Report on the Application of this Deduce Technique in Ethereum with ECDSA

姜舜天

2023 年 7 月 18 日

目录

1 介绍 **1**

1.1 以太坊 1

1.2 ECDSA 算法 2

2 原理 **2**

3 具体实现 **3**

3.1 ECDSA Sign 3

3.2 ECDSA Recovery 3

4 实现效果 **4**

4.1 测试代码 4

4.2 执行结果 5

1 介绍

1.1 以太坊

以太坊（Ethereum）是一个具有智能合约功能的去中心化区块链。以太币（ETH）是平台的原生加密货币。在加密货币中，以太币的市值仅次于比特币。它是开源软件。

以太坊是由程序员 Vitalik Buterin 于 2013 年构想出来的。以太坊允许任何人在其上部署永久且不可变的去中心化应用程序，用户可以与之交互。去中心化金融（DeFi）应用提供金融

工具它们不直接依赖经纪商、交易所或银行等金融中介机构。这有助于以持有的加密货币进行借贷或将其借出以获取利息。以太坊还允许用户创建和交换不可替代代币（NFT），这些代币可以与图像等独特的数字资产绑定。此外，许多其他加密货币在以太坊区块链之上利用 ERC-20 代币标准，并利用该平台进行初始代币发行。

2022 年 9 月 15 日，以太坊在称为"合并"的升级过程中将其共识机制从工作量证明 (PoW) 过渡到权益证明 (PoS)。这使得以太坊的能源使用量减少了 99%。

1.2 ECDSA 算法

椭圆曲线数字签名算法 (Elliptic Curve Digital Signature Algorithm,ECDSA) 是一种基于椭圆曲线密码学的公钥加密算法,1985 年，Koblitz 和 Miller 把数字签名算法移植到椭圆曲线上，椭圆曲线数字签名算法由此诞生。在密码学中，椭圆曲线数字签名算法提供了使用椭圆曲线密码学的数字签名算法 (DSA) 的变体。

与一般的椭圆曲线加密一样，ECDSA 所需的私钥的位大小大约是安全级别大小（以位为单位）的两倍。例如，在 80 位的安全级别上，这意味着攻击者最多需要大约 2^{80} 次查找私钥的操作 - ECDSA 私钥的大小为 160 位。另一方面，DSA 和 ECDSA 的签名大小相同：大约 $4t$ 位，其中 t 是公式中的指数 2^t, 即对于 80 位的安全级别，大约需要 320 位，相当于 2^80 次操作

下面是对 ECDSA 算法的描述

Algorithm 1 ECDSA Sign

选取曲线 $CURVE$ 和 n 阶生成点 G, 选取 d_A 作为私钥, 计算 $Q_A = d_A \cdot G$ 作为公钥

1. 给定消息 m, 计算 $e = HASH(m)$, 其中 $HASH$ 可以选取 SHA-2 等函数
2. 随机选取密码学意义上安全的随机数 $k \in [1, n-1]$
3. 计算点 $(x_1, y_1) = k \cdot G$
4. 计算 $r = x_1 \ mod \ n$, 如果 $r = 0$, 则重新选取 k
5. 计算 $s = k^{-1}(e + rd_A) \ mod \ n$, 如果 $s = 0$, 则重新选取 k
6. 输出 $[r, s]$ 作为签名

---

2 原理

给定签名 (r, s) 和消息值 m，我们通过以下恢复算法可以 (可能) 恢复公钥

Algorithm 2 ECDSA Recovery

1. 验证 r 和 s 是整数 $[1, n-1]$, 否则签名无效
2. 计算曲线点 $R = (x_1, y_1)$, 可能会计算出多个点
3. 计算 $e = HASH(m)$, 此处使用的 HASH 与签名生成时使用的函数相同
4. 计算 $u_1 = -zr^{-1}\ mod\ n$,$u_2 = sr^{-1}\ mod\ n$
5. 计算点 $Q_A = (x_A, y_A) = u_1 \cdot G + u_2 \cdot R$
6. 如果 Q_A 匹配 Alice 的公钥则签名有效

3 具体实现

3.1 ECDSA Sign

Listing 1: ECDSA Sign

```
def Sign(sk, m):
    Hash = hashlib.sha256(m.encode())
    Hash = int(Hash.hexdigest(), 16)
    e = Hash
    n = ecdsa.generator_secp256k1.order()
    k = random.randint(2 ** 160, 2 ** 161)
    R = k * G
    r = pow(R.x(), 1, n)
    s = pow(ecdsa.numbertheory.inverse_mod(k, n) * (e + r * sk), 1,
        n)
    return r, s
```

3.2 ECDSA Recovery

Listing 2: ECDSA Recovery

```
def Recover(r, s, m):
    Hash = hashlib.sha256(m.encode())
    Hash = int(Hash.hexdigest(), 16)
    e = Hash
    x = r
```

```
    curve = ecdsa.curve_secp256k1
    n = ecdsa.generator_secp256k1.order()


    y2 = (pow(x, 3, curve.p()) + (curve.a() * x) + curve.b()) % curve.p()
    print(y2)#y^2  = x^3 + 7

    y = ecdsa.numbertheory.square_root_mod_prime(y2, curve.p())
    print(y)#sqrt(y^2)

    u_1 = (-ecdsa.numbertheory.inverse_mod(r, n) * e) % n
    u_2 = (ecdsa.numbertheory.inverse_mod(r, n) * s) % n

    R_1 = ellipticcurve.PointJacobi(curve, x, y, 1)
    P_1 = u_1*G + u_2 *R_1
    print(P_1)

    R_2 = ellipticcurve.PointJacobi(curve, x, -y, 1)
    P_2 = u_1*G + u_2 *R_2
    print(P_2)
```

4 实现效果

4.1 测试代码

测试代码如下:

Listing 3: test bench

```

sk = random.randint(2 ** 160, 2 ** 161)
pk = sk * G
print("PK:", pk)
M = 'Sunshine Rainbow Pony'
```

```

r, s = Sign(sk, M)
print('r:',r)
print('s:',s)
Recover(r, s, M)
```

4.2 执行结果

执行结果如下：

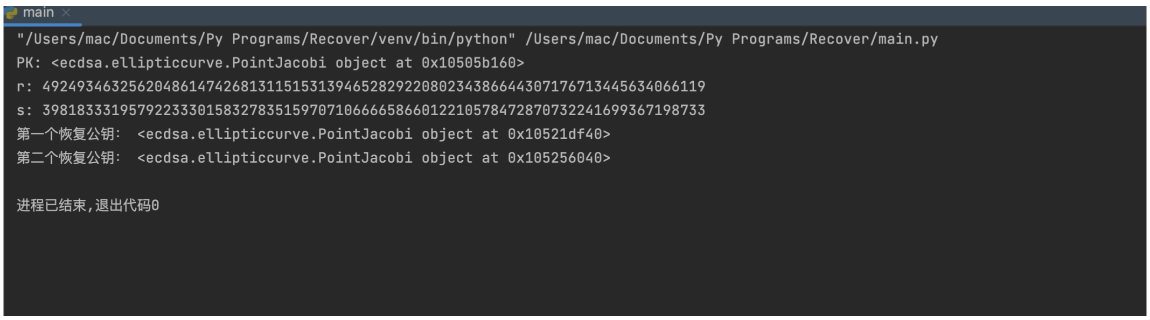

图 1: 执行结果

参考文献

[1] https://en.wikipedia.org/wiki/Ethereum

[2] https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm

[3] https://medium.com/asecuritysite-when-bob-met-alice/can-we-recover-the-public-key-from-an-ecdsa-signature-7af4b56a8a0f