Cryptography Exercises Report

Epameinondas Bakoulas

November 2025

1 Exercise 1: Vigenère Cipher

1.1 Finding the Key

We organize the ciphertext `LXFOPVEFRNHR` into 5 batches based on the key length of 5.

- **Batch 1:** Letters at indices 0, 5, 10 (`L`, `V`, `H`)
- **Batch 2:** Letters at indices 1, 6, 11 (`X`, `E`, `R`)
- **Batch 3:** Letters at indices 2, 7 (`F`, `F`)
- **Batch 4:** Letters at indices 3, 8 (`O`, `R`)
- **Batch 5:** Letters at indices 4, 9 (`P`, `N`)

Frequency table for each batch:

#	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
#1	0	0	0	0	0	0	0	**1**	0	0	0	**1**	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0
#2	0	0	0	0	**1**	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	0	0	0	**1**	0	0
#3	0	0	0	0	0	**2**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
#4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	0	**1**	0	0	0	0	0	0	0	0
#5	0	0	0	0	0	0	0	0	0	0	0	0	0	**1**	0	**1**	0	0	0	0	0	0	0	0	0	0

Table 1: Frequency distribution of letters in each batch

We analyze each batch to find the shift that maps the letters to high-frequency English letters (such as `e`, `t`, `a`, `o`, `i`, `n`).

1.1.1 Batch 3 Analysis

Letters: `F`, `F`

The letter `F` appears 100% of the time. If `F` (5) corresponds to the common letter `t` (19):

$$\text{Shift} = 5 - 19 = -14 = 12 \pmod{26}$$

The 12th letter is **M**.

1.1.2 Batch 4 Analysis

Letters: `O`, `R`

We have `O` (14) and `R` (17), which are 3 spots apart. Common letters separated by 3 spots are `a` (0) and `d` (3).

$$\text{Shift} = 14 - 0 = 14$$

The 14th letter is **O**.

1.1.3 Batch 5 Analysis

Letters: `P`, `N`

If `N` maps to `a` (0), the shift is 13:

$$\text{P: } 15 - 13 = 2 \text{ (c)}$$
$$\text{N: } 13 - 13 = 0 \text{ (a)}$$

"ca" seems valid. The 13th letter is **N**.

1.1.4 Batch 1 Analysis

Letters: `L`, `V`, `H`

If we try Key **L** (11):

$$\text{L: } 11 - 11 = 0 \text{ (a)}$$
$$\text{V: } 21 - 11 = 10 \text{ (k)}$$
$$\text{H: } 7 - 11 = -4 \equiv 22 \text{ (w)}$$

"akw" seems valid.

1.1.5 Batch 2 Analysis

Letters: `X`, `E`, `R`

If we try Key **E** (4):

$$\text{X: } 23 - 4 = 19 \text{ (t)}$$
$$\text{E: } 4 - 4 = 0 \text{ (a)}$$
$$\text{R: } 17 - 4 = 13 \text{ (n)}$$

"tan" seems valid.

1.2 Recovered Key and Decryption

Recovered Key: `LEMON`

We decrypt using the formula $P = (C - K) \bmod 26$:

Cipher	Key	Calculation	Plaintext
L (11)	L (11)	$11 - 11 = 0$	a
X (23)	E (4)	$23 - 4 = 19$	t
F (5)	M (12)	$5 - 12 = -7 \equiv 19$	t
O (14)	O (14)	$14 - 14 = 0$	a
P (15)	N (13)	$15 - 13 = 2$	c
V (21)	L (11)	$21 - 11 = 10$	k
E (4)	E (4)	$4 - 4 = 0$	a
F (5)	M (12)	$5 - 12 = -7 \equiv 19$	t
R (17)	O (14)	$17 - 14 = 3$	d
N (13)	N (13)	$13 - 13 = 0$	a
H (7)	L (11)	$7 - 11 = -4 \equiv 22$	w
R (17)	E (4)	$17 - 4 = 13$	n

Table 2: Decryption process for each character

Decrypted Message: `attackatdawn`

2 Exercise 2: RSA Encryption and Decryption

2.1 Calculate n, $\varphi(n)$, and the Private Exponent d

Given: $p = 197$, $q = 211$, $e = 24377$

2.1.1 Calculate n

$$n = p \times q$$
$$n = 197 \times 211 = 41567$$

2.1.2 Calculate $\varphi(n)$

$$\varphi(n) = (p-1) \times (q-1)$$
$$\varphi(n) = (197-1) \times (211-1)$$
$$\varphi(n) = 196 \times 210 = 41160$$

2.1.3 Find d

We need to find d such that $e \cdot d \equiv 1 \pmod{\varphi(n)}$, or equivalently:

$$24377 \cdot d \equiv 1 \pmod{41160}$$

Using the Extended Euclidean Algorithm:

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

p, q, e = 197, 211, 24377

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

g, d, _ = egcd(e, phi)
d = d % phi  # Ensure d is positive

print(f"n: {n}")
print(f"phi: {phi}")
print(f"d: {d}")
```

Listing 1: Computing the private exponent

Results:

- $n = 41567$
- $\varphi(n) = 41160$
- $d = 17393$

Verification: $e \times d = 24377 \times 17393 = 423989161 \equiv 1 \pmod{41160}$

2.2 Encrypt and Decrypt

2.2.1 Encryption

Using the formula $C = m^e \bmod n$:

$$C = 1234^{24377} \bmod 41567$$

```python
M = 1234
e = 24377
n = 41567

```

```
C = pow(M, e, n)
print(f"Cipher text: {C}")
```

Listing 2: RSA encryption

Result: $C = 26476$

2.2.2 Decryption

Using the formula $M = C^d \bmod n$:

$$M = 26476^{17393} \bmod 41567$$

```
d = 17393
M_decrypted = pow(C, d, n)
print(f"Plain text: {M_decrypted}")
```

Listing 3: RSA decryption

Result: $M = 1234$ (original plaintext successfully recovered)

2.3 Square-and-Multiply Algorithm

To calculate $1234^{24377} \bmod 41567$, we use the square-and-multiply algorithm.

2.3.1 Binary Representation

$$24377_{10} = 101111100111001_2$$

2.3.2 Algorithm Steps

The algorithm processes each bit from left to right. For each '0' bit, we **square**. For each '1' bit, we **square and multiply** by the base. Each step is performed modulo n.

```
def square_and_multiply(base, exponent, modulus):
    binary_exp = bin(exponent)[2:]

    print(f"Exponent: {exponent}")
    print(f"Binary:   {binary_exp}\n")

    print(f"{'Step':<5} | {'Bit':<3} | {'Operation':<17} | "
          f"{'Calculation':<25} | {'Result':<5}")
    print("-" * 70)

    # First step: Initialization (MSB is always 1)
    result = base
    print(f"{'1':<5} | {'1':<3} | {'Initial':<17} | "
          f"{'-':<25} | {result:<5}")

    # Iterate through the rest of the bits
    step_count = 2
    for bit in binary_exp[1:]:
        prev_result = result

        # Always Square first
        result = (result * result) % modulus
        op_name = "Square"
        calc_desc = f"{prev_result}^2"

        # If bit is 1, also Multiply
        if bit == "1":
            result = (result * base) % modulus
            op_name = "Square & Multiply"
            calc_desc = f"({prev_result}^2) * {base}"
```

```
31
32          print(f"{step_count:<5} | {bit:<3} | {op_name:<17} | "
33                f"{calc_desc + ' mod n':<25} | {result:<5}")
34          step_count += 1
35
36      print("-" * 70)
37      print(f"Final Result: {result}")
38
39  m = 1234
40  e = 24377
41  n = 41567
42
43  square_and_multiply(m, e, n)
```

Listing 4: Square-and-multiply implementation

Step	Bit	Operation	Calculation	Result
1	1	Initial	–	1234
2	0	Square	$1234^2 \bmod n$	26344
3	1	Square & Multiply	$(26344^2) \times 1234 \bmod n$	39933
4	1	Square & Multiply	$(39933^2) \times 1234 \bmod n$	583
5	1	Square & Multiply	$(583^2) \times 1234 \bmod n$	11996
6	1	Square & Multiply	$(11996^2) \times 1234 \bmod n$	6384
7	1	Square & Multiply	$(6384^2) \times 1234 \bmod n$	28435
8	0	Square	$28435^2 \bmod n$	29508
9	0	Square	$29508^2 \bmod n$	18115
10	1	Square & Multiply	$(18115^2) \times 1234 \bmod n$	21154
11	1	Square & Multiply	$(21154^2) \times 1234 \bmod n$	26747
12	1	Square & Multiply	$(26747^2) \times 1234 \bmod n$	22757
13	0	Square	$22757^2 \bmod n$	39363
14	0	Square	$39363^2 \bmod n$	35844
15	1	Square & Multiply	$(35844^2) \times 1234 \bmod n$	26476

Table 3: Square-and-multiply steps for computing $1234^{24377} \bmod 41567$

Final Result: $C = 26476$

3 Exercise 3: Block Cipher Modes and Error Propagation

This exercise analyzes error propagation in different block cipher modes when a single bit error occurs in the first ciphertext block C_1 during transmission. We assume 128-bit blocks.

3.1 CBC Mode (Cipher Block Chaining)

In CBC mode, the decryption process is defined as:

$$P_i = D_k(C_i) \oplus C_{i-1} \tag{1}$$

where C_0 is the initialization vector (IV).

3.1.1 Effect on Decrypted Block P_1

Formula: $P_1' = D_k(C_1) \oplus \text{IV}$

Analysis: If C_1 has 1 wrong bit, then the entire output of $D_k(C_1)$ is corrupted because the block cipher decryption is a deterministic function that is highly sensitive to input changes.

Result: The entire block P_1' is completely corrupted (random garbage).

Bits corrupted in P_1: 128 bits

3.1.2 Effect on Decrypted Block P_2

Formula: $P_2' = D_k(C_2) \oplus C_1$

Analysis: C_2 is received correctly, so the output of $D_k(C_2)$ is correct. However, this result is XORed with C_1 (which contains the 1-bit error) to produce P_2'.

Result: P_2' contains exactly one bit error at the same position as the error in C_1.

Bits corrupted in P_2: 1 bit

3.1.3 Total Corruption in CBC Mode

$$\text{Total bits corrupted} = 128 + 1 = 129 \text{ bits} \tag{2}$$

The first 128 bits are in P_1, and 1 bit is in P_2 at the same position where the error in C_1 occurred.

3.2 CFB Mode (Cipher Feedback)

In CFB mode, the decryption process is:

$$P_i = C_i \oplus E_k(C_{i-1}) \tag{3}$$

where C_0 is the initialization vector (IV).

3.2.1 Effect on Decrypted Block P_1

Formula: $P_1' = C_1 \oplus E_k(\text{IV})$

Analysis: The keystream component $E_k(\text{IV})$ is generated using the IV, which is correct. The corruption is only in C_1, which is directly XORed with the keystream.

Result: P_1 contains exactly one bit error at the specific position where C_1 was corrupted.

Bits corrupted in P_1: 1 bit

3.2.2 Effect on Decrypted Block P_2

Formula: $P_2' = C_2 \oplus E_k(C_1)$

Analysis: To decrypt P_2, the algorithm must encrypt the previous ciphertext block C_1 to create the keystream. Since C_1 has a bit error and is the input to the encryption function E_k, the entire output is corrupted.

Result: The entire block P_2' is completely corrupted (random garbage).

Bits corrupted in P_2: 128 bits

3.2.3 Total Corruption in CFB Mode

$$\text{Total bits corrupted} = 1 + 128 = 129 \text{ bits} \tag{4}$$

One bit is corrupted in P_1 at the same position where the error in C_1 occurred, and the remaining 128 bits are corrupted in P_2.

4 Exercise 4: ElGamal Decryption

Given: $p = 23$, $g = 5$, $C_1 = 20$, $C_2 = 22$, private key $x = 6$

4.1 Calculate $C_1^x \bmod p$

$$\begin{aligned} C_1^x \bmod p &= 20^6 \bmod 23 \\ &= 64000000 \bmod 23 \\ &= 16 \bmod 23 \end{aligned}$$

Result: $C_1^x \equiv 16 \pmod{23}$

4.2 Find the Modular Multiplicative Inverse

We need to find the modular multiplicative inverse of 16 modulo 23, i.e., we are looking for $(16)^{-1} \bmod 23$.

Let k be the inverse. We need to satisfy:

$$16 \cdot k \equiv 1 \pmod{23} \tag{5}$$

4.2.1 Using the Extended Euclidean Algorithm

Forward steps:

$$\begin{aligned} 23 &= 1 \cdot 16 + 7 \\ 16 &= 2 \cdot 7 + 2 \\ 7 &= 3 \cdot 2 + 1 \end{aligned}$$

Backward substitution:

$$\begin{aligned} 1 &= 7 - 3(2) \\ &= 7 - 3(16 - 2(7)) \\ &= 7 - 3(16) + 6(7) \\ &= 7(7) - 3(16) \\ &= 7(23 - 16) - 3(16) \\ &= 7(23) - 7(16) - 3(16) \\ &= 7(23) - 10(16) \end{aligned}$$

Therefore:

$$\begin{aligned} 1 &\equiv -10(16) \pmod{23} \\ (16)^{-1} &\equiv -10 \equiv 13 \pmod{23} \end{aligned}$$

Verification:

$$16 \times 13 = 208 = 9 \times 23 + 1 \equiv 1 \pmod{23}$$

4.3 Decrypt to Find the Plaintext M

The ElGamal decryption formula is:

$$M = C_2 \cdot (C_1^x)^{-1} \bmod p \tag{6}$$

Using the values we calculated:

$$\begin{aligned} M &= 22 \cdot 13 \bmod 23 \\ M &= 286 \bmod 23 \\ M &= 10 \end{aligned}$$

Final Plaintext: $M = 10$