Cryptography and Network Security Lab

Assignment 9

Student Details

Name : Krunal Rank
Adm No. : U18CO081

1

```
import os
import random
import hashlib


class CONSTANTS:
   """
   Constants used in the program
   """
   SYSTEM_FILE_PATH = "./system"
   N = 160
   L = 1024
   MILLER_RABIN_ITERATIONS = 128
   LOW_PRIMES = [2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61,
67, 71, 73, 79, 83, 89, 97, 101, 103, 107, 109, 113, 127, 131, 137, 139, 149, 151,
157, 163, 167, 173, 179, 181, 191, 193, 197, 199, 211, 223, 227, 229, 233, 239, 241,
251, 257, 263, 269, 271, 277, 281, 283, 293, 307, 311, 313, 317, 331, 337, 347, 349,
353, 359, 367, 373, 379, 383, 389, 397, 401, 409, 419, 421, 431, 433, 439, 443,
                449, 457, 461, 463, 467, 479, 487, 491, 499, 503, 509, 521, 523,
541, 547, 557, 563, 569, 571, 577, 587, 593, 599, 601, 607, 613, 617, 619, 631, 641,
643, 647, 653, 659, 661, 673, 677, 683, 691, 701, 709, 719, 727, 733, 739, 743, 751,
757, 761, 769, 773, 787, 797, 809, 811, 821, 823, 827, 829, 839, 853, 857, 859, 863,
877, 881, 883, 887, 907, 911, 919, 929, 937, 941, 947, 953, 967, 971, 977, 983, 991,
997]
   BUF_SIZE = 65536

class ERRORS:
   """
   Error messages used in the program
   """
   INVALID_CHOICE = "Please select a valid option."
```

```
    INVALID_USERNAME = "Invalid username. Username must contain only lowercase,
uppercase alphabets, numbers or underscore."
    INVALID_SIGNATURE_NAME = "Invalid signature name. Signature name must contain only
lowercase, uppercase alphabets, numbers or underscore."
    INVALID_AUTH = "Invalid Authorization. Please sign in as a User first."
    INVALID_FILE = "Invalid File Path. File not found."
    INVALID_SIGNATURE = "Invalid signature. Please make sure the signature file is
valid."

    NOT_FOUND_USER = "User not found. Please enter valid user to verify the
signature."
    NOT_FOUND_SIGNATURE = "Signature file not found. Please enter valid signature file
to verify the signature."
    NOT_FOUND_FILE = "File not found. Please enter valid file to verify the
signature."

P = None
Q = None
G = None
X = None
Y = None


def is_prime(n):
    """
    Checks if n is a likely prime
    """

    if n in CONSTANTS.LOW_PRIMES:
        return True

    for prime in CONSTANTS.LOW_PRIMES:
        if n % prime == 0:
            return False

    if n == 2:
        return True
    if n % 2 == 0:
        return False
    s = 0
    d = n - 1
    while d % 2 == 0:
        d //= 2
        s += 1
    for _ in range(CONSTANTS.MILLER_RABIN_ITERATIONS):
        a = random.randint(2, n - 2)
```

```
        x = pow(a, d, n)
        if x == 1 or x == n - 1:
            continue
        for _ in range(s - 1):
            x = pow(x, 2, n)
            if x == n - 1:
                break
        else:
            return False
    return True


def generate_large_prime(bit_size=1024):
    """
    Generates a random prime number of bit_size bit length
    """
    while True:
        p = random.randint(2**(bit_size-1), 2**bit_size)
        if is_prime(p):
            return p


def initialize_system():
    """
    Creates system values (P,Q,G) if they don't exist
    """
    global P, Q, G
    if os.path.exists(CONSTANTS.SYSTEM_FILE_PATH):
        with open(CONSTANTS.SYSTEM_FILE_PATH, "r") as file:
            P = int(file.readline())
            Q = int(file.readline())
            G = int(file.readline())
        print("System Loaded.")
        print("P:", P)
        print("Q:", Q)
        print("G:", G)
        return
    Q = generate_large_prime(CONSTANTS.N)

    while True:
        k = random.randint(2**(456-1), 2**456)
        p = k*Q + 1
        if is_prime(p):
            P = p
            break
    print(Q, P)
```

```
    G = 1
    while G == 1:
        h = random.randint(2, P-2)
        G = int(pow(h, ((P-1)//Q), P))

    with open(CONSTANTS.SYSTEM_FILE_PATH, "w") as file:
        file.write(str(P) + "\n")
        file.write(str(Q) + "\n")
        file.write(str(G) + "\n")


def set_user_dialog():
    """
    Dialog for creating and setting user
    """
    username = input("Enter username: ")

    for c in username:
        ascii_c = ord(c)
        if not ((ascii_c >= 65 and ascii_c <= 90) or (ascii_c >= 97 and ascii_c <=
122) or (ascii_c >= 48 and ascii_c <= 57) or ascii_c == 95):
            raise Exception(ERRORS.INVALID_USERNAME)

    global X, Y, P, Q, G
    if os.path.exists(username):
        with open(username, "r") as file:
            X = int(file.readline())
            Y = int(file.readline())
        print(
            f"Logging with username {username} successful using cached credentials.")
        return

    X = random.randint(1, Q-1)
    Y = int(pow(G, X, P))

    with open(username, "w") as file:
        file.write(str(X) + "\n")
        file.write(str(Y) + "\n")
    print(f"Logging with username {username} successful.")


def sign_file_dialog():
    """
    Dialog for signing a file and creating signature
    """
    if X is None or Y is None:
```

```
        raise Exception(ERRORS.INVALID_AUTH)

    file_path = input("Enter file path: ")
    signature_name = input("Enter signature identity: ")

    for c in signature_name:
        ascii_c = ord(c)
        if not ((ascii_c >= 65 and ascii_c <= 90) or (ascii_c >= 97 and ascii_c <=
122) or (ascii_c >= 48 and ascii_c <= 57) or ascii_c == 95):
            raise Exception(ERRORS.INVALID_SIGNATURE_NAME)

    if not os.path.exists(file_path):
        raise Exception(ERRORS.INVALID_FILE)

    with open(file_path, "rb") as file:
        file_hash = hashlib.sha256(file.read()).hexdigest()
        file_hash_int = int(file_hash, 16)

    k = random.randint(1, Q-1)
    r = 0
    while r==0:
        r = int(pow(G, k, P)) % Q
    s = (pow(k,Q-2,Q)*(file_hash_int + X%Q*r%Q)%Q) % Q

    with open(f"{signature_name}.sig", "w") as file:
        file.write(str(r) + "\n")
        file.write(str(s) + "\n")

    print(f"Signature {signature_name}.sig created successfully.")

def verify_signature_dialog():
    """
    Dialog that verifies signatures
    """
    signature_name = input("Enter signature identity: ")
    file_path = input("Enter file path: ")
    user = input("Enter username: ")

    if not(os.path.exists(user)):
        raise Exception(ERRORS.NOT_FOUND_USER)
    if not(os.path.exists(f"{signature_name}.sig")):
        raise Exception(ERRORS.NOT_FOUND_SIGNATURE)
    if not(os.path.exists(file_path)):
        raise Exception(ERRORS.NOT_FOUND_FILE)

    with open(user, "r") as file:
```

```
        _ = int(file.readline())
        y = int(file.readline())
    with open(f"{signature_name}.sig", "r") as file:
        r = int(file.readline())
        s = int(file.readline())
    with open(file_path, "rb") as file:
        file_hash = hashlib.sha256(file.read()).hexdigest()
        file_hash_int = int(file_hash, 16)

    if (r<0 or r>=Q) or (s<0 or s>=Q):
        raise Exception(ERRORS.INVALID_SIGNATURE)

    w = pow(s, Q-2, Q)
    u1 = (file_hash_int * w) % Q
    u2 = (r * w) % Q
    v = ((pow(G, u1, P) * pow(y, u2, P)) % P) % Q

    if v == r:
        print(f"Signature is valid. The signature {signature_name}.sig verifies that
{file_path} is sent by {user}.")
        return

    print(f"Signature is not valid.")


def main_dialog():
    """
    Runs main dialog
    """
    choice = int(input("1. Access as User\n2. Sign File\n3. Verify Signature\n4.
Exit\nEnter your choice : "))

    try:
        if choice == 1:
            set_user_dialog()
        elif choice == 2:
            sign_file_dialog()
        elif choice == 3:
            verify_signature_dialog()
        elif choice == 4:
            return
        else:
            raise Exception(ERRORS.INVALID_CHOICE)
    except Exception as e:
        print(e)
    main_dialog()
```

```python
if __name__ == "__main__":
    try:
        print("Digital Signature Program\n")
        initialize_system()

        main_dialog()
    except Exception as e:
        print(e)
```

```
kr@arc-warden:/mnt/6AD574E142A88B4D/BTech/Assignments/4th_Year/CNS/Assignment_9$ python3 1.py
Digital Signature Program

System Loaded.
P: 15962082584228820242459650577693178919158089466379276208709748792619279170086620583038985524648818477085614404906828576651920439720049692099391461975827752918709657121709538844684687261 1
Q: 1314536977736406346734474897334307637619363141813
G: 39466191796842432835116907322490676182427113122916656350003531616966816213776074639332093681223255221751363506364169571389319493331833752632970627051859187855513941484522993157404788 33
1. Access as User
2. Sign File
3. Verify Signature
4. Exit
Enter your choice :
```

```
1. Access as User
2. Sign File
3. Verify Signature
4. Exit
Enter your choice : 1
Enter username: kr
Logging with username kr successful.
1. Access as User
2. Sign File
3. Verify Signature
4. Exit
Enter your choice : 2
Enter file path: test_input.txt
Enter signature identity: test_input_signature
Signature test_input_signature.sig created successfully.
1. Access as User
2. Sign File
3. Verify Signature
4. Exit
Enter your choice : 3
Enter signature identity: test_input_signature
Enter file path: test_input.txt
Enter username: kr
Signature is valid. The signature test_input_signature.sig verifies that test_input.txt is sent by kr.
```

```
1. Access as User
2. Sign File
3. Verify Signature
4. Exit
Enter your choice : 1
Enter username: dr
Logging with username dr successful.
1. Access as User
2. Sign File
3. Verify Signature
4. Exit
Enter your choice : 3
Enter signature identity: test_input_signature
Enter file path: test_input.txt
Enter username: dr
Signature is not valid.
```