الجمهورية الجزائرية الديموقرطية الشعبية
وزارة التعليم العالي والبحث العلمي

جامعة الشهيد حمه لخضر بالوادي — السنة الدراسية: **2026/2025**
كلية العلوم الدقيقة — المدة: ســـــاعة ونصف
قسم الإعلام الآلي — السنة: ثانية ماستر (IoT & Cybersecurity)

الاجابة النموذجية لامتحان السداسي الأول في مقياس: **Blockchain and MPC**

Exercise 1: Multiple Choice Questions (MCQ) (7.5 P)

1. Which techniques are commonly used in MPC protocols?	
• A) Secret sharing	X
• B) Proof of Work	
• C) Digital wallets	
• D) Homomorphic encryption	X
2. Which statements about Zero-Knowledge Proofs (ZKP) are correct?	
• A) They guarantee consensus	
• B) They allow proving knowledge without revealing the secret	X
• C) They enhance privacy in blockchain systems	X
• D) They always require interaction	
3. Which statements correctly describe a Merkle Tree?	
• A) It stores transactions in plain text	
• B) It requires a trusted third party	
• C) It allows efficient verification of data integrity	X
• D) A single hash represents all underlying data	X
4. Which of the following are valid uses of hash functions in blockchain?	
• A) Linking blocks together	X
• B) Encrypting transactions	
• C) Verifying data integrity	X
• D) Generating random private keys	
5. Which statements about sharding are correct?	
• A) It increases decentralization automatically	
• B) It is a scalability technique	X
• C) It eliminates the need for consensus	
• D) It divides the network into smaller parts	X
6. Which Ethereum feature allows smart contracts to react to external actions?	
• A) Events	
• B) Oracles	X
• C) Gas	
• D) Nonces	
7. Which protocol is commonly used for secure two-party computation?	
• A) Diffie-Hellman	
• B) RSA	
• C) SHA-256	
• D) Yao’s Garbled Circuits	X
8. Which statement best describes layer-2 solutions?	
• A) They replace the base blockchain	
• B) They eliminate consensus mechanisms	
• C) They improve scalability off-chain	X
• D) They reduce cryptographic security	
9. Which property ensures that transaction data has not been modified?	
• A) Integrity	X
• B) Confidentiality	
• C) Scalability	
• D) Availability	

10. What is the significance of a threshold in secret sharing for MPC?	
• A) It controls the computation speed	
• B) It defines the total number of parties in the computation	
• C) It determines the number of participants required to reconstruct the secret	**X**
• D) It dictates the security of the communication channels	

Exercise 2: Shamir's Secret Sharing (6.5 P)

Shamir's Secret Sharing is a cryptographic method that allows a secret to be split into several parts (shares) such that only a minimum number of participants can recover the secret. Let the secret be S=10. The secret is shared among n = 5 participants with a threshold t = 3.

Questions:

1) Explain in a few sentences the goal of **Shamir's Secret Sharing**.

Shamir's Secret Sharing splits a secret S into n "shares" so that any t shares can reconstruct S, but any set of fewer than t shares reveals nothing (information-theoretically) about S. It's commonly used to distribute trust (e.g., no single person can unlock a master key).

2) Construct a random polynomial ***f(x)*** of degree 2 such that: ***f(0)=S***

$$f(x)=10+4x+7x^2$$

3) Using your polynomial from Question 2, compute the shares for the five participants: $(x_i, f(xi))$ for x_i=1,2,3,4,5

- $f(1)=10+4(1)+7(1*1)=10+4+7=21$
- $f(2)=10+4(2)+7(2*2)=10+8+28=46$
- $f(3)=10+4(3)+7(3*3)=10+12+63=85$
- $f(4)=10+4(4)+7(4*4)=10+16+112=138$
- $f(5)=10+4(5)+7(5*5)=10+20+175=205$

shares $(x_i, f(xi))$: (1,21) , (2,46) , (3,85) , (4,138) , (5,205)

4) Using any three shares, explain how the secret can be reconstructed.

Since the secret was created using a degree-2 polynomial, it has three unknown coefficients (a_0, a_1, a_2). Each share gives one equation involving these coefficients. With three shares, you get three equations, which is enough to solve for all coefficients of the polynomial. Once the polynomial is recovered, the secret is found by evaluating it at x=0, which gives S=10. We use the three shares: (1,21), (2,46), (3,85) $f(x)=a_0+a_1*x+a_2*x^2$. Using the three shares, we get three equations:

- $a_0+a_1+a_2=21$
- $a_0+2a_1+4a_2=46$
- $a_0+3a_1+9a_2=85$

Finally, solving the three equations allows us to find the polynomial and the secret.

5) Explain why two shares are not sufficient to recover the secret.

With only two shares, you only know two points on a quadratic polynomial. But a degree-2 polynomial has three unknown coefficients (a_0, a_1, a_2). Two points give only two equations, so there are infinitely many quadratics that fit those two shares each potentially giving a different value of f(0) (a different "secret"). Intuitively: with t=3, the scheme is designed so that fewer than 3 shares leave the secret underdetermined, so participants learn nothing definitive about S.

Exercise 3: Smart Contract (06 P)

1) **Code Explanation:**

- Explain the purpose of the **msg.sender** instruction.
  msg.sender is the address that called the function (the caller).
- Explain step by step what the **refund(uint amount)** function does.

1. ***Checks that the caller (msg.sender) has contributed at least amount.***
2. ***Subtracts amount from the caller's saved contribution in the mapping.***
3. ***Sends amount of Ether back to the caller's address.***

2) **Optimization and Improvement:** Add what is necessary for the contract to store and display:

- The largest contribution amounts.
- The address of the contributor who made it.

```
            // SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;
contract TripFund {
   address public manager;
   mapping(address => uint) public contributions;
   // NEW: largest contribution tracking
   uint public largestContribution;
   address public largestContributor;
   modifier onlyManager() {
      require(msg.sender == manager, "Only manager");
    _;   }
   constructor() {
      manager = msg.sender;    }
   function contribute() public payable {
      require(msg.value > 0, "Contribution must be greater than 0");
      contributions[msg.sender] += msg.value;
      // NEW: update largest contributor if this person's total is now the biggest
      if (contributions[msg.sender] > largestContribution) {
         largestContribution = contributions[msg.sender];
         largestContributor = msg.sender;      }    }
   function refund(uint amount) public {
      require(contributions[msg.sender] >= amount, "Insufficient contribution");
      contributions[msg.sender] -= amount;
      payable(msg.sender).transfer(amount);    }
   function myContribution() public view returns (uint) {
      return contributions[msg.sender];    }
   // NEW: display largest contribution info
   function getLargest() public view returns (address contributor, uint amount) {
      return (largestContributor, largestContribution);
   }}
```