ECDSA——由签名推出公钥

一、ECDSA 签名的完整描述

一、ECDSA

1. 密钥生成：选择一条椭圆曲线 $E_p(a,b)$ 与基点 G
   n 为 G 的阶，选择私有密钥 $d<n$，利用基点 G 计算公开密钥 $P=dG$
   即 $sk=d$，$pk=P$
2. 签名算法：$sign_{sk}(m)$
   $k \leftarrow Z_n^*$，$R=kG$，$r=R_x \bmod n$（其中 $r\neq 0$，否则重新选择）
   $e=hash(m)$，$s=k^{-1}(e+dr) \bmod n$　输出签名 (r,s)
3. 验证算法：$vrfy_{pk}(m,r,s)$
   $e=hash(m)$　$w=s^{-1} \bmod n$　$(r',s')=e\cdot wG+r\cdot wP$
   当且仅当 $r'==r$ 时，验证通过，输出1；否则输出0
4. 正确性证明：$es^{-1}G+rs^{-1}P=e\cdot wG+r\cdot wP=es^{-1}G+rs^{-1}P$
   $=s^{-1}(eG+rP)=k(e+dr)^{-1}(eG+rP)=k(e+dr)^{-1}(eG+drG)$
   $=k(e+dr)^{-1}(e+dr)G=kG=R=(r',s')$
   $\because r=R_x \bmod n$　$\therefore$ 当且仅当 $r=r'$ 时，验证通过，正确性即证

二、ECDSA 由签名到公钥研究的必要性

以太坊（比特币、区块链）中的每一个产生的区块，会传播到全网上，所以倾向于使得区块变得更小，对于区块在全球的网络广播很有帮助，可以减少所需的流量，提高整体的性能，并且有利于减少区块链整体存储的体积。

而从整体流程来看，首先用户用自己的私钥进行签名，其他人用该用户的公钥来进行验签，因此直观上，会将用户的公钥放在区块中存储，或存储在区块链的某个地方去，则其他人来验证签名时，可以来从中取得用户的公钥后，再进行验证签名。

但是基于整体性能和存储量的考虑，在以太坊中，人们更希望能够通过签名值推算出公钥来，这样可以提高区块链的性能和存储的体积。在实际应用中，也是如此应用的，只是需要额外的一点信息即可。

三、ECDSA 由签名到公钥的推导

为了推出公钥，我们需要提前已知的消息有：原消息 m，签名（r，s），公开参数 如：

（G，n 等），还有额外的信息 v（恢复标识符）。其中，r 和 s 为 32 字节，额外信息 v 为 1 字节。

值得强调的是 v 的重要性。

v 不是 27 (0x1b) 就是 28 (0x1c)。恢复标识符非常重要，因为我们使用的是椭圆曲线算法，仅凭 r 和 s 可计算出曲线上的多个点，因此会恢复出两个不同的公钥（及其对应地址）。v 会告诉我们应该使用这些点中的哪一个。

而且在椭圆曲线点的坐标的运算中，一般是模 p 运算的，而在另外其他的运算（如：kG）中是模 n 运算的，因此在两个不同的域中，需要额外的一些信息来指明，来防止计算出错。

在大多数实现中，v 在内部只是 0 或 1，而 27 是在签署比特币消息时加上的任意数。以太坊也接受了这一点。

从 EIP-155 开始，我们还使用链 ID 来计算 v 值。这可以防止跨链重放攻击：以太坊上签署的交易无法在以太坊经典上使用，反之亦然。

数学推导过程如下：（为了表示方便，以下为手写扫描的推导过程）

已知的信息：消息 m，签名 (r, s)，公开参数：(如 G, n 等)，额外信息 v（恢复标识符）

$r = R_x \bmod n = (kG)_x \bmod n$ $R=(x,y)$，其中 x 为 r（当 $v=27$ 时），或 x 为 $r+n$（当 $v=28$ 时）

由上述可计算出椭圆曲线上的点 $R=(x,y)=kG$

$\because s = k^{-1}(e+dr) \bmod n$ $\therefore s \cdot kG = (e+dr)G \bmod n = eG + rP \bmod n$

$\therefore P = (sR - eG) \cdot r^{-1} \bmod n$（其中 $e = H(m)$ 可计算）

综上所述，可通过如上过程，由签名与额外信息推出公钥 P