

Lab Session: Hash Function Experiments

BSc Blockchain, Crypto Economy & NFTs

Course Instructor

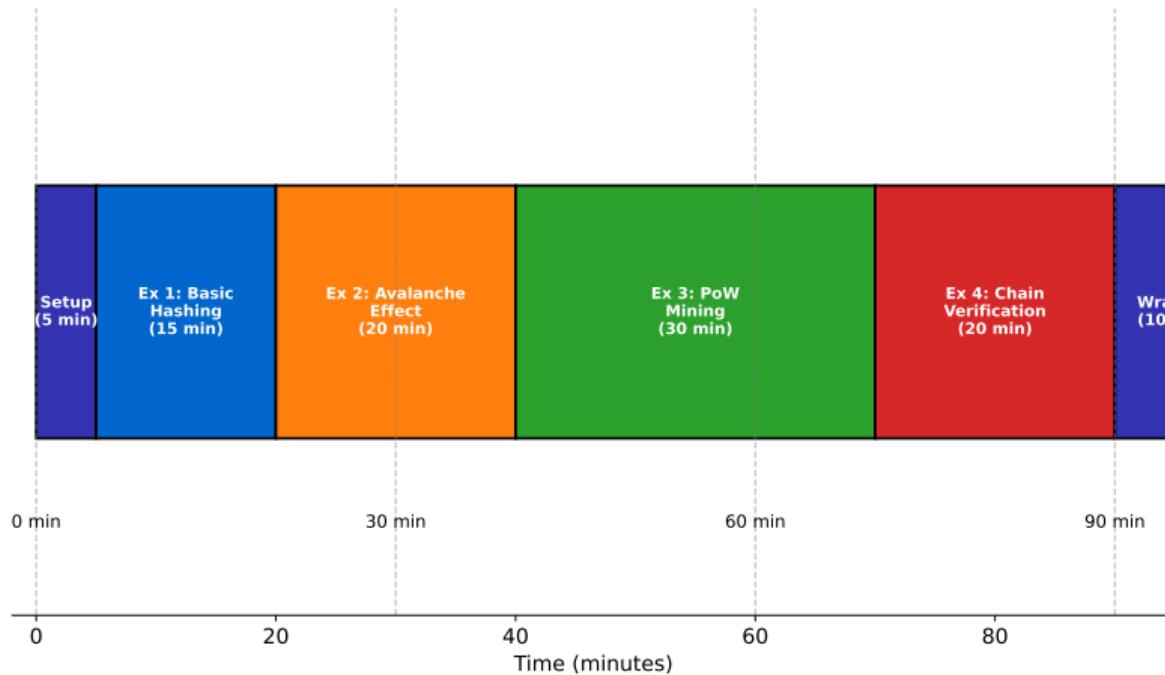
Module A: Blockchain Foundations

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

Lab Session Structure: 90 Minutes Total



Total duration: 90 minutes with 4 hands-on exercises

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

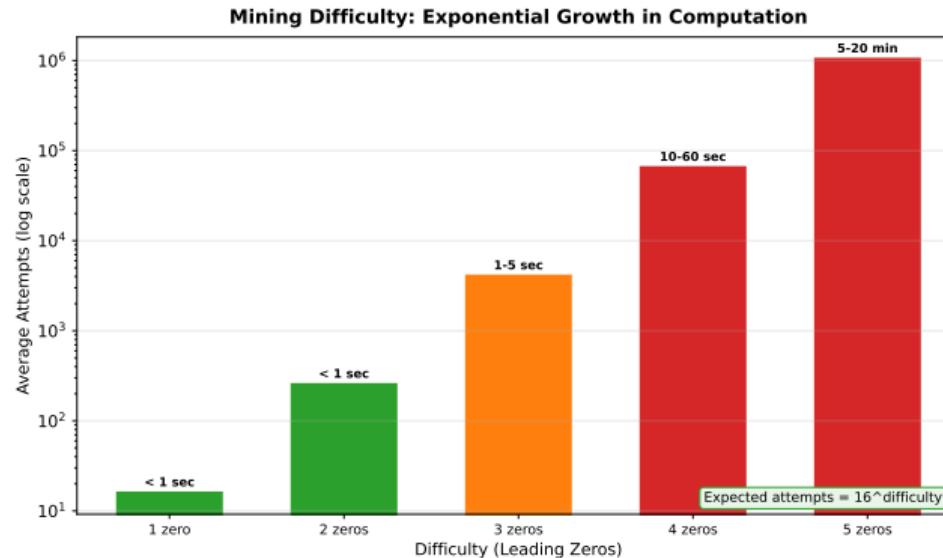
Concept Review:

- 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

Mining Difficulty: Exponential Growth



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

Hash Chain Integrity Verification

Valid Chain: All Hashes Match



Tampered Chain: Hash Mismatch Detected



Tampering breaks hash links, making modifications immediately detectable

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?