Information Security

Semester Project

Group Members

Hunaina Ehsan

Ali Awais Safdar

Zainab Kashif

Back to Agenda Page

Introduction

Selected Attribute: Integrity

DEFINITION

An attribute of information that describes how data is whole, complete, and uncorrupted.

Back to Agenda Page

Introduction

Selected Attribute: Integrity

MOTIVATION

Integrity guarantees that data remains unchanged and uncorrupted, maintaining its authenticity throughout its lifecycle. By choosing integrity, we ensure the reliability and credibility of our system's data, essential for maintaining trust and security in sensitive environments.

Back to Agenda Page

Agenda

Topics Covered

- Design
- Implementation
- Demonstration
- Demonstration of Attack

Start

Design of Security Mechanism

Back to Agenda Page

Design of Security Mechanism

Custom Hashing Algorithm (SIH)

Secure Integrity Hash (SIH) class implements a custom hashing algorithm

Digital Signatures

The project utilizes RSA digital signatures to verify the authenticity and integrity of data.

Blockchain Data Structure

The blockchain data structure is inherently designed to maintain data integrity through its immutability and decentralization.

Back to Agenda Page

Design of Security Mechanism

Transaction Handling

Transactions are securely recorded in the blockchain, ensuring the integrity of data being added to the chain.

File Persistence and Validation

The project includes functionality to save the blockchain to a file after each block addition. This ensures that the integrity of the blockchain is maintained across system

Back to Agenda Page

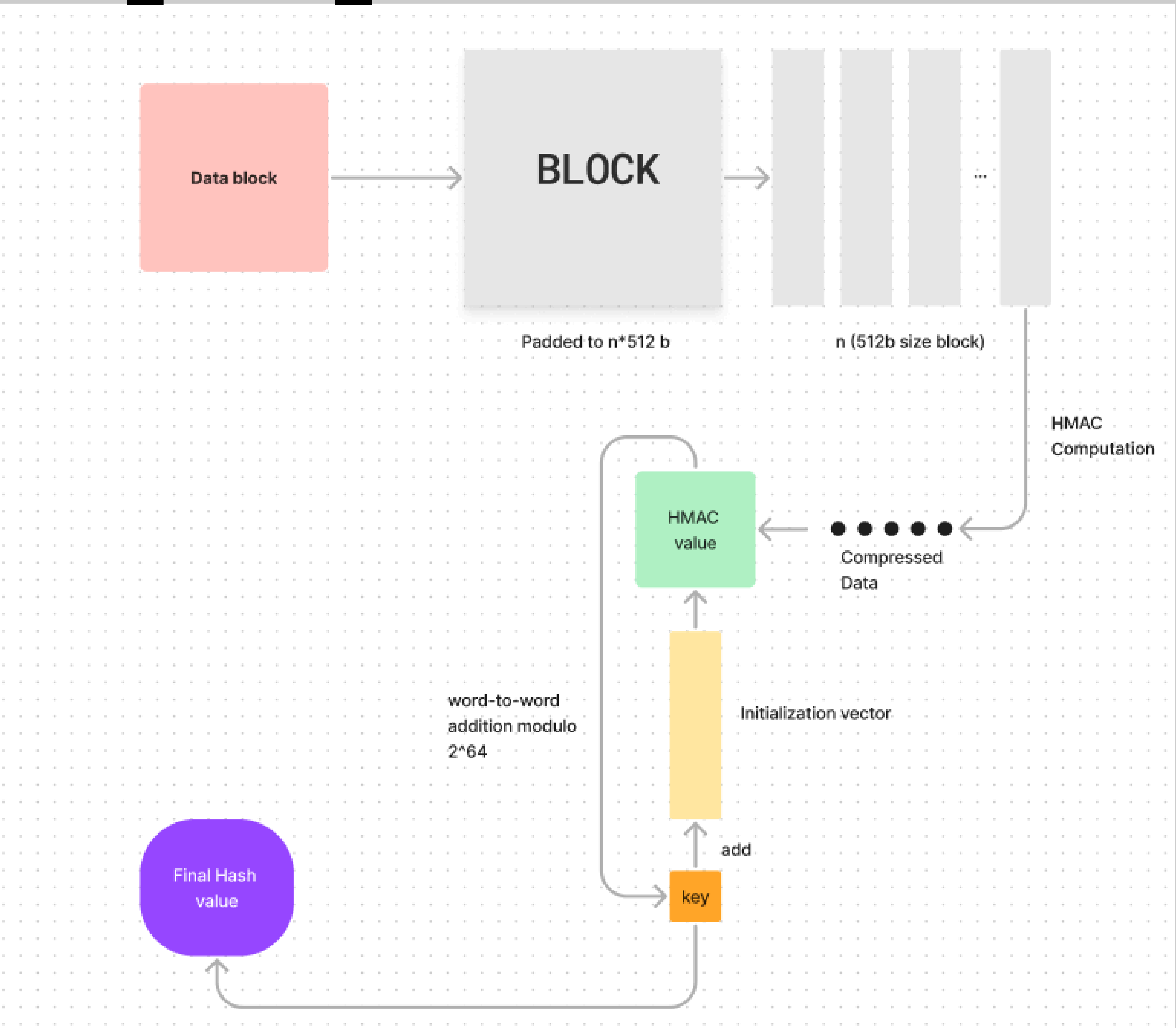
Implementation of Security Mechanism

Back to Agenda Page

Simple Integrity Hash

- The SIH class implements the calculate_hash() method to generate a hash of the data.
- calculate_has() has generate_hmac() and compress_data() functions defined within the method.
- Initially it defines IV as 32 bits of zeroes and key as symmetric_key (unique to every transaction)
- The original message is padded so it can be divided in 512 bit blocks.
- Afterwards it iterates through each segment, extract the current segment, compresses it and generate HMAC code of the compressed segment.
- Then HMAC code is added to the IV via word to word addition modulo 2^64.
- Then this appended value is set as key for the next segment.
- On the final segment, the key is returned as Hash Value.

Simple Integrity Hash



Hash Properties of SIH

The Hash algorithm implemented by us holds the following property:

- Variable input size
- Fixed output size
- Pre-image Resistance
- Avalanche Effect
- Large Hamming Distance
- Second Pre-image Resistance
- Collision Resistance

Hash Properties of SIH

PRE-IMAGE RESISTANCE

We tested the property 10,000 times.

```
print("3. Pre-image Resistance:")
print("It is computationally infeasible to find the original data from the hash value. \n")
target_hash = hash1
for _ in range(10000):
    random_data = ''.join(random.choices('abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789', k=16))
    sih.data = random_data
    random_hash = sih.calculate_hash(symmetric_key)
    if random_hash == target_hash:
        print("Pre-image Resistance property compromised as hash values are same")
        break
print("Pre-image Resistance property is maintained as hash values are different")
```

```
3. Pre-image Resistance:
It is computationally infeasible to find the original data from the hash value.

Pre-image Resistance property is maintained as hash values are different
```

Hash Properties of SIH

SECOND PRE-IMAGE RESISTANCE

We tested the property 10,000 times.

```
print("4. Second Pre-image Resistance:")
print("It is computationally infeasible to find another data with the same hash value as the original data. \n")

sih.data = data1
original_hash = sih.calculate_hash(symmetric_key)
for _ in range(10000):
    random_data = ''.join(random.choices('abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789', k=16))
    if random_data == data1:
        continue
    sih.data = random_data
    random_hash = sih.calculate_hash(symmetric_key)
    if random_hash == original_hash:
        print("Second Pre-image Resistance property compromised as hash values are same")
        break
print("Second Pre-image Resistance property is maintained as hash values are different")
```

```
4. Second Pre-image Resistance:
It is computationally infeasible to find another data with the same hash value as the original data.

Second Pre-image Resistance property is maintained as hash values are different
```

Hash Properties of SIH

COLLISION RESISTANCE

We tested the property 10,000 times.

```
print("5. Collision Resistance:")
print("It is computationally infeasible to find two different data with the same hash value. \n")

seen_hashes = {}
for _ in range(10000):
    random_data = ''.join(random.choices('abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789', k=16))
    sih.data = random_data
    random_hash = sih.calculate_hash(symmetric_key)
    hash_hex = random_hash
    if hash_hex in seen_hashes:
        print(seen_hashes[hash_hex], random_data, "have the same hash value hence Collision Resistance property is compromised")
        break
    seen_hashes[hash_hex] = random_data
print("Collision Resistance property is maintained as hash values are different")
```

```
5. Collision Resistance:
It is computationally infeasible to find two different data with the same hash value.

Collision Resistance property is maintained as hash values are different
```

Hash Properties of SIH

AVALANCHE EFFECT

BOTH HASHES ARE COMPLETELY DIFFERENT

```python
print("1. Avalanche Effect: \n")
data1 = "Hello world"
data2 = "Hello World"
sih = SIH(data1)
hash1 = sih.calculate_hash(symmetric_key)
sih.data = data2
hash2 = sih.calculate_hash(symmetric_key)

if hash1 != hash2:
    print("Hash function has high diffusion. Changing one alphabet in the input data changes the hash value significantly.")
    print("Hash1:", hash1)
    print("Hash2:", hash2, "\n")
```

```
1. Avalanche Effect:

Hash function has high diffusion. Changing one alphabet in the input data changes the hash value significantly.
Hash1: d5d8028f1e600f7f0d8a628a3a05e7f617d1b0fefb96a03a4f180c079413d5cc
Hash2: ec9a581e8fe38153a5cf6e5f7f3d0ea39a7f9556bf2876146ed133c762b1f2fd
```

Hash Properties of SIH

LARGE HAMMING DISTANCE

EVEN SMALL CHANGE IN A SENTENCE YIELD HASHES HAVING LARGE HAMMING DISTANCE

```
print("2. Hamming Distance between 2 Hashes having only one alphabet changed (The Larger the Better).")
print(sum(bin(ord(x) ^ ord(y)).count('1') for x, y in zip(hash1, hash2)), "\n")
```

```
2. Hamming Distance between 2 Hashes having only one alphabet changed (The Larger the Better).
174
```

Digital Signature

- The generate_signature() method in SIH generates a digital signature using RSA encryption with a private key.
- The verify_signature() function in the utility module verifies the signature using the corresponding public key.


```
Digital Signature: b"[r\xbb,\x18\x9d\xf3Jd\x81\xda\x18~\xec\xea]$j!\x1b`\xc5)\x8aW\x9dB\x9a?\xa5^;\x9eq\xea\xa6\xd8\xc5\xae\xbde\xcd\xbf\xb5&\x81\xf8b\xcf\xd3\xc8\
x98I@\x92\xddJ0(\x06\x08\x84!K\x1fG7>\xc5L\x0b)\xbf\xd4d\x9f\xb8t\xc6\xa9m\xfbeU@\xdc\xd8\x8d\x87f\xdb\x1a\x10#G\x12\xf6\x19ZU`\x06\xc7\rr79\x8bQA\x8e\x06\xe0\xbb\
x7fqO\x801/\xe1\xe1\xa2\x17]\xbd\x02\xca\x9d\x03\x87\xd6\xa5\x90\xc0`\x14\x7fXX\xf4q\xb2\xf05\x10\xa1`\xae\xb8\xc47^\x18\r\x93'\x91\xe9P\xb9\xefv\xb0S\x8cobz\x0e\x
a9\x18\xb6\xf1\x15\xb2\xc6\x1b\xdeB\x7f\x99~\x81.\xa8\xd5\x00\xa6\xef\x11\x06\xb8\x8fS\xc7\x9e\xad\xab\xd6\x00\xdc$\xcd4'\x04\xc6\x12\xcc\x98\x99,\xbeeV9}B\xd8\xa2
e\x90m\x0e\x85\xea3\x07#b\xcb\xeeBl\x06\x94\xbe\xb3\xce8\xc4\xdf\xaa\x1c\xca\xc70\xe1\xf2/\xa3\nS\xcc\xf1"
```

Block Chain Data Structure

- The Blockchain class utilizes a list of blocks, where each block contains transactions and a hash of the previous block.
- The new_block() method creates a new block, ensuring integrity by hashing the previous block's hash along with the current block's data.

{} blockchain.json > ...


```
[{"index": 1, "transactions": [{"data": "The data at first index, would be replaced by this data and the blockchain would be updated re
```

Transaction Handling

- The new_transaction() method adds a new transaction to the current block, ensuring the integrity of data being added.
- Each transaction includes the original data, its hash, encrypted data, and symmetric key, ensuring that the data remains intact.

```python
def new_transaction(self, data):
    """
    Add a new transaction to the blockchain.
    """
    self.current_transactions.append(data)
```

File Persistence and Validation

- The Blockchain class includes methods load_chain( ) and save_chain( ) to load and save the blockchain data from/to a file.
- Upon loading, the integrity of the loaded data is verified to ensure its validity.

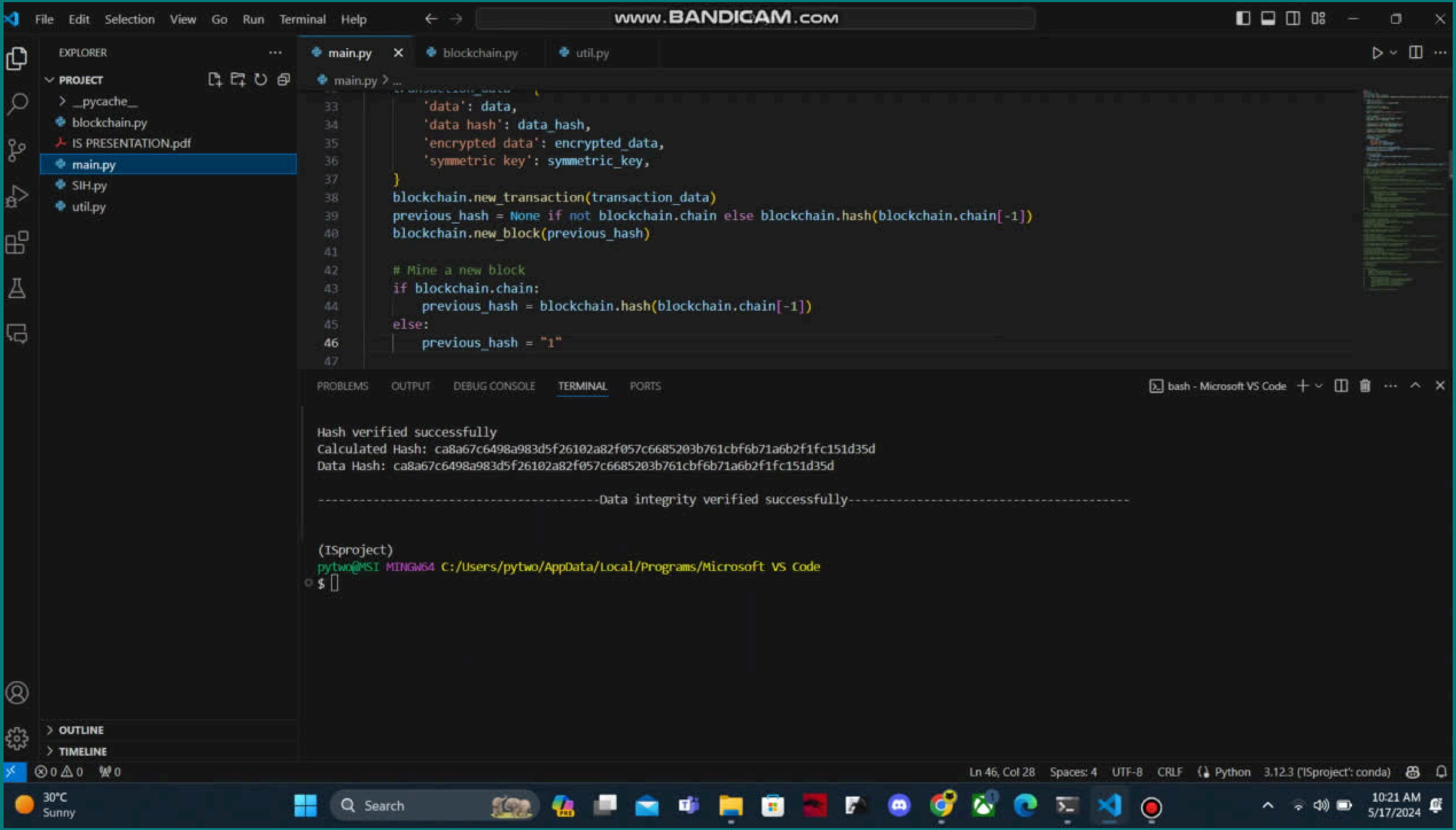
```python
def load_chain(self):
    try:
        with open("blockchain.json", "r") as f:
            data = f.read()
            if data:  # Check if the file is not empty
                self.chain = json.loads(data, object_hook=self.hex_to_bytes)
            else:
                self.chain = []
    except FileNotFoundError:
        # If file not found, start with an empty chain
        self.chain = []
    except json.decoder.JSONDecodeError:
        print("Error loading blockchain. File contains invalid JSON data.")
        self.chain = []
```

```python
def save_chain(self):
    def convert_to_serializable(obj):
        if isinstance(obj, bytes):
            return obj.hex()  # Convert bytes to hexadecimal string
        return obj

    with open("blockchain.json", "w") as f:
        json.dump(self.chain, f, default=convert_to_serializable)
```

Demonstration of Security Mechanism

Back to Agenda Page



Link

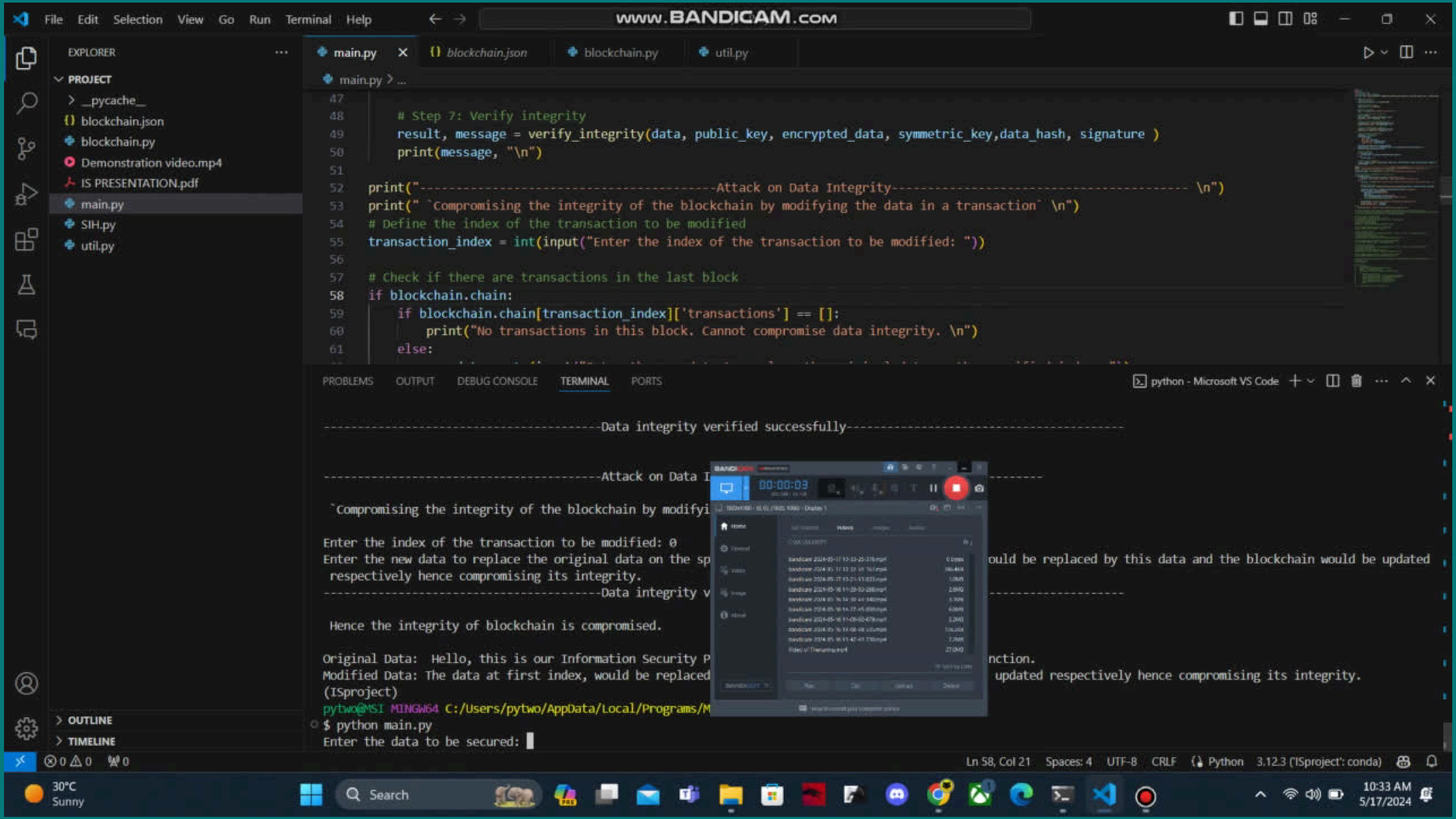
Back to Agenda Page

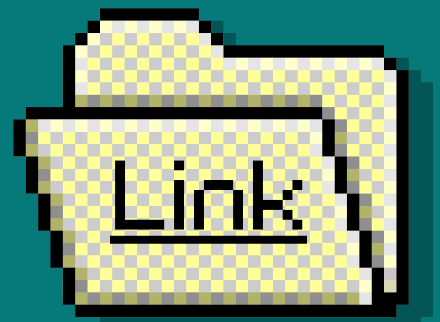
Demonstration of Attack

Back to Agenda Page

Compromising the integrity of the blockchain by modifying the data in a transaction

- This attack assumes that the attacker has gotten access to the blockchain and is aware of the hashing algorithm used to hash blocks.
- The function compromise_blockchain_data_integrity() modifies the selected transaction by replacing its original data with the new data provided. It updates the data field and calculate the hash of the block to replace 'previous hash' of the subsequent block and updates the 'previous hash' field of all the blocks.





Back to Agenda Page

Compromising the integrity by Man-in-the-Middle (MITM) attack

- It assumes the communication is not encrypted and signed.
- The attacker positions themselves between Alice and Bob's communication channel, intercepting messages without their knowledge.
- The attacker knows the hash function and the symmetric key, possibly obtained through a brute force attack.
- Both parties receive messages that appear to come from each other.
- However, the content of the messages has been altered by the attacker, while the hash values match the modified content. This is provided by the method intercept_message().

```
------------------------------------------Attack on Data Integrity-------------------------------------------

`Compromising the integrity of messages by deploying (Man in The Middle Attack)`

[MITM] Intercepting message from Alice to Bob
[MITM] Original message: Hello Bob, this is Alice.
[MITM] Modified message: Modified message from Alice to Bob
[MITM] Modified hash: 9bcd7dce406ed1c382ed5fdf486f4f5cf4c97b90cfc65e30634b6537bca75aa1
[MITM] Intercepting message from Bob to Alice
[MITM] Original message: Hi Alice, I received your message.
[MITM] Modified message: Modified message from Bob to Alice
[MITM] Modified hash: 74cdbf3767fb15047496281d945de5fc52721ad367e7f89b1afc4bdff42eb4f9
[MITM] Intercepting message from Alice to Bob
[MITM] Original message: Hi Bob, I received your response.
```

Compromising the integrity by generating a collision (Birthday Attack)

This attack demonstrates the vulnerability of hash functions to produce collisions, highlighting the importance of using hash functions with strong collision resistance properties. Additionally, it underscores the significance of using sufficiently long hash values to minimize the likelihood of collisions.

```python
print("`Compromising the integrity of hash function by generating a collision (Birthday Attack)` \n")
hash_dict = {}
num_attempts = 0

while True:
    num_attempts += 1
    random_string = generate_random_string(10)
    fake_sih = SIH(random_string)
    hash_value = fake_sih.calculate_hash(symmetric_key)

    if hash_value in hash_dict:
        print(f"Collision found after {num_attempts} attempts!")
        print(f"Original String 1: {hash_dict[hash_value]}")
        print(f"Hash of String 1: {hash_value}")
        print(f"Original String 2: {random_string}")
        print(f"Hash of String 2: {hash_value}")
        break

    hash_dict[hash_value] = random_string
```

Back to Agenda Page

Conclusion and Recommendation

Back to Agenda Page

Conclusion

In conclusion, the implemented security mechanisms have provided significant protection against various integrity-based attacks, such as tampering with data and compromising hash functions. The use of digital signatures, encryption, and a blockchain structure has enhanced the integrity of the system, ensuring that data remains unchanged and authentic throughout its lifecycle. However, certain vulnerabilities were identified, particularly in the form of Man-in-the-Middle (MITM) attacks and hash function collisions, which could potentially compromise data integrity as well as blockchain tampering.

Recommendation

<u>Stronger Hash Functions:</u>

Utilize hash functions with higher collision resistance

<u>Salting and Iterative Hashing:</u>

Incorporate salting and iterative hashing techniques to further strengthen the integrity of the hash function.

<u>Blockchain Consensus Mechanisms:</u>

Employ more robust consensus mechanisms in the blockchain, such as Proof of Stake (PoS) or Practical Byzantine Fault Tolerance (PBFT), to enhance the security and integrity

Division of Work

Hunaina Ehsan

Ali Awais Safdar

Zainab Kashif

Security Analysis and Testing

Security Mechanisms

Blockchain Integration

Back to Agenda Page

References

- https://stuvel.eu/python-rsa-doc/usage.html#signing-and-verification
- https://www.oreilly.com/library/view/enterprise-wide-security-architecture/0738402087/chapter-34.html
- Nakamoto, Satoshi. "Bitcoin: A Peer-to-Peer Electronic Cash System." Available online: https://bitcoin.org/bitcoin.pdf
- Our Course Lectures.

GITHUB

https://github.com/Ali-Awais-Safdar/IntegritySecurityAndAttack-InformationSecurityProject

Back to Agenda Page


Thank you!
^_^