Task 1

Implement the encryption and decryption processes described in Figure 1. The “AES Encryptor” block is the encryptor of an AES-ECB cipher. You can instantiate such an encryptor using the appropriate modules from cryptography.io.

The description of the functions, parameters, and values is provided below in the Python code.

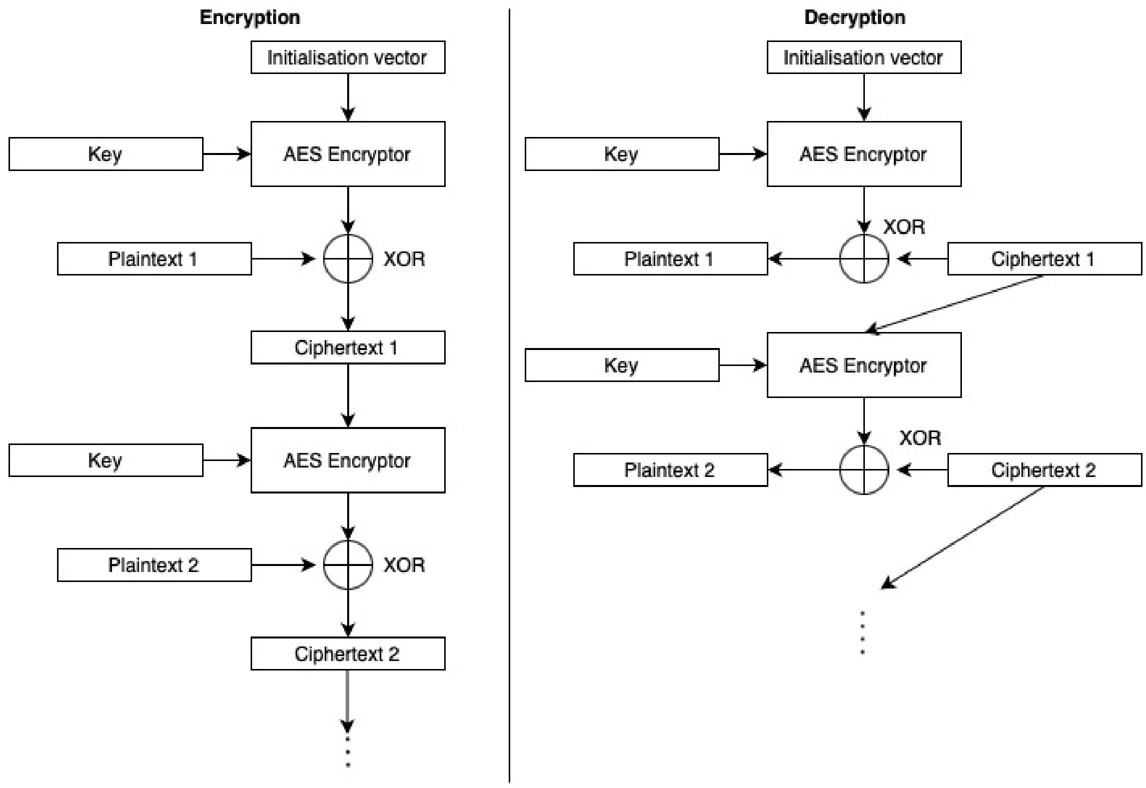

Figure 1 - Encryption and decryption process of an AES mode

```
# -- START OF YOUR CODERUNNER SUBMISSION CODE
# INCLUDE MODULES

# INCLUDE HELPER FUNCTIONS YOU IMPLEMENT

'''
:param key: str: The hexadecimal value of a key to be used for encryption
:param iv: str: The hexadecimal value of an initialisation vector to be
used for encryption
:param data: str: The data to be encrypted
:return: str: The hexadecimal value of encrypted data
'''
def Encrypt(key: str, iv: str, data: str) -> str:

      # TODO YOUR IMPLEMENTATION

      return # TODO: The hexadecimal value of encrypted data
```

```
'''
:param key: str: The hexadecimal value of a key to be used for decryption
:param iv: str: The hexadecimal value of the initialisation vector to be
used for decryption
:param data: str: The hexadecimal value of the data to be decrypted
:return: str: The decrypted data in UTF-8 format
'''
def Decrypt(key: str, iv: str, data: str) -> str:

      # TODO YOUR IMPLEMENTATION

      return # TODO: The decrypted data in UTF-8 format


# -- END OF YOUR CODERUNNER SUBMISSION CODE

# You can test your code in your system (NOT IN YOUR CODERUNNER SUBMISSION)
as follows:

# Main
if __name__ == "__main__":
    # Task 1

    key = "2b7e151628aed2a6abf7158809cf4f3c"
    iv = "000102030405060708090a0b0c0d0e0f"
    text = "Hello World"

    ct = Encrypt(key, iv, text)
    pt = Decrypt(key, iv, ct)

    print(ct)
    print(pt)

'''
TEST CASE OUTPUT:
189b0ba0f64d65d9a86553
Hello World
'''
```

Marking scheme:
* Correct implementation [15% for encryption and 15% for decryption]

Total score for Task 1: 30%

TASK 2

The AES encryption mode in Figure 2 is used to encrypt the following plaintext:

Plaintext1 = "This is your General. Hold position until further orders. I repeat, hold position."

This results in the following ciphertext:

```
Ciphertext =
b'\xf2\x0f\x97#$D\xa8\xda\xa0\xe4`:TQ\x82%\xc3\x15\x9f<*\r\x93\x95\xb5\xef5
8\x1be\x8e?\xcf\x08\x90pqC\xaf\x93\xb5\xabs=\x06b\x8f.\xd4G\x91"`H\xa9\x89\
xf7\xab\\h\x06s\x97.\xc7\x13\xd2plB\xb7\x9e\xf9\xfbz;\x1db\x8e$\xc8I'
```

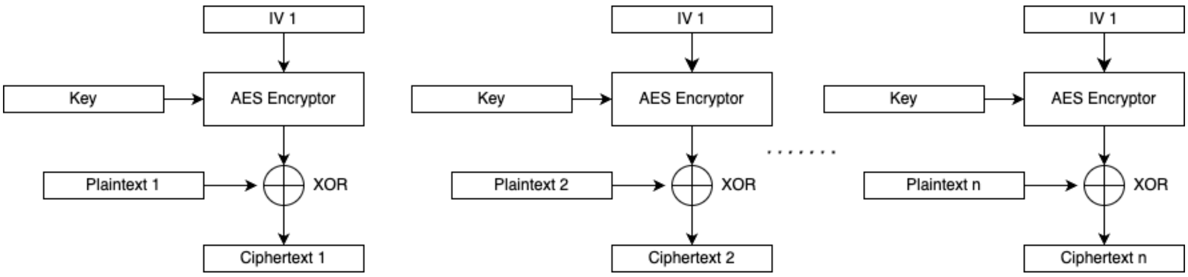

Figure 2 - The AES mode used in task 2

Describe a cryptographic attack you can implement – include this information as comments in your code.

Use your attack to generate the ciphertext for the following plaintext:

ForgedPlaintext = "This is your General. Proceed with the attack at dawn. I repeat, proceed with the attack at dawn."

```
# -- START OF YOUR CODERUNNER SUBMISSION CODE
# INCLUDE MODULES

# INCLUDE HELPER FUNCTIONS YOU IMPLEMENT


'''
:param plaintext1: str: This is Plaintext1
:param ciphertext1: bytes: This is the ciphertext of Plaintext1
:param plaintext2: str: This is Plaintext2
:return: bytes: The ciphertext of Plaintext2
'''
def attackAESMode(plaintext1: str, ciphertext1: bytes, plaintext2: str) ->
bytes:

      # TODO YOUR IMPLEMENTATION

    return # TODO: The ciphertext of Plaintext2 in bytes


# -- END OF YOUR CODERUNNER SUBMISSION CODE
```

```
# You can test your code in your system (NOT IN YOUR CODERUNNER SUBMISSION)
as follows:


# Main
if __name__ == "__main__":

    pt1 = "This is your General. Hold position until further orders. I
repeat, hold position."
    ct1 =
b'\xf2\x0f\x97#$D\xa8\xda\xa0\xe4`:TQ\x82%\xc3\x15\x9f<*\r\x93\x95\xb5\xef5
8\x1be\x8e?\xcf\x08\x90pqC\xaf\x93\xb5\xabs=\x06b\x8f.\xd4G\x91"`H\xa9\x89\
xf7\xab\\h\x06s\x97.\xc7\x13\xd2plB\xb7\x9e\xf9\xfbz;\x1db\x8e$\xc8I'
    pt2 = "This is your General. Proceed with the attack at dawn. I repeat,
proceed with the attack at dawn."

    print(attackAESMode(pt1, ct1, pt2))
```

Marking scheme:
* Clearly explain the attack [15%]
* Correctly implement the attack to generate the required ciphertext [15%]

Total score for Task 2: 30%

TASK 3

Implement a function myHash that will use SHA-256 and calculate the OUTPUT HASH value of any input value following the steps in Figure 3.

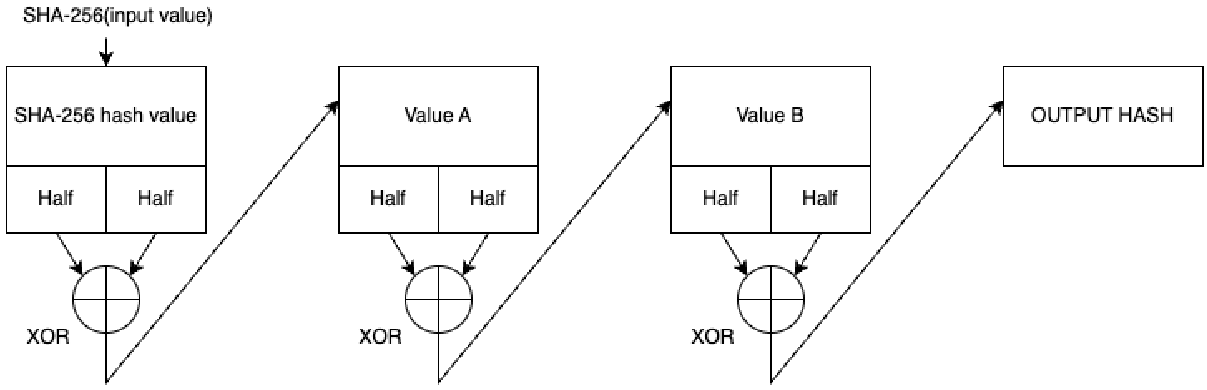

Figure 3 - Steps to implement your hash function

Describe an attack to check if myHash is secure or not – include this information as comments in your code.

Implement your attack in a function myAttack. The function should return YES if the function is secure and NO otherwise. The result (YES or NO) should be the outcome of an actual attack that detects this.

```
# -- START OF YOUR CODERUNNER SUBMISSION CODE
# INCLUDE MODULES

# INCLUDE HELPER FUNCTIONS YOU IMPLEMENT

'''
param: data: bytes: The data to be hashed
return: bytes: The truncated hash
'''
def myHash(data: bytes) -> bytes:

      # TODO YOUR IMPLEMENTATION

      return # OUTPUT HASH

'''
return: str: Return YES if myHash is secure and NO otherwise
'''
def myAttack() -> str:

      # TODO YOUR IMPLEMENTATION

      return # YES or NO


# -- END OF YOUR CODERUNNER SUBMISSION CODE
```

```
# You can test your code in your system (NOT IN YOUR CODERUNNER SUBMISSION)
as follows:


if __name__ == "__main__":
    print(myHash(b”a”)
    print(myAttack())

'''
TEST CASE OUTPUT FOR myHash():
print(myHash(b”a”)
b'\xc5\xf9O\x92'
'''

Marking scheme:
* Correctly implement myHash [10%]
* Clearly explain the attack [15%]
* Correctly implement the myAttack to detect the security of myHash [15%]

Total score for Task 3: 40%
```