3. Smart Contract and Ethereum

IERG5590 Advanced Blockchain
Sherman S. M. Chow

Slides adopted from those by Loi Luu (PhD from NUS) who in turn adopted from Vitalik Buterin and Andrew Miller (UIUC)
Agenda

• Smart contracts
  – 4 “core components”
  – How “smart” it is?
  – Applications

• Ethereum
  – Ethereum language
  – Blockchain state, transactions, contracts
  – Gas
  – Security

• 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.

Wikipedia
Example: bet on an event

```python
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 (i.e. blockchains)
- Quasi-decentralized computer network (i.e. 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 of Contracts

- 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?

- Domain name
- Website
- Money
- Anything tokenizable (e.g. gold, silver, stock share, etc.)
- Game items
- Network bandwidth
- Computation cycles
- ...
- A broad category
Example: top 5 crowdfunding campaigns in history ('16)

<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>Star Citizen</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>Elio Motors</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>Pebble Time</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>Ethereum</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>Coolest Cooler</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
Ethereum Foundation sold 60,102,206 digital tokens to be used in a decentralized network
Top crowdfunding campaigns today (’19)

- 42 projects raised over 10 million

<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>EOS</td>
<td>Blockchain</td>
<td>Ethereum</td>
<td>June 1, 2018</td>
<td>-</td>
<td>$4,000,000,000+[1]</td>
</tr>
</tbody>
</table>

- EOS is a blockchain operating system designed to support commercial decentralized applications.
- Filecoin is a decentralized data storage application.
How “smart” it is?

• Automated processing
  – facilitate, verify, or enforce the execution of a contract

• Trust reduction
  – not depending on a very large number of contract enforcement mechanisms, but the secure environments

• Trackable and irreversible

• Unambiguous, terms clearly expressed in code
  – Question: how to express terms clearly in code?
Smart contracts vs. Legal contracts

- A smart contract is more like a vending machine
  - Follow predetermined rules

<table>
<thead>
<tr>
<th>Legal contracts</th>
<th>Smart contracts</th>
</tr>
</thead>
<tbody>
<tr>
<td>Good at subjective claims (i.e. requiring human judgement)</td>
<td>Good at objective claims (i.e. mathematically evaluable)</td>
</tr>
<tr>
<td>High cost</td>
<td>Low cost</td>
</tr>
<tr>
<td>May require long legal process</td>
<td>Fast and automated</td>
</tr>
<tr>
<td>Relies on penalties</td>
<td>Relies on collateral/security deposits</td>
</tr>
<tr>
<td>Jurisdiction-bound (“by law”)</td>
<td>Potentially international</td>
</tr>
</tbody>
</table>
Smart contracts vs. Legal contracts

• Smart contracts are not very effective for loans
  – Has the capital to provide liquid collateral for a loan, do not need the loan in the first place
  – Can use illiquid collateral though (e.g. domain names)

• Legal contracts are not very effective for anti-spamming
  – amounts at stake are so small
  – spammers can locate themselves in favorable jurisdictions and evade detection
SMART-CONTRACTS-BASED APPLICATIONS
Example: Escrow service for exchange

Two individuals A and B are exchanging their corresponding object X and Y.

Normal Scenario:

Dispute Scenario:

I won't give you Y!
Example: Multisignature

- Require $M$ of $N$ “owners” to agree for a particular digital asset to be transferred
  - Intra-organizational use cases
  - Still make sense in Individual use cases
  - e.g. two-factor authentication
    - two private siging keys are stored in different storage mediums
A lot of interesting applications

• Individual/intra-organizational
  – Complex access policies depending on amount, withdrawal limit, etc.
  – “Dead man’s switch” or “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.
Decentralized Exchange

- Blockchain → Decentralization → Democracy?
- Democratizing access to financial services
- How do you buy foreign currencies now?
- How “Sharing Economy Applications” (SEA) works?
  - AirBnb, Uber, etc.
- Decentralized exchange is like those “SEA”
  - People exchange among themselves instead of with a “dealer”
- except there is no centralized server running the service
- but blockchain (or a Decentralized Autonomous Organization)
QueenBee: Decentralized Search on Decentralized Web

http://cidrdb.org/cidr2019/gongshow.html
ETHEREUM: THE FIRST BLOCKCHAIN-BASED SMART CONTRACT PLATFORM
Ethereum

- Blockchain with an expressive programming language
  - a language making it ideal for smart contracts
- Why? (back then, before ’16)
  - 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

************

OR THIS

bitcoin

MONERO
why not make a protocol that works like

OR THIS

OR THIS

OR THIS
Account-based Model vs. UTXO Model

<table>
<thead>
<tr>
<th>Date</th>
<th>Description</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>01/01/19</td>
<td>...</td>
<td>$25.0</td>
</tr>
<tr>
<td>01/02/19</td>
<td>$17 -&gt; Bob</td>
<td>$8.0</td>
</tr>
</tbody>
</table>

---

<table>
<thead>
<tr>
<th>Date</th>
<th>Description</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>01/01/19</td>
<td>...</td>
<td>$25.0</td>
</tr>
<tr>
<td>01/02/19</td>
<td>Alice -&gt; $17</td>
<td>$17.0</td>
</tr>
</tbody>
</table>
How Ethereum Works

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

data domains[](owner, ip)

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

Stored by the contract
Can be invoked by other accounts
Can be invoked by other accounts
Ethereum Languages

- **Serpent**: Looks like Python
- **Solidity**: Looks like Javascript (types, invariants, ...)
- **Lower-Level Language**: Looks like scheme (functional, macros)

*Slide is courtesy of Andrew Miller*
Example

What you write

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

Bytecode seen by others on the blockchain:

```
6060604052604051600052606040516102506038016040528
```

What people get from the disassembler:

```
PUSH 60
PUSH 40
MSTORE
PUSH 0
CALLDATALOAD
.....
```
Transactions in Ethereum

• Transactions sending tokens between accounts
  – like bitcoin
• “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** (number used *once*, anti-replay-attack)
- **to** (destination address)
- **value** (amount of ETH to send)
- **data** (readable by contract code)
- **gasprice** (amount of ETH 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 ETH per unit gas)
  – **startgas** (maximum gas consumable)
  – **v, r, s** (ECDSA signature values)

• If tx is successful returns
  – new contract’s addr. (derived from creator addr. & nonce)
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 ETH 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:
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>10.0</td>
</tr>
<tr>
<td>0x1a2b3f...</td>
<td>1.0</td>
</tr>
<tr>
<td>0xab123d...</td>
<td>1.1</td>
</tr>
</tbody>
</table>

Ethereum’s state:
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 is not Blockchain State
Account Object

• Every account object contains 4 pieces of data:
  – Nonce
  – Balance
  – Code hash (code = empty string for normal accounts)
  – Storage trie root
Block Mining

Receipt root is the root of the tree of all “receipts”. Each receipt represents an intermediate state root after 1 transaction is executed.
Code Execution

- Every (full) node on the blockchain processes every transaction and stores the entire state.
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

```cpp
uint i = 1;
while (i++ > 0) {
  doNothing();
}
```
Solution: “Gas”

- Charge fee per *computational* step
- Special fee for operations that take up *storage*
Sender has to pay for the gas

- **gasprice**: amount of ETH 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
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
• Transactions are traceable
  – *Analysing transaction graph* [Meiklejohn *et al.* 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
  - Cannot run an OS on blockchain
  - Cannot 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/performance/security
  – May not be compatible w/ all applications

• More research needed
Scalability Solution 2: State Channel

- A channel for off-chain transactions
  - similar to payment channel in, e.g., lightning network, but for states
- Can update the state multiple times
- Only settlement TXs are on-chain
- But not applicable for some apps.
- More research needed
Many Researchers “joined” the Industry!

- Abelian
- Algorand
- Conflux
- Dfinity
- Ergo Platform
- Fractal Platform
- Hcash

- IOHK
- Kyber Network
- Oasis Labs
- StarkWare
- ThunderCore
- Zcash
- ...
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) I 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

Anyone can submit a solution to claim the reward

Owner can update the reward anytime
Scenario 1: SubmitSolution is triggered

Miners

Solution for Puzzle
Random TXs
Other TXs

Block
Random TXs
SubmitSolution
Other TXs

PuzzleSolver Contract
Balance: 0
PuzzleSolver()
SetDifficulty
reward=100
SubmitSolution(solution)
if isCorrect(solution):
Send(reward)
UpdateReward(newReward)
reward=newReward

+100

Random TXs
Balance: 100
Scenario 2: Triggered also UpdateReward

PuzzleSolver Contract
- Balance: 0
- SetDifficulty
  - reward=100
- SubmitSolution(solution)
  - if isCorrect(solution):
    - Send(reward)
- UpdateReward(newReward)
  - reward=newReward

Miners

Solution for Puzzle
Update Reward to $0!
Other TXs

Other TXs

Block
- UpdateReward = 0
- SubmitSolution
- Other TXs
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

TheDAO is the name of a particular DAO

- msg.sender should be a normal account →

- Most expensive vulnerability to date
- msg.sender is a smart contract account
- It calls this vulnerable smart contract again
  - Recursion!
- Keep withdrawing reward for this sender
- balances[msg.sender] = 0; is not executed once.
TheDAO Bug: Honest Scenario

TheDAO

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

Balance: $000
Payout : $000

Receiver

function() {}
TheDAO Bug: Attack Scenario

TheDAO

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

Receiver

Balance: 100
Payout : £00

Function call vs. Contract call!
Solutions to Resolve Security Flaws

• Create developer tools
  – Smart contract analyzer 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