 187, 60, 373, 208, 153, 311, 299, 500, 187, 199, 12, 208, 325, 311, 464, 213, 201, 0, 464, 208, 201, 151, 299, 313, 340, 340, 12, 208, 160, 311, 325, 201, 48, 0, 165, 220, 139, 151, 299, 165, 347, 12, 220, 313, 0, 12, 311, 500, 340, 304, 304, 373, 172, 199, 311, 213, 187, 60, 373, 361, 208, 359, 139, 153, 187, 201, 304, 325, 0, 151, 452, 325, 500, 187, 325, 220, 199, 199, 313, 292, 340, 48, 373, 160, 373, 139, 160, 201, 500, 151, 325, 220, 199, 139, 160, 325, 340, 213, 165, 220, 151, 311, 153, 340, 48, 0, 325, 208, 160, 311, 299, 340, 347, 201, 12, 208, 160, 151, 311, 213, 187, 201, 373, 373, 199, 199, 151, 153, 187, 187, 373, 373, 311, 199, 160, 500, 500, 60, 373, 361, 139, 139, 160, 500, 500, 151, 325, 220, 201, 139, 160, 213, 187, 201, 373, 220, 299, 299, 165, 153, 139, 139, 352, 172, 0, 199, 172, 201, 500, 151, 213, 220, 299, 199, 464, 292, 340, 304, 325, 325, 0, 359, 299, 500, 500, 60, 304, 313, 0, 199, 160, 325, 187, 352, 325, 373, 201, 151, 452, 325, 500, 187, 12, 208, 201, 151, 452, 361, 500, 60, 373, 208, 160, 199, 172, 340, 347, 201, 220, 160, 0, 359, 299, 500, 500, 60, 12, 208, 208, 151, 299, 500, 48, 0, 165, 373, 299, 311, 313, 201, 340, 139, 304, 361, 160, 199, 313, 201, 201, 0, 373, 220, 172, 311, 313, 201, 500, 187, 464, 208, 359, 311, 325, 201, 500, 187, 220, 160, 0, 151, 299, 201, 187, 292, 464, 220, 361, 199, 153, 153, 347, 187, 464, 208, 201, 151, 299, 213, 187, 60, 325, 373, 172, 151, 452, 325, 201, 165, 12, 208, 160, 311, 325, 201, 48, 0, 304, 361, 299, 311, 452, 153, 340, 201, 373, 313, 0, 151, 325, 500, 340, 187, 464, 220, 361, 151, 452, 201, 48, 304]

```
PS C:\Users\prudh\desktop> python Task1-Encryption.py
Choose action (generate/encrypt/decrypt): generate
Choose e(n) (between 2 and 64): 5
Keys generated with length 5 and saved to files 'public_key.
txt' and 'private_key.txt'.
PS C:\Users\prudh\desktop> python Task1-Encryption.py
Choose action (generate/encrypt/decrypt): encrypt
Enter the plaintext file name: message.txt
Enter the public key file name: public_key.txt
Enter the output file name for ciphertext: ciphertext.txt
Encryption complete. Ciphertext saved to ciphertext.txt
PS C:\Users\prudh\desktop> python Task1-Encryption.py
Choose action (generate/encrypt/decrypt): decrypt
Enter the ciphertext file name: ciphertext.txt
Enter the private key file name: private_key.txt
Enter the output file name for decrypted text: decrypted.txt

Decryption complete. Decrypted text saved to decrypted.txt
```

Figure 1 Generation of Public & Private keys of length 7 , Encryption & Decryption

```
PS C:\Users\prudh\desktop> python Task1-Encryption.py
Choose action (generate/encrypt/decrypt): generate
Choose e(n) (between 2 and 64): 7
Keys generated with length 7 and saved to files 'public_key.
txt' and 'private_key.txt'.
PS C:\Users\prudh\desktop> python Task1-Encryption.py
Choose action (generate/encrypt/decrypt): encrypt
Enter the plaintext file name: message.txt
Enter the public key file name: public_key.txt
Enter the output file name for ciphertext: ciphertext.txt
Encryption complete. Ciphertext saved to ciphertext.txt
PS C:\Users\prudh\desktop> python Task1-Encryption.py
Choose action (generate/encrypt/decrypt): decrypt
Enter the ciphertext file name: ciphertext.txt
Enter the private key file name: private_key.txt
Enter the output file name for decrypted text: decrypted.txt

Decryption complete. Decrypted text saved to decrypted.txt
```

Figure 2 Generation of Public & Private keys of length 5 , Encryption & Decryption

CipherText (Key Length-7)

[832, 1224, 336, 273, 390, 889, 1009, 336, 437, 1107, 1006, 395, 946, 712, 733, 770, 671, 907, 294, 273, 390, 476, 411, 1126, 827, 1068, 871, 570, 1032, 156, 689, 887, 278, 1068, 892, 962, 889, 156, 775, 1282, 278, 834, 715, 671, 567, 315, 967, 848, 437, 1401, 850, 731, 390, 273, 1128, 1209, 278, 990, 1048, 848, 946, 320, 1011, 42, 1266, 1245, 673, 962, 432, 198, 967, 731, 278, 951, 673, 962, 668, 1006, 1362, 1126, 320, 1401, 117, 351, 1443, 733, 177, 497, 1032, 1224, 892, 570, 988, 1006, 570, 931, 827, 1401, 850, 629, 375, 876, 177, 411, 1032, 990, 715, 629, 1401, 315, 1128, 726, 278, 990, 829, 1004, 1266, 876, 177, 375, 278, 1224, 673, 785, 668, 1107, 967, 42, 710, 1401, 892, 629, 375, 156, 689, 1126, 1110, 1224, 336, 507, 1224, 593, 411, 892, 671, 990, 949, 276, 375, 156, 1245, 375, 827, 834, 715, 848, 988, 273, 294, 892, 554, 733, 907, 528, 946, 1006, 806, 887, 278, 834, 871, 528, 609, 1149, 1126, 887, 278, 990, 1006, 1004, 434, 156, 689, 1126, 476, 1128, 850, 806, 1165, 551, 1128, 614, 320, 1401, 117, 507, 390, 889, 1362, 726, 1032, 990, 949, 827, 990, 320, 177, 1165, 554, 1167, 949, 351, 1123, 1032, 1084, 42, 671, 1401, 1126, 528, 1123, 1167, 411, 614, 437, 1401, 1048, 453, 390, 198, 689, 726, 437, 556, 673, 629, 887, 278, 177, 731, 554, 1167, 907, 806, 1224, 315, 1128, 726, 278, 1224, 1126, 570, 988, 889, 1126, 775, 554, 1401, 829, 689, 653, 156, 393, 614, 320, 1362, 871, 629, 887, 951, 177, 1282, 1032, 990, 892, 629, 887, 315, 614, 492, 554, 556, 907, 848, 1123, 611, 455, 42, 1110, 1224, 1048, 848, 609, 850, 177, 892, 437, 1245, 892, 689, 390, 333, 1011, 375, 827, 1224, 850, 962, 668, 850, 177, 1282, 1032, 556, 673, 528, 1123, 1107, 689, 614, 320, 1401, 117, 351, 946, 333, 219, 887, 278, 834, 892, 629, 653, 156, 570, 614, 1032, 990, 715, 629, 1165, 476, 689, 1126, 554, 733, 907, 528, 567, 395, 177, 497, 1032, 1224, 892, 671, 988, 1006, 570, 715, 278, 1068, 673, 806, 609, 333, 689, 1126, 710, 733, 1063, 629, 1443, 572, 177, 614, 1032, 834, 715, 671, 1167, 156, 411, 1126, 554, 556, 673, 629, 432, 273, 219, 614, 785, 556, 556, 671, 1443, 1032, 570, 931, 1032, 1284, 673, 806, 887, 829, 731, 42, 437, 1128, 336, 351, 988, 1006, 570, 492, 827, 1107, 829, 629, 946, 320, 848, 42, 554, 1284, 850, 806, 653, 156, 570, 1048, 320, 1401, 829, 411, 375, 156, 733, 375, 437, 1401, 1063, 512, 390, 273, 528, 614, 437, 1401, 1048, 726, 990, 156, 528, 1282, 671, 1128, 829, 1004, 988, 1032, 733, 42, 827, 1284, 850, 962, 434, 156, 775, 931, 671, 1167, 315, 453, 234, 712, 731, 42, 1032, 990, 949, 570, 1123, 551, 1126, 931, 827, 990, 336, 507, 1123, 1149, 1009, 731, 278, 951, 907, 393, 1443, 1107, 336, 492, 1032, 1128, 336, 507, 668, 551, 1084, 770, 671, 834, 949, 393, 609, 336, 1011, 42, 993, 1401, 1126, 689, 390, 829, 411, 1126, 278, 834, 892, 806, 609, 551, 294, 731, 320, 1284, 673, 689, 390, 593, 689, 892, 554, 1128, 1126, 528, 1443, 1032, 967, 887, 629, 556, 673, 629, 432, 754, 1009, 887, 1110, 556, 907, 827, 432, 889, 411, 614, 554, 1167, 1048, 629, 988, 850, 1011, 42, 320, 1401, 850, 411, 390, 889, 967, 1048, 320, 1245, 871, 453, 390, 990, 1362, 609, 1110, 1128, 892, 806, 198, 876]

Public and Private Keys-Length 7