CS 168: Blockchain and Cryptocurrencies

Ethereum Transactions and Smart Contracts

Prof. Tom Austin
San José State University

Transactions

Transactions

- Signed messages triggered by EOA
- Atomic
  - If they fail, they roll back state
  - Gas is still lost
- Flood routing protocol

Transaction Fields

- Gas price
- Gas limit
- Recipient
- Value – ether to send to the destination
- Data – variable length binary data payload
- ECDSA signature fields: v, r, s
- Nonce

Nonce Importance

- Ethereum is account-based, not UTXO-based
  - Simpler
  - Pseudo-anonymity not a major goal (still possible, just more complex).
- Replay attacks are a concern
- Orders transactions from account
  - Transactions are processed in order
  - If a tx fails, subsequent ones will be stuck.

TX Value and Data

Payload may contain either field, both fields, or even neither.

- Neither is a waste of gas, but possible.
- Tx with value is a *payment*.
- Tx with data is an *invocation*.

Smart Contracts

Smart Contracts (Definition from *Mastering Ethereum*)

Immutable computer programs that run deterministically in the context of an Ethereum Virtual Machine as part of the Ethereum network protocol—i.e., on the decentralized world computer.

Smart Contract Life Cycle

1. Published to the *zero address*.
   - **0x0000000000000000000000000000000000000000**
   - Author has no special rights to a contract, unless the contract is written that way.
2. Invoked by transaction.
3. May be destroyed.
   - Only if creator configured it that way.

High-level Languages for EVM

- LLL – Lisp-like language.
  - Oldest, but rarely used.
- Serpent
  - Python-ish
- Solidity
  - JavaScript-ish
- Vyper
  - Also Python-ish
- Bamboo
  - Erlang-ish

High-level Languages for EVM

- LLL – Lisp-like language.
  - Oldest, but rarely used.
- Serpent
  - Python-ish
- **Solidity**
  - JavaScript-ish
- Vyper
  - Also Python-ish
- Bamboo
  - Erlang-ish

Solidity

- Created by Gavin Wood.
- Most popular HLL for Ethereum today.

Solidity Data Types
(not exhaustive)

- `bool`
- `int`, `uint`
  - Variants in 8, 16, 32, …, 256
  - Default is 256
- `fixed`, `ufixed`
  - Not supported yet
- `address`
- Arrays
- Time units
- Ether units: `wei`, `finney`, `szabo`, and `ether`

Global Variables

- `msg` – the transaction call.
  - Fields: sender, value, data, sig
- `tx` – the transaction.
  - Fields: gasprice, origin
- `block` – the block the transaction is in.
  - Fields: coinbase, difficulty, gaslimit, number, timestamp (in seconds since epoch)

Constructing and Destroying Contracts

- Created with `constructor`.
  - Older versions used contract name
- Destroyed with `selfdestruct`.
  - Person who destroys it claims the contract's ether.
  - Only if enabled by author.

Function Syntax

```
function FunctionName
([parameters]) {public|private|
internal|external} [pure|view|
payable]
[modifiers]
[returns (return types)]
```

Visibility

- `public` – same as Java.
  - default
- `private` – same as Java.
- `internal` – can be called by the contract itself, plus derived contracts.
  - Akin to `protected` in Java
- `external` – can *only* be called by an external contract.

Other terms

- `pure` – do not read or modify state
- `view` – do not modify state
  - getters
- `payable` – function accepts ether

Error handling

- Guarantee state.
  - Throw an exception if false.
- `assert`
  - Used only to catch internal programming errors
- `require`
  - Used to validate external input
  - May be given 2nd argument for better error handling

Receive ether function

- `receive() external payable { ... }`
- Only 1 per contract
- Accepts ether
- Replaces default function in older versions of Solidity
- Does not seem to work with JavaScript VM in Remix IDE

Improved Faucet Code

```
contract Faucet {
  address payable owner;
  constructor() {
    owner = msg.sender;
  }
  function withdraw(uint amt) public {
    require(amt <= 0.1 ether, "Don't be greedy.");
    msg.sender.transfer(amt);
  }
  function getBalance() view public returns (uint) {
    return address(this).balance;
  }
  function destroy() public {
    require(msg.sender == owner, "Not yours.");
    selfdestruct(owner);
  }
  function donate() external payable {}
}
```

Lab: Distributed Lottery in Ethereum

There are 3 players. Each player should

1. contribute ether, and
2. pick a random number.

Once the last player has contributed, the winner's address is selected.

The owner of that address can destroy the contract to claim the winnings.

For an extra challenge…

There are several flaws to this lottery's design.

How can a player cheat?

Modify your Solidity code to avoid this issue.