ECE 443/518 – Computer Cyber Security
Lecture 17 Cryptocurrency and Smart Contract

Professor Jia Wang
Department of Electrical and Computer Engineering
Illinois Institute of Technology

October 21, 2024

Outline

Proof of Work

Smart Contract

Reading Assignment

- This lecture: Cryptocurrency, Smart Contract
- Next lecture: Bitcoin Management, Oblivious Transfer

Outline

Proof of Work

Smart Contract

BFT Meets Cryptocurrency

- Consensus on what the next block should be is a must for cryptocurrency.
  - Otherwise adversaries can create branches for double spending.
- Many BFT protocols are too weak to be useful here.
  - E.g. both the ones without or with digital signatures need to know how many traitors are there, but for cryptocurrrency adversary can create arbitrary number of "traitor" accounts.
- The BFT propocol need to weight the participants differently.
  - So adversaries cannot simply overwhelm the protocol by creating more accounts, a.k.a. Sybil attack.

Proof of Work (PoW)

- Who willing to generate the next block needs to perform certain amount of work before it could join the BFT protocol.
  - Typical work: for a block of hash s, find x so that $h(s||x)$ is smaller than a threshold.
  - If h is preimage resistant, one can only find such x via brute-force.
  - Block time: the expected time for some account to find a solution x.
  - When more computational power are available, the threshold is reduced such that the block time remains unchanged.
- Who willing to participate will have an account to receive economic incentives for the work.
- Proof of Work consensus: the branch with the most of work is the correct one
  - The consensus can be reached as long as honest account owners can provide majority of work.

Obtaining the Blockchain

- Consider an account that want to generate the next block.
  - By using a program together with its private key.
- The program connects to Internet to query the blockchain.
  - However, there could be adversaries so the program must decide if the chain it receives is valid or not.
- The genesis block: the first block of the blockchain.
  - The genesis block is assumed to be well-known, usually coded into the program directly.
  - The genesis block could contain data like cryptocurrency parameters and initial balances for certain accounts.
- The program need to validate past transactions.
  - It is necessary to accumulate balances for all accounts to decide if transactions are valid – this is possible now since our computers are actually quite powerful.
- However, recall that valid transactions along cannot prevent branches (and thus double spending).

PoW Fork Choice

- Fork: the program may receive multiple blockchains all with the same correct genesis block.
  - They diverge somewhere back in the history.
  - Resulting from a temporary network partitioning or an attack.
- Choice: with PoW, the program should pick the blockchain with the most of the work as the correct one.
  - The work is measured as the total effort to solve the problems for all the blocks along the chain.
- 51% attack: attackers controlling more than half of the computational power could collude to cause a successful fork.
  - Suppose currently the honest accounts are at the chain $A \to B \to \cdots \to C$ from an earlier block A.
  - The attackers make a fork $A \to B'$ and continue.
  - No matter how many blocks are there between A and C, the attackers can eventually reach D' as $A \to B' \to \cdots \to D'$, that contains more work than the chain created by the honest accounts now as $A \to B \to \cdots \to C \to \cdots \to D$

PoW Finality

- With 51% attack, powerful attackers can revert transactions by creating successful forks.
  - It may take some time but 100% the attack will be successful.
- Can attackers with less computational power revert a block?
  - Finality: we need to define when a block is considered "final" and thus is not supposed to be changed or reverted.
  - Fake check scams are classical examples of attacks on finality for our banking system.
- Consider an attacker controlling 25% of computational power
  - Suppose the current chain is $A \to B$ and honest accounts are working on the block C.
  - If B is considered final immediately, the attacker will attempt to make a fork $A \to B' \to C'$ when A was ready.
  - If C' can be generated ahead of C in time, honest accounts may simply follow the chain $A \to B' \to C'$.
  - With 25% of computational power, this may happen with a probability of $(\frac{25\%}{75\%})^2 = \frac{1}{9}$.
- Practically, one should wait a few blocks to reduce the chance of having forks due to possible attacks.

Economic Incentives

- How could PoW cryptocurrencies actually survive when finality is always probablistic, and when powerful adversaries could have the majority of computational power?
- Economic incentives to attract honest accounts to participate in the BFT protocol.
  - Transaction fees: the account creates the next block will take all the transaction fees.
    - When there is more transactions than what the next block can hold, payers compete by paying more transaction fees.
  - Mining: the account creates the next block is allowed to award itself a predefined amount of money.
    - As a transaction with no payer.
- As a consequence, powerful adversaries have economic incentives to not cheat.
  - It is more rewarding to participate honestly than to make the cryptocurrency useless by attacking it.

Proof of Stake (PoS)

- PoW consensus achieves a great success and enables a lot of honest account owners to participate.
- For a fixed block time, need to increase complexity of work.
  - So more energy is needed to generate one block.
  - hardware depreciation + energy cost + profit = mining income
- Proof of stake: accounts stake a certain amount of the cryptocurrency itself to participate in the consensus process.
  - Without the need of computing complex works (and thus consume less energy) to resist Sybil attacks.
  - Honest accounts are rewarded with transaction fees.
  - Attackers may have their staked cryptocurrency burned.

Outline

Proof of Work

Smart Contract

From Ledger to State Machine

- The ledger as stored in the block chain can be treated as a very simple state machine.
  - Initial state: initial account balances
  - Currect state: current account balances
  - State transitions: each blockchain transaction updates account balances by addition and subtraction.
- The blockchain can support more complex state machines.
  - Allow accounts to define state variables in addition to balance.
  - Allow blockchain transactions to perform more operations on state variables than simple addition and subtraction.
- This is similar to how we build computer hardware and software to support general purpose computing need.
  - E.g. Ethereum Virtual Machine (EVM) defined by the Ethereum blockchain uses 8-bit opcode and a stack to organize its 256-bit registers, and supports high-level programming languages like Solidity.

Smart Contract

- What are the benefits of running state machines and thus programs in a blockchain?
  - Not for efficiency: each computation needs to be executed as many times as anyone would need to validate the blockchain, using the same inputs and generating the same output.
  - Nonrepudiation: the account initiates a computation must sign the request and cannot deny so.
  - Integrity: the outcome is permanentely recorded in the blockchain and cannot be reverted.
  - As long as there is no branch.
- That is what is necessary to execute a contract.
  - Smart contract: a program running inside a blockchain.

A Smart Contract Example

```
pragma solidity 0.8.7;

contract VendingMachine {
  // Declare state variables of the contract
  address public owner;
  mapping (address => uint) public cupcakeBalances;

  // When 'VendingMachine' contract is deployed:
  // 1. set the deploying address as the owner of the contract
  // 2. set the deployed smart contract's cupcake balance to 100
  constructor() {
    owner = msg.sender;
    cupcakeBalances[address(this)] = 100;
  }
  ...
```

- A smart contract that you can buy cupcakes on Ethereum.
- No you don't receive an actual cupcake.
  - What you received could be treated as a ticket or token to redeem a physical cupcake somewhere.

Smart Contract Account

```
...
constructor() {
  owner = msg.sender;
  cupcakeBalances[address(this)] = 100;
}
...
```

- Once created, a smart contract will has its own address, as indicated by `address(this)`
- Other accounts interact with the smart contract by sending (signed) messages to the smart contract account.
- The smart contract will handle these messages in member functions.
  - `constructor` is a special one called for the first message which deploys the smart contract.

The Message Sender

```
contract VendingMachine {
  // Declare state variables of the contract
  address public owner;
  mapping (address => uint) public cupcakeBalances;

  // When 'VendingMachine' contract is deployed:
  // 1. set the deploying address as the owner of the contract
  // 2. set the deployed smart contract's cupcake balance to 100
  constructor() {
    owner = msg.sender;
    cupcakeBalances[address(this)] = 100;
  }
  ...
```

- `msg.sender` indicates who initiates the computation.
  - The payer of cryptocurrency.
- The sender should in addition specify what transactions (member function) is to be performed (called).
  - E.g. one of `constructor`, `refill`, and `purchase`
  - Plus other necessary parameters.

Transactions

```
contract VendingMachine {
  ...
  // Allow the owner to increase the smart contract's cupcake balance
  function refill(uint amount) public {
    require(msg.sender == owner, "Only the owner can refill.");
    cupcakeBalances[address(this)] += amount;
  }

  // Allow anyone to purchase cupcakes
  function purchase(uint amount) public payable {
    require(msg.value >= amount * 1 ether, "1 ETH per cupcake");
    require(cupcakeBalances[address(this)] >= amount, "Not enough in stock");
    cupcakeBalances[address(this)] -= amount;
    cupcakeBalances[msg.sender] += amount;
  }
}
```

- `msg.value` indicates money the sender pays the the contract.
    - The money is transfered from the sender address to the contract address automatically if the computation completes successfully.
- How could one withdraw money from the contract?

Complications

- What if there is an infinite loop into a smart contract?
  - Can be exploited by adversaries to jam the blockchain.
  - In theory, we cannot detect if there is an infinite loop in a program.
  - On blockchain, we can solve the issue by limiting the number of instructions a smart contract may execute by the transaction fee the sender would like to pay.
- Since the program of a smart contract need to be deployed to the blockchain, everyone can see and analyze it.
  - Bugs in the program could be found and exploited by adversaries.

Summary

- Both proof of work (PoW) and proof of stake (PoS) work as the consensus mechanism for cryptocurrencies.
- Smart contracts are programs running inside a blockchain, reacting to blockchain events.