TU856/3
CRYPTOGRAPHY & CYBER SECURITY

ASSIGNMENT 2: FINAL ASSIGNMENT

04/05/2025

PAULINA CZARNOTA C21365726

TASK 1: REFLECTION VIDEO AND PRESENTATION

For Task 1, I prepared a short PowerPoint presentation and recorded a 2-minute video reflection discussing the skills and knowledge I gained throughout the module. The video covers key experiences from the labs and lectures, including my work with classical ciphers, modern encryption like RSA and AES, hands-on practice with Linux tools, and practical tasks like malware analysis and secure communication design. These tasks helped me build confidence for future roles in cybersecurity.

Submitted files:

- PowerPoint: *Crypto_and_Cyber_Security_CA2_Task1_Reflection_Paulina_Czarnota.pptx*
- Video: *Crypto_and_Cyber_Security_CA2_Task1_Reflection_Paulina_Czarnota.mp4*

TASK 2: RSA COMMUNICATION – CONFIDENTIALITY + AUTHENTICATION

1. Objective

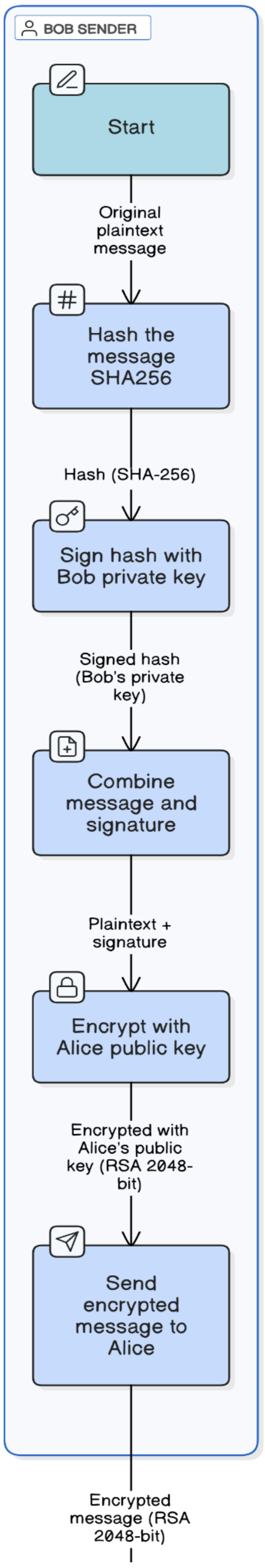
This task demonstrates how to create a secure communication channel using the RSA encryption algorithm, ensuring both authentication and secrecy. The communication occurs between Bob (sender) and Alice (recipient), with Bob encrypting and signing the messages.

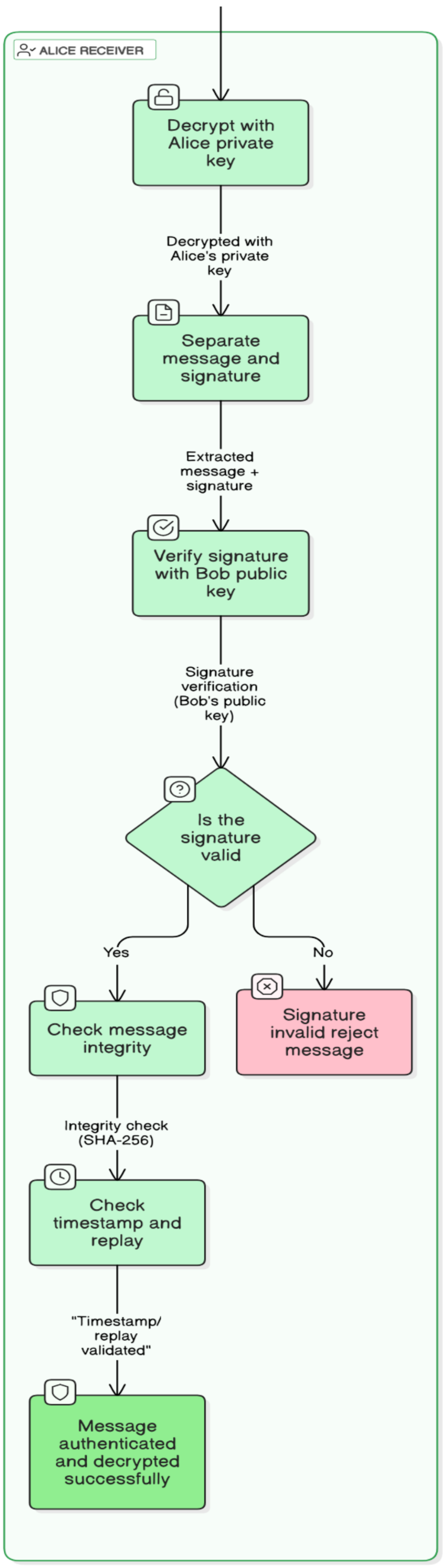
2. RSA Communication Scheme Diagram

The RSA communication process includes:

- Bob creates a message.
- He hashes the message using SHA-256.
- He signs the hash using his private key.
- He combines the original message and signature.
- He encrypts the combined data using Alice's public key (RSA 2048-bit).
- He sends the encrypted message to Alice.
- Alice decrypts it using her private key.
- She separates the message and signature.
- She verifies the signature using Bob's public key.
- She checks message integrity and replay resistance.

RSA Secure Communication: Confidentiality + Authentication


BOB SENDER
Start
Original plaintext message
Hash the message SHA256
Hash (SHA-256)
Sign hash with Bob private key
Signed hash (Bob's private key)
Combine message and signature
Plaintext + signature
Encrypt with Alice public key
Encrypted with Alice's public key (RSA 2048-bit)
Send encrypted message to Alice
Encrypted message (RSA 2048-bit)



ALICE RECEIVER
Decrypt with Alice private key
Decrypted with Alice's private key
Separate message and signature
Extracted message + signature
Verify signature with Bob public key
Signature verification (Bob's public key)
Is the signature valid
Yes
No
Check message integrity
Signature invalid reject message
Integrity check (SHA-256)
Check timestamp and replay
"Timestamp/ replay validated"
Message authenticated and decrypted successfully

Figures 1 & 2: This diagram demonstrates RSA secure communication. Bob signs a SHA-256 hash of the message with his private key, encrypts the signed data using Alice's public key (RSA 2048-bit), and sends it. Alice decrypts using her private key and verifies the signature using Bob's public key. This ensures authentication, confidentiality, and message integrity.

3. RSA Key Generation

Tool Used: Devglan RSA Tool

Alice’s RSA Key Pair

- **Public Key:**

```
-----BEGIN PUBLIC KEY-----

MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAz2oOT1jgigNB6x9z4I8+pPHp6lQmBoW
QAqJ1ZLge15AXpKqRo/cDgg27mcNBOeBHGUJHKNq2bd+i5MErWqb2EceFBKdQzB1TwBNGskW28R
ElrQaYwgwP4K528rqg6p6zZudFN0c8JvWY0yCC/xloWXT7PAYFg26nm1kr5A6HW23/jZ2J9mI7d
D+jGCixsrQ4IhwBa2rWnAV52OGYcjeDDGggtPPGZkQK2C9Npr8fF0Pwxtvv31K4ze6Bu0G3y31U
6UBAlkNpJEBly+ArYJNVmlifoc+XtiBiOWWehOZNraLPPbdcOD2kuiBNuHBcE77RVdlFddRmaL+
QjFo7+ZfPZQIDAQAB

-----END PUBLIC KEY-----
```

- **Private Key (for decryption):**

```
-----BEGIN RSA PRIVATE KEY-----

MIIEvQIBADANBgkqhkiG9w0BAQEFAASCBKcwggSjAgEAAoIBAQDPag5PWOCKA0HrH3Pgjz6k8en
qVCYGhZAConVkuB7XkBekqpGj9wOCDbuZw0E54EcZQkco2rZt36LkwStapvYRx4UEp1DMHVPAE0
ayRbbxESWtBpjCDA/grnbyuqDqnrNm50U3Rzwm9ZjTIIL/GWhZdPs8BgWDbqebWSvkDodbbf+Nn
Yn2Yjt0P6MYKLGytDgiHAFratacBXnY4ZhyN4MMaCC088ZmRArYL02mvx8XQ/DG2+/fUrjN7oG7
QbfLfVTpQECWQ2kkQGXL4Ctgk1WaWJ+hz5e2IGI5ZZ6E5k2tos89t1w4PaS6IE24cFwTvtFV2UV
11GZov5CMWjv5l89lAgMBAAECggEAGIdz2slpgJjjBlHpa9C0+V/MqJQ8DQBlqBIbbQZjS130ld
pAMBJp3UsPeR9Ayv3tmpyeesPm2Dae9ouru2RMbIzae9LnuMPtICWHclTRTlAXUw+ZKDWx+QHY+
lq85hJPqCdIpcFsaTUZVjqdk1qnXxD5Fz+lDkabVsFkceuP8W1pmtzpaTbPxZf0OeNQH8p5BJn/
6lDNyJOpqRzO9RUIuqeOQR7dzTn6BtxSp02Q4tCPR9/a2TeOJ2xbORFcxy0BzyYtcY3RxXi5s0P
ik4WJHKKgLujeIE1Ygq8gkkmDDs6MtmD7udRbcpRwtA3f9SaGM/aJ8bVvzH0S5Wi7pqqScQKBgQ
D7TLSECx2J++31DLa5qKliQce9n0dW5vIhgK0A4NFm+2Q99Rxi8UT6GWVzREdEIkt69S35VsJLT
ATJZ8wfrauQElpwZEUUFsxW8vRSveM5f14pzhVJnopiPgthzz0KWy8Wd4ssOB1fjb8UB0PK05DD
VLFD8KUx0eAfHE9/ImgpXQKBgQDTSzcQc5OJX+LFSWAZZW74/B+ZBXBOoOVb/DvsJS0KMOE9ejF
/TGBp0P44zA7pj2eafBPWSy5MrR20YaSZ5hrMi/xeqcCFqc0zXMqOnjmkkE9exVdpYsvlD7HKpJ
T0XpqD2H9z91ZZD0A8RMRCArYgjSV/q69YdmiOE0oBP9F1qQKBgGgRR+A2yjDOR6jnbvRLyHmMT
C9WurBeS39cTkjC0XUYPlb6HptK1wnLYpvtdqXcne15g2RybVvJS2IsvfeQcC/lqdZJb+H13wFS
86MgR+0TtecSFPa71QsCLqx+qgST5lgjgVr6kg90H+EncQeZvaVmeVJtYbvT8kBv9OvtT71xAoG
AQjmX56sh+fO4wvctjKgzMQzs5mxC+BIL8VnMekZoqIILhtZOv0R0D5Z2p9yla+ULpuru2MK1n0
i/P7M/ARGTIrtZfV4dPoerDkcuIadMD90/U0ftDtXtbD0QD6CoIzOnoF1yDSnfwYg7LOiszrfE1
80FjM+F5Ocv/B/lmSCkiQkCgYEA970L0FI/3ly3y0Bw5DfXH53elJlJQX0qgiXTZ3Qy+Y+C3K+F
G2F1jePYmw9qor+zu0aY4Aq3yv/DmcQkIyTMMkqZ1+PzwykagH9oGiAkoZM096LhVMit6Img5OF
TN17tAaVTK9kABvKbTawausZDUO/tzW63vA+ciZLgifReYR8=

-----END RSA PRIVATE KEY-----
```

4. Encrypting Messages with Alice’s Public Key

Bob encrypts the messages below using Alice’s public key on the Devglan tool.

- **Message 1**
  - **Plaintext:** Hi, how are you today?
  - **Ciphertext:**

```
mJP+tobrHeCNWkDhLFuXdtmwXwsv/6uokKsShtvh5ruTMABKoVDFNp5dNTWNuGQQ6cP6l
MtEJg1Jwb+mX67Xg6cmAZbr1gY61pdTv7KI/ahr0TW5ULqjEV41WSrgBruLkPlQ1aSiNc
YT8xkTYhPr3P66qyJGBXxuYkLak+zLiTfmyD61nTahd33XLAA7j+lXFCd9Jvfu858Oaqe
MqwVysKORQbXaTU37/ykp+b11Wq5D1cHgPLm2OsJM04Xmmex2QHQhyBpq+ncmfuZpdrTI
Ipe68++wdGawxhOubz03YZX8BxQd4Z2E4lB9SfIJJdiAbZGN2KvAvRSMnJCUKj/bMw==
```

- **Message 2**
  - **Plaintext:** Spring vibes are fabulous
  - **Ciphertext:**

```
WfK9G4efEo0AHPx3QxpnTtOx1hTQMTB7WDT8pBlQJ2VtxUhCiOMt7Lc5g77PM3Kts6hec
/X/gvjMaoclAaVQb1oAP4qNEGTM7s/PniZZ8zmVhnOEblo38Ba5y6ly5Kpdhn1EzcFc1O
Og2lWCb95RKVPmsaFm8qQD6+f0Xs+T6SX8zGPFg+FLZ0Q74NaiXk5vlksKlvpSRaPIS9Q
mwyodO3H/d2ES0MXBUZ054WYqei9bF6iaEI+0dOutoKyQQeZdjTpQMgV84dEpGDW/qJPS
AGCiw53SX0pH3bkTb4H0v6RRcUf8YsMcjNk+fOpvtfXAUrjEgSuTqHX818488EaAwA==
```

- **Message 3**
  - **Plaintext:** See you then.
  - **Ciphertext:**

```
I8PoyzkTUrmSbmHPkHxCHwRMvULBxT+M4Vsxi66D/6NALaXAqxWzFcP3oVOpdebVs/APn
Yqm644o3pnXa2JknnNKcOG52S7AE7M7ITsPK5bsEM3NRVHcUGoH9+sceVVZAyBsqTBQko
wBLnHWhEIdsAmySuSV22kzFZefQxd66TpwpQQS23Z5dZUGsWR++shqeEfUTohXLInRpsB
Mv217vo6FDgWV0xmAn3s3P+i2f615JarA9IjJC4LbB99ZfnoUj0mlM+9eaubFYHNm0mhQ
Jlrc9edhIdvy65I/Uqnp8RxHR7+eSjZOXAkmO9Ce9wJ7TNwaYkieyER4nrPOjUM94A==
```

5. Explanation – Why Use RSA?

RSA is a public-key encryption algorithm that uses:

- A public key for encryption
- A private key for decryption

Bob signs the hash of the message with his private key (authenticity), then encrypts the message and signature using Alice’s public key (secrecy).

Only Alice can decrypt the message, and only Bob could have signed it.

Key Technical Points:

- **Encryption algorithm**: RSA 2048-bit
- **Hash function**: SHA-256
- **Signature verification**: Bob’s public key

- **Data confidentiality**: ensured by public key encryption
- **Authentication**: ensured by digital signature

This method defends against:

- Man-in-the-middle (MITM) attacks
- Tampering
- Replay attacks (if timestamping included)

6. Reflection – What I Learned

This task gave me valuable practical experience in applying public key cryptography to achieve both confidentiality and authenticity in digital communication. By generating RSA key pairs, encrypting plaintext messages using a recipient's public key, and simulating digital signatures through the sender's private key, I reinforced the core principles of asymmetric encryption in a hands-on context.

The process directly reflected concepts introduced in Lecture 9: Public Key Cryptography and RSA, where I learned the foundational structure and functions of RSA, including encryption/decryption using key pairs and the role of trapdoor one-way functions. I also applied insights from Lecture 10: Cryptographic Hash Functions, which explained the use of SHA-256 for creating message digests that are integral to digital signatures. Furthermore, the task aligned closely with Lecture 11 Part 1: Digital Signatures, where I studied how digital signatures provide message integrity, origin authentication, and non-repudiation, and how these features are implemented using RSA in real systems.

Additionally, my understanding of the threats to signature systems, such as existential forgeries and replay attacks, was supported by content from Lecture 11 Part 2: User Authentication. This emphasized the role of timestamps and sequence numbers in preventing attacks and ensuring freshness in communication. The real-world lab component, particularly Lab 8: GnuPG, allowed me to mirror these cryptographic operations using command-line tools, enhancing both my theoretical knowledge and technical proficiency.

By completing this task, I now feel confident explaining and applying RSA in real-world use cases such as secure email, digital identity validation, and trusted message exchange. The task showed me how public key cryptography can form the backbone of digital trust in secure communications.

TASK 3: FREQUENCY ANALYSIS – DECRYPTING CIPHERTEXT

1. Objective

This task required decrypting a monoalphabetic substitution cipher using frequency analysis. The method is a classical cryptanalysis technique which identifies patterns and statistical frequencies of letters in English text to break ciphers without knowing the key.

2. Ciphertext Provided

```
CI FUANGEVUZNOA, QUHXWHIFA ZIZRALCL CL GOH LGWMA EQ GOH QUHXWHIFA EQ
RHGGHUL EU VUEWNL EQ RHGGHUL CI Z FCNOHUGHTG. GOH BHGOEM CL WLHM ZL ZI ZCM
GE SUHZKCIV LWSLGCGWGCEI FCNOHUL (H.V. BEIE-ZRNOZSHGCF LWSLGCGWGCEI FCNOHU,
FZHLZU LOCQG FCNOHU, PZGLAZAZIZ FCNOHU). QUHXWHIFA ZIZRALCL FEILCLGL EQ
FEWIGCIV GOH EFFWUUHIFH EQ HZFO RHGGHU CI Z GHTG. QUHXWHIFA ZIZRALCL CL
SZLHM EI GOH QZFG GOZG, CI ZIA VCPHI NCHFH EQ GHTG, FHUGZCI RHGGHUL ZIM
FEBSCIZGCEIL EQ RHGGHUL EFFWU JCGO PZUACIV QUHXWHIFCHL. QEU CILGZIFH, VCPHI
Z LHFGCEI EQ HIVRCLO RZIVWZVH, RHGGHUL H, G, Z ZIM E ZUH GOH BELG FEBBEI,
JOCRH RHGGHUL Y, X ZIM T ZUH IEG ZL QUHXWHIGRA WLHM.
```

3. Tools and Method Used

- **Quipqiup.com:** to decrypt monoalphabetic substitution ciphers
- **dCode Frequency Analyzer:** to analyse character frequency
- **Lecture 3: Classical Encryption Techniques** – provided foundational knowledge on frequency-based attacks against substitution ciphers

4. Step-by-Step Decryption Process

1. Frequency Analysis (dCode Output)

- Common letters found in the ciphertext: H, G, I, Z, L, C, U, E, F
- Mapped against common English letters: E, T, A, O, I, N, S, R, H

2. Pattern Recognition

- “QUHXWHIFA” appears multiple times → likely “FREQUENCY”
- “RHGGHUL” appears repeatedly → likely “MESSAGE”
- “CI” is likely “IN”
- “GOH” behaves like “THE”

3. Decryption (using Quipqiup)

- Quipqiup successfully decrypted the ciphertext using these frequency clues and pattern analysis.

4. Intermediate Decryption Attempt

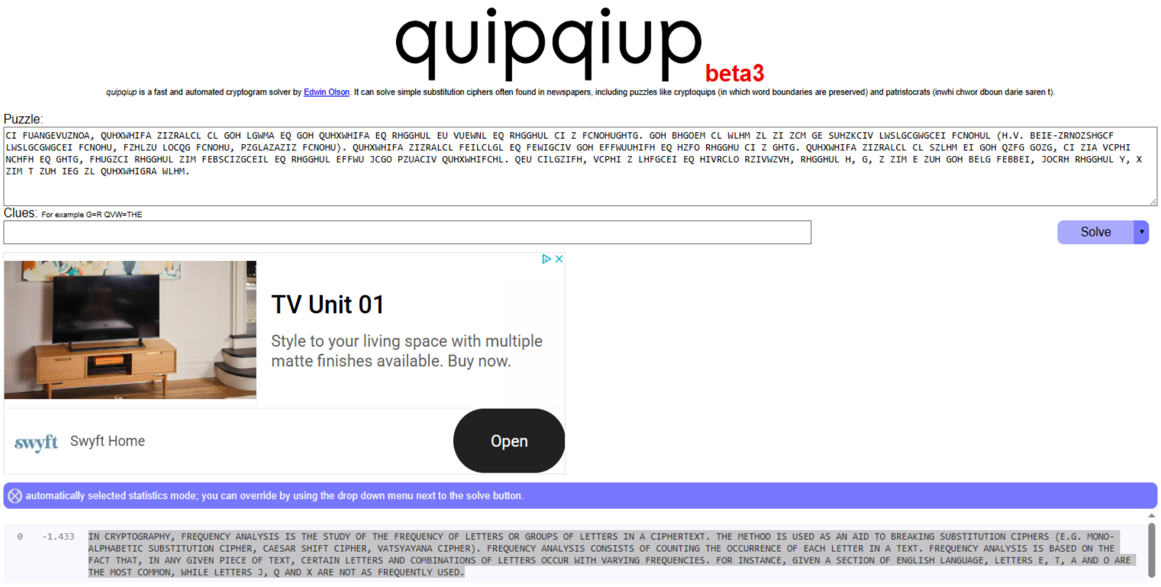
- Using the frequency and pattern clues, I mapped partial substitutions to test segments of the ciphertext.
- Example:
  - **Original segment:** CI FUANGEVUZNOA
  - **Substitutions applied:** C → I, I → N, F → C, U → R, A → Y, N → P, G → R, E → D, V → H, Z → E, O → L
  - **Partially decrypted:** IN CRYPTOGRAPHYEL
- This verified the correctness of my mappings as the word “CRYPTOGRAPHY” became visible early on.

5. Final Decrypted Message

```
IN CRYPTOGRAPHY, FREQUENCY ANALYSIS IS THE STUDY OF THE FREQUENCY OF
LETTERS OR GROUPS OF LETTERS IN A CIPHERTEXT. THE METHOD IS USED AS AN AID
TO BREAKING SUBSTITUTION CIPHERS (E.G. MONO-ALPHABETIC SUBSTITUTION CIPHER,
CAESAR SHIFT CIPHER, VATSYAYANA CIPHER). FREQUENCY ANALYSIS CONSISTS OF
COUNTING THE OCCURRENCE OF EACH LETTER IN A TEXT. FREQUENCY ANALYSIS IS
BASED ON THE FACT THAT, IN ANY GIVEN PIECE OF TEXT, CERTAIN LETTERS AND
COMBINATIONS OF LETTERS OCCUR WITH VARYING FREQUENCIES. FOR INSTANCE, GIVEN
A SECTION OF ENGLISH LANGUAGE, LETTERS E, T, A AND O ARE THE MOST COMMON,
WHILE LETTERS J, Q AND X ARE NOT AS FREQUENTLY USED.
```

6. Substitution Table (Partial)

Cipher	Plain
Q	F
U	R
H	E
X	Q
W	U
I	N
F	C
A	Y
G	T
Z	A
C	I
L	S
M	B
O	G
E	D
V	H
N	P



quipqiup is a fast and automated cryptogram solver by Edwin Olson. It can solve simple substitution ciphers often found in newspapers, including puzzles like cryptoquips (in which word boundaries are preserved) and patristocrats (inwhi chwor dboun darie saren t).

Puzzle:

```
CI FUANGEVUZNOA, QUHXWHIFA ZIZRALCL CL GOH LGWMA EQ GOH QUHXWHIFA EQ RHGGHUL EU VUEWNL EQ RHGGHUL CI Z FCNOHUGHTG. GOH BHGOEM CL WLHM ZL ZI ZCM GE SUHZKCIV LWSLGCGWGCEI FCNOHUL (H.V. BEIE-ZRNOZSHGCF
LWSLGCGWGCEI FCNOHU, FZHLZU LOCQG FCNOHU, PZGLAZAZIZ FCNOHU). QUHXWHIFA ZIZRALCL FEILCLGL EQ FEWIGCIV GOH EFFWUUHIFH EQ HZFO RHGGHU CI Z GHTG. QUHXWHIFA ZIZRALCL CL SZLHM EI GOH QZFG GOZG, CI ZIA VCPHI
NCHFH EQ GHTG, FHUGZCI RHGGHUL ZIM FEBSCIZGCEIL EQ RHGGHUL EFFWU JCGO PZUACIV QUHXWHIFCHL. QEU CILGZIFH, VCPHI Z LHFGCEI EQ HIVRCLO RZIVWZVH, RHGGHUL H, G, Z ZIM E ZUH GOH BELG FEBBEI, JOCRH RHGGHUL Y, X
ZIM T ZUH IEG ZL QUHXWHIGRA WLHM.
```

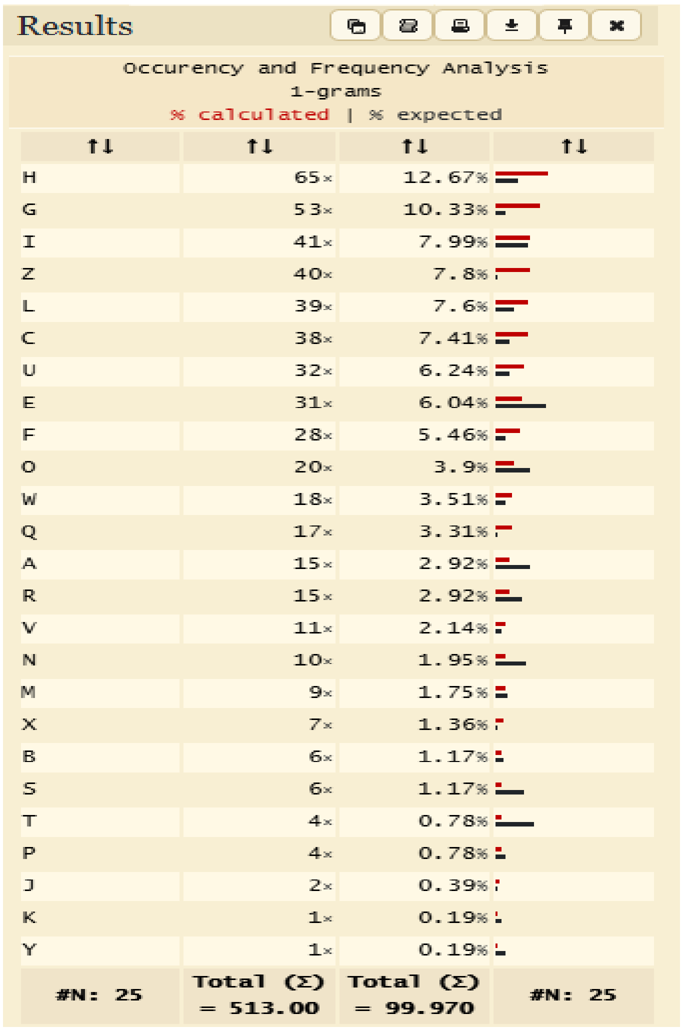
Clues: For example G=R QVW=THE

Solve
TV Unit 01
Style to your living space with multiple matte finishes available. Buy now.
swyft Swyft Home
Open

automatically selected statistics mode; you can override by using the drop down menu next to the solve button.

```
0   -1.433   IN CRYPTOGRAPHY, FREQUENCY ANALYSIS IS THE STUDY OF THE FREQUENCY OF LETTERS OR GROUPS OF LETTERS IN A CIPHERTEXT. THE METHOD IS USED AS AN AID TO BREAKING SUBSTITUTION CIPHERS (E.G. MONO-
             ALPHABETIC SUBSTITUTION CIPHER, CAESAR SHIFT CIPHER, VATSYAYANA CIPHER). FREQUENCY ANALYSIS CONSISTS OF COUNTING THE OCCURRENCE OF EACH LETTER IN A TEXT. FREQUENCY ANALYSIS IS BASED ON THE
             FACT THAT, IN ANY GIVEN PIECE OF TEXT, CERTAIN LETTERS AND COMBINATIONS OF LETTERS OCCUR WITH VARYING FREQUENCIES. FOR INSTANCE, GIVEN A SECTION OF ENGLISH LANGUAGE, LETTERS E, T, A AND O ARE
             THE MOST COMMON, WHILE LETTERS J, Q AND X ARE NOT AS FREQUENTLY USED.
```

Figure: Quipqiup tool output showing successful decryption using monoalphabetic substitution and frequency analysis.



Occurency and Frequency Analysis
1-grams
% calculated | % expected

↑↓	↑↓	↑↓	↑↓
H	65×	12.67%	
G	53×	10.33%	
I	41×	7.99%	
Z	40×	7.8%	
L	39×	7.6%	
C	38×	7.41%	
U	32×	6.24%	
E	31×	6.04%	
F	28×	5.46%	
O	20×	3.9%	
W	18×	3.51%	
Q	17×	3.31%	
A	15×	2.92%	
R	15×	2.92%	
V	11×	2.14%	
N	10×	1.95%	
M	9×	1.75%	
X	7×	1.36%	
B	6×	1.17%	
S	6×	1.17%	
T	4×	0.78%	
P	4×	0.78%	
J	2×	0.39%	
K	1×	0.19%	
Y	1×	0.19%	
#N: 25	Total (Σ) = 513.00	Total (Σ) = 99.970	#N: 25

Figure: Frequency analysis of the ciphertext showing the calculated vs. expected English letter frequencies (generated via dCode).

5. Explanation

Frequency analysis leverages the statistical appearance of letters in a language to guess substitutions.

For example:

- "E" is the most common letter in English → so "H" in the cipher might be "E"
- Common words like "THE" or "IN" help guess multi-letter patterns

By matching high-frequency letters and recurring word patterns, the plaintext was revealed.

This mirrors the decryption approach described in Lecture 3, and also ties to Lab 3 which introduced frequency tools like dCode.

6. Reflection – What I Learned

This task helped me understand the vulnerability of classical ciphers. By using frequency analysis and substitution logic, I could decrypt the message without knowing any key.

This shows why modern cryptographic systems like AES or RSA are needed — because they resist such basic attacks. It also showed the power of tool-assisted decryption and the importance of language patterns in classical cryptanalysis.

TASK 4: KALI QUESTIONS

Question 1: What is Kali? How is it different from other Linux distributions? How does it support cyber security?

Kali Linux is a specialized Debian-based Linux distribution developed by Offensive Security. It is designed specifically for penetration testing, digital forensics, and ethical hacking.

Key differences from other Linux distributions:

- It comes pre-installed with over 600 security tools, including Nmap, Wireshark, Metasploit, John the Ripper, Burp Suite, and Aircrack-ng.
- It is customized for security tasks — unlike Ubuntu or Fedora, which are general-purpose.
- Kali has features like custom kernels, ARM support, and persistence modes for live USB attacks and testing environments.

How it supports cyber security:

- Enables vulnerability assessments, password cracking, wireless attacks, and social engineering simulations.
- It is widely used in professional cyber security fields for red teaming, penetration testing, and ethical hacking training.
- Supports full encryption, secure boot, and live forensics tools for investigations.

Kali is the ethical hacker's primary toolset in practical security testing.

Question 2: What command line is used for sharing your public key in a file?

To export and share your GPG public key in Kali Linux (or any Linux terminal):

```
gpg --export -a "Your Name" > publickey.asc
```

- `--export`: exports the key
- `-a`: ASCII-armors it into readable format
- `"Your Name"`: replace with the name or email tied to your GPG key
- `> publickey.asc`: saves the output to a file

This creates a text file (`publickey.asc`) you can safely send to others so they can encrypt messages for you or verify your signatures.

Question 3: How can you run a brute force attack?

A brute force attack attempts every possible password or key until it finds the correct one. In Kali, tools like Hydra, John the Ripper, and Medusa are commonly used.

Example using Hydra:

```
hydra -l admin -P /usr/share/wordlists/rockyou.txt 192.168.0.101 ssh
```

Explanation:

- `-l admin`: login/username
- `-P`: path to the password list (like `rockyou.txt`)
- `192.168.0.101`: target IP
- `ssh`: the protocol to attack

You can also use John the Ripper on hashed password files:

```
john --wordlist=/usr/share/wordlists/rockyou.txt hashed_passwords.txt
```

Brute force attacks are only ethical when used in authorized penetration testing environments.

Question 4: How can you get your IP computer address?

To get your computer's IP address in Kali or any Linux terminal:

```
ip a
```

Or more specific:

```
ip addr show
```

It will display all network interfaces. Look under the one labelled `eth0`, `wlan0`, or `enp0s3` (depending on the system). Your IP is listed after `inet`, e.g.:

```
inet 192.168.0.101/24
```

Alternative commands:

- `hostname -I` → returns IP only
- `ifconfig` → legacy command, may need to install `net-tools`

Question 5: How can you run Python code?

There are two main ways to run Python code in Kali Linux:

Option 1: Using the Python interactive shell

- Open terminal and type:

  ```
  python3
  ```

- You can now enter Python commands line-by-line.

Example:

```
print("Hello, Kali!")
```

- To exit the shell, use:

  ```
  exit()
  ```

  or press `Ctrl+D`.

Option 2: Running a .py file from the terminal

1. Create a file:

   ```
   nano myscript.py
   ```

2. Write your Python code inside:

   ```
   print("This is a Python script.")
   ```

3. Save and exit:

   Press `Ctrl+X`, then `Y`, then `Enter`.

4. Run:

   ```
   python3 myscript.py
   ```

Note: Kali Linux comes with Python 3 pre-installed. Always use python3 to ensure you're using the correct version.

TASK 5: IMAGE ENCRYPTION USING AES IN ECB/CBC MODE

1. Objective

This task involved exploring how the AES block cipher encrypts an image file in two different modes: ECB and CBC. I was given a Python script using the PyCryptodome library and OpenCV to:

- Encrypt an image (*secret.jpg*) with AES.
- Compare the visual results of ECB vs CBC modes.
- Understand how encryption affects visual patterns in image data.

2. What Technique Does the Code Implement?

The script implements AES (Advanced Encryption Standard), a symmetric block cipher where the same key is used for encryption and decryption. Two AES modes are supported:

- **ECB (Electronic Codebook)** — the simplest mode where each block is encrypted independently. This mode reveals image patterns.
- **CBC (Cipher Block Chaining)** — where each block is XORed with the previous ciphertext block before encryption, hiding image patterns more securely.

The Python code supports both modes and allows switching via:

```
mode = AES.MODE_CBC  # or AES.MODE_ECB
```

3. Encryption Section of the Code

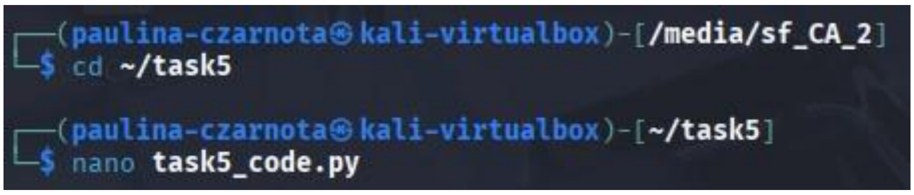
Encryption starts after reading the image and converting it to bytes:

```
imageOrigBytes = imageOrig.tobytes()

key = get_random_bytes(keySize)

iv = get_random_bytes(ivSize)

cipher = AES.new(key, mode, iv) if mode == AES.MODE_CBC else AES.new(key,
mode)

imageOrigBytesPadded = pad(imageOrigBytes, AES.block_size)

ciphertext = cipher.encrypt(imageOrigBytesPadded)
```

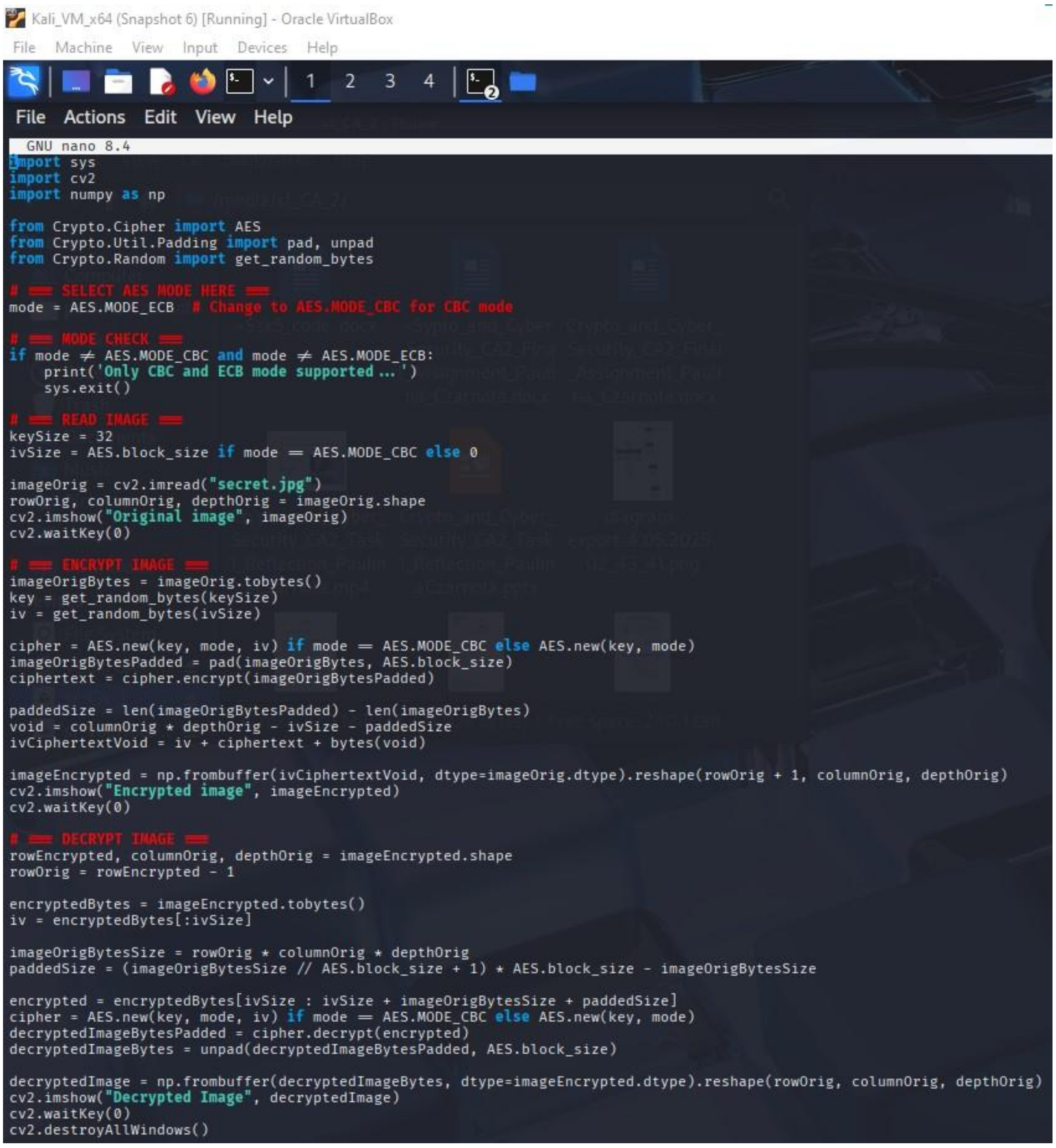
Key steps:

- The image is flattened into raw byte data.
- AES key and IV are generated.

- The AES cipher object is created.
- The image is padded and encrypted.



```
┌──(paulina-czarnota㉿kali-virtualbox)-[/media/sf_CA_2]
└─$ cd ~/task5

┌──(paulina-czarnota㉿kali-virtualbox)-[~/task5]
└─$ nano task5_code.py
```


Kali_VM_x64 (Snapshot 6) [Running] - Oracle VirtualBox
File Machine View Input Devices Help
File Actions Edit View Help
GNU nano 8.4

```python
import sys
import cv2
import numpy as np

from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad
from Crypto.Random import get_random_bytes

# == SELECT AES MODE HERE ==
mode = AES.MODE_ECB  # Change to AES.MODE_CBC for CBC mode

# == MODE CHECK ==
if mode != AES.MODE_CBC and mode != AES.MODE_ECB:
    print('Only CBC and ECB mode supported...')
    sys.exit()

# == READ IMAGE ==
keySize = 32
ivSize = AES.block_size if mode == AES.MODE_CBC else 0

imageOrig = cv2.imread("secret.jpg")
rowOrig, columnOrig, depthOrig = imageOrig.shape
cv2.imshow("Original image", imageOrig)
cv2.waitKey(0)

# == ENCRYPT IMAGE ==
imageOrigBytes = imageOrig.tobytes()
key = get_random_bytes(keySize)
iv = get_random_bytes(ivSize)

cipher = AES.new(key, mode, iv) if mode == AES.MODE_CBC else AES.new(key, mode)
imageOrigBytesPadded = pad(imageOrigBytes, AES.block_size)
ciphertext = cipher.encrypt(imageOrigBytesPadded)

paddedSize = len(imageOrigBytesPadded) - len(imageOrigBytes)
void = columnOrig * depthOrig - ivSize - paddedSize
ivCiphertextVoid = iv + ciphertext + bytes(void)

imageEncrypted = np.frombuffer(ivCiphertextVoid, dtype=imageOrig.dtype).reshape(rowOrig + 1, columnOrig, depthOrig)
cv2.imshow("Encrypted image", imageEncrypted)
cv2.waitKey(0)

# == DECRYPT IMAGE ==
rowEncrypted, columnOrig, depthOrig = imageEncrypted.shape
rowOrig = rowEncrypted - 1

encryptedBytes = imageEncrypted.tobytes()
iv = encryptedBytes[:ivSize]

imageOrigBytesSize = rowOrig * columnOrig * depthOrig
paddedSize = (imageOrigBytesSize // AES.block_size + 1) * AES.block_size - imageOrigBytesSize

encrypted = encryptedBytes[ivSize : ivSize + imageOrigBytesSize + paddedSize]
cipher = AES.new(key, mode, iv) if mode == AES.MODE_CBC else AES.new(key, mode)
decryptedImageBytesPadded = cipher.decrypt(encrypted)
decryptedImageBytes = unpad(decryptedImageBytesPadded, AES.block_size)

decryptedImage = np.frombuffer(decryptedImageBytes, dtype=imageEncrypted.dtype).reshape(rowOrig, columnOrig, depthOrig)
cv2.imshow("Decrypted Image", decryptedImage)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

- **Purpose:** Show the ECB encryption code (before switching to CBC).

4. Decryption Section of the Code

The decryption block mirrors encryption and reverses the operations:

```
cipher = AES.new(key, mode, iv) if mode == AES.MODE_CBC else AES.new(key,
mode)

decryptedImageBytesPadded = cipher.decrypt(encrypted)

decryptedImageBytes = unpad(decryptedImageBytesPadded, AES.block_size)

decryptedImage = np.frombuffer(decryptedImageBytes,
dtype=imageEncrypted.dtype).reshape(rowOrig, columnOrig, depthOrig)
```

Key steps:

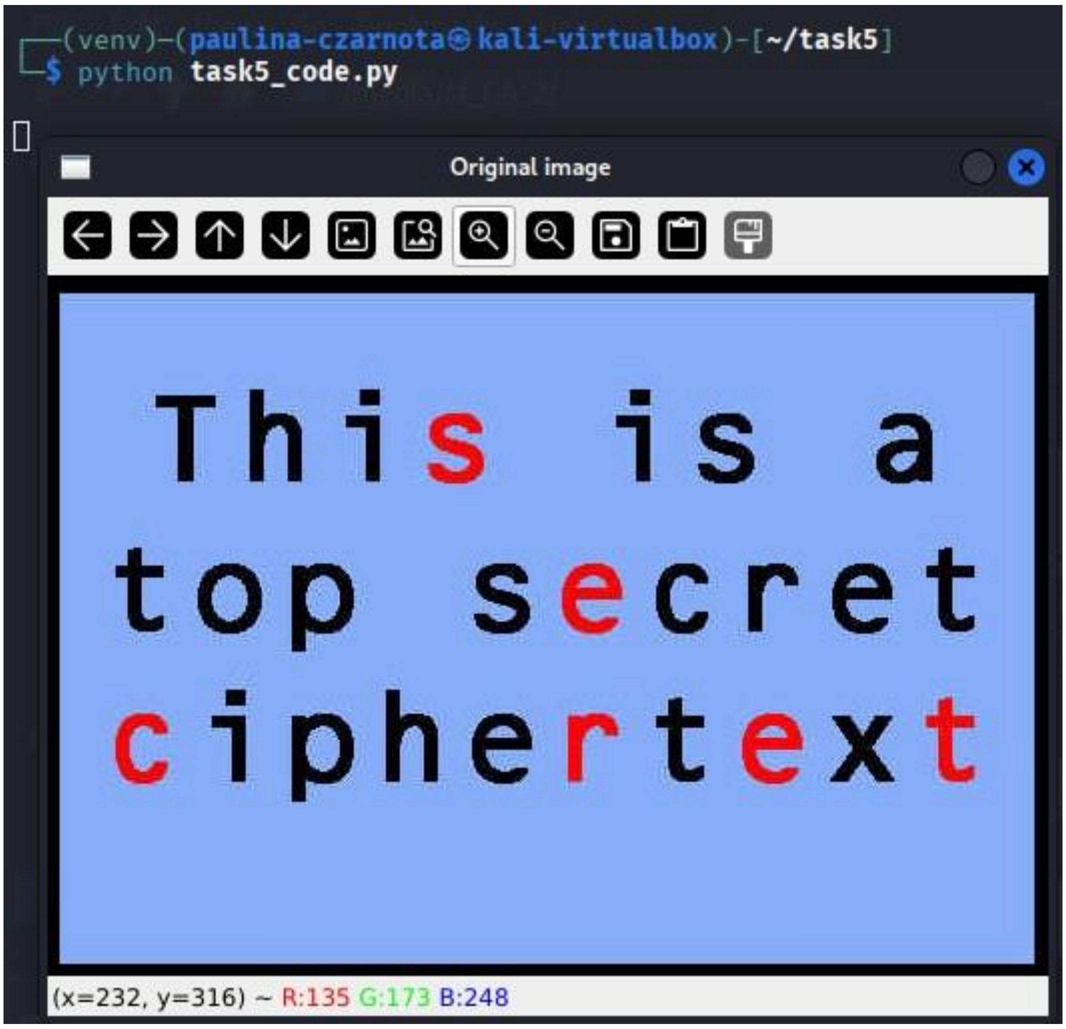
- AES cipher is recreated using the same key and IV.
- Ciphertext is decrypted.
- Padding is removed.
- Decrypted bytes are reshaped into the original image format.

5. The Role of the Image

The image (*secret.jpg*) is used to demonstrate how AES transforms visual data:

- It's loaded via OpenCV.
- Encrypted in memory (not written to disk).
- The encrypted result is reshaped and displayed, showing visual distortion.
- Decryption is then used to recover the original image.

This demonstrates how encryption affects image data and how some modes (like ECB) can reveal patterns.



```
┌──(venv)─(paulina-czarnota㉿kali-virtualbox)-[~/task5]
└─$ python task5_code.py
```

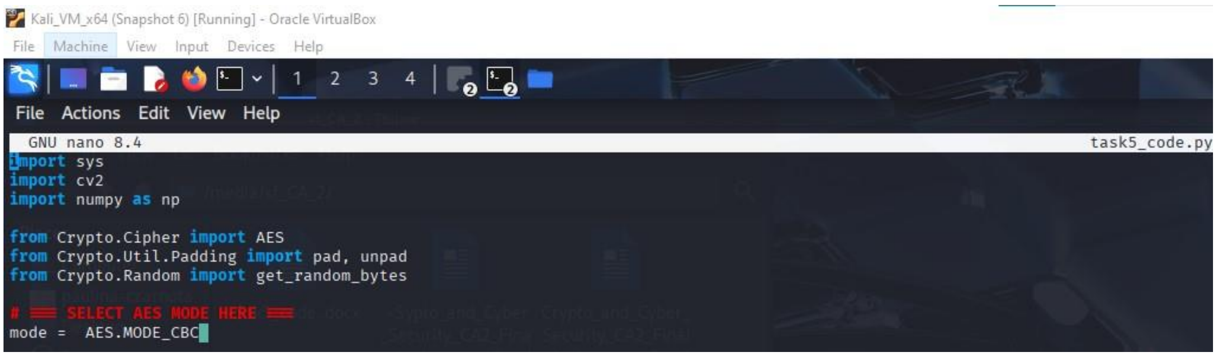
Original image
This is a top secret ciphertext
(x=232, y=316) ~ R:135 G:173 B:248

- **Purpose:** Display the original unencrypted image (ECB Mode)

6. Switching to CBC Mode

To improve security and eliminate visible patterns, I changed the mode to CBC:

```
mode = AES.MODE_CBC
```



Kali_VM_x64 (Snapshot 6) [Running] - Oracle VirtualBox
File Machine View Input Devices Help
File Actions Edit View Help

```
  GNU nano 8.4                                                              task5_code.py
import sys
import cv2
import numpy as np

from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad
from Crypto.Random import get_random_bytes

# ══ SELECT AES MODE HERE ══
mode =  AES.MODE_CBC
```

CBC introduces block chaining using XOR and an IV, making the ciphertext visually randomized and more secure.

Before change:

```
mode = AES.MODE_ECB
```

After change:

```
mode = AES.MODE_CBC
```

This is the only change needed since the code already handles IV generation.

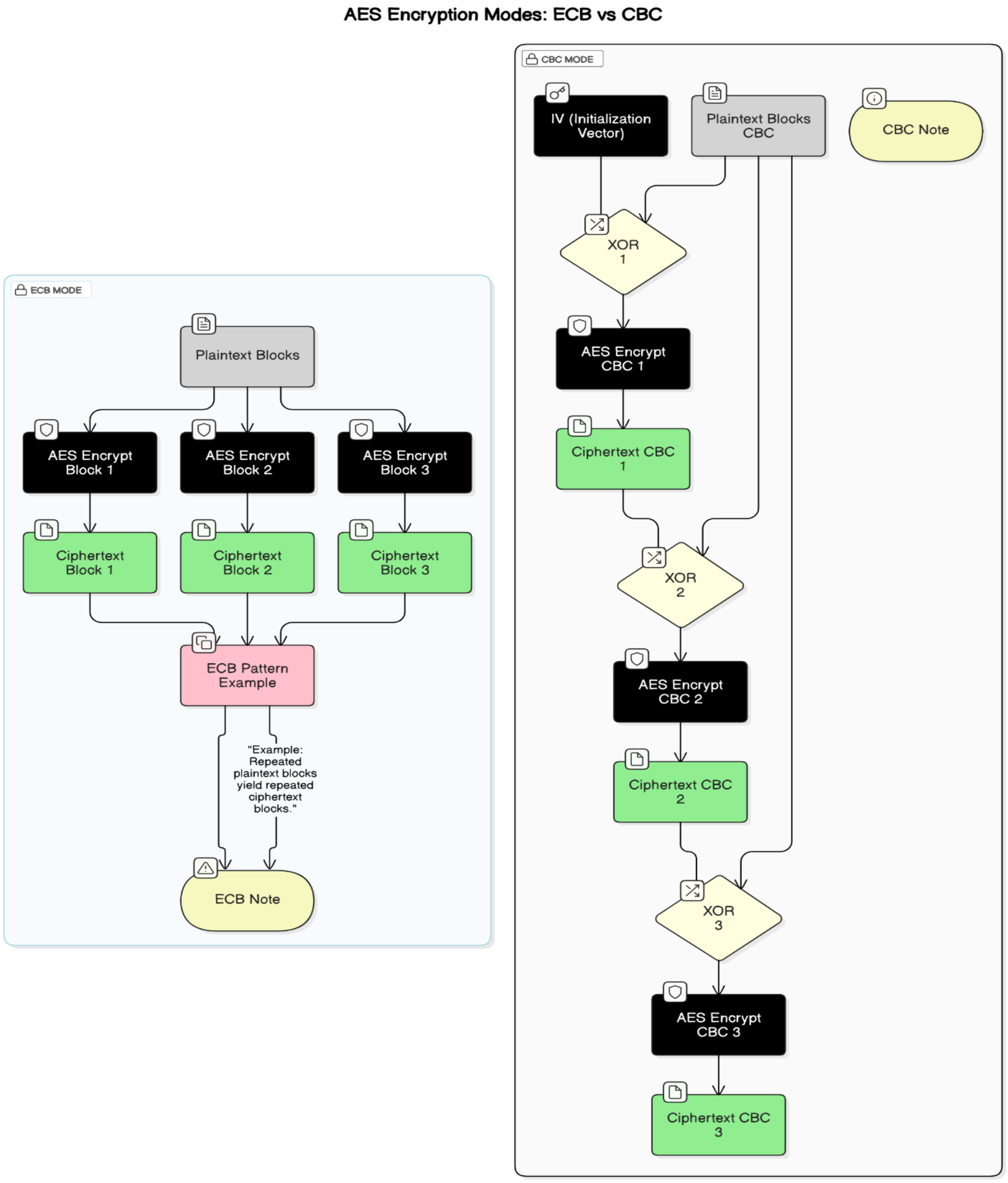
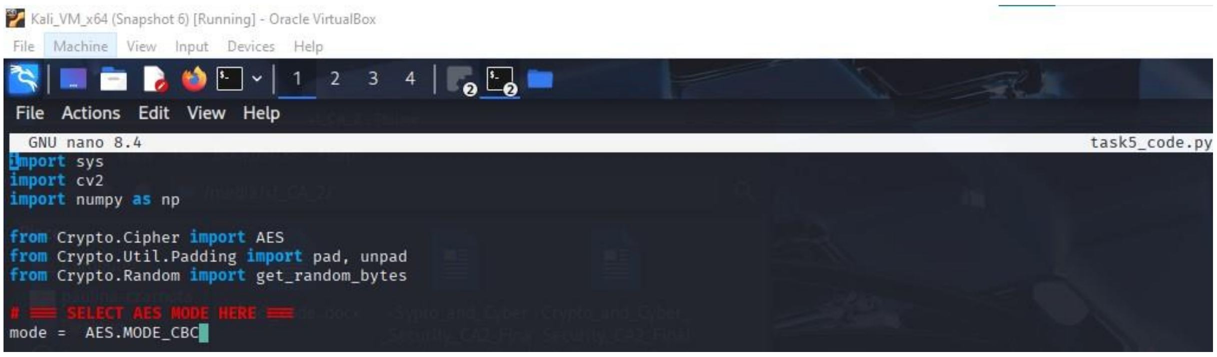
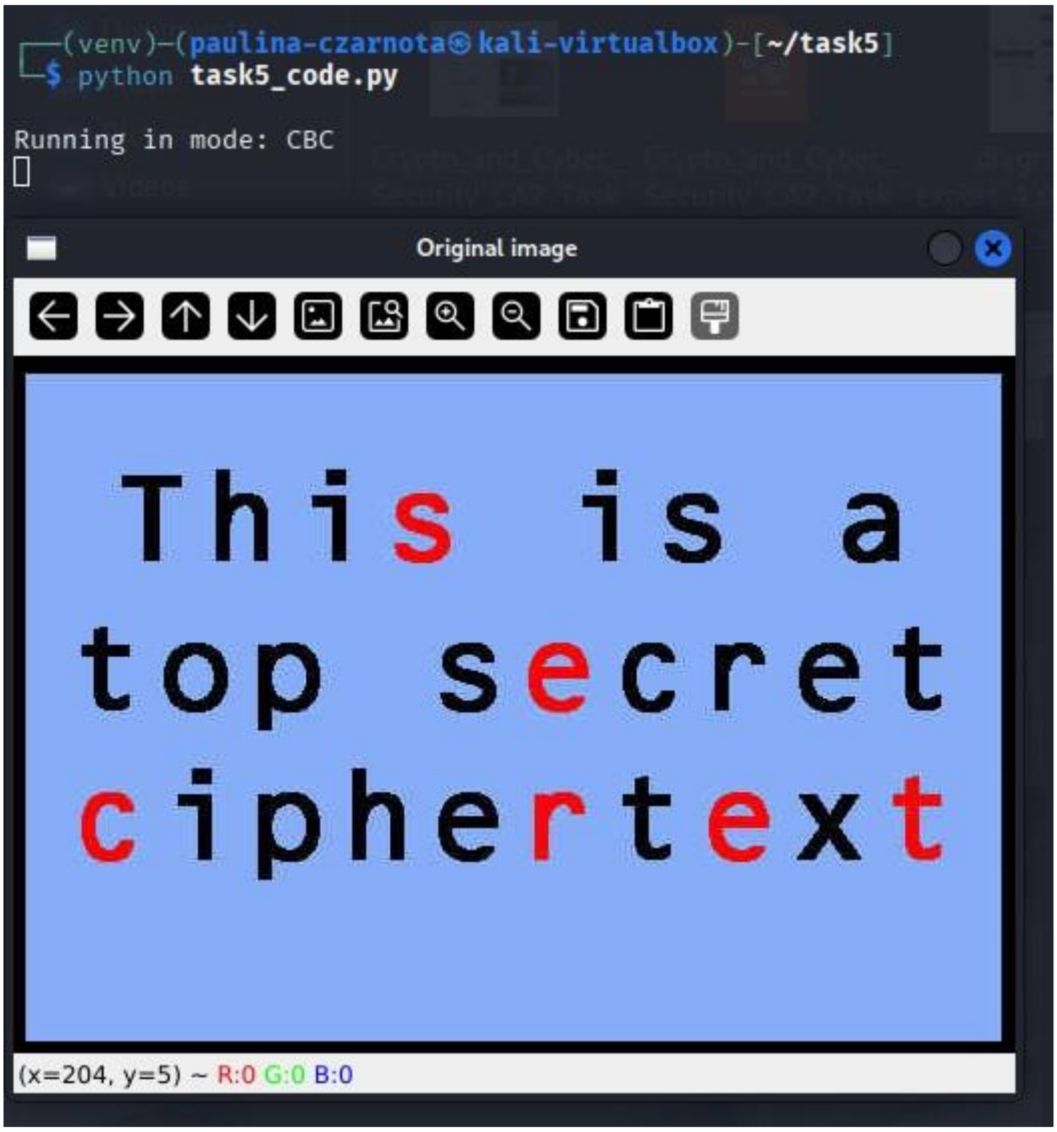


Figure: Visual comparison between AES ECB and CBC encryption modes. ECB mode shows repeated patterns in ciphertext, while CBC mode prevents this by chaining blocks through XOR operations.



Kali_VM_x64 (Snapshot 6) [Running] - Oracle VirtualBox
File Machine View Input Devices Help
File Actions Edit View Help

```
GNU nano 8.4                                                          task5_code.py
import sys
import cv2
import numpy as np

from Crypto.Cipher import AES
from Crypto.Util.Padding import pad, unpad
from Crypto.Random import get_random_bytes

# === SELECT AES MODE HERE ===
mode =  AES.MODE_CBC
```



```
┌──(venv)─(paulina-czarnota㉿kali-virtualbox)-[~/task5]
└─$ python task5_code.py

Running in mode: CBC
```

Purpose: Show that AES.MODE_CBC was selected in code.

- Make sure the IV (initialization vector) is handled correctly — this is already managed in the code with:

```
iv = get_random_bytes(ivSize)
```

7. Reflection – What I Learned

This task helped me understand the practical implications of using different AES encryption modes, particularly the weaknesses of ECB when applied to image data. By working with the Python code, I gained hands-on experience with core cryptographic operations such as byte-level conversion, padding, and reshaping encrypted data into visual formats. The clear visual difference between ECB and CBC modes made it easy to see how patterns in ciphertext can leak information if the wrong mode is chosen.

Switching the encryption mode to CBC deepened my appreciation for how block chaining improves security by eliminating repetitive patterns. This task also strengthened my ability to analyse and modify real encryption scripts, which is an important skill for any cybersecurity role. Overall, it gave me confidence in understanding how AES operates under different conditions and reinforced why the choice of encryption mode matters in real-world applications like secure image storage, file encryption, and network communications.