Module: Modern Cryptography

Exam: Summer 2024

Lecturer: Dr. Martin Harrigan

Instructions:

- Answer any four questions.
- You have 2.5 hours to complete the exam.
- The exam is worth 50% of your final grade.

Question 1 (20 marks)

(a) Consider a Feistel scheme with 8-bit blocks and two rounds. The round function for round i is $f_i(x,K) = x \oplus K_i$ where x is the input to the round, K_1 and K_2 are the first and the last four bits of the main key K respectively, and $\oplus$ is the XOR operation. If the plaintext is 01010101 and the key is 00001111, what is the ciphertext?

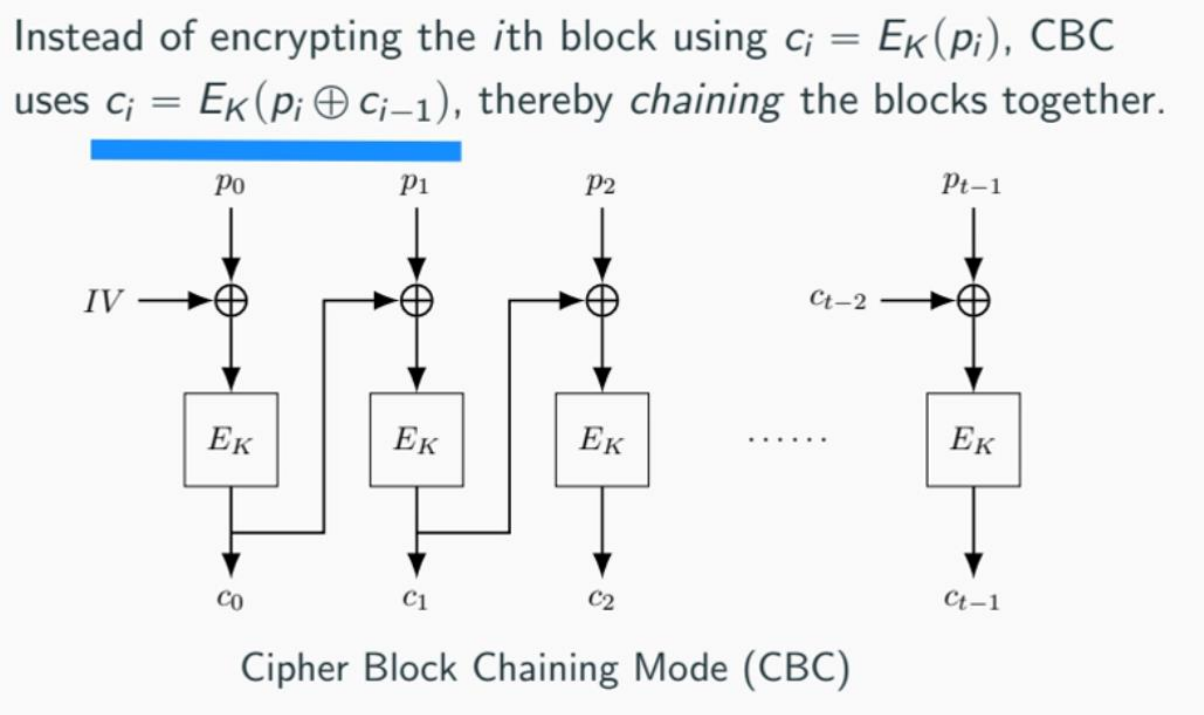
(8 marks)

1. **Key Splitting**: The main key (K = 00001111) is split into two 4-bit keys:
    - (K1 = 0000)
    - (K2 = 1111)
2. **Initial Split**: The plaintext (P = 01010101) is split into two 4-bit halves:
    - Left half (L0 = 0101)
    - Right half (R0 = 0101)
3. **First Round**:
    - Compute (R1 = L0 ¥oplus K1 = 0101 ¥oplus 0000 = 0101)
    - The new left half (L1 = R0 = 0101)
4. **Second Round**:
    - Compute (R2 = L1 ¥oplus K2 = 0101 ¥oplus 1111 = 1010)
    - The new left half (L2 = R1 = 0101)

5. **Combine Halves**: The ciphertext (C) is the combination of (L2) and (R2):
    - (C = L2 || R2 = 0101 || 1010 = 01011010)

So, the ciphertext is **01011010**.

(b) Draw a diagram that illustrates the Cipher Block Chaining (CBC) mode of operation for decryption. Your diagram should show the flow of data from the plaintext blocks to the decryption processes, and to the ciphertext blocks.
(8 marks)



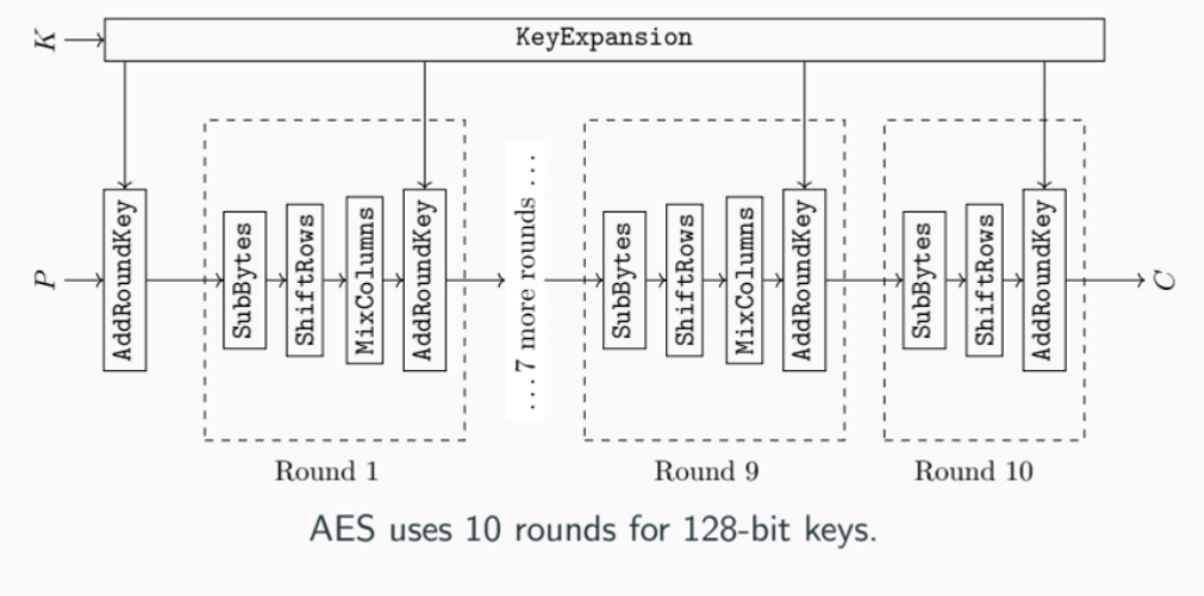
(c) Padding expands a message to fill a complete block by adding extra bytes to the end of the message. Assuming a 16-byte block, describe what gets added to a message by the popular padding scheme PKCS#7 when:
(i) there is one byte of data leftover;
(ii) there are two bytes of data leftover;
(iii) there are fifteen bytes of data leftover; and
(iv) there are no bytes of data leftover, i.e., the length of the message is a multiple of sixteen.
(4 marks)

- **One byte of data leftover**: 15 bytes of padding with the value 0x0F 1.
- **Two bytes of data leftover**: 14 bytes of padding with the value 0x0E 1.
- **Fifteen bytes of data leftover**: 1 byte of padding with the value 0x01 1.
- **No bytes of data leftover**: 16 bytes of padding with the value 0x10 1.

Question 2 (20 marks)

(a) AES uses a substitution-permutation network with a number of rounds that depends on the size of the key. Draw a sketch of the network when the key is 256- bits. Your sketch should include the number of rounds, the SubBytes, ShiftRows, MixColumns and AddRoundKey functions as they occur in the first and last rounds, the KeyExpansion function, and the flow of data from a key and plaintext block to a ciphertext block.

(6 marks)



AES uses 10 rounds for 128-bit keys.

AES 256 would be 14 rounds. Final round the mix columns function is omitted.

(b) The key schedule for AES, KeyExpansion, derives the round keys from the main key. Is it possible to recover the main key from a round key?

(4 marks)

The key schedule for AES, known as KeyExpansion, generates round keys from the main key. The process involves complex transformations that make it computationally infeasible to reverse-engineer the main key from a single round key.

(c) Fast implementations of AES use table-based implementations and/or native instructions. Briefly describe one of these optimisations.
(6 marks)

One popular optimization for fast implementations of AES is the use of table-based implementations. This technique involves precomputing and storing the results of the SubBytes and MixColumns transformations in lookup tables. Another optimization involves using native instructions provided by modern processors.

(d) What are the practical implications of choosing between AES key sizes in terms of encryption strength and computational efficiency. Specifically, address how the increase in key size from 128 bits to 256 bits impacts the resistance of AES to brute-force attacks.
(4 marks)

Encryption Strength:

- **128-bit key**: This key size offers a high level of security and is generally considered sufficient for most applications. It provides (2^{128}) possible combinations, making brute-force attacks practically infeasible with current technology.
- **256-bit key**: Increasing the key size to 256 bits significantly enhances security, offering (2^{256}) possible combinations. This makes brute-force attacks even more infeasible, providing a higher level of protection against future advancements in computational power.

Computational Efficiency:

- **128-bit key**: AES with a 128-bit key is faster and requires less computational power compared to larger key sizes. This makes it suitable for applications where performance is critical, such as real-time encryption and decryption.
- **256-bit key**: While AES with a 256-bit key provides stronger security, it requires more computational resources. This can result in slower performance, which may be a consideration for applications with stringent performance requirements.

Impact on Brute-Force Attacks:

- The increase in key size from 128 bits to 256 bits exponentially increases the resistance to brute-force attacks.

Question 3 (20 marks)

Python's cryptography package provides many cryptographic primitives such as block ciphers, stream ciphers, hashing functions, keyed-hashing functions, etc. All of the following code snippets, except the last one, using functionality from this package.

(a) In the following code snippet, what is the purpose of the length, n, r and p arguments? In what circumstances would they need to be changed?

```
import os
from cryptography.hazmat.primitives.kdf.scrypt import Scrypt

salt = os.urandom(16)
kdf = Scrypt(salt=salt, length=32, n=2**14, r=8, p=1)
```

(4 marks)

1. **Length**: This argument specifies the length of the derived key in bytes. In the provided code snippet, length=32 means the derived key will be 32 bytes long. You might change this value depending on the required key length.
2. **n**: This is the CPU/memory cost parameter. It must be a power of two and controls the overall computational cost of the key derivation process. You might change this value to balance security and performance based on your application's requirements.
3. **r**: This is the block size parameter. It affects the memory usage and the parallelism of the algorithm. You might change this value if you need to fine-tune the memory requirements
4. **p**: This is the parallelization parameter. It controls the number of parallel instances of the Scrypt algorithm. You might change this value to take advantage of parallel processing capabilities

(b) The following code snippet signs a message (message) using a private-key (private key). Why are there two references to a hash function (hashes.SHA256())?

```
from cryptography.hazmat.primitives import hashes
from cryptography.hazmat.primitives.asymmetric import padding

# [snip]

signature = private_key.sign(
message,
padding.PSS(
mgf=padding.MGF1(hashes.SHA256()),
salt_length=padding.PSS.MAX_LENGTH
),
hashes.SHA256()
)
)
```

(4 marks)

The first reference to hashes.SHA256() is used within the padding. The second reference to hashes.SHA256() is used to specify the hash function that will be applied to the message itself before signing.

(b) In the following code snippet, is the value generated by os.urandom(16) cryptographically random? Is it appropriate to use this value as a nonce for AES in CTR mode, or should we take the narrower definition of a nonce, and keep track of all values we used in the past?

```
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms,
modes
import os

# [snip]

nonce_bytes = os.urandom(16)

aes_ctr_cipher = Cipher(algorithms.AES(key_bytes),
```

mode=modes.CTR(nonce_bytes))
(4 marks)
The value generated by os.urandom(16) is random.
While os.urandom(16) provides a high level of randomness, it does not guarantee uniqueness. Therefore, it is generally recommended to keep track of all previously used nonces

(d) When experimenting with the implementation of the AES block cipher provided by cryptography/hazmat/primitives/ciphers/algorithms.py, we used two datasets provided by the National Institute of Standards and Technology (NIST): Known Answer Tests (KATs) and Multiblock Message Tests (MMTs). What is the primary difference between KATs and MMTs? What type of implementation error could be identified by MMTs but not by KATs?
(4 marks)
KATs are designed to verify the correctness of a cryptographic algorithm's implementation focusing on individual operations. MMTs are used to test the implementation of cryptographic algorithms over multiple blocks of data. An implementation error would be related to the handling of data across multiple blocks

(e) What cryptographic primitive is being implemented by the following code snippet?

```
def encrypt_decrypt(input, key):
output = bytearray()
for (b, c) in zip(input, key):
output.append(b ^ c)
return bytes(output)
```

(4 marks)
The cryptographic primitive being implemented by the provided code snippet is the **XOR cipher**

Question 4 (20 marks)

(a) What is meant by the terms attack model, security goal and security notion?
(6 marks)

Attack model refers to the assumed capabilities of an adversary i.e. what they can and can't do. Security goals refer to things a cryptographer can implement to secure a cipher or encryption method and specifies what the system is designed to protect and a security notion is the combination of the two to provide a measurable evaluation of the system.

(b) Suppose you are required to design a web portal to allow students to upload photographs of their medical certificates. State an appropriate security goal and list any three aspects of your attack model.
(5 marks)

Goal – ensure confidentiality, integrity and availability of data.
Attack model – unauthorised access, data tampering, Denial of serivce

(c) Explain Kerckhoffs's Principle.
(3 marks)

Kerckhoffs principles states that you should assume an adversary knows everything about your encryption method and that the security of a cryptographic system should rely only on keeping the key secret.

(d) In the context of black-box attack models, what can an attacker do in each of the following attack models:
(i) Ciphertext-only attack (COA)
(ii) Known-plaintext attack (KPA)
(iii) Chosen-plaintext attack (CPA)
(iv) Chosen-ciphertext attack (CCA)
(6 marks)

COA: only has access to the ciphertexts.
KPA: has access to both the plaintexts and their corresponding ciphertexts.

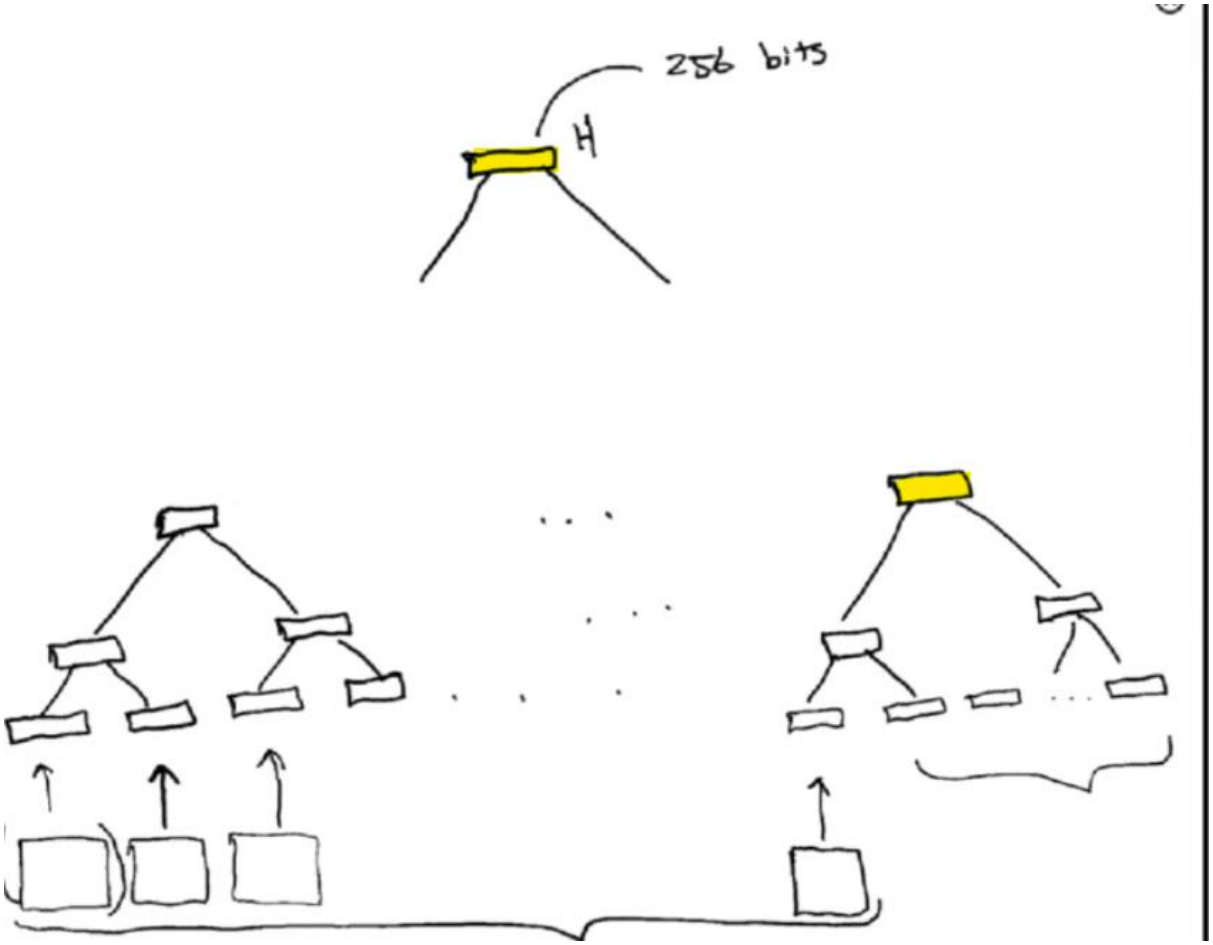
CPA can choose arbitrary plaintexts and obtain their corresponding ciphertexts. This allows them to analyze how the encryption algorithm processes different plaintexts
CCA can choose arbitrary ciphertexts and obtain their corresponding plaintexts. This helps them understand how the decryption algorithm works

Question 5 (20 marks)
(a) In 2017, researchers from CWI Amsterdam and Google Research published two different PDF files that, when hashed using the SHA-1 algorithm, produce the same 160-bit hash. What property of hash functions did this attack break for SHA-1?
(2 marks)
Collision resistance

(b) Name two popular hash functions that are considered secure as of today.
(3 marks)
SHA256
SHA-3

(c) A Merkle tree is a tree where every leaf node contains the hash of a piece of data. What does every non-leaf node contain?
(3 marks)
non-leaf nodes contain the hash of the concatenation of their child nodes hashes.

(d) Draw and annotate the structure of a Merkle tree that fulfils the following criteria:
(i) The tree is binary, i.e., every non-leaf node has two children.
(ii) The tree has sixteen leaf nodes, i.e., it can store the hashes of at most sixteen pieces of data.
(iii) The root node and any one of the leaf nodes are highlighted.
(6 marks)



Something like above 😊

(e) Using your answer from (d), describe a Merkle proof that your tree with the highlighted root contains the highlighted leaf node. How many nodes will the Merkle proof require? Identify the nodes that will be part of the Merkle proof. (6 marks)

Since the tree is binary and has sixteen leaf nodes, the path from any leaf node to the root will involve four levels (including the leaf level). Therefore, the Merkle proof will require four sibling nodes' hashes.