Simonas Riška, ITSfm-24

Public Key Cryptography Lab: Diffie-Hellman, RSA, and ElGamal

Parameters:

A - 36

B - 20

Task I: Diffie Hellman

- *Choose prime number p>30*

  I chose p = 37, which is a prime number greater than 30.

  Answer: p = 37

- *Choose prime root a and prove that it is a prime root using brute force. (more effective method using factorization of Euler totient function psi(p)=p-1 can be applied)*

  I chose $a = 2$ as a candidate for the primitive root of $p = 37$. Then, I used brute force to verify that a is a primitive root by checking if $a^k \ mod\ p$ generates all integers from 1 to $p - 1$ for $k = 1$ to $p - 1$. This was done by calculating the powers of $a \ mod\ p$ and confirming that the results form a complete set of residues modulo p.

  For $a = 2$ and $p = 37$, the results of $2^k \ mod\ 37$ for $k = 1$ to 36 I wrote a Python code:

calculate-modulo-powers.py

C: > Users > simon > Desktop > Cryptography > calculate-modulo-powers.py > ...

```python
def calculate_modulo_powers(a, p, k_max):
    results = [pow(a, k, p) for k in range(1, k_max + 1)]
    print(" ".join(map(str, results)))
    if sorted(set(results)) == list(range(1, k_max + 1)):
        print("The results cover all integers from 1 to", k_max)
    else:
        print("The results do NOT cover all integers from 1 to", k_max)

a = 2
p = 37
k_max = 36

if __name__ == "__main__":
    calculate_modulo_powers(a, p, k_max)
```

The output is this:

```
C:\Users\simon\Desktop\Cryptography>python calculate-modulo-powers.py
2 4 8 16 32 27 17 34 31 25 13 26 15 30 23 9 18 36 35 33 29 21 5 10 20 3 6 12 24 11 22 7 14 28 19 1
The results cover all integers from 1 to 36
```

Answer: Since these values include all integers from 1 to 36, a = 2 is confirmed as a primitive root of p = 37.

- *Using as secret keys A and B calculate public keys PubA, PubB and joint secret key K using Diffie Hellman algorithm*

$$PubA = a^A \bmod p = 2^{36} \bmod 37 = 1$$
$$PubB = a^B \bmod p = 2^{20} \bmod 37 = 33$$
$$K = PubB^A \bmod p = 33^{36} \bmod 37 = 1$$

Here's Python code that I wrote to check:

diffie-hellman.py ×

C: > Users > simon > Desktop > Cryptography > diffie-hellman.py > ...

```python
p = 37
a = 2
A = 36
B = 20

PubA = pow(a, A, p)
PubB = pow(a, B, p)
K = pow(PubB, A, p)

print(f"Public Key A (PubA): {PubA}")
print(f"Public Key B (PubB): {PubB}")
print(f"Joint Secret Key (K): {K}")
```

Here's the output of this Python code:

```
C:\Users\simon\Desktop\Cryptography>python diffie-hellman.py
Public Key A (PubA): 1
Public Key B (PubB): 33
Joint Secret Key (K): 1
```

Answer: Public key A = 1, public key B = 33, joint secret key = 1.

Task II: RSA

- *Choose two prime numbers p>30 and q>30*

  I chose $p = 31$ and $q = 37$ – prime numbers greater than 30.

  Answer: $p = 31, q = 37$

- *Calculate n=pq and Euler totient function psi(n)*

$$n = p \times q = 31 \times 37 = 1147$$
$$\varphi(n) = (p - 1) \times (q - 1) = (31 - 1) \times (37 - 1) = 30 \times 36 = 1080$$

  Here's Python code that I wrote to check:

```
rsa.py
C: > Users > simon > Desktop > Cryptography > rsa.py > ...
p = 31
q = 37

n = p * q
psi_n = (p - 1) * (q - 1)

print(f"Modulus (n): {n}")
print(f"Euler's Totient Function (ψ(n)): {psi_n}")
```

  Here's the output of this Python code:

```
C:\Users\simon\Desktop\Cryptography>python rsa.py
Modulus (n): 1147
Euler's Totient Function (ψ(n)): 1080
```

  Answer: $n = 1147, \varphi(n) = 1080$

- *Choose encryption key e such that GCD(e,psi)=1*

  I selected $e = 7$ as the encryption key. To confirm that is it valid:

$$GCD(7{,}1080) = 1$$

  This ensures that $e = 7$ is valid since it is coprime with $\varphi(n) = 1080$

Here’s Python code that I wrote to check:

```
rsa.py
C: > Users > simon > Desktop > Cryptography > rsa.py > ...
from math import gcd

p = 31
q = 37

n = p * q
psi_n = (p - 1) * (q - 1)

e = 3
while gcd(e, psi_n) != 1:
    e += 2

print(f"Encryption Key (e): {e}")
```

Here’s the output of this Python code:

```
C:\Users\simon\Desktop\Cryptography>python rsa.py
Encryption Key (e): 7
```

Answer: $Encryption\ Key\ (e)\ =\ 7$

- *Calculate decryption key d*

The decryption key d is the modular inverse of $e\ mod\ \varphi(n)$. This means that:

$$e\ \times d\ \equiv 1\ mod\ \varphi(n)$$

To find d, I used the Extended Euclidean Algorithm.

$$a = 1080, b = 7$$

First, I performed a division to calculate the quotient and remainder:

$$1080 \div 7 = 154 \quad remainder: 2$$

Then I repeated the division with new values:

$$a = 7, b = 2$$

$$7 \div 2 = 3 \quad remainder: 1$$

And then again:

$$a = 2, b = 1$$

$$2 \div 1 = 2 \quad remainder: 0$$

Then I worked backwards to find coefficients:

$$1 = 7 - 3\ \times 2$$

I substituted $2 = 1080 - 154\ \times 7$:

$$1 = 7 - 3\ \times (1080 - 154\ \times 7) = 463\ \times 7 - 3 \times 1080 = 463$$

I wrote this Python code to check:

rsa.py

C: > Users > simon > Desktop > Cryptography > rsa.py > ...

```python
from math import gcd

p = 31
q = 37

n = p * q
psi_n = (p - 1) * (q - 1)

e = 3
while gcd(e, psi_n) != 1:
    e += 2

def modular_inverse(a, m):
    m0, x0, x1 = m, 0, 1
    while a > 1:
        q = a // m
        m, a = a % m, m
        x0, x1 = x1 - q * x0, x0
    return x1 + m0 if x1 < 0 else x1

d = modular_inverse(e, psi_n)

print(f"Decryption Key (d): {d}")
```

Here’s the output of the Python code:

```
C:\Users\simon\Desktop\Cryptography>python rsa.py
Decryption Key (d): 463
```

Answer: $Decryption\ key\ (d) = 463$

- *Encrypt A (obtain cyphertext C) with RSA using above parameters*

The ciphertext C is calculated as:

$$C = A^e \ mod \ n$$

Substitute the values:

$$C = \ 36^7 \ mod \ 1147 = 36$$

Here’s the Python code that I wrote to check:

rsa.py

C: > Users > simon > Desktop > Cryptography > rsa.py > ...

```python
from math import gcd

A = 36

p = 31
q = 37

n = p * q
psi_n = (p - 1) * (q - 1)

e = 3
while gcd(e, psi_n) != 1:
    e += 2

def modular_inverse(a, m):
    m0, x0, x1 = m, 0, 1
    while a > 1:
        q = a // m
        m, a = a % m, m
        x0, x1 = x1 - q * x0, x0
    return x1 + m0 if x1 < 0 else x1

d = modular_inverse(e, psi_n)

C = pow(A, e, n)

print(f"Ciphertext (C): {C}")
```

Here's the output of the Python code:

```
C:\Users\simon\Desktop\Cryptography>python rsa.py
Ciphertext (C): 36
```

Answer: $Ciphertext\ (C) = 36$

- *Demonstrate that decryption with key d gives A*

The decrypted message A is calculated as:

$$A = C^d \bmod n$$

After substituting values:

$$A = 36^{463} \bmod 1147 = 36$$

Here's the Python code that I wrote to check:

rsa.py

C: > Users > simon > Desktop > Cryptography > rsa.py > ...

```python
from math import gcd

A = 36

p = 31
q = 37

n = p * q
psi_n = (p - 1) * (q - 1)

e = 3
while gcd(e, psi_n) != 1:
    e += 2

def modular_inverse(a, m):
    m0, x0, x1 = m, 0, 1
    while a > 1:
        q = a // m
        m, a = a % m, m
        x0, x1 = x1 - q * x0, x0
    return x1 + m0 if x1 < 0 else x1

d = modular_inverse(e, psi_n)

C = pow(A, e, n)

decrypted_A = pow(C, d, n)

print(f"Original Message (A): {A}")
print(f"Decrypted Message: {decrypted_A}")
```

Here's the output of the Python code:

```
C:\Users\simon\Desktop\Cryptography>python rsa.py
Original Message (A): 36
Decrypted Message: 36
```

Answer: The decrypted message matches the original message A = 36

Task III: El-Gamal signature

- *Use the same prime number p, primitive root a, private key A and public key PubA*

Answer: Prime number p = 37, primitive root a = 2, private key A = 36, public key PubA = 1.

- *Choose one-time key k such that GCD(k,p-1)=1 and calculate inverse k mod(p-a)*

Here, p – 1 = 36. I chose k = 5, because:

$$GCD(5{,}36) = 1$$

I wrote this Python code to check:

el-gamal.py ×

C: > Users > simon > Desktop > Cryptography > el-gamal.py > ...

```python
from math import gcd

p = 37
a = 2
A = 36
B = 20

PubA = pow(a, A, p)

k = 2
while gcd(k, p-1) != 1:
    k += 1

print(f"One-Time Key (k): {k}")
```

I got this output:

```
C:\Users\simon\Desktop\Cryptography>python el-gamal.py
One-Time Key (k): 5
```

To calculate k^{-1} mod 36, I used the Extended Euclidean Algorithm.

$$a = 36, b = 5$$
$$36 \div 5 = 7 \quad remainder: 1$$
$$5 \div 1 = 5 \quad remainder: 0$$

Then I worked backwards to find coefficients. From the Euclidean algorithm:

$$1 = 36 - 7 \times 5$$
$$1 \equiv -7 \times 5 \; mod \; 36$$
$$k^{-1} \equiv 29 \; mod \; 36$$
$$k = 5, k^{-1} = 29$$

I wrote this Python code to check:

el-gamal.py

C: > Users > simon > Desktop > Cryptography > el-gamal.py > ...

```python
from math import gcd

p = 37
a = 2
A = 36
B = 20

PubA = pow(a, A, p)

k = 2
while gcd(k, p-1) != 1:
    k += 1

def modular_inverse(a, m):
    m0, x0, x1 = m, 0, 1
    while a > 1:
        q = a // m
        m, a = a % m, m
        x0, x1 = x1 - q * x0, x0
    return x1 + m0 if x1 < 0 else x1

k_inverse = modular_inverse(k, p-1)

print(f"Inverse of k mod (p-1): {k_inverse}")
```

I got this output:

```
C:\Users\simon\Desktop\Cryptography>python el-gamal.py
Inverse of k mod (p-1): 29
```

Answer: $k = 5, k^{-1} = 29$

- *Sign as a document parameter B - calculate signature values S1 and S2 (formulas from slides or W.Stallings book page 421)*

To sign $B = 20$, I used the following formulas and calculations:

$$S_1 = a^k \; mod \; p = 2^5 \; mod \; 37 = 32$$

$$S_2 = k^{-1} \times (B - A \times S_1) \; mod \; (p-1) = 29 \times (20 - 36 \times 32) \; mod \; 36 = 4$$

$$S_1 = 32, S_2 = 4$$

I wrote this Python code to check:

el-gamal.py ×

C: > Users > simon > Desktop > Cryptography > el-gamal.py > ...

```python
from math import gcd

p = 37
a = 2
A = 36
B = 20

PubA = pow(a, A, p)

k = 2
while gcd(k, p-1) != 1:
    k += 1

def modular_inverse(a, m):
    m0, x0, x1 = m, 0, 1
    while a > 1:
        q = a // m
        m, a = a % m, m
        x0, x1 = x1 - q * x0, x0
    return x1 + m0 if x1 < 0 else x1

k_inverse = modular_inverse(k, p-1)

S1 = pow(a, k, p)
S2 = (k_inverse * (B - A * S1)) % (p-1)

print(f"Signature Values: S1 = {S1}, S2 = {S2}")
```

I got this output:

```
C:\Users\simon\Desktop\Cryptography>python el-gamal.py
Signature Values: S1 = 32, S2 = 4
```

Answer: $S_1 = 32, S_2 = 4$

- *Verify the signature using only S1 S2 and public parameters p, a and PubA*

To verify, I calculated:

$$v_1 = \left(PubA^{S_1} \times S_1^{S_2}\right) mod\ p = (1^{32} \times 32^4)\ mod\ 37 = (32^4)\ mod\ 37 = 33$$

$$v_2 = a^B\ mod\ p = 2^{20}\ mod\ 37 = 33$$

$$v_1 = v_2 = 33$$

I wrote this Python code to check:

el-gamal.py

C: > Users > simon > Desktop > Cryptography > el-gamal.py > ...

```python
from math import gcd

p = 37
a = 2
A = 36
B = 20

PubA = pow(a, A, p)

k = 2
while gcd(k, p-1) != 1:
    k += 1

def modular_inverse(a, m):
    m0, x0, x1 = m, 0, 1
    while a > 1:
        q = a // m
        m, a = a % m, m
        x0, x1 = x1 - q * x0, x0
    return x1 + m0 if x1 < 0 else x1

k_inverse = modular_inverse(k, p-1)

S1 = pow(a, k, p)
S2 = (k_inverse * (B - A * S1)) % (p-1)

v1 = (pow(PubA, S1, p) * pow(S1, S2, p)) % p
v2 = pow(a, B, p)

print(f"Verification Results: v1 = {v1}, v2 = {v2}, Verified = {v1 == v2}")
```

I got this output:

```
C:\Users\simon\Desktop\Cryptography>python el-gamal.py
Verification Results: v1 = 33, v2 = 33, Verified = True
```

Answer: $v_1 = v_2 = 33$, the signature is valid.