F20CN Coursework Report

Computer Network Security

Coursework 2 Report

Group 7

Name: Fatima Patel
HW ID: H00339652
HW username: fp17

Name: Dhruv Davey
HW ID: H00371626
HW username: dhd2000

due Fri, 23 Nov 2023 (Week 11)

HERIOT WATT UNIVERSITY

1 Task 1: Alternative Method of Public-Key Encryption

- SystemRandom() used to generate random numbers since it relies on underlying OS mechanisms (free from predictable patterns therefore suitable for use in cryptography). Upper bound for random numbers was system size since it's very large for increased security & randomness. sympy is used to check if a number is prime.
- Message is split into blocks the size of key length, then each block is encrypted. If message to be encrypted is shorter than block length, no padding will be added but instead the message will be encrypted as is.

1.1 Pseudo-Code

FUNCTION generate_e_sequence(n):
    return list of random positive int so each value is > than sum of previous.

FUNCTION find_random_prime_greater_than(n):
    generate random number > n, if it's prime return else regenerate

FUNCTION select_prime_and_compute_h(e_sequence):
    generate prime number q > 2*maxValue(e_sequence) and w relatively prime to q
    h_sequence = compute public key from e_sequence using q and w
     return q, w and

FUNCTION encrypt_message(message to be encrypted, public key which is h_sequence):
    convert message to binary and split into blocks then encrypt and return blocks

FUNCTION decrypt_message(encrypted_blocks, e_sequence, q, w):
    w_inv = modular multiplicative inverse of w with respect to q, binary_string = ''
    Loop through each encrypted block and calculate c' = (c * w_inv) % q
        Loop through reverse(e_sequence) & if c' is >= to current element,
            add '1' to binary_string and reduce c' by that element
    Convert binary_string to text and return fully decrypted message

FUNCTION process_message():
    prompt user for message file and operation
    if operation is −e then encrypt_message(message, public key)
    else if operation is −d then dencrypt_message(message, private key)

FUNCTION main():
    Loop asking user if they want to generate keypair, encrypt/decrypt file or terminate
    If generate key pair, ask if short/long key then generate & print key pair:
        e_sequence = generate_e_sequence(short or long key length)
        q, w, h_sequence = select_prime_and_compute_h(e_sequence)
    If user wants to encrypt/decrypt a message: process_message()
    If user wants to terminate, exit main loop

1.2 Testing

- After running main(), the user chooses to either generate key pair, encrypt/decrypt *.txt* file or terminate session [11].
- The user can pick between 2 key lengths: short (8) or long (256). The shorter key is ideal for testing while the longer key is more secure (although it's still not recommended since this is a proven weak encryption/decryption scheme). The key pair is then printed to terminal [1].

```
Enter '1' to generate a public/private key pair or
Enter '2' to encrypt/decrypt a file or
Enter 't' to terminate session:)
1
Enter 's' for a short key length ideal for testing (size 8) or
Enter 'l' for a more secure key (size 256):
s
Public/Private Key Pair Generated :)
Private Key (e_sequence, q, w): [565912615926583285, 907618765081188057, 2445345290267691230, 4918934748370979193,
11812382617453473881, 21107355967994023639, 56862404260886710602, 124565479670448674593] , 257230577893906711001 ,
7991898566767858237

Public Key (h_sequence): 86073308652546949776, 151559335921230004554, 180931658915026927240, 119174810736936443758,
88769448387469800293, 1569953384568833376, 214212878107218490677, 248462887117781150856
```

Figure 1: Screenshot of first key pair generated

- The input format of message to be encrypted/decrypted should be a *.txt* file and the outputted encrypted/decrypted message will be a .txt file with a modified name to reflect the operation done on the file (e.g. `encrypted_filename.txt` or `decrypted_filename.txt`)
- We used the public key from the first key pair shown above, which has key length of 8 and encrypted a simple message file with 'hello'. 'hello' is converted to binary and then split into 5 blocks since it is 40 bits long. The encrypted blocks are separated by a space and written to a .txt file. The contents of the file are as below:

  `42126044322372673208 7 582523835338606916026 42283039660829556546 3 42283039660829556546 885506161833295206996`
- We inputted this cipher text and used the private key from the first key pair shown above, and the output was 'hello'.
- Screenshots of user generating key pairs, inputting 500 char file for encryption/decryption, entering incorrectly formatted inputs and terminating session are in appendix [A.5].

2 Task 2: Known-plaintext attack: Symmetric Cipher

In this task, we write a program that takes a command as an input argument, followed by the mandatory and, optionally, the non-mandatory arguments. We use the *sys* package to read the arguments passed. Here is the pseudo-code for the program:

2.1 Pseudo-code

```
firewall_rules = load_rules() # read and store the rules from ‘ firewall_rules .json ‘.
command = arguments[1] # Here list of ‘arguments‘ retrived by SYS.ARGV
addr, rule, direction = '' # initialize variables for arguments as empty character
# ------------------- check arguments -------------------- #
FOR argument IN arguments[2::] # the rest of the arguments
    IF argument.IS_IP() : addr = argument # checks if arg. is an IP address
    ELIF argument.IS_NUMERIC() : rule = argument # checks if arg. is a (rule) number
    ELIF direction IN ["–in", "–out"] : direction = argument # if arg. is a direction
    ELSE: PRINT_ERROR("invalid argument")
# ------------- check command and call function ------------- #
IF command == "add": add_rule(rule, direction, addr)
ELIF command == "remove": remove_rule(rule, direction, addr)
ELIF command == "list": list_rule(rule, direction, addr)
ELSE : PRINT_ERROR("Invalid command")

FUNCTION add_rule: # ---------- function to add rule ---------- #
    # Check if mandatory arguments are there
    IF NOT addr : PRINT_ERROR("missing argument addr")
    rule = 1 IF rule == '' ELSE rule # set rule as 1 if empty
    IF rule IN firewall_rules : # if rule exists
        add_rule( firewall_rules [rule]) # recall function with incremented rule
     firewall_rules .ADD({rule OR 1 : addr, direction OR "–in and –out"})
FUNCTION remove_rule: # -------- function to remove rule --------- #
    # Check if mandatory arguments are there
    IF not rule : print_error ("missing argument rule")
    # Check if additional argument is there
    IF addr : print_error ("cannot use addr with remove")
    # Remove rule if it exists
    IF firewall_rules .EXISTS(rule) : remove_rule(direction, rule, firewall_rules )
FUNCTION list_rule: # ---------- function to list rules ---------- #
      firewall_rules .SORT() # sort the list by rule rnumber
    # List existing rules
    FOR rule R in firewall_rules:
        IF addr IN ['', R.addr] AND rule == ['', R.rule] AND direction == ['', R.
            direction]:
            PRINT(R)
save_rules( firewall_rules ) # Save updated rules to the JSON file.
```

2.2 Testing

We display the usage of the program when the user enters no command [12].

Adding rules:

- When we add a rule without the mandatory argument, *addr*, it gives an error [2].

```
>>> Coursework-2 python task2-g7.py add
error: missing argument addr
usage: add [rule] [-in|-out] addr
>>> Coursework-2
```

Figure 2: Adding rule without mandatory argument gives error.

- If the optional arguments are not specified, it takes *rule* as *1* and *direction* as *-in and -out* and adds the rule [3.
- The *addr* could be an IP address or IP address range [3].

```
>>> Coursework-2 python task2-g7.py add 10.0.0.100
rule 1 for address 10.0.0.100 added
>>> Coursework-2 python task2-g7.py add 5 -out 10.0.0.10-10.0.0.20
rule 5 for address 10.0.0.10-10.0.0.20 added
>>> Coursework-2 python task2-g7.py list
Rule    Direction       Address
1       -in and -out    10.0.0.100
5       -out            10.0.0.10-10.0.0.20
>>> Coursework-2
```

Figure 3: Adding rules and displaying them using the *list* command

- If the user enters a new rule with an existing rule number, it is added above the current rule by renumbering the rules below it.

```
>>> Coursework-2 python task2-g7.py add 2 -in 10.0.0.35
rule 2 for address 10.0.0.35 added
>>> Coursework-2 python task2-g7.py add -out 10.0.0.40
rule 1 for address 10.0.0.40 added
>>> Coursework-2 python task2-g7.py list
Rule    Direction       Address
1       -out            10.0.0.40
2       -in and -out    10.0.0.100
3       -in             10.0.0.35
5       -out            10.0.0.10-10.0.0.20
>>> Coursework-2
```

Figure 4: Adding rule with existing rule number.

Removing rules:

- The usage is displayed if the mandatory argument, *rule*, is missing or if an extra argument, *addr* is given.

- Rule can be removed by specifying the rule number [5].

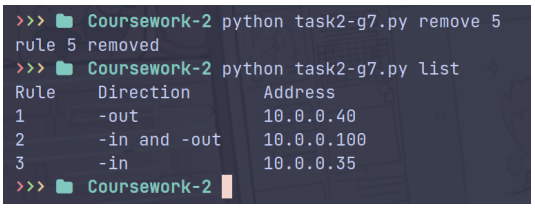

Figure 5: Rule number *5*, which applied for *addr 10.0.010-10.0.0.20* was removed

- By specifying the direction only that direction is removed.

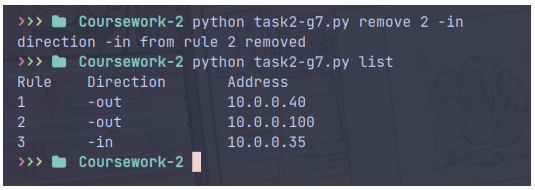

Figure 6: Removing direction *-in* from rule *2*.

Listing rules:

- The user can list all the rules by passing no arguments (seen in above screenshots) [4].
- The user can specify either the address, rule number, direction or any combination of the three to filter the list [7].

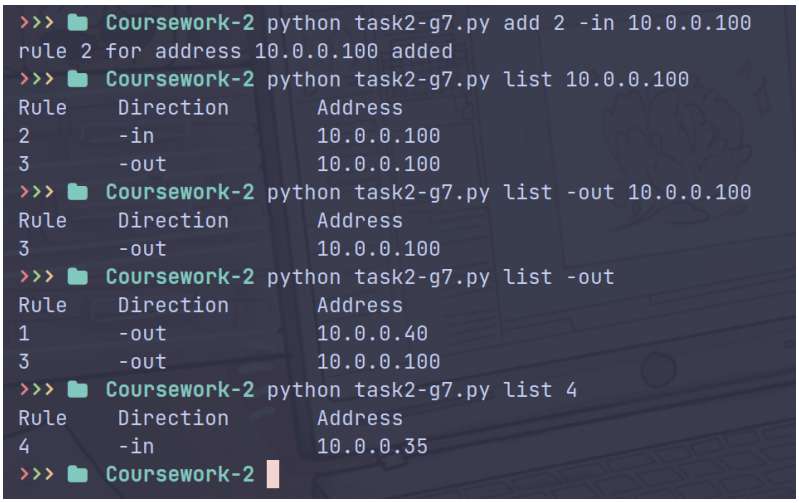

Figure 7: Listing rules with different optional arguments.

More screenshots of error messages and help menu of the program are given in the appendix [B.2].

Appendices

A Appendix Task 1

A.1 Python Program

```
import random
import math
import random
from sympy import isprime
import sys
import os

# Function generates a list of positive integers e such that each value is greater than
    the sum of prev ones.
# @param n: the length of the sequence
# @returns e_sequence: part of the private key
def generate_e_sequence(n):
    # Start list with a random integer.
    # SystemRandom() was used since it's most secure, as it relies on underlying OS
        mechanisms.
    # Upperbound was system size since it is very large for increased security &
        randomness
    # (on a 64-bit system, it's 2^63 - 1)
    e_sequence = [random.SystemRandom().randint(1, sys.maxsize/n)]

    for i in range(1, n):
        # Set the minimum value of the next element to be greater than the sum of all
            previous elements
        min_value = sum(e_sequence) + 1
        # Randomly select a value greater than the sum of previous elements
        # Upperbound chosen as twice the size of min_value so that random values don't
            go out of bounds
```

```
        e_sequence.append(random.randint(min_value, min_value * 2))

    return e_sequence

# function finds a random prime number greater than a specified number n.
# @param n: number to find random primes number greater than
def find_random_prime_greater_than(n):
    while True:
        # Generate a random number greater than n.
        potential_prime = random.SystemRandom().randint(n+1, n+sys.maxsize)
        # Return random number if it's prime
        if isprime(potential_prime):
            return potential_prime

# Function generates public key and part of the private key (q and w)
# @param e_sequence: list of integers with random values
# @returns q: prime number at least twice as large as the max value in e_sequence
# @returns w: number relatively prime to q
# @returns h_sequence: public key
def select_prime_and_compute_h(e_sequence):
    # Finds prime number greater than twice max of e_sequence (i.e., the last value)
    q = find_random_prime_greater_than(2 * max(e_sequence))

    # Loops until a w that is relatively prime to q is found
    while True:
        w = random.SystemRandom().randint(2, sys.maxsize)
        if math.gcd(w, q) == 1 and w != q:
            break

    # Compute public key
    h_sequence = [(w * e) % q for e in e_sequence]

    return q, w, h_sequence

# Function splits message into blocks then encrypts it using the public key, h_sequence.
# @param message: plaintext to be encrypted
# @param h_sequence: public key which is a sequence of integers
# @returns encrypted_blocks: list of integers, which are encrypted blocks of the
    original message.
def encrypt_message(message, h_sequence):
    # Convert the message to a binary string
    binary_str = ''.join(format(ord(char), '08b') for char in message)

    # Calculate the length of each block
    block_length = len(h_sequence)
```

```
    # Split the binary string into blocks of size block_length
    blocks = [binary_str[i:i + block_length] for i in range(0, len(binary_str),
        block_length)]

    # list for encrypted blocks
    encrypted_blocks = []

    # Encrypt each block
    for block in blocks:
        c = 0
        # sets length of the current block (useful if the last block is shorter than
            block_length)
        min_length = min(len(h_sequence), len(block))
        # Iterates through each bit of the block and the corresponding value in
            h_sequence
        for i in range(min_length):
            c += int(block[i]) * h_sequence[i]
        # Appends the encrypted value of the block
        encrypted_blocks.append(c)

    return encrypted_blocks

# Function decrypts a list of encrypted integer blocks using the private key (e_sequence
    , q, w).
# @param encrypted_blocks: list of integers which are encrypted blocks of original
    message
# @param e_sequence: list of integers part of the private key
# @param q: integer part of the private key, used in the modular arithmetic during
    decryption.
# @param w: integer part of the private key, used to calculate the modular
    multiplicative inverse.
# @returns decrypted_message: decrypted message as a string
def decrypt_message(encrypted_blocks, e_sequence, q, w):
    # Initialize empty string to hold decrypted message
    decrypted_message = ''

    # Compute the modular multiplicative inverse of w with respect to q
    w_inv = pow(w, -1, q)

    # Iterate through each encrypted block
    for c in encrypted_blocks:
        # c' is the product of the encrypted block and the modular inverse of w, modulo
            q
        c_prime = (c * w_inv) % q
```

```
            # Initialize empty string for the binary representation of decrypted block
            decrypted_binary_str = ''
            # Iterate through the elements of e_sequence starting from the largest
            for e in reversed(e_sequence):
                # If c' is greater than or equal to the current element, add '1' to the
                    binary string and reduce c' by that element
                if c_prime >= e:
                    decrypted_binary_str = '1' + decrypted_binary_str
                    c_prime -= e
                # If c' is less than the current element, add '0' to the binary string
                else:
                    decrypted_binary_str = '0' + decrypted_binary_str

            # Convert the binary string to text and append to the decrypted message
            decrypted_text = ''.join(chr(int(decrypted_binary_str[i:i+8], 2)) for i in
                range(0, len(decrypted_binary_str), 8))
            decrypted_message += decrypted_text

        # Return fully decrypted message
        return decrypted_message

# Function generates a new filename based on the original filename and the operation.
# @param filename: filename as a string
# @param operation: string indicating the operation, '-e' for encryption and '-d' for
    decryption
# @returns new filename as string with a suffix based on the operation
def generate_output_filename(filename, operation):
    # Split filename into the name and extension
    name, ext = os.path.splitext(filename)
    # Determine suffix based on the operation
    suffix = '_encrypted' if operation == '-e' else '_decrypted'
    return f"{name}{suffix}{ext}"

# Function prompts user for file and operation (encryption/decryption) and processes
    message based on that
def process_message():
    # Continuously prompt for user input until valid input is received
    while True:
        # Prompt user to enter the filename and operation (encryption or decryption)
            using appropriate flag
        user_input = input("Enter your text filename followed by -e for encryption or -
            d for decryption (e.g., 'message.txt -e'): ")
        # Split the input into filename and operation
        parts = user_input.split()
```

```
        # Check if the format of the input is correct
        if len(parts) != 2 or parts[1] not in ['-e', '-d']:
            print("Invalid input format. Please try again!")
            continue

        # Unpack the filename and operation from the input
        filename, operation = parts

        # Check if the file is a .txt file
        if not filename.endswith('.txt'):
            print("Please provide a .txt file. Other file types are not supported.")
            continue

        #Check if specified file exists
        if not os.path.exists(filename):
            print("File does not exist. Please try again!")
            continue

        # Exits loop if valid input is received
        break

# Open and read the contents of the file into message
with open(filename, 'r') as file:
    message = file.read()

# Generate a new filename based on the operation
output_filename = generate_output_filename(filename, operation)

# Iterate until public/private key entered correctly
while True:
    try:
        # Process message based on the operation
        if operation == '-e':
            # Prompt for public key to encrypt message
            h_sequence_input = input("Enter the public key (h_sequence) as comma
                -separated values: ").replace(" ", "")
            # Convert public key into list of integers from string sequence
            h_sequence = [int(x.strip()) for x in h_sequence_input.split(',')]
            # Encrypt the message using the public key
            processed_message = encrypt_message(message, h_sequence)
            # Convert the encrypted blocks to a space-separated string
            processed_message_str = ' '.join(map(str, processed_message))
            break
```

```
            elif operation == '-d':
                # Prompt for private key to decrypt message and parse to remove any
                    spaces
                private_key_input = input("Enter the private key (e_sequence, q, w) as
                    comma-separated values (e.g., [1,2,4],45,47): ").replace(" ", "")

                # Parse e_sequence, q, and w from the input
                closing_bracket_index = private_key_input.find(']')
                e_sequence_str = private_key_input[1:closing_bracket_index]
                q_str, w_str = private_key_input[closing_bracket_index + 2:].split(',',
                    1)

                # Convert e_sequence_str to a list of integers
                e_sequence = [int(x.strip()) for x in e_sequence_str.split(',')]

                # Convert q_str and w_str to integers
                q = int(q_str.strip())
                w = int(w_str.strip())

                # converts contents of message file which are encrypted blocks
                    seperated by space
                # (if the same program was used for encryption) to list of integers
                encrypted_blocks = [int(x) for x in message.split()]

                # Decrypt the message using the private key
                processed_message = decrypt_message(encrypted_blocks, e_sequence, q, w)
                processed_message_str = processed_message
                break

        # If invalid key format used, prompt user again
        except ValueError:
            print("Invalid key format. Please try again!")
            continue

    # Write the processed message to the output file
    with open(output_filename, 'w') as file:
        file.write(processed_message_str)

    # Notify the user of output filename
    print(f"Processed file saved as: {output_filename}")

def main():
    # Prompt user for input until they terminate by inputting 't'
    while True:
        # Ask user if they want to generate a key pair, encrpt/decrypt a file or
```

```
        terminate session
    user_choice = input("\nEnter '1' to generate a public/private key pair or\
        nEnter '2' to encrypt/decrypt a file or\nEnter 't' to terminate session :)\n
        ")

    # If user wants to generate key pair
    if user_choice == '1':
        # Ask user if they want a short/long key
        key_size_choice = input("Enter 's' for a short key length ideal for testing
            (size 8) or\nEnter 'l' for a more secure key (size 256):\n")

        # Assign key size based on user's choice
        if key_size_choice == 's':
            key_size = 8
        elif key_size_choice == 'l':
            key_size = 216
        else:
            # If invalid key size is inputted, alert user and reprompt them
            print("Invalid key size choice. Please try again!")
            continue

        # Generate the public/private key pair
        e_sequence = generate_e_sequence(key_size)
        q, w, h_sequence = select_prime_and_compute_h(e_sequence)

        # Display the keys
        print("Public/Private Key Pair Generated :)")
        print("Private Key (e_sequence, q, w):", e_sequence,",", q,",", w)
        print("\nPublic Key (h_sequence):", ', '.join(map(str, h_sequence)))

    # If user wants to encrypt/decrypt a message
    elif user_choice == '2':
        # function processes users message by encrypting/decrypting as they choose
        process_message()

    # If user wants to terminate session
    elif user_choice == 't':
        print("Session Terminated :)")
        # Exit loop since session terminated
        break
    # Handle any other invalid input
    else:
        print("Invalid input. Please try again!")

main()
```

A.2 Plaintext

A.2.1 First Plain text File: 500char.txt

To Fatima: Here, computational resources may be measured using a variety of different metrics (e.g., number of classical elementary operations, quantum circuit size, etc .). In order for a cryptosystem to satisfy one of the above security requirements, any attack must require computational resources comparable to or greater than the stated threshold, with respect to all metrics that NIST deems to be potentially relevant to practical security. NIST intends to consider a variety of possible metrics

A.3 Key Pairs

A.3.1 First Key Pair

Private Key (e_sequence, q, w): [565912615926583285, 907618765081188057, 2445345290267691230, 4918934748370979193, 11812382617453473881, 21107355967994023639, 56862404260886710602, 124565479670448674593] , 257230577893906711001 , 7991898566767858237

Public Key (h_sequence): 86073308652546949776, 151559335921230004554, 180931658915026927240, 119174810736936443758, 88769448387469800293, 1569953384568833376, 214212878107218490677, 248462887117781150856

A.3.2 Second Key Pair

Private Key (e_sequence, q, w): [754299180829949536, 1284961407874373115, 2321668293611099402, 8506400030271785791, 15997165293695854542, 47491681435300941889, 130774673328344279139, 355984968162959553665] , 717814301760748956721 , 8284722346187226091

Public Key (h_sequence): 550350132105748580151, 656473946195904243088, 1900225119770411325 66, 703745400116901672285, 280260787011075557363, 697863991769239520657, 282503111375198982648, 601622999240227657397

A.4 Ciphertext

A.4.1 Ciphertext generated using public key from First Key Pair

272304100042735281688 88550616183329520699 6 18093165891502692724 0
367342167413017328607 580953881954038082650 453235758957762208928
669723330341507882943 671293283726076716319 580953881954038082650
603088796146651661968 18093165891502692724 0 240328784308699804847
582523835338606916026 665878683680411866229 582523835338606916026
271271060687065560909 18093165891502692724 0 795166760061256573327
885506161833295206996 671293283726076716319 451665805573193375552
701698646075543359784 453235758957762208928 580953881954038082650
453235758957762208928 669723330341507882943 885506161833295206996
637043274715514056140 580953881954038082650 422830396608295565463
18093165891502692724 0 665878683680411866229 582523835338606916026
914341570798193017085 885506161833295206996 701698646075543359784
665878683680411866229 795166760061256573327 582523835338606916026
914341570798193017085 18093165891502692724 0 671293283726076716319
580953881954038082650 788898141078444326701 18093165891502692724 0
546703872943475422471 582523835338606916026 18093165891502692724 0
671293283726076716319 582523835338606916026 580953881954038082650
914341570798193017085 701698646075543359784 665878683680411866229
582523835338606916026 334060948220825765170 18093165891502692724 0
701698646075543359784 914341570798193017085 669723330341507882943
637043274715514056140 796736713445825406703 18093165891502692724 0
580953881954038082650 18093165891502692724 0 667448637064980699605
580953881954038082650 665878683680411866229 669723330341507882943
582523835338606916026 453235758957762208928 788898141078444326701
18093165891502692724 0 885506161833295206996 548273826328044255847
18093165891502692724 0 334060948220825765170 669723330341507882943
548273826328044255847 548273826328044255847 582523835338606916026
665878683680411866229 582523835338606916026 637043274715514056140
453235758957762208928 18093165891502692724 0 671293283726076716319
582523835338606916026 453235758957762208928 665878683680411866229
669723330341507882943 795166760061256573327 914341570798193017085
18093165891502692724 0 269701107302496727533 582523835338606916026
485483938794284051586 796736713445825406703 485483938794284051586
271271060687065560909 18093165891502692724 0 637043274715514056140
701698646075543359784 671293283726076716319 546703872943475422471
582523835338606916026 665878683680411866229 18093165891502692724 0
885506161833295206996 548273826328044255847 18093165891502692724 0
795166760061256573327 422830396608295565463 580953881954038082650
914341570798193017085 914341570798193017085 669723330341507882943
795166760061256573327 580953881954038082650 422830396608295565463
18093165891502692724 0 582523835338606916026 422830396608295565463
582523835338606916026 671293283726076716319 582523835338606916026

63704327471551405614 45323575895776220892 58095388195403808265
66587868368041186622 78889814107844432670 18093165891502692724
88550616183329520699 45166580557319337555 58252383533860691602
66587868368041186622 58095388195403808265 45323575895776220892
66972333034150788294 88550616183329520699 63704327471551405614
91434157079819301708 27127106068706556090 18093165891502692724
70012869269097452640 70169864607554335978 58095388195403808265
63704327471551405614 45323575895776220892 70169864607554335978
67129328372607671631 18093165891502692724 79516676006125657332
66972333034150788294 66587868368041186622 79516676006125657332
70169864607554335978 66972333034150788294 45323575895776220892
18093165891502692724 91434157079819301708 66972333034150788294
75464813206788166652 58252383533860691602 27127106068706556090
18093165891502692724 58252383533860691602 45323575895776220892
79516676006125657332 48548393879428405158 51816399442027787838
48548393879428405158 18093165891502692724 48879167142648095570
63704327471551405614 18093165891502692724 88550616183329520699
66587868368041186622 33406094822082576517 58252383533860691602
66587868368041186622 18093165891502692724 54827382632804425584
88550616183329520699 66587868368041186622 18093165891502692724
58095388195403808265 18093165891502692724 79516676006125657332
66587868368041186622 78889814107844432670 45166580557319337555
45323575895776220892 88550616183329520699 91434157079819301708
78889814107844432670 91434157079819301708 45323575895776220892
58252383533860691602 67129328372607671631 18093165891502692724
45323575895776220892 88550616183329520699 18093165891502692724
91434157079819301708 58095388195403808265 45323575895776220892
66972333034150788294 91434157079819301708 54827382632804425584
78889814107844432670 18093165891502692724 88550616183329520699
63704327471551405614 58252383533860691602 18093165891502692724
88550616183329520699 54827382632804425584 18093165891502692724
45323575895776220892 42126044322372673208 75825238353386069160
18093165891502692724 58095388195403808265 54670387294347542247
88550616183329520699 66744863706498069960 58252383533860691602
18093165891502692724 91434157079819301708 58252383533860691602
79516676006125657332 70169864607554335978 66587868368041186622
66972333034150788294 45323575895776220892 78889814107844432670
18093165891502692724 66587868368041186622 58252383533860691602
70012869269097452640 70169864607554335978 66972333034150788294
66587868368041186622 58252383533860691602 67129328372607671631
58252383533860691602 63704327471551405614 45323575895776220892
91434157079819301708 27127106068706556090 18093165891502692724
58095388195403808265 63704327471551405614 78889814107844432670
18093165891502692724 58095388195403808265 45323575895776220892
45323575895776220892 58095388195403808265 79516676006125657332

88393620844872637362 0 18093165891502692724 0 67129328372607671631 9
70169864607554335978 4 91434157079819301708 5 45323575895776220892 8
18093165891502692724 0 66587868368041186622 9 58252383533860691602 6
70012869269097452640 8 70169864607554335978 4 66972333034150788294 3
66587868368041186622 9 58252383533860691602 6 18093165891502692724 0
79516676006125657332 7 88550616183329520699 6 67129328372607671631 9
45166580557319337555 2 70169864607554335978 4 45323575895776220892 8
58095388195403808265 0 45323575895776220892 8 66972333034150788294 3
88550616183329520699 6 63704327471551405614 0 58095388195403808265 0
42283039660829556546 3 18093165891502692724 0 66587868368041186622 9
58252383533860691602 6 91434157079819301708 5 88550616183329520699 6
70169864607554335978 4 66587868368041186622 9 79516676006125657332 7
58252383533860691602 6 91434157079819301708 5 18093165891502692724 0
79516676006125657332 7 88550616183329520699 6 67129328372607671631 9
45166580557319337555 2 58095388195403808265 0 66587868368041186622 9
58095388195403808265 0 54670387294347542247 1 42283039660829556546 3
58252383533860691602 6 18093165891502692724 0 45323575895776220892 8
88550616183329520699 6 18093165891502692724 0 88550616183329520699 6
66587868368041186622 9 18093165891502692724 0 79673671344582540670 3
66587868368041186622 9 58252383533860691602 6 58095388195403808265 0
45323575895776220892 8 58252383533860691602 6 66587868368041186622 9
18093165891502692724 0 45323575895776220892 8 42126044322372673208 7
58095388195403808265 0 63704327471551405614 0 18093165891502692724 0
45323575895776220892 8 42126044322372673208 7 58252383533860691602 6
18093165891502692724 0 91434157079819301708 5 45323575895776220892 8
58095388195403808265 0 45323575895776220892 8 58252383533860691602 6
33406094822082576517 0 18093165891502692724 0 45323575895776220892 8
42126044322372673208 7 66587868368041186622 9 58252383533860691602 6
91434157079819301708 5 42126044322372673208 7 88550616183329520699 6
42283039660829556546 3 33406094822082576517 0 27127106068706556090 9
18093165891502692724 0 91591152418276185046 1 66972333034150788294 3
45323575895776220892 8 42126044322372673208 7 18093165891502692724 0
66587868368041186622 9 58252383533860691602 6 91434157079819301708 5
45166580557319337555 2 58252383533860691602 6 79516676006125657332 7
45323575895776220892 8 18093165891502692724 0 45323575895776220892 8
88550616183329520699 6 18093165891502692724 0 58095388195403808265 0
42283039660829556546 3 42283039660829556546 3 18093165891502692724 0
67129328372607671631 9 58252383533860691602 6 45323575895776220892 8
66587868368041186622 9 66972333034150788294 3 79516676006125657332 7
91434157079819301708 5 18093165891502692724 0 45323575895776220892 8
42126044322372673208 7 58095388195403808265 0 45323575895776220892 8
18093165891502692724 0 45611161158004871289 0 48879167142648095570 3
73340991188316608984 5 27230410004273528168 8 18093165891502692724 0
33406094822082576517 0 58252383533860691602 6 58252383533860691602 6
67129328372607671631 9 91434157079819301708 5 18093165891502692724 0

45323575895776220892 88550616183329520699 18093165891502692724
05467038729434754224 71 58252383533860691602 18093165891502692724
04516658055731933755 52 88550616183329520699 45323575895776220892
05825238353386069160 26 63704327471551405614 04532357589577622089
28 66972333034150788294 30 58095388195403808265 04228303966082955654
63 42283039660829556546 33 78889141078444326701 18093165891502692724
06658786836804118662 29 58252383533860691602 42283039660829556546
33 58252383533860691602 66744863706498069960 58095388195403808265
04 63704327471551405614 04532357589577622089 18093165891502692724
04532357589577622089 88550616183329520699 18093165891502692724
04516658055731933755 66587868368041186622 58095388195403808265
07951667600612565733 27 45323575895776220892 66972333034150788294
07951667600612565733 27 58095388195403808265 04228303966082955654
63 18093165891502692724 91434157079819301708 58252383533860691602
07951667600612565733 27 70169864607554335978 46 66587868368041186622
66972333034150788294 30 45323575895776220892 78889141078444326701
48548393879428405158 6 18093165891502692724 45611161580048712890
0 48879167142648095570 3 73340991188316608984 5 27230410004273528168
8 18093165891502692724 66972333034150788294 30 63704327471551405614
04532357589577622089 05 58252383533860691602 63704327471551405614
33406094822082576517 0 91434157079819301708 18093165891502692724
04532357589577622089 88550616183329520699 18093165891502692724
07951667600612565733 27 88550616183329520699 63704327471551405614
91434157079819301708 66972333034150788294 33406094822082576517
05825238353386069160 26 66587868368041186622 18093165891502692724
05809538819540380826 50 18093165891502692724 66744863706498069960
05809538819540380826 50 66587868368041186622 66972333034150788294
05825238353386069160 26 45323575895776220892 78889141078444326701
18093165891502692724 88550616183329520699 54827382632804425584
7 18093165891502692724 45166580557319337555 2 88550616183329520699
91434157079819301708 91434157079819301708 66972333034150788294
54670387294347542247 1 42283039660829556546 33 58252383533860691602
18093165891502692724 67129328372607671631 9 58252383533860691602
04532357589577622089 26 66587868368041186622 66972333034150788294
07951667600612565733 27 91434157079819301708

A.4.2 Ciphertext generated using public key from Second Key Pair

20580833380820454360 30 27087473475686870937 19 19002251197704113256 6
16368410493403427463 93 14481194574131730331 51 22481058500590865685 96
17283802444242485904 14 24262442361934881110 71 14481194574131730330 51
14565318104802173448 62 19002251197704113256 6 93673473320697980045 1
21459834491824125537 08 18327449696650460305 87 21459834491824125537 08
11681472907573562105 86 19002251197704113256 6 17306225687883720156 99
27087473475686870937 19 24262442361934881110 71 15502418582898470479 39

2849728849299314225993 2248105850059086568596 1448119457413173033051
2248105850059086568596 1728380244424248590414 2708747347568687093719
2107124348328459436322 1448119457413173033051 1824621236953260453674
1900225119770411325666 1832744969665046030587 2145983449182412553708
2434367968905273687984 2708747347568687093719 2849728849299314225993
1832744969665046030587 1730622568788372015699 2145983449182412553708
2434367968905273687984 1900225119770411325666 2426244236193488111071
1448119457413173033051 2432125644541150262699 1900225119770411325666
1128999569548144358302 2145983449182412553708 1900225119770411325666
2426244236193488111071 2145983449182412553708 1448119457413173033051
2434367968905273687984 2849728849299314225993 1832744969665046030587
2145983449182412553708 1544360449942184896311 1900225119770411325666
2849728849299314225993 2434367968905273687984 1728380244424248590414
2107124348328459436322 2428486560557611536356 1900225119770411325666
1448119457413173033051 1900225119770411325666 2530608961434285551244
1448119457413173033051 1832744969665046030587 1728380244424248590414
2145983449182412553708 2248105850059086568596 2432125644541150262699
1900225119770411325666 2708747347568687093719 1826863561317383878959
1900225119770411325666 1544360449942184896311 1728380244424248590414
1826863561317383878959 1826863561317383878959 2145983449182412553708
1832744969665046030587 2145983449182412553708 2107124348328459436322
2248105850059086568596 1900225119770411325666 2426244236193488111071
2145983449182412553708 2248105850059086568596 1832744969665046030587
1728380244424248590414 1730622568788372015699 2434367968905273687984
1900225119770411325666 4702832989881166899929 2145983449182412553708
1450650402132555193234 2428486560557611536356 1450650402132555193234
1168147290757356210586 1900225119770411325666 2107124348328459436322
2849728849299314225993 2426244236193488111071 1128999569548144358302
2145983449182412553708 1832744969665046030587 1900225119770411325666
2708747347568687093719 1826863561317383878959 1900225119770411325666
1730622568788372015699 1824621236953260453674 1448119457413173033051
2434367968905273687984 2434367968905273687984 1728380244424248590414
1730622568788372015699 1448119457413173033051 1824621236953260453674
1900225119770411325666 2145983449182412553708 1824621236953260453674
2145983449182412553708 2426244236193488111071 2145983449182412553708
2107124348328459436322 2248105850059086568596 1448119457413173033051
1832744969665046030587 2432125644541150262699 1900225119770411325666
2708747347568687093719 1550241858289847047939 2145983449182412553708
1832744969665046030587 1448119457413173033051 2248105850059086568596
1728380244424248590414 2708747347568687093719 2107124348328459436322
2434367968905273687984 1168147290757356210586 1900225119770411325666
2151864857530074705336 2849728849299314225993 1448119457413173033051
2107124348328459436322 2248105850059086568596 2849728849299314225993
2426244236193488111071 1900225119770411325666 1730622568788372015699
1728380244424248590414 1832744969665046030587 1730622568788372015699

2849728849299314225993 1728380244424248590414 2248105850059086568596
1900225119770411325 66 2434367968905273687984 1728380244424248590414
2113005756676121587950 2145983449182412553708 1168147290757356210586
1900225119770411325 66 2145983449182412553708 2248105850059086568596
1730622568788372015699 1450650402132555193234 1071906298228344347326
1450650402132555193234 1900225119770411325 66 1538357732447207457848
2107124348328459436322 1900225119770411325 66 2708747347568687093719
1832744969665046030587 1544360449942184896311 2145983449182412553708
1832744969665046030587 1900225119770411325 66 1826863561317383878959
2708747347568687093719 1832744969665046030587 1900225119770411325 66
1448119457413173033051 1900225119770411325 66 1730622568788372015699
1832744969665046030587 2432125644541150262699 1550241858289847047939
2248105850059086568596 2708747347568687093719 2434367968905273687984
2432125644541150262699 2434367968905273687984 2248105850059086568596
2145983449182412553708 2426244236193488111071 1900225119770411325 66
2248105850059086568596 2708747347568687093719 1900225119770411325 66
2434367968905273687984 1448119457413173033051 2248105850059086568596
1728380244424248590414 2434367968905273687984 1826863561317383878959
2432125644541150262699 1900225119770411325 66 2708747347568687093719
2107124348328459436322 2145983449182412553708 1900225119770411325 66
2708747347568687093719 1826863561317383878959 1900225119770411325 66
2248105850059086568596 1126757245184020933017 2145983449182412553708
1900225119770411325 66 1448119457413173033051 1128999569548144358302
2708747347568687093719 2530608961434285551244 2145983449182412553708
1900225119770411325 66 2434367968905273687984 2145983449182412553708
1730622568788372015699 2849728849299314225993 1832744969665046030587
1728380244424248590414 2248105850059086568596 2432125644541150262699
1900225119770411325 66 1832744969665046030587 2145983449182412553708
2151864857530074705336 2849728849299314225993 1728380244424248590414
1832744969665046030587 2145983449182412553708 2426244236193488111071
2145983449182412553708 2107124348328459436322 2248105850059086568596
2434367968905273687984 1168147290757356210586 1900225119770411325 66
1448119457413173033051 2107124348328459436322 2432125644541150262699
1900225119770411325 66 1448119457413173033051 2248105850059086568596
2248105850059086568596 1448119457413173033051 1730622568788372015699
2010883355799447573062 1900225119770411325 66 2426244236193488111071
2849728849299314225993 2434367968905273687984 2248105850059086568596
1900225119770411325 66 1832744969665046030587 2145983449182412553708
2151864857530074705336 2849728849299314225993 1728380244424248590414
1832744969665046030587 2145983449182412553708 1900225119770411325 66
1730622568788372015699 2708747347568687093719 2426244236193488111071
1550241858289847047939 2849728849299314225993 2248105850059086568596
1448119457413173033051 2248105850059086568596 1728380244424248590414
2708747347568687093719 2107124348328459436322 1448119457413173033051
1824621236953260453674 1900225119770411325 66 1832744969665046030587

2145983449182412553708 2434367968905273687984 2708747347568687093719
2849728849299314225993 1832744969665046030587 1730622568788372015699
2145983449182412553708 2434367968905273687984 1900225119770411132566
1730622568788372015699 2708747347568687093719 2426244236193488111071
1550241858289847047939 1448119457413173033051 1832744969665046030587
1448119457413173033051 1128999569548144358302 1824621236953260453674
2145983449182412553708 1900225119770411132566 2248105850059086568596
2708747347568687093719 1900225119770411132566 2708747347568687093719
1832744969665046030587 1900225119770411132566 2428486560557611536356
1832744969665046030587 2145983449182412553708 1448119457413173033051
2248105850059086568596 2145983449182412553708 1832744969665046030587
1900225119770411132566 2248105850059086568596 1126757245184020933017
1448119457413173033051 2107124348328459436322 1900225119770411132566
2248105850059086568596 1126757245184020933017 2145983449182412553708
1900225119770411132566 2434367968905273687984 2248105850059086568596
1448119457413173033051 2248105850059086568596 2145983449182412553708
1544360449942184896311 1900225119770411132566 2248105850059086568596
1126757245184020933017 1832744969665046030587 2145983449182412553708
2434367968905273687984 1126757245184020933017 2708747347568687093719
1824621236953260453674 1544360449942184896311 1168147290757356210586
1900225119770411132566 3132231960674513208641 1728380244424248590414
2248105850059086568596 1126757245184020933017 1900225119770411132566
1832744969665046030587 2145983449182412553708 2434367968905273687984
1550241858289847047939 2145983449182412553708 1730622568788372015699
2248105850059086568596 1900225119770411132566 2248105850059086568596
2708747347568687093719 1900225119770411132566 1448119457413173033051
1824621236953260453674 1824621236953260453674 1900225119770411132566
2426244236193488111071 2145983449182412553708 2248105850059086568596
1832744969665046030587 1728380244424248590414 1730622568788372015699
2434367968905273687984 1900225119770411132566 2248105850059086568596
1126757245184020933017 1448119457413173033051 2248105850059086568596
1900225119770411132566 1917101836351418303756 1538357732447207457848
2244345456928232555418 2058083338082045436030 1900225119770411132566
1544360449942184896311 2145983449182412553708 2145983449182412553708
2426244236193488111071 2434367968905273687984 1900225119770411132566
2248105850059086568596 2708747347568687093719 1900225119770411132566
1128999569548144358302 2145983449182412553708 1900225119770411132566
1550241858289847047939 2708747347568687093719 2248105850059086568596
2145983449182412553708 2107124348328459436322 2248105850059086568596
1728380244424248590414 1448119457413173033051 1824621236953260453674
1824621236953260453674 2432125644541150262699 1900225119770411132566
1832744969665046030587 2145983449182412553708 1824621236953260453674
2145983449182412553708 2530608961434285551244 1448119457413173033051
2107124348328459436322 2248105850059086568596 1900225119770411132566
2248105850059086568596 2708747347568687093719 1900225119770411132566

1550241858289847047939 1832744969665046030587 1448119457413173033051
1730622568788372015699 2248105850059086568596 1728380244424248590414
1730622568788372015699 1448119457413173033051 1824621236953260453674
190022511977041132566 2434367968905273687984 2145983449182412553708
1730622568788372015699 2849728849299314225993 1832744969665046030587
1728380244424248590414 2248105850059086568596 2432125644541150262699
1450650402132555193234 190022511977041132566 1917101836351418303756
1538357732447207457848 2244345456928232555418 2058083338082045436030
190022511977041132566 1728380244424248590414 2107124348328459436322
2248105850059086568596 2145983449182412553708 2107124348328459436322
1544360449942184896311 2434367968905273687984 190022511977041132566
2248105850059086568596 2708747347568687093719 190022511977041132566
1730622568788372015699 2708747347568687093719 2107124348328459436322
2434367968905273687984 1728380244424248590414 1544360449942184896311
2145983449182412553708 1832744969665046030587 190022511977041132566
1448119457413173033051 190022511977041132566 2530608961434285551244
1448119457413173033051 1832744969665046030587 1728380244424248590414
2145983449182412553708 2248105850059086568596 2432125644541150262699
190022511977041132566 2708747347568687093719 1826863561317383878959
190022511977041132566 1550241858289847047939 2708747347568687093719
2434367968905273687984 2434367968905273687984 1728380244424248590414
1128999569548144358302 1824621236953260453674 2145983449182412553708
190022511977041132566 2426244236193488111071 2145983449182412553708
2248105850059086568596 1832744969665046030587 1728380244424248590414
1730622568788372015699 2434367968905273687984

A.5 Screenshots

Screenshots of user generating key pairs, inputting 500 char file for encryption/decryption, entering incorrectly formatted inputs and terminating session are in appendix:

```
Enter '1' to generate a public/private key pair or
Enter '2' to encrypt/decrypt a file or
Enter 't' to terminate session:)
1
Enter 's' for a short key length ideal for testing (size 8) or
Enter 'l' for a more secure key (size 256):
s
Public/Private Key Pair Generated :)
Private Key (e_sequence, q, w): [565912615926583285, 907618765081188057, 2445345290267691230, 4918934748370979193,
1181238261745347 3881, 2110735596794023639, 56862404260886710602, 124565479670448674593] , 257230577893906711001 ,
7991898566767858237

Public Key (h_sequence): 86073308652546949776, 151559335921230004554, 180931658915026927240, 119174810736936443758,
88769448387469800293, 1569953384568833376, 214212878107218490677, 248462887117781150856

Enter '1' to generate a public/private key pair or
Enter '2' to encrypt/decrypt a file or
Enter 't' to terminate session:)
2
Enter your text filename followed by -e for encryption or -d for decryption (e.g., 'message.txt -e'):
 500char.txt -e
Enter the public key (h_sequence) as comma-separated values:
860733086525469497 76, 151559335921230004554, 180931658915026927240, 119174810736936443758, 88769448387469800293, 15
69953384568833376, 214212878107218490677, 248462887117781150856
Processed file saved as: 500char_encrypted.txt

Enter '1' to generate a public/private key pair or
Enter '2' to encrypt/decrypt a file or
Enter 't' to terminate session:)
2
Enter your text filename followed by -e for encryption or -d for decryption (e.g., 'message.txt -e'):
 500char_encrypted.txt -d
Enter the private key (e_sequence, q, w) as comma-separated values (e.g., [1,2,4],45,47):
[565912615926583285, 907618765081188057, 2445345290267691230, 4918934748370979193, 1181238261745347 3881, 2110735596
7994023639, 568624042608867106 02, 124565479670448674593] , 257230577893906711001 , 7991898566767858237
Processed file saved as: 500char_encrypted_decrypted.txt

Enter '1' to generate a public/private key pair or
Enter '2' to encrypt/decrypt a file or
Enter 't' to terminate session:)
t
Session Terminated :)
```

Figure 8: Generating first key pair then encrypting then decrypting 500 char file

```
Enter '1' to generate a public/private key pair or
Enter '2' to encrypt/decrypt a file or
Enter 't' to terminate session:)
1
Enter 's' for a short key length ideal for testing (size 8) or
Enter 'l' for a more secure key (size 256):
s
Public/Private Key Pair Generated :)
Private Key (e_sequence, q, w): [754299180829949536, 1284961407874373115, 2321668293611099402, 8506400030271785791,
1599716529369585 4542, 47491681435300941889, 130774673328344279139, 3559849681629595536 65] , 717814301760748956721 ,
8284722346187226091

Public Key (h_sequence): 550350132105748580151, 656473946195904243088, 190022511977041132566, 703745400116901672285,
280260787011075557363, 697863991769239520657, 282503111375198982648, 601622999240227657397
```

Figure 9: Generating second key pair

```
Enter '1' to generate a public/private key pair or
Enter '2' to encrypt/decrypt a file or
Enter 't' to terminate session:)
2
Enter your text filename followed by -e for encryption or -d for decryption (e.g., 'message.txt -e'):
 hello.txt -e
Enter the public key (h_sequence) as comma-separated values:
86073308652546949776, 151559335921230004554, 180931658915026927240, 119174810736936443758, 88769448387469800293, 15
69953384568833376, 214212878107218490677, 2484628871177811508 56
Processed file saved as: hello_encrypted.txt

Enter '1' to generate a public/private key pair or
Enter '2' to encrypt/decrypt a file or
Enter 't' to terminate session:)
2
Enter your text filename followed by -e for encryption or -d for decryption (e.g., 'message.txt -e'):
 hello_encrypted.txt -d
Enter the private key (e_sequence, q, w) as comma-separated values (e.g., [1,2,4],45,47):
[565912615926583285, 907618765081188057, 2445345290267691230, 4918934748370979193, 11812382617453473881, 2110735596
7994023639, 568624042608867106 02, 1245654796704486745 93] , 257230577893906711001 , 7991898566767858237
Processed file saved as: hello_encrypted_decrypted.txt

Enter '1' to generate a public/private key pair or
Enter '2' to encrypt/decrypt a file or
Enter 't' to terminate session:)
t
Session Terminated :)
```

Figure 10: Encrypting 500 char file using public key generated by 2nd key pair then decrypting it using the private key

```
Enter '1' to generate a public/private key pair or
Enter '2' to encrypt/decrypt a file or
Enter 't' to terminate session:)
s
Invalid input. Please try again!


Enter '1' to generate a public/private key pair or
Enter '2' to encrypt/decrypt a file or
Enter 't' to terminate session:)
```

Figure 11: If user enters incorrect input

B Appendix Task 2

B.1 Python program

This section contains the implementation of task 2 explained by the pseudo-code [2.1].

```
# ---------- IMPORTS ---------- #
import sys
import json

# ---------- CONSTANTS ---------- #
FILE_NAME = "firewall_rules.json"
HELP = f'''
usage: task2-g7.py command [ arguments ... ]

DESCRIPTION:
    A simple program that manages firewall rules for denying access to IPv4 packets
        through the firewall.

COMMANDS:
    {'add [rule] [-in|-out] addr':30}\t- add rule
    {'remove rule [-in|-out]':30}\t- remove rule
    {'list [rule] [-in|-out] [addr]':30}\t- list rule

ARGUMENTS:
    {'rule':8}- a non-zero positive integer that specifies the rule number
    {'-in':8}- specifies the direction of the rule as incoming
    {'-out':8}- specifies the direction of the rule as outgoing
    {'addr':8}- an IP address or an IP address range
'''

# ---------- FUNCTIONS ---------- #
# main function
def main():
    # load rules from JSON file firewall_rules.json
    firewall_rules = load_rules()

    # check if user has entered a valid command (add, remove, list)
    command = check_command()

    # get the arguments entered by the user,
    direction, rule, addr = check_arguments()

    # if command is add
    if command == "add":
        # check if addr is specified, print error and exit otherwise
```

```
            if not addr:
                print_error(
                    "error: missing argument addr\nusage: add [rule] [-in|-out] addr")
            add_rule(direction, rule, addr, firewall_rules)
            print(f"rule {rule or '1'} for address {addr} added")
        # if command is remove
        elif command == "remove":
            # check if rule is specified, print error and exit otherwise
            if not rule:
                print_error(
                    "error: missing argument rule\nusage: remove rule [-in|-out]")
            # check if addr is not specified, print error and exit otherwise
            if addr:
                print_error(
                    "error: cannot use addr with remove\nusage: remove rule [-in|-out]")
            remove_rule(direction, rule, firewall_rules)

        # if command is list
        elif command == "list":
            list_rule(direction, rule, addr, firewall_rules)

        # save rules to JSON file firewall_rules.json
        save_rules(firewall_rules)


# function to load rules from a JSON file to a global dictionary
# @return firewall_rules: the dictionary with the rules
def load_rules():
    try:
        # Load the rules from a JSON file
        with open(FILE_NAME, "r") as file:
            firewall_rules = json.load(file)
    except FileNotFoundError:
        # create an empty dictionary if file does not exist
        firewall_rules = {}
    return firewall_rules


# function to save rules from dictionary to a JSON file
# @param firewall_rules: the dictionary to save
def save_rules(firewall_rules):
    # Save the rules to a JSON file
    with open(FILE_NAME, "w") as file:
        json.dump(firewall_rules, file, indent=4)
```

```
# function to check if a string is an IP address or an IP address range
# @param addr: the string to check
# @return True if addr is an IP address, False or print error and exit otherwise
def is_ip_address(addr):
    # check if the string is an IP address range
    addr_list = addr.split("-")

    # check if there are more than two IP addresses in the string
    if len(addr_list) > 2:
        return False
    # if there is only one IP address in the string check if it is valid
    if len(addr_list) == 1:
        return check_ip(addr)
    # check if the two IP addresses are valid
    from_range = check_ip(addr_list[0])
    to_range = check_ip(addr_list[1])
    if not from_range or not to_range:
        return False
    # check if the two IP addresses are in the same range
    # and the first IP address is smaller than the second
    if from_range[0:3] == to_range[0:3] and int(from_range[3]) < int(to_range[3]):
        return True
    else:
        print_error(f"error: invalid IP address rage '{addr}'")


# function to check if a string is an IP address
# @param addr: the item to check
# @return addr IP address as a list of numbers if it is valid, False otherwise
def check_ip(addr):
    addr = addr.split(".")
    # check if there are 4 numbers in the IP address
    if len(addr) != 4:
        return False
    # check if each number is numeric and between 0 and 255
    for num in addr:
        if not num.isnumeric() or int(num) < 0 or int(num) > 255:
            return False
    return addr


# function to check if user has entered a valid command
# @return command - if it is valid
def check_command():
```

```
    try:
        # check if command is valid
        if sys.argv[1] in ["add", "remove", "list"]:
            return sys.argv[1]
        # if not, print error and exit
        else:
            print_error(f"error: invalid command '{sys.argv[1]}'\n{HELP}")

    except IndexError:
        print_error(f"error: missing command\n{HELP}")


# function to check if user has entered valid arguments
# @return direction - "-in" or "-out" or empty if not specified
# @return rule - positive integer or empty if not specified
# @return addr - IP address if valid, throw error and exit otherwise
def check_arguments():
    arguments = sys.argv[2::]
    if len(arguments) > 3:
        print_error("error: too many arguments")

    # initialize values
    direction = ""
    rule = ""
    addr = ""

    # check if arguments are valid
    for argument in arguments:
        # check if argument is a rule number
        if argument.isnumeric() and argument != "0" and not rule:
            if rule:
                print_error(f"error: invalid argument '{argument}'")
            rule = argument
        # check if argument is a direction
        elif (argument == "-in" or argument == "-out") and not direction:
            if direction:
                print_error("error: cannot use -in and -out at the same time")
            direction = argument
        # check if argument is an IP address
        elif is_ip_address(argument) and not addr:
            addr = argument
        # if argument is not valid print error and exit
        else:
            print_error(f"error: invalid argument {argument}")
    return direction, rule, addr
```

```
# function to add a rule
# @param direction: "-in" or "-out" or False if not specified
# @param rule: positive integer or False if not specified
# @param addr: IP address
# @param firewall_rules: the dictionary with the rules
# @return rule: the rule number of the new rule
def add_rule(direction, rule, addr, firewall_rules):
    # use rule number 1 if not specified
    if not rule:
        rule = "1"

    # check if rule already exists
    if rule in firewall_rules:
        old_rule = firewall_rules[rule]

        # add old rule with new rule number
        add_rule(old_rule["direction"], str(
            int(rule) + 1), old_rule["address"], firewall_rules)

    updated_rule = {rule: {"direction": direction, "address": addr}}
    firewall_rules.update(updated_rule)
    return rule


# function to remove a rule
# @param direction: "-in" or "-out" or False if not specified
# @param rule: positive integer or False if not specified
# @param firewall_rules: the dictionary with the rules
def remove_rule(direction, rule, firewall_rules):
    # check if rule exists
    if rule not in firewall_rules.keys():
        print_error(f"error: rule {rule} does not exist")

    # check if direction is specified
    if not direction:
        firewall_rules.pop(rule)
        print(f"rule {rule} removed")
    else:
        # if direction is not specified add the opposite direction
        if not firewall_rules[rule]["direction"]:
            if direction == "-in":
                firewall_rules[rule].update({"direction": "-out"})
            else:
```

```
                firewall_rules[rule].update({"direction": "--in"})
            print(f"direction {direction} from rule {rule} removed")
        # check if direction is the same
        elif firewall_rules[rule]["direction"] != direction:
            print_error(
                f"error: rule {rule} does not have direction {direction}")
        # if direction is the same remove the rule
        else:
            firewall_rules.pop(rule)
            print(f"rule {rule} removed")


# function to list rules
# @param direction: "--in" or "--out" or False if not specified
# @param rule: positive integer or False if not specified
# @param addr: IP address or False if not specified
# @param firewall_rules: the dictionary with the rules
def list_rule(inp_direction, inp_rule, inp_addr, firewall_rules):
    # sort the rules by rule number
    rules_list = []
    for rule in firewall_rules.keys():
        # bool to check if the rule should be added to the list
        add_to_list = True
        # if direction is not specified add both directions
        direction = firewall_rules[rule]["direction"]
        if not direction:
            direction = "--in and --out"
        # add rule number, direction and address to a list
        rule = [int(rule), direction, firewall_rules[rule]["address"]]

        # check if the rule should be added to the list
        if inp_rule:
            if rule[0] != int(inp_rule):
                add_to_list = False
        if inp_addr:
            if rule[2] != inp_addr:
                add_to_list = False
        if inp_direction:
            if inp_direction not in rule[1]:
                add_to_list = False
        # add the rule to the list if it should be added
        if add_to_list:
            rules_list.append(rule)

    # sort the list by rule number
```

```
    rules_list .sort(key=lambda x: x[0])

    # check if there are rules to show
    if not rules_list:
        print_error("no rules to show")

    # print the rules
    print(f"{'Rule':8}{'Direction':16}{'Address':16}")
    for rule in rules_list:
        print(f"{str(rule[0]):8}{rule[1]:16}{rule[2]:16}")


# function to print error message and exit
# @param error_message: the error message to print
def print_error(error_message):
    print(error_message)
    exit()


main()
```

B.2 Screenshots

This section contains more screenshots of the program. The screenshots below show the error messages and help message when the command or arguments are invalid.

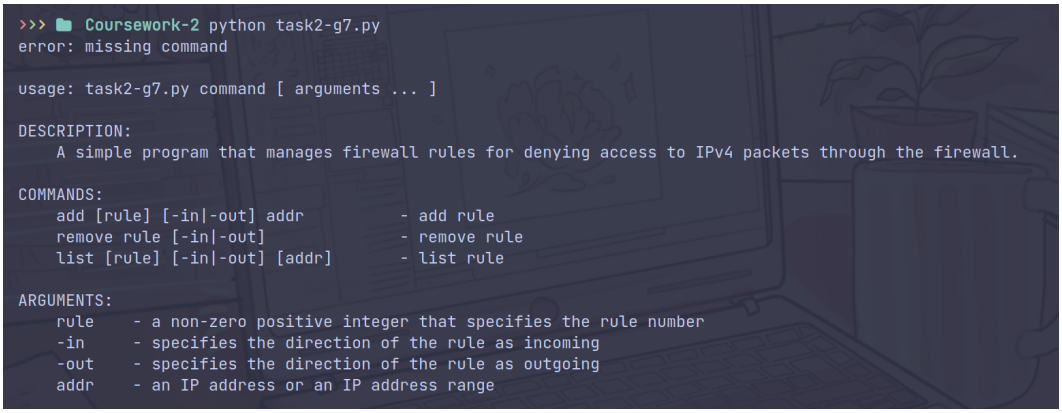

Figure 12: Displaying help when no command is entered.

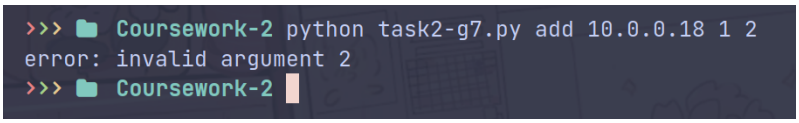

Figure 13: Error when two of the same argument type is given.

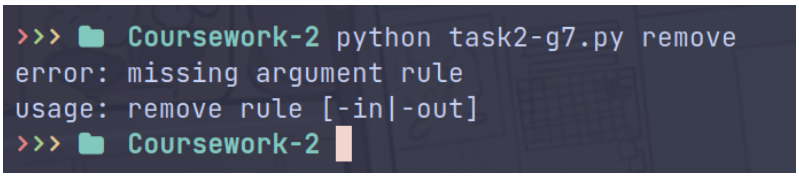

Figure 14: Error message when rule number is not specified.

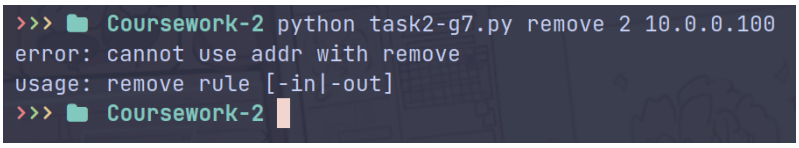

Figure 15: Error when *addr* is used with a remove command.

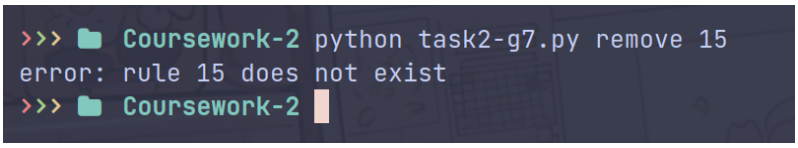

Figure 16: Error when removing rule and rule number doesn't exist.

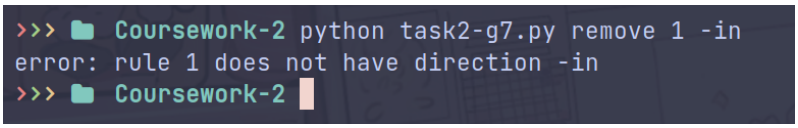

Figure 17: Error when rule doesn't have specified direction when removing rule.

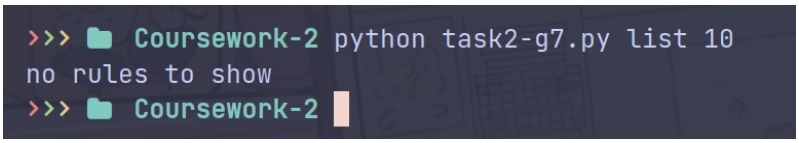

Figure 18: Message when no rules match the list parameters or when the list is empty.

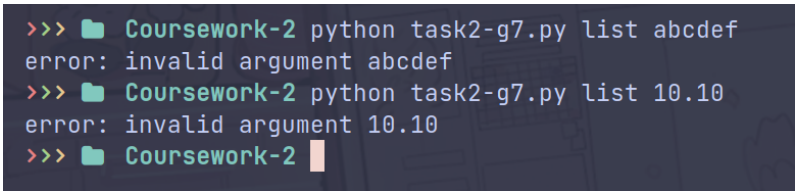

Figure 19: Error when invalid argument is given.

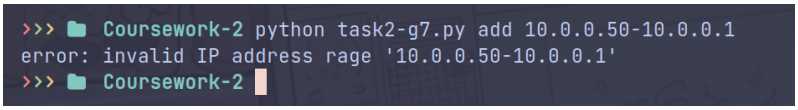

Figure 20: Error when invalid IP address range is given.