Smart contracts

Digital Assets - Week 3 (Pre-record)

Rhys Bidder
rhys.m.bidder@kcl.ac.uk

KBS, QCGBF

Autumn 2024

Disclaimer 2: These notes on cryptography are heavily simplified and leave out many important details. For any real-world security applications these notes should not be relied upon. Instead you should consult appropriate security resources and reliable security professionals.

Disclaimer 3: Cryptoasset transactions are illegal in some jurisdictions. Do not violate these or any other laws. I am not promoting the use of crypto in countries where it is illegal in any form and these slides are not a promotion of crypto or an invitation to participate in crypto-related activities in such countries. They are purely for educational purposes.

Smart contracts

We will now expand a little on the topics of smart contracts:

- Associated in particular with Ethereum - though other blockchains feature them
- Owing to the Turing complete nature of Ethereum's supported languages, smart contracts can implement a wide variety of functionality
- While their adoption is still in its early stages, some important uses are:
  - Implementing tokens issued on a blockchain
  - Enabling consensus mechanisms (staking involves locking a native token by sending to a SC)
  - Organizing interactions among collaborators in 'Decentralized Autonomous Organizations' (DAO)
  - Automating business and regulatory logic (such as in supply chains and banking)
  - Underpinning decentralized market activity (such as in AMMs)

A definition. . .

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 Ethereum world computer

- Antonopoulos and Wood (2019), *Mastering Ethereum*

Smart contracts

Key (perceived) advantages are:

- Immutability, censorship resistance and robust, round the clock operation - deterministically operating on all nodes of the blockchain
  - Once deployed, the code of a smart contract cannot change
  - Inherited from the underlying blockchain
- Transparency and lack of ambiguity in contract operation
  - SC implementation is exposed on the blockchain (though not always in interpretable form - **danger**)
- Replacement of error-prone manual operations and expensive intermediaries
  - Advanced business logic - and possibly innovative approaches - likely infeasible without automation
  - SC interaction can eliminate/accelerate admin steps
- Collocation of data and computation
  - SCs and (some) of the data they use are both ‘on chain’
  - Capable of storing own state

Smart contracts - Creation and compilation

A smart contract is initially **written** as ‘source code’ (like what you wrote in Matlab)

- Most people write code for Ethereum in Solidity

For nodes to run the contracts, the source code needs to be ‘compiled’ into ‘bytecode’

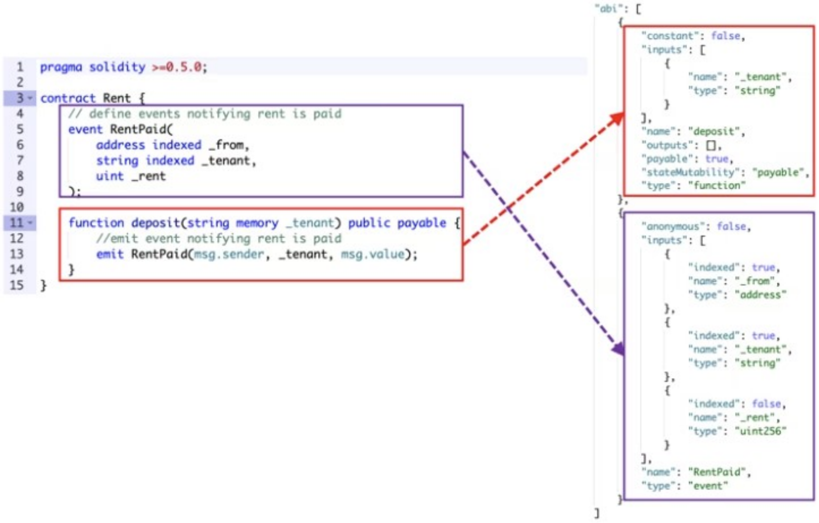
- Bytecode captures the implementation of the contract
- Accompanied by an Application Binary Interface (ABI)
- The ABI lists what functions are included in the smart contract, and how one should interact with them (details about number and nature of inputs and outputs)

Smart contracts - Creation and compilation

```
608060405234801561001057600080fd5b506101f4806100206000
396000f3fe60806040526004361061003b5760003560e01c806360f
e47b1146100405780636d4ce63c1461007f575b600080fd5b348015
61004c57600080fd5b506100556100d0565b6040518082815260200
191505060405180910390f35b6100706100e5565b005b6000602081
905290815260409020545460ff1681565b6000805490509056fea1656
27a7a723058203dcd3f1c7a2f2d9b7ed8d580aaf44c30f03111814a
35c3fb3dceee130bdc2f2b0029
```

- Example of bytecode for a simple smart contract

Smart contracts - Creation and compilation



```
pragma solidity >=0.5.0;

contract Rent {
    // define events notifying rent is paid
    event RentPaid(
        address indexed _from,
        string indexed _tenant,
        uint _rent
    );

    function deposit(string memory _tenant) public payable {
        //emit event notifying rent is paid
        emit RentPaid(msg.sender, _tenant, msg.value);
    }
}
```

```
"abi": [
    {
        "constant": false,
        "inputs": [
            {
                "name": "_tenant",
                "type": "string"
            }
        ],
        "name": "deposit",
        "outputs": [],
        "payable": true,
        "stateMutability": "payable",
        "type": "function"
    },
    {
        "anonymous": false,
        "inputs": [
            {
                "indexed": true,
                "name": "_from",
                "type": "address"
            },
            {
                "indexed": true,
                "name": "_tenant",
                "type": "string"
            },
            {
                "indexed": false,
                "name": "_rent",
                "type": "uint256"
            }
        ],
        "name": "RentPaid",
        "type": "event"
    }
]
```

ABI in JSON form - simple example. Source: Alchemy: *What is an ABI of a Smart Contract?*

Smart contracts - Creation and compilation

[A]n ABI makes it possible for software written in different programming languages or on different platforms to work together seamlessly.

One way to think about an ABI is as a contract between two programs. Just like a legal contract specifies the rights and responsibilities of each party, an ABI specifies how two programs will interact with each other. This makes it possible for developers to write software that works with other software, **even if they don't have direct access to the source code**.

- Understanding ABI and Bytecode in Ethereum

Smart contracts - Deployment

After compilation the bytecode (though not the ABI) is stored on the blockchain

- So it needs to be included in the data payload of a transaction
- The transaction leaves the ‘to’ field empty (the 0x address)
- The (runtime) bytecode of the contract is stored at a contract address
- *Side note:* The contract address is based on the address of the ‘person’ who deploys the contract and the nonce capturing the number of transactions that address has sent

To interact directly with the contract one must send transactions with data and/or value payloads

- These indicate which functions should be called and with what inputs

Smart contracts - Working out the kinks...

Smart contract development is still in its relatively early stages and there are some warnings to keep in mind:

- Immutability is a double edged sword - if it's wrong, it can't be updated
  - Rigorous testing, including on 'testnets' is key
  - Importing/reusing SCs produced by reputable parties, such as OpenZeppelin
  - Strictly there *are* methods to deploy 'upgradable' SCs
- Bugs can be catastrophic
  - Barely a day goes by without some bug leading to an exploit
  - Code re-use (though it has its benefits) makes it hard to know exactly what is going on
  - Bug bounties increasingly common, as are SC auditors

Smart contracts - Working out the kinks...

- Some important blockchains (e.g. Ethereum) may become congested and expensive
  - Volatility in gas fees may interfere with SC operation
- Legal standing of smart contracts, and ambiguity whom to blame in the case of failures
  - Limited regulatory guidance
  - Defi insurers, such as Nexus Mutual, are emerging to insure against SC bugs among other things
- Transparency also has its downsides
  - Makes bugs easily discoverable
  - Also can lead to problems with front running while smart contract transaction is in mempool

Smart contracts - Verification and auditing

The ABI and 'human readable' versions of smart contracts are not immediately available

- Only the (runtime) bytecode is saved to the blockchain
- You may know its address, but not necessarily what it does or how to interact with it
- Makes development difficult - but also dangerous (may be hidden traps)

Ideally, the developer will provide sourcecode

- This can then be compiled and the outcome compared with the bytecode on-chain
- If they match, then it is clear what the code will do
- Etherscan provides one way to do this (see this nice walkthrough), while Tenderly provides another

Smart contracts - Oracles

Many smart contracts that one might want to write, will need data that is not 'native' to the blockchain

- Eg. a financial contract may need info about a stock price, or temperature in a particular location...
- This is the 'oracle problem'

How are they to receive such data?

- This is where 'oracle' protocols come in
- Provide connections between blockchains and real-world data
- In fact, oracles also help connect *multiple* blockchains and legacy systems (will discuss later)

Clearly, it is absolutely vital that SC's receive accurate information

- But, ideally, should be done in a trustless/decentralized way (see one solution here)
- Otherwise, why bother with blockchain? **Do you agree?**