Cadet Justin Nguyen

MA464 WPR 2 (200 Points) DUE April 5th at MIDNIGHT

READ THESE INSTRUCTIONS CAREFULLY BEFORE STARTING WORK.

1. Including this cover page, there are 9 pages to this exam.
2. **Show your work.**

Question	Points	Score
1	20	
2	30	
3	30	
4	30	
5	20	
6	20	
7	30	
8	20	
Total:	200	

Use this page for scratch work.

1. (20 points) Many practical cryptosystems utilize private key algorithms as well as public key ones. What is the advantage of using such hybrid systems?

Allows for secure communication over an unsecure channel such as the internet. The sender can encrypt the symmetric key using the recipients public key. Sends the encrypted key along w/ the encrypted message. The recipient can then decrypt the symmetric key using their own private key and use it to decrypt the message. Ensures that even if an attacker intercepts the message, they can't decipher unless they know the recipients' private key.

2. (30 points) Given are the following parameters of the RSA cryptosystem $p = 137$, $q = 131$, $n = 17947$, and $d = 11787$ compute e and encrypt $x = 513$.

$\text{phi} = (p-1)(q-1)$

$e = d^{-1} \bmod (\text{phi})$

↓

$e = 11787^{-1} \bmod 17680$

to encrypt:

$\text{Ciphen} = m^e \bmod n$

$\text{ciphen} = 513^e \bmod 17947$

$\text{Cipher} = 8363$

```python
def calculate_encryption_exponent(d, p, q, n):
    phi = (p-1) * (q-1)
    e = pow(d, -1, phi)
    return e

def rsa_encrypt(m, e, n):
    c = pow(m, e, n)
    return c

e_e = calculate_encryption_exponent(11787, 137, 131, 17947)
print("e:", e_e)
c = rsa_encrypt(513, e_e, 17947)
print("c:", c)
```

```
OUTPUT  DEBUG CONSOLE  TERMINAL  SQL CONSOLE  ...  C/C++ Compile Run

❯ python3 q2.py
e: 3
c: 8363
```

3. (30 points) Given are the following parameters of the RSA cryptosystem $p = 137$, $q = 131$, $n = 17947$, $d = 11787$, $e = 3$ explain all the steps and write down explicitly the formula(Do not compute!) how to decrypt the message $y = 8363$ using Chinese Remainder Theorem when we know that $y_q = 6$, $d_q = 87$ and $y_q^{d_q} = 120$.

First, calculate the two private exponents corresponding to the prime factors p & q.

$d_p = e^{-1} \bmod (p-1)$
$d_q = e^{-1} \bmod (q-1)$
} uses extend euclidean algo

$d_p = 3^{-1} \bmod 136 \rightarrow = 91 \bmod 136$

$d_q = 3^{-1} \bmod 130 \rightarrow = 87 \bmod 130$

CRT to calculate message m:

$m_p = y_p^{d_p} \bmod p$

$m_q = y_q^{d_q} \bmod q$

$q^{-1} \equiv q^{(p-2)} \pmod p$

$q^{-1} \equiv 131^{135} \bmod 137$

$m_p = (y \bmod p)^{d_p} \bmod p$

$m_q = y_q^{d_q} = 120$

Combine:

$m = m_p \cdot q \cdot q^{-1} + m_q \cdot p \cdot p^{-1} \pmod n$

where

$p^{-1} \equiv p^{(q-2)} \pmod q$ & $q^{-1} \equiv q^{(p-2)} \pmod p$

4. (30 points) State and solve (if possible) the discreet logarithm problem for the following:

i) $2^x \equiv 137 \mod 467$

```python
def discrete_log(a, b, n):
    """
    Brute-force method for solving the discrete logarithm problem.
    Finds an integer x such that a^x ≡ b (mod n).
    """
    x = 0
    while pow(a, x, n) != b:
        x += 1
    return x

a = 2
b = 137
n = 467

x = discrete_log(a, b, n)

print("the discrete log is: ", x)
```

```
❯ python3 q4.py
the discrete log is:  400
```

$x = 400$

AND

ii) Let $G = (\mathbb{Z}_{11}, +)$ and $\alpha = 2$ a primitive element. Solve the DLP for $\beta = 3$.

```python
def discrete_log(a, b, n):
    """
    Brute-force method for solving the discrete logarithm problem.
    Finds an integer x such that a^x ≡ b (mod n).
    """
    x = 0
    while pow(a, x, n) != b:
        x += 1
    return x

a = 2
b = 3
n = 11

x = discrete_log(a, b, n)

print("the discrete log is: ", x)
```

```
❯ python3 q4_2.py
the discrete log is:  8
```

DLP is 8

AND

iii) Let $(E_{17}, \oplus)$ be the elliptic curve given by the equation $y^2 \equiv x^3 + 2x + 2$ and $P = (5, 1)$ a primitive element. State and solve (if possible) the DLP for $P = (5, 1)$ and $Q = (16, 4)$.

```python
# Define the point addition operation
def add(P, Q):
    if P is None:
        return Q
    if Q is None:
        return P
    x1, y1 = P
    x2, y2 = Q
    if x1 == x2 and (y1 + y2) % 17 == 0:
        return None
    if x1 == x2:
        m = (3*x1**2 + 2) * pow(2*y1, -1, 17) % 17
    else:
        m = (y2 - y1) * pow(x2 - x1, -1, 17) % 17
    x3 = (m**2 - x1 - x2) % 17
    y3 = (m*(x1 - x3) - y1) % 17
    return (x3, y3)

# Define the point multiplication operation
def mul(k, P):
    Q = None
    while k > 0:
        if k % 2 == 1:
            Q = add(Q, P)
        P = add(P, P)
        k //= 2
    return Q

# Solve the DLP using brute force
P = (5, 1)
Q = (16, 4)
kP = P
x = 1
while kP != Q:
    x += 1
    kP = add(kP, P)
print(f"The solution is x = {x}")
```

```
[Done] exited with code=0 in 0.036 seconds

[Running] python -u "/Users/justinnguyen/Documents/AY2023_2/MA464/WPR2/q4_iii.py"
The solution is x = 13

[Done] exited with code=0 in 0.051 seconds
```

DLP is 13

AND

5. (20 points) How does Diffie Hellman differ from elliptic curve Diffie Hellman? How do you think this difference affects security?

DH uses DLP over a finite field while ECDH uses DLP over eliptic curves defined over a finite field. Elliptic curves comprable security with much smaller key sizes, making it much more efficient. You cannot use Number field Sieve algorithm on elliptic curves. This is due to the fact that it is generally harder to compute discrete logarithms over elliptic curves.

6. (20 points) Determine all points on the elliptic curve $E : y^3 \equiv x^3 - 2x$ over $\mathbb{Z}_5$.

x	$x^3 - 2x$	solutions
0	0	(0,0)
1	-1	(1,2), (1,3)
2	4	(2,0)
3	21	(3,2), (3,3)
4	60	(4,1), (4,4)

7. (30 points) Let E be the elliptic curve determined by the cubic equation $y^2 \equiv x^3 - 4x$ over $\mathbb{Z}_7$ Suppose Alice and Bob choose the ellitpic curve E and the point $\alpha = (1, 2)$ publicly for their Elliptic Curve Key Exchange. If Alice's secret key is $y = 3$ and Bob sends Alice $x\alpha = (3, 1)$ where x is Bob's secret key, calculate their shared key.

To find shared key, find $y \cdot x\alpha$, so it is $3 \cdot (3,1)$

```python
def elliptic_curve_addition(x1, y1, x2, y2, p):
    if x1 == x2 and y1 == -y2 % p:
        return None, None

    if (x1, y1) == (x2, y2):
        slope_num = (3 * x1 ** 2 - 4)
        slope_denom = (2 * y1)
    else:
        slope_num = (y2 - y1)
        slope_denom = (x2 - x1)
    inverse_denom = pow(slope_denom, -1, p)
    slope = slope_num * inverse_denom % p

    x3 = (slope ** 2 - x1 - x2) % p
    y3 = (slope * (x1 - x3) - y1) % p

    return x3, y3

def scalar_mult(point, scalar, p):
    new_point = point
    for _ in range(scalar - 1):
        new_point = elliptic_curve_addition(new_point[0], new_point[1], point[0],
    return new_point

def main():
    E_p = 7  # Curve parameter p
    alpha = (1, 2)  # Public chosen point

    alice_secret_key = 3
    bob_pub_key = (3, 1)

    result = scalar_mult(bob_pub_key, alice_secret_key, E_p)

    print("Shared key:", result)

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

```
OUTPUT  DEBUG CONSOLE  TERMINAL  ...  Code

[Done] exited with code=1 in 0.052 seconds

[Running] python -u "/Users/justinnguyen/Documents/AY2023_2/MA464/WPR2/q7.py"
Shared key: (3, 6)
```

Shared key = (3,6)

8. (20 points) The main operation in elliptic curve cryptosystems is the multiplication of a point P on the curve by an integer. Show the steps that are being performed for calculating $136P$ through an adaption of the square and multiply algorithm.

Convert 136 to binary → 10001000

for each binary representation of the scalar, point doubling $Q = 2Q$, if bit is 1, perform point addition: $Q = Q + P$

$Q = \infty$

1 $Q = Q + P = P$

0 $Q = 2Q = 2P$

0 $Q = 2Q = 4P$

0 $Q = 2Q = 8P$

1 $Q = 2Q = 16P$, update $Q : Q = Q + P = 17P$

0 $Q = 2Q = 34P$

0 $Q = 2Q = 68P$

0 $Q = 2Q = \underline{136P}$

— ANS