Homework 6

Task 1 - RSA Modulus Generation (10 points)

(a)

Given that $N = PQ$, $P = R - i$ and $Q = R + j$, where R is a k-bit integer and (since primes are dense) i, j are relatively small integers, we have two scenarios:
1. If R is not a prime, then P and Q are neighboring primes in the set of k-bit numbers.
2. If R is a prime, then P and Q are immediate prime neighbors of R.

In both scenarios, since P and Q are close to each other, the value of N can be efficiently factorized. One approach can be to search for primes near the square root of N, which would be approximately $\sqrt{N} \approx R$. This search is feasible in polynomial time relative to k. Once an attacker can factor N and find P and Q, they can efficiently compute the private key by using the public key $PK(N, e)$ and looking for a decryption exponent d in $\mathbb{Z}^*_{(P-1)(Q-1)}$, such that $ed = 1 \pmod{(P-1)(Q-1)}$.

The proximity of P and Q ($|P - Q|$ is small) significantly reduces the complexity of the factorization problem. Since RSA security relies on the difficulty of factoring N, the method mentioned above of generating P and Q compromises the security of RSA modulus.

(b)

$$P = 351230145912301391230119331203122231987162381239182311193820 61$$
$$Q = 351230145912301391230119331203122231987162381239182311193824 47$$

```
from sympy import isprime
from timeit import default_timer as timer
import math

# The RSA modulus N
N = 123362615397576525683206910571962544945300500765564700092323336712076729023858866739705216165335280143754047119747057008326 7

def factor(N):
    # Approximate square root of N
    approx_sqrt_N = int(math.isqrt(N))

    # Search for prime factors near the square root of N
    for i in range(approx_sqrt_N, 1, -1):
        print("Trying i = ", i)
        if N % i == 0 and isprime(i):
            # double check
            if isprime(N // i) and i * (N // i) == N:
                return i, N // i

    print("Error: no factors found")
    return None, None

# Find P and Q
timer_start = timer()
P, Q = factor(N)
timer_end = timer()
print("P = ", P, "Q = ", Q, "Time = ", timer_end - timer_start)

```

Task 2 - ElGamal and DDH (15 points)

(a)

Decryption Algorithm:

procedure $\mathsf{Dec}(\mathsf{SK}, C = (C_1, C_2))$:
if $C_1^{\mathsf{SK}} = C_2$ **then**
    return 0
else
    return 1

Correctness when $b = 0$:
- During encryption, we have $C_1 = g^y$ and $C_2 = PK^y = (g^x)^y = g^{xy}$.
- During decryption, we have $C_1^{\mathsf{SK}} = (g^y)^x = g^{yx} = g^{xy} = C_2$.
- Thus, when $b = 0$, the decryption algorithm will always correctly output 0.

Correctness when $b = 1$:
- During encryption, we have $C_1 = g^y$ and $C_2 = g^z$, where $y \xleftarrow{\$} \mathbb{Z}_p$ and $z \xleftarrow{\$} \mathbb{Z}_p$.
- During decryption, we have $C_1^{\mathsf{SK}} = (g^y)^x = g^{yx}$. Since y and z are independently and uniformly chosen from $\mathbb{Z}_p$and group $\mathbb{Z}_p$ has prime order p, the probability that $g^{yx} = g^z$ (i.e., $C_1^{\mathsf{SK}} = C_2$) is

$$\Pr[yx \equiv z \pmod{p}] = \frac{1}{p}$$

- Thus, the probability of the decryption algorithm incorrectly return 0 is $\frac{1}{p}$, which is very small when the prime order p is sufficiently large.

Therefore, we have shown that the correctness requirement of the scheme may sometimes not hold, but only with very small probability.

(b)

The following oracle satisfies this, which is given access to $\mathsf{DDH}_b[\mathbb{G}, g]$ for $b \in \{0, 1\}$.

oracle $\mathsf{O}^{\mathsf{DDH}_b[\mathbb{G},g]}$:

private **procedure** Init(X, Y, Z): // $(X, Y, Z) \leftarrow \mathsf{DDH}_b[\mathbb{G}, g]$
queried $\leftarrow 0$
return X

public **procedure** Encrypt():
if queried $= 1$ **return** $\perp$
$C \leftarrow (Y, Z)$; queried $\leftarrow 1$
return C

We can see that:
1) $O^{\mathsf{DDH}_0[\mathbb{G},g]} \equiv 1 - \mathsf{LR}_0[\Pi]$:
The Init procedure in oracle O takes input $(X, Y, Z) = (g^x, g^y, g^{xy})$ from $\mathsf{DDH}_0[\mathbb{G}, g]$. The Init procedure then returns X which is analogous to the public key $PK = g^x$ in the $1 - \mathsf{LR}_0[\Pi]$ oracle.
The Encrypt procedure in O outputs (Y, Z) which corresponds to (g^y, g^{xy}). In $1 - \mathsf{LR}_0[\Pi]$, when $b = 0$, the encryption algorithm outputs $(C_1, C_2) = (g^y, PK^y)$. Here, $PK^y = (g^x)^y = g^{xy}$, which matches the output of O under $\mathsf{DDH}_0[\mathbb{G}, g]$.
Therefore, $O^{\mathsf{DDH}_0[\mathbb{G},g]}$ is equivalent to $1 - \mathsf{LR}_0[\Pi]$.

2) $O^{\mathsf{DDH}_1[\mathbb{G},g]} \equiv 1 - \mathsf{LR}_1[\Pi]$:
The Init procedure takes $(X, Y, Z) = (g^x, g^y, g^z)$ from $\mathsf{DDH}_1[\mathbb{G}, g]$ and returns X, which serves as the public key PK.
The Encrypt() procedure in O under $\mathsf{DDH}_1[\mathbb{G}, g]$ outputs (Y, Z), which corresponds to (g^y, g^z). In $1 - \mathsf{LR}_1[\Pi]$, when $b = 1$, the encryption algorithm chooses random y, z in $\mathbb{Z}_p$ and outputs $(C_1, C_2) = (g^y, g^z)$ which matches the output of O under $\mathsf{DDH}_1[\mathbb{G}, g]$.
Therefore, $O^{\mathsf{DDH}_1[\mathbb{G},g]}$ is equivalent to $1 - \mathsf{LR}_1[\Pi]$.

(c)

Using the oracle O from part (b), let D by a polynomial-time distinguisher, then we have:

$$\begin{aligned}
\mathsf{Adv}^{\text{1-ind-cpa}}_{\Pi}(D) &= |\mathsf{Pr}[D^{\text{1-LR}_0[\Pi]} \Rightarrow 1] - \mathsf{Pr}[D^{\text{1-LR}_1[\Pi]} \Rightarrow 1]| \\
&= |\mathsf{Pr}[D^{O^{\mathsf{DDH}_0[\mathbb{G},g]}} \Rightarrow 1] - \mathsf{Pr}[D^{O^{\mathsf{DDH}_1[\mathbb{G},g]}} \Rightarrow 1]| \\
&= \mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G},g}(D') \qquad \text{// where } D' = D^{O^{\mathsf{DDH}_b[\mathbb{G},g]}}
\end{aligned}$$

Further, if D is polynomial time, because O's procedures all run in polynomial time, we also have that D' is polynomial time.
Then, because we assume DDH assumption holds for $\mathbb{G}$ with respect to g, $\mathsf{Adv}^{\mathsf{ddh}}_{\mathbb{G},g}(D)$ is negligible, which means that $\mathsf{Adv}^{\text{1-ind-cpa}}_{\Pi}(D')$ is also negligible (since the input of D' first passed through the Init procedure of O, which takes the output oracle $\mathsf{DDH}_b[\mathbb{G}, g]$ as input).
Therefore, we have shown that if DDH assumption holds for $\mathbb{G}$ with respect to g, then Π is one-time IND-CPA secure.

Task 3 - Chosen-Ciphertext Security (10 points)

(a)

To produce a modified ciphertext C' that encrypts $M \star \delta$ (for a known $\delta \in \mathbb{G}$), we first know that the encryption of $M \star \delta$ should be $(g^r, (M \star \delta) \star \mathsf{PK}^r)$ by the ElGamal scheme.
The operation $\star$ in cyclic group $\mathbb{G}$ is known to be associative and commutative, the expression $(M \star \delta) \star \mathsf{PK}^r$ can be rewritten as $(M \star \mathsf{PK}^r) \star \delta$
Thus, the modified ciphertext C' can be computed as:

$$C' = (g^r, (M \star \mathsf{PK}^r) \star \delta)$$

Since $C = (g^r, M \star \mathsf{PK}^r)$ is known, assuming the group operation $\star$ is efficiently computable, we can efficiently calculate the second component of C' as $(M \star \mathsf{PK}^r) \star \delta$.
Therefore, we have shown that we can efficiently produce the encryption of $M \star \delta$ given the encryption of M and a known δ.

(b)

We know that $M_1 \neq M_2$. Let's assume for contradiction that $M_1 \star \delta = M_2 \star \delta$.
Since $\delta \in \mathbb{G}$, by the inverse property of groups, we know there exists $\delta^{-1} \in \mathbb{G}$ such that $\delta \star \delta^{-1} = 1$.
Then, we can multiply both sides of the equation $M_1 \star \delta = M_2 \star \delta$ by δ^{-1} to get:

$$M_1 \star \delta \star \delta^{-1} = M_2 \star \delta \star \delta^{-1}$$

which, by the identity property of group, simplifies to:

$$M_1 = M_2$$

This contradicts the fact that $M_1 \neq M_2$.
Therefore, by contradiction, we have shown that $M_1 \star \delta \neq M_2 \star \delta$ (i.e. $M_1 \star \delta$ and $M_2 \star \delta$ are distinct).

(c)

An attack on the given ElGamal scheme is demonstrated by the following distinguisher D, which accesses the oracle $\mathsf{LRD}_b[\Pi]$, where Π is the ElGamal scheme.

distinguisher $D^{\mathsf{O}}(\mathsf{PK}, g)$:
$M_0 \leftarrow g^0$
$M_1 \leftarrow g^1$
$\delta \leftarrow g^2$
$(C_1, C_2) \leftarrow \mathsf{O.Encrypt}(M_0, M_1)$
$C' \leftarrow (C_1, C_2 \star \delta)$
$M' \leftarrow \mathsf{O.Decrypt(C')}$
if $M' = M_0 \star \delta$ **then**
    return 1
else return 0

By the above:
When $O = \mathsf{LRD}_0[\Pi]$, the oracle encrypts $M_0 = g^0$. As shown in part(a), C' will be the encryption of $M_0 \star \delta$, and the decryption oracle will return $M_0 \star \delta$, which is not equal to $M_1 \star \delta$ (as shown in part(b)). Thus, D will always return 1.

When $O = \mathsf{LRD}_1[\Pi]$, the oracle encrypts $M_1 = g^1$. As shown in part(a), C' will be an encryption of $M_1 \star \delta$, and the decryption oracle will return $M_1 \star \delta$, which is not equal to $M_0 \star \delta$ (as shown in part(b)). Thus, D will always return 0.

So we have, $\mathsf{Adv}^{\mathsf{ind\text{-}cca}}_{\Pi}(D) = |\mathsf{Pr}[D^{\mathsf{LRD}_0[\Pi]} \Rightarrow 1] - \mathsf{Pr}[D^{\mathsf{LRD}_1[\Pi]} \Rightarrow 1]| = |1 - 0| = 1$.
Therefore, we have shown that the ElGamal encryption scheme is not IND-CCA secure.

Task 4 - AES-Based Signatures (15 points)

(a)

Given only VK, an adversary does not have access to the secret key SK but only the AES encryptions of the secret key.
To forge a signature on any message M^*, the adversary would need to produce a sequence σ^* such that AES$(\sigma_i^*, 0^{128})$ equals C_{i,M_i^*} for each bit i in M^*. The adversary has two options:
1) Reverse the AES function to recover the secret key K_{i,M_i^*} from C_{i,M_i^*}.
2) Guess the secret key K_{i,M_i^*} for each bit i in M^*.

By our assumption, inverting the AES function to recover the correct K_{i,M_i^*} from C_{i,M_i^*}, is hard. Since the AES keys were chosen randomly and independently, the adversary gets no advantage in computing one key from another, so the problem remains hard for each bit of the message.
Thus, the first option is infeasible for "efficient" adversaries.

Moreover, the adversary cannot simply guess the keys since they are 128-bit strings (the search space is 2^{128}), which means the adversary will guess the correct key with very small probability. So, the second option is also infeasible for any "efficient" adversary.

Therefore, any "efficient" adversary cannot forge a signature σ^* for any message M^* such that Ver(VK, M^*, σ^*) outputs 1, which shows that the signature scheme Σ satisfies key-only unforgeability.

(b)

If an adversary is given a valid signature σ for a message M, they still face the same difficulty in part (a) for any bit of a new message $M^* \neq M$.

The reason is that each bit M_i of M is signed with a distinct key K_{i,M_i}. For any bit M_i^* of M^* that differs from M_i, the adversary needs the corresponding key K_{i,M_i^*} to produce a valid signature bit σ_i^*.

Since they only have K_{i,M_i} from the known signature σ, and due to the hardness assumption of AES, they cannot efficiently derive K_{i,M_i^*} from any of the known information.

Therefore, even with a valid signature for a chosen message, the adversary cannot generate a new valid signature for any different message because each bit of the signature is protected by an independent key that the adversary cannot efficiently compute.

(c)

To show that the scheme is not UF-CMA secure, consider the following adversary A, which accesses the oracle $\mathsf{UF}-\mathsf{CMA}[\Sigma]$, where Σ is the AES-based signature scheme proposed in the question.

adversary $A^{O}()$:
$(\sigma_1, ..., \sigma_n) \leftarrow \mathsf{O.Eval}(0^n)$
$(\sigma'_1, ..., \sigma'_n) \leftarrow \mathsf{O.Eval}(1^n)$
$M^* = (1||0^{n-1})$ // only flipping the first bit of 0^n
$\sigma^* \leftarrow (\sigma'_1, \sigma_2, ..., \sigma_n)$ // only replacing σ_1 in σ with σ'_1
return M^*, σ^*

Since M^* is different from both 0^n and 1^n, M^* is unknown to the oracle (i.e., $M^* \notin \mathcal{Q}$).
Since the first bit of M^* matches with 1^n, $\sigma^*_1 = \sigma'_1$ is a valid first signature bit for M^*.
Since all other bits in M^* matches with 0^n, $\sigma^*_i = \sigma_i$ $(1 < i \leq n)$ is a valid signature bit for M^*.
Thus, σ^* is a valid signature for M^*.

The finalization procedure in the oracle will return 1 when the adversary returns M^* and σ^* because for each bit i, $\mathsf{AES}(\sigma^*_i, 0^{128}) = C_{i,M^*_i}$
Therefore, $\mathsf{Adv}^{\mathsf{uf\text{-}cma}}_{\Sigma}(A) = 1$, and A successfully forges a signature, which shows that the scheme Σ is not UF-CMA secure.