Blockchain 2.0: Smart Contracts

Karan Bharadwaj

August 7, 2016

The relevance of blockchain technology to the financial world has grown substantially over the last few years. An important aspect of the evaluation of blockchain tech is understanding its technical underpinnings to properly design for individual industries. The very first and wildly successful blockchain was the bitcoin cryptocurreny. While cryptocurrency based blockchains are largely incompatible with regulated financial industries, it is useful to look at, in some detail, the origins of the powerful paradigms that are being repurposed to service a myriad use cases. The most fundamental of these are **smart contracts**. In this paper we'll take a look at the earliest and most basic smart contracts as they were designed to serve the bitcoin blockchain and then move onto a much more sophisticated implementation via Ethereum. We'll also take a look at existing security loopholes and offer preliminary solutions in the context of enterprise private blockchain implementations.

The Origins of Smart Contracts

The bitcoin blockchain is a database of transactions. The transaction structure is comprised of 'inputs' and 'outputs'. These outputs are called unspent transaction outputs or **UTXO**s. UTXOs are the fundamental building blocks of the bitcoin blockchain. The bitcoin client keeps track of all UTXOs and the bitcoin balance in a particular wallet is constructed from these UTXOs registered to the corresponding wallet address(or public key). Each transaction generates outputs which are registered to the recipient. These UTXOs are locked with a script called the locking script that specifies the conditions necessary to spend that output. Most commonly the locking script locks the output to a specific bitcoin address(or public key). The inputs of a transaction are pointers to these scattered UTXOs in the blockchain. An unlocking script that allows for their

ownership to be transferred is generated by the wallet once the appropriate UTXOs are selected .

Table 1: The Structure of a transaction.

Size	**Field**	**Description**
4 bytes	Version	Kind of transaction
1-9 bytes	Input Counter	How many inputs are included
Variable	Inputs	One or more inputs
1-9 bytes	Output Counter	How many outputs are included
Variable	Outputs	One or more outputs
4 bytes	Locktime	A timestamp or blocknumber

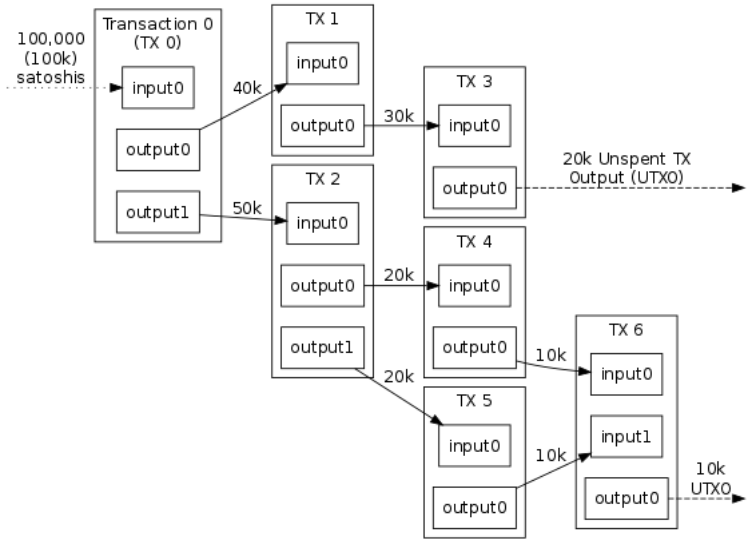

Figure 1: Triple-Entry Bookkeeping (Transaction-To-Transaction Payments) As Used By Bitcoin

The bitcoin client executes the locking and unlocking scripts to validate transactions and populate the blockchain. These scripts are written in a primitive language called **Script**. This simple language is intentionally **Turing-incomplete**, which means that no script can be looped, to prevent malicious attacks on the bitcoin network. While Script's simplicity is deliberate for security and computational reasons, the possibilities for creating more complex conditions for spending are tantalizing. These advanced scripted transactional rules are called Smart Contracts. Within the bitcoin ecosystem, simple smart

contracts are implemented through the pay-to-script-hash(P2SH) transaction. In this kind of transaction, the locking script includes a hash of the encoded smart contract(called the redeem script), which is unlocked by the recipient who possesses the redeem script.

The implications smart contracts hold for the financial world are substantial. The technical underpinnings of transactions on the blockchain can provide a potent mechanism for transfers beyond those involving currencies. Sophisticated embedded code in the transactional scripts can give rise to **autonomous** and **self-sufficient** execution of transfers of value. Once two parties sign a smart contract, depending on the conditions encoded in the locking script, the contract executes autonomously. Neither party has the ability to change the terms of the contract once it has been executed. This autonomy coupled with the immutability of the blockchain, can result in a very secure digital trust mechanism. Self-sufficiency in the context of smart contract is the ability for the contract to access or dispense additional resources like obtaining processing power; as in the case of Ethereum. The ability of smart contracts to interact with other contracts and accounts

Smart Contracts Evolved: Ethereum

Ethereum is the most successful application of a smart contracts platform today. It was designed to be a more generalized blockchain where complex smart contracts can be created and executed. The Ethereum blockchain shares significant similarities with the Bitcoin blockchain. Transactions are appended in both networks via a proof-of-work consensus mechanism. The proof-of-work consensus is a computational exercise undertaken by the participants of the Ethereum network to create new blocks of transactions via a process called mining. In Ethereum two kinds of accounts can be created. The first is called an externally owned account(EOA), which is similar to a bitcoin account that stores an amount of currency. Transactions between two EOAs is the same as a bitcoin transaction between two wallets. Ethereum has a second kind of account called a contract. A contract account stores the contracts code and has it's own data storage. A transaction from an EOA to another EOA simply transfers a quantity of ether. A transaction to a contract account runs the contract code. The contract code has the ability to read the code, quantity of ether, and write to it's own storage. More importantly the contract can send a transaction to another

contract account. When a contract executes it's code, it has the ability to create another contract as well. These last two properties of contract accounts give rise to powerful tools that can be used to create complicated smart contracts that satisfy a number of use cases

Ethereum deviates significantly from the Bitcoin blockchain in the sophistication of it's scripting language. As stated above, the bitcoin scripting language was left intentionally primitive for security and scalability reasons. Ethereum's scripting language, called Solidity, is designed to create complex smart contracts. This added complexity comes at a cost though. The structure of the Etheruem network necessitates the running of all smart contracts on all of it's nodes(similar to the Bitcoin network). A malicious smart contract would be devastating to the entire network. The paradigm of the digital currency is reworked for the Ethereum case. The concept of gas, procured with ether, defines the number of computational steps an account can carry out. So a malicious attack which relied on abusing the halting problem scenario(explained below) will need a continuous supply of gas to run it's computational steps. An elegant solution that removes security vulnerabilities at the most basic level. Unfortunately even though the Ethereum network is responsibly audited for security, applications built upon it may not have been subjected to a similar level of scrutiny. Let's assume a smart contract is created by A to pay out 20$ to B if the temperature on some day x is above 30 degrees c for the price of 10$. Such a contract can run a marketplace of such bets and lets people find, bid, and settle them. Two security problems arise from this arrangement. Our contract developer A can be a malicious individual and steal the money instead of settling or he could be an incompetent programmer whose poorly written code fails at some point of the execution. While both these situations are different, there are significant similarities in the possible solutions for these security loopholes.

The Halting Problem

The Goldbach conjecture states that every even integer greater than 2 can be expressed as the sum of two primes. What if we wrote a program that only halts if the Goldbach conjecture fails?

```
def goldbach():
  r=4
  while True:
    for p1 in range(r)
      for p2 in range(r)
            if (prime(p1) && prime (p2)) && (p1+p2 == k)
                  ok= True
    if not ok:
          halt()
    r+=2
```

The above program only halts if the Goldbach conjecture is false. So far mathematicians have verified the Goldbach conjecture for r $\leq 10^{18}$. The pertinent question is whether we can create a program that if fed the above code can tell us if it halts or goes on forever. Intuitively one can see that such a program cannot exist and it's straightforward to prove as such .

Security Concerns and Smart Contracts

The proposed security paradigm is divided into low level solutions and high level solutions. The low level security solutions necessitate the division of programs into two parts: core and interface. The core of the application is the smart contract itself which is stipulated to be small and extensively audited for security. The interface is the HTML and Javascript code that communicates with the core. At the higher level security solutions are harder to implement. The Ethereum programming language Solidity is currently being integrated with formal verification tools. Formal verification is the application of mathematical analysis by computer programs on other computer programs. In Ethereum, the currently experimental why3, a formal verification engine is being integrated into Solidity. However, there are complications in the application of this technology to smart contracts. The act of defining what is to be proved is extremely

hard which makes the higher level implementation difficult and complex.

The Decentralized Autonomous Organization(DAO) built by the Slock.it for the Ethereum network was launched earlier this year. Technically speaking a DAO can be anything that is not owned by a central authority. Therefore the bitcoin network would be a DAO as well. The DAO being discussed here is a particular company that launched their program on the Ethereum network in April 2016. Generally a DAO is created when a group of developers write code(smart contracts) that defines the organization rules. As with the DAO created by Slock.it, there is a period of funding where a restricted ownership(limited to voting rights) is procured by users by buying coins(ether in this case). Once the funding period is over, the DAO can begin operation and it's members can decide on how the funds can be used by voting on proposals. The DAO launched on Ethereum was wildly successful and raised over $150m through 11,000 members over the period of a month. The DAO was unfortunately the victim of a hack through a known recursive calling vulnerability by calling the split function recursively. The result was that 3.6m ether($72 m) was drained into a child DAO with the same structure. The Ethereum developers responded with a temporary soft fork which froze the ability for accounts on the DAO from spending ether and subsequently a hard fork which recovered the stolen ether.

The promise of smart contracts is not without significant security concerns. While Ethereum brings sophisticated smart contracts functionality, it raises significant concerns in an industry where data security is paramount. While a private or permissioned network can exist within the Ethereum ecosystem, the mechanisms to ensure a well developed framework for permissioned networks are in their infancy.

Proposed Paradigm of Smart Contracts

The proposed solution is a permissioned blockchain with three defined layer to the protocol. At the lowest layer is the blockchain implementation where the ledger and various kinds of account structures exist. By design the lowest layer is use-case agnostic and can be comprehensively customized for a large variety of applications. In contrast to the Nakamoto lottery(proof of work) for the bitcoin and Ethereum blockchains, the choice of consensus is based on the Ripple Labs network consensus mechanism. In a public blockchain like Ethereum or bitcoin, a computaionally expensive proof of work mechanism is a necessity to

prevent double spending. In a permissioned blockchain, consensus mechanisms are much lighter computationally and with very low settlement times(at the scale of seconds). Furthermore, permissioned blockchains run on trusted nodes which are inaccessible to the outside world which improves the security outlook of the application when compared to one on a public blockchain.

The middle layer is the core layer of the smart contract implementation. Individual smart contracts will be defined in a heavily audited system to ensure no lower level vulnerabilities exist in the code. The middle layer communicates with the blockchain layer directly. One manner in which smart contract robustness will be achieved is through completely private and highly customized development of the smart contract for the particular use case. In the banking use case, this core layer will perform the necessary identification, authentication, and communication without allowing for direct access to the lowest level which is the implementation of the blockchain. The top layer is the interface layer through with end customers will avail the service in question. This layer is similar to the web front end and will be written in similar ways.

References

[1] Antonopoulos, Andreas. *Mastering Bitcoin.* O'Reilly, 2014.

[2] Swan, Melanie. *Blockchain: Blueprint For A New Economy.* O'Reilly, 2015.

[3] Buterin, Vitalik. *Ethereum: Platform Review.* 2016.

[4] Buterin, Vitalik. *Hard Fork Completed* . www.blog.ethereum.org/2016/07/20/hard-fork-completed/, July, 2016.