Lab Session: Hash Function Experiments

BSc Blockchain, Crypto Economy & NFTs

Course Instructor

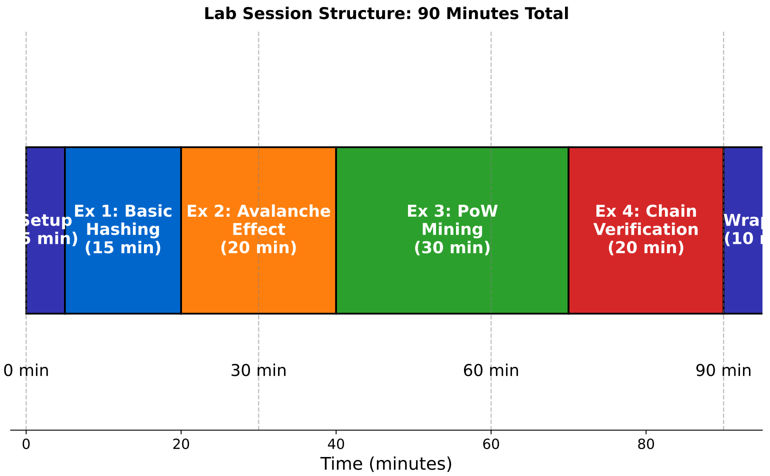
Module A: Blockchain Foundations

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 Session Structure



Total duration: 90 minutes with 4 hands-on exercises

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: Hash "Hello, Blockchain!" and verify output

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: “Blockchain”, “blockchain”, “Blockchain ” (with space)
3. Compare the outputs and observe differences
4. Create a text file and compute its hash
5. Modify one character and recompute

Expected Outcome: 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 should change approximately 50% of output bits
- Critical property for cryptographic security

Tasks:

1. Create a function that compares two hashes bit-by-bit
2. Hash "The quick brown fox jumps over the lazy dog"
3. Hash "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

Expected Result: Approximately 50% of bits should differ

Exercise 3: Simple Proof-of-Work Mining

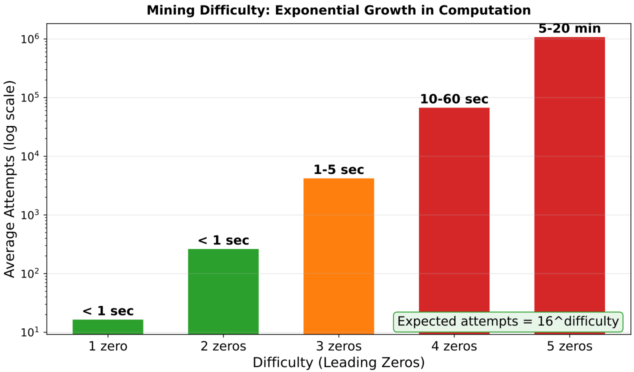
Objective: Build a basic mining simulation

Concept Review:

- Mining = finding a nonce such that hash 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 (number, data, previous hash, nonce)
2. Implement mining function that increments nonce until valid
3. Mine blocks with difficulty 1, 2, 3, 4
4. Record time taken and nonces tried for each difficulty



Each additional zero increases difficulty by approximately 16x

Exercise 4: Hash Chain Verification

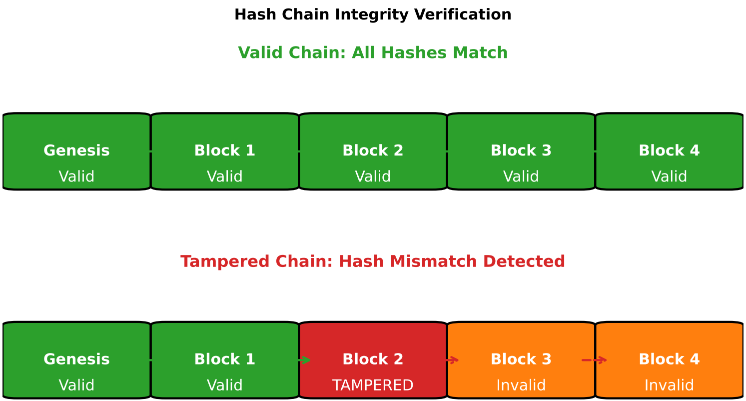
Objective: Build and verify a simple blockchain

Tasks:

1. Create a genesis block (first block with previous_hash = "0")
2. Create a function to add new blocks
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
5. Tamper with block 3's data and observe verification failure

Expected Outcome: Understand immutability through hash chains

Chain Integrity Verification



Tampering breaks hash links, making modifications immediately detectable

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:
   - Avalanche effect results (bit difference percentages)
   - Mining performance table (difficulty vs. time)
   - Screenshot of chain verification before and after tampering

Submission Deadline: One week from lab session date

Key Takeaways

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

Real-World Applications:

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

Discussion Questions

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