智能合约安全的新进展

智能合约

- 智能合约是 1990s 年代由尼克萨博提出的理念
- 自我执行，自我验证
- 不需要人为干预

Smart=灵活

Smart！=人工智能

以太坊

- 为智能合约提供可信环境
- a platform for running decentralized apps
- 在区块链上的虚拟机 EVM
  - EVM 是图灵完备的
  - 合约（包括代码，数据）以字节码的形式存储在区块链上

一个发行代币的例子


```
pragma solidity ^0.4.0;

contract Coin {
    address public minter;
    mapping (address => uint) public balances;
    event Sent(address from, address to, uint amount);
    function Coin() {
        minter = msg.sender;
    }
    function mint(address receiver, uint amount) {
        if (msg.sender != minter) return;
        balances[receiver] += amount;
    }
    function send(address receiver, uint amount) {
        if (balances[msg.sender] < amount) return;
        balances[msg.sender] -= amount;
        balances[receiver] += amount;
        Sent(msg.sender, receiver, amount);
    }
  function balances(address _account) returns (uint balance) {
    return balances[_account];
    }
}
```

- Coin 构造函数设定了minter的所有权
- Minter 分配代币数量
- 用户提取代币
- 用户查询代币余额

ICO

- = Initial Coin Offering
- 加密货币的众筹
- 因为对智能合约友好，绝大部分的ico 是在eth上的
- ICO 通常发行代币（Token）
- 给项目开发组eth 来交换未来收益或项目使用权。

```
pragma solidity ^0.4.8;
import "./Token.sol";// 从标准token 抽象类中继承
contract StandardToken is Token {
    function transfer(address _to, uint256 _value) returns (bool success) {
        if (balances[msg.sender] >= _value && _value > 0) {
            balances[msg.sender] -= _value;
            balances[_to] += _value;
            Transfer(msg.sender, _to, _value);
            return true;
        } else { return false; }
    }
    function transferFrom(address _from, address _to, uint256 _value) returns (bool
success) {
        if (balances[_from] >= _value && allowed[_from][msg.sender] >= _value && _value >
0) {
            balances[_to] += _value;
            balances[_from] -= _value;
            allowed[_from][msg.sender] -= _value;
            Transfer(_from, _to, _value);
            return true;
        } else { return false; }
    }
    function balanceOf(address _owner) constant returns (uint256 balance) {
        return balances[_owner];
    }
    function approve(address _spender, uint256 _value) returns (bool success) {
        allowed[msg.sender][_spender] = _value;
        Approval(msg.sender, _spender, _value);
        return true;
    }
    function allowance(address _owner, address _spender) constant returns (uint256
remaining) {
      return allowed[_owner][_spender];
    }
    mapping (address => uint256) balances;
    mapping (address => mapping (address => uint256)) allowed;
}
```

火币 huobi.com

ERC20 标准

- 动机：交易所友好
- 支持转账和查询余额等
- 大部分ICO 代币发行都支持 ERC20
- 成熟的代币会被交易所引进
- 更好的经济激励 促进了Dapp 的开发

	NAME	ROI SINCE ICO	ETH ROI SINCE ICO
	Swarm City Buy Instantly	+5867.76%	+3538.13%
augur	Augur Buy Instantly	+5247.60%	+31170.63%
	Golem Buy Instantly	+5019.10%	+3666.83%
	Antshares	+4953.87%	+3140.21%
Lisk	Lisk Buy Instantly	+4026.96%	+35079.46%
	Komodo	+3900.65%	+3202.68%
	Bitcrystals Buy Instantly	+2909.70%	+131236.67%
	DigixDAO Buy Instantly	+2801.97%	+5017.01%
	Etheroll	+2781.67%	+2898.55%

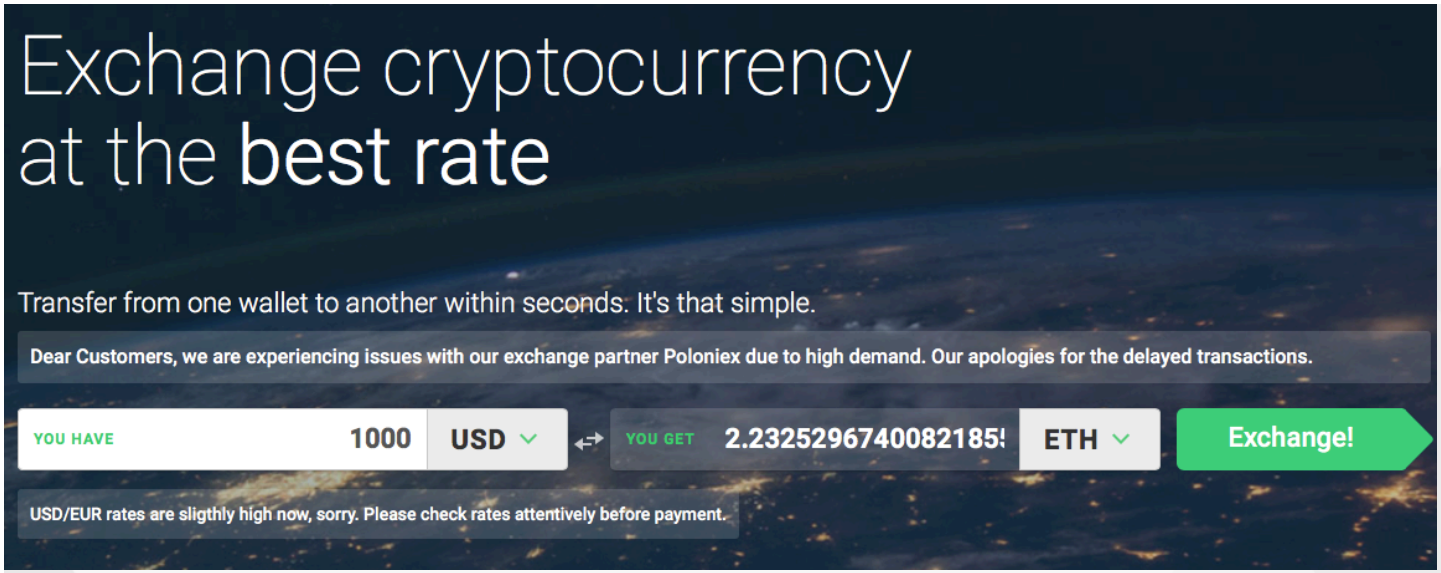
AUGUR

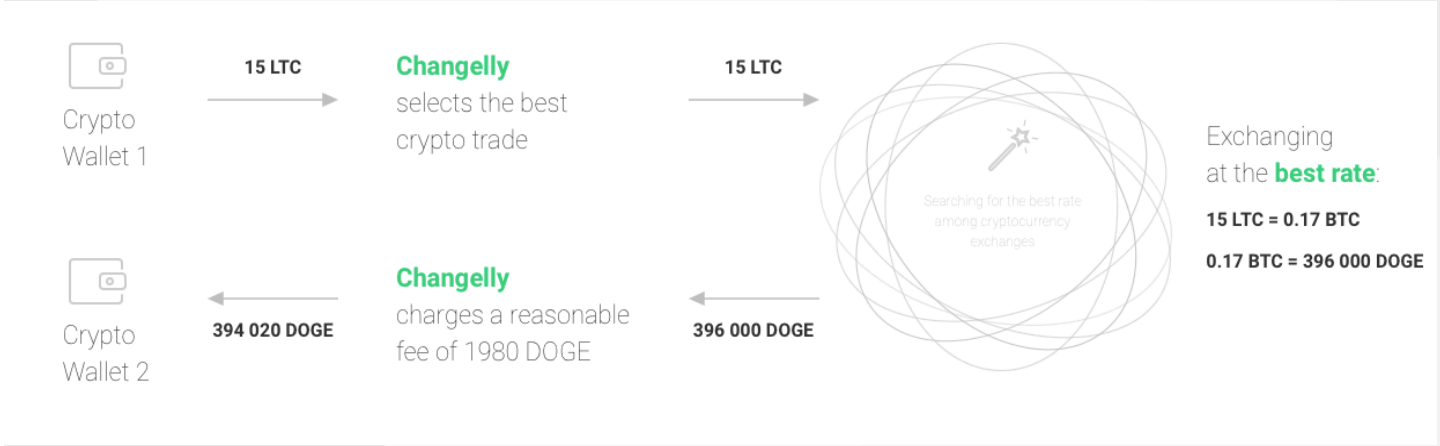
- 去中心化的预测市场
- 73位学生聚集瓶子中糖豆的数量。尽管大部分学生的估计很不准确，但是他们估计的平均值是1151，与真实数值1116只有3%的误差。
- 用利益激励匿名群体给出预测
- global financial markets on anything

Golem

- 空闲算力变现
- 买卖双方交易算力和代币
- 去中心化匹配

- 智能合约通过代币沟通了供需信息，提高了效率。





火币 huobi.com

Changelly

- 去中心化的兑换中心
- 便宜，高效，可信任

金融：
智能合约降低了信任成本

智能合约的安全事故

the DAO 代码逻辑缺陷

- 分布式自治组织
- ICO 后根据金额分配投票权，集体投资其他项目
- The DAO的智能合约编写漏洞： “split” 函数
- 黑客盗取了360多万个以太币(ETH)的资金，占到The DAO众筹额的30%以上，而且理论上资金可被全部偷走。
- ”too big to fail” =>硬分叉 ETH & ETC
- 项目报废。

The casino with a public RNG seed

```javascript
var seed =
web3.toBigNumber(web3.eth.getStorageAt("0x5fe5b7546d1628f7348b023
a0393de1fc825a4fd", 1);

var num = (Math.pow((~~(seed * 3 + 1) / 2) % 10), 9) +
eth.getBlock("latest").number + eth.getBlock("latest").difficulty
+ eth.getBlock("latest").timestamp +
eth.getBlock("latest").gasLimit) % 37
num = Math.floor(num);
```

javascript

1. 使用公开的随机数种子
2. 矿工可以预先得知随机数
3. 更漂亮的做法是使用合约攻击合约，
   “攻击合约”先计算随机数，
   然后自动化的参与赌博

自己做轮子写随机数生成器是很危险的。

Ponzi

```
while (balance > persons[payoutCursor_Id_].deposit / 100 * 115) {
  uint MultipliedPayout = persons[payoutCursor_Id_].deposit / 100 * 115;
  persons[payoutCursor_Id].etherAddress.send(MultipliedPayout);

  balance -= MultipliedPayout;
  payoutCursor_Id_++;
}
```

拼写错误！
导致可以获得1.15倍回报，直至抽光所有资金。
项目终止。
代码已经不能通过最新版本的编译器和优化器。

King of the Ether Throne

-- 交易细节设置失误

- 使用send 函数默认设置gas 上限为2300
- 普通账户间交易需要2100
- 合约账户间消耗掉gas 超过2300 会转账失败

- 项目上线第二天被紧急停止，之后手工refund

- 类似的事情发生在外国交易所转账给云币充值时，外国交易所使用2100 gas上限，云币网使用合约地址收款。

漏洞	详情	涉及项目
时间戳依赖	矿工预先得知	
随机数依赖	矿工预先得知	The casino with a public RNG seed
代码逻辑出错	代码拼写错误，逻辑漏洞	FirePonzi
所有权失控	构造函数拼写错误，导致任何人可以获得控制权	Rubixi 和 MakerDAO
外部函数调用	send，callvalue，split 函数	the DAO
交易gas 限制	send 函数默认2300 不支持支付到合约地址	king of the Ether Throne
上溢 / 下溢	20种溢出情况	
Dos with Throw	低成本阻塞，比如投标的合约	
Dos with Block Gas Limit	for 循环 内的转账	Governmental
call Depth	调用超过次数限制，EIP150已经修复	

火币 huobi.com

举个例子

剪刀石头布的安全进化过程

```
def player_input(choice):
  if num_players < 2 and msg.value == 1000:
    reward += msg.value
    player[num_players].address = msg.sender
    player[num_players].choice = choice
    num_players = num_players + 1
    return(0)
  else:
    return(-1)
def finalize():
  p0 = player[0].choice
  p1 = player[1].choice
  # If player 0 wins
  if check_winner[p0][p1] == 0:
      send(0,player[0].address, reward)
  return(0)
  # If player 1 wins
  elif check_winner[p0][p1] == 1:
      send(0,player[1].address, reward)
  return(1)
  # If no one wins else:
      send(0,player[0].address, reward/2)
      send(0,player[1].address, reward/2)
      return(2)
```


- 版本1
- 实现逻辑
  - 1. 双方给钱
  - 2. 猜拳判定
  - 3. 输出结果
- 但是安全漏洞很多

```
def player_input(choice):
  if num_players < 2 and msg.value == 1000:
    reward += msg.value
    player[num_players].address = msg.sender
    player[num_players].choice = choice
    num_players = num_players + 1
    return(0)
  else:
    return(-1)
def finalize():
  p0 = player[0].choice
  p1 = player[1].choice
  # If player 0 wins
  if check_winner[p0][p1] == 0:
      send(0,player[0].address, reward)
  return(0)
  # If player 1 wins
  elif check_winner[p0][p1] == 1:
      send(0,player[1].address, reward)
  return(1)
  # If no one wins else:
      send(0,player[0].address, reward/2)
      send(0,player[1].address, reward/2)
      return(2)
```


- 版本1
- 实现逻辑
  - 1. 双方给钱
  - 2. 猜拳判定
  - 3. 输出结果
- 但是安全漏洞很多

```
def player_input(choice):
  if num_players < 2 and msg.value == 1000:
    reward += msg.value
    player[num_players].address = msg.sender
    player[num_players].choice = choice
    num_players = num_players + 1
    return(0)
  else:
    return(-1)
def finalize():
  p0 = player[0].choice
  p1 = player[1].choice
  # If player 0 wins
  if check_winner[p0][p1] == 0:
      send(0,player[0].address, reward)
  return(0)
  # If player 1 wins
  elif check_winner[p0][p1] == 1:
      send(0,player[1].address, reward)
  return(1)
  # If no one wins else:
      send(0,player[0].address, reward/2)
      send(0,player[1].address, reward/2)
      return(2)
```


- 版本1 的问题

1. 第三个进入的人的钱就消失了
2. 可以窥测对手的input。（input是明文传输，并且记录在区块链中）
3. Reward 不是正好1000 怎么办

```
def player_input(commitment):
  if num_players < 2 and msg.value >= 1000:
    reward += msg.value
    player[num_players].address = msg.sender
    player[num_players].commit = commitment
    num_players = num_players + 1
    if msg.value - 1000 > 0:
      send(msg.sender, msg.value-1000)#多余的钱退回去
    return(0)
  else:
    send(msg.sender, msg.value)
    return(-1)
def open(choice, nonce):# 公开结果
    if not num_players == 2: return(-1) # Determine which
player is opening
    if msg.sender == player[0].address:
        player_num = 0
    elif msg.sender == player[1].address:
        player_num = 1
    else:
        return(-1)
    # Check the commitment is not yet opened
    if sha3([msg.sender, choice, nonce], items=3) ==
player[player_num].commit and not
    \ player[player_num].has_revealed:
        # Store opened value in plaintext
        player[player_num].choice = choice
        player[player_num].has_revealed = 1 return(0)
    else:
        return(-1)
    # code omitted:  判定过程。
```


- 版本2的解决了大部分问题
- 业务逻辑修复，退回资金
- 通过加密算法保证隐私性
- 杜绝了第三方的攻击
- 但是 有可能被对手劫持

```
def check():
    if self.test_callstack() != 1: return(-1)
    #Check to make sure at least 10 blocks have been given
for both players to reveal their play.
    if block.number - self.timer_start < 10: return(-2)
    #check to see if both players have revealed answer
    if self.player[0].has_revealed and
self.player[1].has_revealed:
        p0_choice = self.player[0].choice
        p1_choice = self.player[1].choice
        #If player 0 wins
        if self.check_winner[p0_choice][p1_choice] == 0:
            send(0,self.player[0].address, self.reward)
            return(0)
        #If player 1 wins
        elif self.check_winner[p0_choice][p1_choice] == 1:
            send(0,self.player[1].address, self.reward)
            return(1)
        #If no one wins
        else:
            send(0,self.player[0].address, self.reward/2)
            send(0,self.player[1].address, self.reward/2)
            return(2)

    #if p1 revealed but p2 did not, send money to p1
    elif self.player[0].has_revealed and not
self.player[1].has_revealed:
        send(0,self.player[0].address, self.reward)
        return(0)
```


- 版本3是最终安全版本

- 要求截止日期之前解密
- 确保公平性

- 合约流程：
- A，B提交加密 input 到合约
- A，B 分别在ddl前解密
- 计算结果。

安全的编码策略

任何代码都有可能是有bug。做好breaker
功能解耦：
- 正交分解，缩小代码工作量
- 逻辑简单
- 不重复开发轮子
- 只对区中心化的部分使用区块链技术

保持更新(代码，文档，bug fixed，服务)
注意区块链技术的性质：
- 交易公开
- 私有信息的界限
- 交易Gas cost！ 和区块gas limit
- 矿工也是潜在的攻击者

安全的开发流程策略

- 单元测试
- 私有链测试
- 测试链测试

- 安全服务化：
  - 更新代码
  - 更新安全工具
  - 容灾

安全相关的开发工具

- **Oyente**：开发中，可以检测部分已知的安全问题 时间戳依赖，随机数生成等
- **Zeppelin**：安全代码组件。方便生成对ICO 的测试，已经和**Golem**，**Aragon** 等 项目合作，可以配合**truffle** 使用。
- **SolCover**：覆盖测试，生成html 格式的覆盖测试报告
- **Solgraph**：可视化函数调用。高亮风险代码
- **Solint**：一致性

Oyente 评估合约

```
root@4bba7337b1ff:/oyente# python oyente.py -s test.sol
WARNING:root:You are using evm version 1.6.5. The suppor
WARNING:root:You are using solc version 0.4.12, The supp
INFO:root:Contract test.sol:Coin:
INFO:symExec:Running, please wait...
INFO:symExec:    ============ Results ===========
INFO:symExec:      CallStack Attack:      False
INFO:symExec:      Concurrency Bug:       False
INFO:symExec:      Time Dependency:       False
INFO:symExec:      Reentrancy bug exists: False
INFO:symExec:    ====== Analysis Completed ======
root@4bba7337b1ff:/oyente#
```

检测四类错误，更多功能开发中

```
pragma solidity ^0.4.4;
contract testTime {
  function generateRand() returns (uint) {
    uint now_time = now + 1;
    if (now % 2 == 0) {
      // the now can be manipulated by the miner
      return now;
    }
    return 0;
  }
}
```



```
root@58c712aa91db:/oyente# python oyente.py -s timeTest.s
ol
WARNING:root:You are using evm version 1.6.5. The su port
ed version is 1.6.1
WARNING:root:You are using solc version 0.4.12, The suppo
rted version is 0.4.10
INFO:root:Contract timeTest.sol:testTime:
INFO:symExec:Running, please wait...
INFO:symExec:    ========== Results ==========
INFO:symExec:      CallStack Attack:      False
INFO:symExec:      Concurrency Bug:       False
INFO:symExec:      Time Dependency:       True
INFO:symExec:      Reentrancy bug exists: False
INFO:symExec:    ====== Analysis Completed ======
root@58c712aa91db:/oyente#
```

```
function Puzzle(){
  owner = msg.sender;
  reward = msg.value;
  locked = false;
  diff = bytes32(11111); //pre-defined difficulty
}
function(){ //main code, runs at every invocation
  if (msg.sender == owner){ //update reward
    if (locked)
      throw;
    owner.send(reward);
    reward = msg.value;
  }else
    if (msg.data.length > 0){ //submit a solution
      if (locked) throw;
      if (sha256(msg.data) < diff){
        msg.sender.send(reward); //send reward
        solution = msg.data;
        locked = true;
      }
    }
```



```
========== Results ==========
   CallStack Attack:      False
=>>>>>> New PC: []
Reentrancy_bug? True
=>>>>>> New PC: []
Reentrancy_bug? True
=>>>>>> New PC: []
Reentrancy_bug? True
=>>>>>> New PC: []
Reentrancy_bug? True
=>>>>>> New PC: []
Reentrancy_bug? True
=>>>>>> New PC: []
INFO:analysis:Reentrancy_bug? True
INFO:symExec:    Concurrency Bug:       False
INFO:symExec:    Time Dependency:       False
INFO:symExec:    Reentrancy bug exists: True
INFO:symExec:  ====== Analysis Completed ======
```

Zeppelin+truffle 进行ICO 测试

bash

```
[~]$ npm install zeppelin-solidity
Installing [============] 100%
> zeppelin-solidity@1.0.0 install
> scripts/install.sh
[~]$ vim ExampleToken.sol
[~]$ truffle console
> var myToken = ExampleToken.deployed();
> myToken.totalSupply()
10000
> myToken.transfer(...)
true
```

vim ExampleToken.sol

```
pragma solidity ^0.4.4;
import "./StandardToken.sol";

contract ExampleToken is StandardToken {
 string public name = "ExampleToken";
 string public symbol = "EGT";
 uint public decimals = 18;
 uint public INITIAL_SUPPLY = 10000;

 function ExampleToken() {
  totalSupply = INITIAL_SUPPLY;
  balances[msg.sender] = INITIAL_SUPPLY;
 }
}
```

Zeppelin 提供了更安全的组件

包括：

- 安全数值运算： +-* / max， min （防止溢出）
- 所有权管理 Ownable ,Shareable 等
- 支付，余额，限制每日交易数量等
- 多重签名
- Token 抽象类和安全实践

Zeppelin 是相对成熟的轮子，都是sol 代码
通过导入和继承使用。

```
pragma solidity ^0.4.11;
/**
 * Math operations with safety checks
 */
library SafeMath {
  function mul(uint a, uint b) internal returns (uint) {
    uint c = a * b;
    assert(a == 0 || c / a == b);
    return c;
  }

  function div(uint a, uint b) internal returns (uint) {
    // assert(b > 0); // Solidity automatically throws when dividing by 0
    uint c = a / b;
    // assert(a == b * c + a % b); // There is no case in which this doesn't hold
    return c;
  }

  function sub(uint a, uint b) internal returns (uint) {
    assert(b <= a);
    return a - b;
  }

  function add(uint a, uint b) internal returns (uint) {
    uint c = a + b;
    assert(c >= a);
    return c;
  }
```

火币 huobi.com

比如防溢出的运算

Token 抽象类

- Zeppelin 提供了模版和对应代币工厂
- Token 根据业务逻辑，也需要增加其他特性，同时也易出漏洞。

比如

  - 无上限的mintable
  - 中止
  - 多地址共享所有权
  - 单笔转账有上限的token
  - 无上限众筹

- Zeppelin 提供了对应安全实践

```
import './StandardToken.sol';
import '../ownership/Ownable.sol';


contract MintableToken is StandardToken, Ownable {
  event Mint(address indexed to, uint256 amount);
  event MintFinished();

  bool public mintingFinished = false;


  modifier canMint() {
    if(mintingFinished) throw;
    _;
  }

  function mint(address _to, uint256 _amount) onlyOwner canMint returns (bool) {
    totalSupply = totalSupply.add(_amount);
    balances[_to] = balances[_to].add(_amount);
    Mint(_to, _amount);
    return true;
  }

  function finishMinting() onlyOwner returns (bool) {
    mintingFinished = true;
    MintFinished();
    return true;
  }
}
```

火币 huobi.com

1. 继承标准代币（ERC 20）
2. Ownable中实现的 onlyOwner 保证了 minter 只有一个

3. 根据业务逻辑，实现了mintable 的特性，可以无限发币

```
import "./StandardToken.sol";

contract CrowdsaleToken is StandardToken {
  string public constant name = "CrowdsaleToken";
  string public constant symbol = "CRW";
  uint256 public constant decimals = 18;
  address public constant multisig = 0x0;


  // 1 ether = 500 example tokens
  uint256 public constant PRICE = 500;

  function () payable {
    createTokens(msg.sender);
  }

  function createTokens(address recipient) payable {
    if (msg.value == 0) {
      throw;
    }

    uint256 tokens = msg.value.mul(getPrice());
    totalSupply = totalSupply.add(tokens);

    balances[recipient] = balances[recipient].add(tokens);

    if (!multisig.send(msg.value)) {
      throw;
    }
  }

  function getPrice() constant returns (uint256 result) {
    return PRICE;
  }
}
```

火币 huobi.com

1.发行众筹代币
2.设定 代币代码，数量，发行量
3.允许ETH购买代币
4.设定价格

发展

	Web 2.0	Web 3.0 (dApps)	Status
Scalable computation	Amazon EC2	?	Not started
File storage	Amazon S3	?	Not started
External data	3rd party APIs	?	Not started
Monetization	Ads, selling goods	?	Not started
Payments	Credit Cards, Paypal	Bitcoin	Ready

2014 VS 2017

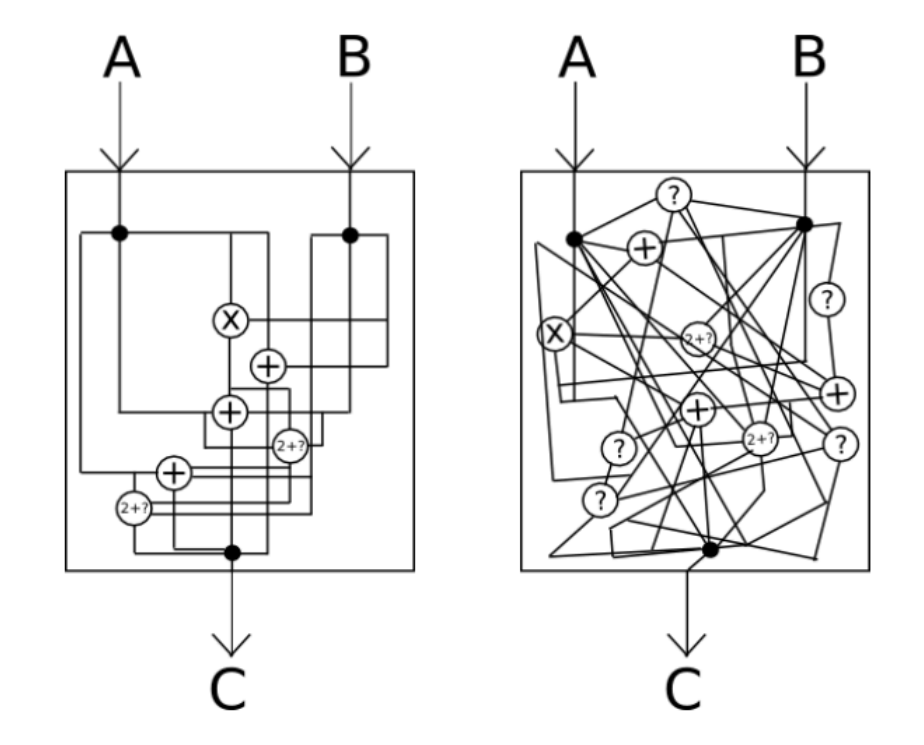
	Web 2.0	Web 3.0 (dApps)	Status
Scalable computation	Amazon EC2	Ethereum, Truebit	In progress
File storage	Amazon S3	IPFS/Filecoin, Storj	In progress
External data	3rd party APIs	Oracles (Augur)	In progress
Monetization	Ads, selling goods	Token model	Ready
Payments	Credit Cards, Paypal	Ethereum, Bitcoin, state channels, 0x	Ready

安全的新议题

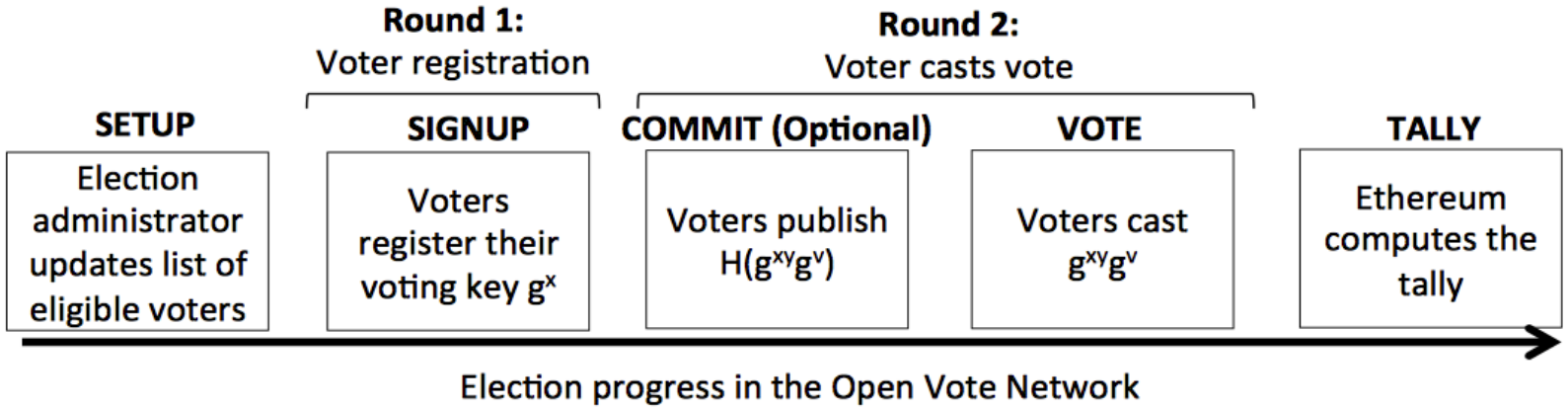
- 匿名性改善
- 随机数生成
- 形式化验证

隐私保护-混淆

- 等价命题<=>比特币 coinJoin
- 区块链知道input 和output
  但是不知道合约内容。
- 想要匿名投票，但是别人知道我
  的地址
- 在 5 个人不告知各自薪水的情况下，怎么求平均工资



匿名投票的解决方案



Round1 和Round 2 都使用了零知识证明技术来进行验证
这样可以最大程度的保护投票者隐私

随机数生成

Randao 项目：

获取sha3(s)-> 获取s->生成随机数 -> 分配利益

比如 $r = sha3(sn + sha3(sn - 1 + \ldots(sha3(s2 + s1))))$

1.随机数生成交给参与者

2.设置窗口期，避免矿工拒绝

随机数生成

- Randao++
- 使用 sequential PoW， 不再依赖“获取s”

$$r = sha3^{1000000000}(x[0] + block[N].hash)$$
$$XOR$$
$$sha3^{1000000000}(x[1] + block[N].hash)...$$

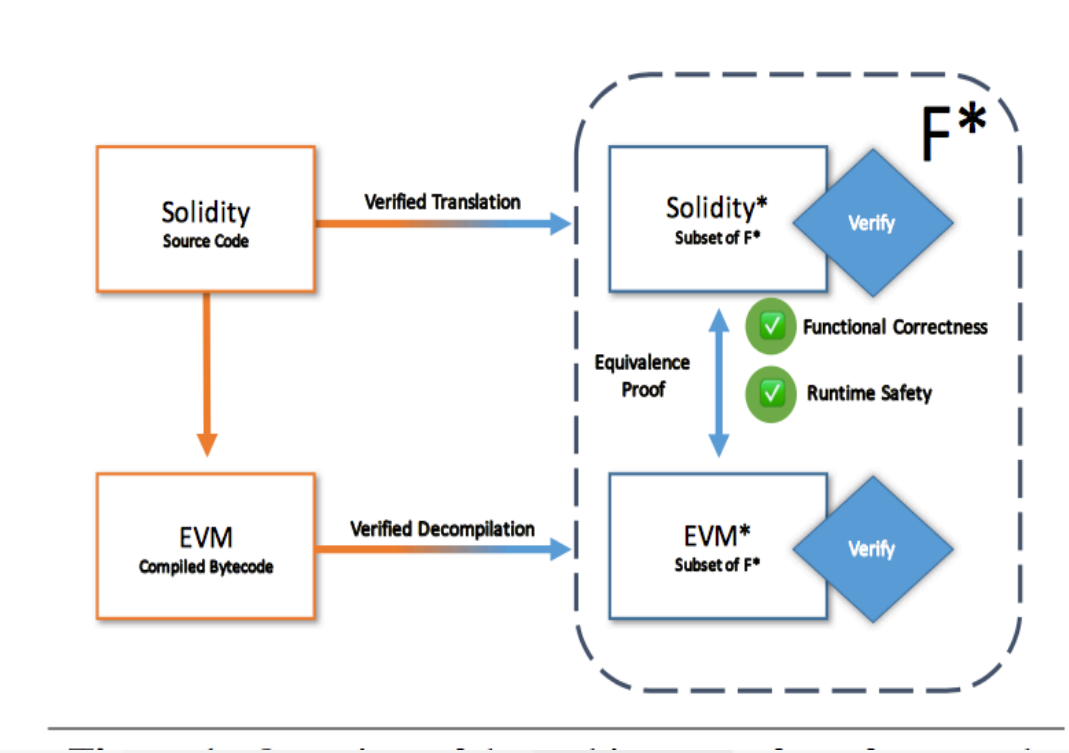
- N=锁定的block 数量 = 窗口期

其他方案

- **Bitcoin Beacon**
- 引入Bitcoin 作为信息源
- 所有人可验证
- 对抗恶意矿工

形式化验证

- 单元测试 不够完美
- 形式化验证数学上证明没有 Bug
- 仍在进行中



新的问题

- DDoS 攻击持续
- ICO 流量上等同于 DDoS
- ETH 下一个大版本 可能引入很多技术特性
  - 分片 （目前区块体积膨胀迅速）
  - PoS+PoW
- 新的安全工具漏洞

谢谢