Smart Contracts and Ethereum

Loi Luu
National University of Singapore

Winter School on Cryptocurrency and Blockchain Technologies
Shanghai, Jan. 15-17 2017

Some slides are courtesy of Vitalik Buterin
Agenda

• Smart contracts and applications
• Ethereum
• Interesting Ethereum-based projects
• Problems & challenges
SMART CONTRACTS
Definition

A smart contract is a computer program executed in a secure environment that directly controls digital assets.
A smart contract is a computer program executed in a secure environment that directly controls digital assets.
A computer program is a collection of instructions that performs a specific task when executed by a computer. A computer requires programs to function, and typically executes the program's instructions in a central processing unit.

Wikipedia
Example: bet on an event

if HAS_EVENT_X_HAPPENED() is true:
    send(party_A, 1000)
else:
    send(party_B, 1000)
A smart contract is a computer program executed in a **secure environment** that directly controls digital assets.
Properties of Secure Environments

• Correctness of execution
  – The execution is done correctly, is not tampered

• Integrity of code and data

• Optional properties
  – Confidentiality of code and data
  – Verifiability of execution
  – Availability for the programs running inside
Examples of secure environments

• Servers run by trusted parties
• Decentralized computer network (ie. blockchains)
• Quasi-decentralized computer network (ie. consortium blockchains)
• Servers secured by trusted hardware (e.g. SGX)
A smart contract is a computer program executed in a secure environment that *directly controls* digital assets
Example

• Legal contract: “I promise to send you $100 if my lecture is rated 1*”
• Smart contract: “I send $100 into a computer program executed in a secure environment which sends $100 to you if the rating of my lecture is 1*, otherwise it eventually sends $100 back to me”
A smart contract is a computer program executed in a secure environment that directly controls **digital assets**
What are digital assets?

• A broad category
  – Domain name
  – Website
  – Money
  – Anything tokenisable (e.g. gold, silver, stock share etc)
  – Game items
  – Network bandwidth, computation cycles
Example: top 5 crowdfunding campaigns in history

<table>
<thead>
<tr>
<th>Rank</th>
<th>Project</th>
<th>Category</th>
<th>Platform</th>
<th>Campaign end date</th>
<th>Campaign target</th>
<th>Amount raised</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><em>Star Citizen</em></td>
<td>Video game</td>
<td>Kickstarter, independent</td>
<td>Ongoing</td>
<td>$500,000</td>
<td>$90,009,649</td>
</tr>
<tr>
<td>2</td>
<td><em>Elio Motors</em></td>
<td>Automotive - Low-cost, high mileage vehicle</td>
<td>Independent</td>
<td>Ongoing</td>
<td>-</td>
<td>$21,161,869</td>
</tr>
<tr>
<td>3</td>
<td><em>Pebble Time</em></td>
<td>Smartwatch</td>
<td>Kickstarter</td>
<td>Mar 27, 2015</td>
<td>$500,000</td>
<td>$20,338,986</td>
</tr>
<tr>
<td>4</td>
<td><em>Ethereum</em></td>
<td>Cryptocurrency</td>
<td>Bitcoin, Independent</td>
<td>Sep 2, 2014</td>
<td>-</td>
<td>$18,439,086</td>
</tr>
<tr>
<td>5</td>
<td><em>Coolest Cooler</em></td>
<td>Product Design</td>
<td>Kickstarter</td>
<td>Aug 29, 2014</td>
<td>$50,000</td>
<td>$13,285,226</td>
</tr>
</tbody>
</table>
Star Citizen sold virtual spaceships in their game for $500 each
<table>
<thead>
<tr>
<th>Rank</th>
<th>Project</th>
<th>Category</th>
<th>Platform</th>
<th>Campaign end date</th>
<th>Campaign target</th>
<th>Amount raised</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><em>Star Citizen</em></td>
<td>Video game</td>
<td>Kickstarter, independent</td>
<td>Ongoing</td>
<td>$500,000</td>
<td>$90,009,649</td>
</tr>
<tr>
<td>2</td>
<td><em>Elio Motors</em></td>
<td>Automotive - Low-cost, high mileage vehicle</td>
<td>Independent</td>
<td>Ongoing</td>
<td>-</td>
<td>$21,161,869</td>
</tr>
<tr>
<td>3</td>
<td><em>Pebble Time</em></td>
<td>Smartwatch</td>
<td>Kickstarter</td>
<td>Mar 27, 2015</td>
<td>$500,000</td>
<td>$20,338,986</td>
</tr>
<tr>
<td>4</td>
<td><em>Ethereum</em></td>
<td>Cryptocurrency</td>
<td>Bitcoin, Independent</td>
<td>Sep 2, 2014</td>
<td>-</td>
<td>$18,439,086</td>
</tr>
<tr>
<td>5</td>
<td><em>Coolest Cooler</em></td>
<td>Product Design</td>
<td>Kickstarter</td>
<td>Aug 29, 2014</td>
<td>$50,000</td>
<td>$13,285,226</td>
</tr>
</tbody>
</table>

Ethereum Foundation sold 60,102,206 digital tokens which will be useful in a decentralized network.
What are smart contracts’ applications?
Example: escrow service for exchange

A → E → B

A

E

B

A

E

B

A

E

B

A

E

B

A

E

B


I won't give you Y!
Example: multisig

• Require M of N “owners” to agree in order for a particular digital asset to be transferred
  – Individual use cases
    • eg. two-factor authentication
  – Intra-organizational use cases
A lot more interesting applications

• Individual/intra-organizational
  – Complex access policies depending on amount, withdrawal limits, etc
  – Dead man’s switch, “digital will”
    • E.g When the owner dies, transfer all assets to someone

• General
  – Prediction markets
  – Insurance
  – Micro-payments for computational services (file storage, bandwidth, computation, etc)
Why smart contracts?

• Automated processing
• Trust reduction
  – Trust the secure environments, not a very large number of contract enforcement mechanisms
• Unambiguous, terms clearly expressed in code
  – Question: how to express terms clearly in code?
ETHEREUM: THE FIRST BLOCKCHAIN-BASED SMART CONTRACT PLATFORM
Ethereum

• Blockchain with expressive programming language
  – Programming language makes it ideal for smart contracts

• Why?
  – Most public blockchains are cryptocurrencies
    • Can only transfer coins between users
  – Smart contracts enable much more applications
Analogy: Most existing blockchain protocols were designed like

OR THIS
why not make a protocol that works like

OR THIS

OR THIS

OR THIS
How Ethereum Works

• Two types of account:
  – **Normal account** like in Bitcoin
    • has balance and address
  – **Smart Contract account**
    • like an object: containing (i) code, and (ii) private storage (key-value storage)
    • Code can
      – Send ETH to other accounts
      – Read/write storage
      – Call (ie. start execution in) other contracts
DNS: The “Hello World” of Ethereum

```python
def register(addr):
    if not self.domains[addr].owner:
        self.domains[addr].owner = msg.sender

def set_ip(addr, ip):
    if self.domains[addr].owner == msg.sender:
        self.domains[addr].ip = ip
```

Private Storage

Can be invoked by other accounts
Ethereum Languages

- Ethereum VM Bytecode Stack Language
- Lower-Level Language
  - Serpent (Looks like Python)
  - Solidity (Types, invariants, looks like Javascript)

Functional, macros, looks like Scheme

Looks like Forth. Defined in Yellowpaper

Slide is courtesy of Andrew Miller
Example

```
contract Greetings {
    string greeting;
    function Greetings (string _greeting) public {
        greeting = _greeting;
    }
    /* main function */
    function greet() constant returns (string) {
        return greeting;
    }
}
```

What you write

What people get from the disassembler

What other see on the blockchain

```
606060405260040516102503
80380610250833981016040
528........
```

PUSH 60
PUSH 40
MSTORE
PUSH 0
CALLDATALOAD

.....
Transactions in Ethereum

• Normal transactions like Bitcoin transactions
  – Send tokens between accounts
• Transactions to contracts
  – like function calls to objects
  – specify which object you are talking to, which function, and what data (if possible)
• Transactions to create contracts
Transactions

- **nonce** (anti-replay-attack)
- **to** (destination address)
- **value** (amount of ETH to send)
- **data** (readable by contract code)
- **gasprice** (amount of ether per unit gas)
- **startgas** (maximum gas consumable)
- **v, r, s** (ECDSA signature values)
How to Create a Contract?

- Submit a transaction to the blockchain
  - nonce: previous nonce + 1
  - to: empty
  - value: value sent to the new contract
  - data: contains the code of the contract
  - gasprice (amount of ether per unit gas)
  - startgas (maximum gas consumable)
  - v, r, s (ECDSA signature values)

- If tx is successful
  - Returns the address of the new contract
How to Interact With a Contract?

• Submit a transaction to the blockchain
  – **nonce**: previous nonce + 1
  – **to**: contract address
  – **value**: value sent to the new contract
  – **data**: data supposed to be read by the contract
  – **gasprice** (amount of ether per unit gas)
  – **startgas** (maximum gas consumable)
  – **v, r, s** (ECDSA signature values)

• If tx is successful
  – Returns outputs from the contract (if applicable)
Blockchain State

Bitcoin’s state consists of key value mapping addresses to account balance

<table>
<thead>
<tr>
<th>Address</th>
<th>Balance (BTC)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x123456...</td>
<td>10</td>
</tr>
<tr>
<td>0x1a2b3f...</td>
<td>1</td>
</tr>
<tr>
<td>0xab123d...</td>
<td>1.1</td>
</tr>
</tbody>
</table>

Ethereum’s state consists of key value mapping addresses to account objects

<table>
<thead>
<tr>
<th>Address</th>
<th>Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x123456...</td>
<td>X</td>
</tr>
<tr>
<td>0x1a2b3f...</td>
<td>Y</td>
</tr>
<tr>
<td>0xab123d...</td>
<td>Z</td>
</tr>
</tbody>
</table>

Blockchain != Blockchain State
Every account object contains 4 pieces of data:

- Nonce
- Balance
- Code hash (code = empty string for normal accounts)
- Storage trie root
Block Mining

- Miners

Block:
- Previous block
- A set of TXs
- New State Root
- Receipt Root
- Nonce

Verify transactions & execute all code to update the state

SHA3(Block) < D

Broadcast Block
Code execution

• Every (full) node on the blockchain processes every transaction and stores the entire state and state transitions.
Dos Attack Vector

• Halting problem
  – Cannot tell whether or not a program will run infinitely
  – A malicious miner can DoS attack full nodes by including lots of computation in their txs
• Full nodes attacked when verifying the block

```c
uint i = 1;
while (i++ > 0) {
  donothing();
}
```
Solution: Gas

- Charge fee per computational step ("gas")
  - Special gas fees for operations that take up storage

<table>
<thead>
<tr>
<th>Operation</th>
<th>Gas</th>
<th>GasCost</th>
</tr>
</thead>
<tbody>
<tr>
<td>PUSH1</td>
<td>111741</td>
<td>3</td>
</tr>
<tr>
<td>PUSH1</td>
<td>111738</td>
<td>3</td>
</tr>
<tr>
<td>MSTORE</td>
<td>111726</td>
<td>12</td>
</tr>
<tr>
<td>CALLDATASIZE</td>
<td>111724</td>
<td>2</td>
</tr>
<tr>
<td>ISZERO</td>
<td>111721</td>
<td>3</td>
</tr>
<tr>
<td>PUSH2</td>
<td>111718</td>
<td>3</td>
</tr>
<tr>
<td>JUMPI</td>
<td>111708</td>
<td>10</td>
</tr>
</tbody>
</table>
Sender has to pay for the gas

- **gasprice**: amount of ether per unit gas
- **startgas**: maximum gas consumable
  - If **startgas** is less than needed
    - Out of gas exception, revert the state as if the TX has never happened
    - Sender still pays all the gas
- **TX fee = gasprice * consumedgas**
- **Gas limit**: similar to block size limit in Bitcoin
  - Total gas spent by all transactions in a block < Gas Limit
INTERESTING ETHEREUM-BASED PROJECTS
BTCRelay

- A bridge between the Bitcoin blockchain & the Ethereum blockchain
  - Allow to verify Bitcoin transactions within Ethereum network
  - Allow Ethereum contracts to read information from Bitcoin blockchain
BTCRelay – How it works

Relayers constantly submit Bitcoin block headers.

A Bitcoin transaction is submitted, BTCRelay verifies TX based on the block header.

The verified Bitcoin transaction is relayed to the smart contract.

Relayers receive the transaction fee.
BTCRelay Application: ETH-BTC atomic swaps

50 ETH for anyone who sends 1 BTC to my address

Send 1 BTC to Alice address, here is the proof P

Send 50 ETH

BTCRelay

Check proof P

Send 1 BTC to Alice address

Bitcoin Network
BTCRelay Application: Contracts can read information of Bitcoin blockchain

E.g. betting on the outcomes of events on Bitcoin blockchain
Other Work-in-progress Relays

- Project Alchemy
  - Zcash relay
- Dogecoin/ Litecoin Relay
  - Dogecoin light client on Ethereum by Vitalik
  - Interactive verification for Scrypt pow by Christian

Question: can we build a decentralized exchange between cryptocurrencies using all the relays?
SmartPool

- Decentralized Mining Pools using Smart Contracts
- Problem: mining centralization
  - Miners go to mining pools for stable and frequent rewards
  - Decentralized platforms are secured by centralized entities
    - Transaction censorships
    - Single point of failures
Pooled mining

- Pools track miners’ contribution by using shares
  - A share is similar to a block, but required less work to find
P2Pool: decentralized mining pool

• Miners maintain the pool’s contributions by themselves
  – Maintain a share-chain within the pool (just like the blockchain)
  – Pay miners in proportional to their contributions
    • Done in the coinbase transaction
• When a miner finds a share
  – Broadcast to all miners
  – Check if the coinbase tx is correct and extend the share-chain
Why P2Pool is Inefficient and not scalable?

• Millions of messages per block (each per share)
  – Expensive to everyone

• Reducing the number of shares?
  – No, will increase the variance of reward
SmartPool: Efficient P2Pool using SmartContract

- Track miners’ contributions to the pool in a contract
- Allows batch submissions, e.g. billions of shares in a claim
  - Reduce number of messages (txs) to the contract significantly
- Use probabilistic verification to check a submission
  - Randomly verify only one share per submission
  - Probability of cheating being detected is proportional to the amount of cheating
SmartPool: Disincentivize cheating

- Payment scheme: pay 0 for a submission if cheating detected
  - Expected reward is the same whether cheating or not
  - Miners have no incentive to cheat

Reward = 1

- Probabilistic verification
  - Passed
  - Get 1.5 Reward with 2/3 probability
  - Detected
  - Get 0 Reward with 1/3 probability

Expected reward = 1
More in the paper

• How to prevent miners from stealing others’ shares?
• How to prevent claiming a share multiple times
  – Within a submission
  – Across submissions
• How to verify Ethash PoW?
  – Require huge memory and storage
SmartPool.io is calling for donation

WE ARE CALLING FOR DONATIONS

Current donated amount: 1,150.2 ETH

Our addresses

Ethereum: 0x98F62d8aD5a884C8bbcf262591DFF55DAb263B80
Bitcoin: 1Cs3D54RqjhNwHurj97qQpbiDSYw1EkjPC
ZCash: t1eZFVNbvfgGShyPX4RzScLd76apdVoD2qN
A lot more interesting apps

- **TownCrier** and **Oraclize**
  - allow contracts to fetch external data from real websites
  - Enable a lots of applications: betting, insurance, bounty based on real world event
- **Augur** and **Gnosis**
  - Prediction market: predict the outcome of real world event to get reward
- Many others: theDao, iConomi, Golem, etc
PROBLEMS/ CHALLENGES
Privacy

• Ethereum blockchain guarantees correctness and availability, not privacy for smart contracts
  – Everything on the Ethereum blockchain is public
    • Cannot execute on private data (e.g. death will remains secret until the owner dies)

• Transactions are traceable
  – Analysing transaction graph [IMC’13]
Privacy Solution

• **Hawk** (Kosba et al. IEEE S&P’16)
  • Privacy-Preserving Smart Contracts
  • Execute confidential, fair, multiparty protocols

• **ZeroCash over Ethereum**, **Ring signatures on Ethereum**
  – Mixing coins with others
Scalability

- Resources on blockchain are expensive
  - Full nodes perform the same on-chain computations
  - Full nodes store the same data
- Gas-limit is relatively small
  - Can’t run an OS on blockchain
  - Can’t increase gas-limit: DoS vector
Scalability Solution 1: Sharding

• Divide the network into sub-networks
  – each stores and manages a fraction of the blockchain (a shard)
  – Allow scaling up as the network grows

• There is a catch
  – May affect usability or performance
  – May not be compatible with all existing applications
Scalability Solution 2: State Channel

- Similar to payment channel (e.g. lightning network) but for states
  - Scaling by using off-chain transactions
  - Can update the state multiple times
  - Only settlement transactions are on-chain

- Challenges
  - Cannot create state channel for all applications
  - Still early research, more work needed
Scalability Solutions: Other approaches

• Storage rental
  – Problem: data fee is charged once
  – Idea: Charge more fees if store data longer
    • Similar to resource tax
    • Incentivize users to remove unnecessary data

• Hardware-rooted trust
  – Using SGX to build state channel? (Inspired by teechan protocol)
Security Flaws

• Due to abstraction of semantic
  – Transaction ordering dependence
  – Reentrancy bug
    • Which exploited the DAO
• Obscure VM rules
  – Maximum stack depth is 1024: not many devs know
  – Inconsistent Exception Handling in EVM

The DAO Attacked: Code Issue Leads to $60 Million Ether Theft

Michael del Castillo (@DelRayMan) | Published on June 17, 2016 at 14:00 GMT
Example 1: Transaction Ordering Dependence

PuzzleSolver Contract

```
Balance: 100

PuzzleSolver()
SetPuzzle
reward=100

SubmitSolution(solution)
if isCorrect(solution):
Send(reward)

UpdateReward(newReward)
reward=newReward
```

Owner can update the reward anytime

Anyone can submit a solution to claim the reward
Scenario 1: SubmitSolution is triggered

Random TXs

Scenario 1:
SubmitSolution is triggered

PuzzleSolver Contract

Miners

+100

Solution for Puzzle

Random TXs

Other TXs

SubmitSolution

Random TXs

SubmitSolution

Other TXs

Balance: 0

PuzzleSolver()
SetDifficulty
reward=100

SubmitSolution(solution)
if isCorrect(solution):
Send(reward)

UpdateReward(newReward)
reward=newReward
Scenario 2: Both SubmitSolution and UpdateReward are triggered

Miners

Solution for Puzzle

Update Reward to $0!

Other TXs

Block

UpdateReward = 0

SubmitSolution

Other TXs

PuzzleSolver Contract

Balance: 0

PuzzleSolver()
SetDifficulty
reward=100

SubmitSolution(solution)
if isCorrect(solution):
Send(reward)

UpdateReward(newReward)
reward=newReward

69
Transaction Ordering Dependence

• Observed state != execution state
  • Transactions do not have atomicity property

• Can be coincidence
  • Two transactions happen at the same time

Solution for Puzzle
Other TXs
Update Reward to $0!
Transaction Ordering Dependence

• Observed state $\neq$ execution state
  • Transactions do not have atomicity property

• Can be coincidence
  • Two transactions happen at the same time

• Can be a malicious intention
  • Saw the targeted TX from the victim
  • Submit the second TX to update the reward
  • Both TXs enter the race
Example 2: Reentrancy Bug --- TheDAO Bug

- Reentrancy vulnerability
  - Most expensive vulnerability to date
- Call before balance update

```solidity
... // Burn DAO Tokens
if (balances[msg.sender] == 0)
    throw;
withdrawRewardFor(msg.sender);
totalSupply -= balances[msg.sender];
balances[msg.sender] = 0;
paidOut[msg.sender] = 0;
return true;
```
TheDAO Bug: Honest Scenario

The DAO

- `splitDAO(proposal, address)`
- `withdrawRewardFor(msg.sender)`
- `rewardAccount.payOut(_account, reward)`
- `balances[msg.sender] = 0;`

Receiver

- `function() {}`

Balance: $0
Payout: $0
TheDAO Bug: Attack Scenario

TheDao

- splitDAO(proposal, address)
- withdrawRewardFor(msg.sender)
- rewardAccount.payOut(_account, reward)

Receiver

- splitDAO()
Solutions to Resolve Security Flaws

• Create developer tools
  – Smart contract analyser based on symbolic exec: Oyente
  – Testing and deployment framework: truffle
  – Formal verification for smart contracts: eth-isabelle, why3

• Design better semantic [CCS’16]

• Educate users

• Idea
  – Create security certificates for smart contracts?
Closing thought

Ethereum and Smart contract are awesome, build your own Dapp today!
  – Pay more attention to security
Oyente: An Analyzer for Smart Contracts
Architecture

• Based on symbolic execution
• Have separate modules
  – Can add more analysis separately

6060604052123
123123528.....

ByteCode → EXPLORER → CORE ANALYSIS → VALIDATOR → Visualizer
Ethereum State

Z3 Bit-Vector Solver

CFG BUILDER
Symbolic Execution

Is there any value of x?

\[ C_1 \land C_2 \land C_3 \land (z = x + 2) \]

Theorem Prover

\[ \begin{align*} C_1 &: (x > 0) \\ C_2 &: (z < 15) \\ C_3 &: (z < 8) \end{align*} \]

\[ z = x + 2; \]

Inputs

Control Flow Graph

Execution Trace

\[ x = 10 \]
What Can Oyente Do?

• Detect Bugs In Existing Smart Contracts
  – Run with 19,366 contracts
  – 30 mins timeout per contract

• Test generation
  – Cover all possible paths of each program
Oyente is Open Source

• https://github.com/ethereum/oyente

• Future work
  – Support more opcodes
  – Handle loops
  – Combine static and dynamic symbolic executions