The application of deduce technique in Ethereum with ECDSA

201900460002 鲁逸夫

此篇报告主要介绍在以太坊交易中是通过 ECDSA 签名恢复出公钥的原理，以及该恢复技术是对于区块链系统性能提升的影响。

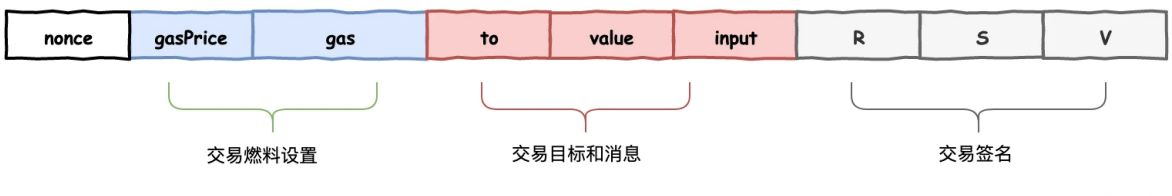
以太坊交易

在深入阐释原理之前，我们需要先了解一般以太坊交易的格式[1]。

一笔以太坊交易在经过解析后，其数据结构可以按如下 json 格式表示：

```
{
 "nonce": "0x16",
 "gasPrice": "0x2",
 "gas": "0x1",
 "to": "0x0100000000000000000000000000000000000000",
 "value": "0x0",
 "input": "0x616263646566",
 "v": "0x25",
 "r": "0x3c46a1ff9d0dd2129a7f8fbc3e45256d85890d9d63919b42dac1eb8dfa443a32",
 "s": "0x6b2be3f225ae31f7ca18efc08fa403eb73b848359a63cd9fdeb61e1b83407690",
 "hash": "0xb848eb905affc383b4f431f8f9d3676733ea96bcae65638c0ada6e45038fb3a6"
}
```

按照不同的用途，上述数据结构可分为如下4个部分：



以太坊技术与实现

- 一笔交易的开头是一个 uint64 类型的 Nonce 值，称之为随机数，用于撤销交易、防止双花和修改交易信息。

- 第二部分是关于交易执行限制的设置，gas 为愿意供以太坊虚拟机运行的燃料上限。 gasPrice 是愿意支付的燃料单价。gasPrcie $\times$ gas 是愿意为这笔交易支付的最高手续费。
- 第三部分是交易发送者输入以太坊虚拟机执行此交易的初始信息，其中包含：虚拟机操作对象（接收方 To）、从交易发送方转移到操作对象的资产（Value），以及虚拟机运行时入参(input)。
- 最后是与此文要阐述的点压缩技术密切相关的数据，是交易发送方对交易的签名结果，只有拥有正确签名的交易才能被执行。

ECDSA 介绍

以太坊（或比特币等区块链平台）使用椭圆曲线数字签名算法 （ECDSA，Elliptic Curve Digital Signature Algorithm）认证交易的合法性，每一笔区块链交易执行之前都必须进行权限校验，以确保该交易是由账户对应的私钥签发。这里给出 ECDSA 的描述[2]:

参数	说明
CURVE	使用的椭圆曲线方程
G	椭圆曲线的基点，由此作为生成元可在曲线上生成阶为素数 n 的子群
n	群*G*的阶
d_A	私钥（随机选择）
Q_A	公钥 $d_A \times G$ （由椭圆曲线计算）
m	要发送的消息

签名算法

1. 计算 $z = \mathtt{HASH}(m)$.
2. 选取随机整数 $k \in [1, n-1]$.
3. 计算椭圆曲线上的点 $(x_1, y_1) = k \times G$.
4. 计算 $r = x_1 \bmod n$. 如果 $r = 0$ 则重新选取整数 k.
5. 计算 $s = k^{-1}(z + rd_A) \bmod n$. 如果 $s = 0$ 则重新选取整数 k.
6. 输出签名为 (r, s).

验签算法

1. 验证 r 和 s 都是在 $[1, n-1]$ 的整数，如不满足则签名不合法。
2. 计算 $z = \mathtt{HASH}(m)$，这里使用的 HASH 函数与签名算法中相一致。
3. 计算 $u_1 = zs^{-1} \bmod n$ 和 $u_2 = rs^{-1} \bmod n$.
4. 计算椭圆曲线上的点 $(x_1, y_1) = u_1 \times G + u_2 \times Q_A$. 如果 $(x_1, y_1) = O$，则签名不合法。
5. 如果 $r \equiv x_1 \pmod n$ 则签名合法，否则不合法

算法的正确性容易看出，下面介绍**在不提供额外信息**，仅已知消息 m 和签名(r, s)， 如何恢复出签名者的公钥：

恢复公钥

1. 验证 r 和 s 都是在 $[1, n-1]$ 的整数，如不满足则签名不合法。
2. 计算曲线上的点 $R = (x_1, y_1)$，这里 x_1 是 $r, r+n, r+2n \ldots$ 等其中的一个值，y_1 由确定 x_1 后通过曲线的方程计算得到。（注意到可能存在许多点 R 满足条件，每个点会恢复出不同的公钥）
3. 计算 $z = \mathtt{HASH}(m)$，这里使用的 HASH 函数与签名算法中相一致。
4. 计算 $u_1 = -zr^{-1} \bmod n$ 和 $u_2 = sr^{-1} \bmod n$.
5. 计算椭圆曲线上的点 $Q_A = (x_A, y_A) = u_1 \times G + u_2 \times R$.

从恢复的流程不难看出，当给定的签名合法时，存在一个由上述方法确定的 Q_A 恰为签名者的公钥。但是此方法在第二步计算点 R 时需要不断尝试可能的 x_1 取值，效率低下，因此引出了以太坊交易中的恢复技术——提供额外信息唯一确定公钥。

【注】上述对于各算法的介绍较简略，省去了一些细节和可能的误用造成的安全问题，进一步详细了解可以查看维基百科[2]

公钥恢复技术

在前面以太坊交易一节中，可以发现数据结构中交易签名部分是由 (r, s, v) 3个部分组成， 这里多出来的值 v 就是Recovery Identifier（recid） 用于从签名唯一确定对应的公钥，以便加快验证恢复地址。接下来对以太坊使用的曲线参数和 recid 不同取值的含义进行具体解释。

Secp256k1

在[以太网官方实现（go-ethereum）](https://github.com/ethereum/go-ethereum/tree/master/crypto/secp256k1)中可以找到以太坊协议中使用的曲线为 Secp256k1.

该曲线定义为：$y^2 = x^3 + 7,\ x, y \in \mathbb{F}_p$，具体参数为：

$$p = \mathtt{FFFFFFFF\ FFFFFFFF\ FFFFFFFF\ FFFFFFFF\ FFFFFFFF\ FFFFFFFF\ FFFFFFFE\ FFFFFC2F}$$
$$= 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$$

$$G = (\mathtt{79BE667E\ F9DCBBAC\ 55A06295\ CE870B07\ 029BFCDB\ 2DCE28D9\ 59F2815B\ 16F81798},$$

$$\mathtt{483ADA77\ 26A3C465\ 5DA4FBFC\ 0E1108A8\ FD17B448\ A6855419\ 9C47D08F\ FB10D4B8})$$

G 的阶
$$n = \mathtt{FFFFFFFF\ FFFFFFFF\ FFFFFFFF\ FFFFFFFE\ BAAEDCE6\ AF48A03B\ BFD25E8C\ D0364141}$$
$$\approx 2^{256} - 2^{128}$$

因此对于 `Secp256k1` 而言，曲线上的点 R 的坐标 x 和 y 是模 p 得到，取值范围约为 $(0, 2^{256} - 2^{32})$；而签名中的 r 和 s 是模 n 得到，取值范围约为 $(0, 2^{256} - 2^{128})$。当 x 在 n 和 p 之间时，r 被还原为 $x - n$。因此如果 r 的取值小于 $2^{128} - 2^{32}$，可能有两个有效的 x 值与之相对应。

所以 v 的作用就是帮助验签的一方区分 $R = (x, y)$ 的取值是4种情况（x 取 r 还是 $r + n$，y 的奇偶性）中的哪一种。

Recovery Identifier

这里整理成一张表给出不同 v 值的含义：

v 的取值	y 的奇偶性	x 的取值
27	偶数	小于群 G 的阶 n
28	奇数	小于群 G 的阶 n
29	偶数	大于群 G 的阶 n
30	奇数	大于群 G 的阶 n

注意到 Secp256k1 曲线的参数，当 r 的取值在 1 和 $(p \bmod n)$ 之间时，才有可能对应两个可能的 x 取值，这有大约 $\frac{2^{128}-2^{32}}{2^{256}-2^{128}} \approx \frac{1}{2^{128}} \approx 0.00000000000000000000000000373\%$ 的概率发生。所以 v 的值取 29 和 30 是极其罕见的情形，现实中只在专门生成的例子中出现过，也因此在很多实现中[4]，v 在函数内部取值只有 0 或 1。还有一点需要说明的是，这里 27 是任意挑选的数字与 v 的取值相加应用于签署比特币交易信息，以太坊中也采用了 27 这个数字。

在 EIP 115 之后较新的协议中，为防止以太坊经典的交易重放攻击，v 值的计算方式改为 `v = {0,1} + CHAIN_ID * 2 + 35`，这里 `{0,1}` 表示 y 的奇偶性，所以 `CHAIN_ID` 为 1 的以太坊主网中 v 可能取 37 或 38.

总结

为了减少存储空间，希望每笔交易信息占用空间越小越好，以太坊选用 ECDSA 签名来认证交易，这样便可以利用 ECDSA 自身的性质，从消息中恢复出签名者的公钥，而减少存储。又因为存在不止一个有效的公钥可供签名检索，所以需要额外的信息 v 来恢复唯一正确的公钥。

参考资料

[1] 以太坊技术与实现

[2] ECDSA Wikipedia

[3] Secp256k1 Wikipedia

[4] pybitcointools