IDATT2503 - Cryptography Assigment 4

Edvard Berdal Eek

November 2025

Task 1

Factorize $n = 275621053$

Using Fermat's factorization in Python:

```
def fermat(n):
  if n & 1 == 0:
    return (2, n/2)
  a = math.ceil(math.sqrt(n))
  if a * a == n:
    return (a, a)
  a += 1
  for _ in range(n):
    b_squared = a * a - n
    b = math.ceil(math.sqrt(b_squared))
    if b * b == b_squared:
      return (a + b, a - b)
    a += 1
  return None
```

This returned the (17021, 16193) as the factors.

Task 2

a) Explain why Alice should use $q = 2027$ for the RSA system to work and to be most secure

Alice chose $p = 1283$ and $d = 3$. For RSA to work we need d to have multiplicative inverse e so that $e \cdot d \equiv 1 \bmod (p-1)(q-1)$. We also need that their $\gcd(d, (p-1)(q-1)) = 1$, which means that with $d = 3$, $(p-1)(q-1)$ must not be divisible by 3.

Using Euclidean algorthim i found that $q = 1879$ was did not result in a $\gcd(d, (p-1)(q-1)) = 1$ so this would not work in RSA, however the others do.

With $q = 1307$ the difference between p and q is very small $p - q = 24$ which would make it vulnerable to Fermat's factorization.

$q = 2003$ has $q - 1 = 2002 = 2 \cdot 7 \cdot 11 \cdot 13$, i.e product of small prime factors. Pollard's p-1 might more easily reveal the value of p as $B \geq 13$ means that $B!$ doesnt have to be that large and is therefore more easy to compute.

Leaving $q = 2027$ as the best choice as it is not as vulnerable to Fermat's factorization or Pollard's p-1.

b) Find the corresponding public key e using the extended Euclidean algorithm. Write a program to do the calculation.

I used the Python implementation of Euclidean extended from the lecture, and found the corresponding public key $e = 1731555$.

The function returned a negative value for e so i had to do extract e from the returned value x using $e = x \bmod (p-1)(q-1)$

c) Encrypt the message 111 using repeated squaring. Implement the algorithm yourself.

I implemented the following function in Python:

```
def mod_exp(m, e, n):
    result = 1
    m = m % n
    while e > 0:
        if e & 1:
            result = (result * m) % n
        m = (m * m) % n
        e >>= 1
    return result
```

This produced the ciphertext 1509208 from 111:

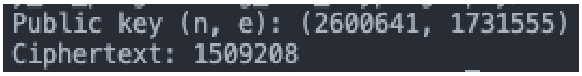

Figure 1: RSA Encryption of 111

Task 3

a) Let $n = 1829$ and $B = 5$. Find a prime factor of n by using Pollard (p - 1) attack.

```
def pollard(n, B):
    a = 2
    B_factorial = math.factorial(B)
    A = a**B_factorial % n

    gcd, _, _ = extended_Euclid(A - 1, n)
    if 1 < gcd < n:
        return gcd
    return None
```

Using the pollard python function and the python functions previously defined i found the value for $gcd(A-1, n) = 31$ with $A = a^{B!} \bmod n = 311$.

Pollard p - 1 returns the non-trivial factor 31.

b) Let $n = 18779$. Using Pollard (p - 1), how small B can be used for the attack to be successful.

Using a simple for loop i tested Pollard on different values of B incrementing by 1 for each iteration until a factor was returned.

Figure 2: Pollard Factor

```python
n = 18779
B = 0
while True:
  B += 1
  f = pollard(n, B)
  if f == None:
    continue
  break
```

Figure 3: B = 7

Task 4

a) Show that encryption in RSA has the following property, $e_K(x_1)e_K(x_2) \bmod n = e_K(x_1x_2) \bmod n$:

Encryption with RSA is with the public key $K = (n, e)$:

$$e_K(x) = x^e \bmod n$$

Taking two messages x_1 and x_2 we get:

$$e_K(x_1)e_K(x_2) \bmod n = (x_1^e \bmod n)(x_2^e \bmod n) \bmod n$$

$$e_K(x_1)e_K(x_2) \bmod n = x_1^e x_2^e \bmod n$$

$$e_K(x_1)e_K(x_2) \bmod n = (x_1x_2)^e \bmod n$$

$$e_K(x_1)e_K(x_2) \bmod n = e_K(x_1x_2) \bmod n$$

b) Show how RSA is vulnerable to chosen cipher text attack:

$$x' = d_K(y') = (y')^d \bmod n$$
$$x' = (y \cdot r^e)^d \bmod n = y^d \cdot r^{ed} \bmod n$$

Eulers theorem tells us that if $x \equiv y \bmod \phi(n)$, then $a^x \equiv a^y \bmod n$. Therefore, $r^{ed} \equiv r \bmod n$ because $ed \equiv 1 \bmod \phi(n)$. We can also know that from the definition of RSA encryption $x = y^d$, so we are now left with:

$$x' = x \cdot r \bmod n$$

We multiply with the multiplicative inverse of r so that we are left with a way to compute the plaintext x:

$$x \equiv x' \cdot r^{-1} \bmod n$$