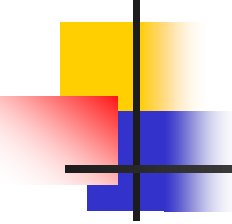


区块链技术与应用



Chapter 10 另类币和加密货币生态系统

苏 明



介绍

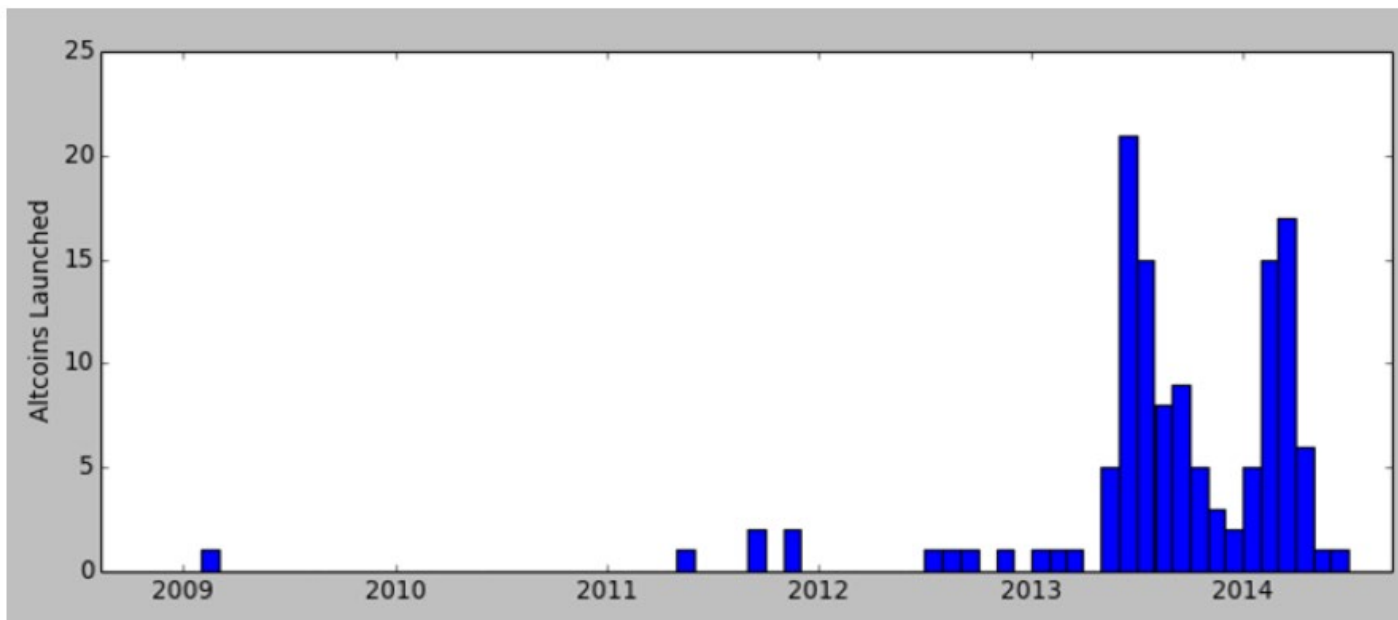
- 另类币的介绍
- 不可分割的交叉链交换
- 以太坊和智能合约



另类币的介绍

- Limitation of Bitcoin (High Latency, Low TPS)
- 可以对脚本语言进行扩充以增加交易种类和安全属性
- 采用与**bitcoin**不同的挖矿方式以及共识算法

另类币的介绍



困难的地方在于如何让别人逐步接受你的货币，吸引开发者、矿工、投资者、商家、客户、支付服务商加入货币的生态圈



另类币的介绍

- 吸引矿工特别重要：没有足够的算力支撑，双重支付和复制修改代码就容易发生
- 让一个社区的人相信该另类币的价值



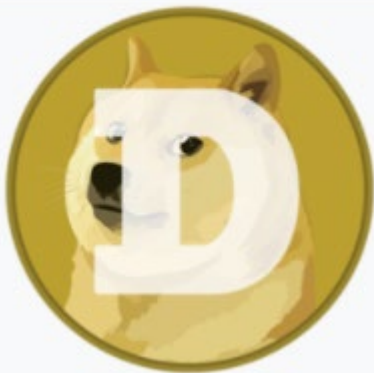
另类币



莱特币（Litecoin）是一种点对点的电子加密货币，也是MIT/X11许可下的一个开源软件项目。莱特币受到了比特币（**BTC**）的启发，并且在技术上具有相同的实现原理，莱特币的创造和转让基于一种开源的加密协议，不受到任何中央机构的管理。莱特币旨在改进比特币，与其相比，莱特币具有三种显著差异。第一，莱特币网路大约**每2.5分钟**（而不是10分钟）就可以处理一个块，因此可以提供更快的交易确认。第二，莱特币网路预期产出**8400**万个莱特币，是比特币网路发行货币量的**四倍**之多。第三，莱特币在其工作量证明算法中使用了由Colin Percival首次提出的script加密算法，这使得相比于比特币，在普通计算机上进行莱特币挖掘更为容易（在**ASIC**矿机诞生之前）。

另类币

Dogecoin



Monero is a privacy-focused cryptocurrency

Zcash



更注重于隐私，以及对交易透明的可控性



另类币



The First Pure Proof of Stake Blockchain Platform



作为一个开放的区块链网络，**Conflux** 期望通过去中心化理念及一系列技术创新为未来商业赋能，实现资产和数据互通互信，创建一个更具价值的网络世界。

Conflux 致力于利用自主研发的**树图高性能共识算法**，构建一个无需准入并拥有极高包容性及延展性的区块链网络。



10.5 不可分割的交叉链互换

Atomic Cross-chain Swap (跨链交易)

- **Alice** want to sell a quantity **a** of **altcoin** to **Bob** in exchange for a quantity **b** of his **bitcoin**



10.5 不可分割的交叉链互换

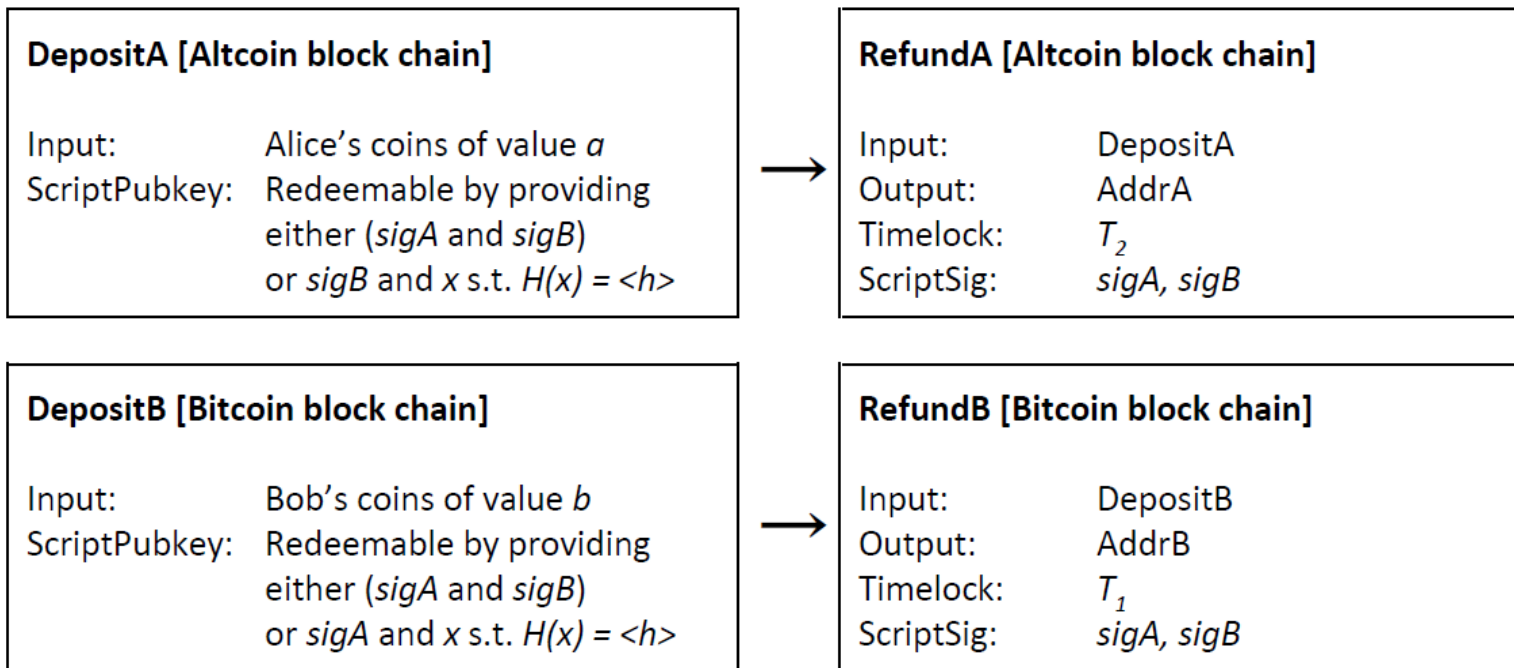
- Complete in atomic fashion, without having to trust each other or relying on an exchange service?
- Clever solution involves cryptographic commitments and time-locked deposits



10.5 不可分割的交叉链互换

1. Alice generates a refundable deposit of a altcoins as follows:
 - 1.1 Alice generates a random string x and computes the hash $h=H(x)$
 - 1.2 Alice generates **DepositA** as shown below, but doesn't publish it yet
 - 1.3 Alice generates **RefundA**, and gets Bob's signature on it
 - 1.4 Once Bob signs **RefundA**, she publishes DepositA (but doesn't publish **RefundA**)
2. Bob generates a refundable deposit of b bitcoins as follows:
 - 2.1 Bob generates **DepositB** as shown below, but doesn't publish it yet
 - 2.2 Bob generates **RefundB**, and gets Alice's signature on it
 - 2.2 Once Alice signs **RefundB**, he publishes **DepositB** (but doesn't publish **RefundB**)
3. Case 1: Alice goes through with the swap
 - 3.1 Alice claims the bitcoins by time T_1 , revealing x to Bob (and everyone) in the process
 - 3.2 Bob claims the altcoins by time T_2Case 2: Alice changes her mind, does not claim the altcoins, does not reveal x to Bob
 - 3.1 Bob claims his altcoin refund at time T_1
 - 3.2 Alice claims her Bitcoin refund at time T_2

10.5 不可分割的交叉链互换



Atomic cross-chain swap protocol




10.5 不可分割的交叉链互换

- 去中心化兑换成为可能
- 跨链交易
- 潜在影响了一系列金融活动，比如交易所...



以太坊和智能合约

- 
- 编程语言（**Solidity**） 图灵完备
 - 支持通用的计算功能
 - 不再局限于数字货币的应用场景

以太坊：一个开源的有智能合约功能的公共区块链平台，**Blockchain 2.0**



以太坊和智能合约

- **智能合约 (smart contract)**：存储在区块链上的程序，由各节点运行，需要运行程序的人支付手续费给节点的矿工或权益人。
- **代币 (tokens)**：智能合约可以创造代币供分布式应用程序使用。分布式应用程序的代币化让用户、投资者以及管理者的利益一致。代币也可以用来进行首次代币发行。
- **权益证明 (proof-of-stake)**：相较于工作量证明更有效率，可节省大量在挖矿时浪费的电脑资源，并避免特殊应用集成电路造成网络中心化。
- **燃料 (gas)**：由交易手续费的概念扩展，在运行各种运算时需计算燃料消耗量，并缴交燃料费，包括发送以太币或者其他代币也被视为一种运算动作。
- **分布式应用程序**：以太坊上的分布式应用程序不会停机，也不能被关掉。
- **叔块 (uncle block)**：将因为速度较慢而未及时被收入母链的较短区块链并入，以提升交易量。使用的是有向无环图的相关技术。



以太坊和智能合约

- 以太坊最重要的技术贡献就是智能合约。
智能合约是存储在区块链上的程序

```
pragma solidity ^0.5.16;
```

```
...
```

```
function check_new_member(address ad)internal{  
    for (uint i = 0; i < users.length; i++){  
        if (users[i] == ad)  
            return;  
    }  
    users.push(ad);  
}
```




以太坊和智能合约

Smart Contract Programming Model

- Anybody can create an Ethereum *contract*, for a small fee, by uploading its program code in a special transaction. This contract is written in *bytecode* and executed by a special Ethereum-specific virtual machine, usually just called **EVM**



以太坊和智能合约

```
contract NameRegistry {
    mapping(bytes32 => address) public registryTable;
    function claimName(bytes32 name) {
        if (msg.value < 10) {
            throw;
        }
        if (registryTable[name] == 0) {
            registryTable[name] = msg.sender;
        }
    }
}
```

A simple Ethereum smart contract implementing a name registry



以太坊和智能合约

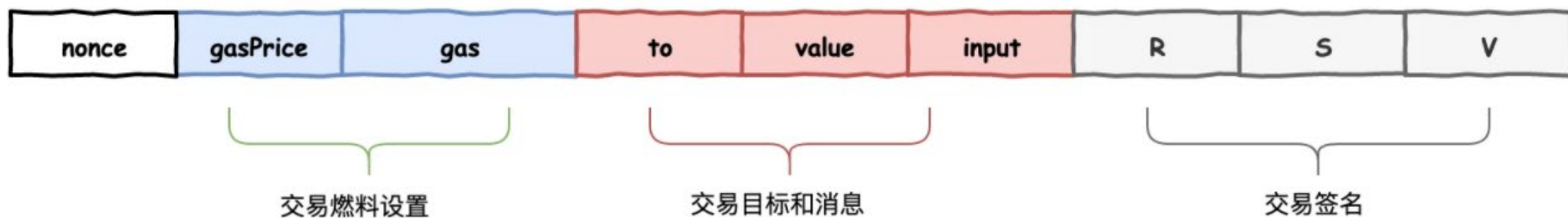
货币单位	
1	以太 (Ether)
10^{-3}	芬尼 (finney)
10^{-6}	萨博 (szabo)
10^{-18}	维 (wei)
货币符号	Ξ ^[33] , ETH



Gas, incentives, and security

以太坊支持循环语句
通过**Gas**来限制计算、存储资源

以太坊和智能合约

■ 交易数据结构

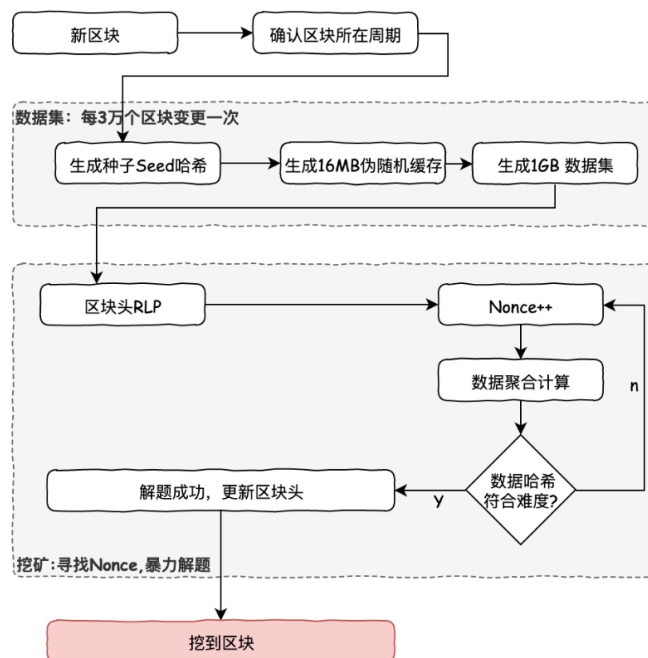


Transaction	
0x8d0650e43846c3bca286c28c5d5228d754a9622426928bc691f7c10bc039d6a0	
Thursday, March 9, 2017 2:01 AM (a month ago, 108 Confirmations)	
Amount	50.00 ETHER
From	 0x511f4DF03B2F6CE6855fc3374f0fA76d6dFD3c
To	 0x375ac88f21515885EFA1E05C756e0b5c0D59f
Fee paid	0.00042 ETHER
Gas used	21,000
Gas price	0.02 ETHER PER MILLION GAS
Block	181 0xe8b2180abc7bc0f839c9337bde807f91b208a4...

以太坊和智能合约

以太坊中存在三种共识算法：

- **Ethash**：是以太坊 1.0 的工作量证明算法
- PoA：权威证明算法，服务于测试网、私有链、联盟链等。
- PoS：将在以太坊2.0中需要实现的一套股权证明算法





以太坊和智能合约

Solidity

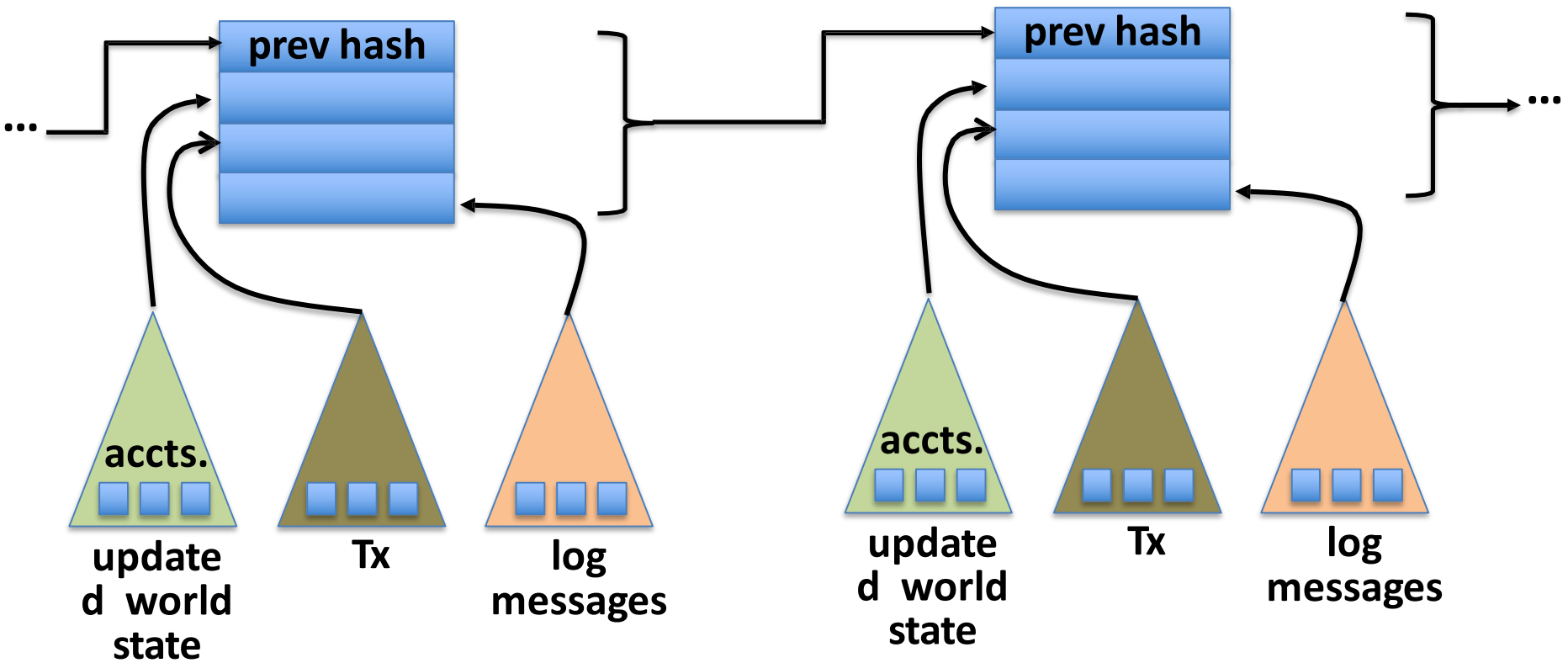
- 一种静态类型的编程语言，用于开发在EVM (the runtime environment for smart contracts in Ethereum)上运行的智能合约。Solidity被编译为可在EVM上运行的字节码
- 开发平台
 - Remix, Solidity官方IDE
 - Microsoft Visual Studio



Recap: Transactions

- **To:** 32-byte address (0 → create new account)
- **From:** 32-byte address
- **Value:** # Wei being sent with Tx
- **Tx fees** (EIP 1559): `gasLimit`, `maxFee`, `maxPriorityFee`
- **data:** what contract function to call & arguments
 - if `To = 0`: create new contract `code = (init, body)`
- **[signature]:** if Tx initiated by an owned account

The Ethereum blockchain: abstractly





EVM mechanics: execution environment

Write code in Solidity (or another front-end language)

⇒ compile to EVM *bytecode*

**(recent projects use WASM or BPF
bytecode)**

**⇒ miners use the EVM to execute contract
bytecode in response to a Tx**



Every EVM instruction costs gas

SSTORE **addr** (32 bytes), **value** (32 bytes)

- **zero → non-zero:** **20,000 gas**
- **non-zero → non-zero:** **5,000 gas**
- **non-zero → zero:** **15,000 gas refund**

Refund is given for reducing size of blockchain state

SELFDESTRUCT **addr:** kill current contract. **24,000 gas refund**

CREATE : **32,000 gas**

CALL **gas, addr, value, args**



Gas calculation

Why charge gas?

- Tx fees (gas) **prevents submitting Tx that runs for many steps.**
- During high load: miners choose Tx from the mempool that *maximize their income.*

Old EVM: (prior to EIP1559, live on 8/2021)

- Every Tx contains a gasPrice ``bid'' (gas → Wei conversion price)
- Miners choose Tx with highest gasPrice ($\max \sum(\text{gasPrice} \times \text{gasLimit})$)



Gas calculation: EIP1559

Every block has a “**baseFee**”:
the minimum gasPrice for all Tx in the block

baseFee is computed from total gas in earlier blocks:

- earlier blocks at gas limit (30M gas) \Rightarrow base fee goes up 12.5%
- earlier blocks empty \Rightarrow base fee decreases by 12.5%

} interpolate
in between

If earlier blocks at “target size” (15M gas) \Rightarrow base fee does not change



Gas calculation

EIP1559 Tx specifies three parameters:

- **gasLimit**: *max total gas allowed for Tx*
- **maxFee**: maximum allowed gas price (max gas → Wei conversion)
- **maxPriorityFee**: additional “tip” to be paid to miner

Computed *gasPrice* bid:

$$\text{gasPrice} \leftarrow \min(\text{maxFee}, \text{baseFee} + \text{maxPriorityFee})$$

$$\text{Max Tx fee: } \text{gasLimit} \times \text{gasPrice}$$



Gas calculation (simplified)

- (1) if $\text{gasPrice} < \text{baseFee}$: abort
- (2) If $\text{gasLimit} \times \text{gasPrice} > \text{msg.sender.balance}$: abort

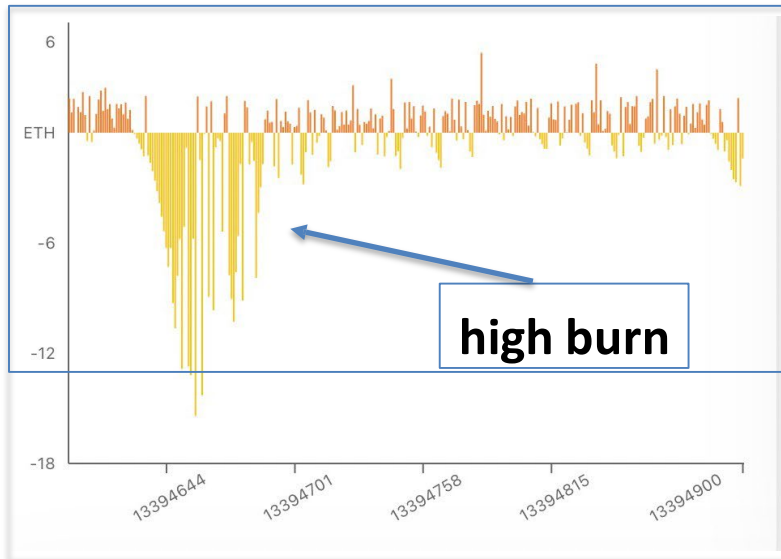
- (3) deduct $\text{gasLimit} \times \text{gasPrice}$ from $\text{msg.sender.balance}$
- (4) set $\text{gasLeft} \leftarrow \text{gasLimit}$
- (5) execute Tx: deduct gas from gasLeft for each instruction
if at end ($\text{gasLeft} < 0$): Tx is invalid (miner keeps $\text{gasLimit} \times \text{gasPrice}$)
- (6) refund $|\text{gasLeft}| \times \text{gasPrice}$ to $\text{msg.sender.balance}$

- (7) $\text{gasUsed} \leftarrow \text{gasLimit} - \text{gasLeft}$
 - (7a) BURN $\text{gasUsed} \times \text{baseFee}$
 - (7b) send $\text{gasUsed} \times (\text{gasPrice} - \text{baseFee})$ to miner

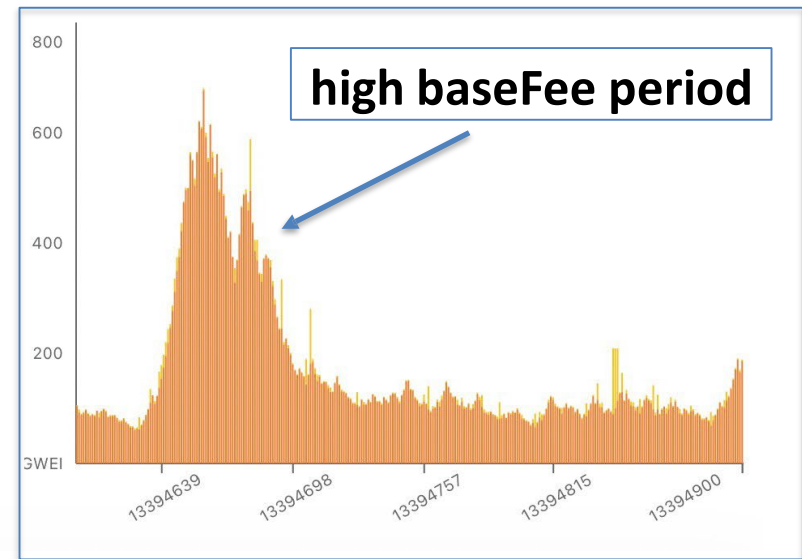


Burn results in practice

block reward (2ETH) minus
Total baseFee burned in block



baseFee for block (Wei)



... sometimes burn exceeds block rewards \Rightarrow ETH deflation

Let's look at the Ethereum blockchain

etherscan.io:

Latest Blocks		
Bk	13405036 39 secs ago	Miner F2Pool Old 125 txns in 19 secs
Bk	13405035 58 secs ago	Miner Ethermine 188 txns in 2 secs
Bk	13405034 1 min ago	Miner 2Miners: PPLNS 85 txns in 3 secs
Bk	13405033 1 min ago	Miner F2Pool Old 269 txns in 61 secs

From/to
address

Tx value

From 0x8875f31e963d42f2f18...	0 Eth
To 0xdac17f958d2ee523a2...	
From 0xcac29d3938e3a8330e...	0 Eth
To 0x7be8076f4ea4a4ad08...	
From 0xe37b696deffbc7e2639...	0.09378 Eth
To 0xd9e1ce17f2641f24ae8...	
From 0x578b076f33c021ca8ec...	0.18 Eth
To 0x7be8076f4ea4a4ad08...	



Contracts cannot keep secrets!

Contract 0x1a2a1c938ce3ec39b6d47113c7955baa9dd454f2
(Axie Infinity: Ronin Bridge)

Anyone can read contract
state in storage array
⇒ never store secret keys
in contract!

etherscan.io

Code Read Contract (storage) Write Contract (see API)

Read Contract Information

1. admin

[0x23d4817717fc407ee8266dc45f4f8a1ccc5338fa](#) address

5. paused

False *bool*

⋮

Solidity variables
stored in S[] array



Contract structure

```
contract IERC20Token {  
    function transfer(address _to, uint256 _value) external returns  
        (bool);  
    function totalSupply() external view returns (uint256);  
    ...  
}  
  
contract ERC20Token is IERC20Token {    //  
    inheritance address owner;  
    constructor() public { owner = msg.sender; }  
    function transfer(address _to, uint256 _value) external returns (bool) {  
        ... implentation ...  
    }  
}
```



Value types

- uint256
- address (bytes32)
 - `_address.balance`, `_address.send(value)`, `_address.transfer(value)`
 - `call`: send Tx to another contract

```
bool success = _address.call(data).value(amount).gas(amount);
```
 - `delegatecall`: load code from another contract into current context
- bytes32
- bool



Reference types

- **structs**
- **arrays**
- **bytes**
- **strings**
- **mappings:**
 - **Declaration:** **mapping (address => unit256)**
 - **Assignment:** **balances; balances[addr] = value;**

```
struct Person {  
    uint128 age;  
    uint128  
    balance;  
    address addr;  
}
```

```
Person[10] public people;
```



Globally available variables

- block: .blockhash, .coinbase, .difficulty, .gaslimit, .number, .timestamp
- gasLeft()
- msg: .data, .sender, .sig, .value
- tx: .gasprice, .origin
- abi: encode, encodePacked, encodeWithSelector, encodeWithSignature
- Keccak256(), sha256(), sha3()
- require, assert e.g.: `require(msg.value > 100, "insufficient funds sent")`

A → B → C → D:
at D: msg.sender == C
 tx.origin == A



Function visibilities

- ***external***: function can only be called from outside contract.
Arguments read from calldata
- ***public***: function can be called externally and internally.
Arguments copied from calldata to memory
- ***private***: only visible inside contract
- ***internal***: only visible in this contract and contracts deriving from it
- ***view***: only read storage (no writes to storage)
- ***pure***: does not touch storage

```
function f(uint a) private pure returns (uint b) { return a + 1; }
```

Reference Books



蔡亮，李启雷，梁秀波 著

区块链技术进阶与实战
人民邮电出版社 **2018**



林冠宏 著

清华大学出版社 **2019**