Report on the application of this deduce technique in Ethereum with ECDSA

ECDSA

- ECDSA(Elliptic Curve Digital Signature Algorithm)是使用椭圆曲线密码（ECC）对数字签名算法（DSA）的模拟。ECDSA于1999年成为ANSI标准，并于2000年成为IEEE和NIST标准。
- ECDSA是ECC与DSA的结合，整个签名过程与DSA类似，所不一样的是签名中采取的算法为ECC，最后签名出来的值是(r, s)
- **ECDSA**在经典 Weierstrass 形式的有限域上使用加密**椭圆曲线**(EC) 。这些曲线由它们的**EC 域参数描述**，由各种加密标准指定，例如**SECG：SEC 2**和**Brainpool (RFC 5639)**。

简述ECDSA过程

密码学中的椭圆曲线需要定义：

> **Generator point G**, used for scalar multiplication on the curve (multiply integer by EC point)
>
> **Order *n*** of the subgroup of EC points, generated by **G**, which defines the length of the private keys (e.g. 256 bits)

1,key generation

$P = dG$ 其中G 为椭圆曲线群生成元，d 是选取的私钥, n 是循环群的阶

2,Sign

- 随机选取参数k ,计算$R = kG$
- 取点R 的x坐标即为输出签名的第一部分
- $e = hash(m)$
- $s = k^{-1}\ (e + dr)\ mod\ n$
- 输出签名(r, s)

3,verify signature

- $e = hash(m)$

- $w = s^{-1} \, mod \, n$
- $(r', s') = e * wG + r * wP$
- Check if $r' == r$
- Holds for correct signature

椭圆曲线上的运算实现（python）

```python
# 点的加法运算
def point_add(P,Q):


    if P is None:
        return Q
    elif Q is None:
        return P
    elif P[0]==Q[0] and P[1]!=Q[1]:
        return None
    #elif P == point_neg(Q):
        return None

    x1, y1 = P[0], P[1]
    x2, y2 = Q[0], Q[1]

    if P == Q:
        lam = ((3 * x1 * x1 + a) * invert(2*y1,p)) % p
    else:
        lam = ((y2 - y1) * invert(x2-x1,p)) % p

    x3 = (lam * lam - x1 - x2) % p
    y3 = (lam * (x1 - x3) - y1) % p

    return (x3,y3)

# 点的取反运算
def point_neg(P):
    if P is None:
        return None

    x, y = P
    return x, (-y) % p

# 标量乘运算

def point_mul(P,h):
    h=h%q
    if h % q == 0 or P is None:
        return None

    if h < 0:
        #return point_neg(point_mul(-k, P))
        return point_mul(P,q-h)

    Q = None
    while h:
        if h & 1:
            Q = point_add(Q, P)
        P = point_add(P, P)
```

```
        h >>= 1
    return Q
```

ECDSA实现关键部分（python）

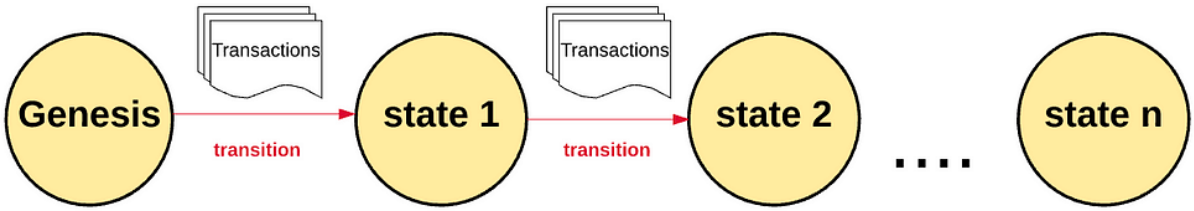
```
def signature(mes,d):
    k=random.getrandbits(256)%q
    R=point_mul(G,k)
    r=R[0]%q
    e=sm3_hash(mes.encode())
    e=int(e,16)
    s=(invert(k,q)*(e+d*r))%q
    return (r,s)

def verify_sig(mes,Q,sig):
    e=sm3_hash(mes.encode())
    e=int(e,16)
    r=sig[0]
    s=sig[1]
    w=invert(s,q)
    ewG=point_mul(G,e*w)
    #print('r',r)
    rwP=point_mul(Q,r*w)
    (r1,s1)=point_add(ewG,rwP)
    if r1==r:
        print('签名验证成功')
    else:
        print('签名验证失败')
```

Ethereum(ETH)

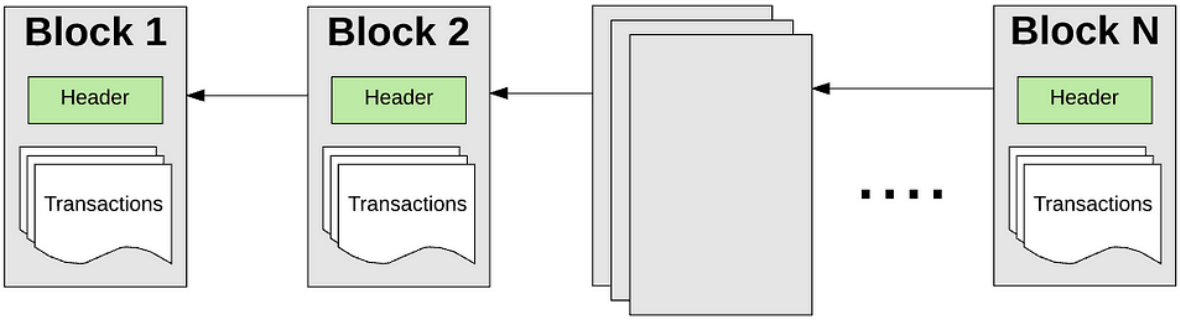
注意，以太坊的hash是KECCAK-256 hash。

以太坊模型说明：

- 以太坊的本质就是一个基于交易的状态机(transaction-based state machine)。在计算机科学中，一个 状态机 是指可以读取一系列的输入，然后根据这些输入，会转换成一个新的状态出来的东西。
- 根据以太坊的状态机，我们从创世纪状态(genesis state)开始。这差不多类似于一片空白的石板，在网络中还没有任何交易的产生状态。当交易被执行后，这个创世纪状态就会转变成最终状态。在任何时刻，这个最终状态都代表着以太坊当前的状态。



- 以太坊的状态有百万个交易。这些交易都被“组团”到一个区块中。一个区块包含了一系列的交易，每个区块都与它的前一个区块链接起来。

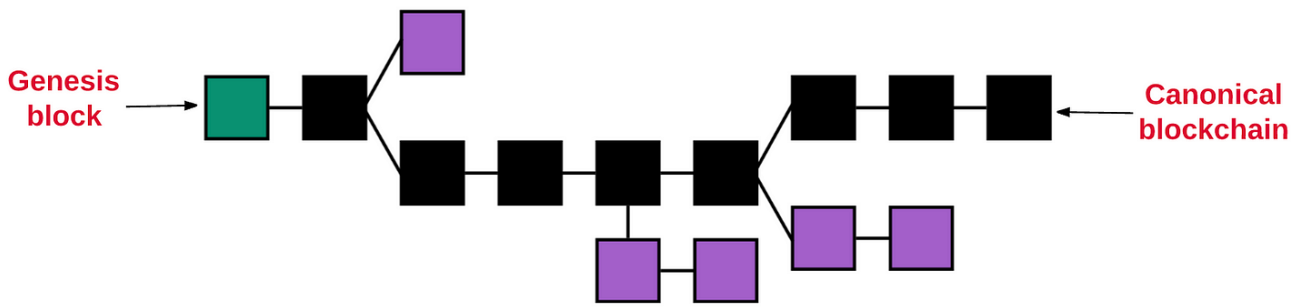


- 为了让一个状态转换成下一个状态，交易必须是有效的。为了让一个交易被认为是有效的，它必须要经过一个验证过程，此过程也就是挖矿。
- 为了让一个区块添加到主链上，一个矿工必须要比其他矿工更快的提供出这个“证明”。通过矿工提供的一个数学机制的“证明”来证实每个区块的过程称之为工作量证明(proof of work)。

确定有效路径以及防止链分支：GHOST

> GHOST = Greedy Heaviest Observed Subtree

GHOST协议就是让我们必须选择一个在其上完成计算最多的路径。一个方法确定路径就是使用最近一个区块（叶子区块）的区块号，区块号代表着当前路径上总的区块数（不包含创世纪区块）。



_

以太坊一些重要概念（与ECDSA相关）

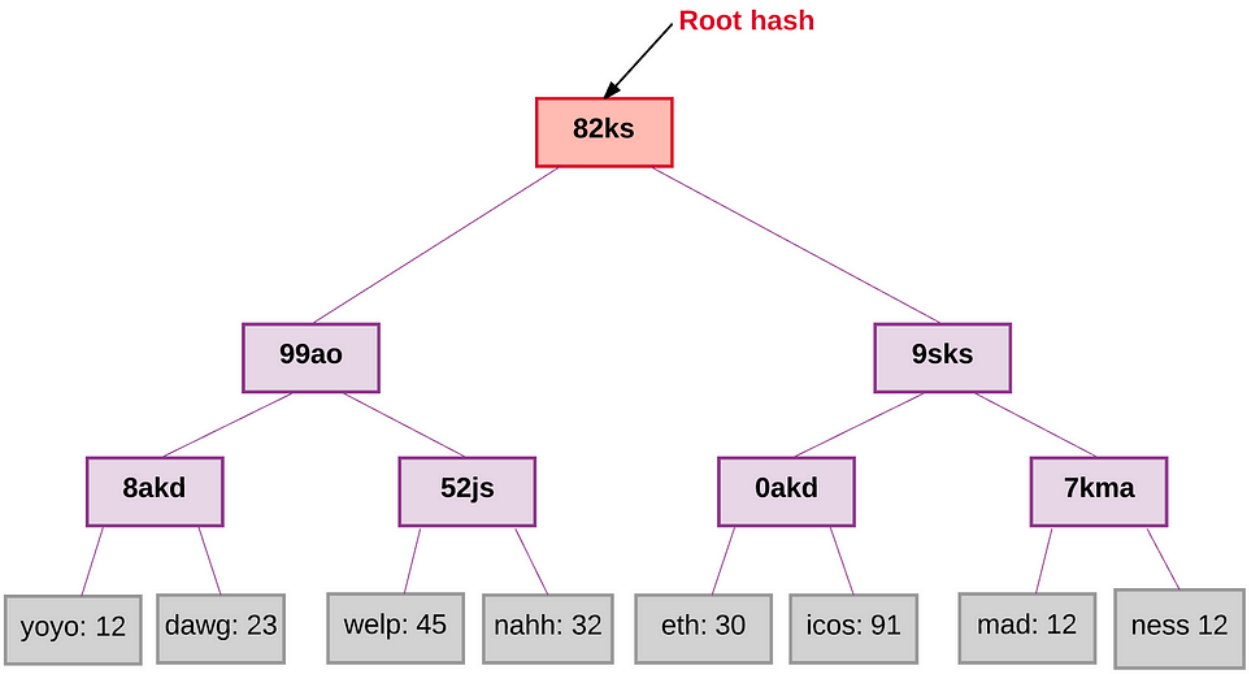
1,世界状态

> 以太坊的全局状态就是由账户地址和账户状态的一个映射组成。这个映射被保存在一个叫做Merkle Patricia树的数据结构中。

Merkle Tree（也被叫做Merkle trie）是一种由一系列节点组成的二叉树，这些节点包括：

1. 在树的最底层的包含了源数据的大量叶子节点
2. 一系列的中间的节点，这些节点是两个子节点的Hash值
3. 一个根节点，同样是两个子节点的Hash值，代表着整棵树

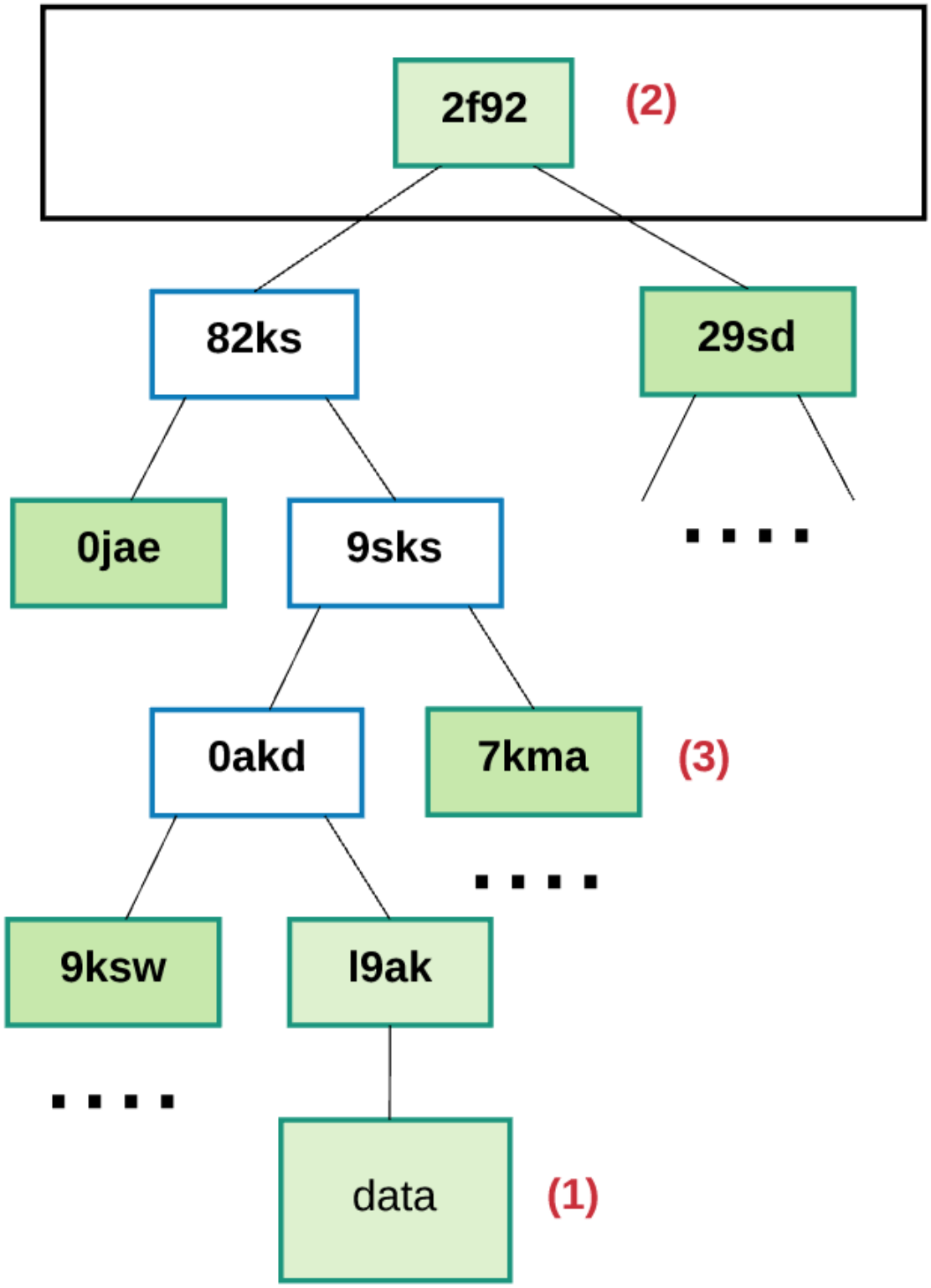
树底的数据是这样产生的：通过将我们想要保存的数据划分成chunks，然后将chunks分成buckets，再然后再获取每个bucket的hash值并一直重复直到最后只剩下一个Hash：根Hash。



Merkle 证明

任何节点想要验证一些数据都可以通过Merkle证明来进行验证，Merkle 证明的组成：

1. 一块需要验证的数据
2. 树的根节点Hash
3. 一个“分支”（从 chunk到根这个路径上所有的hash值）

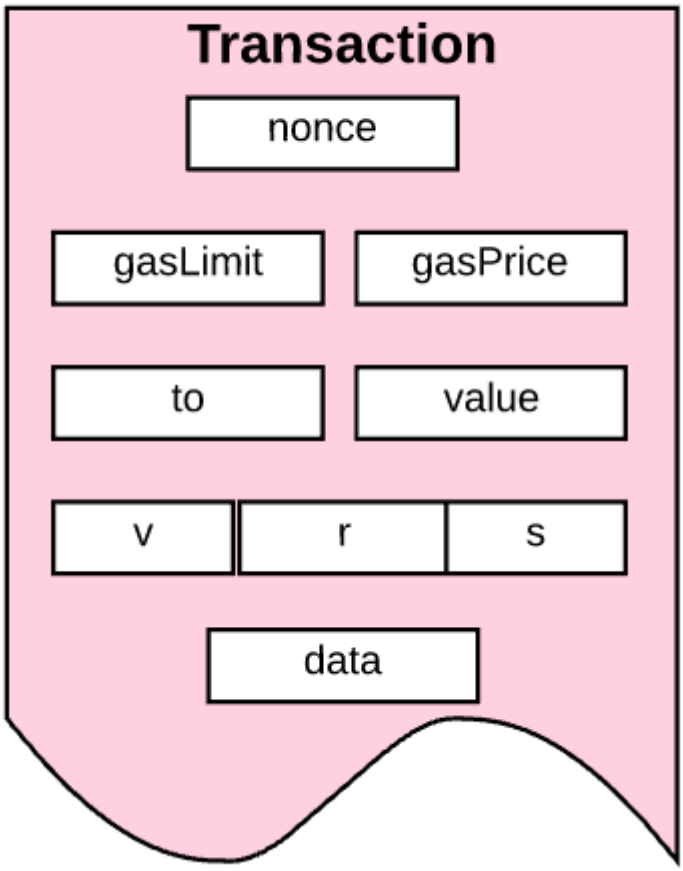


2,交易

> 最基本的概念，一个交易就是被外部拥有账户生成的加密签名的一段指令，序列化，然后提交给区块链。

交易中包含nonce（发送交易数的计数），gasPrince，gasLimit，to（接收方地址），value，(v, r, s) (标记交易发生的签名)

> ECDSA**用于发送方对交易进行签名**



3, Mining proof of work(挖矿工作量证明)

以太坊的工作量证明算法称之为"Ethash"（之前叫做Dagger-Hashimoto）。 算法正式定义为：

$$(m, n) = \mathrm{PoW}(H_{\not n}, H_n, \mathbf{d})$$

区块头中存在如下两项：

1. mixHash：一个Hash值，当与nonce组合时，证明此区块已经执行了足够的计算
2. nonce：一个Hash值，当与mixHash组合时，证明此区块已经执行了足够的计算

PoW函数为每个区块计算一个"种子"。每个"时期"的种子都不一样，每个时期是30,000个区块长度。对于第一时期，种子就是32位0的hash值。对于后续的每个时期，种子就是前一个种子hash值的hash值。使用这个种子，节点可以计算一个伪随机"缓存"。

4,智能合约

> 智能合约是一种以编程方式定义和执行合约条款的自动化计算代码。它们运行在区块链上，具有自动执行、验证和执行功能，无需第三方中介。

智能合约的产生 是为了实现在区块链上自动执行、验证和执行合约的目的。

有了智能合约，你和我就可以在以太坊上写上游戏规则，然后把"钱"（币）打到智能合约的账户上。第二天，智能合约自动抓取官网的天气消息，并将总价值200美元的以太币转移或原路返回给赢家。

一旦智能合约被写入，它就不能以任何方式被编辑或改变。因此，你可以确信，无论合同规定什么，它都会被执行。

比特币实现了交易记录的不可篡改，有了智能合约的以太坊，在此基础上做到了更丰富场景下的去"信任中介"，即不需要第三方来做担保下完成交易。

下面是几个常见的应用示例：

1. **去中心化金融（DeFi）** 以太坊智能合约可以用于创建各种去中心化金融应用（DeFi），如借贷、稳定币、去中心化交易所等。智能合约可以实现自动化的贷款和放款过程，根据预设的条件自动执行各种金融交易。

2. **数字身份认证** 以太坊智能合约可以用于建立和管理数字身份认证系统。每个用户可以通过智能合约创建一个唯一的身份标识，并且可以使用该身份标识进行身份验证和授权访问。

综上，ECDSA在以太坊

综上，ECDSA在以太坊中应用包括如下：

- 账户地址和私钥：以太坊使用ECDSA生成公私钥对来管理账户。每个以太坊账户都有一个唯一的地址，该地址是由用户的公钥通过哈希函数计算得到的。私钥用于对交易进行签名，以证明交易的合法性。

- 交易签名：在以太坊网络中，每笔交易都需要被发送方使用其私钥进行签名。ECDSA签名算法用于验证交易的真实性和完整性，并确保只有合法的私钥拥有者才能修改账户的状态或发送交易。

- 智能合约：以太坊的智能合约也使用ECDSA作为其签名机制。智能合约可以编写和执行具有预定义条件和逻辑的代码。当调用智能合约时，需要提供有效的签名，以验证参与者的身份

和权限，从而确保只有合法的操作才能被执行。

参考文献

How does Ethereum work, anyway?.

ECDSA: Elliptic Curve Signatures

190816-secp256k1-ecdsa-dangers