Next Steps with Blockchain Technology

Eliezer Kanal & Gabriel Somlo
Previous models of computing

Data Storage: 
Database

Program Execution: 
Local
Blockchain

Data Storage: Blockchain or Network

Program Execution: Network
Next Steps with Blockchain Technology

Txn
Txn
Txn
Txn

Hash: 45af...
Prev: 39e1...

Hash: 39e1...
Prev: 90f9...

Hash: 90f9...
Prev: a1c4...

Hash: a1c4...
Prev: 5668...

Time
Hash: 45af...
Prev: 39e1...

Hash: 39e1...
Prev: 90f9...

Hash: 90f9...
Prev: a1c4...

Hash: a1c4...
Prev: 5668...

Time
<table>
<thead>
<tr>
<th>Candidate</th>
<th>Votes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>0</td>
</tr>
<tr>
<td>Jim</td>
<td>0</td>
</tr>
<tr>
<td>Frank</td>
<td>0</td>
</tr>
</tbody>
</table>

State: 1
State: 1

<table>
<thead>
<tr>
<th>Candidate</th>
<th>Votes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>0</td>
</tr>
<tr>
<td>Jim</td>
<td>0</td>
</tr>
<tr>
<td>Frank</td>
<td>0</td>
</tr>
</tbody>
</table>

Bob: 1 vote
Frank: 1 vote
<table>
<thead>
<tr>
<th>Candidate</th>
<th>Votes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>1</td>
</tr>
<tr>
<td>Jim</td>
<td>0</td>
</tr>
<tr>
<td>Frank</td>
<td>1</td>
</tr>
</tbody>
</table>

State: 2
State: 1

<table>
<thead>
<tr>
<th>Candidate</th>
<th>Votes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>0</td>
</tr>
<tr>
<td>Jim</td>
<td>0</td>
</tr>
<tr>
<td>Frank</td>
<td>0</td>
</tr>
</tbody>
</table>

State: 2

<table>
<thead>
<tr>
<th>Candidate</th>
<th>Votes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>1</td>
</tr>
<tr>
<td>Jim</td>
<td>0</td>
</tr>
<tr>
<td>Frank</td>
<td>1</td>
</tr>
</tbody>
</table>
Equivalent to:

State: 1

<table>
<thead>
<tr>
<th>Candidate</th>
<th>Votes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bob</td>
<td>0</td>
</tr>
<tr>
<td>Jim</td>
<td>0</td>
</tr>
<tr>
<td>Frank</td>
<td>0</td>
</tr>
</tbody>
</table>

State: 2

State 1 plus...

Bob: 1 vote
Frank: 1 vote
Blockchain: Executive Summary

Pros:
- Authentication built-in
- Easy to audit history
- Easy to detect data manipulation
- Very difficult to disrupt

Cons:
- Proof-of-work very inefficient
- State updates are slow
- Best for simple computations
Bitcoin: Mining

Input

- Previous block signature
- Bunch of transactions
- Random number

<table>
<thead>
<tr>
<th>Signature</th>
<th>Transactions</th>
<th>Random #</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>482AA...</td>
<td>txn 1, 17, 88, 452</td>
<td>1</td>
<td>854A3...</td>
</tr>
<tr>
<td>482AA...</td>
<td>txn 1, 17, 88, 452</td>
<td>2</td>
<td>B4221...</td>
</tr>
<tr>
<td>482AA...</td>
<td>txn 1, 17, 88, 452</td>
<td>3</td>
<td>0249F...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
### Block #509169

<table>
<thead>
<tr>
<th>Summary</th>
<th>Hashes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number Of Transactions</td>
<td>1915</td>
</tr>
<tr>
<td>Output Total</td>
<td>10,289.281302864 BTC</td>
</tr>
<tr>
<td>Estimated Transaction Volume</td>
<td>1,818.689254656 BTC</td>
</tr>
<tr>
<td>Transaction Fees</td>
<td>0.4883376 BTC</td>
</tr>
<tr>
<td>Height</td>
<td>509169 (Main Chain)</td>
</tr>
<tr>
<td>Timestamp</td>
<td>2018-02-14 15:16:59</td>
</tr>
<tr>
<td>Received Time</td>
<td>2018-02-14 15:16:59</td>
</tr>
<tr>
<td>Delayed By</td>
<td>58COIN</td>
</tr>
<tr>
<td>Difficulty</td>
<td>2,874,874,234,415.94</td>
</tr>
<tr>
<td>Elta</td>
<td>3932292665</td>
</tr>
<tr>
<td>Size</td>
<td>1132.416 MB</td>
</tr>
<tr>
<td>Weight</td>
<td>3992.574 kWh</td>
</tr>
<tr>
<td>Version</td>
<td>0xc000000000000000</td>
</tr>
<tr>
<td>Nonce</td>
<td>1858880081</td>
</tr>
<tr>
<td>Block Reward</td>
<td>12.5 BTC</td>
</tr>
</tbody>
</table>

#### Transactions

<table>
<thead>
<tr>
<th>Transaction Details</th>
<th>2018-02-14 15:16:59</th>
</tr>
</thead>
<tbody>
<tr>
<td>ab0a64eal4e3a34fl2cd0881d3b3e11f8a14e7d9520c2718b09d0f4fa0df</td>
<td>12.0803378 BTC</td>
</tr>
<tr>
<td></td>
<td>0 BTC</td>
</tr>
<tr>
<td>a8d65301a8d242a7a2b290b105d395400f0a40a47a68c1f8020b7f747443d52b0a25cd629253767a1</td>
<td>12.9803378 BTC</td>
</tr>
<tr>
<td>11852ZpIfmMnrwBlytepJ8BYvMjYYnEiwWdpvP</td>
<td>0.4980 BTC</td>
</tr>
<tr>
<td></td>
<td>0.1495 BTC</td>
</tr>
<tr>
<td>1H6Z2pzIfmMnrwBlytepJ8BYvMjYYnEiwWdpvP</td>
<td>5.01651602 BTC</td>
</tr>
<tr>
<td></td>
<td>5.06431602 BTC</td>
</tr>
</tbody>
</table>
[root@btc-1 btc-1.terra.com.conf]# btcl getwalletinfo

```
{
  "walletversion": 130000,
  "balance": 0.00000000,
  "unconfirmed_balance": 0.00000000,
  "immature_balance": 0.00000000,
  "txcount": 10,
  "keypoololdest": 1535482256,
  "keypoolsize": 100,
  "paytxfee": 0.00000000,
  "hdmasterkeyid": "6dc4df9fb26336764d0476183c5ec7d56361c0"
}
```

[root@btc-1 btc-1.terra.com.conf]#
DeltaUser1
12.0.0.46

CharlieUser1
12.0.0.54

topgen.info

[btc-1.terra.com]

[root@btc-1 btc-1.terra.com.conf]# btcli getwalletinfo
{
  "walletversion": 138000,
  "balance": 14.82300000,
  "unconfirmed balance": 0.00000000,
  "immature_balance": 0.00000000,
  "txcount": 1,
  "keypoololdest": 1535483250,
  "keypoolsize": 100,
  "paytxfee": 0.00000000,
  "hdmasterkeyid": "72282965abc8c218892c66c6a532a45f389b06"
}

[root@btc-1 btc-1.terra.com.conf]#
DeltaUser1

CharlieUser1

gettx () {
  local TXID=$1
  btccli decoderawtransaction "${btccli getrawtransaction "$TXID"}
}

[btc-1 terra.com]

sendtoaddress mzLLFxpnJjASLzQMUMZwXaWb6sU6CWMN14.823
4aad741923b21bf64d16f664c63f7be6a3b81ca65719dd41bf8068230e3d87bc

[btc-1 terra.com]
Koinbase, Inc is a financial institution registered in the U.S. One of the services it offers is to exchange customers' Bitcoin for U.S. Dollars. Customers send BTC to Koinbase's public address:

Koinbase "mtDqnenHstY1grsnoDscdnurbX598NzfH"

In return, Koinbase deposits an amount of USD matching the current exchange rate into the each customer's Koinbase checking account.

Four customers sent approximately BTC 50.00 each to Koinbase's public address today (name and BTC address listed below):

Alice "mk8PhnCTpq5Nk3C7bMg65dG4CY2XJNip6Z"
Bob "mrd57X4286Hkm3ca1LPv56uiyqltJEVU61"
Charlie "mxnhKxrXmzTDUPY58Ydantp7oZX7VYywN"
Diane "mne3hNXyhtetpzbhG0CJzvmx5TsJawtmpV"

Koinbase monitors a blacklist of reported illegal transactions, and is required to flag and refer suspicious transactions to law enforcement for further investigation. The list of known illegal transactions currently contains one instance of a ransomware attack being paid off, in the amount of BTC 100.00. That transaction's TxID (hash) is:

014dd5ff639fbaed2e23f94d6d73e5bb4bc0cc45bafc8f88bf6b25a5023122c8

As Koinbase's security analyst, you are tasked with determining which, if any, of these four customers' transactions to flag as potentially suspicious.
Tracing Transaction History
Blockchain programming is hard!

- Over $40\text{M}$ were stolen from TheDAO due to a bug in the implementation (June 2016)
- $32\text{M}$ were stolen due to a bug in a commonly used contract (June 2017)
- Bugs in smart contracts cannot be fixed after deployment

We want to build correct software, but current approaches have been shown to have security vulnerabilities
Obsidian: a new programming language

• Obsidian is a blockchain-based language with the goal of minimizing the risk of common security vulnerabilities

• Obsidian contains core features to allow users to write safe programs easily and effectively

• Obsidian programs consist of contracts, which contain fields, states, and transactions
Obsidian: a new programming language

Goals

- Make certain vulnerabilities impossible
- Make it easier to write correct programs
- Show effectiveness and correctness

Components

1. Typestate-oriented programming
2. Resource types
Typestate

- Blockchain programs commonly state-oriented
- Obsidian makes state first-class
  - An object in Obsidian has a **state** that restricts which **transactions** can be invoked on it.
- **State transitions** can change the state of an object
Typestate

```
contract LibraryPatron {
    state NoCard {
        transaction getCard() {
            ...
            -> HasCard;
        }
    }

    state HasCard {
        transaction getBook() {
            ...
        }
    }
}
```

- A LibraryPatron is always in either the NoCard or HasCard state
Typestate

```solidity
contract LibraryPatron {
    state NoCard {
        transaction getCard() {
            ...
            ->HasCard;
        }
    }

    state HasCard {
        transaction getBook() {
            ...
        }
    }
}
```

- A `LibraryPatron` is always in either the `NoCard` or `HasCard` state
- `getBook` can only be called in `HasCard`; calling from `NoCard` state results in compile-time error
Typestate

```solidity
contract LibraryPatron {
    state NoCard {
        transaction getCard() {
            ... ->HasCard;
        }
    }

    state HasCard {
        transaction getBook() {
            ...
        }
    }
}
```

- A `LibraryPatron` is always in either the `NoCard` or `HasCard` state.
- `getBook` can only be called in `HasCard`; calling from `NoCard` state results in a compile-time error
- `->HasCard` is a state transition
Typestate – Other common applications

Voting

• Not Eligible
• Eligible, not voted
• Eligible, voted
Typestate – Other common applications

Supply chain
- Browsing
- Purchasing
- Order in processing
- Shipping
- Delivering to customer
- Return requested
- Delivering to business
- Returned
Typestate – Other common applications

Supply chain
• Browsing
• Purchasing
• Order in processing
• Shipping
• Delivering to customer
• Return requested
• Delivering to business
• Returned

Customer management
Manufacturing
Add steps for wholesale distributor
Linear Types

- Blockchain programs often manage some kind of resource
  - e.g., cryptocurrency, votes, items in supply chain
- **Linear types** allow the compiler to enforce “resource safety”:
  - Resources cannot be used more than once
  - Resources must be used before leaving the current scope (i.e., don’t lose it)
Linear Types

```
resource contract Money {...};

contract Account {
    Money balance;

    transaction closeAccount(Account a) {
        a.withdraw(balance);
        balance = new Money(0);
    }

    transaction withdraw(Money m) {...}
}
```

- Financial application example
- balance is a type of Money, which is a linear type
Linear Types

resource contract Money {...};

contract Account {
    Money balance;

    transaction closeAccount(Account a) {
        a.withdraw(balance);
        balance = new Money(0);
    }

    transaction withdraw(Money m) {...}
}

• When balance is moved out of scope, we need to create a new Money object to replace it

• The new Money is created with a value of 0

• Note that we’re not referring to the actual amount of money, we’re referring to the code used to track the money

• Code security, not accounting
Linear Types

```solidity
resource contract Money {...};

contract Account {
    Money balance;

    transaction closeAccount(Account a) {
        a.withdraw(balance);
        a.withdraw(balance);
        balance = new Money(0);
    }

    transaction withdraw(Money m) {...}
}
```

- Introducing bug... spending more `Money` than available
- Program would fail when trying to compile, rather than when the user tried to run the program
Usability

Programmers should be able to write correct Obsidian code easily and effectively. Creating an intuitive language is hard! Many difficult design choices exist.
Which is “correct”?
Usability study

Participants were given a description of a voter registration system for a hypothetical democratic nation.
Usability study

1. Write pseudocode to implement program.
2. Given a state diagram modeling the voter registration system, modify pseudocode.
3. Given Obsidian tutorial (with no information on state transitions) invent syntax for state transitions and complete an Obsidian contract.
4. Shown three options for state transitions, complete a brief contract for each option.
5. Choose one of the three options and use it to complete the Obsidian program from part 3.
Usability study – Findings

- Programmers do not naturally consider state-based design when architecting code
- Most intuitive design: include all possible state actions explicitly within the state
Summary

**Bitcoin simulator**
- Virtual implementation of Bitcoin network
- Useful for forensic analysis

**Obsidian**
- Secure-by-design language for blockchain development
- Typestate and linear resources help users write safe programs easily and effectively
- Usable programming language design requires iteration and user testing
Contact Information

Presenters

Elli Kanal
Technical Manager
Email: ekanal@cert.org

Gabe Somlo
Cybersecurity Researcher
Email: gisomlo@cert.org