Lab Session: Hash Function Experiments

BSc Blockchain, Crypto Economy & NFTs

Course Instructor

Module A: Blockchain Foundations

Learning Objectives

By the end of this lab session, you will be able to:

- Use Python's `hashlib` library to compute SHA-256 hashes
- Demonstrate the avalanche effect experimentally
- Understand collision resistance through brute-force attempts
- Build a simple proof-of-work mining simulation
- Verify hash-based integrity in practical scenarios

Lab Overview

Structure:

1. Environment setup (5 minutes)
2. Exercise 1: Basic hashing (15 minutes)
3. Exercise 2: Avalanche effect demonstration (20 minutes)
4. Exercise 3: Simple proof-of-work mining (30 minutes)
5. Exercise 4: Hash chain verification (20 minutes)
6. Wrap-up and deliverables (10 minutes)

Total Duration: 90 minutes

Prerequisites:

- Python 3.8+ installed
- Basic Python programming knowledge
- Understanding of hash functions from Lesson 3

Environment Setup

Required Libraries:

- `hashlib` (standard library)
- `time` (standard library)
- `json` (standard library)

Setup Instructions:

1. Create a new directory: `hash_lab`
2. Create a Python file: `hash_experiments.py`
3. Import required libraries
4. Test installation by computing a simple hash

Verification Command:
Check that running a basic hash function produces the expected output for the string "Hello, Blockchain!"

Exercise 1: Basic Hashing

Objectives:

- Compute SHA-256 hashes of strings
- Compute SHA-256 hashes of files
- Compare different hash algorithms (MD5, SHA-1, SHA-256)

Tasks:

1. Create a function that takes a string and returns its SHA-256 hash
2. Hash the following inputs:
    - "Blockchain"
    - "blockchain" (lowercase)
    - "Blockchain " (with trailing space)
3. Compare the outputs and observe differences
4. Create a text file and compute its hash
5. Modify one character in the file and recompute

Expected Outcome: Understand that tiny input changes produce completely different hashes

Exercise 2: Avalanche Effect Demonstration

Objective: Experimentally verify the avalanche effect

The Avalanche Effect:

- Changing a single bit in input should change approximately 50% of output bits
- Critical property for cryptographic security
- Makes pattern detection impossible

Tasks:

1. Create a function that compares two hashes bit-by-bit
2. Hash the string "The quick brown fox jumps over the lazy dog"
3. Hash the string "The quick brown fox jumps over the lazy dof" (last letter changed)
4. Count how many bits differ between the two hashes
5. Calculate the percentage of bits that changed
6. Repeat with other single-character modifications

Expected Result: Approximately 50% of bits should differ

Exercise 2: Implementation Hints

Converting Hash to Binary:

- Hash output is hexadecimal (64 characters for SHA-256)
- Convert each hex digit to 4 binary bits
- Total: 256 bits

Bit Comparison:

- Use XOR operation to find differing bits
- Count the number of 1s in the XOR result
- Divide by 256 to get percentage

Test Cases:

- Same string should have 0% difference
- One character change should have 50% difference
- Completely different strings should have 50% difference

Exercise 3: Simple Proof-of-Work Mining

Objective: Build a basic mining simulation

Concept Review:

- Mining = finding a nonce such that hash(block_data + nonce) meets difficulty target
- Difficulty target = hash must start with N leading zeros
- No shortcut: must try different nonces sequentially

Tasks:

1. Create a block structure with:
    - Block number
    - Data (e.g., "Transaction: Alice sends 10 BTC to Bob")
    - Previous block hash
    - Nonce (initially 0)
2. Implement a mining function that increments nonce until hash starts with N zeros
3. Mine blocks with difficulty 1, 2, 3, 4
4. Record time taken and nonces tried for each difficulty

Exercise 3: Implementation Structure

Block Data Structure:

- Combine all fields into a single string
- Format: "block_number:data:previous_hash:nonce"
- Hash this concatenated string

Mining Algorithm:

1. Start with nonce = 0
2. Compute hash of block data + nonce
3. Check if hash meets difficulty (starts with N zeros)
4. If yes: return nonce
5. If no: increment nonce and repeat

Difficulty Verification:

- Difficulty 1: hash starts with "0"
- Difficulty 2: hash starts with "00"
- Difficulty 3: hash starts with "000"
- Each additional zero increases difficulty by 16x

Exercise 3: Expected Results

Performance Observations:

Difficulty	Avg. Attempts	Approx. Time
1 zero	16	¡ 1 second
2 zeros	256	¡ 1 second
3 zeros	4,096	1-5 seconds
4 zeros	65,536	10-60 seconds
5 zeros	1,048,576	5-20 minutes

Key Insights:

- Exponential growth in computation time
- No way to predict nonce value
- Bitcoin uses difficulty 19 leading zeros (current)
- Real mining uses specialized hardware (ASICs)

Exercise 4: Hash Chain Verification

Objective: Build and verify a simple blockchain

Tasks:

1. Create a genesis block (first block with previous_hash = "0")
2. Create a function to add new blocks that:
   - Takes previous block's hash
   - Includes new transaction data
   - Mines with difficulty 2
3. Build a chain of 5 blocks
4. Implement a verification function that checks:
   - Each block's hash is valid
   - Each block correctly references previous hash
   - Chain integrity from genesis to tip
5. Tamper with block 3's data and observe verification failure

Expected Outcome: Understand immutability through hash chains

Exercise 4: Tamper Detection

Scenario: Attacker modifies a transaction in the middle of the chain

Experiment:

1. Build a valid 5-block chain
2. Verify the entire chain (should pass)
3. Modify the data in block 3
4. Run verification again (should fail)

Why Verification Fails:

- Changing block 3's data changes its hash
- Block 4 references the old hash of block 3
- Hash chain breaks at this link
- Verification detects mismatch

Discussion Question: What would an attacker need to do to successfully modify block 3?

Answer: Re-mine block 3 and all subsequent blocks (computationally expensive)

Lab Deliverables

Submit the following:

1. **Python script** (`hash_experiments.py`) containing:
    - All four exercises implemented
    - Clear function names and comments
    - Test cases demonstrating functionality
2. **Lab report** (PDF, 2-3 pages) including:
    - Exercise 2: Avalanche effect results (bit difference percentages)
    - Exercise 3: Mining performance table (difficulty vs. time)
    - Exercise 4: Screenshot of chain verification before and after tampering
    - Brief reflection on blockchain immutability
3. **Bonus (optional):** Implement SHA-1 collision detection using known collision examples

Submission Deadline: One week from lab session date

Grading: Pass/Fail based on completeness and correctness

Key Takeaways

- Hash functions are easy to compute but infeasible to reverse
- The avalanche effect ensures that small input changes produce unpredictable output changes
- Proof-of-work mining is computationally expensive by design
- Hash chains create tamper-evident data structures
- Blockchain immutability comes from the cost of re-mining modified blocks

Real-World Applications:

- Bitcoin/Ethereum mining
- Git version control (commit hashes)
- File integrity verification (checksums)
- Password storage (salted hashes)

Discussion Questions

1. Why is it important that hash functions are deterministic (same input always produces same output)?
2. In your mining experiments, did you notice any patterns in which nonces produced valid hashes?
3. If you wanted to modify a transaction in block 3 of a 100-block chain, approximately how many hashes would you need to recompute?
4. How does increasing the mining difficulty affect the security of a blockchain?
5. What would happen if a hash function did not exhibit the avalanche effect?

Next Lesson Preview: L05 Public Key Cryptography

Topics to be covered:

- Asymmetric encryption fundamentals
- Elliptic Curve Digital Signature Algorithm (ECDSA)
- Digital signatures and verification
- Public/private key pair generation
- Bitcoin and Ethereum address derivation
- Key security best practices

Preparation:

- Review symmetric vs. asymmetric encryption concepts
- Read about public key infrastructure (PKI)
- Explore how digital signatures differ from physical signatures