Simple Blockchain Ledger
Coded in Python

Sprawozdanie

Prowadzący:	*Autorzy:*		
mgr inż. Jakub Hamerliński	**Julia Chabora 145218**	*Grupa dziekańska:*	***L1***

1. **You are given a set of 10 transactions between different parties. Each transaction includes a sender, recipient, and amount (e.g., Alice sends 5 coins to Bob). Create a table to represent these transactions.**

```python
# transactions
transactions = [
      {'transaction_number': 1, 'sender': 'Alice', 'recipient': 'Bob', 'amount': 5},
      {'transaction_number': 2, 'sender': 'Bob', 'recipient': 'Charlie', 'amount': 10},
      {'transaction_number': 3, 'sender': 'Charlie', 'recipient': 'Alice', 'amount': 7},
      {'transaction_number': 4, 'sender': 'David', 'recipient': 'Eve', 'amount': 3},
      {'transaction_number': 5, 'sender': 'Eve', 'recipient': 'Frank', 'amount': 8},
      {'transaction_number': 6, 'sender': 'Frank', 'recipient': 'Alice', 'amount': 6},
      {'transaction_number': 7, 'sender': 'Alice', 'recipient': 'David', 'amount': 2},
      {'transaction_number': 8, 'sender': 'Bob', 'recipient': 'Eve', 'amount': 4},
      {'transaction_number': 9, 'sender': 'Eve', 'recipient': 'Alice', 'amount': 9},
      {'transaction_number': 10, 'sender': 'Frank', 'recipient': 'Bob', 'amount': 3}
]
```

Code version

```
1       Alice Bob 5
2       Bob Charlie 10
3       Charlie Alice 7
4       David Eve 3
5       Eve Frank 8
6       Frank Alice 6
7       Alice David 2
8       Bob Eve 4
9       Eve Alice 9
10      Frank Bob 3
```

Printed out version

2. **In pairs, choose one transaction from the table and encrypt it using a simple encryption method (e.g., Caesar cipher, a basic substitution cipher). Exchange the encrypted transaction with your partner, and attempt to decrypt each other's transactions.**

```
#encrypt caesar
def encrypt(text, shift):
      encrypted_text = ""
      for char in text:
      if char.isalpha():
            ascii_offset = 0
            if char.isupper():
                  ascii_offset = ord('A')
            else:
                  ascii_offset = ord('a')
            encrypted_char = chr((ord(char) - ascii_offset
                                    + shift) % 26 + ascii_offset)
            encrypted_text += encrypted_char
      elif char.isdigit():
            encrypted_digit = str((int(char) + shift) % 10)
            encrypted_text += encrypted_digit
      else:
            encrypted_text += char
      return encrypted_text
```

Encryption function

```
#decrypt caesar
def decrypt(encrypted_text, shift):
      decrypted_text = ""
      for char in encrypted_text:
      if char.isalpha():
            ascii_offset = 0
            if char.isupper():
                  ascii_offset = ord('A')
            else:
                  ascii_offset = ord('a')
            decrypted_char = chr((ord(char) - ascii_offset
                                    - shift) % 26 + ascii_offset)
            decrypted_text += decrypted_char
      elif char.isdigit():
            decrypted_digit = str((int(char) - shift) % 10)
            decrypted_text += decrypted_digit
      else:
            decrypted_text += char
```

```
    return decrypted_text
```

Decryption function

```
Encrypted:
Sender:  Dolfh
Recipient:  Ere
Amount:  8


Decrypted:
Sender:  Alice
Recipient:  Bob
Amount:  5
```

e.g. of usage

3. **Calculate a simple hash for each transaction in the table using a basic hash function, such as adding the ASCII values of the characters and taking the remainder when divided by a prime number. Add a column to the table for the hash values.**

```
#hash
def calculate_hash(text):
    prime_number = 59
    hash_value = sum(ord(char) for char in text) % prime_number
    return hash_value

for transaction in transactions:
    transaction['hash'] = calculate_hash(transaction['sender'] +
transaction['recipient'] + str(transaction['amount']))
```

Hash function

4. **Discuss the concept of digital signatures and how they could be applied to the transactions. Choose one transaction and create a simple digital signature for it using a basic method, such as appending a unique identifier known only to the sender (e.g., Alice#123). Share the transaction with its "digital signature" with your partner and discuss how this method helps ensure the authenticity and integrity of the transaction.**

   Verifying digital signature inside of the transaction ensures anyone that the transaction is valid and wasn't in any shape or form tampered. If anyone tried to modify the data, the signature would change. So it protects the integrity of the data. It

also ensures, if the signature is indeed correct, that the transaction was sent by the real sender.

```
# signature
transaction = transactions[0]
unique_identifier = "Alice#123"
transaction['signature'] = unique_identifier
```

Signature in code

5. **Now, create a simple blockchain ledger by organizing the transactions into "blocks." Group the transactions into sets of three (leaving one transaction ungrouped). For each group, calculate a combined hash by concatenating the individual transaction hashes and applying the basic hash function again. This combined hash will represent the "block" hash.**

```
for transaction in transactions:
      current_block['transactions'].append(transaction)
      if len(current_block['transactions']) == 3:
      blocks.append(current_block)
      current_block = {'transactions': []}

if current_block:
      blocks.append(current_block)

for block in blocks:
      hash = ''
      for transaction in block['transactions']:
          hash += str(transaction['hash'])
      block_hash = calculate_hash(hash)
      block['block_hash'] = block_hash
```

Creation of blocks

```
Block Hash: 35
Transactions:
1      Alice Bob 5 39
2      Bob Charlie 10 6
3      Charlie Alice 7 49


Block Hash: 21
Transactions:
4      David Eve 3 1
5      Eve Frank 8 16
```

```
6       Frank Alice 6 27


Block Hash: 15
Transactions:
7       Alice David 2 13
8       Bob Eve 4 25
9       Eve Alice 9 56


Block Hash: 49
Transactions:
10      Frank Bob 3 57
```

Created blocks with hashes

6. **Link the blocks together by including the previous block's hash in the current block's hash calculation, creating a "chain" of blocks. For the first block, use a predefined hash value (e.g., "0000").**

```
for transaction in transactions:
      current_block['transactions'].append(transaction)
      if len(current_block['transactions']) == 3:
      hash = ''
      for transaction in current_block['transactions']:
            hash += str(transaction['hash'])
      block_hash = calculate_hash(current_block['previous_hash'] + hash)
      current_block['block_hash'] = block_hash
      blocks.append(current_block)
      current_block = {'transactions': [], 'previous_hash':
str(block_hash)}

if current_block['transactions']:
      hash = ''
      for transaction in current_block['transactions']:
           hash += str(transaction['hash'])
      block_hash = calculate_hash(current_block['previous_hash'] + hash)
      current_block['block_hash'] = block_hash
      blocks.append(current_block)
```

Creation of blocks

```
Block Hash: 50
Previous Hash: 0000
Transactions:
```

```
1       Alice Bob 5 39
2       Bob Charlie 10 6
3       Charlie Alice 7 49


Block Hash: 4
Previous Hash: 50
Transactions:
4       David Eve 3 1
5       Eve Frank 8 16
6       Frank Alice 6 27


Block Hash: 8
Previous Hash: 4
Transactions:
7       Alice David 2 13
8       Bob Eve 4 25
9       Eve Alice 9 56


Block Hash: 46
Previous Hash: 8
Transactions:
10      Frank Bob 3 57
```

Created blocks with hashes

7. **Discuss with your partner how this simple blockchain ledger provides transparency, immutability, and security for the transactions. Consider the steps an attacker would need to take to tamper with a transaction and how the blockchain structure would make it difficult to do so.**

   Transparency - blockchain stores the full history of transactions, which are linked to each other. Every person that can access the ledger can also verify if the transactions are correct and have visibility of every action that happens inside of the blockchain.

   Immutability - each block contains a hash value that was created based on previous blocks. If any block would be tampered with then it would be clearly visible, because hash values would become incorrect. Any change to one block would require change to all other blocks, which becomes very inefficient in a large number of transactions, so the blockchain is mostly resistant to any form of modifying.

Security - the security is created based on hash functions, which protect the blockchain from unwanted tampering, but also by other cryptographic approaches, such as encrypting, which additionally secures the data stored inside of the ledger.