The University of Queensland

School of Electrical Engineering and Computer Science (EECS)

Semester 2, 2024

CYBR3000

Information Security

Assignment 2

Due: 3:00 pm, 4 November 2024

Marks: 100
Weight: 25% of your final grade.
Version: 1.9

Revision history:

- Version 1.0: 1 Oct 2024 release
- Version 1.1: 2 Oct 2024
  - Q1: Remove the requirement of default-backend due to deprecation.
  - Q3: Add the specification of using ord() and chr() to perform RSA encryption and decryption.
- Version 1.2: 7 Oct 2024
  - Q3: Replace the example image.
- Version 1.3: 9 Oct 2024
  - Q1: Add more examples.
  - Q1: Clarify that the padding scheme should be done manually instead of using a library.
  - Q2: Add the valid range for private keys.
  - Q3: Update the example.
  - Q3: Add error handling for both p and q should be greater than 10.
- Version 1.4: 14 Oct 2024
  - Q1: Fix the examples with incorrect lengths of ciphertexts for ECB and CBC modes.
  - Q1: Add more examples with salt and IV.
  - Q3: Fix the example.
- Version 1.5: 15 Oct 2024
  - Add requirements: do not change the name and purpose of the functions provided in the templates.
  - Q5: Clarify the requirements according to the lecture material.
  - Q5: Fix and add new examples.
- Version 1.6: 21 Oct 2024
  - General Instructions: Clarify error handlings.
  - Q1: Adjust the order of error handlings based on the workflow in the template.
  - Q4: An example private_key.pem is provided on the BlackBoard. You can use this private key to test if your certificate matches the certificate provided in the Q4 example. The certificate example in Q4 is updated based on this given private key.
  - Q5: Clarify the structure of ESP packet.
  - Q5: Provide hints for packet constructions in tunnel mode and transport mode.
- Version 1.7: 21 Oct 2024
  - Q4: Clarify the usage of the given private key example
  - Q5: Provide clear structures for ESP packet.
  - Q5: Specify use of byte as the unit for all the components in ESP packet.
- Version 1.8: 22 Oct 2024
  - Q5: Fix the examples given for transport mode.
- Version 1.9: 24 Oct 2024
  - Q4: Remove error handling requirements for empty p, q and messages due to inconsistency behaviours across different operating systems.

Introduction

This assignment challenges you to implement cryptographic algorithms and secure communication protocols. You are permitted to use specific libraries for some tasks, and others require manual implementation. The objective is to deepen your understanding of cryptographic principles and secure key exchanges.

General Instructions:

- You must use Python 3.9 for all tasks.
- You are restricted to using only the following libraries: cryptography, scapy, hashlib, and pycryptodome. You need to ask if you can use any additional libraries if you are not sure about it. Each question has allowed libraries.
- For certain questions, you are required to implement functions manually without using these libraries. Read the task requirements carefully.
- Template for each question is provided on course BlackBoard. Please read this spec sheet with the templates given to you for better understanding.
- Comment your code to explain each step.
- Submit your code files separately for each question.
- Each question can be deducted up to 5 marks due to wrong error handling. Your mark will be deducted by 1 point for each error-handling issue, up to a maximum of 5 points for each question.
- Follow the provided templates with appropriate comments.
- The templates are modifiable. You can add functions as you like. Do not modify the name or purpose of the given functions in the templates.
- Replace all the pass sections with your own code in the templates.
- Please implement error handling according to the order given in the spec.
- The templates already contain the code in each question to catch the ValueErrors raised in the script. When a ValueError is raised, the format of output is: Error: <Error message from raised ValueError>.

Q1: Advanced AES Encryption with Different Modes of Operation (20 marks)

Task: Implement AES encryption and decryption using various modes: ECB, CBC, CFB, OFB, CTR, and AES-GCM.

Requirements:

- Use cryptography library for AES implementation.
- Key Derivation: Implement PBKDF2 for key derivation from a user-provided password and salt.
    - Algorithm: SHA-256
    - Length of the key: 32 bytes
    - Iterations: 100000
- Design a manual padding scheme for the modes that require padding.
    - Padding should follow PKCS#7 padding scheme. This scheme adds padding bytes where each byte's value is equal to the number of padding bytes added. For example, if 3 bytes of padding are needed, the padding would be 03 03 03.
- Encrypt and decrypt plaintext using a certain mode.
- **Execution:** Run the script as $ python3 Q1.py.

- The encrypted output should be consistent if the inputs remain the same.
- Include error handling for invalid inputs.
    - If the password is empty, raise a ValueError that says: “Password cannot be empty.”.
    - If the provided salt or IV is not a hex string, raise a ValueError that says: “Invalid hex string.”.
    - If the plaintext is empty, raise a ValueError that says: “Plaintext cannot be empty.”.
    - If the mode is not correct or empty, raise a ValueError that says: “Unsupported mode.”. The mode is case-sensitive.
- **Examples**

```
Enter password: 123123
Enter salt (leave blank for default):
Enter IV (leave blank for default):
Enter plaintext: abcd
Enter AES mode (ECB, CBC, CFB, OFB, CTR, GCM): ECB
Encrypted: USqTGNR5Z1q4fkkl+sEkgw==
Decrypted: abcd
```

```
Enter password: 123123
Enter salt (leave blank for default):
Enter IV (leave blank for default):
Enter plaintext: abcd
Enter AES mode (ECB, CBC, CFB, OFB, CTR, GCM): CBC
Encrypted: jYFNS4vgTFe3z0LUPyF2wg==
Decrypted: abcd
```

```
Enter password: 123123
Enter salt (leave blank for default):
Enter IV (leave blank for default):
Enter plaintext: Hello World Hello World
Enter AES mode (ECB, CBC, CFB, OFB, CTR, GCM): CFB
Encrypted: IMFPDgoKhFPtZQ+RExmNcKxELd3SKow=
Decrypted: Hello World Hello World
```

```
Enter password: 123123
Enter salt (leave blank for default):
Enter IV (leave blank for default):
Enter plaintext: Hello World Hello World
Enter AES mode (ECB, CBC, CFB, OFB, CTR, GCM): OFB
Encrypted: IMFPDgoKhFPtZQ+RExmNcJZIOcfXCm4=
Decrypted: Hello World Hello World
```

```
Enter password: 123123
Enter salt (leave blank for default):
Enter IV (leave blank for default):
Enter plaintext: Hello World Hello World
Enter AES mode (ECB, CBC, CFB, OFB, CTR, GCM): CTR
Encrypted: IMFPDgoKhFPtZQ+RExmNcAdo0NBzqLg=
Decrypted: Hello World Hello World
```

```
Enter password: 123123
Enter salt (leave blank for default):
Enter IV (leave blank for default):
Enter plaintext: Hello World Hello World
Enter AES mode (ECB, CBC, CFB, OFB, CTR, GCM): GCM
Encrypted: iHFUihXVc8xxp/lj+efIkamYZl+46WU=
Decrypted: Hello World Hello World
```

```
Enter password: 123123
Enter salt (leave blank for default): 019323b20d95fc9f1d43e301c366fc4c
Enter IV (leave blank for default): 48a372e1c0cef5832a1f74bc54c7bb68
Enter plaintext: abcd
Enter AES mode (ECB, CBC, CFB, OFB, CTR, GCM): ECB
Encrypted: mdGLbH+NQfoIB1PVYCoULQ==
Decrypted: abcd
```

```
Enter password: 123123
Enter salt (leave blank for default): 019323b20d95fc9f1d43e301c366fc4c
Enter IV (leave blank for default): 48a372e1c0cef5832a1f74bc54c7bb68
Enter plaintext: abcd
Enter AES mode (ECB, CBC, CFB, OFB, CTR, GCM): CBC
Encrypted: SuBmFsrhf55wviz4pVSxcw==
Decrypted: abcd
```

```
Enter password: 123123
Enter salt (leave blank for default): 019323b20d95fc9f1d43e301c366fc4c
Enter IV (leave blank for default): 48a372e1c0cef5832a1f74bc54c7bb68
Enter plaintext: Hello Bob, this is Alice.
Enter AES mode (ECB, CBC, CFB, OFB, CTR, GCM): CFB
Encrypted: KN/Mlfm/bWB7ohG2eIqkRpcoQ4sbAesqlA==
Decrypted: Hello Bob, this is Alice.
```

```
Enter password: 123123
Enter salt (leave blank for default): 019323b20d95fc9f1d43e301c366fc4c
Enter IV (leave blank for default): 48a372e1c0cef5832a1f74bc54c7bb68
Enter plaintext: Hello Bob, this is Alice.
Enter AES mode (ECB, CBC, CFB, OFB, CTR, GCM): OFB
Encrypted: KN/Mlfm/bWB7ohG2eIqkRs47XnI7AeC6pA==
Decrypted: Hello Bob, this is Alice.
```

```
Enter password: 123123
Enter salt (leave blank for default): 019323b20d95fc9f1d43e301c366fc4c
Enter IV (leave blank for default): 48a372e1c0cef5832a1f74bc54c7bb68
Enter plaintext: Hello Bob, this is Alice.
Enter AES mode (ECB, CBC, CFB, OFB, CTR, GCM): CTR
Encrypted: KN/Mlfm/bWB7ohG2eIqkRhDoNbDVLRMliQ==
Decrypted: Hello Bob, this is Alice.
Enter password: 123123
Enter salt (leave blank for default): 019323b20d95fc9f1d43e301c366fc4c
Enter IV (leave blank for default): 48a372e1c0cef5832a1f74bc54c7bb68
Enter plaintext: Hello Bob, this is Alice.
Enter AES mode (ECB, CBC, CFB, OFB, CTR, GCM): GCM
Encrypted: 4jbzMbLaPuKxRdk3gm5WMwJT0ZlZIEp1Ag==
Decrypted: Hello Bob, this is Alice.
```

Q2: Implement Diffie-Hellman Key Exchange Manually (10 marks)

Task: Manually simulate the Diffie-Hellman key exchange process.

Requirements:

- Simulate two parties, Alice and Bob, each with their own private key and public key.
- Use built-in `pow` function for modular exponentiation.
- Implement the `check_prime()` function to check if a number is prime. If it is not a prime number, raise a ValueError that says: "Not a prime number." Only apply this check function to numbers that should be prime.
- The shared secret should be computed without relying on any external libraries.
- **Execution:** Run the script as `$ python3 Q2.py`.
- Ensure robust input validation for prime numbers and handle edge cases appropriately.
    - If a user input is empty, raise a ValueError that says: "Empty value is not allowed."
    - If a use input is not an integer, raise a ValueError that says: "Only integer values are allowed.".
    - If a private key is invalid, raise a ValueError that says: "Invalid private key.". The range for the private keys is $1 \leq$ private key $< p$.
- **Examples**

```
Enter prime number p: 23
Enter generator g: 5
Enter Alice's private key: 6
Enter Bob's private key: 15
Alice's private key: 6
Bob's private key: 15
Alice's public key: 8
Bob's public key: 19
Shared secret: 2 (Alice) | 2 (Bob)
```

Q3: Implement RSA Encryption with Manual Key Generation (20 marks)

Task: Write a Python program for RSA encryption and decryption including key generation.
Requirements:

- Include two prime numbers p and q as command-line arguments.
    - Both p and q should be greater than 10.
- Implement modular arithmetic functions manually.
- Compute the public and private keys using these primes.
- Select the smallest possible e that is coprime with ϕ.
- Encrypt and decrypt the message using ord() and chr().
- **Execution:** Run the script as $ python3 Q3.py [prime_number_p] [prime_numer_q] [message].
- Ensure input validation for prime numbers and manage edge cases.
    - If the command-line arguments are invalid, raise a ValueError that says: "Usage: python Q3.py <prime_p> <prime_q> <message>".
    - ~~If p or q is empty, raise a ValueError that says: "Empty value is not allowed.".~~
    - ~~If the message is empty, raise a ValueError that says: "Empty message is not allowed.".~~
    - If p or q is not an integer, raise a ValueError that says: "Only integer values are allowed.".
    - If p or q is smaller or equal to 10, raise a ValueError that says: "Both p and q need to be greater than 10.".
    - If p or q is not a prime number, raise a ValueError that says: "Both p and q need to be prime numbers.".
    - If p and q are the same number, raise a ValueError that says: "p and q cannot be equal.".

- **Example**

Q4: Simulate PKI and Digital Certificate Creation (10 marks)

Task: Develop a program to create a digital certificate using RSA.

Requirements:

- Use cryptography library for RSA key generation and certificate signing.
- For the first time running Q4.py, the script should generate an RSA key and save it to the same folder as private_key.pem and use this key to generate the digital certificate. For any later run of Q4.py, the script should read this RSA key and use it as the key to generate your digital certificate. Include the generated private_key.pem in your submission.
- An example private_key.pem is provided on the BlackBoard to test your code and verify if your generated certificate matches with the screenshot. However, before the submission, please remove the given private_key.pem and rerun your code to generate your own private_key.pem, so that you can include your own private key in your submission.
- Certificate should include issuer, public key, serial number, and validity period. For the issuer:
    - COUNTRY_NAME: “AU”
    - STATE_OR_PROVINCE_NAME: "Queensland"
    - LOCALITY_NAME: "Brisbane”
    - ORGANIZATION_NAME: "UQ"
    - COMMON_NAME: "uq.com"
- Serialize keys and certificate in PEM format, hint: serialization.Encoding.PEM
- Serial number, not valid before and not valid after are provided in the template.
- **Execution:** Run the script as $ python3 Q4.py.
- Overwrite certificate.pem whenever you run your script.
- The private_key.pem you generated will be different from the given private key example.
- The certificate you generate should be the same whenever you run this script.
- **Example**
    - Your generated private_key.pem should look like this:

```
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAsdgrAN4dv4/NMBJ668A8CU5mZeTAncvw363y292QNSY4+eb+
REAArkWAJOKtvzJbCnS9kDGv5ZRwS1HUO34RRwwetNP+V4SJqGsBE8Dg/WwpBKFu
/lhJ15PITcLUkMWe03XaU8iC/Sz+H1Q9sgGfDfvj8IQ8XSbZWOnwQi8S67ZKk9wa
kcw+UxE5y5Kx7NPJFjNGyFeXzDJRb9qZygTB6ky/8AMakATNJRmo07SzAlHQEdaf
byoH1a+hpGvcn8GmKZ6c3YXt+7lR7wAKSX8bZLeZ4gKQKc3/Pupjd+eeLUUtHPxx
FK2q39CBLKKUBQls5PQQ0aOi17h0A2y+XFNd9wIDAQABAoIBAAG+I86VAE+FZ9sg
ALVp3Lo4g2dQNc5cyO9fxihP7lnwPnls3ENRRGZV1h8anwYWnnIwu0zrKqzvrXEv
geIQahaB7vMY/3QbCzmobpDm4gNt8RAdmy2Sq31PUK99VgFjEb0oGzrcNv4SqMwa
o1TDv4Nd/Q9oTtmmM7QYgbfLDNML8sTKxT1Sjp+GF5dDf7KFEW0JGsUYCVV2fWA/
xHawi8BzOaj0AyObqNXxAribXdGZ5Zhpa1XpRL3IGkfH+6oUzjgGlq1V0mTNgjjc
DPSp8i/rrZXrfyJNsYiarskRt4CE4Ad+csENWYZD8MIkEj64guEO4gldXFk6tcFq
jbThOvECgYEA9dG+1Bgb/79VKTYRkWZo1yftwvp0qbv2qcLH68G8sVeYV23cJHP0
ygwyPtZr5oUI4kOAMM3HEeivxh0+Ubr0tJVimPK8AuCSvygNbFk41Lz1SaTs5fAl
ceOYtBzcWZXifsrk8o8KUVFrSb5K70DQnNwAy4cZeJSNj65Qa9nq1XUCgYEAuTW7
HhKMDemtkFD36Uw/1NpWtlRMtZMX/XEzi1vXeiA5oWChcuIXOGDNIm6AGcWyJmnQ
2OOcZV+XpTGZhIg3cftjApAU13itXGrnZzsxH9pquq80dpwaad2Pxx7BsY1f3F8z
7aoPa2q7ilqq/aiP9LZMoSCm2rjdKpuQkQEh/DsCgYEA6wY9oEToyC1ju8IxK8Yx
KLzJSyjwz7Ohkw9i25pg0WHBip3D46pl+aONipyxjzXUl5gYARFNamcBX7Zn9fZ8
rgdBvKAfd9TFGTSvGM95UZTUJDQmN3XSJ0CGqHYLiy+N3NIZ8XsnTzY8EMZ6mKNs
kPybWfxyMuI9Molj5Kn2J10CgYBx0Y5bjcRaLHgIT2n7CmvT0BYn2KiHkYzVqMKn
atqB59nizbKKGl1wvTRza3r326F+r9qz3zTWq6dS0qdoZhnEx1tyUL8uJ7BKMjxh
GWQ4Eg8zCMbx+T7WPl3EXadlCN+XWz5bWP4+4gm8nbmlRviox3tSLfG8So1ISO/P
3OVYzwKBgEezEAEApKsjoXWMmlGWoT1s/v+LkoRPgVUig1FWCQGwXtd8AF3lKwjP
tTRpn+cgsd2AiemCPBdznYQq4pkNeFPLRZdDQKg7v6NHYOJGUaE4yF9qVNgQ2G3Q
FDngwbuwj+Z7FlQuFmv0N3WeQ9hL0bDOxb9r95SSeJ35zgKMoCPT
-----END RSA PRIVATE KEY-----
```

- Your generated certificate.pem should look like this:

```
-----BEGIN CERTIFICATE-----
MIIDIDCCAgigAwIBAgICA+gwDQYJKoZIhvcNAQELBQAwUzELMAkGA1UEBhMCQVUx
EzARBgNVBAgMClF1ZWVuc2xhbmQxETAPBgNVBAcMCEJyaXNiYW5lMQswCQYDVQQK
DAJVUTEPMA0GA1UEAwwGdXEuY29tMB4XDTI0MTAwMTAwMDAwMFoXDTI0MTAzMTAw
MDAwMFowUzELMAkGA1UEBhMCQVUxEzARBgNVBAgMClF1ZWVuc2xhbmQxETAPBgNV
BAcMCEJyaXNiYW5lMQswCQYDVQQKDAJVUTEPMA0GA1UEAwwGdXEuY29tMIIBIjAN
BgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAsdgrAN4dv4/NMBJ668A8CU5mZeTA
ncvw363y292QNSY4+eb+REAArkWAJOKtvzJbCnS9kDGv5ZRwS1HUO34RRwwetNP+
V4SJqGsBE8Dg/WwpBKFu/lhJ15PITcLUkMWe03XaU8iC/Sz+H1Q9sgGfDfvj8IQ8
XSbZWOnwQi8S67ZKk9wakcw+UxE5y5Kx7NPJFjNGyFeXzDJRb9qZygTB6ky/8AMa
kATNJRmo07SzAlHQEdafbyoH1a+hpGvcn8GmKZ6c3YXt+7lR7wAKSX8bZLeZ4gKQ
Kc3/Pupjd+eeLUUtHPxxFK2q39CBLKKUBQls5PQQ0aOi17h0A2y+XFNd9wIDAQAB
MA0GCSqGSIb3DQEBCwUAA4IBAQBcfnJx3WeVHxTuc2prP4V2BvOTlj5jj3FYoRdh
w8Rcneyv/TWgvTTDZYHT3LlUQ9mDvifNAGZZOmpxibF6lUPLv96TnQdC0cPExeeC
GdGZ0lfUuLc8YYjCu7PSppfKvxfSVEI85iOZXf6ITPHQ8Rsoqq+L+yQspvwNjxuR
nraDha2zbuhNKFtF0XHk5FkwJ0ZsgQRiOkHPcu7cNDyiVPXHNynp61s9iXhO8xjA
ch7odr1VVSGco62iVzdGImzkbWONnAF89wjxHnqkYt2pvHO6AlPcvFUEAV2iiwbx
ZOTqW8nERAqXANnvPjIX/iKBKXUxwasW9X8S7CnKR4YSaJAp
-----END CERTIFICATE-----
```

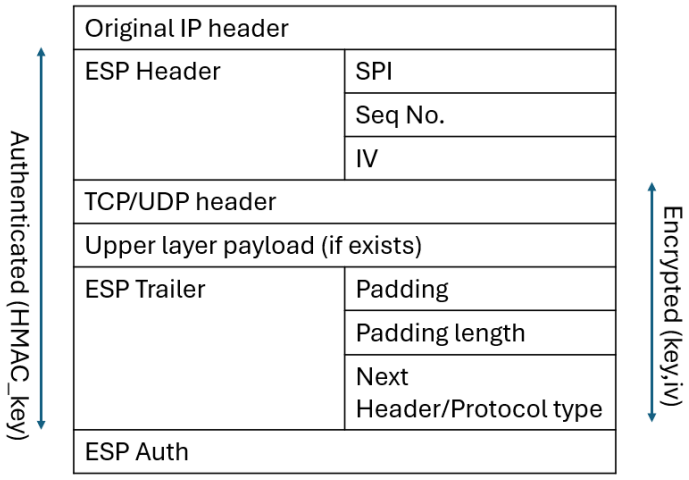
Q5: Implement IPsec ESP Protocol Using Open Libraries (40 marks)

Task: Simulate the IPsec ESP protocol focusing on AES encryption and packet construction for both tunnel and transport modes.

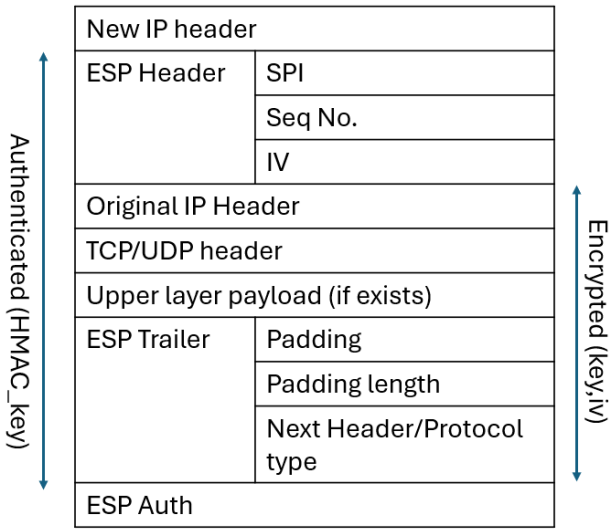
Requirements:

- Use the pycryptodome library for encryption.
- Read the packet from the given .pcap file. Your script should be able to parse the absolute path to read the .pcap file. The .pcap file will always contain one single packet.
- For every component in your ESP packet, use byte as the unit. To convert things into bytes format, use bytes().
- PKCS#7 padding scheme is used for the ESP trailer.
    - The next header/protocol type should be the same protocol as the original packet.
    - Add padding to the part that needs to be encrypted. For the simplicity of this question, add padding the same way you did in Q1 and then add the extra bytes of padding length and the next header. It is fine that the payload after padding is not a multiple of AES block size since it will be handled inside the encryption function provided in the template.
- Use AES-CBC to get the encrypted part of the ESP packet.
    - The encryption function, key and IV are provided in the template.
- Generate the ESP packet header with the following parameters:
    - Security Parameter Index (SPI): bytes([1]) * 4
    - Sequence number: b'\x00\x00\x00\x01'
    - For the sake of simplicity, no need to include Reserved field in the ESP packet.
- Generate the ESP authentication.
    - Use HMAC for ESP Authentication. (Function provided in the template)
- Construct the ESP packet based on “tunnel” or “transport” mode. Use 192.168.99.99 as the new source IP if necessary. The structure of the ESP packet follows the lecture content. IV should be included since we are using CBC to perform the encryption.
- You can use show() to check if your ESP packet is constructed under the correct mode.
- For the simplicity, the ESP packet you generated will be slightly different from real cases. You will generate headers only using the fields mentioned in the diagram below. For example, for the new IP header in the “tunnel mode”, you only need to specify the source IP address and destination IP address.

Transport Mode



Tunnel Mode



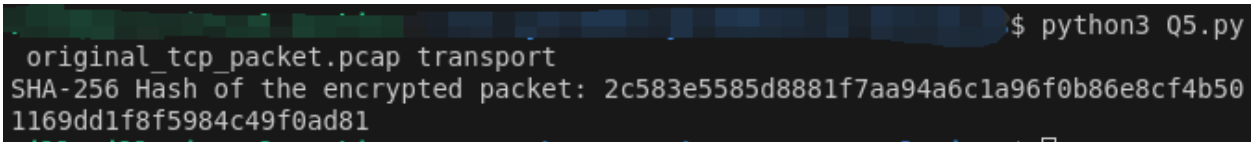
- For “transport” mode, the ESP packet should be constructed by keeping the original IP header and replacing the payload in the original packet with your ESP components. Hint: remove_payload(), add_payload().

- For “tunnel” mode, the ESP packet should be constructed by adding an IP layer to your ESP components. No need to add the link layer. Please use scapy IP to generate your new IP layer. You only need to specify the source IP address and destination IP address and simply ignore all the other fields.

- Calculate and print out the hash value of the whole ESP packet.
- **Execution:** Run the script as $ python3 Q5.py [path_to_pacp_file] [mode].
- Please follow the order of command line arguments as above while running the script.
- Please ignore this kind of warning:

```
WARNING: No IPv4 address found on anpi2 !
WARNING: No IPv4 address found on anpi1 !
WARNING: more No IPv4 address found on anpi0 !
```

- Error Handling:
    - If a wrong format of command-line arguments is provided, raise a ValueError that says: “Usage: python3 Q5.py [path_to_pcap_file] [mode]”.
    - If the path to the .pcap file is wrong or the file is invalid, raise a ValueError that says: “Failed to read the pcap file.”.
    - If an unexpected mode is provided, raise a ValueError that says: “Invalid mode. Choose 'tunnel' or 'transport'.”.
- **Example**



Submission

You should submit your assignment using the submission link in the Assignment 2 folder in the course blackboard (under Assessment). Compress your code for each question into one zipped (compressed) file with YourStudentID_A2.zip (e.g., 12345678_A2.zip):

e.g.
12345678_A2.zip
- private_key.pem
- Q1.py
- Q2.py
- Q3.py
- Q4.py
- Q5.py

Please make sure that the code you submit is executable. Also please make sure to complete your submission by clicking the submission button. Note that it is not allowed to manually change any part of your submitted Python code and text files, including modifying lines, whether they are commented out or added for testing purposes.

Integrity

All code submissions will be checked for similarity and plagiarism. All code in your submission that was not written by you (or was written by you for a previous assignment) must be correctly referenced in IEEE format in your code. This includes code written by generative AI.

Note that *the course coordinator and casual academic may interview some students if there are any suspicious behaviours.*

Marking Breakdown

- No partial marks will be given.

Question No.		Marks
1.	Can correctly generate derived key.	2
	Can correctly encrypt/decrypt message using AES-ECB.	3
	Can correctly encrypt/decrypt message using AES-CBC.	3
	Can correctly encrypt/decrypt message using AES-CFB.	3
	Can correctly encrypt/decrypt message using AES-OFB.	3
	Can correctly encrypt/decrypt message using AES-CTR.	3
	Can correctly encrypt/decrypt message using AES-GCM.	3
	Sub-total	**20**
2.	Can correctly generate Alice's public key.	3
	Can correctly generate Bob's public key.	3
	Can correctly generate a shared secret.	4
	Sub-total	**10**
3.	Can correctly generate the public key and private key.	10
	Can correctly encrypt message using RSA.	5
	Can correctly decrypt message using RSA.	5
	Sub-total	**20**
4.	Can generate the correct certificate.	10
	Sub-total	**10**
5.	Can correctly generate ESP UDP packet using "tunnel" mode.	10
	Can correctly generate ESP UDP packet using "transport" mode.	10
	Can correctly generate ESP TCP packet using "tunnel" mode.	10
	Can correctly generate ESP TCP packet using "transport" mode.	10
	Sub-total	**40**
	Total	**100**