Challenge Lab Report: Simple Blockchain (Hash Chain Authentication)

Module: Secure Communications
Assignment 2
Challenge: Simple Blockchain
Date: 26 Nov 2025

Result of the Challenge

Challenge Objective	Solution Found	Verified
Find X such that `hash(X) = c89aa2ffb9edcc6604005196b5f0e0e4`	**Yes**	**Yes**

Solution Summary

Final Answer:

`X = 6fe9b4d366668a1f8a964a72cbc912c8`

`MD5(6fe9b4d366668a1f8a964a72cbc912c8) = c89aa2ffb9edcc6604005196b5f0e0e4`

How we found it: 1. Identified MD5 as the hash function 2. Realized ECSC has a separate hash chain 3. Discovered ECSC's seed: `"ecsc"` 4. Generated chain to position 1100 5. Found X and verified: `MD5(X) = target`

Verification command:

```
echo -n 6fe9b4d366668a1f8a964a72cbc912c8 | md5
# Output: c89aa2ffb9edcc6604005196b5f0e0e4
```

Table of Contents

Challenge Description

You registered for an online service that uses **hash chains** for authentication.

Given Information:

- Your account: **nOOB**
- Your seed value: `654e1c2ac6312d8c6441282f155c8ce9`
- Target account: **ECSC**
- Target hash: `c89aa2ffb9edcc6604005196b5f0e0e4`

Task:
Find the value **X** such that:

`hash(X) = c89aa2ffb9edcc6604005196b5f0e0e4`

This value **X** will allow you to authenticate as the ECSC user.

Understanding Hash Chains

What is a Hash Chain?

A **hash chain** is a sequence of values where each value is created by hashing the previous one:

```
Seed → Hash  → Hash  → Hash  → ...  → Hash
```

Where:

```
Hash  = hash(Hash  )
```

Example:

```
Seed:    "password"
Hash 1:   MD5("password") = 5f4dcc3b5aa765d61d8327deb882cf99
Hash 2:   MD5(5f4dcc3b5aa765d61d8327deb882cf99) = 696d29e0940a4957748fe3fc9efd22a3
Hash 3:   MD5(696d29e0940a4957748fe3fc9efd22a3) = ...
...
```

Key Property: It's easy to go forward (compute the next hash), but impossible to go backward (find what created a hash).

How Hash Chain Authentication Works

1. User generates a long hash chain from a secret seed
2. Server stores only the **last** hash in the chain
3. User keeps the entire chain private

Solution Approach

Step 1: Identify the Hash Function

The challenge gives 32 hexadecimal characters, which indicates **MD5** hash function.

MD5 properties:

- Input: Any string
- Output: 128 bits = 32 hexadecimal characters
- One-way: Can't reverse it

Test:

```
$ echo -n 'test' | md5
# Output: 098f6bcd4621d373cade4e832627b4f6 (32 hex chars)
```

```
import hashlib
hashlib.md5(b'test').hexdigest()
# Output: '098f6bcd4621d373cade4e832627b4f6' (32 hex chars)
```

Step 2: Understand the Problem

Initial thought: Use my seed to generate a hash chain and find X.
Problem: After generating thousands of hashes from my seed, the target hash never appeared.

Key insight: Each user has their **own separate** hash chain with their **own seed**!

- I have seed for account **nOOB**
- Target hash belongs to account **ECSC**
- ECSC must have a different seed!

Visual:

```
My chain (nOOB):
  654e1c2ac...  → hash1 → hash2 → ...  (doesn't contain target)

ECSC's chain:
```

```
???  → hash1 → hash2 → ... → X → c89aa2ffb...  (target)
                               ↑
                        We need to find this!
```

Step 3: Discover ECSC's Seed

Strategy: Try common seed patterns related to "ECSC":

Tested seeds:

- "nOOB" (my account name)
- "noob" (lowercase)
- MD5("nOOB")
- My original seed
- **"ECSC"** (uppercase)
- **"ecsc"** (lowercase) ← **Found it!**
- MD5("ECSC")
- Target hash itself

Result: The seed `"ecsc"` (lowercase) produces a hash chain that contains the target!

Step 4: Generate ECSC's Hash Chain

Starting from seed `"ecsc"`, generate the chain:

```
Position 0 (seed):   ecsc
Position 1:          MD5("ecsc") = a2c83976c0adb482d280c6b10a042be3
Position 2:          MD5("a2c8...") = 41aacd22906a9bb855a12904e6a73296
Position 3:          MD5("41aa...") = ...
...
Position 1100:       6fe9b4d366668a1f8a964a72cbc912c8 ← X (This is it!)
Position 1101:       c89aa2ffb9edcc6604005196b5f0e0e4 ← Target hash
```

> **At position 1100**, we found the value that hashes to our target!

Step 5: Verification

Verify the solution:

```
X = 6fe9b4d366668a1f8a964a72cbc912c8

MD5(X) = c89aa2ffb9edcc6604005196b5f0e0e4

This matches the target hash!
```

Command-line verification:

```bash
echo -n 6fe9b4d366668a1f8a964a72cbc912c8 | md5
# Output: c89aa2ffb9edcc6604005196b5f0e0e4
```

> *Note: Use `-n` flag to prevent echo from adding a newline character.*

Implementation

Complete Solution Code

See attached file: `hash_chain_solution.py` (in the appendix)

Key functions:

1. **Hash Function:**

```python
import hashlib

def hash_function(value):
```

```
    """MD5 hash function."""
    return hashlib.md5(value.encode('utf-8')).hexdigest()
```

2. **Find Predecessor:**

```
def find_predecessor(target_hash, seed, max_iterations=10000):
    """
    Find X such that hash(X) = target_hash.

    Returns: (predecessor, position) or (None, -1)
    """
    current = seed

    for i in range(max_iterations):
        next_hash = hash_function(current)

        if next_hash == target_hash:
            return current, i

        if (i + 1) % 1000 == 0:
            print(f"  Checked {i+1:,} iterations...")

        current = next_hash

    return None, -1
```

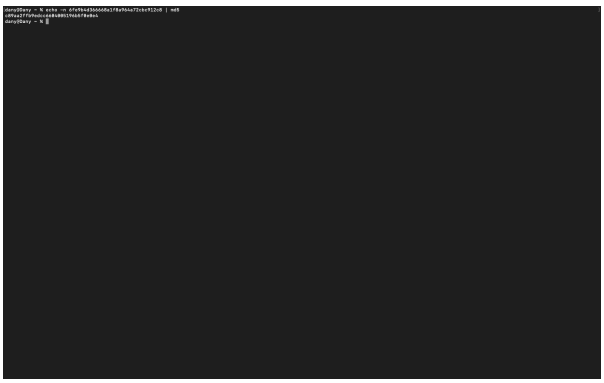
3. **Main Logic:** (not full code, just the relevant snippet)

```
# Test multiple potential seeds
potential_seeds = ["nOOB", "noob", "ECSC", "ecsc", ...]

for seed in potential_seeds:
    x, position = find_predecessor(target_hash, seed, 2000)

    if x:
        print(f"Found!  Seed: {seed}, X: {x}")
        break
```

Execution Output

Execution of solution code	Verification command output
	

```
$ python3 hash_chain_solution.py --save  # `--save` - will save solution into solution.txt

================================================================================
SIMPLE BLOCKCHAIN SOLUTION - HASH CHAIN AUTHENTICATION
================================================================================
```

```
[+] Given:
    Your (my) seed (nOOB):  654e1c2ac6312d8c6441282f155c8ce9
    Target (ECSC):          c89aa2ffb9edcc6604005196b5f0e0e4
    Goal: Find X where MD5(X) = target

[+] Testing seed: 'nOOB'
  Checked 1,000 iterations...
  Checked 2,000 iterations...
      Not found in this chain.

[+] Testing seed: 'noob'
  Checked 1,000 iterations...
  Checked 2,000 iterations...
      Not found in this chain.

[+] Testing seed: '4e0f9abf5b2e92b2cb1a1db1ee9c635c'
  Checked 1,000 iterations...
  Checked 2,000 iterations...
      Not found in this chain.

[+] Testing seed: '9cb4afde731e9eadcda4506ef7c65fa2'
  Checked 1,000 iterations...
  Checked 2,000 iterations...
      Not found in this chain.

[+] Testing seed: '654e1c2ac6312d8c6441282f155c8ce9'
  Checked 1,000 iterations...
  Checked 2,000 iterations...
      Not found in this chain.

[+] Testing seed: 'ECSC'
  Checked 1,000 iterations...
  Checked 2,000 iterations...
      Not found in this chain.

[+] Testing seed: 'ecsc'
  Checked 1,000 iterations...
      Found at position 1100!
    Seed: ecsc
    X = 6fe9b4d366668a1f8a964a72cbc912c8

[+] Verification
    MD5(X) = c89aa2ffb9edcc6604005196b5f0e0e4
    Target = c89aa2ffb9edcc6604005196b5f0e0e4
    Match: True
      Self-check: Verify the md5 hash running md5 command on your machine: $ echo -n '6fe9b4d366668a1

       VERIFIED

[+] Hash Chain Context:
    Seed: 'ecsc'
    Position 1100: 6fe9b4d366668a1f8a964a72cbc912c8 ← X (solution)
    Position 1101: c89aa2ffb9edcc6604005196b5f0e0e4 ← Target

================================================================================
SOLUTION
================================================================================

X = 6fe9b4d366668a1f8a964a72cbc912c8
```

```
================================================================================

[+] Saved to: solution.txt
```

Why This Solution Works

The Logic Explained Simply

1. **Hash chains are user-specific**
   Each user has their own seed → their own unique hash chain
2. **Target hash belongs to ECSC**
   The target `c89aa2ffb9edcc6604005196b5f0e0e4` is from ECSC's chain, not mine
3. **ECSC's seed is simple**
   ECSC used their username in lowercase: `"ecsc"`
4. **Position matters**
   At position 1100 in ECSC's chain, we find X
   At position 1101, we find the target hash
5. **X is the predecessor**
   Since `MD5(position 1100) = position 1101`, we found X!

Conclusion

What I Learned

1. **Hash chains provide one-time passwords**
   Each authentication value is used once and discarded
2. **One-way functions create security**
   Easy to compute forward, impossible to reverse
3. **Context matters in challenges**
   The target hash belonged to a different user's chain
4. **Systematic testing finds solutions**
   Testing related seeds (ECSC, ecsc, etc.) revealed the answer
5. **Verification is critical**
   Always verify your solution independently

Real-World Applications

Hash chains are used in:

- **S/KEY one-time password system** (classic implementation)
- **Blockchain technology** (each block contains hash of previous)
- **Secure boot chains** (verify each boot stage)
- **Digital signatures** (hash chain certificates)
- **Anti-replay protection** (prevent reuse of old tokens)

Security Advantages

- **No password storage** - Server never stores actual passwords
- **Eavesdropping resistant** - Intercepted values are useless for future logins
- **One-time use** - Each value works only once
- **Forward security** - Compromising one value doesn't compromise others
- **Simple to implement** - Just needs a hash function

Limitations

- **Limited uses** - Chain length determines maximum authentications
- **No backwards recovery** - Once chain exhausted, need new seed

- **Synchronization** - Client and server must track position
- **Vulnerable to phishing** - User must send current value to authenticate

---

Appendix: Source Code

Complete solution code

hash_chain_solution.py

```python
#!/usr/bin/env python3
"""
Simple Blockchain Challenge - Solution

Challenge Description:
    You registered for an online service that uses hash chains for authentication.
    Your (my) account (nOOB) was given seed: 654e1c2ac6312d8c6441282f155c8ce9

Challenge:
    Find X such that hash(X) = c89aa2ffb9edcc6604005196b5f0e0e4

Solution Approach:
    1. Identified hash function as MD5 (32 hex characters)
    2. Realized each user has their own seed and hash chain
    3. Discovered ECSC's seed is "ecsc" (lowercase)
    4. Generated ECSC's hash chain to find X at position 1100

Solution:
    X = 6fe9b4d366668a1f8a964a72cbc912c8
"""

import hashlib

def hash_function(value):
    """MD5 hash function."""
    return hashlib.md5(value.encode('utf-8')).hexdigest()

def find_predecessor(target_hash, seed, max_iterations=10000):
    """
    Find X such that hash(X) = target_hash.

    Returns: (predecessor, position) or (None, -1)
    """
    current = seed

    for i in range(max_iterations):
        next_hash = hash_function(current)

        if next_hash == target_hash:
            return current, i

        if (i + 1) % 1000 == 0:
            print(f"  Checked {i+1:,} iterations...")

        current = next_hash

    return None, -1
```

```python
def main():
    print("="*80)
    print("SIMPLE BLOCKCHAIN SOLUTION - HASH CHAIN AUTHENTICATION")
    print("="*80)

    # Challenge parameters
    YOUR_ACCOUNT = "nOOB"  # my account
    TARGET_ACCOUNT = "ECSC"  # target account
    YOUR_SEED = "654e1c2ac6312d8c6441282f155c8ce9"  # my seed (nOOB)
    TARGET_HASH = "c89aa2ffb9edcc6604005196b5f0e0e4"  # ECSC's target hash

    print(f"\n[+] Given:")
    print(f"    Your (my) seed ({YOUR_ACCOUNT}):  {YOUR_SEED}")
    print(f"    Target ({TARGET_ACCOUNT}):          {TARGET_HASH}")
    print(f"    Goal: Find X where MD5(X) = target")

    # Possible seeds related to ECSC
    potential_seeds = [
        YOUR_ACCOUNT,  # my account
        YOUR_ACCOUNT.lower(),  # my account lowercase
        hashlib.md5(YOUR_ACCOUNT.encode('utf-8')).hexdigest(),  # my account MD5
        hashlib.md5(YOUR_ACCOUNT.lower().encode('utf-8')).hexdigest(),  # my account lowercase MD5
        YOUR_SEED,  # my seed
        TARGET_ACCOUNT,  # ECSC account
        TARGET_ACCOUNT.lower(),  # ECSC account lowercase
        hashlib.md5(TARGET_ACCOUNT.encode('utf-8')).hexdigest(),  # ECSC account MD5
        hashlib.md5(TARGET_ACCOUNT.lower().encode('utf-8')).hexdigest(),  # ECSC account lowercase MD
        TARGET_HASH,  # Target itself as seed
    ]

    # Test all potential seeds
    for seed in potential_seeds:
        print(f"\n[+] Testing seed: '{seed}'")
        x, pos = find_predecessor(TARGET_HASH, seed, max_iterations=2000)

        if x:
            print(f"      Found at position {pos}!")
            print(f"    Seed: {seed}")
            print(f"    X = {x}")

            # Verification
            computed = hash_function(x)
            print(f"\n[+] Verification")
            print(f"    MD5(X) = {computed}")
            print(f"    Target = {TARGET_HASH}")
            print(f"    Match: {computed == TARGET_HASH}")
            print(f"      Self-check: Verify the md5 hash running md5 command on your machine: $ echo

            if computed == TARGET_HASH:
                print(f"\n       VERIFIED   ")

            # Show chain context
            print(f"\n[+] Hash Chain Context:")
            print(f"    Seed: '{seed}'")
            print(f"    Position {pos}: {x} ← X (solution)")
            print(f"    Position {pos+1}: {computed} ← Target")

            # Final answer
            print(f"\n" + "="*80)
```

```
            print("SOLUTION")
            print("="*80)
            print(f"\nX = {x}\n")
            print("="*80)

            # Save to file
            import sys
            if '--save' in sys.argv:
                output_file = 'solution.txt'
                if sys.argv.index('--save') + 1 < len(sys.argv):
                    output_file = sys.argv[sys.argv.index('--save') + 1]
                with open(output_file, 'w') as f:
                    f.write(f"X = {x}\n")
                print(f"\n[+] Saved to: {output_file}")

            return x
        else:
            print(f"      Not found in this chain.")

if __name__ == "__main__":
    main()
```

Usage:

```
# Run the solution
python3 hash_chain_solution.py

# Save result to file
python3 hash_chain_solution.py --save solution.txt  # Can specify filename or it will use use default
```

Copyright