区块链第一次实验报告

马振鹏 PB20111634

实验内容

完成签名和验签对应的函数部分

实验过程

Sign函数

根据给定的实验步骤逐步实现

1. 我们已知z和满足eG=P的e。
2. 随机选取k。
3. 计算R=kG,及其x轴坐标r。
4. 计算 s=(z+re)/k。
5. (r,s) 即为签名结果。

```
func (ecc *MyECC) Sign(msg []byte, secKey *big.Int) (*Signature, error) {
    // 将签名内容进行哈希
    m := crypto.Keccak256(msg)
    z := new(big.Int).SetBytes(m[:])
    // 选取随机数k,并确保符合要求
    k, err := newRand()
    if err != nil {
        return nil, err
    }
    if err := checkBigIntSize(k); err != nil {
        return nil, fmt.Errorf("k error: %s", err)
    }
    // R=kG,r为R的x轴坐标
    r := Multi(G, k).X
    r.Mod(r, N)
    if r.Sign() == 0 {
        return nil, fmt.Errorf("r is zero")
    }
    // re
    s := new(big.Int).Mul(secKey, r)
    // re+z
    s.Add(s, z)
    // 1/k
    k_inv := Inv(k, N)
    // s=(z+re)/k
    s.Mul(s, k_inv)
    s.Mod(s, N)
    if s.Sign() == 0 {
```

```
        return nil, fmt.Errorf("s is zero")
    }
    return &Signature{r: r, s: s}, nil
}
```

其中需要注意，在返回结果时需指定对应的r与s，否则有可能返回顺序颠倒，在后续验签过程中产生错误。

VerifySignature函数

同样根据给定的实验步骤逐步进行实现

1. 接收签名者提供的(r,s)作为签名，z是被签名的内容的哈希值。P是签名者的公钥（或者公开的点）。
2. 计算 u=z/s 和 v=r/s。
3. 计算 uG + vP = R。
4. 如果R的x轴坐标等于r，则签名是有效的

```
func (ecc *MyECC) VerifySignature(msg []byte, signature *Signature, pubkey *Point)
bool {
    // 将签名内容进行哈希
    m := crypto.Keccak256(msg)
    z := new(big.Int).SetBytes(m[:])
    // 1/s
    s_inv := Inv(signature.s, N)
    // u=z/s
    u := new(big.Int).Mul(z, s_inv)
    u.Mod(u, N)
    // v=r/s
    v := new(big.Int).Mul(signature.r, s_inv)
    v.Mod(v, N)
    // uG+vP=R的x轴坐标
    x := Add(Multi(G, u), Multi(pubkey, v)).X
    x.Mod(x, N)
    // 返回结果
    return x.Cmp(signature.r) == 0
}
```

实验结果

于在线检测平台提交检测，提示“步骤一检测通过”，说明代码运行结果正确。

实验总结

学习了新的编程语言--go语言，对于椭圆曲线算法的实现有了了解，感受了区块链加密算法的巧妙性。