Smart Contracts Verification (89400)

Yoni Zohar – Bar Ilan University

Seminar
Outline

1. Seminar Plan
2. Blockchain and Smart Contracts
3. Verification
4. Seminar Overview
Seminar Goals

- Learn how to read a paper in CS
  - Focus on the important results
  - Cover necessary background
- Learn how to present
  - Who are you presenting to
  - What is the important message
  - Keep audience engaged
- Discover interesting research and tools
  - Active field of research
  - Many new techniques and tools

And many more institutions and startups
Technical Details

- 10:00 – 11:30
- Present one paper (80%)
- Attend and Participate (20%)
- [https://u.cs.biu.ac.il/~zoharyo1/sc-seminar/index.html](https://u.cs.biu.ac.il/~zoharyo1/sc-seminar/index.html)
Guidelines

- Make enough time for reading the paper
  - Read background material
  - Be prepared to answer questions from audience
  - Presenting a demo of the tools themselves is recommended

- Make enough time for preparing the slides
  - Slides should be clear
  - Practice the time frame

- When there are difficulties:
  - Ask me questions
  - Meet with me
  - Start early
Motivation

- Transfer money between parties **directly**
- Not going through a bank
- Retain security without a trusted verifying third-party

---

**Bitcoin: A Peer-to-Peer Electronic Cash System**

Satoshi Nakamoto  
satoshin@gmx.com  
www.bitcoin.org

**Abstract.** A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of
**Blockchain**

**What is a blockchain?**

- Linked list
- Elements are called **blocks**
- Each block has:
  - ID
  - data (set of transactions)
  - Pointer to previous block
  - Hash of previous block
- Allowed operations: append

**Main Property**

- Data remains forever
- Blocks are **cryptographically immutable**
- If \( A \) changes a block, \( B \) can (easily) notice it
  - Hash function
  - Remember the pointer and hash to the head
Bitcoin

- **Bitcoin** is a currency
- Distributed
- Operated through the **bitcoin p2p network**
- Uses the **bitcoin blockchain**

The Bitcoin Blockchain

- Decentralized
- Public
- Used as a **ledger**
- The blocks data consists of transactions
  - Optimization: Several transactions in each block
### Bitcoin Transactions

#### Transfer Transaction
- **Transfer** = destroy and create
- Address and signature of sender
- Address of recipient
- Coins to be used
- Pointer to creation of coins

#### Creation Transaction
- **Data:**
  - Address of recipient
  - Value
- When are these issued?
  - Genesis block
  - Every addition of a block
**Bitcoin Transactions**

### Submitting a Transaction

1. **A** wants to send $n$ coins to **B**
2. **A** broadcasts the transaction details to the entire bitcoin network
3. **A** waits for the transaction to be completed.

### Completing a Transaction

- The network decides: Include it in a block on the blockchain?
- Each node makes its own decision
- Honest nodes:
  - Only include valid transactions in their blocks
  - Always add blocks to the longest valid branch
- Assumption: Most nodes are honest

Is the transaction then **completed**?
What does it even mean?

I own a coin

=  

I am able to spend a coin  

=  

When I submit a transaction with this coin  
the transaction will be added to the blockchain
## Bitcoin Transactions

### What does it even mean?

<table>
<thead>
<tr>
<th>I own a coin</th>
<th>I am able to spend a coin</th>
</tr>
</thead>
<tbody>
<tr>
<td>=</td>
<td>=</td>
</tr>
</tbody>
</table>

When I submit a transaction with this coin, the transaction will be added to the **longest valid branch in the blockchain**.
I broadcast a transaction where I transfer money to Amici’s Pizza

My transaction is added to the longest valid branch

Should Amici’s start preparing my pizza?
  
  Will this transaction stay on the longest valid branch?

The more Amici’s wait, the better

6 blocks should be enough ($\approx 1$ hour)
Consensus

Transactions are broadcasted to the entire network
Each node maintains a block with all the new transactions
A hopefully random node gets to add its block to the chain
Where? hopefully appending to the longest valid branch
  Adding to a branch = confirming validity

Hope

How to fulfill hopes?
How to choose a random node?
How to encourage nodes to being honest?
Achieving Honesty: Incentives

**Block Creation Fee**
- Every block includes a special transaction to its creator.
- Fixed amount
- Nodes want their blocks to appear in the longest valid branch
- Otherwise, the reward is useless

**Transaction Fee**
- Transactions *may* add a transaction fee to the block creator
- Fee is useless unless on the longest valid branch
Achieving Randomness: Mining

You Gotta Work For It!

- Nodes (miners) compete for the right to create blocks
- They need to prove that they worked for it
  - Look for a number $x$ such that $\text{hash}(x \# \text{txs}) < \epsilon$ and put it in the block
  - Assumption: the hash function is secure
  - No way other than exhaust the search space
  - Ensures randomness of block creator
- Searching for $x = \text{mining}$
- A node that searches for $x$: miner
The Need for Altcoins

“Script”: The Language of Transactions

- Transactions are written in “Script”
- Limited scripting language
- Stack-based, no loops
- Allows for limited variants of the above two transaction types

- These are not “Smart Contracts” yet
- “Turing-complete”-blockchain
Bitcoin Recap

- Public, distributed blockchain
- Relies of honest majority
- Never 100%, but exponentially reliable over time

- Transactions are broadcasted
- Written to blocks
- Blocks are added to the blockchain

- Miners create blocks
- Achieves randomness
- They get coins for it

- Transactions are written in scripts
- Limited language
Ethereum

- Like bitcoin, but with a Turing-complete scripting language
- Also has a blockchain
- Scripts = smart contracts
  - Code = meaning of contract
  - Execution = enforcement of contract
- Contracts are added to the blockchain via transactions
- Contracts are assigned with an address and a balance

Ether and Beyond

- Ether = The Ethereum currency
- General-purpose blockchain
- Other currencies
- Other purposes
Smart Contracts

- Deployed as bytecode
- Run by Ethereum Virtual Machine (EVM)
- Usually written in a high-level language: **Solidity**
- Stateful
- Other high-level languages are considered

```solidity
pragma solidity 0.4.8;
contract ControlStructure {
    address public a;
    function ControlStructure(uint input1) {
        while(input1 >= 0){
            if(input1 == 5)
                input1 = input1 - 1;
            a++;
        }
    }
}
```
Gas

- Preventing contracts from running forever: Gas
- Each VM instruction has a fixed cost in gas units
- When publishing a transaction to the network, the sender specifies:
  - how much (s)he will pay per gas unit
  - gas limit
- If gas limit is hit, the execution is reverted
- The miner gets the gas value
Blockchain Technology, and in particular the Ethereum blockchain are (relatively) new fields.
A lot of research subjects naturally arise.
To name a few:

- Cryptographic protocols
- Consensus Protocols
- Incentives
- Estimation of gas costs
  - Decide whether to submit a transaction
  - Decide what gas limit to put
- Verification of smart contracts
  - Find bugs
  - Know what the contract does
Reasoning about Smart Contracts

- Solidity is a programming language
- We would like to verify some properties of smart contracts
- Examples:
  - Safety w.r.t. particular attacks
  - Termination
  - Not running out of gas
  - Specification by examples
- Challenges:
  - Non-standard control flow
    - Contracts are called by other contracts whose code is unknown
    - Need for modularity
  - Need to reason about second-order concepts
  - Sum, count,...
  - Is gas an internal or external notion to the contract?
Example 1: Tokens

**Tokens**

- The Ethereum blockchain is used not only for Ether
- It is a general-purpose blockchain
- Many currencies are created within it, they are called *tokens*
- Tokens may differ in their logic / rules / functionality.

**ERC20 Standard**

- A standard for tokens
- Tokens should include several functions, e.g.:
  - `totalSupply()`
  - `balanceOf(address)`
  - `transfer(to, tokens)`
  - ...
**Example 1: Tokens**

```solidity
contract SimpleToken {
  def ts : uint //total supply
  def b : address -> uint //balances
  method burn(a : uint, s : address) { //amount, sender
    ts = ts - a
    if (b[s] >= a) {
      b[s] = b[s] - a
    }
  }
}
```

We would like to prove an invariant: $\text{Sum(balances)} = \text{totalSupply}$

$\sum b = ts \Rightarrow (ts' = ts - a \land (b[s] \geq a \Rightarrow b' = b[s] \leftarrow [s] - a)) \land (b[s] < a \Rightarrow b' = b)) \Rightarrow \sum b' = ts'$

Not Valid!
Example 1: Tokens

```
contract SimpleToken {
    def ts : uint //total supply
    def b : address -> uint //balances
    method burn(a : uint, s : address) { //amount, sender
        if (b[s] >= a) {
            b[s] = b[s] - a
            ts = ts - a
        }
    }
}
```

We would like to prove an invariant: \( \text{Sum(balances)} = \text{totalSupply} \)

\[
( \Sigma b = ts \Rightarrow ( (b[s] \geq a \Rightarrow (b' = b[s] \leftarrow [s] - a) \land ts' = ts - a ) ) \land
(b[s] < a \Rightarrow ( b' = b \land ts' = ts ))) \) \Rightarrow \Sigma b' = ts'

Valid!
Example 2: Wallets

Multi-signature Wallets

- In some cases, it makes sense to have a shared wallet
- $n$ owners, at least $m$ must sign for each transaction
- Examples:
  - Spouse joint account
  - Company board of directors
  - Buyer, seller, trustee
Example 2: Wallets

```solidity
contract Wallet {
    def req : uint // number of required signatures
    def os : address -> bool // owners
    method removeOwner(o: address) {
        os[o] = false
    }
}
```

We would like to prove an invariant: \( \text{Count}(os) \geq req \)

\( \text{CountTrue}(os) \geq req \Rightarrow (os'[o] = false \Rightarrow \text{Count}(os') \geq req) \)

Not Valid!
contract Wallet {
    def req : uint //number of required signatures
    def n: uint //number of owners
    def os : address -> bool //owners
    method removeOwner(o: address) {
        if n > req {
            os[o] = false
            n = n-1
        }
    }
}

We would like to prove an invariant: $n \geq req \land n = \text{Count}(os)$

$$( n \geq req \land n = \text{Count}(os) ) \Rightarrow (( n > req \Rightarrow (os' = os[o \leftarrow false] \land n' = n - 1 )) \Rightarrow ( n' \geq req \land n' = \text{Count}(os') ) )$$

Valid!
Background Recap

We have seen:
- The Blockchain data structure
- Bitcoin
- Ethereum & Solidity
- Verification

We have not seen:
- Cryptography
- Consensus
- Low-level details
  ...

Resources:
- Princeton Bitcoin book
- Stanford course
- Tons of other resources

Thanks:
- Shelly Grossman
- Mooly Sagiv
- You
Topics

High Level Topic
Verification of Smart Contracts

Sub-topics
- Smart Contract Languages and their vulnerability
- General-purpose Verification Techniques
- Specific Verification Techniques for Smart Contracts
## Smart Contract Languages and Vulnerabilities

### Languages
- Script
- Solidity and Ethereum Bytecode
- Move
- Michelson (Tezos)
- ...

### Vulnerabilities
- Real assets are transferred
- No safety net
- Private contract storage vs. shared blockchain storage
- Callbacks and interactions between contracts
- ...

---

Smart Contracts Verification
Verification

Verification, Testing, Auditing

- Verification: 100% correctness, non-terminating
- Testing: Low coverage, terminating
- Auditing: Mostly manual
- Combinations: e.g., verification techniques for test generation

Rice’s Theorem

- It is undecidable to determine whether a given program satisfies a certain (semantic, non-trivial) property
- Verification is impossible?
- Heuristics, incompleteness, application-guided research
Verification Despite Rice’s Theorem

**Satisfiability Modulo Theories (SMT)**

- Core Technique: Translating programs into a logical formula
- SMT-solvers: general-purpose logical solvers
- Translation is straight-forward without (unbounded) loops
- Loops require dedicated techniques
Verification of Smart Contracts

Specific Challenges and Techniques

- Gas
- Special vulnerabilities
- Basic SW verification techniques work to a certain extent
- Specific techniques are developed for Smart Contracts

Tools

- solc-verify (SRI)
- Verisol (Microsoft Research)
- The Move Prover (Facebook, Stanford)
- Solidity’s internal checker (Ethereum Foundation)
- ...
Demos

- The Move Prover
- cvc5: https://cvc4.github.io/app/
Diverse and Interesting topic: Practical tools + deep theory
Please email me by next lecture your preferred papers
Seminar Website:
https://u.cs.biu.ac.il/~zoharyo1/sc-seminar/index.html