

一个数字引发的探索——ECDSA解析

原创 李辉忠 FISCO BCOS开源社区 2月18日



李辉忠

FISCO BCOS 高级架构师

和我微信交流



FISCO BCOS交易签名算法基于ECDSA原理进行设计，ECDSA也是比特币和以太坊采用的交易签名算法。

本文介绍ECDSA及椭圆曲线加密（ECC）相关知识、ECDSA的Recover机制和实现方式、FISCO BCOS交易签名和验签的底层原理。内容偏硬（shu）核（xue），欢迎对密码学原理、区块链底层原理感兴趣的开发者一起交流。

..... FISCO BCOS

故事开始

故事要从以太坊中一个神奇的魔数开始说起。

Transactions are signed using recoverable ECDSA signatures. This method utilises the SECP-256k1 curve as described by Courtois et al. [2014], and is implemented similarly to as described by Gura et al. [2004] on p. 9 of 15, para. 3.

It is assumed that the sender has a valid private key p_r , which is a randomly selected positive integer (represented as a byte array of length 32 in big-endian form) in the range $[1, \text{secp256k1n} - 1]$.

We assume the existence of functions ECDSAPUBKEY, ECDSASIGN and ECDSARECOVER. These are formally defined in the literature, e.g. by Johnson et al. [2001].

(277) $\text{ECDSAPUBKEY}(p_r \in \mathbb{B}_{32}) \equiv p_u \in \mathbb{B}_{64}$
 (278) $\text{ECDSASIGN}(e \in \mathbb{B}_{32}, p_r \in \mathbb{B}_{32}) \equiv (v \in \mathbb{B}_1, r \in \mathbb{B}_{32}, s \in \mathbb{B}_{32})$
 (279) $\text{ECDSARECOVER}(e \in \mathbb{B}_{32}, v \in \mathbb{B}_1, r \in \mathbb{B}_{32}, s \in \mathbb{B}_{32}) \equiv p_u \in \mathbb{B}_{64}$

Where p_u is the public key, assumed to be a byte array of size 64 (formed from the concatenation of two positive integers each $< 2^{256}$), p_r is the private key, a byte array of size 32 (or a single positive integer in the aforementioned range) and e is the hash of the transaction, $h(T)$. It is assumed that v is the 'recovery identifier'. The recovery identifier is a 1 byte value specifying the parity and finiteness of the coordinates of the curve point for which r is the x-value; this value is in the range of $[27, 30]$, however we declare the upper two possibilities, representing infinite values, invalid. The value 27 represents an even y value and 28 represents an odd y value.

We declare that an ECDSA signature is invalid unless all the following conditions are true⁵:

(280) $0 < r < \text{secp256k1n}$
 (281) $0 < s < \text{secp256k1n} \div 2 + 1$
 (282) $v \in \{27, 28\}$

以太坊黄皮书中，关于交易签名的阐述讲到两个特殊的数「27，28」，实际上是从「0，1」通过加了一个27演变得到「27，28」，所以本质上是一个特殊的数27。

这个特殊的数字27代表了什么含义呢？

一次侦探之旅开始了...

这像是一个bug

//////////

搜索发现此前已有许多关于该问题的讨论，其中，Stack Exchange的一篇帖子指出这是一个设计bug。以太坊源码github上，也有一个相关issue，该issue被打上了「type:bug」的标签。

What's the magic numbers meaning of 27 or 28 in VRS (use to ecover the sender) ? why not 0 or 1?

Asked 27 days ago Active 27 days ago Viewed 26 times



1 What's the magic numbers meaning of 27 or 28 in VRS (use to ecover the sender) ? why not 0 or 1? Ecover need 0 or 1, but transaction RLP is 27 or 28. Before ecover, sub 27 to get 0 or 1. Why not encode the transaction use 0 or 1 directly? What's the magic number 27 and 28 mean?

signature ecover

share edit

add a comment

asked Jan 13 at 12:37

Jimmy
111 • 1

1 Answer

active oldest votes

1 This issue is reported at [package 'ethereumjs-util' / function 'fromRpcSig'](#).

As you can see, the documentation of this function states:

- NOTE: all because of a bug in geth: <https://github.com/ethereum/go-ethereum/issues/2053>

And as you can see, the values 0 and 1 are supported, while the function adds 27 to the last byte in the 65-byte buffer (the last two characters in the 130-character string) in order to support the values 27 and 28 as well.

A similar question has been answered [here](#).

Welcome!

This is a collaboratively and answer site for use the decentralized app and smart contract er. It's 100% free, no regis

Got a question about th is the place to talk abo questions are appropri should use, etc.

about

Blog

How Shapeways' sol printing at scale

The Overflow #10: T

Featured on Meta

TLS 1.0 and TLS 1.1 Exchange services

Linked

ethereum / go-ethereum

Watch 2k Unstar 25.4k Fork 9.2k

Code Issues 281 Pull requests 78 Actions Projects 9 Wiki Security Insights

RPC: eth.sign should return the ethereum canonical version of the signature #2053

New issue

Closed obscuren opened this issue on 5 Dec 2015 · 8 comments



obscuren commented on 5 Dec 2015

Member + 👤 ...

Ethereum defines the `v` param as `v + 27` (Py relic)

👍 2

obscuren added `type:bug` `RPC` labels on 5 Dec 2015

Assignees

No one assigned

Labels

area:rpc

type:bug

Projects

Stack Exchange帖子中有一个链接给出了修复该Bug的代码，请看下面截图（红框）。在注释说

明和代码可见，fromRpcSig函数对27这个魔数进行了特殊处理。从RPC过来的签名中，v值如果小于27（可能是0-3），则直接加上27作为新v值，fromRpcSig函数通过这种方式兼容ECDSA原始v值（也就是recoveryID）和以太坊v值。

```
66  /**
67   * Convert signature format of the `eth_sign` RPC method to signature parameters
68   * NOTE: all because of a bug in geth: https://github.com/ethereum/go-ethereum/issues/2053
69   */
70  export const fromRpcSig = function(sig: string): ECDSASignature {
71    const buf: Buffer = toBuffer(sig)
72
73    // NOTE: with potential introduction of chainId this might need to be updated
74    if (buf.length !== 65) {
75      throw new Error('Invalid signature length')
76    }
77
78    let v = buf[64]
79    // support both versions of `eth_sign` responses
80    if (v < 27) {
81      v += 27
82    }
83
84    return {
85      v: v,
86      r: buf.slice(0, 32),
87      s: buf.slice(32, 64),
88    }
89  }
90
```

这真是以太坊设计的一个bug吗？

回到刚才那个fromRpcSig的源代码文件，详细看其各接口实现，我们发现有这样一行代码「v: chainId ? recovery + (chainId * 2 + 35) : recovery + 27」，这行为v赋值的代码透露了三个信息，分别是魔数27、魔数35和ChainID。


```

12  /**
13   * Returns the ECDSA signature of a message hash.
14   */
15  export const ecdsign = function(
16    msgHash: Buffer,
17    privateKey: Buffer,
18    chainId?: number,
19  ): ECDSASignature {
20    const sig = secp256k1.sign(msgHash, privateKey)
21    const recovery: number = sig.recovery
22
23    const ret = {
24      r: sig.signature.slice(0, 32),
25      s: sig.signature.slice(32, 64),
26      v: chainId ? recovery + (chainId * 2 + 35) : recovery + 27,
27    }
28
29    return ret
30  }

```

于是，疑问更多了，魔数35是什么？ChainID又是什么？

这不像是一个Bug

////////////////////

带着这些疑问，再一次查阅相关设计材料，我们看到，以太坊EIP155中描述了有关ChainID的设计。基于以太坊源码构建的网络，实际运行的链有很多，为了防止一条链的交易被提交上链到另一条链，造成重放攻击，引入了ChainID的设计，在块高2,675,000的位置进行分叉实现。

eip	title	author	type	category	status	created
155	Simple replay attack protection	Vitalik Buterin (@vbuterin)	Standards Track	Core	Final	2016-10-14

Hard fork

Spurious Dragon

Parameters

- `FORK_BLKNUM` : 2,675,000
- `CHAIN_ID` : 1 (main net)

Specification


If `block.number >= FORK_BLKNUM` and `v = CHAIN_ID * 2 + 35` or `v = CHAIN_ID * 2 + 36`, then when computing the hash of a transaction for purposes of signing or recovering, instead of hashing only the first six elements (i.e. nonce, gasprice, startgas, to, value, data), hash nine elements, with `v` replaced by `CHAIN_ID`, `r = 0` and `s = 0`. The currently existing signature scheme using `v = 27` and `v = 28` remains valid and continues to operate under the same rules as it does now.

明白了ChainID的作用，另一个疑问又产生了——以太坊中，有NetworkID来区分不同网络，为什么还需要ChainID？

这要从NetworkID和ChainID的作用范围来解释。NetworkID主要在网络层面进行链的隔离，节点在建立相互连接的时候需要交换NetworkID，拥有一致的NetworkID才能完成握手连接。ChainID是交易层面，防止不同网络的交易被交叉重复攻击。


以太坊（ETH）和经典以太坊（ETC）的主网NetworkID都是1，需要通过 ChainID机制才能防止交易在ETH和ETC网络之间交叉重放，ETH主网的ChainID是1，ETC主网的ChainID是61。

说到这里其实还是没有搞清楚为什么是27，为什么是35？我们在EIP github的Issue#155中看到Jan和Buterin的交流记录，看来27是来自比特币的产物。

 **janx** commented on 14 Oct 2016 • edited ▾ Member + 😊 ...

Out of curious: why make `v=27|28` instead of `v=0|1` at first place?

👍 3

 **vbuterin** commented on 15 Oct 2016 Contributor Author + 😊 ...

It was a carry-over from Bitcoin's Electrum wallet.

顺藤摸瓜，打开electrum的github，我们在electrum/electrum/ecc.py中找到如下代码

```
451 def sign_message(self, message: bytes, is_compressed: bool, algo=lambda x: sha256d(msg_magic(x))) -> bytes:
452     def bruteforce_recid(sig_string):
453         for recid in range(4):
454             sig65 = construct_sig65(sig_string, recid, is_compressed)
455             try:
456                 self.verify_message_for_address(sig65, message, algo)
457                 return sig65, recid
458             except Exception as e:
459                 continue
460         else:
461             raise Exception("error: cannot sign message. no recid fits..")
462
463     message = to_bytes(message, 'utf8')
464     msg_hash = algo(message)
465     sig_string = self.sign(msg_hash,
466                           sigencode=sig_string_from_r_and_s,
467                           sigdecode=get_r_and_s_from_sig_string)
468     sig65, recid = bruteforce_recid(sig_string)
469     return sig65
---
```

```
496 def construct_sig65(sig_string: bytes, recid: int, is_compressed: bool) -> bytes:
497     comp = 4 if is_compressed else 0
498     return bytes([27 + recid + comp]) + sig_string
```

```
286 def verify_message_for_address(self, sig65: bytes, message: bytes, algo=lambda x: sha256d(msg_magic(x))) -> None:
287     assert_bytes(message)
288     h = algo(message)
289     public_key, compressed = self.from_signature65(sig65, h)
290     # check public key
291     if public_key != self:
292         raise Exception("Bad signature")
293     # check message
294     self.verify_message_hash(sig65[1:], h)
---
```

```

209     def from_signature65(cls, sig: bytes, msg_hash: bytes):
210         if len(sig) != 65:
211             raise Exception("Wrong encoding")
212         nV = sig[0]
213         if nV < 27 or nV >= 35:
214             raise Exception("Bad encoding")
215         if nV >= 31:
216             compressed = True
217             nV -= 4
218         else:
219             compressed = False
220         recid = nV - 27
221         return cls.from_sig_string(sig[1:], recid, msg_hash), compressed

```

从代码中可见，electrum在签名时，为原本只有0-3之间的recid（recoveryID）加上了27，还有一个压缩标记，如果有压缩则再加上4，recid的值范围在27-34。

至此可知，27和35大概来源于此，以太坊继承比特币的设计，在比特币源码bitcoin/src/key.cpp的CKey::SignCompact函数中也确定了该实现方式，但是比特币为什么如此设计，仍未可知。

ECDSA才是“bug”

//////////

故事到这里，我们对以太坊代码中那个魔数27的前世今生有大概了解，但这仅仅是故事的开端，由此引发我们进一步思考一个问题：recoveryID是什么？

为了解释清楚这个问题，我们需要从ECDSA算法着手，从数学角度理解其背后的原理。ECDSA是FISCO BCOS采用的交易签名算法，由此我们会发现，ECDSA算法有一种Recover机制，它才是真正“bug”级别的功能。

ECDSA（Elliptic Curve Digital Signature Algorithm）是基于椭圆曲线的数字签名算法。数字签名算法是采用公私钥体系实现类似写在纸上的普通签名，用于鉴别数字信息的方法，常见的数字签名算法包括DSA、RSA和ECDSA等。

椭圆曲线密码（ECC）是基于椭圆曲线数学的公钥加密算法，建立在椭圆曲线离散对数困难问题之上，常用的协议有ECDH、ECDSA和ECIES等。

椭圆曲线的参数可以有多种配置方式，也就存在多种不同的曲线，例如secp256k1、secp256r1、Curve25519等，不同曲线的安全性存在一些区别，在SafeCurves中有相关对比描

述。

ECDSA算法主要包括以下四个关键功能：

产生密钥GenKey

- 选择一条椭圆曲线 $E_P(a,b)$ ，选择基点 G ， G 的阶数为 n
- 选择随机数 $d \in n$ 为私钥，计算公钥 $Q = d \cdot G$

签名算法Sign

- 对消息 m 使用消息摘要算法，得到 $z = \text{hash}(m)$
- 生成随机数 $k \in n$ ，计算点 $(x, y) = k \cdot G$
- 取 $r = x \bmod n$ ，若 $r = 0$ 则重新选择随机数 k
- 计算 $s = k^{-1}(z + rd) \bmod n$ ，若 $s = 0$ 则重新选择随机数 k
- 上述 (r,s) 即为ECDSA签名

验证算法Verify

使用公钥 Q 和消息 m ，对签名 (r,s) 进行验证。

- 验证 $r, s \in n$
- 计算 $z = \text{hash}(m)$
- 计算 $u_1 = zs^{-1} \bmod n$ 和 $u_2 = rs^{-1} \bmod n$
- 计算 $(x, y) = u_1 \cdot G + u_2 \cdot Q \bmod n$
- 判断 $r == x$ ，若相等则签名验证成功

恢复算法Recover

已知消息 m 和签名 (r,s) ，恢复计算出公钥 Q 。

- 验证 $r, s \in n$
- 计算 $R = (x, y)$ ，其中 $x = r, r+n, r+2n \dots$ ，代入椭圆曲线方程计算获得 R
- 计算 $z = \text{hash}(m)$
- 计算 $u_1 = -zr^{-1} \bmod n$ 和 $u_2 = sr^{-1} \bmod n$
- 计算公钥 $Q = (x', y') = u_1 \cdot G + u_2 \cdot R$

为了回答recoveryID的问题，我们重点关注「恢复算法Recover」。

在计算R的步骤可以看到，存在多个x的取值可能性，导致存在多个R的可能性，因此计算得到的Q也存在多个可能的结果，需要通过和已知的公钥对比，确定哪一个Q是正确的。如果遍历x的所有可能都未找到正确的Q，说明该消息和签名是不对应的，或者是一个未知的公钥。

为了确定正确的Q，需要遍历x的所有可能取值，跑多轮Recover算法，这个时间开销是比较大的。为了提高Recover的时间效率，采用空间换时间的思路，在签名中增加一个v值，用于快速确定x，避免遍历查找试探，这个v值就是recoveryID。

在区块链系统中，客户端对每笔交易进行签名，节点对交易签名进行验证。

如果采用「验证算法Verify」，那节点必须首先知道签发该交易所对应的公钥，因此需要在每笔交易中携带公钥，这需要消耗很大带宽和存储。

如果采用「恢复算法Recover」，并且在生成的签名中携带recoveryID，就可以快速恢复出签发该交易对应的公钥，根据公钥计算出用户地址，然后在用户地址空间执行相应操作。

这里潜藏了一个区块链设计哲学，区块链上的资源（资产、合约）都是归属某个用户的，如果能够构造出符合该用户地址的签名，等同于掌握了该用户的私钥，因此节点无需事先确定用户公钥，仅从签名恢复出公钥，进而计算出用户地址，就可以执行这个用户地址空间的相应操作。

FISCO BCOS基于这个原理设计实现了交易签名和验签。

recoveryID的计算

//////////

关于JavaSDK性能优化的文章（[记一次JavaSDK性能从8000提升至30000的过程](#)）中提到一个关键优化点——recoveryID的计算，这里仔细展开讨论。

ECDSA签名 (r, s) ，其中r是椭圆曲线上一个点 $kG(x, y)$ 对应的 $x \bmod n$ ，相当于签名信息中只留下了X轴坐标相关的值，丢弃了Y轴相关的值。在「恢复算法Recover」中尝试找回Y轴对应的值构造R，进而恢复出公钥。

由于 $r = x \bmod n$ ，因此 $r, r+n, r+2n \dots$ 都可能是合法的原始 x 值，不同的椭圆曲线存在不同数量这样合法的 x 值，FISCO BCOS采用的secp256k1曲线存在两个可能 $r, r+n$ 。

每一个 X 轴坐标对应两个可能的 Y 坐标，因此FISCO BCOS中具备四种可能的 $R, (r, y) (r, -y) (r+n, y') (r+n, -y')$ 。但是，对于一个 r 值存在两个 X 轴坐标的概率极低，低到几乎可以忽略，以太坊中就忽略了这两种小概率事件。

那这个小概率事件的概率具体有多小呢？这要从secp256k1曲线的参数说起，通常描述一个椭圆曲线的点 (x, y) 的时候， x 和 y 的值是 $\bmod p$ 的结果， p 是曲线的参数，它是一个大素数，之前提到的 n 也是曲线的参数，等于这条曲线上点的数量（曲线上点的数量为 $n \cdot h$ ， h 也是曲线参数，该曲线 $h=1$ ），在secp256k1中， n 和 p 的值非常接近，具体可见下图。

2.4.1 Recommended Parameters secp256k1

The elliptic curve domain parameters over \mathbb{F}_p associated with a Koblitz curve **secp256k1** are specified by the sextuple $T = (p, a, b, G, n, h)$ where the finite field \mathbb{F}_p is defined by:

$$\begin{aligned} p &= \text{FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFE} \\ &\quad \text{FFFFFFFF} \\ &= 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1 \end{aligned}$$

The curve $E: y^2 = x^3 + ax + b$ over \mathbb{F}_p is defined by:

$$\begin{aligned} a &= \text{00000000 00000000 00000000 00000000 00000000 00000000 00000000} \\ &\quad \text{00000000} \\ b &= \text{00000000 00000000 00000000 00000000 00000000 00000000 00000000} \\ &\quad \text{00000007} \end{aligned}$$

The base point G in compressed form is:

$$G = \text{02 79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9} \\ \text{59F2815B 16F81798}$$

and in uncompressed form is:

$$G = \text{04 79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9} \\ \text{59F2815B 16F81798 483ADA77 26A3C465 5DA4FBFC 0E1108A8 FD17B448} \\ \text{A6855419 9C47D08F FB10D4B8}$$

Finally the order n of G and the cofactor are:

$$\begin{aligned} n &= \text{FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF BAAEDCE6 AF48A03B BFD25E8C} \\ &\quad \text{D0364141} \\ h &= \text{01} \end{aligned}$$

由于 $r = x \bmod n$ ， x 是 $\bmod p$ 的结果， r 是 $\bmod n$ 的结果， x 值的范围是 $[0, p-1]$ ， r 值的范围是 $[0, n-1]$ 。如果 $r+n$ 也是曲线上的点，则 r 的值必须小于 $p-n$ ，概率为 $(p-n) / p$ ，大约为 $3.73 \cdot 10^{-39}$ ，这个概率是非常小的。

基于签名结果 (r, s) 和签名过程中生成的随机点 (x, y) 的 y 值, recoveryID 的计算方式如下:

1. $\text{id} = y \& 1$; // 「签名算法Sign」中 kG 点的 y 坐标, 根据奇偶性设置 id 值, 因为 y 是 $\text{mod } p$ 的结果, 其奇偶性与坐标轴的正负性是完全对应的
2. $\text{id} |= (x \neq r ? 2 : 0)$; // 小概率事件, 如前文解释
3. if $(s > n / 2)$ $\text{id} = \text{id} \wedge 1$; // 签名计算得出的 s 如果大于 $n/2$ 就会取 $n-s$ 作为 s 值, 因此这里做相应转换, 这两个转换是同时发生的

[JavaSDK性能优化](#) 的文章就是基于这个计算公式, 将遍历探寻 recoveryID 改为计算获得, 大幅提升了性能。

后话

从一个神奇的数字开始, 查阅相关资料, 了解设计原理, 进而闯入 ECDSA 的世界, 在一堆数学公式中迷茫、游荡, 问题一个接着一个。一开始雾里看花, 似懂非懂, 靠着处女座的洁癖精神, 总算把心中疑问一一化解。

精妙绝伦的密码协议, 高深莫测的数学理论, 做一个区块链码农, 要学习的东西还很多。唯有苦其心志, 劳其筋骨, 善待每一个疑点, 不放过每一处细节。

总会有一天, 那时——
拨开云雾见天日, 守得云开见月明。

参考资料

Ethereum yellow paper

<https://ethereum.github.io/yellowpaper/paper.pdf>

Stack Exchange

<https://ethereum.stackexchange.com/questions/78929/whats-the-magic-numbers-meaning-of-27-or-28-in-vrs-use-to-recover-the-sender/78930?newreg=a613981e27424bc4b1c2c0e00d939327>

ISSUE

<https://github.com/ethereum/go-ethereum/issues/2053>

EIP155

<https://github.com/ethereum/EIPs/blob/master/EIPS/eip-155.md>

EIP155 Issue

<https://github.com/ethereum/eips/issues/155>

ChainID

<https://chainid.network>

ECC wiki

https://en.wikipedia.org/wiki/Elliptic-curve_cryptography

ECDSA wiki

https://en.wikipedia.org/wiki/Elliptic_Curve_Digital_Signature_Algorithm

Curve25519 wiki

<https://en.wikipedia.org/wiki/Curve25519>

SafeCurves

<https://safecurves.cr.yp.to/index.html>

ECC SEC1

<http://www.secg.org/sec1-v2.pdf>

ECC SEC2

<https://www.secg.org/sec2-v2.pdf>

FISCO BCOS的代码完全开源且免费

下载地址↓↓↓

<https://github.com/FISCO-BCOS/FISCO-BCOS>

