Problem 1 Symmetric/Asymmetric Encryption

Q1

Consider a group of 30 people in a room who wish to be able to establish pairwise secure communications in the future. How many keys need to be exchanged in total:
(a) Using symmetric cryptography?
(b) Using public key cryptography?

My Solution:
(a) For symmetric pairwise, (N * (N-1)) / 2 = (30*29) / 2 = 435 unique connections between nodes. There are 435 keys that need to be exchanged in total
(b) For using public key cryptography, it would be 30 keys.

Q2

Note: I wrote a python program to solve this question. The code implementation is in the script called 'hw3_1_Q2.py'. The full output of this script is the picture below. I will answer the question with the solutions' explanation and the key code on each sub-question. For detailed information, please check the script. Here is the final output for each sub-question.

```
hw3_1_Q2 (1)
D:\SoftWareDev\IDE\Anaconda3\envs\env_ml\python.exe D:/WorkSpace/PyWorkSpace/cs528/hw3/problem3/hw3_1_Q2.py
Q2 - 1st:  N = 68254939 Φ(N) = 68238256
Q2 - 2nd: Find the greatest common divisor (GCD) of e = 3 and Φ(N) = 68 238 256 is 1
Q2 - 3rd: The inverse of e mod Φ(N) is 45492171
Q2 - 4th: Let C be the resulting ciphertext. The ciphertext is: 23081176
Q2 - 5th: Decryption successful. Decrypted P is equal to original P.
Q2 - 6th: The decrypted plaintext is: 35654065
Q2 - 7th: Encryption successful. Encrypted C' is equal to original C'.

Process finished with exit code 0
```

The following question has you use RSA. You may use a program that you write or any other computer program to help you solve this problem.
Let p = 9, 497 and q = 7, 187 and e = 3.

1st

What is N? What is Φ(N)?
N = p*q = 9497 * 7187 = 68254939
Φ(N) = (p-1)(q-1) = (9497 - 1) * (7187-1) = 9476 * 7186 = 68238256

2nd

Verify that e is relatively prime to Φ(N). What method did you use to verify this?

e is relatively prime to Φ(N) because the greatest common divisor (GCD) of e = 3 and Φ(N) = 68 238 256 is 1.

Here is the way I verify it:

Two integers are said to be relatively prime if they do not have any common factors other than 1 (the greatest common divisor (GCD) of the two integers is 1.). Hence, to verify that e is relatively prime to Φ(N), we need to check whether the greatest common divisor (GCD) of e and Φ(N) is equal to 1. In other words, we need to check whether e and Φ(N) have any common factors other than 1.

Here is the code implementation:

```
'''
Q2 - 2nd: check if the greatest common divisor (GCD) of e and Φ(N) is 1.
'''
def gcd(a, b):
    if isinstance(a, tuple):
        a = a[0]
    if isinstance(b, tuple):
        b = b[0]
    while b != 0:
        a, b = b, a % b
    return a
```

```
if __name__ == "__main__":
    # Q2 - 1st
    cal_n_phi_N()

    # Q2 - 2nd
    e = 3
    Φ_n = 68238256
    print('Q2 - 2nd: Find the greatest common divisor (GCD) of e = 3 and Φ(N) = 68 238 256 is', gcd(e, Φ_n))
```

```
Run: Problem_1_Q2 (1)
D:\SoftWareDev\IDE\Anaconda3\envs\env_ml\python.exe D:/WorkSpace/PyWorkSpace/cs528/hw3
Q2 - 1st:  N = 68254939 Φ(N) = 68238256
Q2 - 2nd: Find the greatest common divisor (GCD) of e = 3 and Φ(N) = 68 238 256 is 1
```

I get the greatest common divisor (GCD) of e = 3 and Φ(N) = 68 238 256 is 1. Hence, e is relatively prime to Φ(N).

3rd

Compute d as the inverse of e mod Φ(N). What is d?

d is 45492171.

Here is the way I compute d: We need to find an integer d such that $e * d \equiv 1 \bmod \Phi(N) = 3 * d \equiv$

1 mod Φ(N). In other words, we need to find an integer d such that when we multiply e and d and take the remainder modulo Φ(N), we get a remainder of 1. We can use the extended Euclidean algorithm to find d. This algorithm finds the GCD of two integers a and b, and at the same time computes the coefficients x and y that satisfy the equation: a * x + b * y = GCD(a, b). Hence, we need to find the coefficients x and y that satisfy: e * x + Φ(N) * y = 1

We can use the extended Euclidean algorithm to solve this equation for x and y.

```python
def extended_euclidean_algorithm(a, b):
    if b == 0:
        return a, 1, 0
    else:
        gcd, x, y = extended_euclidean_algorithm(b, a % b)
        return gcd, y, x - (a // b) * y
```

Output:

```
Run:  hw3_1_Q2
D:\SoftWareDev\IDE\Anaconda3\envs\env_ml\python.exe D:\WorkSpace\PyWorkSpace\cs528\hw3\problem3\hw3_1_Q2.py
Q2 - 1st:  N = 68254939 Φ(N) = 68238256
Q2 - 2nd: Find the greatest common divisor (GCD) of e = 3 and Φ(N) = 68 238 256 is 1
Q2-3: The inverse of e mod Φ(N) is 45492171
```

Get the inverse of e mod Φ(N) is 45492171.

4th

Encrypt the value P = 22446688 with the RSA primitive and the values for N and e above. Let C be the resulting ciphertext. What is C?

C = 23081176.

To encrypt the plaintext P = 22446688 using RSA, we need to compute its ciphertext C using the public key (N, e): C = P^e mod N. Here is my code implementation:

```python
def encrypt():
    '''Q2 - 4'''
    p = 9497
    q = 7187
    e = 3
    N = p * q
    phi_N = (p - 1) * (q - 1)

    # Set the plaintext value
    P = 22446688

    # Calculate the ciphertext value
    # Equivalent to base**exp with 2 arguments or base**exp % mod with 3 arguments
    # C = P^e mod N
    C = pow(P, e, N)

    # Print the ciphertext
    print("Q2 - 4th: Let C be the resulting ciphertext. The ciphertext is:", C)
```

Output:

```
Q2 - 4th: Let C be the resulting ciphertext. The ciphertext is: 23081176
```

5th

Verify that you can decrypt C using d as the private exponent to get back P. What method did you use to verify this?

We already know that C = 23081176, P = 22446688, d = 45492171. To verify that we can decrypt C using d as the private exponent to get back P, we can raise C to the power of d modulo N: P = C^d mod N, where d is the private exponent. Here is my code output:

```
Q2 - 5th: Decryption successful. Decrypted P is equal to original P.
```

Here is my code implementation:

```
def decryption_c():
    C = 23081176
    p = 22446688
    d = 45492171
    N = 68254939
    decrypted_p = pow(C, d, N)
    if decrypted_p == p:
        print("Q2 - 5th: Decryption successful. Decrypted P is equal to original P.")
    else:
        print("Q2 - 5th: Decryption failed. Decrypted P is not equal to original P.")
```

6th

Decrypt the value C′ = 11335577 using the RSA primitive and your values for N and d above. Let P′ be the resulting plaintext. What is P′?

P' = 35654065

We already know N = 68254939, d = 45492171, C' = 11335577. To decrypt the ciphertext C' = 11335577 using RSA and the values for N and d that we have calculated earlier, we need to compute the plaintext P' using the following formula: P' = C'^d mod N.

Here is my code output:

```
Q2 - 6th: The decrypted plaintext is: 35654065
```

Here is my code implementation:

```
def decryption_c_p():
    p = 9497
    q = 7187
    d = 45492171
    C_prime = 11335577
    N = p * q
    P_prime = pow(C_prime, d, N)
    print("Q2 - 6th: The decrypted plaintext is:", P_prime)
```

7th

Verify that you can encrypt P′ using e as the public exponent to get back C′. What method did you use to verify this?

We know that P' = 62523021, e = 3, C' = 11335577. To verify that we can encrypt P' using e as the public exponent to get back C', we can calculate C' using the formula: C' = P'^e mod N.

Here is my code implementation:

```
def encript_p_prime():
    p = 9497
    q = 7187
    d = 45492171
    C_prime = 11335577
    N = p * q
    P_prime = pow(C_prime, d, N)

    encrypted_C2 = pow(P_prime, e, N)
    # print("Encrypted C' =", encrypted_C2)
    # print("Original C' =", C_prime)

    if encrypted_C2 == C_prime:
        print("Q2 - 7th: Encryption successful. Encrypted C' is equal to original C'.")
    else:
        print("Q2 - 7th: Encryption failed. Encrypted C' is not equal to original C'.")
```

Here is my code output:

```
Q2 - 7th: Encryption successful. Encrypted C' is equal to original C'.
```

Q3

Consider a Diffie-Hellman key exchange with p = 29 and g = 2. Suppose that Alice picks x = 3 and Bob picks y = 5. What will each party send to the other, and what shared key will they agree on? Show your details.

My solution:

We get that Alice and Bob agree on a prime number p = 29 and a base g =2, and each of them chooses a secret integer in the Diffie-Hellman key exchange: Alice picks x = 3 and Bob picks y = 5. Then, they need to exchange values derived from their secret integers which is x = 3, y=5 to compute a shared secret key.

Step 1. To generate their values:

Alice computes $A = g^x \bmod p \rightarrow A = g^x \bmod p = 2^3 \bmod 29 = 8$

Bob computes $B = g^y \bmod p \rightarrow B = g^y \bmod p = 2^5 \bmod 29 = 3$

Alice sends A = 8 to Bob, and Bob sends B = 3 to Alice.

Step 2. To compute the shared secret key:

Alice computes the shared key = B^x mod p ➔ $\text{SharedKey}_{\text{Alice}}$ = B^x mod p = 3^3 mod 29 = 27 mod 29 = 27

Bob computes the shared key = A^y mod p ➔ $\text{SharedKey}_{\text{Bob}}$ = A^y mod p = 8^5 mod 29 = 32768 mod 29 = 27

We see: Alice and Bob have computed the same shared secret key which is 27. Hence, in this Diffie-Hellman key exchange, Alice sends the value A = 8 to Bob, Bob sends the value B = 3 to Alice, and they both agree on the shared secret key = 27.

Problem 2 Homomorphic Encryption: Pallier encryption

Let N = pq where p and q are two prime numbers. Let g ∈ [0, N^2] be an integer satisfying g = aN+1(mod N^2) for some integer a ≤ N. Consider the following encryption scheme. The public key is (N, g). The private key is (p, q, a). To encrypt a (integer) message m, one picks a random integer h, and computes C = $g^m h^N \bmod N^2$. Our goal is to develop a decryption algorithm and to show the homomorphic property of the encryption scheme.

(1)

Show the discrete log problem "mod N^2 base g" is easy when knowing the private key. That is, show that given g and B =g^x mod N^2, there is an efficient algorithm to recover x mod N. Use the fact that g = aN + 1 for some integer a ≤ N.

My solution:
Since we can use the fact that g = aN + 1 for some integer a ⩽ N.
Given g and B = gx mod N2, we can recover x mod N efficiently using this information:
Compute B' = B ⋆ (aN + 1)^(−x) mod N^2.
Since B = g^x mod N^2 and g = aN + 1, we have B' = (aN + 1)^x ⋆ (aN + 1)^(−x) mod N^2 = 1.
Since B' ≡ 1 (mod N), we have B' − 1 is divisible by N. So we can compute x as x ≡ ((B' − 1)/N ⋆ a^(−1)) (mod N).

Using the definition of g➔g ≡ aN+1(mod N^2), which implies g ≡ 1+aN(mod N^2). Therefore, we can express B as:

B ≡ g^x(mod N^2)
≡ (1+aN)^x(mod N^2)
≡ 1 + axN + (aN)^2⋆C(mod N^2), where C is a polynomial in aN.

Now, we can define a new value A = (B-1)/N. Notice that A is an integer because B-1 is divisible by N, which implies A = (B-1)/N is also an integer. Using this definition, we can rewrite the previous expression as: A ≡ x + aN⋆C(mod N)
Therefore, we have reduced the problem of finding x mod N to the problem of finding A mod N,

which can be efficiently solved using the private key (p, q, a) by computing the following: u ≡ a^(-1)(mod q), v ≡ a^(-1)(mod p), w ≡ A^u(mod q), z ≡ B^v(mod p), x ≡ (vNw + uNz) mod N^2. This gives us the value of x mod N, as required. Therefore, we have shown that the discrete log problem "mod N^2 base g" is easy when knowing the private key.

(2)

Show that given the public and private key, decrypting $C = g^m h^N$ mod N^2 can be done efficiently. Hint: consider $C^{\phi(N)}$ mod N^2. Use the fact by Euler's theorem $x^{\phi(N2)} = 1$ mod N^2 for any x.

My solution:

Given the public and private key, we can decrypt C = gmhN mod N2 efficiently using the following steps:

1. Compute ϕ(N) = (p-1)(q-1), where p and q are the prime factors of N.
2. Compute C' = C^ϕ(N) mod N^2.
3. Since C' ≡ (gmhN)^ϕ(N) ≡ g^(mϕ(N)) (mod N^2), we can recover mϕ(N) mod N using the discrete log problem "mod N2 base g" which we have shown to be easy when knowing the private key.
4. Finally, we can recover m by computing m ≡ (mϕ(N) * (ϕ(N))^(−1)) (mod N).

Here are detailed steps:

1) compute ϕ(N) = (p-1)(q-1) since p and q are prime numbers.
2) compute λ(N) = lcm(p-1, q-1) = ϕ(N) if p and q are distinct, or λ(N) = ϕ(N)/2 if p = q. This is the Carmichael function of N.
3) compute u = h^λ(N) mod N^2 using the fast modular exponentiation algorithm. Since g and N are coprime, we have g^ϕ(N) ≡ 1(mod N), and by Euler's theorem, we have g^ϕ(N) ≡ 1(mod p) and g^ϕ(N) ≡ 1(mod q). Therefore, we have: g^ϕ(N) ≡ 1(mod λ(N)). Then, we have:

   C^λ(N) = (g^mh^N)^λ(N) mod N^2
   = g^(mλ(N)) * h^(Nλ(N)) mod N^2
   = g^(mλ(N)) * u^(N*(ϕ(N)/λ(N))) mod N^2
   = g^(m*λ(N)) mod N^2
4) multiplying both sides by C^(-λ(N)) gives: C^(-λ(N)) * C^λ(N) = g^(m*λ(N)) * C^(-λ(N)) mod N^2
   Since C and λ(N) are known, we can compute C^(-λ(N)) using the extended Euclidean algorithm to find the multiplicative inverse of C mod N^2. Therefore, we obtain: g^(m*λ(N)) = C^(-λ(N)) * C^λ(N) mod N^2
5) then compute m*λ(N) mod λ(N^2) by taking the discrete logarithm base g of both sides, which is easy to do using the private key (p, q, a).
6) Finally, we can recover the plaintext message m by computing m = (m*λ(N) mod λ(N^2)) / (λ(N) mod N).

This algorithm allows us to efficiently decrypt C = gmhN mod N2 when knowing the public and private key.

(3)

Show that this encryption scheme is additive homomorphic. Let x, y, z be integers in [1,N]. Show that given the public key <N, g> and ciphertexts of a and b it is possible to construct a ciphertext of x + y and a ciphertext of zx. More precisely, show that given ciphertexts $C1 = g^x h_1^N, C2 = g^y h_2^N$, it is possible to construct ciphertexts $C_3 = g^{x+y} h_3^N$ and $C_4 = g^{zx} h_4^N$.

My solution:
Given the public key <N, g> and ciphertexts of a and b it is possible to construct a ciphertext of x + y and a ciphertext of zx. Given ciphertexts C1 = g^x * h_1^N and C2 = g^y * h_2^N, we can construct ciphertexts C3 = g^(x+y) * h_3^N and C4 = g^(zx) * h_4^N as follows:
To construct C3 = g^(x+y) * h_3^N, we can compute C3 = C1 * C2 mod N^2.
Since C1 * C2 ≡ (g^x * h_1^N) * (g^y * h_2^N) ≡ g^(x+y) * (h_1*h_2)^N (mod N^2), we have constructed a valid ciphertext for x + y.
To construct C4 = g^(zx) * h_4^N, we can compute C4 = (C1)^z mod N^2.
Since (C1)^z ≡ (g$^{x)}$z*(h_1)^zN ≡ g$^{(zx)*(h_1)}$zN(mod N^2), we have constructed a valid ciphertext for zx.

Let h_3 = h_1 * h_2 mod N^2, and let h_4 = (h_1^z) mod N^2.
Then we have:
C3 = g^(x+y) * h_3^N = (g^x * g^y) * (h_1 * h_2)^N
= (g^x * h_1^N) * (g^y * h_2^N) = C1 * C2

C4 = g^(zx) * h_4^N = g^(zx) * (h_1^z)^N mod N^2
= (g^z)^x * h_1^(zN) mod N^2
= (g^x * h_1^N)^z mod N^2 = C1^z

Therefore, we have shown that given the public key <N, g> and ciphertexts C1 = g^x * h_1^N and C2 = g^y * h_2^N, we can construct ciphertexts C3 = g^(x+y) * h_3^N and C4 = g^(zx) * h_4^N for some integers x, y, and z. This proves that the encryption scheme is additive homomorphic.