lab2 report

PB21051012 刘祥辉

Merkle树部分

func NewMerkleTree(data [][]byte) *MerkleTree //生成Merkle树

```
// NewMerkleTree creates a new Merkle tree from a sequence of data
func NewMerkleTree(data [][]byte) *MerkleTree {
    var nodes []MerkleNode

    if len(data)%2 != 0 {
        data = append(data, data[len(data)-1])
    }

    for _, datum := range data {
        node := NewMerkleNode(nil, nil, datum)
        nodes = append(nodes, *node)
    }

    height := int(math.Ceil(math.Log2(float64(len(data)))))

    for i := 0; i < height ; i++ {
        var newLevel []MerkleNode
        for j := 0; j < len(nodes); j += 2 {
            node := NewMerkleNode(&nodes[j], &nodes[j+1], nil)
            newLevel = append(newLevel, *node)
        }
        if len(newLevel)%2 != 0 {
            newLevel = append(newLevel, newLevel[len(newLevel)-1])
        }
        nodes = newLevel
    }

    return &MerkleTree{RootNode: &nodes[0], Leaf: data}
}
```

1. 创建叶子节点，每个节点包含一个数据的哈希值。
2. 计算Merkle树的高度。
3. 使用循环构建树的层级结构，将相邻的节点组合成父节点。
4. 返回根节点指针和叶子节点的数据。

func NewMerkleNode(left, right *MerkleNode, data []byte) *MerkleNode // 生成Merkle树节点

```
func NewMerkleNode(left, right *MerkleNode, data []byte) *MerkleNode {
    node := new(MerkleNode)

    if left == nil && right == nil {
        hash := sha256.Sum256(data)
        node.Data = hash[:]
    } else {
        prevHashes := append(left.Data, right.Data...)
        hash := sha256.Sum256(prevHashes)
        node.Data = hash[:]
    }
    node.Left = left
    node.Right = right

    return node
}
```

创建Merkle树的节点，根据给定的数据和子节点，计算节点的哈希值，并返回一个新的节点指针。

func (t *MerkleTree) SPVproof(index int) ([][]byte, error) //提供SPV path

```
func (t *MerkleTree) SPVproof(index int) ([][]byte, error) {
    leafCount := len(t.Leaf)
    if index >= leafCount {
        return nil, fmt.Errorf("no such leaf")
    }

    h := 0
    cnt := leafCount
    for cnt > 1 {
        cnt = cnt/2 + cnt%2
        h++
    }
    var path [][]byte
    node := t.RootNode
    for i := h; i > 0; i-- {
        signal := 1 << (i - 1)

        if index&signal == 0 {
            path = append(path, node.Right.Data)
            node = node.Left
        } else {
            path = append(path, node.Left.Data)
            node = node.Right
        }
    }
    return path, nil
}
```

这个方法根据叶子节点的索引生成SPV证明，返回一个证明路径。证明路径是一个二维字节切片，包含了从叶子节点到根节点的路径上的所有节点的哈希值。

func (t *MerkleTree) VerifyProof(index int, path [][]byte) (bool, error) //验证SPV路径

```
func (t *MerkleTree) VerifyProof(index int, path [][]byte) (bool, error) {
    if index >= len(t.Leaf) {
        return false, fmt.Errorf("no Such Leaf")
    }
    data := sha256.Sum256(t.Leaf[index])
    signal := 1

    for i := len(path) - 1; i >= 0; i-- {

        if index&signal != 0 {
            tmp := append(path[i], data[:]...)
            data = sha256.Sum256(tmp)
        } else {
            tmp := append(data[:], path[i]...)
            data = sha256.Sum256(tmp)
        }
        signal = signal << 1
    }

    return bytes.Equal(data[:], t.RootNode.Data), nil
}
```

Transaction部分

func (t *Transaction) IsCoinBase() bool //coinbase交易判断

```
func (t *Transaction) IsCoinBase() bool {

    return len(t.Vin[0].Txid) == 0 &&  t.Vin[0].Vout == -1 && len(t.Vin) == 1
}
```

Wallet部分

func (w *Wallet) GetAddress() []byte //获取公钥对应的地址

```
func checksum(payload []byte) []byte {
    inter := sha256.Sum256(payload)
    res := sha256.Sum256(inter[:])
    return res[:checkSumlen]
}

func (w *Wallet) GetAddress() []byte {
    pubKeyHash := HashPublicKey(w.PublicKey)

    // 添加版本字节 (0x00 for mainnet, 0x6f for testnet)
    versionedPayload := append([]byte{version}, pubKeyHash...)

    // 计算校验和 (双 SHA-256 哈希的前四个字节)
    checksum := checksum(versionedPayload)

    // 拼接版本字节、公钥哈希和校验和
    fullPayload := append(versionedPayload, checksum...)

    // 进行 Base58 编码
    address := base58.Encode(fullPayload)

    return []byte(address)
}
```

TXOutput部分

func (out *TXOutput) Lock(address []byte)   //设置锁定脚本PubkeyHash部分

```go
func (out *TXOutput) Lock(address []byte) {
    pubkeyHash := base58.Decode(string(address))
    pubkeyHash = pubkeyHash[1 : len(pubkeyHash)-4]
    out.PubKeyHash = pubkeyHash
}
```